University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

UP Manila Course Timetabling System (UPM CTS)

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

John Renz P. Ladia

May 2015

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "UP Manila Course Timetabling System (UPM CTS)" prepared and submitted by John Renz P. Ladia in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Perlita E. Gasmen, M.S. (candidate)
Adviser

EXAMINERS:

		Approved	Disapproved
1.	Gregorio B. Baes, Ph.D. (candidate)		
2.	Avegail D. Carpio, M.S.		
3.	Richard Bryann L. Chua, Ph.D. (candidate)		
4.	Perlita E. Gasmen, M.S. (candidate)		
5.	Marvin John C. Ignacio, M.S. (candidate)		
6.	Vincent Peter C. Magboo, M.D., M.S.		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.S.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

University course timetabling is a complex and tedious task which involves a lot of variables and constraints. In the University of the Philippines Manila, course timetabling is still being done manually every semester. Doing it this way takes a lot of time and personnel, and the resulting timetables are still not optimal. The UP Manila Course Timetabling System (UPM CTS) is a web application which automates the process of assigning rooms and timeslots to classes in the generation of timetables in UP Manila using a two-stage optimization algorithm consisting of iterative forward search algorithm, hill climbing algorithm, great deluge algorithm, and simulated annealing algorithm. The algorithm tries to find the most optimal timetable based on the constraints provided until a predetermined time limit is reached. If a complete timetable, where all classes are assigned to rooms and timeslots, cannot be found, a subset of classes with assigned rooms and timeslots is still returned.

Table of Contents

I. Introduction

A. Background of the Study

A timetable is a plan of times at which events are scheduled to take place, especially toward a particular end [1]. Timekeeping is an integral part of people's daily lives. In the ever-evolving society humans live in, time is very precious. Everything is happening fast and everyone must keep up with the things happening around him/her to not get left behind. The obsession with timekeeping dates back in ancient times where ancient schedules revolved around annual, seasonal, monthly, or daily rhythms [2].

In schools, especially in universities, timetables are very important for organizing the flow of events. According to Lewis, university timetabling problems can be arranged into two main categories: exam timetabling problems and course timetabling problems [3]. This paper focuses on course timetabling specifically curriculum-based course timetabling (CB-CTT) where the schedule is based on the curricula published by the university as opposed to post enrollment course timetabling where conflicts between courses are set according to the students' data [4]. In the succeeding part of this paper, course timetabling will refer to CB-CTT.

Course timetabling is a complicated scheduling problem which involves students, teachers, courses, and classrooms. It also involves many factors such as the capacity and type of classrooms, average travel time of students between classes, type of facility required for a particular course, and teacher's availability in a particular schedule.

Conventionally, course timetabling is done manually. In some cases such as small schools, this can be done easily. However, manual timetabling is very time-consuming for big universities, since there are a lot of courses offered. At the University of the Philippines Manila, course timetabling is still being done manually. Because of the lack of rooms and faculty, creating a timetable for UP Manila is extra challenging. At the UP Manila College of Arts and Sciences, classes are manually assigned to instructors and timeslots first, and rooms are assigned by trial and error afterwards. This whole process takes a lot of time and resources. This tedious task is done every semester. Because of this difficulty, some universities are now using computers to solve this problem to save time and labor. Using computers to perform course timetabling can not only consolidate the preferences of the people concerned but can also enable achievement of high satisfaction in spite of the many constraints [5].

However, timetabling is known to be a non-polynomial complete problem which means that there is no known efficient way to locate a solution [6]. Therefore, computer scientists are using heuristic approaches to solve timetabling problems. A heuristic approach proceeds to a solution by trial and error or by rules that are only loosely defined [7]. This method will not necessarily lead to the best solution but it will give a set of good solutions to the problem.

In solving all kinds of timetabling problems, there are always two kinds of constraints that are considered. The first type of constraint is called hard constraint which must never be violated. The second type is called soft constraint which may not necessarily be satisfied, but violations should be desirably minimized [4]. The most optimal timetable is the timetable with the least amount of violations on its constraints.

The survey made by Lewis in techniques for solving university timetabling problems mentioned that in the past decade-or-so, there has been a surge in interest of applying metaheuristic algorithms to timetabling problems [3]. Metaheuristics.org defined metaheuristic as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem. According to the website, the five main metaheuristic models are simulated annealing (SA), tabu search (TS), iterated local search (ILS), evolutionary algorithms (EC), and ant colony optimization (ACO) [8].

B. Statement of the Problem

Currently, course timetabling is being done manually at the University of the Philippines Manila. At the UP Manila College of Arts and Sciences for example, each department manually creates schedules for classes which include the assigned instructor for the class and the class' section. This process takes a few days to do. The lists of schedules are then sent to the office of college secretary (OCS) of the departments. The office of college secretary is responsible for designating an available room to each class considering the size of the class and the type of class (lecture class, laboratory class, PE class, etc.), which may need specific facilities. This step also takes a few days to carry out.

The problem with this current method is that it is very time-consuming, and the resulting schedules are not optimal in terms of the utilization of time schedules and rooms. Since there are a lot of possible combinations in assigning time periods and rooms to courses, doing it manually does not yield to an optimal result because it is almost impossible to look at many combinations manually and choosing the best one. Also, because the assignment of the time and the room for classes are done sequentially by different people (assignment of time is done first before the assignment of rooms), there might not be available rooms for a specific time. These conflicts in schedules are commonly experienced at the OCS which further delays the generation of a valid timetable.

Another problem is that there are only a few constraints in generating UPM course schedules. Adding more constraints makes the manual timetabling more time consuming and complex to do. It is limiting instructors in expressing their preferences in the classes' timeslots and rooms. Instructors may prefer some timeslots than others, maybe because of commitments outside the university at particular times and other various reasons. They are not always available at any time. Even though it is possible to schedule a class at the instructor's preferred time, it is not always considered since it will complicate the timetabling procedure. This results to the adjustment of class schedules during the start of the semester or absences of some instructors due to some prior outside commitments. One might argue that the schedules for the faculty need not to be optimal since the university is paying them to teach at whatever time their respective departments want with some little considerations. It does not mean that the departments do not want to give the optimal schedules to their faculty, but it is just that considering the preferences of every instructor in making timetables is very difficult using the manual method. Lecture classes are sometimes scheduled after the laboratory classes of a course. Some rooms assigned to some classes are too big while some rooms are too small. These problems can have negative effects on teaching and learning as a result of non-optimal course timetabling [5].

There are already existing software for course timetabling but these software solve general timetabling problems and are not applicable to the specific problems with the current timetabling process in UPM.

C. Objectives of the Study

- General:

  Develop an interactive automated timetabling system for the University of the Philippines Manila which can generate optimal timetables in a just a short amount of time that satisfy the constraints provided by the users as much as possible, but may not necessarily have assigned rooms or time schedules to all of its classes.

- Specific:

  1. System/University Admin
     - View the generated timetable of courses for each college
     - Export course timetables in PDF, CSV, or XLS format
     - Create, edit, or delete colleges and college admin accounts
     - Create, edit, or delete semesters
     - Manage his/her account information
  2. College Admin
     - Create, edit, or delete departments and department admin accounts linked to the college
     - Add or remove classrooms linked to the college
     - Edit details (capacity and facilities it has) of rooms linked to the college
     - Generate a valid timetable of courses for a particular semester
     - View the generated college timetable

- View the generated timetables for other colleges
- Export course timetables in PDF, CSV, or XLS format
- Manage his/her account information

3. Department Chairman
   - Create, edit, or delete units and unit head accounts linked to the department
   - Create, edit, or delete instructors linked to any unit of the department
   - Create or delete a course offered by any unit in the department
   - Edit details and constraints (facilities needed, number of units, etc.) of any course offered by any unit of the department
   - Add or delete classes offered by any unit on the department for any particular semester
   - Edit details and constraints (assigned instructor/s, assigned/preferred room/s, and assigned/preferred time slot/s) of any class offered by any unit on the department
   - Create or delete a group of preferred rooms or time slots which can be assigned to any class offered by the department
   - Add or remove prohibited time slots for instructors (linked to any unit on the department) for any particular semester
   - Create, edit, or delete constraints (e.g. same room, same time, same days, different time, etc.) for group of classes offered by the college
   - Create, edit, delete block sections linked to the department

- Add or remove classes for a block section on a particular semester to avoid overlapping of schedules
- View the course timetable generated for the department
- View the generated timetables of other colleges
- Export course timetables in PDF, CSV, or XLS format
- Manage his/her account information

4. Unit Head
    - Create, edit, or delete instructors linked to the unit
    - Create or delete a course offered by the unit
    - Edit details and constraints (facilities needed, number of units, etc.) of any course offered by the unit
    - Add or delete classes offered by the unit for a particular semester
    - Edit details and constraints (assigned instructor, assigned/preferred room/s, and assigned/preferred time slot/s) of any class offered by any unit
    - Create or delete a group of preferred rooms or time slots which can be assigned to any class offered by the department
    - Add or remove prohibited time slots of instructors linked to the unit for any particular semester
    - Create, edit, or delete constraints (e.g. same room, same time, same days, different time, etc.) for group of classes offered by the college
    - View the course timetable generated for the department
    - View the generated timetables of other colleges
    - Export course timetables in PDF, CSV, or XLS format

- Manage his/her account information

D. Significance of the Study

Having an automated way of generating course timetables for the University of the Philippines Manila makes the process of timetabling significantly faster. With this, using trial and error method manually to assign time periods and rooms for generating timetables which takes a lot of time and effort are not be needed anymore. The previous process which usually takes days to complete only takes a few minutes by using the automated method. This saves a lot of labor that can be used in some other more productive work. Also, since the generated timetables can be exported in different digital formats, it is easier to put it in the UPM's database. It saves the time of encoding the data from a physical format.

The generated course timetables using the automated system are better than the ones which are created manually, since it considers more constraints based on the preferences of the people involved. This makes the process of timetabling more interactive. Maintaining each department's sense of ownership in the timetables that are produced is also an important factor in their acceptance of the solutions produced by an automated timetabling process [9]. This system is the one that assists them and not replace them.

Adding preferred rooms and time slots to a particular class is easily done. This enables people to make more efficient schedules. Conflicts between critical required courses for each year can be avoided as much as possible to facilitate students' course retake and credit makeup [5]. Selection of rooms based on the sizes of the classes are optimal. All of these considerations improves the teaching and learning experience at the university.

Since the automated process can consider much more possible combinations in timetabling than the manual method, the conflicts that arise by using the manual method does not happen anymore unless there is really not enough rooms to accommodate all the classes. In using the manual method, a timetable is chosen as long as there are no conflicts without considering other possible timetables that might be better than the chosen one. By using the automated method, many feasible timetables (timetables with no conflicts in it) can be considered, and the most optimal timetable is chosen among those feasible timetables.

This system can also be used in other universities with similar type of course timetabling process since the rooms, faculty, and courses can be easily customized. It can be a catalyst for other Philippine Universities to adapt to automated timetabling and to take part in the research on the timetabling problem in order to come up with better solutions.

E. Scope and Limitations

a. This system is specifically made for course timetabling in the University of the Philippines Manila, but it may also be used by other universities with similar constraints.
b. Since timetabling is an NP-complete problem, the created timetables by the system can only provide the most optimal timetables based on the overall minimization of violations of constraints provided to the system.

c. The assignment of faculty to classes will be manually done by the department chair or unit head, and not by this system.

d. Scheduling of newly created classes after enrollment is outside of the system's scope.

e. Changes in class schedules after enrollment is outside of the system's scope.

f. There is no functionality for rescheduling just a few classes when regenerating a timetable without rescheduling the other classes.

F. Assumptions

a. The scheduling is in view of the block system being implemented in UP Manila

II. Review of Related Literature

The aim of the algorithms for solving timetabling problems is to find a timetable with the least amount of violations in the constraints. According to Lewis' survey regarding timetabling literature, most metaheuristic algorithms for timetabling fall into the following three categories: one-stage optimization algorithms where the satisfaction of both the hard and soft constraints is attempted simultaneously; two-stage optimization algorithms where the satisfaction of soft constraints is attempted only once a feasible timetable (all hard constraints are satisfied) has been found; and three-stage optimization algorithms where violations of the hard constraints are disallowed from the outset by relaxing some other feature of the problem, and attempts are then made to try to satisfy the soft constraints, whilst also giving consideration to the task of eliminating these relaxations [3]. Each of these approaches has its advantages and disadvantages depending on the specific type of problem and other requirements to be considered. Lewis also mentioned that this method of classification should not be interpreted as a definitive taxonomy, and some algorithms might belong to more than one of the categories mentioned.

Metaheuristic algorithms can also be classified into two types: population based algorithms and local area search-based algorithms [10]. The difference between these two types is the way they scan the search space. Population based algorithms perform a multidirectional search and start with a number of solutions. These solutions are refined to obtain global optimal solutions in the whole search space. A population based algorithm combines the good characteristics of the different solutions to come up with better solutions. On the other hand, local area search-based algorithms focus on exploitation rather than exploration, which means it start with a solution and move in one direction without performing a wider scan of the search space.

One of the meta-heuristic algorithms that is widely used for solving timetabling problems is genetic algorithm (GA), a type of evolutionary algorithm, because of its multidirectional search property [10]. GA simulates the biological evolution process of chromosomes using selection, crossover and mutation operators where chromosomes present problem solutions and are evaluated based on their fitness function to select parents [11]. Like evolution, the ones which have the better characteristics survive to give rise to better descendants in the next generations and the inferior ones get extinct. Both [12] and [13] tested GA in solving the timetabling problem and it produced acceptable results.

Others used GA together with other algorithms in order to create hybrid algorithms which combines the good properties of GA with some desirable properties of the other algorithms. They did this because of the random nature of GA which makes the convergence time too long [10]. In order to improve the performance of GA, [14] used case-based reasoning to store and retrieve previous solutions using the genetic algorithm so that when a new problem is given, the system can check for previous timetables that can solve the problem. [15] proposed an algorithm based on the random non ascendant method (RNA) which is a local search algorithm and combined it with the genetic algorithm which resulted in a better performance than using RNA or GA alone. [16] implemented a GA as a two-stage optimization algorithm with a local search algorithm applied at each stage. The addition of the local search greatly speed up the algorithm in improving the maximum fitness of the population. [10] proposed three new hybrid genetic algorithms for solving course timetabling which combine genetic algorithm and fuzzy logic with three types of local search algorithm: randomized iterative local search, simulated annealing, and tabu search algorithm. [17] also proposed a hybrid genetic algorithm by combining GA with tabu search algorithm to prevent searching previously visited areas in order to obtain optimal solution in a shorter time.

[5] and [18] applied particle swarm optimization (PSO) to solve the course timetabling problem. Chen and Shih used it because of its fast convergence, fewer parameter setting, and the ability to fit dynamic environmental characteristics. According to [19], PSO is a global optimization algorithm based on swarm intelligence which comes from the research on the bird and fish flock movement behavior. Since there is a high probability for the solution space of the PSO to be trapped in a local optimal solution after generation evolution, [5] and [18] both used local search algorithms as a disturb mechanism in order to increase the probability of escaping from the local optimal solution.

Some other widely used local search metaheuristic algorithms for solving timetabling problems are tabu search (TS) algorithm and simulated annealing (SA). According to [20], tabu search is a method to avoid getting trapped in a local minimum based on the notion of a tabu list which contains previously encountered configurations. The strategy is to prevent the configurations in the tabu list from being recognized for a number of next iterations. SA on the other hand is based on the way thermodynamic systems go from one energy level to another [11]. It simulates the change in system energy, subject to cooling process, until it converges to a steady or frozen state. It allows downward steps in order to escape from a local maxima [21].

[22] implemented both tabu search algorithm and simulated annealing as two-stage optimization algorithms in solving timetabling problems and compared it to genetic algorithm performance. In this study, tabu search and SA both gave good solutions to the problem and generally outperformed the GA algorithm in both stages of the algorithm. In [4], the authors proposed an adaptive tabu search algorithm (ATS) as a three-stage optimization algorithm. They used a fast greedy heuristic to construct a feasible solution. Then, they adaptively combined the intensification and diversification of the solution using tabu search and perturbation operation from iterative local search respectively in order to reduce the number of soft constraint violations without further breaking any hard constraints. [23] introduced a hybrid two-stage optimization algorithm that combined the useful features of TS and SA to obtain good quality solutions with reasonable computation time.

[24] proposed a hybrid two-stage optimization algorithm that composed of several algorithms. In the first stage of the algorithm, an iterative forward search is used to find a feasible solution which makes use of conflict-based statistics to prevent itself from cycling by memorizing conflicting assignments. In the second stage, several local search algorithms are used to improve the solution. First, a hill climbing algorithm is used to find the local optimum. The hill climbing algorithm always selects the best assignment out of all the neighbors [20]. The great deluge (GD) algorithm, a local search algorithm, is then used to improve the solution. The GD algorithm explores neighboring solutions in a way that the solutions are accepted only if they are better than the best solution so far or

if the value of the solution is less than a defined bound. After a certain bound is reached using GD algorithm, a simulated annealing algorithm is used. If the best solution is not improved for some time, the system is reheated (gives high acceptance probability) and the control goes back to the hill climbing algorithm. The search cycle ends after a predetermined time limit has been reached and the best solution found is returned. Because of this iterative nature of the algorithm, the solver can easily start, stop, or continue from any feasible solution, either complete or incomplete [9].

Just like what Lewis has noted in his survey of metaheuristic algorithms for solving timetabling problems, the algorithms that has been proposed in this review are focused on the authors' own problem instances so it is difficult to assess how well an algorithm is capable of performing in comparison to the other algorithms [3].

III. Theoretical Framework

A. Timetable

A timetable shows when particular events are to take place [20]. Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives [25]. [26] showed that the problem of timetable construction is NP-compete in five independent ways which explains why the construction of a timetable is very difficult.

B. Curriculum-based Course Timetabling Problem

The university course timetabling problem is a type of academic timetabling problem which consists of scheduling a set of lectures for each course within a given number of rooms and time periods [27]. Course timetabling has two categories: post enrollment-based course timetabling and curriculum-based course timetabling (CB-CTT). Muller defined these two types of course timetabling as follows [24]:

- Post Enrollment based Course Timetabling – A timetabling problem where students are given a choice of classes that they wish to attend, and the timetable is then constructed according to these choices (i.e., the timetable is to be constructed after students have selected the classes they wish to attend).
- Curriculum based Course Timetabling - The Curriculum-based timetabling problem consists of the weekly scheduling of classes for several university courses within a given number of rooms and time periods. Conflicts between

courses are determined according to the curricula published by the University and not on the basis of enrolment data.

Furthermore, Muller defined the following entities that must be considered when solving CB-CTT problems [24]:

- Days, Timeslots, and Periods - A number of teaching days in the week are given (typically 5 or 6). Each day is split into a fixed number of timeslots, which is equal for all days. A period is a pair composed of a day and a timeslot. The total number of scheduling periods is the product of the number of days times the number of daily timeslots.
- Courses and Teachers - Each course consists of a fixed number of lectures to be scheduled in distinct periods, is attended by a given number of students, and is taught by a teacher. For each course there is a minimum number of days over which the lectures of the course should be spread, moreover, there are some periods during which the course cannot be scheduled.
- Rooms - Each room has a capacity, expressed as the number of available seats. All rooms are equally suitable for all courses (if large enough).
- Curricula - A curriculum is a group of courses such that any pair of courses in the group has students in common. Conflicts between courses, and other soft constraints, are based on curricula.

C. Timetabling Constraints

Timetabling problems belong to the class of hard-to-solve combinatorial problems so solving it involves many constraints. Each school or university has its specific rules and constraints for the timetable. This is especially true for universities. Therefore, a program written for creating the timetable of a university can hardly be used for a different one [27].

In an automated timetabling, these constraints are generally grouped into two categories: the hard constraints and the soft constraints [3]. Lewis gave brief descriptions to these constraints. According to him, hard constraints have a higher priority than soft constraints. In most systems, the satisfaction of all hard constraints is required. A timetable with no violations in its hard constraints is called a feasible timetable. On the other hand, soft constraints are the ones that we want to satisfy if only possible. Soft constraints are the desirable traits but are not absolutely essential [20]. And, in real world scenarios, it is usually impossible to satisfy all the soft constraints. The "goodness" of a timetable is generally determined by the number of soft constraints violated. An optimal timetable is a timetable with no hard and soft constraints violations.

Some examples of hard and soft constraints are below which was used in the Second International Timetabling Competition [4]:

Hard Constraints

- Lectures: All lectures of a course must be scheduled to a distinct period and a room.
- Room Occupancy: Any two lectures cannot be assigned in the same period and the same room.

- Conflicts: Lectures of courses in the same curriculum or taught by the same teacher cannot be scheduled in the same period, i.e., any period cannot have an overlapping of students or teachers.
- Availability: If the teacher of a course is not available at a given period, then no lectures of the course can be assigned to that period.

Soft Constraints

- Room Capacity: For each lecture, the number of students attending the course should not be greater than the capacity 2 of the room hosting the lecture.
- Room Stability: All lectures of a course should be scheduled at the same room. If this is impossible, the number of occupied rooms should be as few as possible.
- Minimum Working Days: The lectures of a course should be spread into the given minimum 10 number of days.
- Curriculum Compactness: For a given curriculum a violation is counted if there is one lecture not adjacent to any other lecture belonging to the same curriculum within the same day, which means the agenda of students should be as compact as possible.

D. Iterative Forward Search Algorithm

The iterative forward search (IFS) algorithm is a type of local search algorithm, an algorithm that performs incomplete exploration of the search space by modifying an infeasible complete assignment into a better one [20]. Muller proposed an IFS algorithm that operates over feasible, though not necessarily complete solutions. This means that all hard constraints are always satisfied but in doing so, some variables can be left unassigned. The advantage of this method compared to the complete infeasible assignments that usually occur in local search techniques is that even though the solver cannot find a solution, it can still return the largest feasible partial assignment. Therefore, the algorithm can easily start, stop, or continue from any feasible assignment, either complete or incomplete which is good in interactive timetabling applications.

Below is the pseudocode of the implementation of an IFS algorithm by Muller:

```
procedure ifs(V,D,C,$\alpha$) // an initial assignment $\alpha$ is the parameter
   $\sigma$ = $\alpha$;           // current assignment
   $\beta$ = $\alpha$;           // best assignmen
   while canContinue($\sigma$) do          // CSP problem $\Phi$=(V,D,C) is
      A = selectVariable($\sigma$);          // a global parameter
      a = selectValue($\sigma$, A);          // for all used functions
      $\eta$ = conflicts($\sigma$, A, a); //conflicting assignments
      $\sigma$ = ($\sigma$ - $\eta$) $\cup$ {A/a};     //next assignment
      if better($\sigma$, $\beta$) then $\beta$ = $\sigma$;
   end while
   return $\beta$;
end procedure
```

Figure 1: Iterative Forward Search algorithm psuedocode

Where,

V = Variables
D = Values
C = Constraints
α = Initial solution
β = Best solution
σ = current partial feasible solution
η = conflicting assignments

The algorithm schema is parameterized by the following functions:

- The termination condition (function canContinue) – This function determines when the algorithm should finish. Examples of events which can finish the algorithm are:

    - when the maximum number of iterations is reached;
    - when a given timeout value is reached;
    - when the current assignment is good enough, e.g., all variables are assigned and/or some other solution parameters are in the required ranges; or
    - the user terminated the process.
- The solution comparator (function better) – This function compares the current assignment with the best assignment found based on several criteria.
- The variable selection (function selectVariable) – This function is equivalent to the variable selection criterion in constraint programming. In selecting a variable in this IFS algorithm, two criterion are considered. First, if some variables remain unassigned, the "worst" variable among them is selected, i.e., first-fail principle is applied. Second, when all variables are already assigned, choose a variable whose change of a value can introduce the best improvement of the assignment, i.e., the variable whose value violates the highest number of soft constraints.
- The value selection (function selectValue) – This function is responsible for the assignment of a value to the selected variable which is the most preferred and will cause the least trouble.

E. Hill Climbing Algorithm

Hill climbing algorithm is a type of local search algorithm where any local change that improves the current value of the objective function is selected [28]. The algorithm is traditionally terminated when there is no better assignment than the current one, which means that the search is stuck in a local optimum [20].

F. Great Deluge Algorithm

Great deluge (GD) algorithm is a local search algorithm which is similar in many ways to hill climbing algorithm and simulated annealing. A new candidate solution is accepted if it is better or equal than the current solution. A candidate solution worse than the current solution will only be accepted if the penalty of the candidate solution is less than or equal to a pre-defined bound called water level [29]. In GD, the water level is set to a value higher than the expected penalty of the best solution at the start of the search [30]. Then the water level is decreased in a linear fashion during the search until it reaches a value of zero. According to Muller, the bound starts at the value

$$B = GD_{ub} \cdot S_{best}$$

where S_{best} is the value of the current best solution and GD_{ub} is a problem specific parameter (upper bound coefficient) [24]. The bound's value decreases after each iteration by multiplying it to a cooling rate GD_{cr}.

$$B = B \cdot GD_{cr}$$

It continues until B reaches the lower limit ${GD_{lb}}^{at} \cdot S_{best}$, where ${GD_{lb}}^{at}$is the lower bound coefficient. When the lower limit is reached, the value of B is reset back to ${GD_{ub}}^{at} \cdot S_{best}$.

The parameter at is a counter that starts at 1 and is increased by 1 whenever the lower limit is reached wherefore increasing the bound. It resets back to 1 whenever the previous best solution is improved. This allows the algorithm to get out of a deep local minimum.

G. Simulated Annealing Algorithm

Simulated annealing (SA) is a local search algorithm proposed by Kirkpatrick, Gelatt, and Vecchi that simulates the physical process whereby a solid is slowly cooled until it reach a "frozen" structure, which happens at a minimum energy configuration [31]. According to Jat, the material or solid is first heated to a high energy (where the state frequently changes) so that its molecules and atoms are set randomly [32]. Then, it is gradually cooled down to a low energy (where the state rarely changes) so that its molecules reach the state of the minimum energy. The decreasing of the temperature is made slowly because if the cooling process is too fast, unstable structures might appear instead of those with the minimum energy. In Muller's implementation of SA algorithm, a generated neighbor assignment is accepted if it does not worsen the overall value of the current solution, or with the following probability based on the temperature and the change in the evaluation function [24]

$$P_{accept} = e^{-\Delta/T}$$

where, Δ is the increase in the overall value of the solution when a detrimental move is assigned and T is the current temperature. The temperature T starts at an initial value SA_{it} and is cooled down by multiplying it with the cooling rate SA_{cr} after each $SA_{cc} \cdot TL$ iterations. SA_{cc} is a cooling coefficient and TL is an instance specific number (temperature length), computed as the sum of domain sizes of all variables. If the best solution is not improved after$SA_{rc} \cdot SA_{cc} \cdot TL$ iterations (SA_{rc} is a reheat coefficient), the temperature is reheated to a value of

$$T = T \cdot {SA_{rc}}^{-1.7/SA_{rc}}$$

H. Process Flow

Figure illustrates the suggested flow of processes needed to be done by the users of the system in order to create a timetable for a semester.

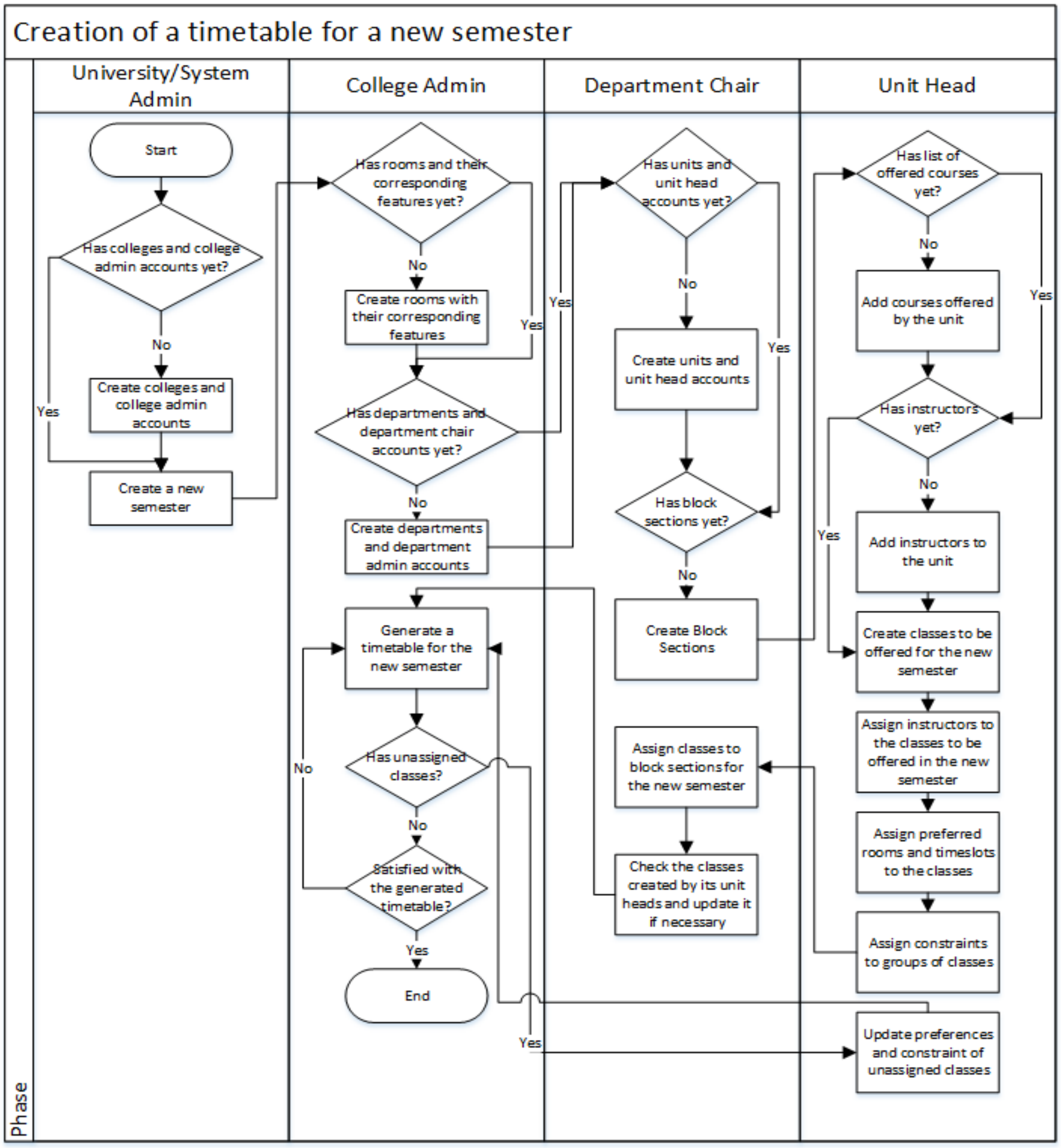

Creation of a timetable for a new semester
University/System Admin
College Admin
Department Chair
Unit Head
Phase
Start
Has colleges and college admin accounts yet?
No
Yes
Create colleges and college admin accounts
Create a new semester
Has rooms and their corresponding features yet?
Create rooms with their corresponding features
Has departments and department chair accounts yet?
Create departments and department admin accounts
Generate a timetable for the new semester
Has unassigned classes?
Satisfied with the generated timetable?
End
Has units and unit head accounts yet?
Create units and unit head accounts
Has block sections yet?
Create Block Sections
Assign classes to block sections for the new semester
Check the classes created by its unit heads and update it if necessary
Has list of offered courses yet?
Add courses offered by the unit
Has instructors yet?
Add instructors to the unit
Create classes to be offered for the new semester
Assign instructors to the classes to be offered in the new semester
Assign preferred rooms and timeslots to the classes
Assign constraints to groups of classes
Update preferences and constraint of unassigned classes

Figure 2: Process flow diagram for creating a new timetable

IV. Design and Implementation

A. Use Case Diagram

The system is only composed of a Web application. It has four types of users: system/university admin, college admin, department chair, and unit head.

The system admin uses the system to create, edit, and delete college admin accounts. He/she is also responsible for creating new semesters.

The college admin can create, edit, and delete departments and department chair accounts. Also, He/she is the one responsible for managing the rooms belonging to the college. Once the departments have finalized their offered classes for a particular semester, the college admin is responsible for the generation of the course timetable for the college. Once the timetable is generated, he/she can view, or export it.

The department chair is responsible for managing the units belonging to the department. He/she can create, edit, and delete units and unit head accounts. He/she can do all the functionalities that are available to unit heads and override any of their input. Also, he/she is responsible for managing the block sections in the department.

The unit head is responsible for creating, editing, and deleting offered courses. He/she is also the one who adds instructor information to the system. These instructors are linked to the unit. Every semester, he/she is the one who creates classes from the courses offered by the unit. Included in this task is the assignment of instructor/s, preferred rooms, and preferred time schedule to the classes. He/she can further assign constraints to a group of classes.

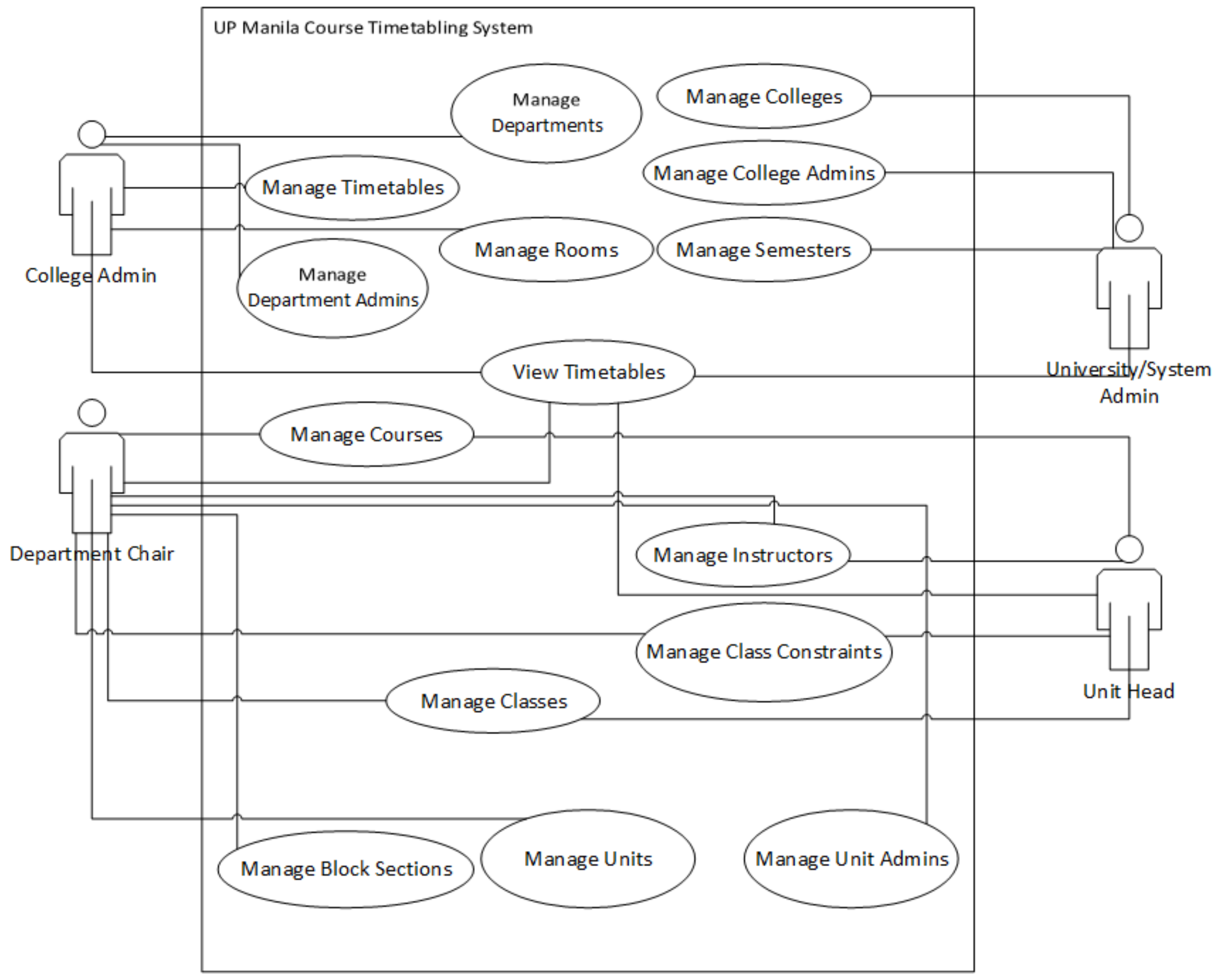


Figure 3: General Use Case Diagram for UPM Course Timetabling System

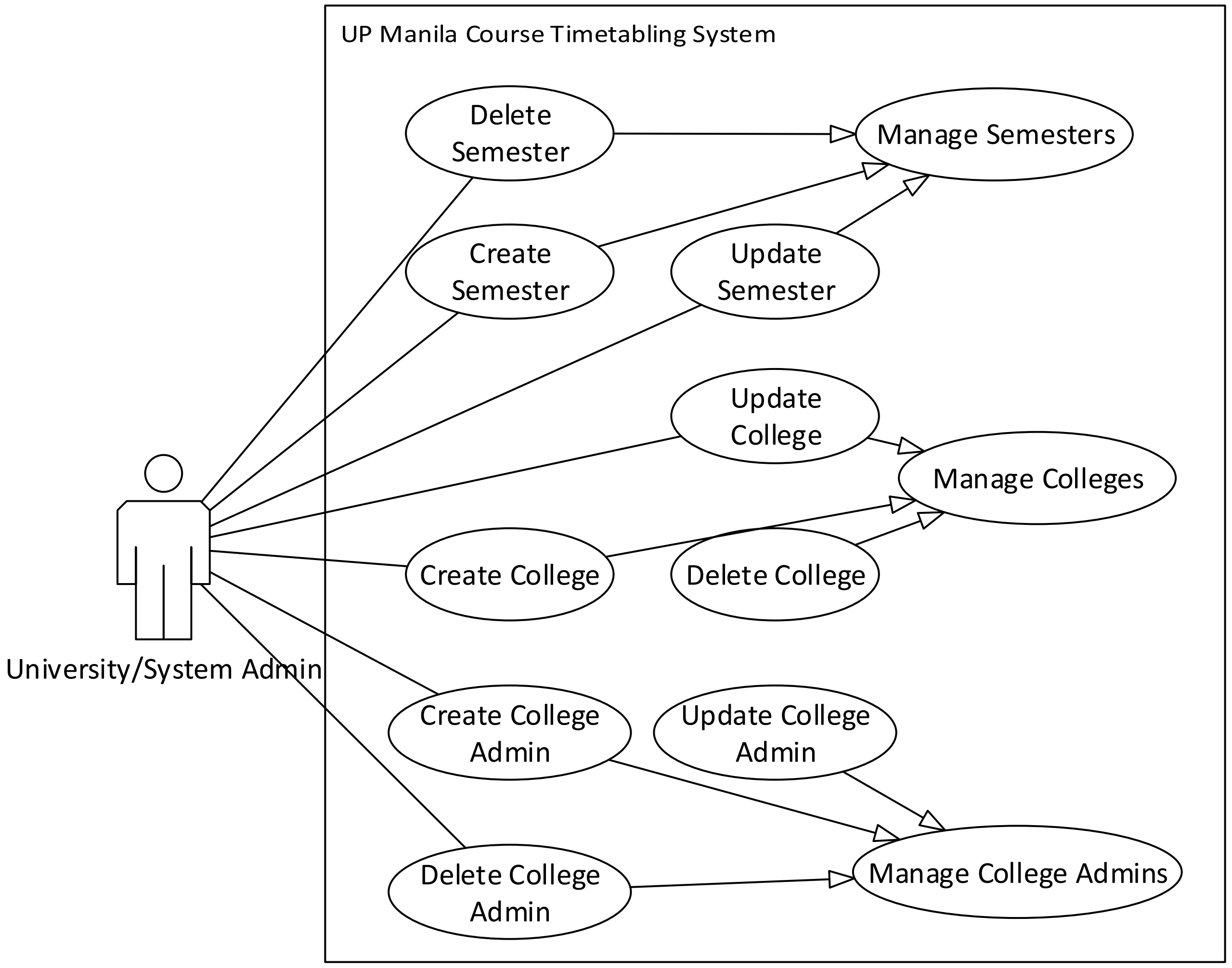


Figure 4: University Admin's Use Case Diagram

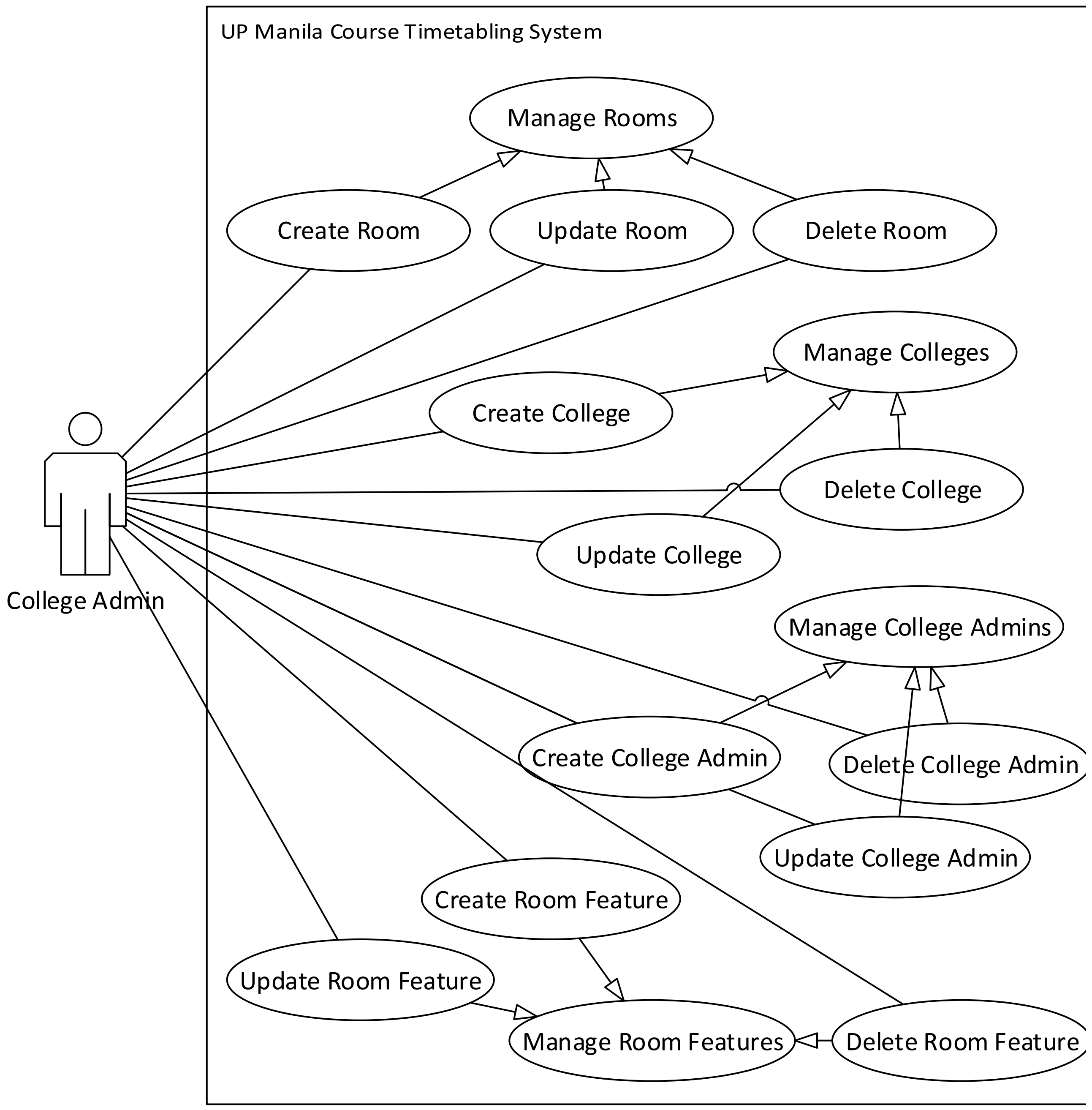


Figure 5: College Admin's Use Case Diagram

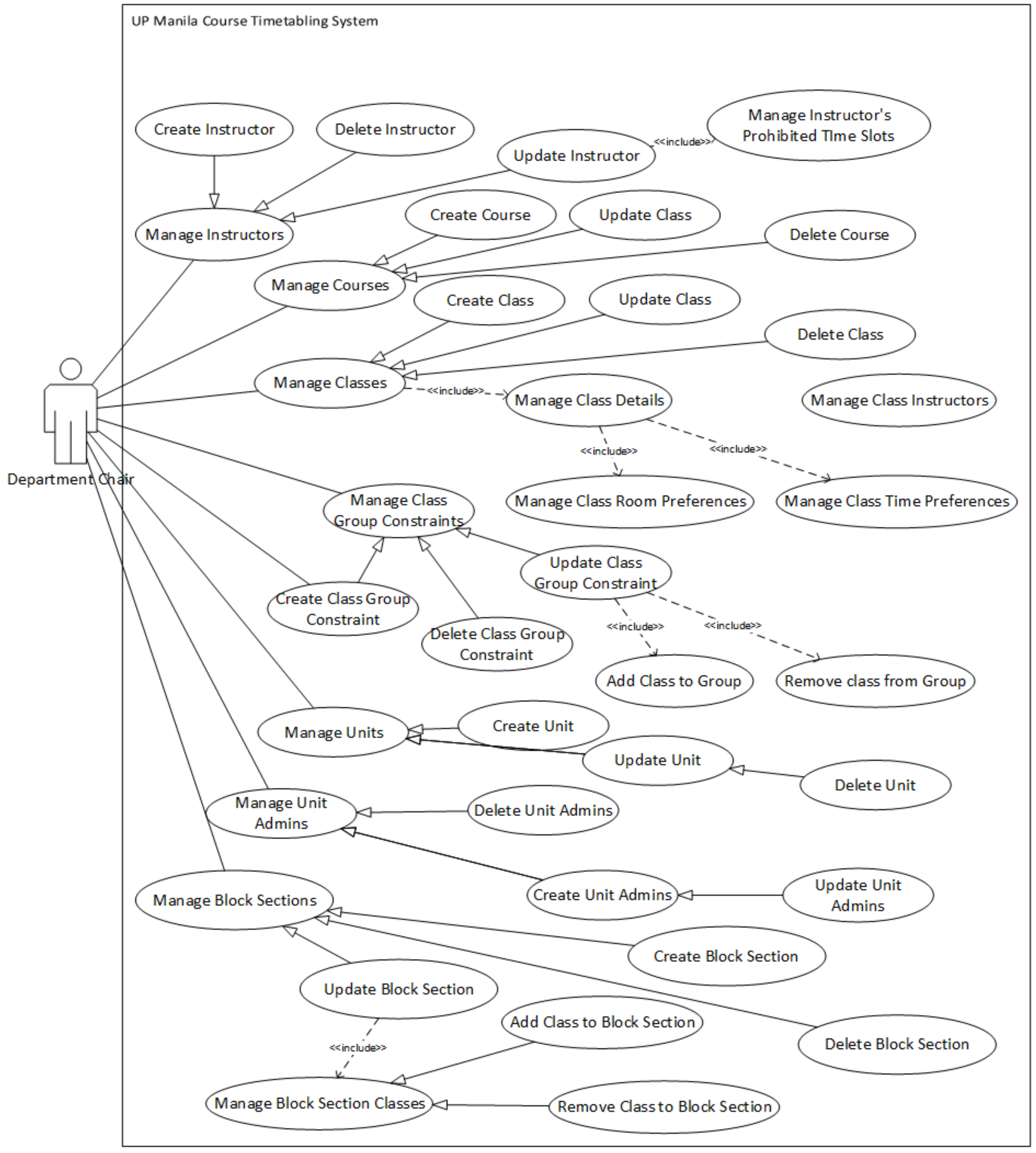


Figure 6: Department Chair's Use Case Diagram

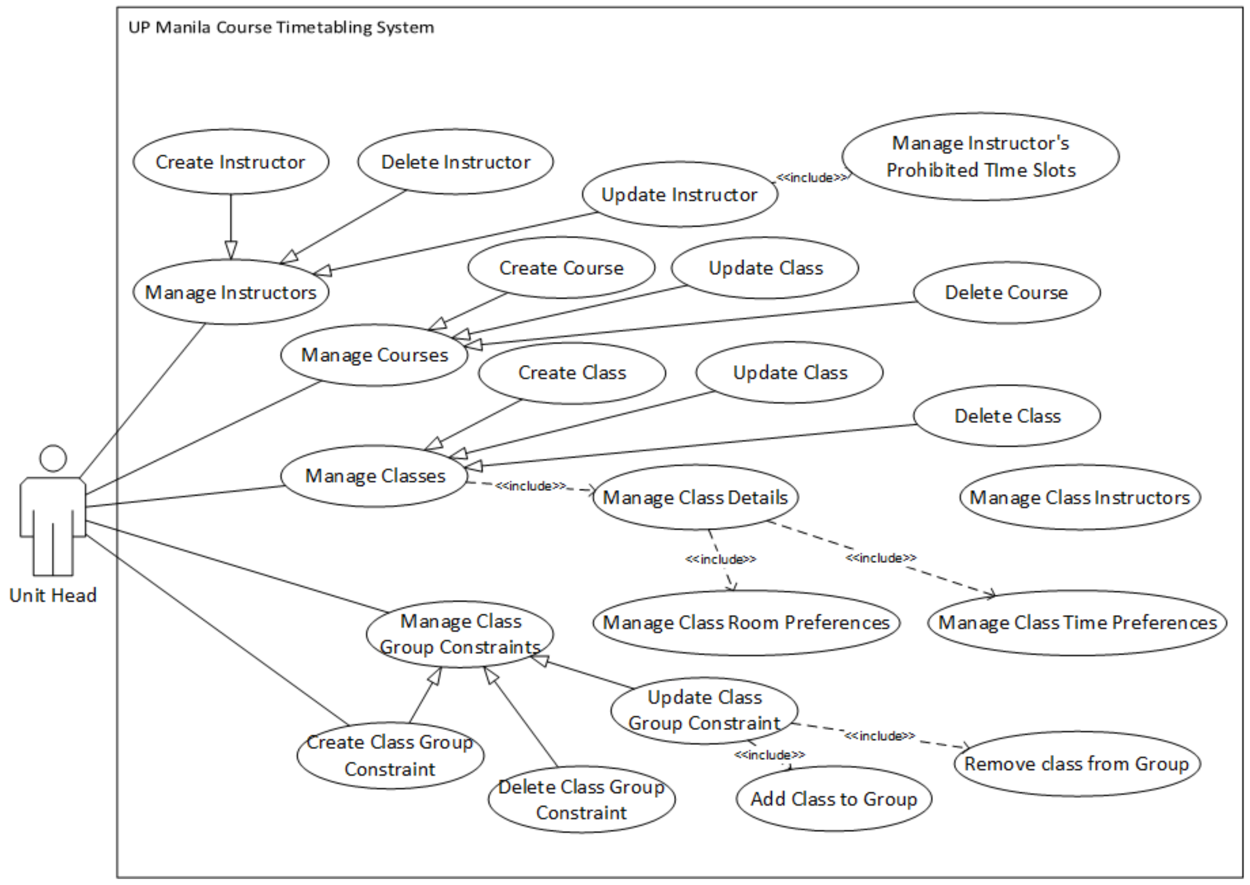


Figure 7: Unit Head's Use Case Diagram

B. Context Diagram

The system has the following types of users: system/university admin, college admin, department chair, and unit head.

The system admin’s task is to give data for the college admin accounts and the colleges’ details. The system admin can also view the timetable for the university once it has been generated.

The college admin can ask the system to generate a course timetable for the current semester. The system will then fetch the needed data to create a timetable from the database. These data are provided by the college admin, department chairs, and unit heads. After generating the timetable, the college admin will be able to view or export the timetable. If he/she is not satisfied with the generated timetable, he/she can regenerate it any time. Also, he also provides the details of the rooms of the college to the system.

The unit head is responsible for providing the data of the courses and instructors to the system. The unit head creates the classes together with its constraints using these data. The system will give the department the timetable for the courses it is offering for the semester.

The department chair is responsible for overseeing the unit heads. He/she can override the information provided by the unit heads. He/she is also responsible for managing the department's block sections data.

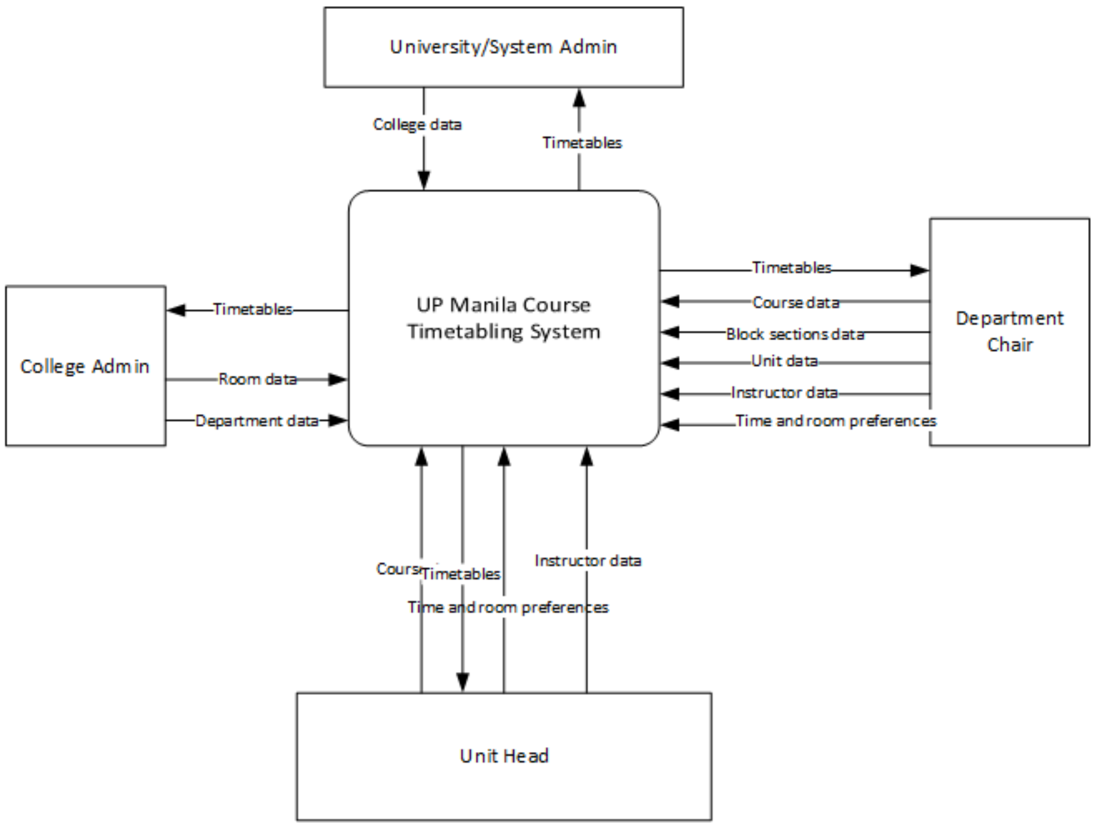


Figure 8: Context diagram for UPM Course Timetabling System

C. Entity Relationship Diagram

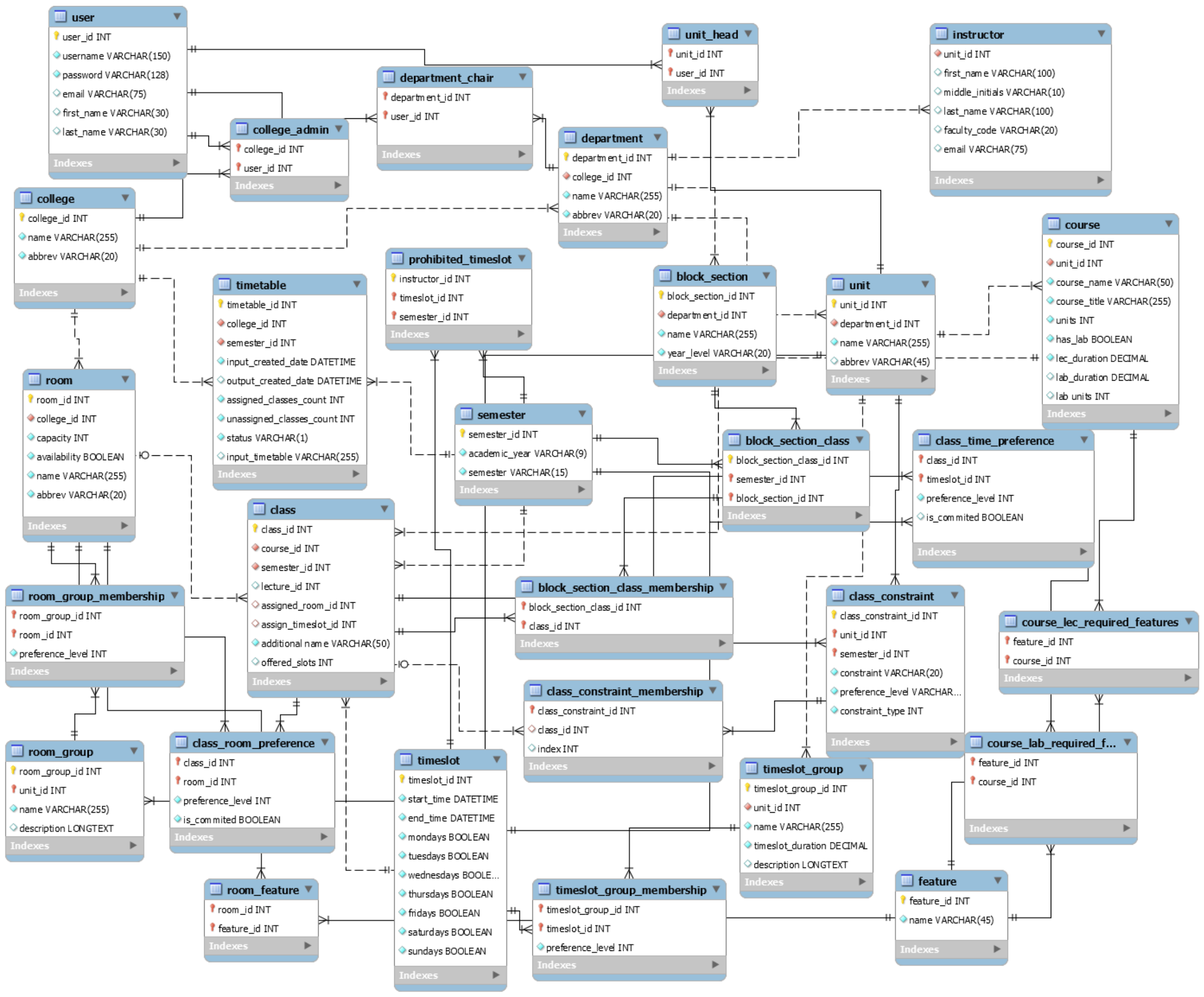


Figure 9: Entity relationship diagram for UPM Course Timetabling System

D. Data Dictionary

Table: instructor

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
unit_id	INT	Yes	No	Yes		Foreign key to a unit table object in

						which the instructor belongs
first_name	VARCHAR(100)	No	No	No		The first name of the instructor
middle_initials	VARCHAR(10)	No	No	No		The middle initials of the instructor
last_name	VARCHAR(100)	No	No	No		The last name of the instructor
faculty_code	VARCHAR(20)	No	No	No		The faculty code of the instructor
email	VARCHAR(75)	No	No	No		The email address of the instructor

Table 1: instructor table

Table: room

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
room_id	INT	Yes	Yes	No		Unique identifier for rooms
college_id	INT	Yes	No	Yes		Foreign key to a college table object in which the room belongs
capacity	INT	Yes	No	No		The seating capacity of the room
availability	TINYINT(1)	Yes	No	No	1	Determines if the room if available for use

name	VARCHAR(255)	Yes	No	No		Full name of the room
abbrev	VARCHAR(20)	Yes	No	No		A short name for the room

Table 2: room table

Table: college

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
college_id	INT	Yes	Yes	No		Unique identifier for colleges
name	VARCHAR(255)	Yes	No	No		Name of the college
abbrev	VARCHAR(20)	Yes	No	No		A short name for the college

Table 3: college table

Table: department

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
department_id	INT	Yes	Yes	No		Unique identifier for departments
college_id	INT	Yes	No	Yes		Foreign key to a college table object.

						Used for determining the college in which the department belongs
name	VARCHAR(255)	Yes	No	No		The name of the department
abbrev	VARCHAR(20)	Yes	No	No		A short name for the department

Table 4: department table

Table: user

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
user_id	INT	Yes	Yes	No		Unique identifier for user accounts
username	VARCHAR(150)	Yes	No	No		Will be used for logging into the system
password	VARCHAR(128)	Yes	No	No		Encrypted password of user using PBKDF2 algorithm with a SHA256 hash
email	VARCHAR(75)	No	No	No		Email address of the user
first_name	VARCHAR(30)	No	No	No		The first name of the user

last_name	VARCHAR(30)	No	No	No		The last name of the user

Table 5: user table

Table: class

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Defau lt**	**Comment**
class_id	INT	Yes	Yes	No		Unique identifier for classes
course_id	INT	Yes	No	Yes		Foreign key to a course table object
semester_i d	INT	Yes	No	Yes		Foreign key to a semester table object in which the class is offered
lecture_id	INT	No	No	No		Foreign key to a class table object. Used for determining the lecture class of laboratory classes.
assigned_r oom_id	INT	No	No	Yes		Foreign key to a room table object. Used for determining

						the final room for the class
assign_timeslot_id	INT	No	No	Yes		Foreign key to a timeslot table object. Used for determining the final timeslot for the class
additional name	VARCHAR(50)	Yes	No	No		Additional name for the class
offered_slots	INT	No	No	No	20	Number of slots available for the class

Table 6: class table

Table: course

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
course_id	INT	Yes	Yes	No		Unique identifier for courses
unit_id	INT	Yes	No	Yes		Foreign key to a unit table object in which the course is offered

course_name	VARCHAR(50)	Yes	No	No		The unique name of the course
course_title	VARCHAR(255)	Yes	No	No		The course title
units	INT	Yes	No	No		The number of units credited to the instructors for the lecture of the course
has_lab	TINYINT(1)	Yes	No	No	0	Used to determine if the course has laboratory classes
lec_duration	DECIMAL	Yes	No	No		The duration of lectures for the course
lab_duration	DECIMAL	No	No	No		The duration of laboratory classes for the course
lab units	INT	No	No	No		The number of units) credited to the instructors for the laboratory classes of the course (if any

Table 7: course table

Table: timetable

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
timetable_id	INT	Yes	Yes	No		Unique identifier for timetables
college_id	INT	Yes	No	Yes		Foreign key to a college table object
semester_id	INT	Yes	No	Yes		Foreign key to a semester table object
input_created_date	DATETIME	Yes	No	No		The date in which the timetable is created
output_created_date	DATETIME	No	No	No		The date in which the timetable output is created
assigned_classes_count	INT	Yes	No	No	0	The number of classes with assigned rooms

						and timeslots
unassigned_classes_count	INT	Yes	No	No	0	The number of classes with unassigned rooms or timeslots
status	VARCHAR(1)	Yes	No	No	'Q'	The current status of the generation of timetable. 'Q' - queued, 'I' - generating input, 'O' - generating output, 'S' - successful, 'F' - failed
input_timetable	VARCHAR(255)	No	No	No		The complete file name of the xml input for the timetable

Table 8: timetable table

Table: prohibited_timeslot

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
instructor_id	INT	Yes	Yes	No		Foreign key to an instructor table object
timeslot_id	INT	Yes	Yes	Yes		Foreign key to a timeslot table object
semester_id	INT	Yes	Yes	Yes		Foreign key to a semester table object

Table 9: prohibited_timeslot table

Table: unit -

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
unit_id	INT	Yes	Yes	No		Unique identifier for units
department_id	INT	Yes	No	Yes		Foreign key to a department table objects in which the unit belongs
name	VARCHAR(255)	Yes	No	No		The name of the unit
abbrev	VARCHAR(45)	No	No	No		A short name for the unit

Table 10: unit table

Table: semester

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
semester_id	INT	Yes	Yes	No		Unique identifier for semesters
academic_year	VARCHAR(9)	Yes	No	No		Example value is "2015-2016"
semester	VARCHAR(15)	Yes	No	No		The choices are "First Semester", Second Semester", and "Short Term".

Table 11: semester table

Table: college_admin

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
college_id	INT	Yes	Yes	Yes		Foreign key to a college table object
user_id	INT	Yes	Yes	Yes		Foreign key to a user table object

Table 12: college_admin table

Table: department_chair

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
department_id	INT	Yes	Yes	Yes		Foreign key to a department table object
user_id	INT	Yes	Yes	Yes		Foreign key to a user table object

Table 13: department_chair table

Table: unit_head

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
unit_id	INT	Yes	Yes	Yes		Foreign key to a unit table object
user_id	INT	Yes	Yes	Yes		Foreign key to a user table object

Table 14: unit head table

Table: feature

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
feature_id	INT	Yes	Yes	No		Unique identifier for features

name	VARCHAR(45)	Yes	No	No		The name of the feature

Table 15: feature table

Table: room_feature

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
room_id	INT	Yes	Yes	Yes		Foreign key to a room table object
feature_id	INT	Yes	Yes	Yes		Foreign key to a feature table object

Table 16: room_feature table

Table: block_section

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
block_section_id	INT	Yes	Yes	No		Unique identifier for block sections
department_id	INT	Yes	No	Yes		Foreign key to the department table object in which the block section

						belongs.
name	VARCHAR(255)	Yes	No	No		The name of the block section
year_level	VARCHAR(20)	Yes	No	No		The year level of students in the block section. The choices are "First Year", Second Year", Third Year", Fourth Year", "Fifth Year", and "Sixth Year".

Table 17: block_section table

Table: course_lec_required_features

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
feature_id	INT	Yes	Yes	Yes		Foreign key to a feature table object
course_id	INT	Yes	Yes	Yes		Foreign key to a course table object

Table 18: course_lec_required_features table

Table: course_lab_required_features

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
feature_id	INT	Yes	Yes	Yes		Foreign key to a feature table object
course_id	INT	Yes	Yes	Yes		Foreign key to a course table object

Table 19: course_lab_required_features table

Table: timeslot

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
timeslot_id	INT	Yes	Yes	No		Unique identifier for timeslots
start_time	DATETIME	Yes	No	No		Starting time for the time slot
end_time	DATETIME	Yes	No	No		Ending time for the time slot
mondays	TINYINT(1)	Yes	No	No	0	Used to determine if the time slot occurs on Mondays
tuesdays	TINYINT(1)	Yes	No	No	0	Used to determine if the time slot occurs on Tuesdays
wednesdays	TINYINT(1)	Yes	No	No	0	Used to determine

						if the time slot occurs on Wednesdays
thursdays	TINYINT(1)	Yes	No	No	0	Used to determine if the time slot occurs on Thursdays
fridays	TINYINT(1)	Yes	No	No	0	Used to determine if the time slot occurs on Fridays
saturdays	TINYINT(1)	Yes	No	No	0	Used to determine if the time slot occurs on Saturdays
sundays	TINYINT(1)	Yes	No	No	0	Used to determine if the time slot occurs on Sundays

Table 20: timeslot table

Table: class_time_preference

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
class_id	INT	Yes	Yes	Yes		Foreign key to a class table object in which the preference is linked
timeslot_id	INT	Yes	Yes	Yes		Foreign key to a timeslot table object
preference_level	INT	Yes	No	No		Preference level for the

						time slot. Preference choices: -2 - Strongly Preferred, -1 - Preferred, 0 - Neutral, 1 - Discouraged, 2 - Strongly Discouraged
is_commited	TINYINT(1)	No	No	No		Used for determining if the timeslot assignment for the class is final

Table 21: class_time_preference table

Table: class_room_preference

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
class_id	INT	Yes	Yes	Yes		Foreign key to a class table object
room_id	INT	Yes	Yes	Yes		Foreign key to a room table object
preference_level	INT	Yes	No	No	0	Preference level for the room. Preference choices: -2 - Strongly Preferred, -1 -

						Preferred, 0 - Neutral, 1 - Discouraged, 2 - Strongly Discouraged
is_commited	TINYINT(1)	Yes	No	No	0	Used for determining if the room assignment for the class is final

Table 22: class_room_preference table

Table: class_constraint

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
class_constraint_id	INT	Yes	Yes	No		Unique identifier for class constraints
unit_id	INT	Yes	Yes	Yes		Foreign key to a unit table object in which the class constraint belongs
semester_id	INT	Yes	Yes	Yes		Foreign key to a semester table object in which the class constraint if effective
constraint	VARCHAR (20)	Yes	No	No	'DIFF_TIME'	The constraint given to the class group. The choices

						are: 'SAME_ROOM', 'SAME_TIME', 'SAME_START', 'SAME_DAYS', 'BTB_TIME', 'BTB', 'NHB_GTE(1)', 'NHB_LT(6)', 'DIFF_TIME', 'BTB_DAY', 'PRECEDENCE', 'MIN_ROOM_USE', 'NDB_GT_1', 'FOLLOWING_DAY', 'EVERY_OTHER_DAY'
preference_level	VARCHAR (2)	Yes	No	No	'R'	Preference level for the constraint. Preference choices: 'R'- Required, '-2' - Strongly Preferred, '-1' - Preferred, '0' - Neutral, '1' - Discouraged, '2' - Strongly Discouraged, 'P' - Prohibited
constraint_type	INT	Yes	No	No	0	

Table 23: class_constraint table

Table: class_constraint_membership

Attributes

Name	Type	Not Null	PK	FK	Default	Comment
class_constraint_id	INT	Yes	Yes	Yes		Foreign key to a class_constraint table object
class_id	INT	No	No	Yes		Foreign key to a class object
index	INT	No	No	No		The order in which the class is added to the class constraint group

Table 24: class_constraint_membership table

Table: block_section_class

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
block_section_class_id	INT	Yes	Yes	No		Unique identifier for block section classes
semester_id	INT	Yes	Yes	Yes		Foreign key to a semester table object
block_section_id	INT	Yes	Yes	Yes		Foreign key to a block_section table object

Table 25: block_section_class table

Table: block_section_class_membership

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
block_section_class_id	INT	Yes	Yes	Yes		Foreign key to a block section class table object.
class_id	INT	Yes	Yes	Yes		Foreign key to a class table object

Table 26: block_section_class_membership table

Table: room_group

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
room_group_id	INT	Yes	Yes	No		Unique identifier for room groups
unit_id	INT	Yes	Yes	Yes		Foreign key to a unit table object in which the room group belongs
name	VARCHAR(255)	Yes	No	No		The name of the room group
description	LONGTEXT	No	No	No		Description for the room group

Table 27: room_group table

Table: room_group_membership

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
room_group_id	INT	Yes	Yes	Yes		Foreign key to a room_group table object
room_id	INT	Yes	Yes	Yes		Foreign key to a room table object
preference_level	INT	Yes	No	No		Preference level for the room. Preference choices: -2 - Strongly Preferred, -1 - Preferred, 0 - Neutral, 1 - Discouraged, 2 - Strongly Discouraged

Table 28: room_group_membership table

Table: timeslot_group

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
timeslot_group_id	INT	Yes	Yes	No		Unique identifier for timeslot groups

unit_id	INT	Yes	No	Yes		Foreign key to a unit table object in which the time slot group is linked
name	VARCHAR(255)	Yes	No	No		The name of the timeslot group
timeslot_duration	DECIMAL	Yes	No	No		The duration of timeslots in the timeslot group
description	LONGTEXT	No	No	No		The description of the timeslot group

Table 29: timeslot_group table

Table: timeslot_group_membership

Attributes						
Name	**Type**	**Not Null**	**PK**	**FK**	**Default**	**Comment**
timeslot_group_id	INT	Yes	Yes	Yes		Foreign key to a timeslot_group table object

timeslot_id	INT	Yes	Yes	Yes		Foreign key to a timeslot table object
preference_level	INT	Yes	No	No		Preference level for the timeslot. Preference choices: -2 - Strongly Preferred, -1 - Preferred, 0 - Neutral, 1 - Discouraged, 2 - Strongly Discouraged

Table 30: timeslot_group_membership table

D. Timetable Solver Constraints

Hard Constraints

- Any two classes cannot take place in the same room at the same time period.
- An instructor must not be assigned to more than one class at the same time period.
- Classes of courses in the same curriculum must not be scheduled at the same time period.
- An instructor cannot be assigned to teach a class in a time period where he/she is not available.
- A class must be assigned to a room that contains the facilities that the class need.

Soft Constraints

- The number of slots for a class must not exceed or be less than the capacity of the assigned room.

- A class should take place at a time period and room that are preferred by the department chair or unit head (which may have the input of instructors).
- All meetings of a class should take place at the same room and at the same time.
- The meetings of a class must be spread into a given number of days.
- A group of classes with an assigned precedence must be scheduled in that order.
- The number of rooms used by a group of classes must be minimized (if specified by the department chair or unit head).
- A lecture class must take place before its corresponding laboratory classes.
- A class must be assigned to a room that is preferred by the department chair or unit head.

E. Timetable Solver Implementation

Data Input

The server fetches the data from the database. These data will be provided by the users of the system.

Solver algorithm flow

The algorithm that is used to solve the timetabling problem is a two-stage optimization algorithm. It consists of four phases to generate a timetable like the one used in [24]. The first phase is used in order to find a feasible timetable (all hard constraints are satisfied). The next three phases are used in order to improve the solution. The search ends when the predetermined time limit is reached, and the best solution is returned afterwards.

- First stage

The first stage consists of the construction phase. The aim of the construction phase is to generate a timetable with no violations in the hard constraints. The iterative forward search algorithm is used in this phase. The algorithm starts with all the classes unassigned to a room and timeslot. Every iteration, an unassigned class is selected and a room and timeslot from its domain is assigned to it. If this assignment causes hard constraints violations with other assignments, these conflicting assignments are unassigned. The phase ends when all the classes are assigned to a room and timeslot.

- Second stage

The second stage consists of the following phases: the hill climbing phase, the great deluge phase, and the simulated annealing phase. These phases are used in order to improve the feasible solution found in the construction phase. First, the hill climbing algorithm is used to find the local optimum. Once the solution cannot be improved further, it switches to the great deluge phase where the great deluge algorithm is used to find a new local optimum. Once the GD algorithm can no longer improve the solution, the simulated annealing phase starts. The SA algorithm is used to again improve the current solution. The control goes back to the hill climbing phase when the solution cannot be further improved in the simulated annealing phase. The cycle continues until a time limit is reached.

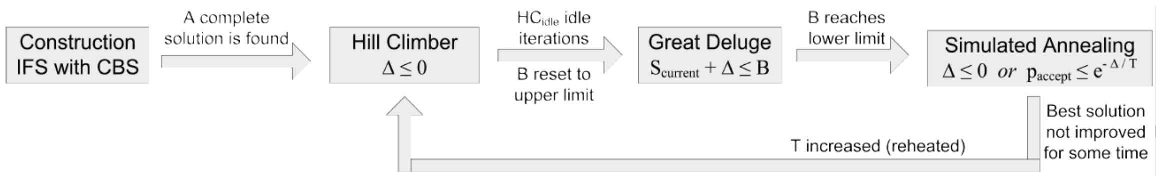

Construction IFS with CBS
A complete solution is found
Hill Climber $\Delta \leq 0$
HC_{idle} idle iterations
B reset to upper limit
Great Deluge $S_{current} + \Delta \leq B$
B reaches lower limit
Simulated Annealing $\Delta \leq 0$ or $p_{accept} \leq e^{-\Delta/T}$
Best solution not improved for some time
T increased (reheated)

Figure 5: Flow of the timetabling algorithm by Muller

F. System Architecture

The system is composed of a Web application which handles both the input of the data, and the manipulation of it to generate timetables. The online application is made using Python. The system connects to a PostgreSQL database.

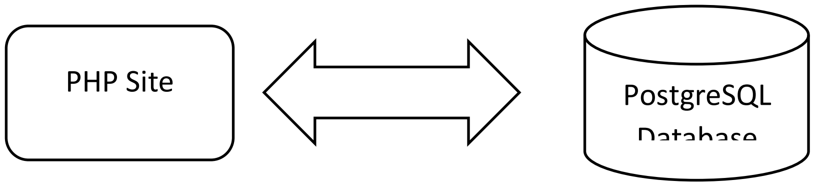


Figure 6: System Architecture

G. Technical Architecture

Using the online application, users can input the necessary data for generating timetables. The college admin is responsible for generating timetables which is also done using the Web application. The data is being saved to a PostgreSQL database.

Minimum System Requirements

1. 1 GB RAM

2. 1.80GHz processor

3. 32-bit operating system

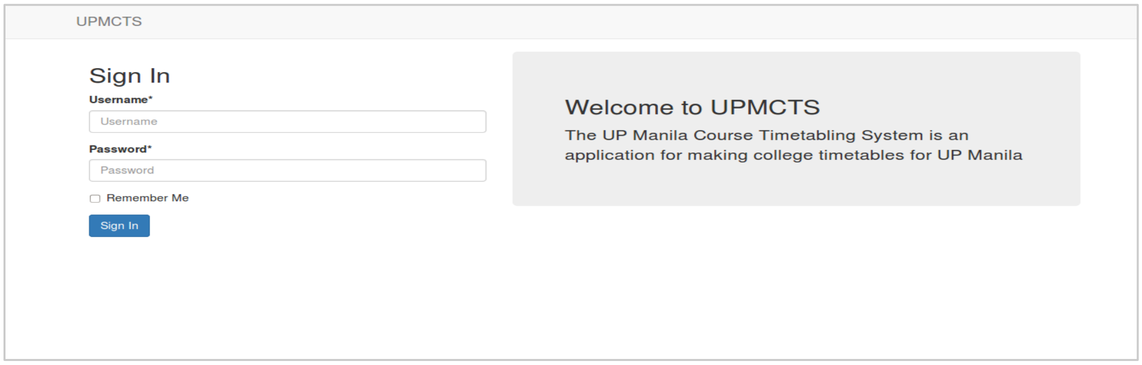
4. Linux operating system (for the server) with Python version 2.7 or greater, PostgreSQL server, and JRE (Java Runtime Environment)

6. Reliable internet connection

V. Results

A. Home/Login Page

Visiting the site will take users to the home/login page which is shown in Figure 10. Only accounts created by the admins can be used to log into the system.



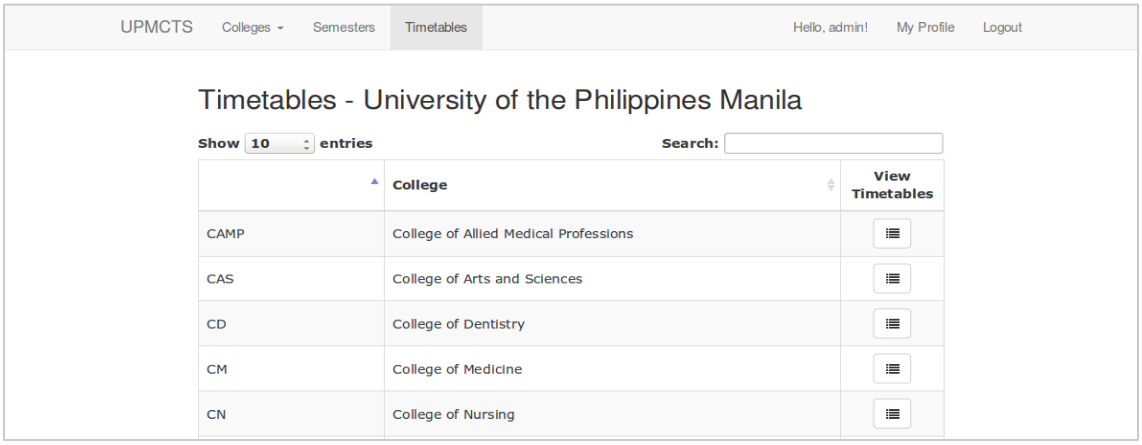
Welcome to UPMCTS

The UP Manila Course Timetabling System is an application for making college timetables for UP Manila

Figure 10: Home/Login page

B. Timetables

All authenticated users will be able to view the timetables of each college. Clicking the Timetables tab on the navigation menu will take the user to the list of colleges as shown in Figure 11.



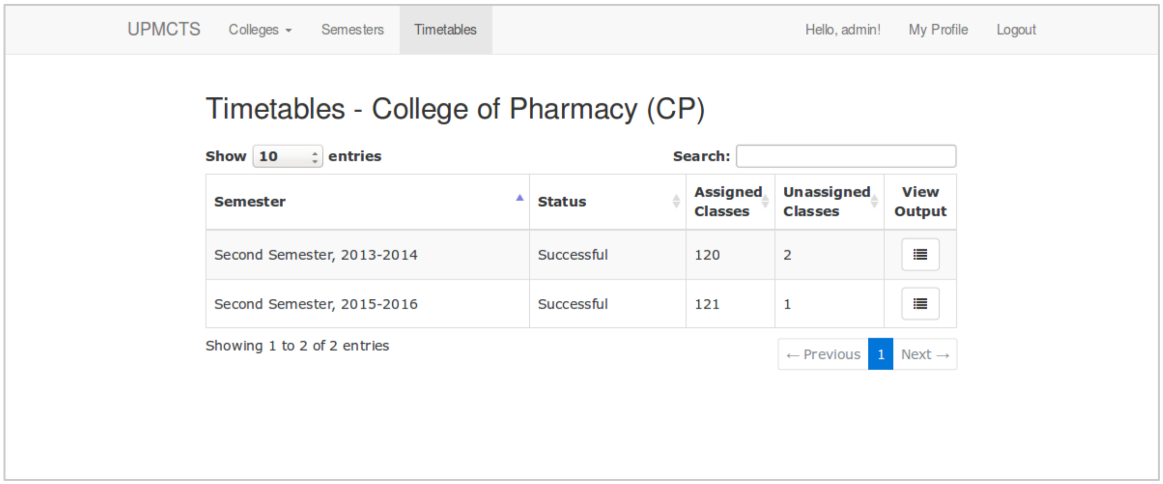
Timetables - University of the Philippines Manila

Show 10 entries Search:

	College	View Timetables
CAMP	College of Allied Medical Professions	☰
CAS	College of Arts and Sciences	☰
CD	College of Dentistry	☰
CM	College of Medicine	☰
CN	College of Nursing	☰

Figure 11: Timetables page

Clicking a button on the View Timetables tab will take the user to the list of timetables of the corresponding college. As shown in Figure 12, there can only be one timetable per semester. Included in the table are the status of the generated timetable, and the number of assigned and unassigned classes.



Timetables - College of Pharmacy (CP)

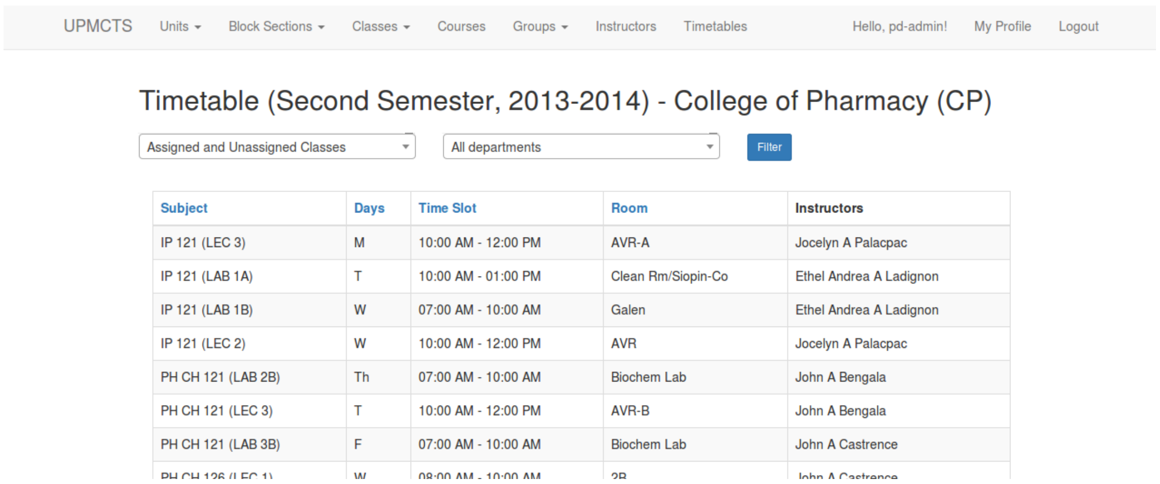
Show 10 entries Search:

Semester	Status	Assigned Classes	Unassigned Classes	View Output
Second Semester, 2013-2014	Successful	120	2	☰
Second Semester, 2015-2016	Successful	121	1	☰

Showing 1 to 2 of 2 entries

Figure 12: Semester Timetables page

Clicking the View Output shown in Figure 13 will take the user to the Timetable Output page that lists all the classes offered by the college for a particular semester. A dropdown box is available for filtering the timetable by department.

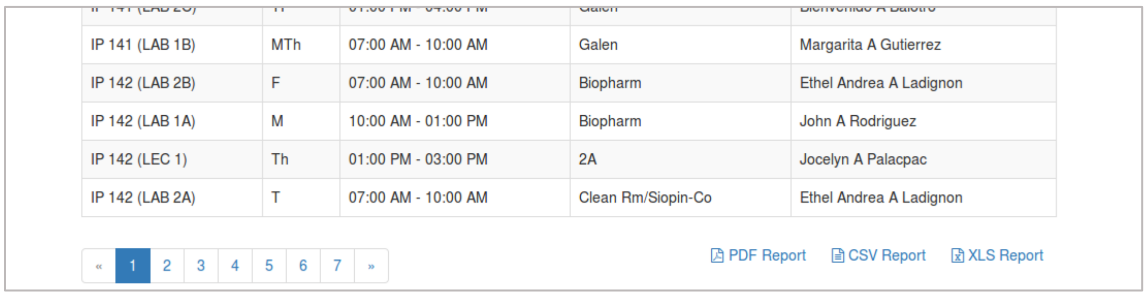


Timetable (Second Semester, 2013-2014) - College of Pharmacy (CP)

Subject	Days	Time Slot	Room	Instructors
IP 121 (LEC 3)	M	10:00 AM - 12:00 PM	AVR-A	Jocelyn A Palacpac
IP 121 (LAB 1A)	T	10:00 AM - 01:00 PM	Clean Rm/Siopin-Co	Ethel Andrea A Ladignon
IP 121 (LAB 1B)	W	07:00 AM - 10:00 AM	Galen	Ethel Andrea A Ladignon
IP 121 (LEC 2)	W	10:00 AM - 12:00 PM	AVR	Jocelyn A Palacpac
PH CH 121 (LAB 2B)	Th	07:00 AM - 10:00 AM	Biochem Lab	John A Bengala
PH CH 121 (LEC 3)	T	10:00 AM - 12:00 PM	AVR-B	John A Bengala
PH CH 121 (LAB 3B)	F	07:00 AM - 10:00 AM	Biochem Lab	John A Castrence
PH CH 126 (LEC 1)	W	08:00 AM - 10:00 AM	2B	John A Castrence

Figure 13: Timetable Output page

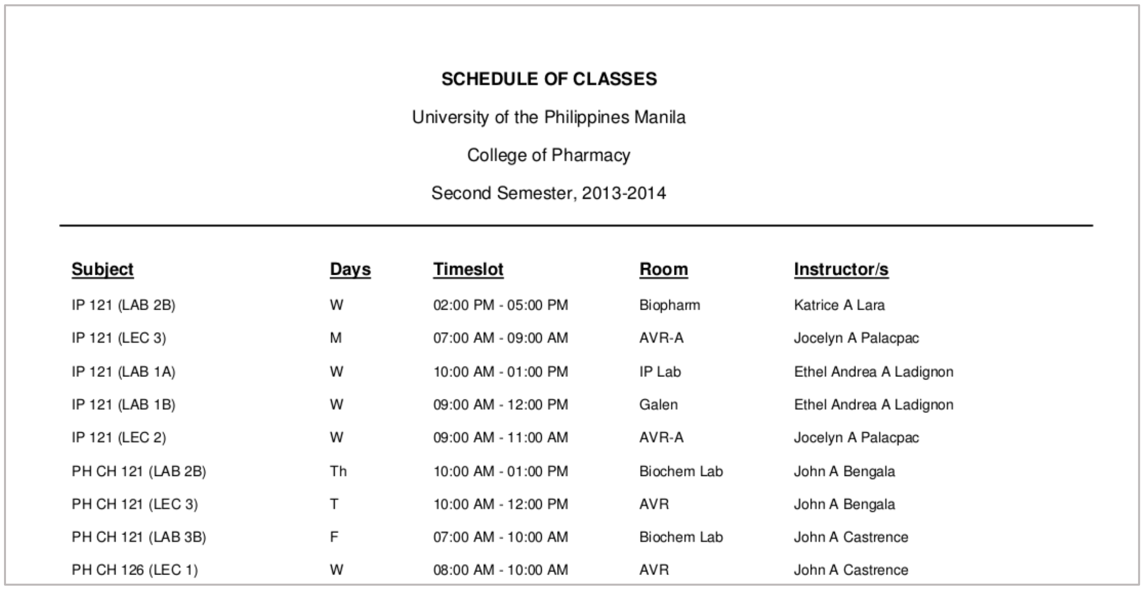
The user will be able to export the timetable in PDF, CSV, or XLS format by clicking one of the buttons at the bottom of the page (See Figure 14).



IP 141 (LAB 1B)	MTh	07:00 AM - 10:00 AM	Galen	Margarita A Gutierrez
IP 142 (LAB 2B)	F	07:00 AM - 10:00 AM	Biopharm	Ethel Andrea A Ladignon
IP 142 (LAB 1A)	M	10:00 AM - 01:00 PM	Biopharm	John A Rodriguez
IP 142 (LEC 1)	Th	01:00 PM - 03:00 PM	2A	Jocelyn A Palacpac
IP 142 (LAB 2A)	T	07:00 AM - 10:00 AM	Clean Rm/Siopin-Co	Ethel Andrea A Ladignon

Figure 14: Buttons for exporting the timetable

A sample PDF output is shown in Figure 15.



SCHEDULE OF CLASSES

University of the Philippines Manila

College of Pharmacy

Second Semester, 2013-2014

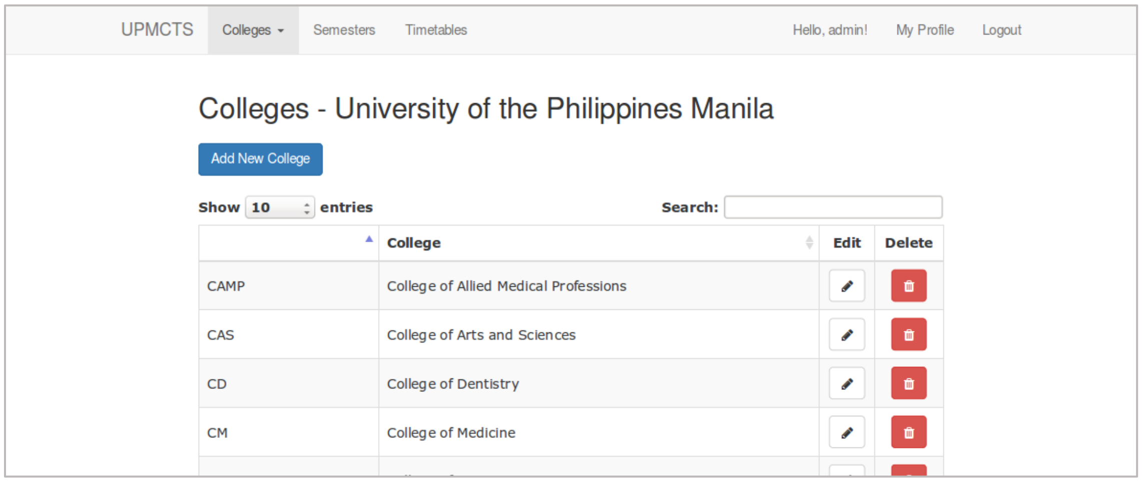
Subject	Days	Timeslot	Room	Instructor/s
IP 121 (LAB 2B)	W	02:00 PM - 05:00 PM	Biopharm	Katrice A Lara
IP 121 (LEC 3)	M	07:00 AM - 09:00 AM	AVR-A	Jocelyn A Palacpac
IP 121 (LAB 1A)	W	10:00 AM - 01:00 PM	IP Lab	Ethel Andrea A Ladignon
IP 121 (LAB 1B)	W	09:00 AM - 12:00 PM	Galen	Ethel Andrea A Ladignon
IP 121 (LEC 2)	W	09:00 AM - 11:00 AM	AVR-A	Jocelyn A Palacpac
PH CH 121 (LAB 2B)	Th	10:00 AM - 01:00 PM	Biochem Lab	John A Bengala
PH CH 121 (LEC 3)	T	10:00 AM - 12:00 PM	AVR	John A Bengala
PH CH 121 (LAB 3B)	F	07:00 AM - 10:00 AM	Biochem Lab	John A Castrence
PH CH 126 (LEC 1)	W	08:00 AM - 10:00 AM	AVR	John A Castrence

Figure 15: timetable output in PDF format

University/System Admin

1. Colleges

Upon logging in, the university admin will be redirected to the page that list all the colleges in the university (See Figure 16). On this page, the university admin can create, edit, and delete colleges.



Colleges - University of the Philippines Manila

	College	Edit	Delete
CAMP	College of Allied Medical Professions	✎	🗑
CAS	College of Arts and Sciences	✎	🗑
CD	College of Dentistry	✎	🗑
CM	College of Medicine	✎	🗑

Figure 16: College List page

As shown in Figure 17, clicking the Add New College button will open a popup for creating a new college. After submitting the form, the popup will be closed, and the list of college will be updated to include the newly created one.

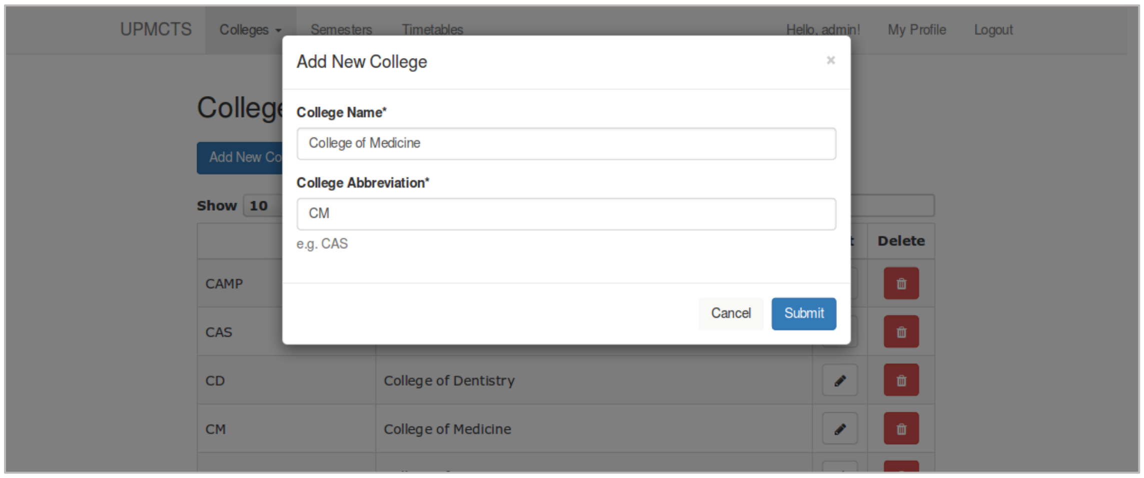


Figure 17: College Create form

The College Update form is similar to the College Create form (See Figure 18), except that it is already prepopulated with the information of the college being updated.

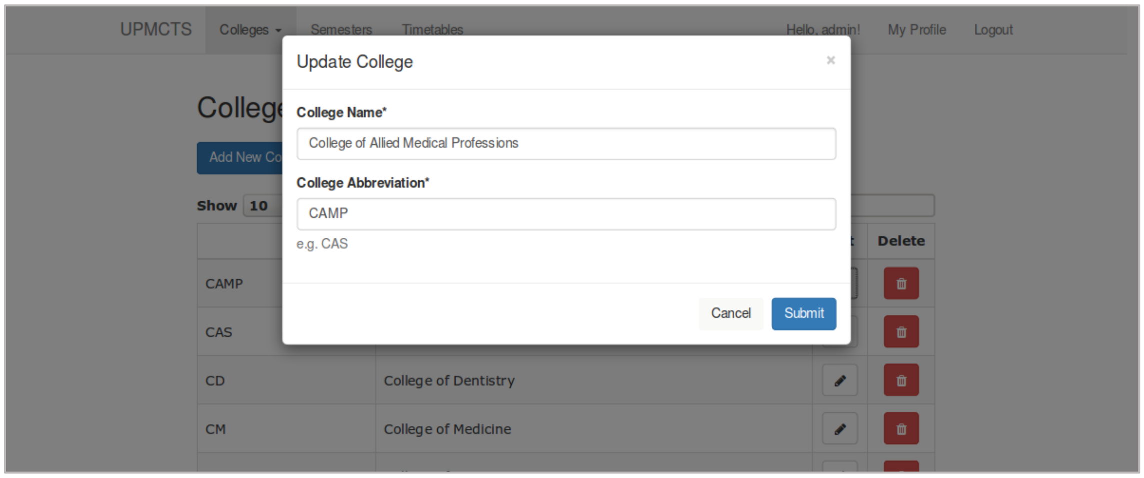
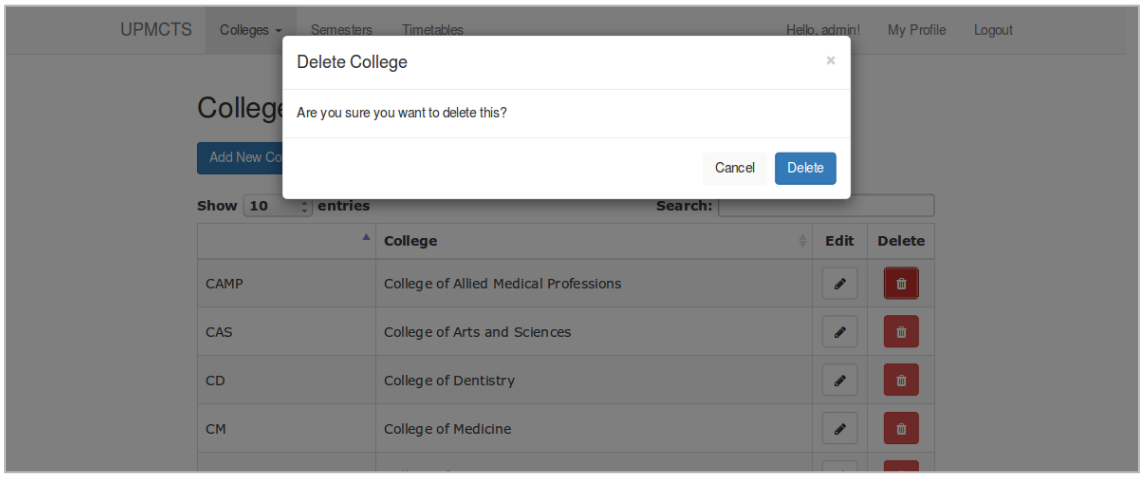


Figure 18: College Update form

On the College List page, clicking the red button at the right side of a college will open a confirmation popup for deleting the corresponding college (See Figure 19). After confirming, the popup will close, and the list of colleges will be updated to remove the recently deleted college.



Delete College

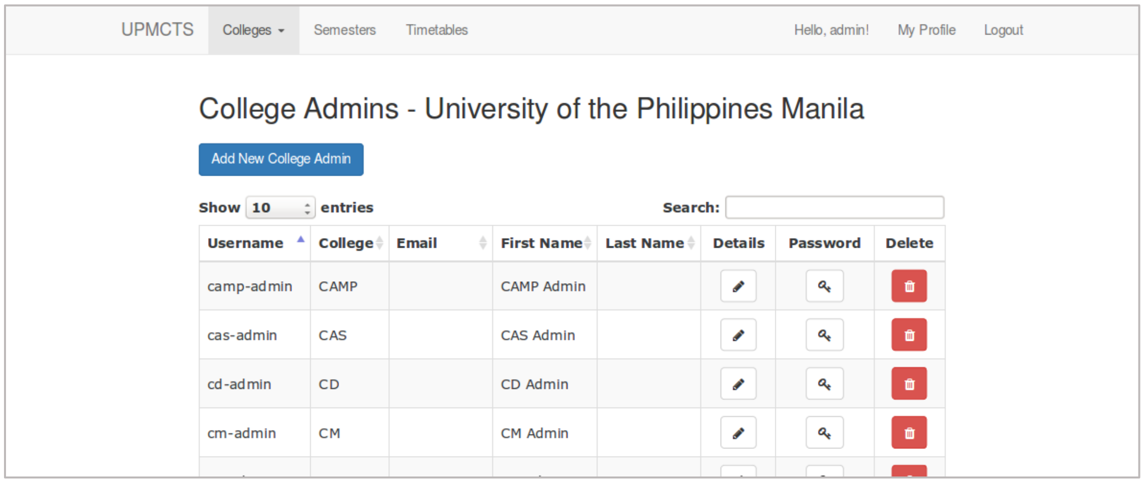
Are you sure you want to delete this?

	College	Edit	Delete
CAMP	College of Allied Medical Professions		
CAS	College of Arts and Sciences		
CD	College of Dentistry		
CM	College of Medicine		

Figure 19: College Delete form

2. College Admins

Under the Colleges tab on the navigation bar, there is a link name “College Admins” which will take the user to the College Admin List page (See Figure 20). On this page, the university admin can create, edit, and delete college admin accounts.

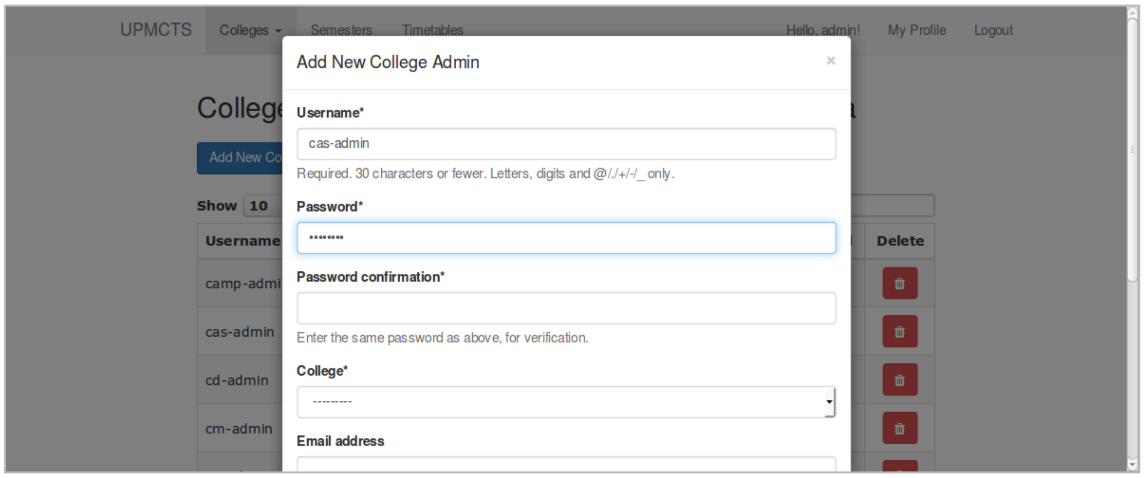


College Admins - University of the Philippines Manila

Username	College	Email	First Name	Last Name	Details	Password	Delete
camp-admin	CAMP		CAMP Admin				
cas-admin	CAS		CAS Admin				
cd-admin	CD		CD Admin				
cm-admin	CM		CM Admin				

Figure 20: College Admin List page

Clicking the Add New College Admin button will open a popup for creating a new college admin which can be seen in Figure 21. The form includes the following fields: username, password, password confirmation, college, email address, first name, and last name. After submitting the form, the new college admin will be added to the list and an account will be created which can be used for logging in as a college admin.



Add New College Admin

Username*
cas-admin
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password*
••••••••

Password confirmation*

Enter the same password as above, for verification.

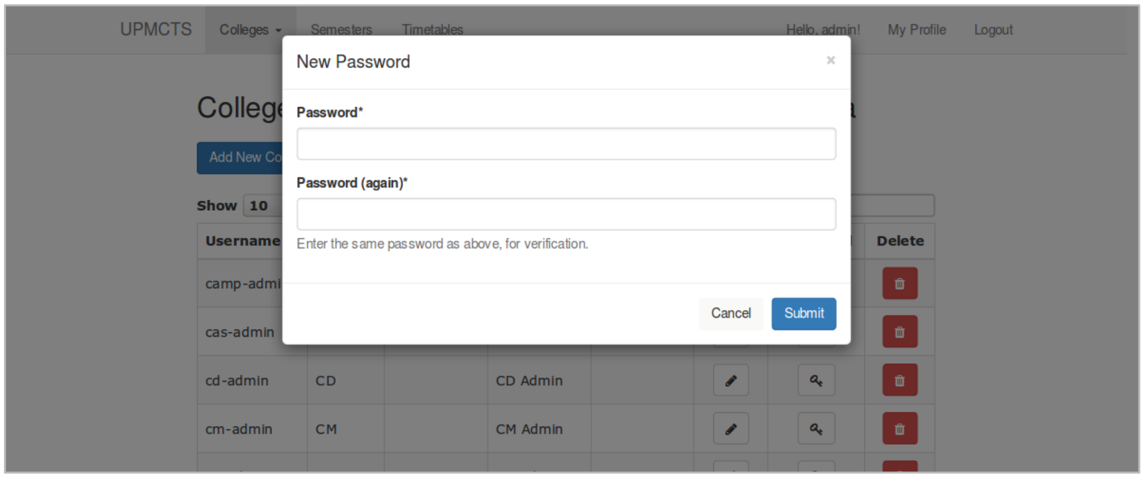
College*

Email address

camp-admi
cas-admin
cd-admin
cm-admin

Figure 21: College Admin Create form

Clicking a button under the Details column in Figure 21 will open a similar popup to the College Create form where the university admin can update the username, college, email, first name, and last name of the associated college admin. Clicking a button under the Password column shown in Figure 20 will open a popup (See Figure 22) where the university admin can update the password of the college admin.



New Password

Password*

Password (again)*

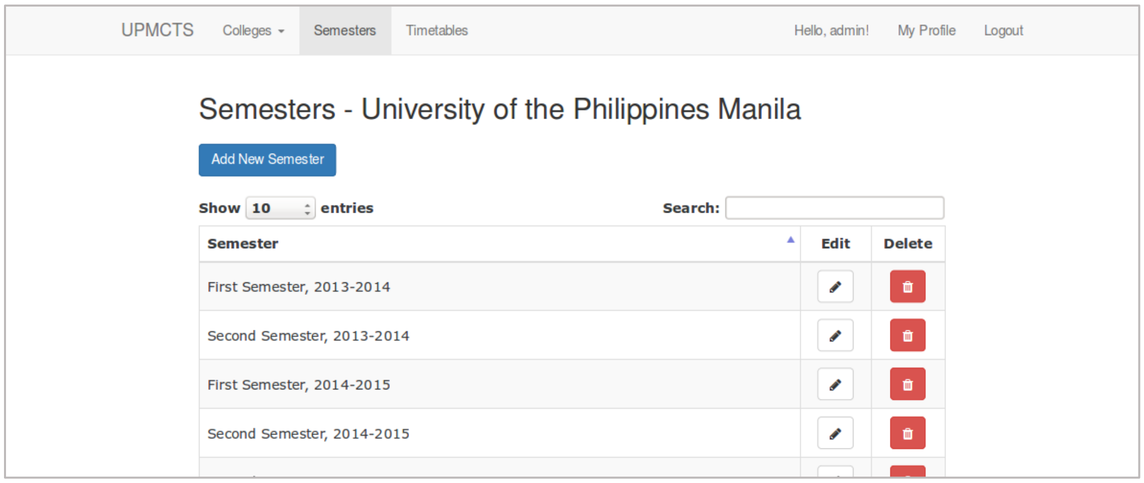
Enter the same password as above, for verification.

Username				Delete
camp-admi				
cas-admin				
cd-admin	CD		CD Admin	
cm-admin	CM		CM Admin	

Figure 22: College Admin Password Change form

3. Semesters

The first thing that must be done when creating a timetable is creating a semester. Classes are linked to semesters, so it is mandatory to create a semester first before other users can add classes to the system. The university admin can do this by clicking the Semesters tab on the navigation menu. This will take him/her to the Semester List page (See Figure 23) where he/she will be able to create, edit, and remove semesters.

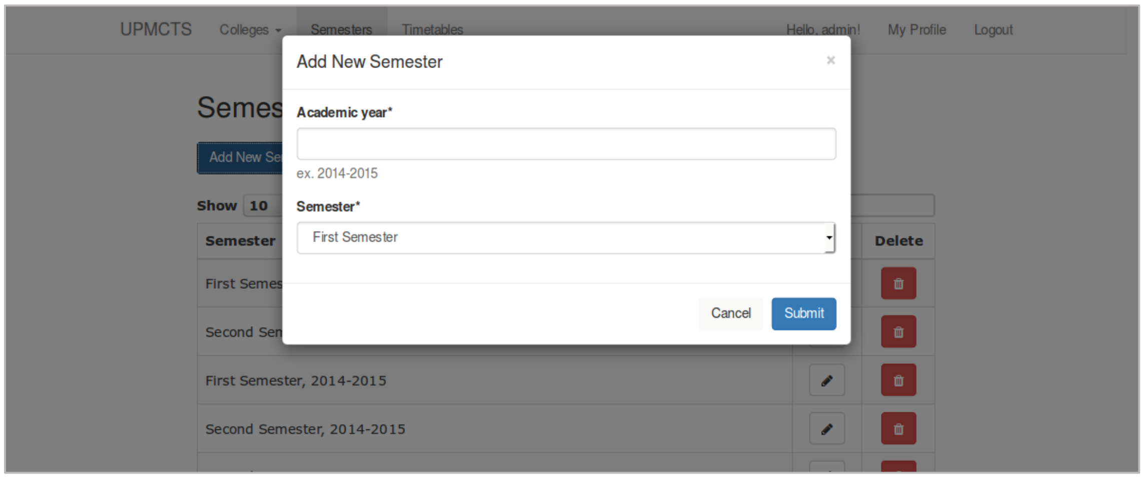


Semesters - University of the Philippines Manila

Semester	Edit	Delete
First Semester, 2013-2014		
Second Semester, 2013-2014		
First Semester, 2014-2015		
Second Semester, 2014-2015		

Figure 23: Semester List page

Clicking the Add New Semester button will open a popup where the university admin can create a new semester as shown in Figure 24. Submitting the form will add the semester to the list. The Edit and Delete buttons are also available for updating or deleting a particular semester.



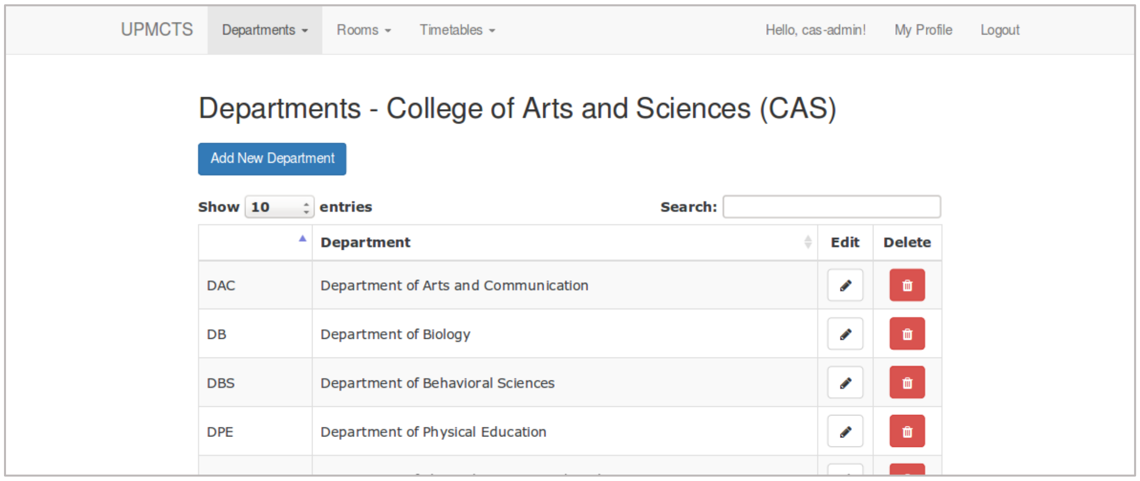
Semester
First Semester, 2014-2015
Second Semester, 2014-2015

Figure 24: Semester Create Form

C. College Admin

1. Departments

The college admin is responsible for managing the departments in the college. He/she can do this by clicking the Departments tab under the Department dropdown on the navigation bar. Upon clicking, the college admin will see the list of departments in the college as seen in Figure 25.



Departments - College of Arts and Sciences (CAS)

	Department	Edit	Delete
DAC	Department of Arts and Communication		
DB	Department of Biology		
DBS	Department of Behavioral Sciences		
DPE	Department of Physical Education		

Figure 25: Department List page

Clicking the Add New Department button will open a popup for creating a new department (See Figure 26). Similar to the College List page, the Department List page also has buttons for updating or deleting departments.

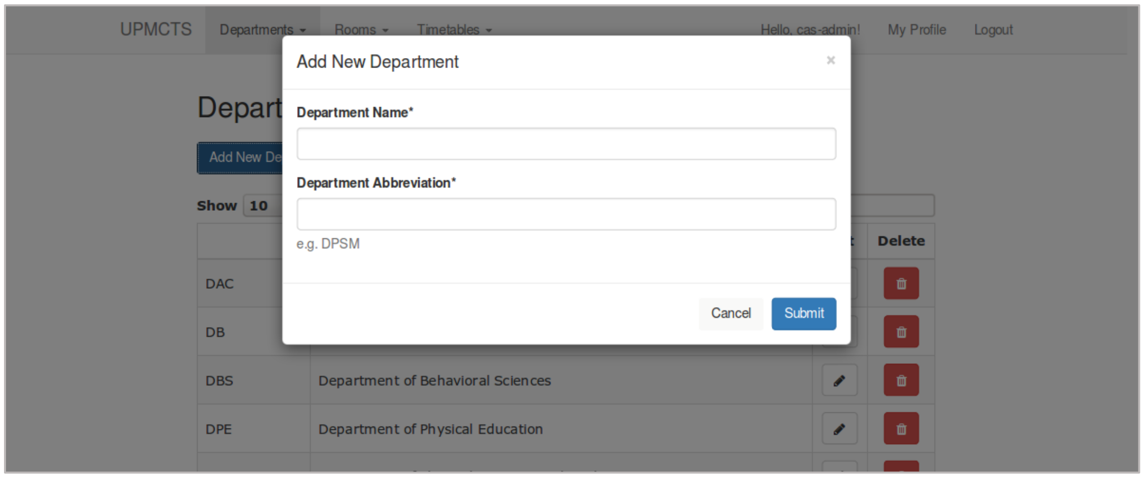
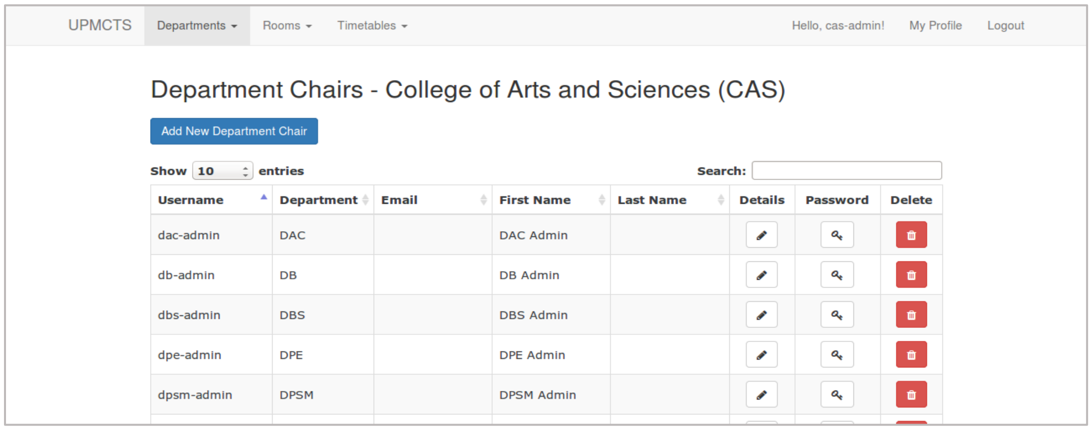


Figure 26: Department Create Form

2. Department Admins

The college admin can create, update, or delete department admin account by clicking the Department Admins tab under the Department dropdown on the navigation bar (See

Figure 27). The page has the same features as the College Admins page of the university admin, except that the accounts are linked to departments instead of colleges.



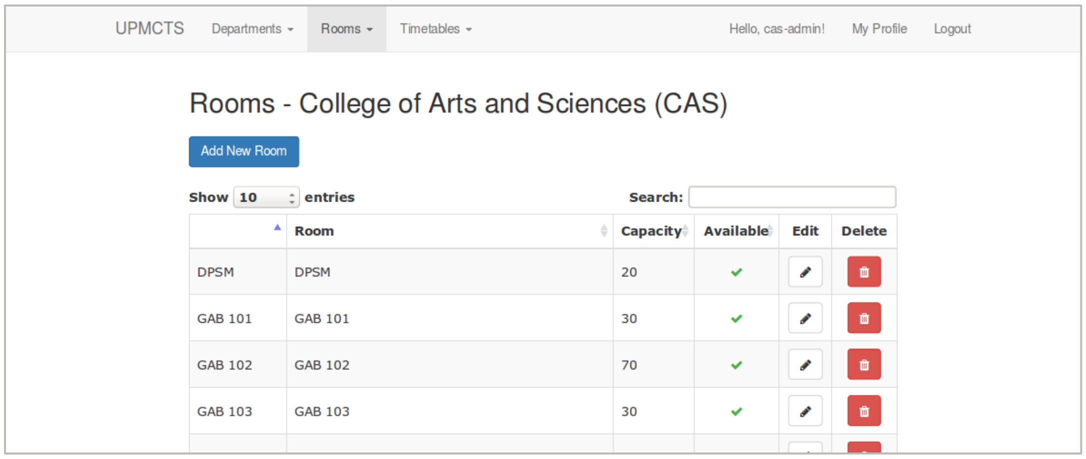
Department Chairs - College of Arts and Sciences (CAS)

Username	Department	Email	First Name	Last Name	Details	Password	Delete
dac-admin	DAC		DAC Admin				
db-admin	DB		DB Admin				
dbs-admin	DBS		DBS Admin				
dpe-admin	DPE		DPE Admin				
dpsm-admin	DPSM		DPSM Admin				

Figure 27: Department Admin List page

3. Rooms

College admins are responsible for managing the rooms in their respective colleges. He/she can do this by clicking the Rooms tab under the Rooms dropdown on the navigation bar. On this page (See Figure 28), the college admin can create, edit, and delete rooms related to the college.

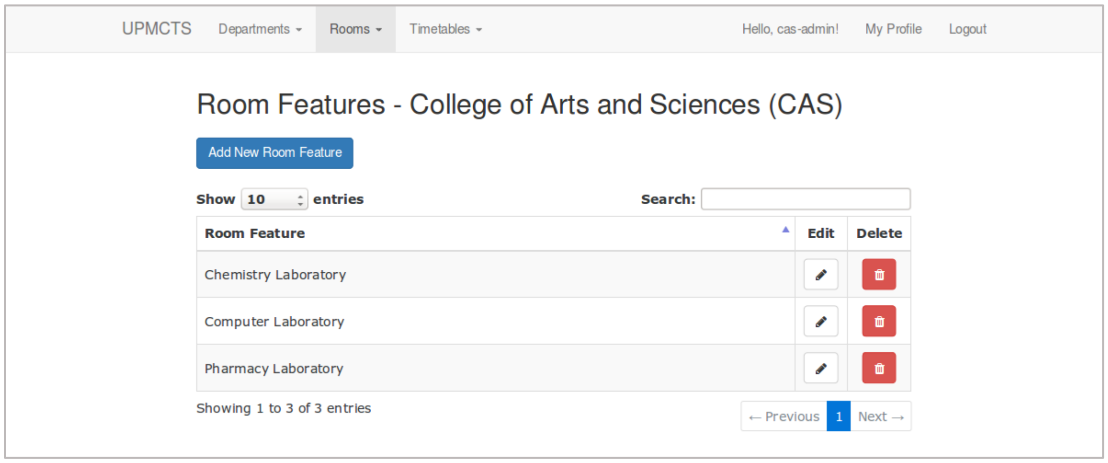


Rooms - College of Arts and Sciences (CAS)

	Room	Capacity	Available	Edit	Delete
DPSM	DPSM	20	✓		
GAB 101	GAB 101	30	✓		
GAB 102	GAB 102	70	✓		
GAB 103	GAB 103	30	✓		

Figure 28: Room List page

He/she can also create room features which can then be attached to any room as seen in Figure 29. This is useful when choosing a room for a class which require a particular feature.



Room Features - College of Arts and Sciences (CAS)

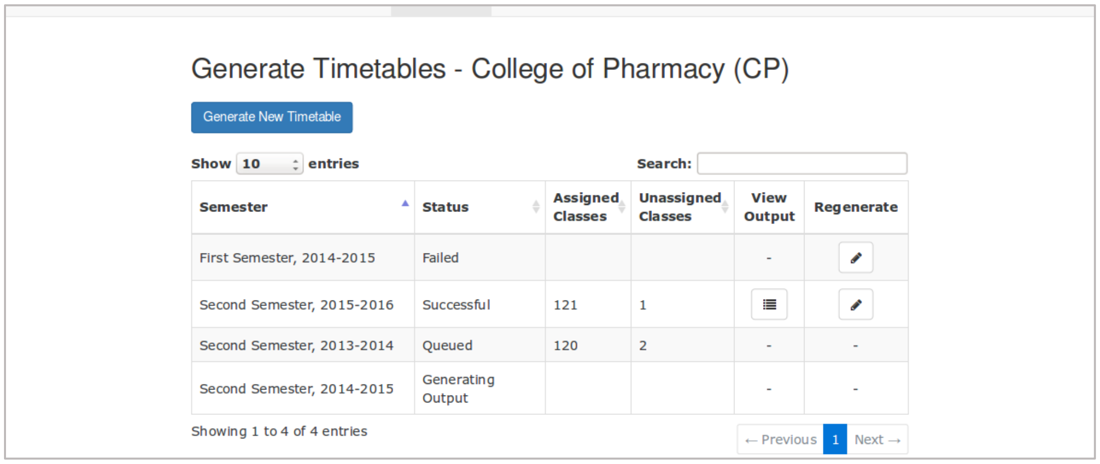
Room Feature	Edit	Delete
Chemistry Laboratory		
Computer Laboratory		
Pharmacy Laboratory		

Showing 1 to 3 of 3 entries

Figure 29: Room Feature List page

4. Timetable Generation

Once the departments have finalized the classes to be offered for a semester, the college admin can proceed to the Timetable Generation page by clicking the Generate Timetables tab under the Timetables dropdown on the navigation bar. As seen in Figure 30, the page lists the timetables created/being created.



Generate Timetables - College of Pharmacy (CP)

Generate New Timetable

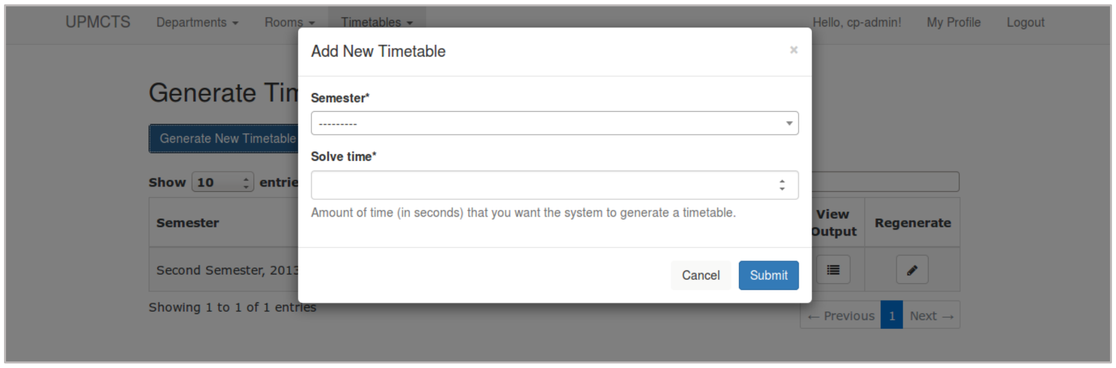
Show 10 entries Search:

Semester	Status	Assigned Classes	Unassigned Classes	View Output	Regenerate
First Semester, 2014-2015	Failed			-	✎
Second Semester, 2015-2016	Successful	121	1	☰	✎
Second Semester, 2013-2014	Queued	120	2	-	-
Second Semester, 2014-2015	Generating Output			-	-

Showing 1 to 4 of 4 entries ← Previous 1 Next →

Figure 30: Timetable Generation page

To create a new timetable for a semester, the college admin must click the Generate New Timetable which will open a popup (See Figure 31) where the college admin can select a semester to generate a timetable.



Add New Timetable ×

Semester*

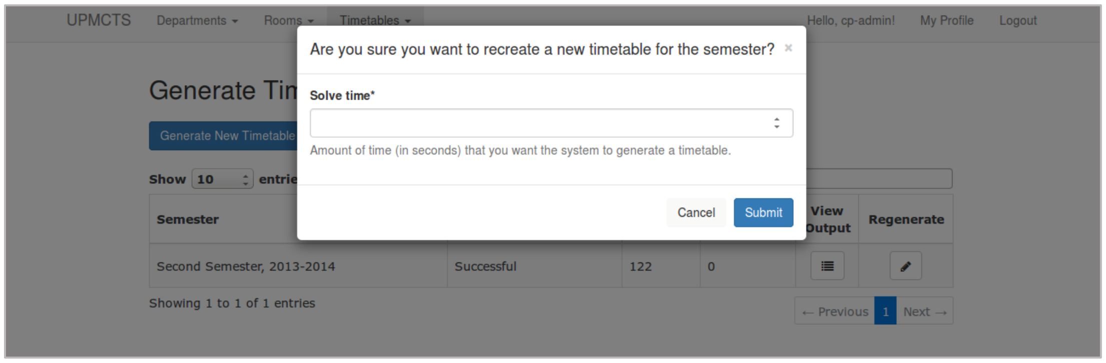
Solve time*

Amount of time (in seconds) that you want the system to generate a timetable.

Cancel Submit

Figure 31: Timetable Create form

The college admin can a regenerate college timetable if its status is successful or failed as shown in Figure 32. Clicking the View Output button of a successful timetable will take the user to its corresponding Timetable Output page.



Generate Tim

Are you sure you want to recreate a new timetable for the semester?

Solve time*

Amount of time (in seconds) that you want the system to generate a timetable.

Show 10 entries

Semester				View Output	Regenerate
Second Semester, 2013-2014	Successful	122	0		

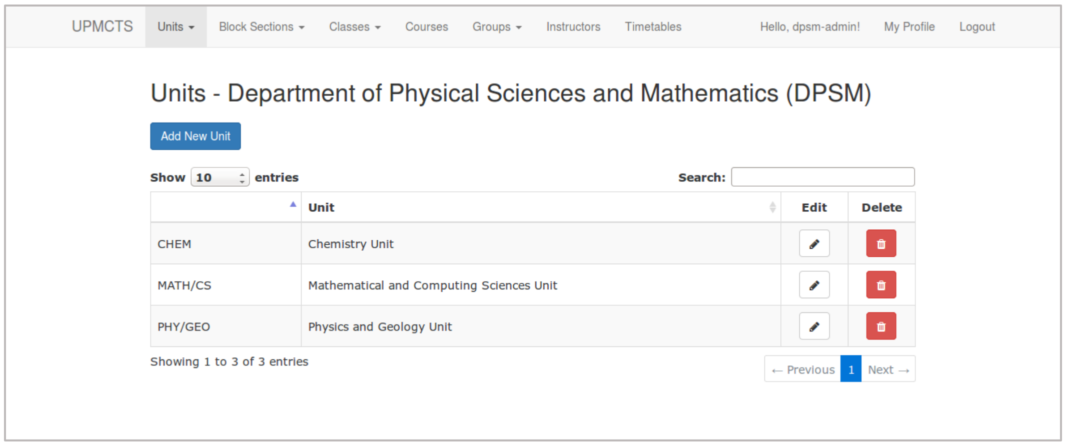
Showing 1 to 1 of 1 entries

Figure 32: Timetable Recreate form

D. Department Chairman

1. Units

Each department can be further subdivided into small units. The department chair can do this by clicking the Units tab under Units dropdown in the navigation bar. As seen in Figure 33, a page similar to the College List page and Department List page is available for viewing, creating, editing, and deleting Units. If the department is small enough, the department can opt to not create any units since the department chair can do all the functionalities that unit heads have.



Units - Department of Physical Sciences and Mathematics (DPSM)

Show 10 entries

Search:

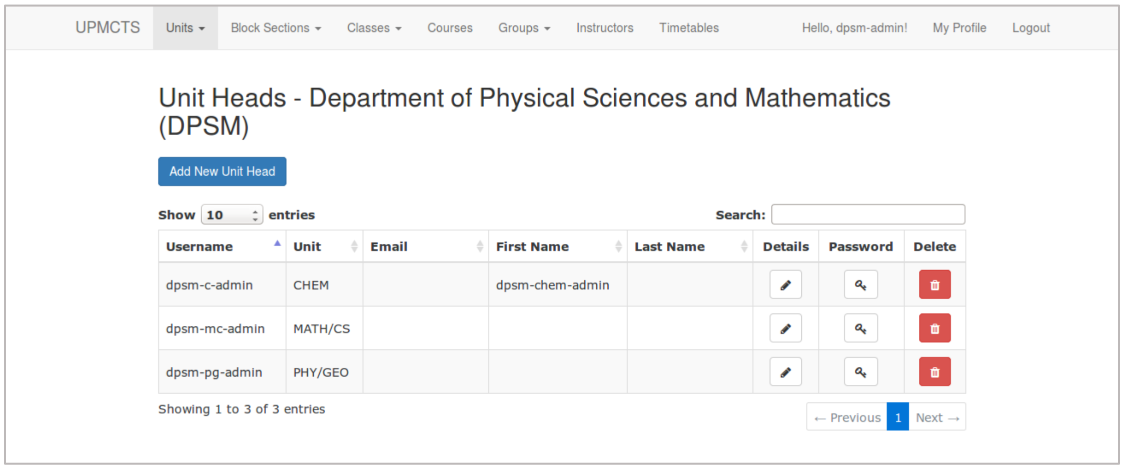
	Unit	Edit	Delete
CHEM	Chemistry Unit		
MATH/CS	Mathematical and Computing Sciences Unit		
PHY/GEO	Physics and Geology Unit		

Showing 1 to 3 of 3 entries

Figure 33: Unit List page

2. Unit Heads

The department chair can create unit head accounts by clicking the Unit Heads tab under Units dropdown in the navigation bar. As seen in Figure 34, the page has similar features to the College Admins page and Department Chairs page. The only difference is that unit head accounts are created instead of college admins or department chairs



Unit Heads - Department of Physical Sciences and Mathematics (DPSM)

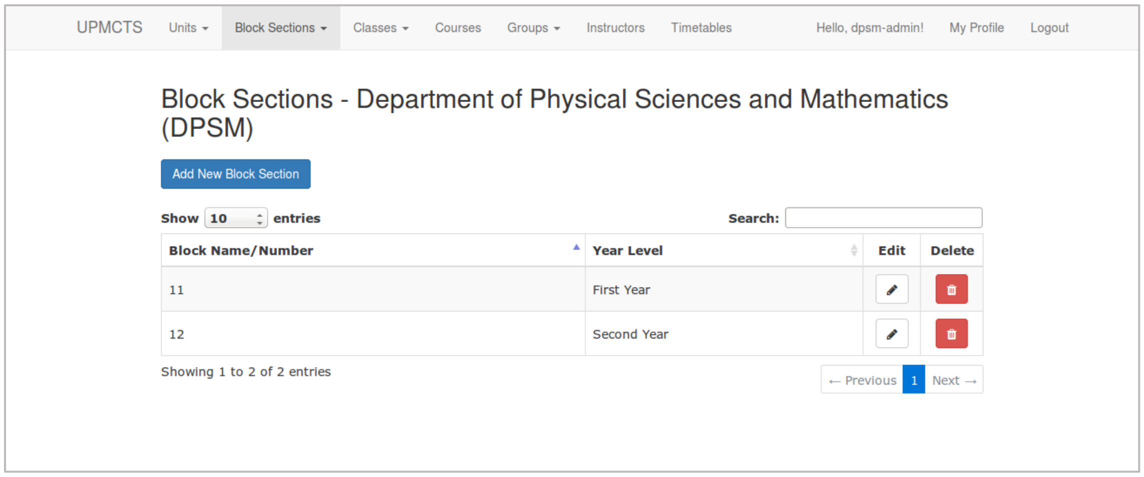
Username	Unit	Email	First Name	Last Name	Details	Password	Delete
dpsm-c-admin	CHEM		dpsm-chem-admin				
dpsm-mc-admin	MATH/CS						
dpsm-pg-admin	PHY/GEO						

Showing 1 to 3 of 3 entries

Figure 34: Unit Head List page

3. Block Sections

The department chair is also responsible for managing block sections (using the page in Figure 35) and their classes for a particular semester (See Figure 36).



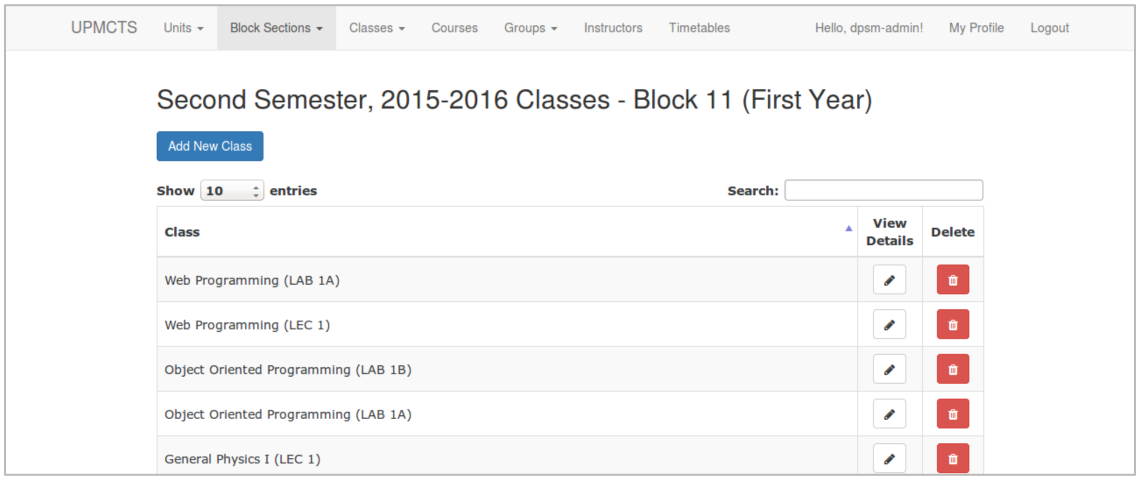
Block Sections - Department of Physical Sciences and Mathematics (DPSM)

Show 10 entries Search:

Block Name/Number	Year Level	Edit	Delete
11	First Year		
12	Second Year		

Showing 1 to 2 of 2 entries

Figure 35: Block Section List page



Second Semester, 2015-2016 Classes - Block 11 (First Year)

Show 10 entries Search:

Class	View Details	Delete
Web Programming (LAB 1A)		
Web Programming (LEC 1)		
Object Oriented Programming (LAB 1B)		
Object Oriented Programming (LAB 1A)		
General Physics I (LEC 1)		

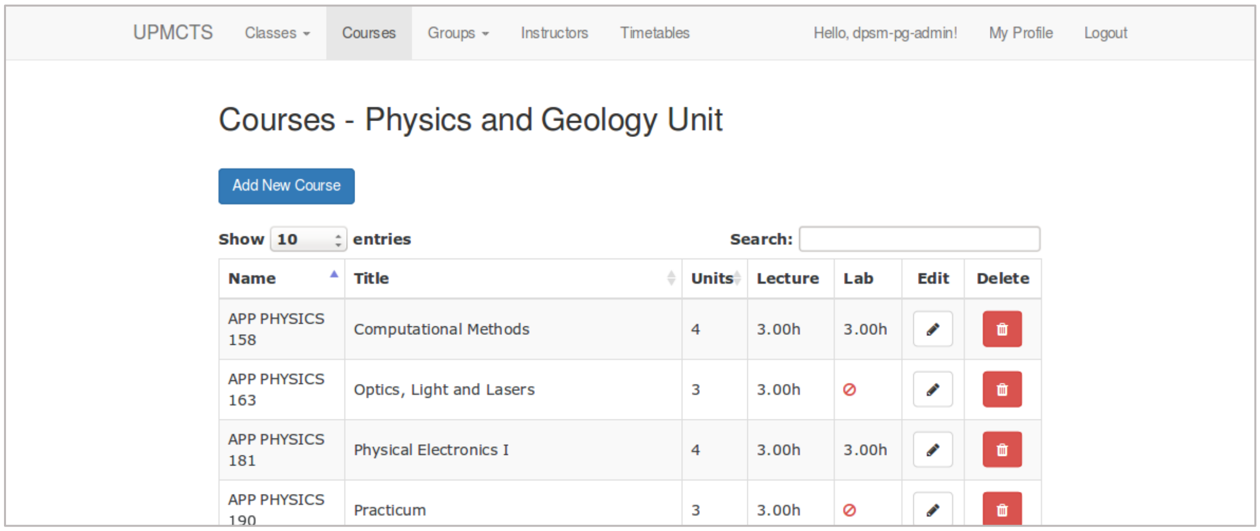
Figure 36: Block Section Class List page

E. Unit Head

Unit Heads are responsible for managing the courses, classes, and instructors. Since the department chair have the final say to the classes for every semester, he/she also has the functionalities available to unit heads and can override any data provided by them.

1. Courses

Unit Heads can add courses by clicking the Courses Tab on the navigation bar. In the Course List page shown in Figure 37, the unit head can view, create, edit, and delete courses offered by the unit. When adding courses, the following details must be provided: course name, course name abbreviation, the course title, number of units, lecture duration, and features needed by the lecture. If the unit admin specified that the course has lab classes, he/she needs to provide the duration of the laboratory class and its needed features.



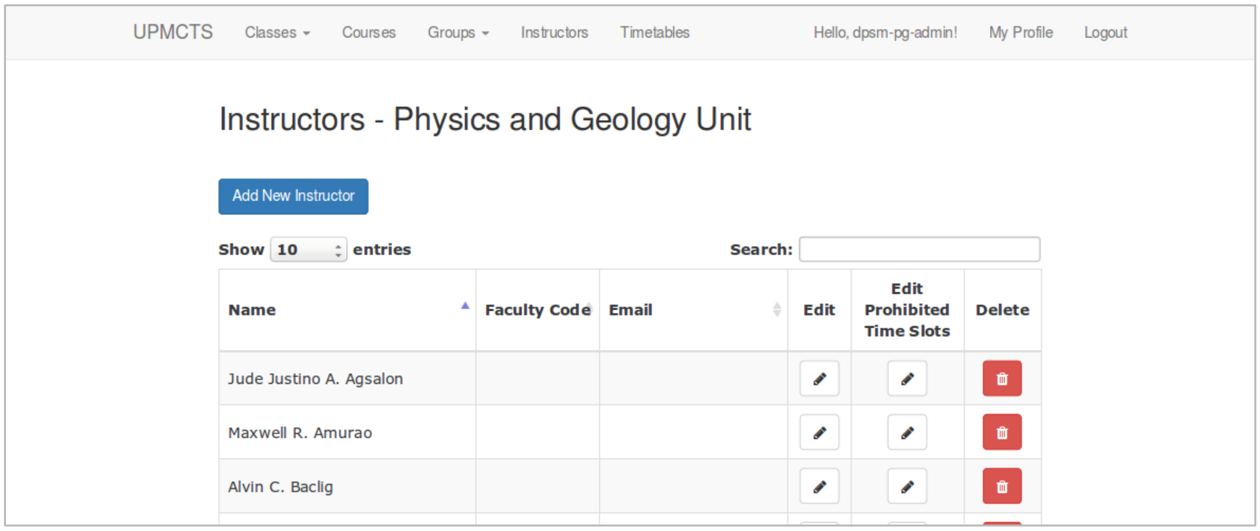
Courses - Physics and Geology Unit

Name	Title	Units	Lecture	Lab	Edit	Delete
APP PHYSICS 158	Computational Methods	4	3.00h	3.00h		
APP PHYSICS 163	Optics, Light and Lasers	3	3.00h	⊘		
APP PHYSICS 181	Physical Electronics I	4	3.00h	3.00h		
APP PHYSICS 190	Practicum	3	3.00h	⊘		

Figure 37: Courses List page

2. Instructors

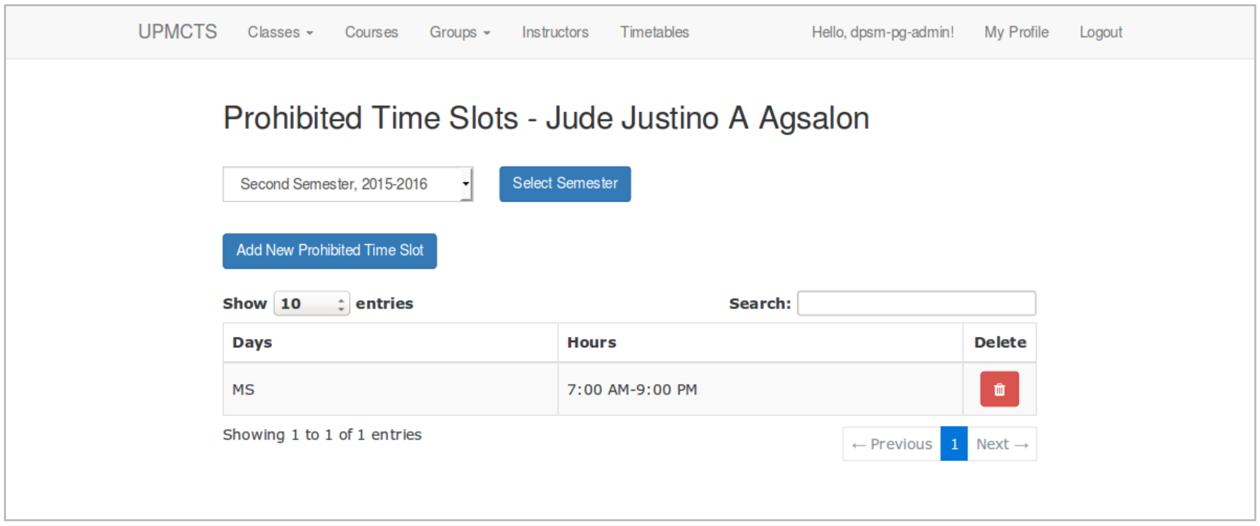
Included in the responsibilities of unit heads is the management of instructors (See Figure 38). The unit admin can set the times where the instructors are not available as shown in Figure 39. These prohibited time slots will be considered when assigning instructors to classes.



Instructors - Physics and Geology Unit

Name	Faculty Code	Email	Edit	Edit Prohibited Time Slots	Delete
Jude Justino A. Agsalon					
Maxwell R. Amurao					
Alvin C. Baclig					

Figure 38: Instructor List page



Prohibited Time Slots - Jude Justino A Agsalon

Days	Hours	Delete
MS	7:00 AM-9:00 PM	

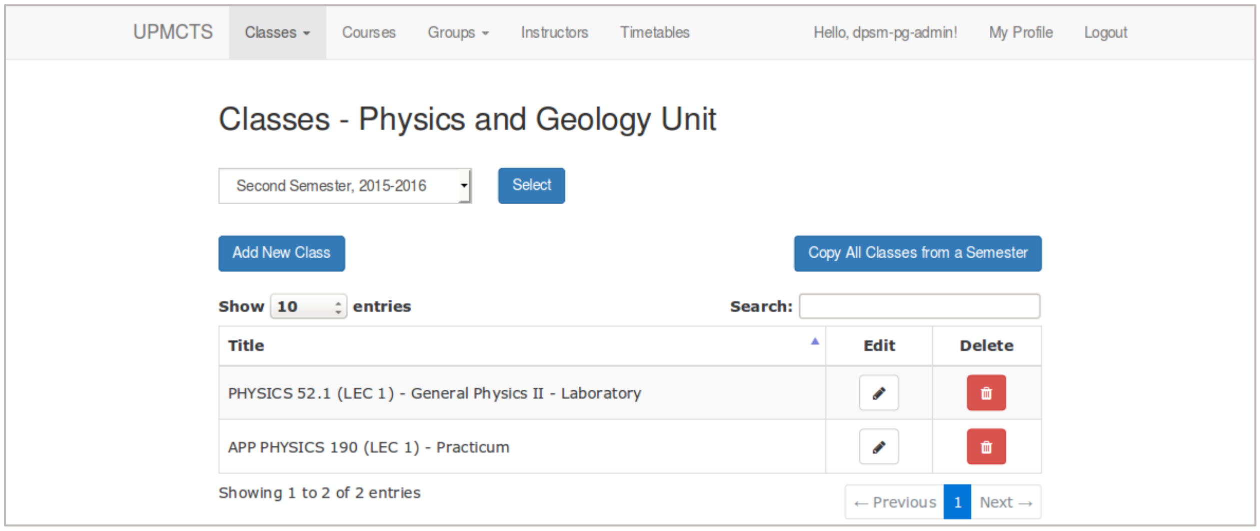
Showing 1 to 1 of 1 entries

Figure 39: Prohibited Time Slots page

3. Classes

Unit heads can manage classes by clicking the Classes tab under the Classes Dropdown on the navigation bar (See Figure 40). When creating classes, the unit head must choose the course to be offered, and if the course has lab classes, the unit admin must specify the number of lab classes that the lecture class should have.

If the unit head wishes to copy all classes from a particular semester so save time, he/she can click the Copy All Classes from a Semester button. Clicking this button will delete the classes on the current semester and replace it with the classes from the selected semester. The assigned slots, assigned instructors, preferred rooms, and preferred time slots will be copied together with the classes. Note that the custom constraints made by the users will not be copied so the unit head will still need to recreate it.



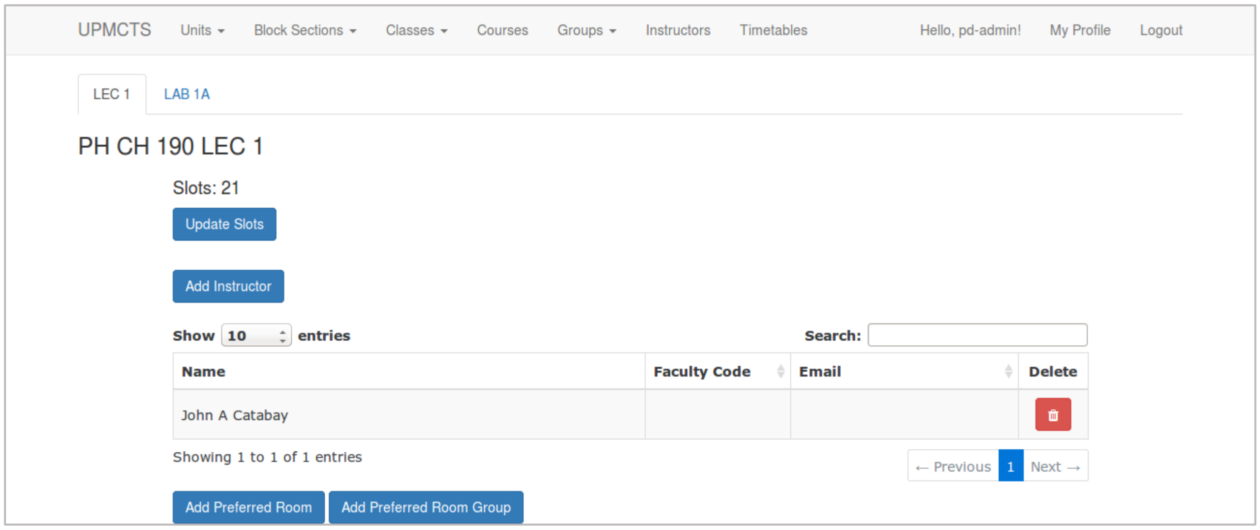
Classes - Physics and Geology Unit

Title	Edit	Delete
PHYSICS 52.1 (LEC 1) - General Physics II - Laboratory		
APP PHYSICS 190 (LEC 1) - Practicum		

Showing 1 to 2 of 2 entries

Figure 40: Class List page

Clicking the Edit button of a class in the Class List page will take the user to the Class Details page (See Figure 41) where he/she can update the following details of the lecture class and its corresponding lab classes: slots offered, assigned instructors (See Figure 42), preferred rooms (See Figure 43), and preferred time slots (See Figure 44).



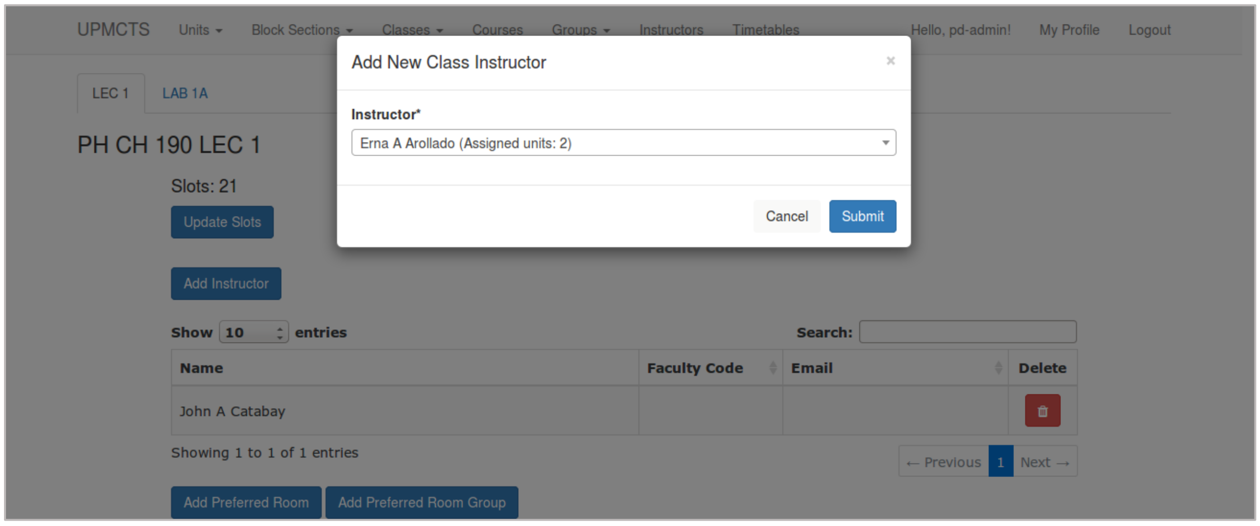
PH CH 190 LEC 1

Slots: 21

Name	Faculty Code	Email	Delete
John A Catabay			🗑

Showing 1 to 1 of 1 entries

Figure 41: Class Details page



PH CH 190 LEC 1

Slots: 21

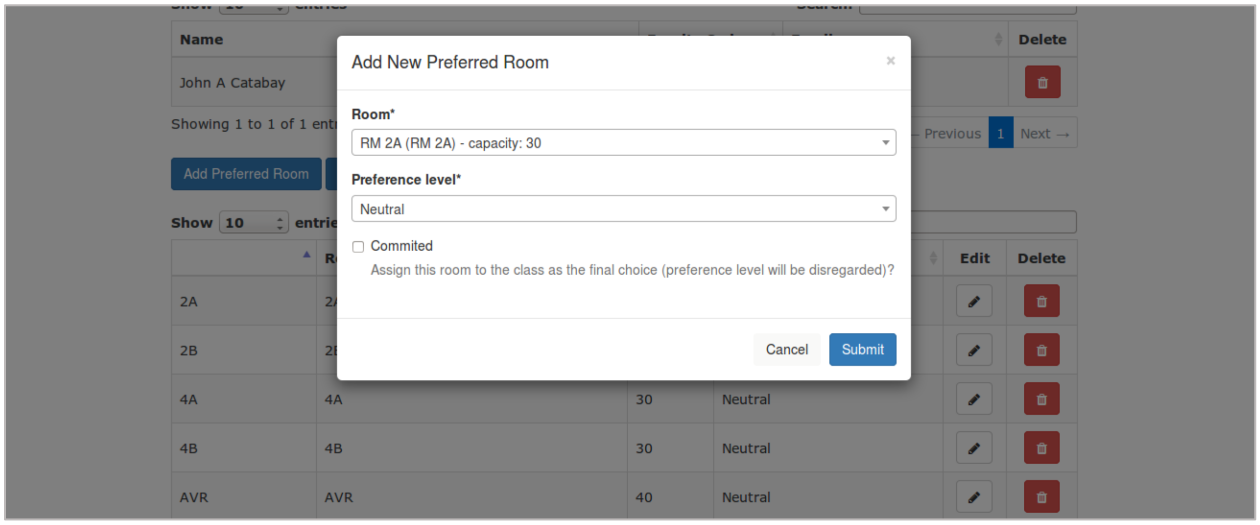
Add New Class Instructor

Instructor*: Erna A Arollado (Assigned units: 2)

Name	Faculty Code	Email	Delete
John A Catabay			🗑

Showing 1 to 1 of 1 entries

Figure 42: Class Instructor Create form


Add New Preferred Room

Room*

RM 2A (RM 2A) - capacity: 30

Preference level*

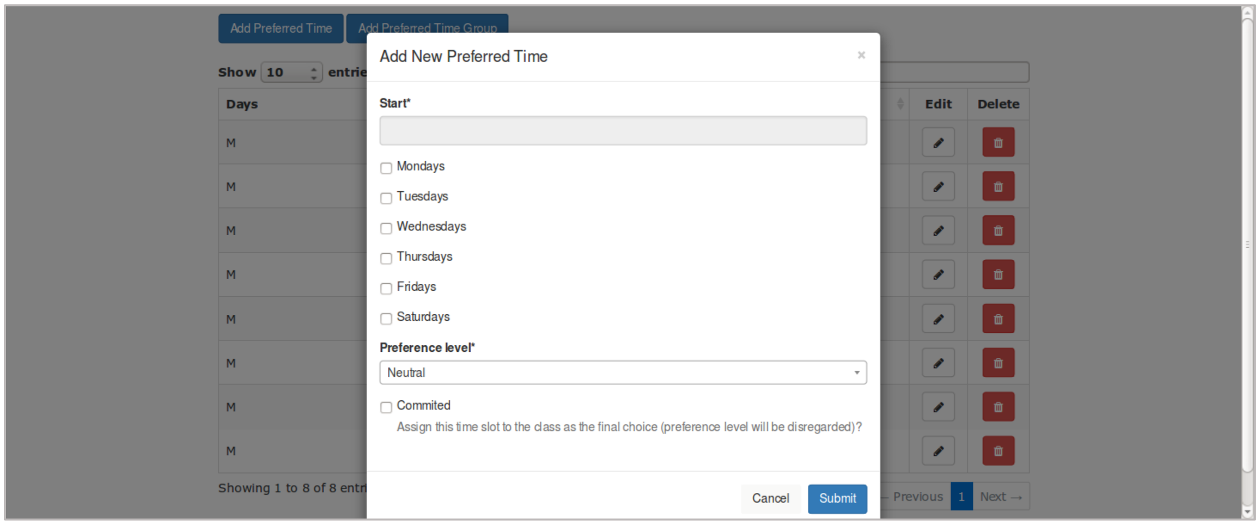
Neutral

Commited

Assign this room to the class as the final choice (preference level will be disregarded)?

Name				Edit	Delete
John A Catabay					
2A	2A				
2B	2B				
4A	4A	30	Neutral		
4B	4B	30	Neutral		
AVR	AVR	40	Neutral		

Figure 43: Class Preferred Room Create form


Add New Preferred Time

Start*

Mondays

Tuesdays

Wednesdays

Thursdays

Fridays

Saturdays

Preference level*

Neutral

Commited

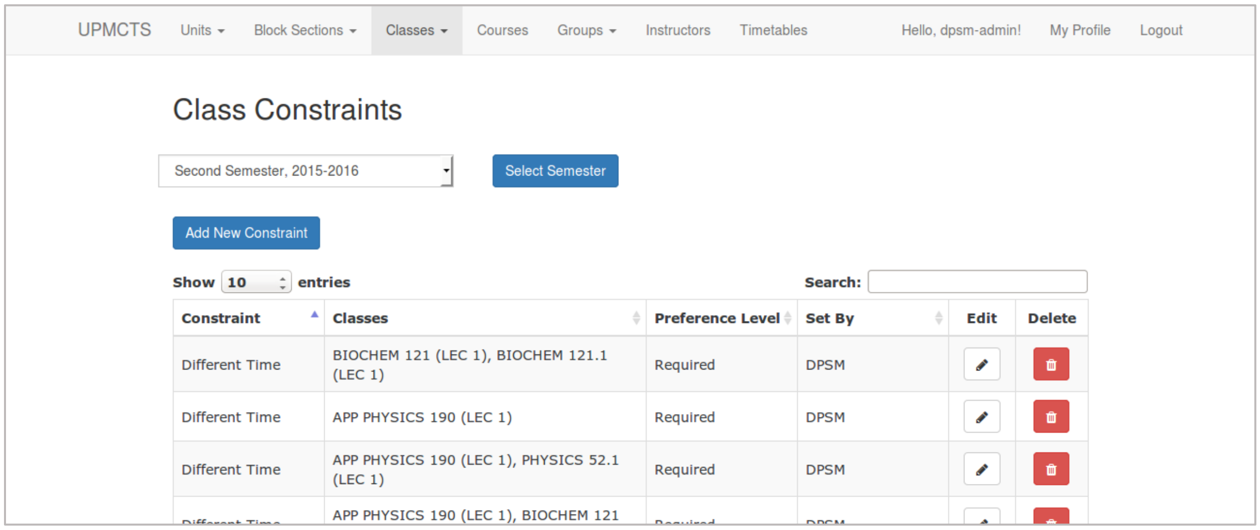
Assign this time slot to the class as the final choice (preference level will be disregarded)?

Showing 1 to 8 of 8 entries

Figure 44: Class Preferred Time Slot Create form

4. Class Group Constraints

For more customizations, the unit head can assign several class constraints to groups of classes. Figure 45 shows the page for managing the class constraints for a particular semester.



Class Constraints

Second Semester, 2015-2016

Select Semester

Add New Constraint

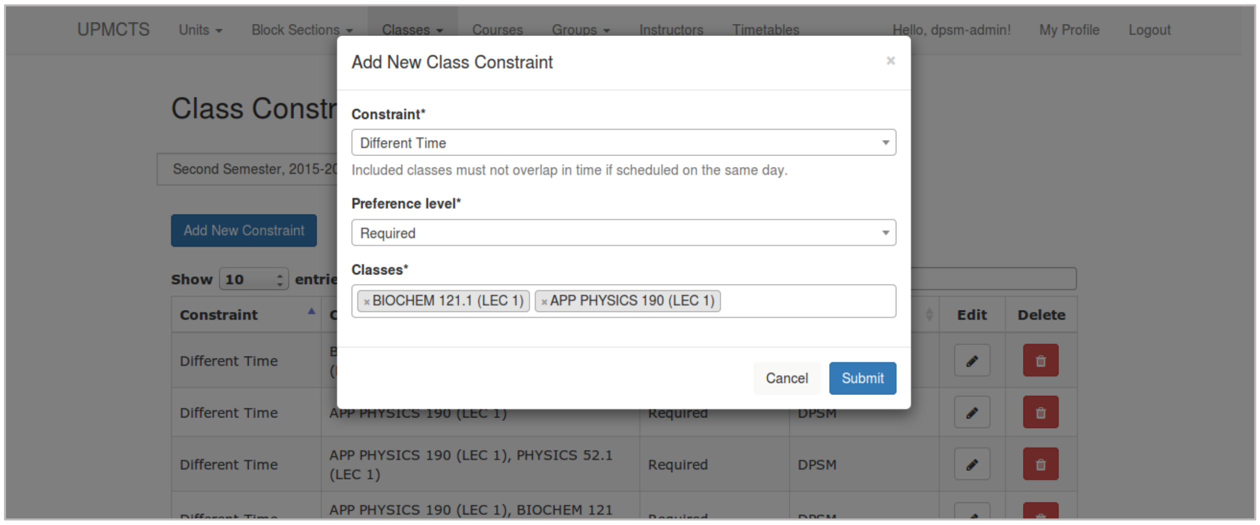
Show 10 entries

Search:

Constraint	Classes	Preference Level	Set By	Edit	Delete
Different Time	BIOCHEM 121 (LEC 1), BIOCHEM 121.1 (LEC 1)	Required	DPSM		
Different Time	APP PHYSICS 190 (LEC 1)	Required	DPSM		
Different Time	APP PHYSICS 190 (LEC 1), PHYSICS 52.1 (LEC 1)	Required	DPSM		
	APP PHYSICS 190 (LEC 1), BIOCHEM 121				

Figure 45: Class Group Constraint List page

Figure 46 show the form for creating a class group constraint where the unit head must specify the following: the type of constraint, the preference level for the constraint, and the classes that will be constrained.



Add New Class Constraint

Constraint*: Different Time

Included classes must not overlap in time if scheduled on the same day.

Preference level*: Required

Classes*: BIOCHEM 121.1 (LEC 1), APP PHYSICS 190 (LEC 1)

Figure 46: Class Group Constraint Create form

VI. Discussion

The UP Manila Course Timetabling System is an application developed to make the timetabling process at the University of the Philippines Manila faster and better. It was developed as a Web application to make it easily accessible to the intended users. Also, having a single application that interacts with all the users makes it easier to consolidate and process all the data being provided by the users.

Python was used to create the Web application. Django, a high level Python Web framework, was used to greatly improve the development process by providing libraries which vastly reduce the amount of code needed to be written. The Web application uses Bootstrap as the front-end framework. To improve the interaction between the Web application and the users, asynchronous JavaScript were used for most of the form interaction between the user and the system.

The system uses Muller's constraint solver, an open-source Java application, for the generation of the timetabling solutions [20]. The generation of the timetable takes several steps. First, the input data is fetched from the database. Second, the data from the database is parsed and converted to XML. Third, the system feeds the resulting XML file to the constraint solver which then outputs another XML which contains the generated timetable. Lastly, the XML is parsed back by the system and the resulting data is saved into the database.

A complete timetable is assessed based on the minimization of the overall solution value which is the result of adding the corresponding values for violated soft constraints. Table 31 shows the values for the preference types available to the users of the system. Table 32 shows the corresponding weights of some soft constraints. To illustrate the minimization of the overall solution value, consider a room which is strongly preferred for a class. If the room is assigned to the class, the product of the preference value (which is -2) and the weight for the room preference (which is 1) will be added to the overall solution value. In this case, -2 will be added which will minimize the overall solution value.

Preference	Value
Strongly preferred	-2
preferred	-1
Neutral	0
Discouraged	1
Strongly discouraged	1

Table 31: Preference values

Constraint	Weight
Time Preferences	0.3
Room Preferences	1.0
Useless slots	0.1
Too big room	0.1

Table 32: Weight of some soft constraint being used by the timetabling solver

The system was tested using the second semester, 2013-2014 class schedule data from the College of Pharmacy in which a total of 122 classes were considered. By setting the maximal solver time for the constraint solver to just 10 seconds, the system was able to consistently generate a complete timetable where all classes are assigned. On one of the results using a 10 second solver timeout, the best solution with an overall solution value of 9.40 was found after 9.6 seconds in a total of 221 iterations. On the second test, the timeout was set to 5 minutes. The best solution has an overall solution value of 4.80 after 1568882 iterations in 4.21 minutes.

VII. Conclusion

The UP Manila Course Timetabling System is a Web application that generates optimal college timetables based on the constraints provided by the users. It was developed in the hope of replacing the manual generation of college timetables at the University of the Philippines Manila. The system aims to significantly reduce the amount of time that is needed to come up with a college timetable. Moreover, it aims to provide timetables that are tailored based on the constraints provided by the users.

The system consists of the following types of user: university/system admin, college admin, department chair, and unit head. This division of roles is based on the current setup in the University of the Philippines Manila for the creation of class. Having a hierarchy of roles makes the creation of timetables more organized. The system also provides a user friendly interface for users to manage the data to be used in the generation of the timetables. Tools are provided to make the customization of classes based on some preferences and constraints easy for users.

Using the class data from the College of Pharmacy for the second semester of 2013-2014, the system was able to consistently find complete timetables in just under 10 seconds. Increasing the time for the generation of the timetable resulted in a better overall solution value. Of course, the results will vary based on the strictness or looseness of the constraints set to the classes. There may be times where a complete solution cannot be found even after a significant amount of time. The strength of the system is that even though a complete solution cannot be found on some cases, it will still return a timetable where some classes are not assigned to a room or time slot. With this, users will be able to easily find the classes that are not assigned and change their corresponding constraints in the hope of finding a complete solution to the timetable.

The system greatly speeds up the generation of college timetables compared to the manual generation being used in UP Manila. It provides an easy way of creating classes for a particular semester, assigning constraints to a group of classes, and viewing the timetables generated.

VIII. Recommendation

There are still a lot of areas that can be improved on the system. First, the user interface can be improved in a lot of ways. In assigning preferred time slots to classes, having a calendar interface would make the process easier to visualize. The addition of a notification system and communication system between users will make the system more interactive.

Second, there are features that can be added to greatly improve the timetables being generated. Giving college admins the ability to commit all classes after generating a timetable and allowing unit heads to uncommit classes that they want to reschedule further will give them more control in the generation of schedules for timetables. The addition of a map for rooms can be used to determine the distance between classrooms. This information can be used to consider the travel time of instructors and students when scheduling back to back classes. Showing a list of rooms which are not being utilized on certain time slots or not being assigned at all will give admins a more control in the overall utilization of rooms by assigning those rooms more frequently. The addition of accounts for instructors where they can provide their preferred schedules to be considered in creating timetables will greatly improve their satisfaction, and will give them a way to voice their input in the generation of timetables.

Lastly, the algorithm used to generate timetables can still be improved. Currently, the system uses a hybrid two-stage optimization algorithm which combines iteration forward search algorithm, hill climbing algorithm, great deluge algorithm, and simulated annealing to come up with an optimal solution using a linear search [20]. Fortunately, there is a lot of interest in creating better algorithms for timetabling. The International Timetabling Competition was created with an overall aim of developing this interest and have a better understanding between researchers by allowing emerging techniques to be trialed and tested on real world models of timetabling problems [33].

IX. Bibliography

[1] "timetable [Def. 1.1]," Oxford Dictionaries, [Online]. Available: http://www.oxforddictionaries.com/us/definition/american_english/timetable. [Accessed January 2015].

[2] J. C. Renée, Science Unshackled, Johns Hopkins University Press, 2014.

[3] R. Lewis, "A Survey of Metaheuristic-based Techniques fo University Timetabling Problems," *OR Spectrum,* vol. 30, no. 1, pp. 167-190, 2008.

[4] Z. Lu and J.-K. Hao, "Adaptive Tabu Search for Course Timetabling," *European Journal of Operational Research,* vol. 200, no. 1, pp. 235-244, 2010.

[5] R.-M. Chen and H.-F. Shih, "Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search," *Algorithms,* vol. 6, pp. 227-244, 2013.

[6] "On the Complexity of Timetable and Multi-Commodity Flow Problems," *IEEE Symposium on Foundations of Computer Science,* p. 184–193, 1975.

[7] "heuristic [Def. 1.1]," Oxford Dictionaries, [Online]. Available: http://www.merriam-webster.com/dictionary/heuristic. [Accessed January 2015].

[8] "Metaheuristics Network," [Online]. Available: http://www.metaheuristics.org/. [Accessed January 2015].

[9] K. Murray, T. Muller and H. Rudova, "Modeling and Solution of a Complex University Course Timetabling Problem," in *Practice and Theory of Automated Timetabling VI*, Springer Berlin Heidelberg, 2007, pp. 189-209.

[10] M. S. Kohshori and M. S. Abadeh, "Hybrid Genetic Algorithms for University Course Timetabling," *International Journal of Computer Science,* vol. 9, no. 2, pp. 446-455, 2012.

[11] Z. Beheshti and S. M. H. Shamsuddin, "A Review of Population-based Meta-Heuristic Algorithms," *International Journal of Advances in Soft Computing & Its ApplicAtions,* vol. 5, no. 1, 2013.

[12] B. Sigl, M. Golub and V. Mornar, "Solving Timetable Scheduling Problem by Using Genetic Algorithms," *Information Technology Interfaces,* pp. 519-524, 2003.

[13] S. Lukas, A. Aribowo and M. Muchri, "Solving Timetable Problem by Genetic Algorithm and Heuristic Search Case Study: Universitas Pelita Harapan Timetable," *Applications of Digital Information and Web Technologies,* pp. 629-633, 2009.

[14] E. F. Q. Nunez, A Hybrid Genetic Algorithm for the Student- Aware University Course Timetabling Problem, 2010.

[15] A. Cerdeira-Pena, L. Carpente, A. Farina and D. Seco, "New Approaches for the School Timetabling Problem," *Artificial Intelligence,* pp. 261-267, 2008.

[16] S. Massoodian and A. Esteki, "A Hybrid Genetic Algorithm for Curriculum Based Course Timetabling," *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling,* pp. 1-11, 2008.

[17] P. A. Sonawane and L. Ragha, "Hybrid Genetic Algorithm and TABU Search Algorithm to Solve Class Time Table Scheduling Problem," *International Journal of Research Studies in Computer Science and Engineering,* vol. 1, no. 4, pp. 19-26, 2014.

[18] E. Montero, M.-C. Riff and L. Altamirano, "A PSO algorithm to solve a Real Course+Exam Timetabling Problem(," in *International conference on swarm intelligence*, 2011.

[19] Q. Bai, "Analysis of Particle Swarm Optimization Algorithm," *Computer and Information Science,* vol. 3, pp. 180-184, 2010.

[20] T. Muller, Constraint-based Timetabling, Prague, 2005.

[21] P. Kenekayoro, "Comparison of simulated annealing and hill climbing in the course timetabling problem," *African Journal of Mathematics and Computer Science Research,* vol. 5, no. 11, pp. 176-178, 2012.

[22] A. Gunawan and K. M. Ng, "Solving the Teacher Assignment Problem by Two Metaheuristics," *International Journal of Information and Management Sciences,* vol. 22, pp. 73-86, 2011.

[23] A. Gunawan, N. Kien Ming and P. Kim Leng, "A Hybrid Algorithm for the University Course Timetabling Problem," in *7th International Conference on the Practice and Theory of Automated Timetabling*, Montreal, 2008.

[24] T. Muller, "ITC2007 Solver Description: A Hybrid Approach," *Annals of Operations Research,* vol. 172, no. 1, pp. 429-446, 2009.

[25] E. K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research,* vol. 140, no. 2, pp. 266-280, 2002.

[26] T. B. Cooper and J. H. Kingston, "The complexity of timetable construction problems," in *Practice and Theory of Automated Timetabling*, 1996, pp. 281-295.

[27] A. Schaerf, "A Survey of Automated Timetabling," *Artificial Intelligence Review,* vol. 13, no. 2, pp. 87-127, 1999.

[28] B. Selman and C. P. Gomes, "Hill-climbing Search," in *Encyclopedia of Cognitive Science*, 2006.

[29] D. Landa-Silva and J. H. Obit, "Great deluge with non-linear decay rate for solving course timetabling problems," in *4th International IEEE Conference on Intelligent Systems*, 2008.

[30] E. S. Sin and N. S. M. Kham, "Hyper heuristic based on great deluge and its variants for exam timetabling problem," *International Journal of Artificial Intelligence & Applications,* vol. 3, no. 1, 2012.

[31] D. Bertsimas and J. Tsitsiklis, "Simulated Annealing," *Statistical Science,* vol. 8, no. 1, pp. 10-15, 1993.

[32] S. N. Jat, Genetic Algorithms for University Course Timetabling Problems, 2012.

[33] "International TImetabling Competition," 10 May 2010. [Online]. Available: http://www.cs.qub.ac.uk/itc2007/. [Accessed 7 May 2016].

[34] A. Nanda, M. . P. Pai and A. Gole, "An Algorithm to Automatically Generate Schedule for School Lectures Using a Heuristic Approach," *International Journal of Machine Learning and Computing,* vol. 2, no. 4, pp. 492-495, 2012.

X. Appendix

Source Code

1. config/settings/common.py

```
# -*- coding: utf-8 -*-
"""
Django settings for upmcts project.

For more information on this file, see
https://docs.djangoproject.com/en/dev/topics/se
ttings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/dev/ref/settin
gs/
"""
from __future__ import absolute_import,
unicode_literals

import sys
import environ

ROOT_DIR = environ.Path(__file__) - 3  #
(/a/b/myfile.py - 3 = /)
APPS_DIR = ROOT_DIR.path('upmcts')

env = environ.Env()

reload(sys)
sys.setdefaultencoding('utf-8')

# APP CONFIGURATION
# ------------------------------------------------------
----------------------
DJANGO_APPS = (
    # Default Django apps:
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Useful template tags:
    # 'django.contrib.humanize',

    # Admin
    'django.contrib.admin',
)
THIRD_PARTY_APPS = (
    'crispy_forms', # Form layouts
    'allauth',  # registration
    'allauth.account',  # registration
    'allauth.socialaccount',  # registration
    'djangojs',  # Tables
    'eztables',  # Tables
    'djangobower',
    'bootstrap3',
    'django_select2',
    'django_cleanup',
    'django_tables2',
    'django_tables2_reports',
    'easy_pdf',

)

# Apps specific for this project go here.
LOCAL_APPS = (
    'upmcts.colleges',
    'upmcts.departments',
    'upmcts.instructors',
    'upmcts.units',
    'upmcts.university',
    'upmcts.users',  # custom users app
    'upmcts.utils',
)

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#installed-apps
INSTALLED_APPS = DJANGO_APPS +
THIRD_PARTY_APPS + LOCAL_APPS

# MIDDLEWARE CONFIGURATION
# ------------------------------------------------------
----------------------
MIDDLEWARE_CLASSES = (
    # Make sure
djangosecure.middleware.SecurityMiddleware
is listed first

'django.contrib.sessions.middleware.SessionMi
ddleware',

'django.middleware.common.CommonMiddle
ware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.Authentication
Middleware',

'django.contrib.messages.middleware.Message
Middleware',

'django.middleware.clickjacking.XFrameOptio
nsMiddleware',
)
```

```
# MIGRATIONS CONFIGURATION
# ------------------------------------------------------
------------------------
MIGRATION_MODULES = {
    'sites': 'upmcts.contrib.sites.migrations'
}

# DEBUG
# ------------------------------------------------------
------------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#debug
DEBUG = env.bool("DJANGO_DEBUG",
False)

# FIXTURE CONFIGURATION
# ------------------------------------------------------
------------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#std:setting-FIXTURE_DIRS
FIXTURE_DIRS = (
    str(APPS_DIR.path('fixtures')),
)

# EMAIL CONFIGURATION
# ------------------------------------------------------
------------------------
EMAIL_BACKEND =
env('DJANGO_EMAIL_BACKEND',
default='django.core.mail.backends.smtp.Email
Backend')

# MANAGER CONFIGURATION
# ------------------------------------------------------
------------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#admins
ADMINS = (
    ("""John Renz P. Ladia""",
'renzladia@gmail.com'),
)

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#managers
MANAGERS = ADMINS

# DATABASE CONFIGURATION
# ------------------------------------------------------
------------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#databases
DATABASES = {
    # Raises ImproperlyConfigured exception if
DATABASE_URL not in os.environ
    'default': env.db("DATABASE_URL",
default="postgres:///upmcts"),
}
DATABASES['default']['ATOMIC_REQUEST
S'] = True


# GENERAL CONFIGURATION
# ------------------------------------------------------
------------------------
# Local time zone for this installation. Choices
can be found here:
#
http://en.wikipedia.org/wiki/List_of_tz_zones_
by_name
# although not all choices may be available on
all operating systems.
# In a Windows environment this must be set to
your system time zone.
TIME_ZONE = 'UTC'

TIME_INPUT_FORMATS = [
    '%I:%M %p',     # '02:30 pm'
    '%-I:%M %p',    # '2:30 pm'
    '%H:%M:%S',     # '14:30:59'
    '%H:%M:%S.%f',  # '14:30:59.000200'
    '%H:%M',        # '14:30'
]

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#language-code
LANGUAGE_CODE = 'en-us'

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#site-id
SITE_ID = 1

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#use-i18n
USE_I18N = True

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#use-l10n
USE_L10N = True

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#use-tz
USE_TZ = True

# TEMPLATE CONFIGURATION
# ------------------------------------------------------
-----------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#templates
TEMPLATES = [
   {
      # See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#std:setting-TEMPLATES-BACKEND
      'BACKEND':
'django.template.backends.django.DjangoTemp
lates',
      # See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#template-dirs
      'DIRS': [
         str(APPS_DIR.path('templates')),
      ],
      'OPTIONS': {
         # See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#template-debug
         'debug': DEBUG,
         # See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#template-loaders
         #
https://docs.djangoproject.com/en/dev/ref/temp
lates/api/#loader-types
         'loaders': [

'django.template.loaders.filesystem.Loader',

'django.template.loaders.app_directories.Loade
r',
         ],
         # See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#template-context-processors
         'context_processors': [

'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.template.context_processors.i18n',

'django.template.context_processors.media',

'django.template.context_processors.static',

'django.template.context_processors.tz',

'django.contrib.messages.context_processors.m
essages',
            # Your stuff: custom template context
processors go here

'django.core.context_processors.request',

'django.core.context_processors.static',
         ],
      },
   },
]

# See: http://django-crispy-
forms.readthedocs.org/en/latest/install.html#te
mplate-packs
CRISPY_TEMPLATE_PACK = 'bootstrap3'

# STATIC FILE CONFIGURATION
# ------------------------------------------------------
-----------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#static-root
STATIC_ROOT =
str(ROOT_DIR('staticfiles'))

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#static-url
STATIC_URL = '/static/'

# See:
https://docs.djangoproject.com/en/dev/ref/contr
ib/staticfiles/#std:setting-STATICFILES_DIRS
STATICFILES_DIRS = (
   str(APPS_DIR.path('static')),
)

# See:
https://docs.djangoproject.com/en/dev/ref/contr
ib/staticfiles/#staticfiles-finders
STATICFILES_FINDERS = (

'django.contrib.staticfiles.finders.FileSystemFin
der',

'django.contrib.staticfiles.finders.AppDirectorie
sFinder',
   'djangobower.finders.BowerFinder',
)

# MEDIA CONFIGURATION
# ------------------------------------------------------
-----------------------
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#media-root
MEDIA_ROOT = str(ROOT_DIR('media'))
```

```
# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#media-url
MEDIA_URL = '/media/'

# BOWER CONFIGURATION
# ------------------------------------------------------
----------------------
BOWER_COMPONENTS_ROOT =
str(APPS_DIR.path('static'))

BOWER_INSTALLED_APPS = (
    'jquery#2.2.3',
    'font-awesome#4.6.2',
    'bootstrap#3.3.6',
    'noty#2.3.8',
    'select2#4.0.2',
    'pickadate',
    'moment',
    'jquery-ui-sortable',
)

# URL Configuration
# ------------------------------------------------------
----------------------
ROOT_URLCONF = 'config.urls'

# See:
https://docs.djangoproject.com/en/dev/ref/settin
gs/#wsgi-application
WSGI_APPLICATION =
'config.wsgi.application'

# AUTHENTICATION CONFIGURATION
# ------------------------------------------------------
----------------------
AUTHENTICATION_BACKENDS = (

'django.contrib.auth.backends.ModelBackend',

'allauth.account.auth_backends.Authentication
Backend',
)

# Some really nice defaults
ACCOUNT_AUTHENTICATION_METHOD
= 'username'
ACCOUNT_EMAIL_REQUIRED = False
ACCOUNT_EMAIL_VERIFICATION =
'none'
ACCOUNT_LOGIN_ON_EMAIL_CONFIRM
ATION = False
ACCOUNT_LOGOUT_ON_GET = True
ACCOUNT_SIGNUP_FORM_CLASS =
'upmcts.users.forms.UserSignupForm'
ACCOUNT_LOGIN_ATTEMPTS_LIMITAC
COUNT_LOGIN_ATTEMPTS_LIMIT = 5
ACCOUNT_LOGIN_ATTEMPTS_TIMEOUT
= 300

# Custom user app defaults
# Select the correct user model
AUTH_USER_MODEL = 'users.User'
LOGIN_REDIRECT_URL = 'users:redirect'
LOGIN_URL = 'account_login'

# SLUGLIFIER
AUTOSLUG_SLUGIFY_FUNCTION =
'slugify.slugify'

########## CELERY
# INSTALLED_APPS +=
('upmcts.taskapp.celery.CeleryConfig',)
# # if you are not using the django database
broker (e.g. rabbitmq, redis, memcached), you
can remove the next line.
# INSTALLED_APPS +=
('kombu.transport.django',)
# BROKER_URL =
env("CELERY_BROKER_URL",
default='django://')
########## END CELERY

# Location of root django.contrib.admin URL,
use {% url 'admin:index' %}
ADMIN_URL = r'^admin/'

# django-tables2-reports
EXCEL_SUPPORT = 'xlwt'  # or 'openpyxl' or
'pyexcelerator'


# Timetabling Solver
TIMETABLING_SOLVER_ROOT =
str(ROOT_DIR('timetabling_solver'))
TIMETABLE_OUTPUT_FILE = 'solution.xml'
```

2. config/urls.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.conf import settings
from django.conf.urls import include, url
from django.conf.urls.static import static
from django.contrib import admin
from django.views.generic import
TemplateView
from django.views import defaults as
default_views

from allauth.account import views as
allauthviews

urlpatterns = [
    url(r'^$', allauthviews.login, name="home"),
    url(r'^about/$',
TemplateView.as_view(template_name='pages
/about.html'), name="about"),

    # Django Admin, use {% url 'admin:index'
%}
    url(settings.ADMIN_URL,
include(admin.site.urls)),

    # User management
    url(r'^users/', include("upmcts.users.urls",
namespace="users")),
    url(r'^university/',
include("upmcts.university.urls",
namespace="university")),
    url(r'^colleges/',
include("upmcts.colleges.urls",
namespace="colleges")),
    url(r'^departments/',
include("upmcts.departments.urls",
namespace="departments")),
    url(r'^units/', include("upmcts.units.urls",
namespace="units")),
    url(r'^instructors/',
include("upmcts.instructors.urls",
namespace="instructors")),
    url(r'^accounts/', include('allauth.urls')),
    url(r'^djangojs/', include('djangojs.urls')),
    url(r'^select2/',
include('django_select2.urls')),

    # Your stuff: custom urls includes go here


] + static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)

if settings.DEBUG:
    # This allows the error pages to be debugged
during development, just visit
    # these url in browser to see how these error
pages look like.
    urlpatterns += [
        url(r'^400/$', default_views.bad_request),
        url(r'^403/$',
default_views.permission_denied),
        url(r'^404/$',
default_views.page_not_found),
        url(r'^500/$', default_views.server_error),
    ]
```

3. requirements/base.txt

```
# Bleeding edge Django
django==1.8.5

# Configuration
django-environ==0.4.0
django-secure==1.0.1
whitenoise==2.0.4


# Forms
django-braces==1.8.1
django-crispy-forms==1.5.2
django-floppyforms==1.5.2

# Models
django-model-utils==2.3.1

# Images
Pillow==3.0.0

# For user registration, either via email or
social
# Well-built with regular release cycles!
django-allauth==0.23.0


# Python-PostgreSQL Database Adapter
psycopg2==2.6.1

# Unicode slugification
unicode-slugify==0.1.3
django-autoslug==1.9.3

# Time zones support
pytz==2015.6

# Redis support
django-redis==4.2.0
redis>=2.10.0


celery==3.1.18


# Custom requirements
django-bower==5.0.4
django-eztables==0.3.2
django-select2==5.5.0
django-cleanup==0.4.2
xmltodict==0.10.1
django-bootstrap3==7.0.1

django-tables2==1.2.1
django-tables2-reports==0.0.10

# Spreadsheets support
xlwt==1.0.0

# PDF printing
reportlab==2.7
xhtml2pdf==0.0.6
django-easy-pdf==0.1.0
```

4. requirements/local.txt

```
# Local development dependencies go here
-r base.txt
coverage==4.0.1
Sphinx
django-extensions==1.5.7
Werkzeug==0.10.4
django-test-plus==1.0.9
factory_boy==2.6.0

# django-debug-toolbar that works with Django
1.5+
django-debug-toolbar==1.4

# improved REPL
ipdb==0.8.1

# Required by maildump. Need to pin
dependency to gevent beta to be Python 3-
compatible.
gevent==1.0.2
# Enables better email testing
maildump==0.5.1
```

5. requirements/production.txt

```
# Pro-tip: Try not to put anything here. There
should be no dependency in
#            production that isn't in development.
-r base.txt



# WSGI Handler
# ------------------------------------------------
gevent==1.0.2
gunicorn==19.3.0

# Static and Media Storage
# ------------------------------------------------
boto==2.38.0
django-storages-redux==1.3


# Mailgun Support
# ---------------
django-mailgun==0.7.2

# Raven is the Sentry client
# --------------------------
Raven
```

6. upmcts/colleges/forms.py

```
from django import forms

from crispy_forms.helper import FormHelper
from django_select2.forms import
Select2Widget
from upmcts.departments.models import
Department, DepartmentAdmin
from upmcts.taskapp.celery import
generate_timetable
from upmcts.units.models import
ClassRoomPreference
from upmcts.university.forms import
CollegeAdminUpdateForm
from upmcts.university.models import
Semester
from upmcts.users.forms import
UserCreateForm
from upmcts.users.models import User

from .models import RoomFeature, Room,
Timetable

from django_select2.forms import
Select2MultipleWidget


class DepartmentForm(forms.ModelForm):

    class Meta:
        model = Department
        fields = ('name', 'abbrev', )

    def __init__(self, *args, **kwargs):
        self.college = kwargs.pop('college', None)

        super(DepartmentForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        department = super(
            DepartmentForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.college:
            department.college = self.college

        if commit:
            department.save()

        return department


class
DepartmentAdminCreateForm(UserCreateFor
m):
    department =
forms.ModelChoiceField(queryset=Department
.objects.all())

    class Meta:
        model = User
        fields = (
            'username', 'password1', 'password2',
'department',
            'email', 'first_name', 'last_name', )
        widgets = {
            'department': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.college = kwargs.pop('college', None)
        super(DepartmentAdminCreateForm,
self).__init__(*args, **kwargs)

        self.fields['department'].queryset =
Department.objects.filter(
            college=self.college
        )

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class
DepartmentAdminUpdateForm(CollegeAdmin
UpdateForm):
    class Meta:
        model = DepartmentAdmin
        fields = (
            'username', 'department', 'email',
'first_name', 'last_name',)
        widgets = {
            'department': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.college = kwargs.pop('college', None)
        super(DepartmentAdminUpdateForm,
self).__init__(*args, **kwargs)

        self.fields['department'].queryset =
Department.objects.filter(
            college=self.instance.college
        )

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class RoomFeatureForm(forms.ModelForm):
    class Meta:
        model = RoomFeature
        fields = ('name', )

    def __init__(self, *args, **kwargs):
        super(RoomFeatureForm,
self).__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class RoomForm(forms.ModelForm):

    class Meta:
        model = Room
        fields = ('name', 'abbrev', 'capacity',
'is_available', 'features')
        widgets = {
            'features': Select2MultipleWidget
        }

    def __init__(self, *args, **kwargs):
        self.college = kwargs.pop('college', None)
        super(RoomForm, self).__init__(*args,
**kwargs)
        self.feature_list = []
        is_new = not self.instance or not
self.instance.pk
        if not is_new:
            self.feature_list =
self.instance.features.values_list('id', flat=True)
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        room = super(
            RoomForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        features = self.cleaned_data.get('features',
None)
        feature_ids = features.values_list('id',
flat=True)

        if is_new and self.college:
            room.college = self.college

        if commit:
            room.save()
            updated_features = not
set(self.feature_list) == set(feature_ids)
            if not is_new and updated_features:
```

```
            preferences =
ClassRoomPreference.objects.filter(
                room=room,

).select_related('course_class__course')
            for pref in preferences:
                if (pref.course_class.lecture is
None):
                    class_room_features_id_list =
pref.course_class.course.required_lec_features.
values_list('id', flat=True)
                else:
                    class_room_features_id_list =
pref.course_class.course.required_lab_features.
values_list('id', flat=True)
                if (not
set(class_room_features_id_list) <
set(feature_ids) and
                        not
set(class_room_features_id_list) <
set(feature_ids)):
                    pref.delete()
            self.save_m2m()
        return room


class TimetableForm(forms.ModelForm):

    solve_time = forms.IntegerField(
        min_value=1,
        help_text="Amount of time (in seconds)
that you want the system to generate a
timetable.")

    class Meta:
        model = Timetable
        fields = ('semester',)
        widgets = {
            'semester': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.college = kwargs.pop('college', None)
        super(TimetableForm,
self).__init__(*args, **kwargs)

        if not self.instance or not self.instance.pk:
            semester_qs =
Semester.objects.exclude(

pk__in=self.college.timetables.all().values_list(
                'semester', flat=True))
            self.fields['semester'].queryset =
semester_qs

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        timetable = super(
            TimetableForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        solve_time =
self.cleaned_data.get('solve_time', 60)
        if is_new and self.college:
            timetable.college = self.college

        if commit:
            timetable.save()
            generate_timetable.delay(timetable,
solve_time)

        return timetable


class TimetableTimeoutForm(forms.Form):
    solve_time = forms.IntegerField(
        min_value=1,
        help_text="Amount of time (in seconds)
that you want the system to generate a
timetable.")

    def __init__(self, *args, **kwargs):
        self.college = kwargs.pop('college', None)
        super(TimetableTimeoutForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.disable_csrf = True
        self.helper.template_pack = 'bootstrap3'
```

7. upmcts/colleges/models.py

```
import os

from django.db import models
from django.db.models import Q
from django.db.models.signals import
post_delete
from django.dispatch import receiver
from django.utils.translation import
ugettext_lazy as _

from model_utils import Choices

from upmcts.university.models import
Semester
from upmcts.users.models import User


class RoomManager(models.Manager):
    def xml_dict(self, college):
        rooms = super(RoomManager,
self).get_queryset().filter(college=college).valu
es('id', 'capacity')
        rooms_dict = {
            'room': list(rooms)
        }
        return rooms_dict


class College(models.Model):
    name = models.CharField(
        verbose_name=_(u'College Name'),
        max_length=255,
        unique=True)
    abbrev = models.CharField(
        verbose_name=_(u'College
Abbreviation'),
        max_length=20,
        help_text=_(u'e.g. CAS'),
        unique=True)

    def __unicode__(self):
        return '{} ({})'.format(self.name,
self.abbrev)


class CollegeAdmin(models.Model):
    user = models.OneToOneField(User)
    college = models.ForeignKey(College,
related_name='college_admins')

    def __unicode__(self):
        return '{} ({})'.format(self.user.username,
self.college)


class RoomFeature(models.Model):
    name = models.CharField(
        verbose_name=_(u'Feature Name'),
        max_length=255,
        unique=True)

    def __unicode__(self):
        return self.name


class Room(models.Model):
    college = models.ForeignKey(College,
related_name='rooms', null=True)
    name = models.CharField(
        verbose_name=_(u'Room Name'),
        max_length=255,
        unique=True)
    abbrev = models.CharField(
        verbose_name=_(u'Room Abbreviation'),
        max_length=20,
        help_text=_(u'e.g. RH 114'),
        unique=True)
    capacity =
models.PositiveIntegerField(verbose_name=_(
u'Maximum Room Capacity'))
    is_available = models.BooleanField(
        default=True,
        help_text=_(u'Is the room curently
available for use?'))
    features = models.ManyToManyField(
        RoomFeature,
        verbose_name=_(u'Room Features'),
        related_name='features',
        blank=True)

    objects = RoomManager()

    def __unicode__(self):
        return '{} ({}) - capacity:
{}'.format(self.name, self.abbrev, self.capacity)


def get_input_filename(self, filename):
    path = 'timetables/input/' +
self.college.abbrev + '/'
    new_filename = self.semester.sem + '__' +
self.semester.academic_year + '.xml'
    new_filename = new_filename.replace(" ",
"_").lower()
    return os.path.join(path, new_filename)


def get_output_filename(self, filename):
    path = 'timetables/output/' +
self.college.abbrev + '/'
    new_filename = self.semester.sem + '__' +
self.semester.academic_year + '.xml'
    new_filename = new_filename.replace(" ",
"_").lower()
    return os.path.join(path, new_filename)


class Timetable(models.Model):
    QUEUED = 'Q'
    GENERATING_INPUT = 'I'
    GENERATING_OUTPUT = 'O'
    SUCCESSFUL = 'S'
    FAILED = 'F'
    STATUS_CHOICES = Choices(
        (QUEUED, _(u'Queued')),
        (GENERATING_INPUT, _(u'Generating
Input')),
        (GENERATING_OUTPUT,
_(u'Generating Output')),
        (SUCCESSFUL, _(u'Successful')),
        (FAILED, _(u'Failed')),
    )
    college = models.ForeignKey(College,
related_name='timetables')
    semester = models.ForeignKey(Semester,
related_name='timetables')
    input_created_date = models.DateTimeField(
        _(u'input created date'), blank=True,
null=True, default=None)
    output_created_date =
models.DateTimeField(
        _(u'output created date'), blank=True,
null=True, default=None)
    assigned_classes_count =
models.PositiveIntegerField(
        verbose_name=_(u'assigned classes
count'), blank=True, null=True, default=None)
    unassigned_classes_count =
models.PositiveIntegerField(
        verbose_name=_(u'unassigned classes
count'), blank=True, null=True, default=None)
    status = models.CharField(
        _(u'input data status'),
        max_length=1,
        choices=STATUS_CHOICES,
        default=QUEUED,
        help_text=_(u'Status of the generation of
timetable')
    )
    input_timetable = models.FileField(
        _(u'input timetable'),
        blank=True,
        upload_to=get_input_filename,
        null=True,
    )

    def __unicode__(self):
        return '[{}] {} ({})'.format(self.status,
self.college, self.semester)

    output_timetable = models.FileField(
        _(u'output timetable'),
        blank=True,
        upload_to=get_output_filename,
        null=True,
    )

    def get_filename(self, data_type):
        path = 'timetables/' + data_type + '/' +
self.college.abbrev + '/'
        new_filename = self.semester.sem + '__' +
self.semester.academic_year + '.xml'
        new_filename = new_filename.replace(" ",
"_").lower()
        return os.path.join(path, new_filename)

    def assign_class_statuses_count(self):
        assigned_classes_count = None
        unassigned_classes_count = None
        if self.status == self.SUCCESSFUL:
            assigned_classes_count =
self.semester.classes.filter(

course__unit__department__college=self.colle
ge,
            ).exclude(
                Q(assigned_room=None) |
                Q(assigned_time_slot=None)
            ).count()

            unassigned_classes_count =
self.semester.classes.filter(

course__unit__department__college=self.colle
ge,
            ).filter(
                Q(assigned_room=None) |
                Q(assigned_time_slot=None)
            ).count()

        self.assigned_classes_count =
assigned_classes_count
        self.unassigned_classes_count =
unassigned_classes_count

        self.save()

    class Meta:
        unique_together = ('college', 'semester', )


@receiver(post_delete, sender=CollegeAdmin)
def post_delete_user(sender, instance, *args,
**kwargs):
    if instance.user:  # just in case user is not
specified
        instance.user.delete()
```

8. upmcts/colleges/tables.py

```
import django_tables2 as tables
from django_tables2_reports.tables import
TableReport

from upmcts.units.models import Class


class TimetableOutputTable(TableReport):
    subject = tables.Column(
        accessor='course.title',
        verbose_name="Subject")

    days = tables.Column(
        accessor='assigned_time_slot.days',
        verbose_name="Days",
        order_by=('-assigned_time_slot.mondays',
'-assigned_time_slot.tuesdays',
                '-assigned_time_slot.wednesdays', '-
assigned_time_slot.thursdays',
                '-assigned_time_slot.fridays', '-
assigned_time_slot.saturdays',
                '-assigned_time_slot.sundays'))

    time_slot = tables.Column(
        accessor='assigned_time_slot.start',
        verbose_name="Time Slot")

    room = tables.Column(
        accessor='assigned_room.abbrev',
        verbose_name="Room")

    instructors = tables.Column(
        accessor='instructor_list',
        verbose_name="Instructors",
        orderable=False)

    class Meta:
        model = Class
        empty_text = 'No class schedules to
display.'
        per_page = 20
        fields = (
            'subject', 'days', 'time_slot', 'room',
'instructors',)

    def __init__(self, *args, **kwargs):
        filename = kwargs.pop('filename',
'timetable')
        super(TimetableOutputTable,
self).__init__(*args, **kwargs)
        self.param_report = filename

    def render_subject(self, record):
        return '{} ({})'.format(record.course.name,
record.additional_name)

    def render_time_slot(self, record):
        hours =
record.assigned_time_slot.start.strftime('%I:%
M %p')
        hours = hours + ' - ' +
record.assigned_time_slot.end.strftime('%I:%
M %p')
        return hours
```

9. upmcts/colleges/urls.py

```
from django.conf.urls import url

from . import views


urlpatterns = [

    url(
        r'^departments/$',
        views.DepartmentListView.as_view(),
        name='department_list'
    ),
    url(
        r'^departments/objects/$',

views.ObjectDepartmentDatatablesView.as_vie
w(),
        name='DT_department_objects'
    ),
    url(
        r'^departments/new/$',
        views.DepartmentCreateView.as_view(),
        name='department_create'
    ),
    url(
        r'^departments/(?P<pk>\d+)/delete/$',
        views.DepartmentDeleteView.as_view(),
        name='department_delete'
    ),
    url(r'^departments/(?P<pk>\d+)/update/$',
        views.DepartmentUpdateView.as_view(),
        name='department_update'),

    url(
        r'^department_admins/$',

views.DepartmentAdminListView.as_view(),
        name='department_admin_list'
    ),
    url(
        r'^department_admins/objects/$',

views.ObjectDepartmentAdminDatatablesView
.as_view(),
        name='DT_department_admin_objects'
    ),
    url(
        r'^department_admins/new/$',

views.DepartmentAdminCreateView.as_view()
,
        name='department_admin_create'
    ),
    url(

r'^department_admins/(?P<pk>\d+)/delete/$',

views.DepartmentAdminDeleteView.as_view()
,
        name='department_admin_delete'
    ),
    url(

r'^department_admins/(?P<pk>\d+)/update/$',

views.DepartmentAdminUpdateView.as_view(
),
        name='department_admin_update'
    ),

    url(
        r'^room_features/$',
        views.RoomFeatureListView.as_view(),
        name='room_feature_list'
    ),
    url(
        r'^room_features/objects/$',

views.ObjectRoomFeatureDatatablesView.as_v
iew(),
        name='DT_room_feature_objects'
    ),
    url(
        r'^room_features/new/$',

views.RoomFeatureCreateView.as_view(),
        name='room_feature_create'
    ),
    url(
        r'^room_features/(?P<pk>\d+)/delete/$',

views.RoomFeatureDeleteView.as_view(),
        name='room_feature_delete'
    ),
    url(r'^room_features/(?P<pk>\d+)/update/$',

views.RoomFeatureUpdateView.as_view(),
        name='room_feature_update'),

    url(
        r'^rooms/$',
        views.RoomListView.as_view(),
        name='room_list'
    ),
    url(
        r'^rooms/objects/$',

views.ObjectRoomDatatablesView.as_view(),
        name='DT_room_objects'
    ),
    url(
        r'^rooms/new/$',
        views.RoomCreateView.as_view(),
        name='room_create'
    ),
    url(
        r'^rooms/(?P<pk>\d+)/delete/$',
        views.RoomDeleteView.as_view(),
        name='room_delete'
    ),
    url(r'^rooms/(?P<pk>\d+)/update/$',
        views.RoomUpdateView.as_view(),
        name='room_update'),

    url(
        r'^timetables/$',
        views.TimetableListView.as_view(),
        name='timetable_list'
    ),
    url(
        r'^timetables/objects/$',

views.ObjectTimetableDatatablesView.as_vie
w(),
        name='DT_timetable_objects'
    ),

    url(
        r'^timetables/new/$',
        views.TimetableCreateView.as_view(),
        name='timetable_create'
    ),

    url(

r'^timetables/(?P<timetable_pk>\d+)/recreate/$'
,
        views.TimetableRecreateView.as_view(),
        name='timetable_recreate'
    ),

    url(

r'^timetables/(?P<timetable_pk>\d+)/output/$',
        views.TimetableOutputView.as_view(),
        name='timetable_output'
    ),

]
```

10. upmcts/colleges/views.py

```
import json

from django.contrib.auth.models import Group
from django.core.exceptions import
PermissionDenied
from django.core.urlresolvers import reverse
from django.db.models import Q
from django.shortcuts import redirect,
get_object_or_404
from django.views.generic import
TemplateView, CreateView, DeleteView,
UpdateView, FormView

from braces.views import
JSONResponseMixin, LoginRequiredMixin
from django_tables2_reports.views import
ReportTableView
from eztables.views import DatatablesView

from upmcts.departments.models import
Department, DepartmentAdmin
```

```
from upmcts.departments.forms import
DepartmentDropdownForm
from upmcts.taskapp.celery import
generate_timetable
from upmcts.units.models import Class
from upmcts.university.models import
Semester
from upmcts.users.views import
AjaxTemplateMixin
from upmcts.utils.utils import LazyEncoder

from .forms import (
    DepartmentForm,
DepartmentAdminCreateForm,
    DepartmentAdminUpdateForm, RoomForm,
    RoomFeatureForm, TimetableForm,
TimetableTimeoutForm)
from .models import RoomFeature, Room,
Timetable
from .tables import TimetableOutputTable


class
CollegeAdminMixin(LoginRequiredMixin):
    def dispatch(self, request, *args, **kwargs):
        if request.user.is_college_admin():
            return super(
                CollegeAdminMixin,
self).dispatch(request, *args, **kwargs)
        else:
            raise PermissionDenied

    def get_college(self):
        return
self.request.user.collegeadmin.college

    def get_context_data(self, **kwargs):
        context = super(CollegeAdminMixin,
self).get_context_data(**kwargs)
        context['college'] = self.get_college()
        return context


class
CollegeAdminWithObjectMixin(CollegeAdmi
nMixin):
    def dispatch(self, request, *args, **kwargs):
        if (request.user.is_college_admin() and
                (self.get_college() ==
self.get_related_college() or
                    self.allowed_none)):
            return super(
                CollegeAdminMixin,
self).dispatch(request, *args, **kwargs)
        else:
            raise PermissionDenied
```

```
class
CollegeAdminFormMixin(CollegeAdminMixin
):
    def get_form_kwargs(self):
        kwargs =
super(CollegeAdminFormMixin,
self).get_form_kwargs()
        kwargs.update({'college':
self.get_college()})
        return kwargs


class
DepartmentListView(CollegeAdminMixin,
TemplateView):
    template_name =
'colleges/department_list.html'

    def get_context_data(self, **kwargs):
        context = super(DepartmentListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] =
'departments'
        context['active_tab'] = 'departments'
        return context


class
ObjectDepartmentDatatablesView(CollegeAd
minMixin, DatatablesView):
    model = Department
    fields = {
        'abbrev': 'abbrev',
        'name': 'name',
        'college': 'college__abbrev',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectDepartmentDatatablesView,
self).get_queryset()
        return qs.filter(college=self.get_college())


class
DepartmentCreateView(CollegeAdminFormMi
xin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = DepartmentForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:department_list')
        return
reverse('colleges:department_create')
```

```
    def get_context_data(self, **kwargs):
        context = super(DepartmentCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Department'
        return context


class
DepartmentDeleteView(CollegeAdminWithOb
jectMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = Department

    def get_related_college(self):
        department = self.get_object()
        return department.college

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:department_list')
        return reverse('colleges:department_list')

    def get_context_data(self, **kwargs):
        context = super(DepartmentDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete
Department'
        return context


class
DepartmentUpdateView(CollegeAdminWithO
bjectMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = Department
    form_class = DepartmentForm

    def get_related_college(self):
        department = self.get_object()
        return department.college

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:department_list')
        return
reverse('colleges:department_update')

    def get_context_data(self, **kwargs):
        context = super(DepartmentUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update
Department'
        return context


class
DepartmentAdminListView(CollegeAdminMix
in, TemplateView):
    template_name =
'colleges/department_admin_list.html'

    def get_context_data(self, **kwargs):
        context =
super(DepartmentAdminListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] =
'departments'
        context['active_tab'] =
'department_admins'
        return context


class
ObjectDepartmentAdminDatatablesView(Colle
geAdminMixin, DatatablesView):
    model = DepartmentAdmin
    fields = {
        'username': 'user__username',
        'department': 'department__abbrev',
        'email': 'user__email',
        'first_name': 'user__first_name',
        'last_name': 'user__last_name',
        'pk': 'pk',
        'user_pk': 'user__pk',
    }

    def get_queryset(self):
        qs =
super(ObjectDepartmentAdminDatatablesView
, self).get_queryset()
        return
qs.filter(department__college=self.get_college(
))


class
DepartmentAdminCreateView(CollegeAdminF
ormMixin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = DepartmentAdminCreateForm

    def form_valid(self, form):
        user = form.save()
        department =
form.cleaned_data.get('department')
        g =
Group.objects.get(name='DepartmentAdmins')
        g.user_set.add(user)
```

```
DepartmentAdmin.objects.create(user=user,
department=department)

        return redirect(self.get_success_url())

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:department_admin_list')
        return
reverse('colleges:department_admin_create')

    def get_context_data(self, **kwargs):
        context =
super(DepartmentAdminCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Department Admin'
        return context


class
DepartmentAdminDeleteView(CollegeAdmin
WithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = DepartmentAdmin

    def get_related_college(self):
        department_admin = self.get_object()
        return
department_admin.department.college

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:department_admin_list')
        return
reverse('colleges:department_admin_list')

    def get_context_data(self, **kwargs):
        context =
super(DepartmentAdminDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete
Department Admin'
        return context


class
DepartmentAdminUpdateView(CollegeAdmin
WithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = DepartmentAdmin
    form_class = DepartmentAdminUpdateForm

    def get_related_college(self):
        department_admin = self.get_object()
        return
department_admin.department.college

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:department_admin_list')
        return
reverse('colleges:departmentadmin_update')

    def get_context_data(self, **kwargs):
        context =
super(DepartmentAdminUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update
Department Admin'
        return context


class
RoomFeatureListView(CollegeAdminMixin,
TemplateView):
    template_name =
'colleges/room_feature_list.html'

    def get_context_data(self, **kwargs):
        context = super(RoomFeatureListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'rooms'
        context['active_tab'] = 'room_features'
        context['college'] = self.get_college()
        return context


class
ObjectRoomFeatureDatatablesView(CollegeAd
minMixin, DatatablesView):
    model = RoomFeature
    fields = {
        'name': 'name',
        'pk': 'pk',
    }


class
RoomFeatureCreateView(CollegeAdminMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = RoomFeatureForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:room_feature_list')
```

```
        return
reverse('colleges:room_feature_create')

    def get_context_data(self, **kwargs):
        context = super(RoomFeatureCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Roomfeature'
        return context


class
RoomFeatureDeleteView(CollegeAdminMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = RoomFeature

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:room_feature_list')
        return
reverse('colleges:room_feature_list')

    def get_context_data(self, **kwargs):
        context = super(RoomFeatureDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Room
Feature'
        return context


class
RoomFeatureUpdateView(CollegeAdminMixin
, JSONResponseMixin, AjaxTemplateMixin,
UpdateView):
    template_name = 'base_form.html'
    model = RoomFeature
    form_class = RoomFeatureForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('colleges:room_feature_list')
        return
reverse('colleges:room_feature_update')

    def get_context_data(self, **kwargs):
        context =
super(RoomFeatureUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Room
Feature'
        return context
```

```
class RoomListView(CollegeAdminMixin,
TemplateView):
    template_name = 'colleges/room_list.html'

    def get_context_data(self, **kwargs):
        context = super(RoomListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'rooms'
        context['active_tab'] = 'rooms'
        return context


class
ObjectRoomDatatablesView(CollegeAdminMi
xin, DatatablesView):
    model = Room
    fields = {
        'abbrev': 'abbrev',
        'name': 'name',
        'college': 'college__abbrev',
        'capacity': 'capacity',
        'is_available': 'is_available',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs = super(ObjectRoomDatatablesView,
self).get_queryset()
        return qs.filter(Q(college=None) |
Q(college=self.get_college()))


class
RoomCreateView(CollegeAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = RoomForm
    allowed_none = True

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('colleges:room_list')
        return reverse('colleges:room_create')

    def get_context_data(self, **kwargs):
        context = super(RoomCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Room'
        return context


class
RoomDeleteView(CollegeAdminWithObjectM
ixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = Room
    allowed_none = True

    def get_related_college(self):
        room = self.get_object()
        return room.college

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('colleges:room_list')
        return reverse('colleges:room_list')

    def get_context_data(self, **kwargs):
        context = super(RoomDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Room'
        return context


class
RoomUpdateView(CollegeAdminWithObject
Mixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = Room
    form_class = RoomForm
    allowed_none = True

    def get_related_college(self):
        room = self.get_object()
        return room.college

    def post(self, request, *args, **kwargs):
        return super(RoomUpdateView,
self).post(request, *args, **kwargs)

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('colleges:room_list')
        return reverse('colleges:room_update')

    def get_context_data(self, **kwargs):
        context = super(RoomUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Room'
        return context


class TimetableListView(CollegeAdminMixin,
TemplateView):
    template_name =
'colleges/timetable_list.html'

    def get_context_data(self, **kwargs):
        context = super(TimetableListView,
self).get_context_data(**kwargs)
        context['sem_list'] =
json.dumps(dict(Semester.SEM_CHOICES),
cls=LazyEncoder)
        context['status_list'] =
json.dumps(dict(Timetable.STATUS_CHOICE
S), cls=LazyEncoder)
        context['active_tab_parent'] = 'timetables'
        context['active_tab'] =
'generate_timetables'
        return context


class
ObjectTimetableDatatablesView(CollegeAdmi
nMixin, DatatablesView):
    model = Timetable
    fields = {
        'semester_sem': 'semester__sem',
        'semester_academic_year':
'semester__academic_year',
        'status': 'status',
        'assigned_classes_count':
'assigned_classes_count',
        'unassigned_classes_count':
'unassigned_classes_count',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectTimetableDatatablesView,
self).get_queryset()
        return qs.filter(college=self.get_college())


class
TimetableCreateView(CollegeAdminFormMixi
n, JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = TimetableForm
    allowed_none = True

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('colleges:timetable_list')
        return reverse('colleges:timetable_create')

    def get_context_data(self, **kwargs):
        context = super(TimetableCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Timetable'
        return context


class
TimetableRecreateView(CollegeAdminFormM
ixin, JSONResponseMixin,
AjaxTemplateMixin, FormView):
    template_name = 'base_form.html'
    form_class = TimetableTimeoutForm

    def dispatch(self, request, *args, **kwargs):
        self.timetable =
get_object_or_404(Timetable,
pk=self.kwargs['timetable_pk'])
        return super(TimetableRecreateView,
self).dispatch(request, *args, **kwargs)

    def form_valid(self, form):
        timeout =
form.cleaned_data.get('solve_time', 60)
        if self.timetable.status ==
Timetable.SUCCESSFUL or
self.timetable.status == Timetable.FAILED:
            self.timetable.status =
Timetable.QUEUED
            self.timetable.save()
            generate_timetable.delay(self.timetable,
timeout)
        return redirect(self.get_success_url())

    def get_related_college(self):
        return self.timetable.college

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('colleges:timetable_list')
        return reverse('colleges:timetable_list')

    def get_context_data(self, **kwargs):
        context = super(TimetableRecreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Are you sure you
want to recreate a new timetable for the
semester?'
        return context


class
TimetableOutputView(CollegeAdminMixin,
ReportTableView):
    template_name =
'colleges/timetable_output.html'

    table_class = TimetableOutputTable
    model = Class

    def dispatch(self, request, *args, **kwargs):
        self.timetable =
get_object_or_404(Timetable,
pk=self.kwargs['timetable_pk'])
        self.college = self.timetable.college
        self.semester = self.timetable.semester
        return super(TimetableOutputView,
self).dispatch(request, *args, **kwargs)

    def get_table(self, **kwargs):
        filename = self.semester.sem + '__' +
self.semester.academic_year
        filename = filename.replace(" ",
"_").lower()
        kwargs.update({
            'filename': filename,
        })
        return super(TimetableOutputView,
self).get_table(**kwargs)

    def get_queryset(self):
        qs = super(TimetableOutputView,
self).get_queryset()
        qs = qs.filter(
            semester=self.semester,

course__unit__department__college=self.colle
ge,
        )
        department =
self.request.GET.get('department', None)
        if department:
            qs =
qs.filter(course__unit__department=department
)
        return qs

    def get_context_data(self, **kwargs):
        context = super(TimetableOutputView,
self).get_context_data(**kwargs)
        department =
self.request.GET.get('department', None)
        context['timetable'] = self.timetable
        context['department_form'] =
DepartmentDropdownForm(
            college=self.college,
department=department
        )
        return context
```

11. upmcts/departments/forms.py

```
from django import forms
from django.utils.translation import
ugettext_lazy as _

from crispy_forms.helper import FormHelper
from django_select2.forms import
Select2Widget, Select2MultipleWidget

from upmcts.departments.models import
Department
from upmcts.units.models import (
    Class, Unit, UnitAdmin,
BlockSectionClasses)
from upmcts.university.forms import
CollegeAdminUpdateForm
```

```
from upmcts.users.forms import
UserCreateForm
from upmcts.users.models import User

from .models import BlockSection


class UnitForm(forms.ModelForm):

    class Meta:
        model = Unit
        fields = ('name', 'abbrev', )

    def __init__(self, *args, **kwargs):
        self.department =
kwargs.pop('department', None)

        super(UnitForm, self).__init__(*args,
**kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        unit = super(
            UnitForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.department:
            unit.department = self.department

        if commit:
            unit.save()

        return unit


class UnitAdminCreateForm(UserCreateForm):
    unit =
forms.ModelChoiceField(queryset=Unit.object
s.exclude(name='None'))

    class Meta:
        model = User
        fields = (
            'username', 'password1', 'password2',
'unit',
            'email', 'first_name', 'last_name', )
        widgets = {
            'unit': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.department =
kwargs.pop('department', None)
        super(UnitAdminCreateForm,
self).__init__(*args, **kwargs)

        self.fields['unit'].queryset =
Unit.objects.filter(
            department=self.department
        ).exclude(name='-----')

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class
UnitAdminUpdateForm(CollegeAdminUpdate
Form):
    class Meta:
        model = UnitAdmin
        fields = (
            'username', 'unit', 'email', 'first_name',
'last_name',)

    def __init__(self, *args, **kwargs):
        self.department =
kwargs.pop('department', None)
        super(UnitAdminUpdateForm,
self).__init__(*args, **kwargs)

        self.fields['unit'].queryset =
Unit.objects.filter(
            department=self.department
        ).exclude(name='-----')

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class BlockSectionForm(forms.ModelForm):

    class Meta:
        model = BlockSection
        fields = ('name', 'year_level', )

        widgets = {
            'year_level': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.department =
kwargs.pop('department', None)

        super(BlockSectionForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        block_section = super(
            BlockSectionForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.department:
            block_section.department =
self.department

        if commit:
            block_section.save()

        return block_section


class
BlockSectionClassesForm(forms.ModelForm):

    class Meta:
        model = BlockSectionClasses
        fields = ('block_section', 'semester', )

        widgets = {
            'block_section': Select2Widget,
            'semester': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.department =
kwargs.pop('department', None)

        super(BlockSectionClassesForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        block_section = super(
            BlockSectionClassesForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.department:
            block_section.department =
self.department

        if commit:
            block_section.save()

        return block_section


class
BlockSectionClassesClassForm(forms.Form):
    classes = forms.ModelMultipleChoiceField(
        queryset=Class.objects.all(),
        widget=Select2MultipleWidget)

    def __init__(self, *args, **kwargs):
        self.department =
kwargs.pop('department', None)
        self.block_section_classes =
kwargs.pop('block_section_classes', None)
        super(BlockSectionClassesClassForm,
self).__init__(*args, **kwargs)

        class_qs =
Class.objects.filter(course__unit__department=
self.department)

        class_qs =
class_qs.exclude(pk__in=self.block_section_cl
asses.classes.all())

        self.fields['classes'].queryset = class_qs

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        classes = self.cleaned_data.get('classes',
None)
        if commit:

self.block_section_classes.classes.add(*classes)

        return classes


class DepartmentDropdownForm(forms.Form):
    department = forms.ModelChoiceField(
        widget=Select2Widget,
        queryset=Department.objects.all(),
        required=True,
        empty_label=_('All departments'),
    )

    def __init__(self, college, department, *args,
**kwargs):
        super(DepartmentDropdownForm,
self).__init__(*args, **kwargs)
        self.fields['department'].queryset =
Department.objects.filter(college=college)
```

```
        self.fields['department'].initial =
department
        self.fields['department'].label = ''

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.disable_csrf = True
        self.helper.template_pack = 'bootstrap3'
```

12. upmcts/departments/models.py

```
from django.db import models
from django.utils.translation import
ugettext_lazy as _

from model_utils import Choices

from upmcts.colleges.models import College
from upmcts.users.models import User


class Department(models.Model):
    college = models.ForeignKey(College,
related_name='departments')
    name = models.CharField(
        verbose_name=_(u'Department Name'),
        max_length=255,
        unique=True)
    abbrev = models.CharField(
        verbose_name=_(u'Department
Abbreviation'),
        max_length=20,
        help_text=_(u'e.g. DPSM'),
        unique=True)

    def __unicode__(self):
        return '{} ({})'.format(self.name,
self.abbrev)


class DepartmentAdmin(models.Model):
    user = models.OneToOneField(User)
    department = models.ForeignKey(
        Department,
        related_name='department_admins')

    def __unicode__(self):
        return '{} ({})'.format(self.user.username,
self.department.abbrev)


class BlockSection(models.Model):
    FIRST = 'First Year'
    SECOND = 'Second Year'
    THIRD = 'Third Year'
    FOURTH = 'Fourth Year'
    FIFTH = 'Fifth Year'
    SIXTH = 'Sixth Year'
    YEAR_LEVEL_CHOICES = Choices(
        (FIRST, _(u'First Year')),
        (SECOND, _(u'Second Year')),
        (THIRD, _(u'Third Year')),
        (FOURTH, _(u'Fourth Year')),
        (FIFTH, _(u'Fifth Year')),
        (SIXTH, _(u'Sixth Year')),
    )
    department =
models.ForeignKey(Department,
related_name='block_sections')
    name = models.CharField(
        verbose_name=_(u'Block Name/Number'),
        max_length=30)
    year_level = models.CharField(
        _(u'Year Level'),
        max_length=20,
        choices=YEAR_LEVEL_CHOICES,
        default=FIRST)

    class Meta:
        unique_together = ('name', 'year_level', )

    def __unicode__(self):
        return '{} ({})'.format(self.name,
self.get_year_level_display())
```

13. upmcts/department/urls.py

```
from django.conf.urls import url

from . import views


urlpatterns = [

    url(
        r'^units/$',
        views.UnitListView.as_view(),
        name='unit_list'
    ),
    url(
        r'^units/objects/$',

views.ObjectUnitDatatablesView.as_view(),
        name='DT_unit_objects'
    ),
    url(
        r'^units/new/$',
        views.UnitCreateView.as_view(),
        name='unit_create'
    ),
    url(
        r'^units/(?P<pk>\d+)/delete/$',
        views.UnitDeleteView.as_view(),
        name='unit_delete'
    ),
    url(r'^units/(?P<pk>\d+)/update/$',
        views.UnitUpdateView.as_view(),
        name='unit_update'),
  url(
    r'^unit_admins/$',
    views.UnitAdminListView.as_view(),
    name='unit_admin_list'
  ),
  url(
    r'^unit_admins/objects/$',

views.ObjectUnitAdminDatatablesView.as_vie
w(),
    name='DT_unit_admin_objects'
  ),
  url(
    r'^unit_admins/new/$',
    views.UnitAdminCreateView.as_view(),
    name='unit_admin_create'
  ),
  url(
    r'^unit_admins/(?P<pk>\d+)/delete/$',
    views.UnitAdminDeleteView.as_view(),
    name='unit_admin_delete'
  ),
  url(
    r'^unit_admins/(?P<pk>\d+)/update/$',
    views.UnitAdminUpdateView.as_view(),
    name='unit_admin_update'
  ),

  url(
    r'^block_sections/$',
    views.BlockSectionListView.as_view(),
    name='block_section_list'
  ),
  url(
    r'^block_sections/objects/$',

views.ObjectBlockSectionDatatablesView.as_v
iew(),
    name='DT_block_section_objects'
  ),
  url(
    r'^block_sections/new/$',
    views.BlockSectionCreateView.as_view(),
    name='block_section_create'
  ),
  url(
    r'^block_sections/(?P<pk>\d+)/delete/$',
    views.BlockSectionDeleteView.as_view(),
    name='block_section_delete'
  ),
  url(r'^block_sections/(?P<pk>\d+)/update/$',

views.BlockSectionUpdateView.as_view(),
    name='block_section_update'),

  url(
    r'^block_section_classes/$',
views.BlockSectionClassesListView.as_view(),
    name='block_section_class_list'
  ),
  url(
    r'^block_section_classes/objects/$',

views.ObjectBlockSectionClassesDatatablesVi
ew.as_view(),
    name='DT_block_section_class_objects'
  ),
  url(
    r'^block_section_classes/new/$',

views.BlockSectionClassesCreateView.as_vie
w(),
    name='block_section_class_create'
  ),
  url(

r'^block_section_classes/(?P<pk>\d+)/delete/$',

views.BlockSectionClassesDeleteView.as_vie
w(),
    name='block_section_class_delete'
  ),

  url(

r'^block_section_classes/(?P<pk>\d+)/details/$'
,

views.BlockSectionClassesDetailView.as_view
(),
    name='block_section_class_details'
  ),

  url(

r'^block_section_classes/(?P<section_classes_p
k>\d+)/classes/objects$',

views.ObjectBlockSectionClassesClassListDat
atablesView.as_view(),

name='DT_block_section_class_details_objects
'
  ),

  url(

r'^block_section_classes/(?P<section_classes_p
k>\d+)/classes/create$',

views.BlockSectionClassesClassCreateView.as
_view(),
    name='block_section_class_add'
  ),
    url(

r'^block_section_classes/(?P<section_classes_p
k>\d+)/classes/(?P<class_pk>\d+)/delete$',

views.BlockSectionClassesClassDeleteView.as
_view(),
        name='block_section_class_remove'
    ),
]
```

14. upmcts/departments/views.py

```
import json
import re

from django.contrib.auth.models import Group
from django.core.exceptions import
PermissionDenied
from django.core.urlresolvers import reverse
from django.shortcuts import redirect,
get_object_or_404
from django.utils.six import text_type
from django.views.generic import (
    TemplateView, CreateView, DeleteView,
UpdateView, FormView)

from braces.views import
JSONResponseMixin, LoginRequiredMixin
from eztables.views import DatatablesView

from upmcts.units.models import (
    Class, Unit, UnitAdmin,
BlockSectionClasses, Instructor)
from upmcts.university.models import
Semester
from upmcts.users.views import
AjaxTemplateMixin
from upmcts.utils.utils import LazyEncoder

from .forms import (
    UnitForm, UnitAdminCreateForm,
    UnitAdminUpdateForm, BlockSectionForm,
    BlockSectionClassesForm,
BlockSectionClassesClassForm)
from .models import BlockSection


RE_FORMATTED = re.compile(r'\{(\w+)\}')


class
DepartmentAdminMixin(LoginRequiredMixin)
:
    def dispatch(self, request, *args, **kwargs):
        if request.user.is_department_admin():
            return super(
                DepartmentAdminMixin,
self).dispatch(request, *args, **kwargs)
        else:
            raise PermissionDenied

    def get_department(self):
        return
self.request.user.departmentadmin.department

    def get_context_data(self, **kwargs):
        context = super(DepartmentAdminMixin,
self).get_context_data(**kwargs)
        context['department'] =
self.get_department()
        return context


class
DepartmentAdminWithObjectMixin(Departme
ntAdminMixin):
    def dispatch(self, request, *args, **kwargs):
        if (request.user.is_department_admin()
and
                self.get_department() ==
self.get_related_department()):
            return super(
                DepartmentAdminMixin,
self).dispatch(request, *args, **kwargs)
        else:
            raise PermissionDenied


class
DepartmentAdminFormMixin(DepartmentAdm
inMixin):
    def get_form_kwargs(self):
        kwargs =
super(DepartmentAdminFormMixin,
self).get_form_kwargs()
        kwargs.update({'department':
self.get_department()})
        return kwargs


class UnitListView(DepartmentAdminMixin,
TemplateView):
    template_name = 'departments/unit_list.html'

    def get_context_data(self, **kwargs):
        context = super(UnitListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'units'
        context['active_tab'] = 'units'
        return context
```

```
class
ObjectUnitDatatablesView(DepartmentAdmin
Mixin, DatatablesView):
    model = Unit
    fields = {
        'abbrev': 'abbrev',
        'name': 'name',
        'department': 'department__abbrev',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs = super(ObjectUnitDatatablesView,
self).get_queryset()
        return
qs.filter(department=self.get_department()).exc
lude(name='-----')


class
UnitCreateView(DepartmentAdminFormMixin
, JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = UnitForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('departments:unit_list')
        return reverse('departments:unit_create')

    def get_context_data(self, **kwargs):
        context = super(UnitCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Unit'
        return context


class
UnitDeleteView(DepartmentAdminWithObject
Mixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = Unit

    def get_related_department(self):
        unit = self.get_object()
        return unit.department

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('departments:unit_list')
        return reverse('departments:unit_list')

    def get_context_data(self, **kwargs):
        context = super(UnitDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Unit'
        return context


class
UnitUpdateView(DepartmentAdminWithObjec
tMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = Unit
    form_class = UnitForm

    def get_related_department(self):
        unit = self.get_object()
        return unit.department

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('departments:unit_list')
        return reverse('departments:unit_update')

    def get_context_data(self, **kwargs):
        context = super(UnitUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Unit'
        return context


class
UnitAdminListView(DepartmentAdminMixin,
TemplateView):
    template_name =
'departments/unit_admin_list.html'

    def get_context_data(self, **kwargs):
        context = super(UnitAdminListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'units'
        context['active_tab'] = 'unit_admins'
        return context


class
ObjectUnitAdminDatatablesView(Department
AdminMixin, DatatablesView):
    model = UnitAdmin
    fields = {
        'username': 'user__username',
        'unit': 'unit__abbrev',
        'email': 'user__email',
        'first_name': 'user__first_name',
        'last_name': 'user__last_name',
        'pk': 'pk',
        'user_pk': 'user__pk',
    }

    def get_queryset(self):
        qs =
super(ObjectUnitAdminDatatablesView,
self).get_queryset()
        return
qs.filter(unit__department=self.get_department
())


class UnitAdminCreateView(
    DepartmentAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = UnitAdminCreateForm

    def form_valid(self, form):
        user = form.save()
        unit = form.cleaned_data.get('unit')
        g =
Group.objects.get(name='UnitAdmins')
        g.user_set.add(user)

        UnitAdmin.objects.create(user=user,
unit=unit)

        return redirect(self.get_success_url())

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:unit_admin_list')
        return
reverse('departments:unit_admin_create')

    def get_context_data(self, **kwargs):
        context = super(UnitAdminCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Unit
Admin'
        return context


class UnitAdminDeleteView(
    DepartmentAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = UnitAdmin

    def get_related_department(self):
        unit_admin = self.get_object()
        return unit_admin.unit.department

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:unit_admin_list')
        return
reverse('departments:unit_admin_list')

    def get_context_data(self, **kwargs):
        context = super(UnitAdminDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Unit
Admin'
        return context


class UnitAdminUpdateView(
    DepartmentAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
UpdateView):
    template_name = 'base_form.html'
    model = UnitAdmin
    form_class = UnitAdminUpdateForm

    def get_related_department(self):
        unit_admin = self.get_object()
        return unit_admin.unit.department

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:unit_admin_list')
        return
reverse('departments:unitadmin_update')

    def get_context_data(self, **kwargs):
        context = super(UnitAdminUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Unit
Admin'
        return context


class
BlockSectionListView(DepartmentAdminMixi
n, TemplateView):
    template_name =
'departments/block_section_list.html'

    def get_context_data(self, **kwargs):
        context = super(BlockSectionListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] =
'block_sections'
        context['active_tab'] = 'block_sections'
        return context


class
ObjectBlockSectionDatatablesView(Departme
ntAdminMixin, DatatablesView):
    model = BlockSection
    fields = {
        'name': 'name',
        'year_level': 'year_level',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectBlockSectionDatatablesView,
self).get_queryset()
        return
qs.filter(department=self.get_department())


class BlockSectionCreateView(
    DepartmentAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = BlockSectionForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:block_section_list')
        return
reverse('departments:block_section_create')

    def get_context_data(self, **kwargs):
        context = super(BlockSectionCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Blocksection'
        return context


class BlockSectionDeleteView(
    DepartmentAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = BlockSection

    def get_related_department(self):
        block_section = self.get_object()
        return block_section.department

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:block_section_list')
        return
reverse('departments:block_section_list')

    def get_context_data(self, **kwargs):
        context = super(BlockSectionDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Block
Section'
        return context


class BlockSectionUpdateView(
    DepartmentAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
UpdateView):
    template_name = 'base_form.html'
    model = BlockSection
    form_class = BlockSectionForm

    def get_related_department(self):
        block_section = self.get_object()
        return block_section.department

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:block_section_list')
        return
reverse('departments:block_section_update')

    def get_context_data(self, **kwargs):
        context = super(BlockSectionUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Block
Section'
        return context


class
BlockSectionClassesListView(DepartmentAdm
inMixin, TemplateView):
    template_name =
'departments/block_section_class_list.html'

    def get_context_data(self, **kwargs):
        context =
super(BlockSectionClassesListView,
self).get_context_data(**kwargs)
        context['sem_list'] =
json.dumps(dict(Semester.SEM_CHOICES),
cls=LazyEncoder)
        context['active_tab_parent'] =
'block_sections'
        context['active_tab'] =
'block_section_classes'
        return context


class
ObjectBlockSectionClassesDatatablesView(De
partmentAdminMixin, DatatablesView):
    model = BlockSectionClasses
    fields = {
        'block_section_name':
'block_section__name',
        'block_section_year_level':
'block_section__year_level',
        'semester_academic_year':
'semester__academic_year',
        'semester_sem': 'semester__sem',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectBlockSectionClassesDatatablesVi
ew, self).get_queryset()
        return
qs.filter(block_section__department=self.get_d
epartment())

    def sort_col_2(self, direction):
        return ('%ssemester__academic_year' %
direction, '%ssemester__sem' % direction)


class BlockSectionClassesCreateView(
    DepartmentAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = BlockSectionClassesForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:block_section_class_list')
        return
reverse('departments:block_section_class_creat
e')

    def get_context_data(self, **kwargs):
        context =
super(BlockSectionClassesCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
BlockSectionClasses'
        return context


class BlockSectionClassesDeleteView(
    DepartmentAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = BlockSectionClasses

    def get_related_department(self):
        block_section_class = self.get_object()
        return
block_section_class.block_section.department

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:block_section_class_list')
        return
reverse('departments:block_section_class_list')

    def get_context_data(self, **kwargs):
        context =
super(BlockSectionClassesDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete
BlockSectionClasses'
        return context


class
BlockSectionClassesDetailView(DepartmentA
dminMixin, TemplateView):
    template_name =
'departments/block_section_class_details.html'

    def get_context_data(self, **kwargs):
        context =
super(BlockSectionClassesDetailView,
self).get_context_data(**kwargs)
        context['block_section_classes'] =
get_object_or_404(BlockSectionClasses,
pk=self.kwargs['pk'])
        context['active_tab_parent'] =
'block_sections'
        context['active_tab'] =
'block_section_classes'
        return context


class
ObjectBlockSectionClassesClassListDatatables
View(DepartmentAdminMixin,
DatatablesView):
    model = Class
    fields = {
        'title': 'course__title',
        'additional_name': 'additional_name',
        'pk': 'pk',
    }

    def dispatch(self, request, *args, **kwargs):
        self.block_section_classes =
get_object_or_404(BlockSectionClasses,
pk=self.kwargs['section_classes_pk'])
        return super(

ObjectBlockSectionClassesClassListDatatables
View, self).dispatch(request, *args, **kwargs)

    def get_row(self, row):
        '''Add class pk'''
        row.update({
            'section_classes_pk':
self.block_section_classes.pk
        })

        updated_fields = dict(self.fields)

updated_fields.update({'section_classes_pk':
'section_classes_pk'})

        if isinstance(self.fields, dict):
            return dict([
                (key, text_type(value).format(**row)
if RE_FORMATTED.match(value) else
row[value])
                for key, value in
updated_fields.items()
            ])
        else:
            return [text_type(field).format(**row) if
RE_FORMATTED.match(field)
                    else row[field]
                    for field in updated_fields]

    def get_queryset(self):
        return
self.block_section_classes.classes.all()


class BlockSectionClassesClassCreateView(
    DepartmentAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
FormView):
    template_name = 'base_form.html'
    form_class = BlockSectionClassesClassForm

    def dispatch(self, request, *args, **kwargs):
        self.block_section_classes =
get_object_or_404(BlockSectionClasses,
pk=self.kwargs['section_classes_pk'])
        return super(
            BlockSectionClassesClassCreateView,
self).dispatch(request, *args, **kwargs)

    def get_form_kwargs(self):
        kwargs =
super(BlockSectionClassesClassCreateView,
self).get_form_kwargs()
        kwargs.update({'block_section_classes':
self.block_section_classes})
        return kwargs

    def form_valid(self, form):
        form.save()
        return redirect(self.get_success_url())

    def get_related_department(self):
        return
self.block_section_classes.block_section.depart
ment

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('departments:block_section_class_detai
ls', kwargs={'pk':
self.block_section_classes.pk})
        return
reverse('departments:block_section_class_detai
ls', kwargs={'pk':
self.block_section_classes.pk})

    def get_context_data(self, **kwargs):
        context =
super(BlockSectionClassesClassCreateView,
self).get_context_data(**kwargs)
        context['block_section_classes'] =
self.block_section_classes
        context['popup_title'] = 'Add New Class
for the Block'
        return context


class
BlockSectionClassesClassDeleteView(Departm
entAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
TemplateView):
    template_name = 'base_delete_form.html'

    def get_related_department(self):
        self.block_section_classes =
get_object_or_404(BlockSectionClasses,
pk=self.kwargs['section_classes_pk'])
        return
self.block_section_classes.block_section.depart
ment

    def get(self, request, *args, **kwargs):
        context = self.get_context_data(**kwargs)
        return self.render_to_response(context)

    def post(self, request, *args, **kwargs):
        course_class = get_object_or_404(Class,
pk=self.kwargs['class_pk'])

self.block_section_classes.classes.remove(cour
se_class)
        return
super(BlockSectionClassesClassDeleteView,
self).get(request, *args, **kwargs)

    def get_success_url(self):
        if self.request.is_ajax():
```

```
        return
reverse('departments:block_section_class_detai
ls', kwargs={'pk':
self.block_section_classes.pk})
      return
reverse('departments:block_section_class_detai
ls', kwargs={'pk':
self.block_section_classes.pk})

   def get_context_data(self, **kwargs):
      context =
super(BlockSectionClassesClassDeleteView,
self).get_context_data(**kwargs)
      context['popup_title'] = 'Delete Class from
Block'
      return context
```

15. upmcts/static/css/project.css

```
.alert-debug {
  background-color: #fff;
  border-color: #d6e9c6;
  color: #000; }

.alert-error {
  background-color: #f2dede;
  border-color: #eed3d7;
  color: #b94a48; }

.navbar-static-top {
  margin-bottom: 19px; }

/* Override Datatables CSS*/
.pagination {
  display: inline-block;
  padding-left: 0;
  margin-top: 1rem;
  margin-bottom: 1rem;
  border-radius: .25rem; }

.pagination > ul > li {
  display: inline; }

.pagination > ul > li > a,
.pagination > ul > li > span {
  position: relative;
  float: left;
  padding: .5rem .75rem;
  margin-left: -1px;
  line-height: 1.5;
  color: #0275d8;
  text-decoration: none;
  background-color: #fff;
  border: 1px solid #ddd; }

.pagination > ul > li:first-child > a,
.pagination > ul > li:first-child > span {
  margin-left: 0;
  border-top-left-radius: .25rem;
  border-bottom-left-radius: .25rem; }

.pagination > ul > li:last-child > a,
.pagination > ul > li:last-child > span {
  border-top-right-radius: .25rem;
  border-bottom-right-radius: .25rem; }

.pagination > ul > li > a:focus,
.pagination > ul > li > a:hover,
.pagination > ul > li > span:focus,
.pagination > ul > li > span:hover {
  color: #014c8c;
  background-color: #eceeef;
  border-color: #ddd; }

.pagination > ul > .active > a,
.pagination > ul > .active > a:focus,
.pagination > ul > .active > a:hover,
.pagination > ul > .active > span,
.pagination > ul > .active > span:focus,
.pagination > ul > .active > span:hover {
  z-index: 2;
  color: #fff;
  cursor: default;
  background-color: #0275d8;
  border-color: #0275d8; }

.pagination > ul > .disabled > span,
.pagination > ul > .disabled > span:focus,
.pagination > ul > .disabled > span:hover,
.pagination > ul > .disabled > a,
.pagination > ul > .disabled > a:focus,
.pagination > ul > .disabled > a:hover {
  color: #818a91;
  cursor: not-allowed;
  background-color: #fff;
  border-color: #ddd; }

.dataTables_wrapper {
  font: 14px "Lucida
Grande",Verdana,Arial,Helvetica,sans-serif
!important;
  color: #373a3c;
  line-height: 1;
  background-color: #fff; }
  .dataTables_wrapper .table th,
.dataTables_wrapper .table td {
    vertical-align: middle !important; }

.dataTables_processing {
  text-align: center; }

.column-edit, .column-delete {
  text-align: center; }

.success {
  color: #4CAE4C; }
```

```
.danger {
  color: #D9534F; }

.text-center {
  text-align: center; }

.margin-top-10 {
  margin-top: 10px; }

.margin-top-20 {
  margin-top: 20px; }

.margin-top-30 {
  margin-top: 30px; }

.margin-bottom-10 {
  margin-bottom: 10px; }

.margin-bottom-20 {
  margin-bottom: 20px; }

.margin-bottom-30 {
  margin-bottom: 30px; }

.margin-right-10 {
  margin-right: 10px; }

.margin-right-20 {
  margin-right: 20px; }

.margin-right-30 {
  margin-right: 30px; }

.margin-left-10 {
  margin-left: 10px; }

.margin-left-20 {
  margin-left: 20px; }

.margin-left-30 {
  margin-left: 30px; }
```

16. upmcts/static/js/colleges/department.js

```
var render_update = function(data, type, row) {

   return '<button data-url="'+

Django.url('colleges:department_update', data)
+
          '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

   return '<button data-url="'+
          Django.url('colleges:department_delete',
data) +
          '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#department-
table').dataTable({
   "bPaginate": true,
   "sPaginationType": "bootstrap",
   "bProcessing": true,
   "bServerSide": true,
   "sAjaxSource":
Django.url('colleges:DT_department_objects'),
   "aoColumns": [
       { "mData": "abbrev" },
       { "mData": "name" },
       { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
       { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},

   ]
});
```

17. upmcts/static/js/colleges/department_admin.js

```
var render_update_details = function(data,
type, row) {

   return '<button data-url="'+

Django.url('colleges:department_admin_update
', data) +
          '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_update_password = function(data,
type, row) {

   return '<button data-url="'+
          Django.url('users:password_update',
data) +
          '" class="popup-button btn btn-
default"><i class="fa fa-key"></i></button>';
};

var render_delete = function(data, type, row) {

   return '<button data-url="'+

Django.url('colleges:department_admin_delete',
data) +
          '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#department-admin-
table').dataTable({
   "bPaginate": true,
   "sPaginationType": "bootstrap",
   "bProcessing": true,
   "bServerSide": true,
   "sAjaxSource":
Django.url('colleges:DT_department_admin_o
bjects'),
   "aoColumns": [
      { "mData": "username" },
      { "mData": "department" },
      { "mData": "email" },
      { "mData": "first_name" },
      { "mData": "last_name" },
      { "mData": "pk", "bSortable": false,
'mRender': render_update_details,  "sClass":
"column-edit" },
      { "mData": "user_pk", "bSortable": false,
'mRender': render_update_password,  "sClass":
"column-edit" },
      { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


   ]
});
```

18. upmcts/static/js/colleges/room.js

```
var render_boolean = function(data, type, row)
{
   if (data) {
      return '<i class="fa fa-check success
center"></i>';
   } else {
      return '<i class="fa fa-close danger
center"></i>';

   }
};

var render_update = function(data, type, row) {

   return '<button data-url="'+
         Django.url('colleges:room_update',
data) +
         '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

   return '<button data-url="'+
         Django.url('colleges:room_delete', data)
+
         '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#room-table').dataTable({
   "bPaginate": true,
   "sPaginationType": "bootstrap",
   "bProcessing": true,
   "bServerSide": true,
   "sAjaxSource":
Django.url('colleges:DT_room_objects'),
   "aoColumns": [
      { "mData": "abbrev" },
      { "mData": "name" },
      { "mData": "capacity" },
      { "mData": "is_available", 'mRender':
render_boolean, 'sClass': 'text-center'},
      { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
      { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},

   ]
});
```

19. upmcts/static/js/colleges/room_feature.js

```
var render_update = function(data, type, row) {

   return '<button data-url="'+

Django.url('colleges:room_feature_update',
data) +
         '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

   return '<button data-url="'+

Django.url('colleges:room_feature_delete',
data) +
         '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#room_feature-
table').dataTable({
   "bPaginate": true,
   "sPaginationType": "bootstrap",
   "bProcessing": true,
   "bServerSide": true,
    "sAjaxSource":
Django.url('colleges:DT_room_feature_objects'
),
    "aoColumns": [
        { "mData": "name" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

20. upmcts/static/js/colleges/timetable.js

```
var render_sem = function(data, type, row) {
    return sem_list[data] + ', ' +
row.semester_academic_year;
};

var render_status = function(data, type, row) {
    return status_list[data];
};

var render_update = function(data, type, row) {
    if (row.status == 'S' || row.status == 'F') {
        return '<button data-url="'+

Django.url('colleges:timetable_recreate', data)
+
                '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button';
    }
    return '-';
};

var render_view = function(data, type, row) {
    if (row.status == 'S') {
        return '<a href="' +
Django.url('colleges:timetable_output', data) +
               '" class="btn btn-default"><i class="fa
fa-list"></i></a>';
    }
    return '-';
};

var pollTimetables = function() {
    setTimeout(function(){
        $('#timetable-
table_processing').css('color', '#FFF');
        datatable.fnDraw();
        pollTimetables();
    }, 5000);
};

popupSubmitSuccessFunction = function() {
    $('#timetable-table_processing').css('color',
'#373A3C');
};

var datatable = $('#timetable-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('colleges:DT_timetable_objects'),
    "aoColumns": [
        { "mData": "semester_sem", 'mRender':
render_sem},
        { "mData": "status", 'mRender':
render_status},
        { "mData": "assigned_classes_count" },
        { "mData": "unassigned_classes_count" },
        { "mData": "pk", "bSortable": false,
'mRender': render_view,  "sClass": "column-
edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },

    ]
});

$(document).ready(function() {
    pollTimetables();
});
```

21. upmcts/static/js/departments/block_section.js

```
var render_update = function(data, type, row) {

    return '<button data-url="'+

Django.url('departments:block_section_update',
data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('departments:block_section_delete',
data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button';
};

var datatable = $('#block_section-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('departments:DT_block_section_obj
ects'),
    "aoColumns": [
        { "mData": "name" },
        { "mData": "year_level" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

22. upmcts/static/js/departments/block_section_class_details.js

```
var render_title = function(data, type, row) {
    return data + ' (' + row.additional_name + ')';
};

var render_details = function(data, type, row) {
    return '<a href="' +
Django.url('units:class_details', data) +
            '" class="btn btn-default"><i class="fa
fa-pencil"></i></a>'
};


var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('departments:block_section_class_r
emove', row.section_classes_pk, data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#class-table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('departments:DT_block_section_cla
ss_details_objects', {section_classes_pk:
section_classes_pk}),
    "aoColumns": [
        { "mData": "title", 'mRender': render_title
},
        { "mData": "pk", "bSortable": false,
'mRender': render_details ,  "sClass": "column-
edit"},
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

23. upmcts/static/js/departments/block_section_class_list.js

```
var render_sem = function(data, type, row) {
    return sem_list[data] + ', ' +
row.semester_academic_year;
};

var render_update = function(data, type, row) {

    return '<a href="' +
Django.url('departments:block_section_class_d
etails', data) +
            '" class="btn btn-default"><i class="fa
fa-pencil"></i></a>'
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('departments:block_section_class_d
elete', data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#block_section_classes-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('departments:DT_block_section_cla
ss_objects'),
    "aoColumns": [
        { "mData": "block_section_name" },
        { "mData": "block_section_year_level" },
        { "mData": "semester_sem", 'mRender':
render_sem },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},
    ]
});
```

24. upmcts/static/js/departments/unit.js

```
var render_update = function(data, type, row) {

    return '<button data-url="'+
            Django.url('departments:unit_update',
data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+
            Django.url('departments:unit_delete',
data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#unit-table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('departments:DT_unit_objects'),
    "aoColumns": [
        { "mData": "abbrev" },
        { "mData": "name" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

25. upmcts/static/js/departments/unit_admin.js

```
var render_update_details = function(data,
type, row) {

    return '<button data-url="'+

Django.url('departments:unit_admin_update',
data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_update_password = function(data,
type, row) {

    return '<button data-url="'+
            Django.url('users:password_update',
data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-key"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('departments:unit_admin_delete',
data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#unit-admin-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('departments:DT_unit_admin_objec
ts'),
    "aoColumns": [
        { "mData": "username" },
        { "mData": "unit" },
        { "mData": "email" },
        { "mData": "first_name" },
        { "mData": "last_name" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update_details,  "sClass":
"column-edit" },
        { "mData": "user_pk", "bSortable": false,
'mRender': render_update_password,  "sClass":
"column-edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},

    ]
});
```

26. upmcts/static/js/units/class_constaint.js

```
popupInitializeFunction = function() {
    $('#id_classes').select2_sortable();

    var constraint = $('#id_constraint').val();
    if (constraint in constraint_help_text_list) {

$('#hint_id_constraint').text(constraint_help_te
xt_list[constraint]);
```

```
    }
};

popupSubmitFunction = function() {

$("#id_class_order").val($("#id_classes").val())
;
};

var render_constraint = function(data, type,
row) {
    return constraint_choice_list[data];
};

var render_preference_level = function(data,
type, row) {
    return preference_level_list[data];
};

var render_dept_unit = function(data, type,
row) {
    var str = data;
    if (row.unit) {
        str += ' (' + row.unit + ')';
    }
    return str
};

var render_update = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:class_constraint_update',
data) + unit_parameter +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:class_constraint_delete', data)
+ unit_parameter +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#class-constraint-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_class_constraint_objects',
{semester_pk: semester_pk}) + unit_parameter,
    "aoColumns": [
        { "mData": "constraint", 'mRender':
render_constraint },
        { "mData": "classes_display" },
        { "mData": "preference_level", 'mRender':
render_preference_level },
        { "mData": "department", 'mRender':
render_dept_unit },
        { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender': render_update,
"sClass": "column-edit" },
        { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender': render_delete ,
"sClass": "column-delete"}
    ]
});

(function($){

    $(document).on('change', '#id_constraint',
function() {
        var constraint = $(this).val();
        if (constraint in constraint_help_text_list)
{

$('#hint_id_constraint').text(constraint_help_te
xt_list[constraint]);
        } else {
            $('#hint_id_constraint').text('');
        }
    });

    $.fn.extend({
        select2_sortable: function(){
            var select = $(this);
            $(select).select2({
                width: '100%',
                createTag: function(params) {
                    return undefined;
                }
            });
            var ul = $(select).next('.select2-
container').first('ul.select2-
selection__rendered');
            ul.sortable({
                placeholder : 'ui-state-highlight',
                forcePlaceholderSize: true,
                items       : 'li:not(.select2-
search__field)',
                tolerance   : 'pointer',
                stop: function() {
                    $($(ul).find('.select2-
selection__choice').get().reverse()).each(functio
n() {
                        var id = $(this).data('data').id;
                        var option =
select.find('option[value="' + id + '"]')[0];
                        $(select).prepend(option);
                    });
            }
          });
        }
      });

    }(jQuery));
```

27. upmcts/static/js/units/class_details.js

```
var popupSubmitSuccessFunction =
function(pk) {
    if (pk) {
        var slots = $('#id_slots').val();
        if ($('#id_slots').length > 0) {
            $('#' + pk).text($('#id_slots').val());
            if (pk.toString().indexOf('slots') == 0) {
                room_pk = pk.replace("slots",
"room");
                datatable_list[room_pk].fnDraw();
            }
        }
    }
};

var render_commited = function(data, type,
row) {
    if (row.commited) {
        return '<i class="fa fa-check success
center"></i>';
    } else {
        return '<i class="fa fa-times danger
center"></i>';
    }
};


var render_preference_level = function(data,
type, row) {
    return preference_level_list[data];
};

var render_days = function(data, type, row) {
    days = '';
    days = (row.mondays) ? days + 'M': days;
    days = (row.tuesdays) ? days + 'T': days;
    days = (row.wednesdays) ? days + 'W': days;
    days = (row.thursdays) ? days + 'Th': days;
    days = (row.fridays) ? days + 'F': days;
    days = (row.saturdays) ? days + 'S': days;
    return days;
};

var render_hours = function(data, type, row) {
    // 2016-01-01 is irrelevant. we just want the
time
    var start = new moment('2016-01-01T' +
row.start);
    var end = new moment('2016-01-01T' +
row.end);
    return start.format('LT') + '-' +
end.format('LT');
};


var render_room_update = function(data, type,
row) {

    return '<button data-url="'+

Django.url('units:preferred_room_update', data)
+
        '" class="popup-button btn btn-default"
data-pk="room-' + row.class_pk + '"><i
class="fa fa-pencil"></i></button>';
};

var render_room_delete = function(data, type,
row) {

    return '<button data-url="'+

Django.url('units:preferred_room_delete', data)
+
        '" class="popup-button btn btn-danger"
data-pk="room-' + row.class_pk + '"><i
class="fa fa-trash"></i></button>';
};

var render_time_update = function(data, type,
row) {

    return '<button data-url="'+

Django.url('units:preferred_time_update', data)
+
        '" class="popup-button btn btn-default"
data-pk="time-' + row.class_pk + '"><i
class="fa fa-pencil"></i></button>';
};

var render_time_delete = function(data, type,
row) {

    return '<button data-url="'+

Django.url('units:preferred_time_delete', data)
+
        '" class="popup-button btn btn-danger"
data-pk="time-' + row.class_pk + '"><i
class="fa fa-trash"></i></button>';
};

var render_instructor_name = function(data,
type, row) {

    return row.first_name + ' ' +
row.middle_initials + ' ' + row.last_name;
};

var render_instructor_delete = function(data,
type, row) {
   return '<button data-url="'+

Django.url('units:class_instructor_delete',
row.class_pk, row.pk) +
         '" class="popup-button btn btn-danger"
data-pk="instructor-' + row.class_pk + '"><i
class="fa fa-trash"></i></button>';
};

var datatable_list = [];

for (var i = 0; i < classPKList.length; i++) {
   datatable_list['room-' + classPKList[i]] =
$('#preferred-room-table-' +
classPKList[i]).dataTable({
      "bPaginate": true,
      "sPaginationType": "bootstrap",
      "bProcessing": true,
      "bServerSide": true,
      "sAjaxSource":
Django.url('units:DT_class_room_preference_o
bjects', {class_pk: classPKList[i]}),
      "aoColumns": [
         { "mData": "abbrev" },
         { "mData": "name" },
         { "mData": "capacity" },
         { "mData": "preference_level",
'mRender': render_preference_level },
         { "mData": "commited", 'mRender':
render_commited,  "sClass": "column-edit" },
         { "mData": "pk", "bSortable": false,
'mRender': render_room_update ,  "sClass":
"column-edit"},
         { "mData": "pk", "bSortable": false,
'mRender': render_room_delete ,  "sClass":
"column-delete"},
      ]
   });

   datatable_list['time-' + classPKList[i]] =
$('#preferred-time-table-' +
classPKList[i]).dataTable({
      "bPaginate": true,
      "sPaginationType": "bootstrap",
      "bProcessing": true,
      "bServerSide": true,
      "sAjaxSource":
Django.url('units:DT_class_time_preference_o
bjects', {class_pk: classPKList[i]}),
      "aoColumns": [
         { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender': render_days },
         { "mData": "start", "bSortable": false,
'mRender': render_hours },
         { "mData": "preference_level",
'mRender': render_preference_level },
         { "mData": "commited", 'mRender':
render_commited,  "sClass": "column-edit" },
         { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender':
render_time_update ,  "sClass": "column-
edit"},
         { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender':
render_time_delete ,  "sClass": "column-
delete"},
      ]
   });

   datatable_list['instructor-' + classPKList[i]] =
$('#instructor-table-' +
classPKList[i]).dataTable({
      "bPaginate": true,
      "sPaginationType": "bootstrap",
      "bProcessing": true,
      "bServerSide": true,
      "sAjaxSource":
Django.url('units:DT_class_instructor_objects',
{class_pk: classPKList[i]}),
      "aoColumns": [
         { "mData": "last_name", "bSortable":
false, 'mRender': render_instructor_name },
         { "mData": "faculty_code" },
         { "mData": "email" },
         { "mData": "pk",  "bSearchable": false,
"bSortable": false, 'mRender':
render_instructor_delete ,  "sClass": "column-
delete"},
      ]
   });
};
```

28. upmcts/static/js/units/class_list.js

```
popupInitializeFunction = function() {

   var course = $('#id_course').val();
   if (course &&
classes_with_lab.indexOf(parseInt(course)) > -
1) {
      $('#div_id_lab_count').slideDown();
   } else {
      $('#div_id_lab_count').slideUp();
   }
};


var render_title = function(data, type, row) {
   var title = row.name;
   if (row.additional_name) {
      title += ' (' + row.additional_name + ')';
   }
```

```
    title += ' - ' + data;
    return title;
};

var render_update = function(data, type, row) {
    return '<a href="' +
Django.url('units:class_details', data) +
          '" class="btn btn-default"><i class="fa
fa-pencil"></i></a>'
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+
          Django.url('units:class_delete', data)+
unit_parameter +
          '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#class-table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_class_objects',
{semester_pk: semester_pk})+ unit_parameter,
    "aoColumns": [
        { "mData": "title" , 'mRender':
render_title},
        { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender': render_update,
"sClass": "column-edit" },
        { "mData": "pk", "bSearchable": false,
"bSortable": false, 'mRender': render_delete ,
"sClass": "column-delete"},


    ]
});

$(document).ready(function() {
    $(document).on('change', '#id_course',
function() {
        var course = $(this).val();
        if (course &&
classes_with_lab.indexOf(parseInt(course)) > -
1) {
            $('#div_id_lab_count').slideDown();
        } else {
            $('#div_id_lab_count').slideUp();
        }
    });
});
```

29. upmcts/static/js/units/course.js

```
popupInitializeFunction = function() {
    var has_lab =
$("#id_has_lab").is(':checked');
    if (!has_lab) {
        $('#div_id_lab_units').hide();
        $('#div_id_lab_duration').hide();
        $('#div_id_required_lab_features').hide();
    }
};


var hide_lab_fields = function(data, type, row)
{
    return data + 'h';
};

var render_duration = function(data, type, row)
{
    return data + 'h';
};


var render_lab = function(data, type, row) {
    if (row.has_lab) {
        return data + 'h';
    } else {
        return '<i class="fa fa-ban danger
center"></i>';
    }
};

var render_update = function(data, type, row) {

    return '<button data-url="'+
          Django.url('units:course_update', data)
+ unit_parameter +
          '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+
          Django.url('units:course_delete', data) +
unit_parameter +
          '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#course-table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_course_objects') +
unit_parameter,
    "aoColumns": [
        { "mData": "name" },
        { "mData": "title" },
        { "mData": "units" },
        { "mData": "lec_duration", "bSortable": 
false, 'mRender': render_duration },
        { "mData": "lab_duration", "bSortable": 
false, 'mRender': render_lab },
        { "mData": "pk", "bSearchable": false, 
"bSortable": false, 'mRender': render_update, 
"sClass": "column-edit" },
        { "mData": "pk", "bSearchable": false, 
"bSortable": false, 'mRender': render_delete , 
"sClass": "column-delete"},


    ]
});

$(document).ready(function() {
    $(document).on('change', '#id_has_lab', 
function() {
        var has_lab = 
$("#id_has_lab").is(':checked');
        if (!has_lab) {
            $('#div_id_lab_units').slideUp();
            $('#div_id_lab_duration').slideUp();

$('#div_id_required_lab_features').slideUp();
        } else {
            $('#div_id_lab_units').slideDown();
            $('#div_id_lab_duration').slideDown();

$('#div_id_required_lab_features').slideDown()
;
        }
    });
});
```

30. upmcts/static/js/units/instructor.js

```
var render_name = function(data, type, row) {
    return row.first_name + ' ' + 
row.middle_initials + '. ' + row.last_name
};

var render_update = function(data, type, row) {

    return '<button data-url="'+
            Django.url('units:instructor_update', 
data) + unit_parameter +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_update_timeslots = function(data, 
type, row) {

    return '<a href="' + 
Django.url('units:instructor_prohibited_time_sl
ot_list', data) +
            '" class="btn btn-default"><i class="fa 
fa-pencil"></i></a>'
};


var render_delete = function(data, type, row) {

    return '<button data-url="'+
            Django.url('units:instructor_delete', 
data) + unit_parameter +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#instructor-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource": 
Django.url('units:DT_instructor_objects') + 
unit_parameter,
    "aoColumns": [
        { "mData": "last_name", 'mRender': 
render_name },
        { "mData": "faculty_code" },
        { "mData": "email" },
        { "mData": "pk", "bSearchable": false, 
"bSortable": false, 'mRender': render_update, 
"sClass": "column-edit" },
        { "mData": "pk", "bSearchable": false, 
"bSortable": false, 'mRender': 
render_update_timeslots,  "sClass": "column-
edit" },
        { "mData": "pk", "bSearchable": false, 
"bSortable": false, 'mRender': render_delete , 
"sClass": "column-delete"},


    ]
});
```

31. upmcts/static/js/units/instructor_prohibited_tim
e_slot.js

```
var render_days = function(data, type, row) {
    days = '';
    days = (row.mondays) ? days + 'M': days;
    days = (row.tuesdays) ? days + 'T': days;
    days = (row.wednesdays) ? days + 'W': days;
    days = (row.thursdays) ? days + 'Th': days;
    days = (row.fridays) ? days + 'F': days;
    days = (row.saturdays) ? days + 'S': days;
    return days;
};
```

```
var render_hours = function(data, type, row) {
    // 2016-01-01 is irrelevant. we just want the
time
    var start = new moment('2016-01-01T' +
row.start);
    var end = new moment('2016-01-01T' +
row.end);
    return start.format('LT') + '-' +
end.format('LT');
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:instructor_prohibited_time_sl
ot_delete', data)+ unit_parameter +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#instructor-prohibited-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_instructor_prohibited_tim
e_slot_objects', {pk: instructor_pk,
semester_pk: semester_pk}) + unit_parameter,
    "aoColumns": [
        { "mData": "pk", "bSortable": false,
'mRender': render_days },
        { "mData": "start", "bSortable": false,
'mRender': render_hours },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

32. upmcts/static/js/units/room_group_details.js

```
var render_preference_level = function(data,
type, row) {
    return preference_level_list[data];
};

var render_update = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:room_group_member_update'
, data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:room_group_member_delete',
data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#room-group-member-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_room_group_member_ob
jects', {room_group_pk: room_group_pk}),
    "aoColumns": [
        { "mData": "abbrev" },
        { "mData": "name" },
        { "mData": "capacity" },
        { "mData": "preference_level", 'mRender':
render_preference_level },
        { "mData": "pk", "bSortable": false,
'mRender': render_update ,  "sClass": "column-
edit"},
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},
    ]
});
```

33. upmcts/static/js/units/room_group_list.js

```
var render_update = function(data, type, row) {
    return '<button data-url="'+
            Django.url('units:room_group_update',
data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button>';
};

var render_update_rooms = function(data, type,
row) {
    return '<a href="' +
Django.url('units:room_group_details', data) +
           '" class="btn btn-default"><i class="fa
fa-pencil"></i></a>'
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+
            Django.url('units:room_group_delete',
data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button';
};

var datatable = $('#room-group-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_room_group_objects'),
    "aoColumns": [
        { "mData": "name" },
        { "mData": "description" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update_rooms,  "sClass":
"column-edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

34. upmcts/static/js/units/time_slot_group_details.js

```
var render_days = function(data, type, row) {
    days = '';
    days = (row.mondays) ? days + 'M': days;
    days = (row.tuesdays) ? days + 'T': days;
    days = (row.wednesdays) ? days + 'W': days;
    days = (row.thursdays) ? days + 'Th': days;
    days = (row.fridays) ? days + 'F': days;
    days = (row.saturdays) ? days + 'S': days;
    return days;
};

var render_hours = function(data, type, row) {
    // 2016-01-01 is irrelevant. we just want the
time
    var start = new moment('2016-01-01T' +
row.start);
    var end = new moment('2016-01-01T' +
row.end);
    return start.format('LT') + '-' +
end.format('LT');
};

var render_preference_level = function(data,
type, row) {
    return preference_level_list[data];
};

var render_update = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:time_slot_group_member_up
date', data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:time_slot_group_member_del
ete', data) +
            '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button';
};

var datatable = $('#time-slot-group-member-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_time_slot_group_membe
r_objects', {time_slot_group_pk:
time_slot_group_pk}),
    "aoColumns": [
        { "mData": "pk", "bSortable": false,
'mRender': render_days },
          { "mData": "start", "bSortable": false,
'mRender': render_hours },
        { "mData": "preference_level", 'mRender':
render_preference_level },
        { "mData": "pk", "bSortable": false,
'mRender': render_update ,  "sClass": "column-
edit"},
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},
    ]
});
```

35. upmcts/static/js/units/time_slot_group_list.js

```
var render_update = function(data, type, row) {
    return '<button data-url="'+

Django.url('units:time_slot_group_update',
data) +
            '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button';
};
```

```
var render_update_rooms = function(data, type, row) {
    return '<a href="' +
Django.url('units:time_slot_group_details', data) +
          '" class="btn btn-default"><i class="fa fa-pencil"></i></a>'
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('units:time_slot_group_delete', data) +
           '" class="popup-button btn btn-danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#time-slot-group-table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('units:DT_time_slot_group_objects'),
    "aoColumns": [
        { "mData": "name" },
        { "mData": "description" },
        { "mData": "time_slot_duration" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update_rooms,  "sClass": "column-edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-delete"},


    ]
});
```

36. upmcts/static/js/university/college.js

```
var render_update = function(data, type, row) {

    return '<button data-url="'+
           Django.url('university:college_update', data) +
           '" class="popup-button btn btn-default"><i class="fa fa-pencil"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+
           Django.url('university:college_delete', data) +
           '" class="popup-button btn btn-danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#college-table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('university:DT_college_objects'),
    "aoColumns": [
        { "mData": "abbrev" },
        { "mData": "name" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-delete"},


    ]
});
```

37. upmcts/static/js/university/college_admin.js

```
var render_update_details = function(data, type, row) {

    return '<button data-url="'+

Django.url('university:college_admin_update', data) +
           '" class="popup-button btn btn-default"><i class="fa fa-pencil"></i></button>';
};

var render_update_password = function(data, type, row) {

    return '<button data-url="'+
           Django.url('users:password_update', data) +
           '" class="popup-button btn btn-default"><i class="fa fa-key"></i></button>';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+

Django.url('university:college_admin_delete', data) +
        '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button>';
};

var datatable = $('#college-admin-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('university:DT_college_admin_obje
cts'),
    "aoColumns": [
        { "mData": "username" },
        { "mData": "college" },
        { "mData": "email" },
        { "mData": "first_name" },
        { "mData": "last_name" },
        { "mData": "pk", "bSortable": false,
'mRender': render_update_details,  "sClass":
"column-edit" },
        { "mData": "user_pk", "bSortable": false,
'mRender': render_update_password,  "sClass":
"column-edit" },
        { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},


    ]
});
```

38. upmcts/static/js/university/college_timetable_list.js

```
var render_sem = function(data, type, row) {
    return sem_list[data] + ', ' +
row.semester_academic_year;
};

var render_status = function(data, type, row) {
    return status_list[data];
};

var render_view = function(data, type, row) {
    if (row.status == 'S') {
        return '<a href="' +
Django.url('university:college_timetable_outpu
t', data) +
            '" class="btn btn-default"><i class="fa
fa-list"></i></a>';
    }
    return '-';
};

var pollTimetables = function() {
    setTimeout(function(){
        $('#college-timetable-
table_processing').css('color', '#FFF');
        datatable.fnDraw();
        pollTimetables();
    }, 5000);
};

popupSubmitSuccessFunction = function() {
    $('#college-timetable-
table_processing').css('color', '#373A3C');
};

var datatable = $('#college-timetable-
table').dataTable({
    "bPaginate": true,
    "sPaginationType": "bootstrap",
    "bProcessing": true,
    "bServerSide": true,
    "sAjaxSource":
Django.url('university:DT_college_timetable_o
bjects', {college_pk: college_pk} ),
    "aoColumns": [
        { "mData": "semester_sem", 'mRender':
render_sem},
        { "mData": "status", 'mRender':
render_status},
        { "mData": "assigned_classes_count" },
        { "mData": "unassigned_classes_count" },
        { "mData": "pk", "bSortable": false,
'mRender': render_view,  "sClass": "column-
edit" },

    ]
});

$(document).ready(function() {
    pollTimetables();
});
```

39. upmcts/static/js/university/semester.js

```
var render_update = function(data, type, row) {

    return '<button data-url="'+
        Django.url('university:semester_update',
data) +
        '" class="popup-button btn btn-
default"><i class="fa fa-pencil"></i></button';
};

var render_delete = function(data, type, row) {

    return '<button data-url="'+
        Django.url('university:semester_delete',
data) +
        '" class="popup-button btn btn-
danger"><i class="fa fa-trash"></i></button';
};
```

```
var datatable = $('#semester-table').dataTable({
   "bPaginate": true,
   "sPaginationType": "bootstrap",
   "bProcessing": true,
   "bServerSide": true,
   "sAjaxSource":
Django.url('university:DT_semester_objects'),
   "aoColumns": [
      { "mData": "sem" },
      { "mData": "pk", "bSortable": false,
'mRender': render_update,  "sClass": "column-
edit" },
      { "mData": "pk", "bSortable": false,
'mRender': render_delete ,  "sClass": "column-
delete"},
   ]
});
```

40. upmcts/static/js/university/university_timetable.js

```
var render_link = function(data, type, row) {
   return '<a href="' +
Django.url('university:college_timetable_list',
data) +
         '" class="btn btn-default"><i class="fa
fa-list"></i></a>';
};

var datatable = $('#university-timetable-
table').dataTable({
   "bPaginate": true,
   "sPaginationType": "bootstrap",
   "bProcessing": true,
   "bServerSide": true,
   "sAjaxSource":
Django.url('university:DT_college_objects'),
   "aoColumns": [
      { "mData": "abbrev" },
      { "mData": "name" },
      { "mData": "pk", "bSortable": false,
'mRender': render_link,  "sClass": "column-
edit" },
   ]
});
```

41. upmcts/static/js/project.js

```
var popupInitializeFunction = function() {};
var popupSubmitFunction = function() {};
var popupSubmitSuccessFunction =
function(pk) {};

/* Project specific Javascript goes here. */
(function($, Django, UPMCTS){


   "use strict";

   $.noty.defaults = {
      layout: 'bottomLeft',
      theme: 'relax', // or 'relax'
      dismissQueue: true, // If you want to use
queue feature set this true
      template: '<div
class="noty_message"><span
class="noty_text"></span><div
class="noty_close"></div></div>',
      animation: {
         open: {height: 'toggle'}, // or
Animate.css class names like: 'animated
bounceInLeft'
         close: {height: 'toggle'}, // or
Animate.css class names like: 'animated
bounceOutLeft'
         easing: 'swing',
         speed: 500 // opening & closing
animation speed
      },
      timeout: 3000, // delay for closing event.
Set false for sticky notifications
      force: false, // adds notification to the
beginning of queue when set to true
      modal: false,
      maxVisible: 5, // you can set max visible
notification for dismissQueue true option,
      killer: false, // for close all notifications
before show
      closeWith: ['click'], // ['click', 'button',
'hover', 'backdrop'] // backdrop click will close
all notifications
      callback: {
         onShow: function() {},
         afterShow: function() {},
         onClose: function() {},
         afterClose: function() {},
         onCloseClick: function() {},
      },
      buttons: false // an array of buttons
   };

   var formAjaxSubmit = function(form,
modal, pk) {
      var $submit_btn = $('.submit');
      $(form).submit(function (e) {
         $submit_btn.button('loading');
         e.preventDefault();
         popupSubmitFunction();
         $.ajax({
            type: $(this).attr('method'),
            url: $(this).attr('action'),
            data: $(this).serialize(),
            success: function (xhr, ajaxOptions,
thrownError) {
               if ( $(xhr).find('.has-error').length
> 0 || $(xhr).find('.alert-danger').length > 0) {
                  $(modal).find('#popup-
content').html(xhr);
```

```
                    pk = $('.modal-form').data('pk');
                    reinitializeElements();
                    formAjaxSubmit(form, modal,
pk);
                } else {

popupSubmitSuccessFunction(pk);
                    $(modal).modal('toggle');
                    if (pk != null && pk in
datatable_list) {
                        datatable_list[pk].fnDraw();
                    } else if (typeof datatable !==
"undefined"){
                        datatable.fnDraw();
                    }
                    noty({
                      text: 'Success!',
                      type: 'success'
                    });
                }
                $submit_btn.button('reset');
            },
            error: function (xhr, ajaxOptions,
thrownError) {
                $submit_btn.button('reset');
            }
        });
    });
  }

   var reinitializeElements = function() {
     $(".timeinput").pickatime({
        format: 'h:i A',
        interval: 15,
        min: new Date(2015,3,20,7),
        max: new Date(2015,7,14,21)
     });
     $('.select2').css('width', '100%');
     popupInitializeFunction();
   }

  $(function(){



     $(document).on('click', '.popup-button',
function() {
          var url = $(this).data('url');
          var pk = $(this).data('pk');
          $('.modal-content').load(url, function ()
{
               $('#popup').modal('toggle');
               formAjaxSubmit('#popup-content
form', '#popup', pk);
               reinitializeElements();
               // Fix bug where select2 widget does
not have full width
          });
        });
     });


}(window.jQuery, window.Django,
window.UPMCTS));
```

42. upmcts/taskapp/celery.py

```
from __future__ import absolute_import
import os
import shutil
from celery import Celery
from django.apps import AppConfig
from django.conf import settings
from upmcts.colleges.models import Timetable
from upmcts.utils.utils import
generate_timetabling_input_data,
generate_timetabling_output_data

if not settings.configured:
    # set the default Django settings module for
the 'celery' program.

os.environ.setdefault("DJANGO_SETTINGS_
MODULE", "config.settings.local")  # pragma:
no cover


app = Celery('upmcts')


class CeleryConfig(AppConfig):
    name = 'upmcts.taskapp'
    verbose_name = 'Celery Config'

    def ready(self):
        # Using a string here means the worker
will not have to
        # pickle the object when using Windows.

app.config_from_object('django.conf:settings')
        app.autodiscover_tasks(lambda:
settings.INSTALLED_APPS, force=True)

        if hasattr(settings, 'RAVEN_CONFIG'):
            # Celery signal registration
            from raven import Client
            from raven.contrib.celery import
register_signal
            client =
Client(dsn=settings.RAVEN_CONFIG['dsn'])
            register_signal(client)


@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))  #
pragma: no cover


@app.task(name='generate_timetable')
def generate_timetable(timetable_orig,
timeout=20):
    try:
        timetable =
Timetable.objects.get(pk=timetable_orig.pk)
        if not timetable.status ==
Timetable.QUEUED:
            timetable.status = Timetable.QUEUED
            timetable.save()
        newline = os.linesep  # Defines the
newline based on your OS.

        source_fp =
open(settings.TIMETABLING_SOLVER_RO
OT + '/base.cfg', 'r')
        target_fp =
open(settings.TIMETABLING_SOLVER_RO
OT + '/config.cfg', 'w')
        source_fp.readline()  # and discard
        timeout_settings =
'Termination.TimeOut=' + str(timeout) +
newline
        target_fp.write(timeout_settings)
        shutil.copyfileobj(source_fp, target_fp)

generate_timetabling_input_data(timetable)

generate_timetabling_output_data(timetable)
    except Exception as e:
        timetable.status = Timetable.FAILED
        timetable.save()
```

43. upmcts/taskapp/task.py

```
from __future__ import absolute_import

from upmcts.taskapp.celery import app
from upmcts.colleges.models import Timetable
from upmcts.utils.utils import
generate_timetabling_input_data,
generate_timetabling_output_data


@app.task(name='generate_timetable')
def generate_timetable(timetable_orig):
    try:
        timetable =
Timetable.objects.get(pk=timetable_orig.pk)
        if not timetable.status ==
Timetable.QUEUED:
            timetable.status = Timetable.QUEUED
            timetable.save()

generate_timetabling_input_data(timetable)

generate_timetabling_output_data(timetable)
    except Exception as e:
        timetable.status = Timetable.FAILED
        timetable.save()
```

44. upmcts/templates/colleges/department_admin_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

  <div class="col-sm-10 col-sm-push-1">
    <h2>{% trans "Department Chairs - " %}
{{ college }}</h2>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'colleges:department_admin_create' %}">{%
trans "Add New Department Chair"
%}</button>
      </div>
    </div>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <table id="department-admin-table"
class="display table table-bordered table-
striped">
          <thead>
            <tr>
              <th class="col-md-2">{%
trans "Username" %}</th>
              <th class="col-md-1">{%
trans "Department" %}</th>
              <th class="col-md-2">{%
trans "Email" %}</th>
              <th class="col-md-2">{%
trans "First Name" %}</th>
              <th class="col-md-2">{%
trans "Last Name" %}</th>
              <th class="col-md-1 column-
edit">{% trans "Details" %}</th>
              <th class="col-md-1 column-
edit">{% trans "Password" %}</th>
              <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>
    </div>
    </div>
</div>

{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {% js "js/colleges/department_admin.js" %}
{% endblock javascript %}
```

45. upmcts/templates/colleges/department_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Departments - " %} {{
college }}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'colleges:department_create' %}">{% trans
"Add New Department" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="department-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-2"></th>
                            <th class="col-md-8">{%
trans "Department" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>

        </div>

    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js "js/colleges/department.js" %}
{% endblock javascript %}
```

46. upmcts/templates/colleges/room_feature_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables  i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Room Features - " %} {{
college }}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-10">
                <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'colleges:room_feature_create' %}">{% trans
"Add New Room Feature" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="room_feature-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-10">{%
trans "Room Feature" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>
    </div>
</div>

{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {% js "js/colleges/room_feature.js" %}
{% endblock javascript %}
```

47. upmcts/templates/colleges/room_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Rooms - " %} {{ college
}}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'colleges:room_create' %}">{% trans "Add
New Room" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="room-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-2"></th>
                            <th class="col-md-6">{%
trans "Room" %}</th>
                            <th class="col-md-1">{%
trans "Capacity" %}</th>
                            <th class="col-md-1">{%
trans "Available" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>
    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js "js/colleges/room.js" %}
{% endblock javascript %}
```

48. upmcts/templates/colleges/timetable_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Generate Timetables - "
%} {{ college }}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'colleges:timetable_create' %}">{% trans
"Generate New Timetable" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="timetable-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-5">{%
trans "Semester" %}</th>
                            <th class="col-md-3">{%
trans "Status" %}</th>
                            <th class="col-md-1">{%
trans "Assigned Classes" %}</th>
                            <th class="col-md-1">{%
trans "Unassigned Classes" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "View Output" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Regenerate" %}</th>
```

```
                </tr>
              </thead>
              <tbody></tbody>
            </table>
          </div>

        </div>

      </div>
    </div>
    {% endblock content %}

    {% block javascript %}
      {{ block.super }}
      {% django_js %}
      {% datatables_js %}
      {% datatables_bootstrap_js %}

      {# Reinitialize to remove interference of
    django_js with select2 #}
      <script src="{% static
    'bower_components/select2/dist/js/select2.js'
    %}"></script>

      <script type="text/javascript">
        var sem_list = {{sem_list|safe}};
        var status_list = {{status_list|safe}};
      </script>

      {% js "js/colleges/timetable.js" %}
    {% endblock javascript %}
```

49. upmcts/templates/colleges/timetable_output.ht
ml

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
django_tables2 eztables i18n %}

{% block content %}
<div class="row margin-bottom-20">
  <div class="input-group col-xs-10 col-xs-
push-1">
    <h2>{% trans "Timetable " %}
({{timetable.semester}}) -
{{timetable.college}} </h2>
    <form method="get" action='{{
request.path }}' class="margin-top-20">
      <span class="col-sm-6">
        {% crispy department_form %}
      </span class="col-sm-6">
      <span class="col-sm-6">
        <button type="submit" class="btn
btn-primary btn-sm">{% trans "Select
Department" %}</button>
      </span>
    </form>
  </div>
</div>

<div class="row">
  <div class="col-sm-10 col-sm-push-1">
    {% render_table table %}
  </div>
</div>
{% endblock content %}
```

50. upmcts/templates/departments/block_section_c
lass_details.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

  <div class="col-sm-10 col-sm-push-1">

<h2>{{block_section_classes.semester}}{%
trans " Classes - Block " %} {{
block_section_classes.block_section }}</h2>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'departments:block_section_class_add'
block_section_classes.pk %}">{% trans "Add
New Class" %}</button>
      </div>
    </div>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <table id="class-table" class="display
table table-bordered table-striped">
          <thead>
            <tr>
              <th class="col-md-10">{%
trans "Class" %}</th>
              <th class="col-md-1 column-
delete">{% trans "View Details" %}</th>
              <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>
    </div>
  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
```

```
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    <script type="text/javascript">
    var section_classes_pk =
{{block_section_classes.pk|safe}};
    </script>

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js
"js/departments/block_section_class_details.js"
%}
{% endblock javascript %}
```

51. upmcts/templates/departments/block_section_class_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Block Sections Classes - "
%} {{ department }}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary" data-url="{% url
'departments:block_section_class_create'
%}">{% trans "Add Block Section to
Semester" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="block_section_classes-
table" class="display table table-bordered
table-striped">
                    <thead>
                        <tr>
                            <th class="col-md-5">{%
trans "Block Name" %}</th>
                            <th class="col-md-2">{%
trans "Year Level" %}</th>
                            <th class="col-md-2">{%
trans "Semester" %}</th>
                            <th class="col-md-2 column-
edit">{% trans "Edit Classes" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>

    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>
    <script type="text/javascript">
    var sem_list = {{sem_list|safe}};
    </script>
    {% js
"js/departments/block_section_class_list.js" %}
{% endblock javascript %}
```

52. upmcts/templates/departments/block_section_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Block Sections - " %} {{
department }}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'departments:block_section_create' %}"
                        >{% trans "Add New Block
Section" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="block_section-table"
class="display table table-bordered table-
striped">
                    <thead>
                <tr>
                    <th class="col-md-6">{%
trans "Block Name/Number" %}</th>
                    <th class="col-md-4">{%
trans "Year Level" %}</th>
                    <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                    <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                </tr>
            </thead>
            <tbody></tbody>
        </table>
      </div>
    </div>
  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  {# Reinitialize to remove interference of
django_js with select2 #}
  <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

  {% js "js/departments/block_section.js" %}
{% endblock javascript %}
```

53. upmcts/templates/departments/unit_admin_list.
html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

  <div class="col-sm-10 col-sm-push-1">
    <h2>{% trans "Unit Heads - " %} {{
department }}</h2>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <button type="button" class="popup-
button btn btn-primary"
            data-url="{% url
'departments:unit_admin_create' %}"
            >{% trans "Add New Unit
Head" %}</button>
      </div>
    </div>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <table id="unit-admin-table"
class="display table table-bordered table-
striped">
          <thead>
            <tr>
              <th class="col-md-2">{%
trans "Username" %}</th>
              <th class="col-md-1">{%
trans "Unit" %}</th>
              <th class="col-md-2">{%
trans "Email" %}</th>
              <th class="col-md-2">{%
trans "First Name" %}</th>
              <th class="col-md-2">{%
trans "Last Name" %}</th>
              <th class="col-md-1 column-
edit">{% trans "Details" %}</th>
              <th class="col-md-1 column-
edit">{% trans "Password" %}</th>
              <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>

    </div>

  </div>
</div>

{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  {% js "js/departments/unit_admin.js" %}
{% endblock javascript %}
```

54. upmcts/templates/departments/unit_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

  <div class="col-sm-10 col-sm-push-1">
    <h2>{% trans "Units - " %} {{
department }}</h2>
    <div class="row margin-top-20">
      <div class="col-sm-12">
```

```
            <button type="button" class="popup-
button btn btn-primary"
                    data-url="{% url
'departments:unit_create' %}"
                    > {% trans "Add New Unit"
%}</button>
        </div>
    </div>
    <div class="row margin-top-20">
        <div class="col-sm-12">
            <table id="unit-table" class="display
table table-bordered table-striped">
                <thead>
                    <tr>
                        <th class="col-md-2"></th>
                        <th class="col-md-6">{%
trans "Unit" %}</th>
                        <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                        <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </div>
    </div>
  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  {# Reinitialize to remove interference of
django_js with select2 #}
  <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

  {% js "js/departments/unit.js" %}
{% endblock javascript %}
```

55. upmcts/templates/django_tables2/table.html

```
{% load querystring from django_tables2 %}
{% load trans blocktrans from i18n %}
{% load bootstrap3 %}

{% if table.page %}
  <div class="table-container">
{% endif %}

{# From:
https://gist.github.com/shayh/7291015 #}

{% block table %}
  <table class="display table table-bordered
table-striped"{% if table.attrs %} {{
table.attrs.as_html }}{% endif %}>
      {% block table.thead %}
        <thead>
        <tr>
          {% for column in table.columns %}
            {% if column.orderable %}
              <th {{ column.attrs.th.as_html
}}><a href="{% querystring
table.prefixed_order_by_field=column.order_b
y_alias.next %}">{{ column.header
}}</a></th>
            {% else %}
              <th {{ column.attrs.th.as_html
}}>{{ column.header }}</th>
            {% endif %}
          {% endfor %}
        </tr>
        </thead>
      {% endblock table.thead %}
      {% block table.tbody %}
        <tbody>
        {% for row in
table.page.object_list|default:table.rows %} {#
support pagination #}
          {% block table.tbody.row %}
            <tr class="{% cycle "odd" "even"
%}">
              {% for column, cell in row.items
%}
                <td {{
column.attrs.td.as_html }}>{{ cell }}</td>
              {% endfor %}
            </tr>
          {% endblock table.tbody.row %}
        {% empty %}
          {% if table.empty_text %}
            {% block table.tbody.empty_text
%}
              <tr><td colspan="{{
table.columns|length }}">{{ table.empty_text
}}</td></tr>
            {% endblock
table.tbody.empty_text %}
          {% endif %}
        {% endfor %}
        </tbody>
      {% endblock table.tbody %}
      {% block table.tfoot %}
        <tfoot></tfoot>
      {% endblock table.tfoot %}
  </table>
{% endblock table %}

{% if table.page %}
  {% block pagination %}
```

```
        {% bootstrap_pagination table.page
url=request.get_full_path %}
        {#{ table.page|pagination }#}
    {% endblock pagination %}
{% endif %}
```

56. upmcts/templates/django_tables2_reports/table.html

```
{% extends "django_tables2/table.html" %}

{% load i18n django_tables2 %}

{% block pagination %}
    {{ block.super }}
    {% block table.report %}
        {% if table.is_configured %}
            <form method="post" action='{% url
"university:college_timetable_pdf_output" %}'
style="display:inline;">
                {% csrf_token %}
                <input type="hidden"
name="timetable_pk" value="{{ timetable.pk
}}">
                {% if department %}
                    <input type="hidden"
name="department_pk" value="{{ department
}}">
                {% endif %}

                <div class="pull-right btn-group">
                    <button type="submit" class="btn
btn-link">
                        <i class="fa fa-file-pdf-o"
style="margin-right: 5px;" aria-
hidden="true"></i>{% trans "PDF Report" %}
                    </button>
                    {% for label, format in
table.formats %}
                        <a class="btn btn-link"
href="{% querystring
table.param_report=format %}">
                        {% if format == 'xls' %}
                            <i class="fa fa-file-excel-o"
aria-hidden="true"></i>
                        {% elif format == 'csv' %}
                            <i class="fa fa-file-text-o"
aria-hidden="true"></i>
                        {% else %}
                            <i class="fa fa-file-text" aria-
hidden="true"></i>
                        {% endif %}
                        {{ label }}
                        </a>
                    {% endfor %}
                </div>
            </form>
            <div class="clearfix"></div>
        {% endif %}
    {% endblock table.report %}
{% endblock %}
```

57. upmcts/templates/pages/home.html

```
{% extends "base.html" %}

{% load i18n %}

{% block home_nav %}active{% endblock
home_nav %}

{% block content %}

<div class="jumbotron">
        <h2>{% trans "Welcome to UPMCTS"
%}</h2>
        <p>{% trans "The UP Manila Course
Timetabling System is an application for
making college timetables for UP Manila "
%}</p>
</div>

{% endblock content %}
```

58. upmcts/templates/pdf/timetable.html

```
{% load i18n l10n %}

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>{{ title|default:"" }}</title>

  {% block style_base %}
    {% block layout_style %}
      <style type="text/css">

        @page {
          /* first page */
          size: {{ pagesize|default:"A4" }};
          @frame title {
        -pdf-frame-content:page-title;
          margin-left: auto;
          margin-right: 1cm;
          margin-top: 1cm;
          /* -pdf-frame-border: 1; */
          /* for debugging */
        }
        @frame contact {
        -pdf-frame-content:page-contact;
        margin-bottom: 5cm;
        margin-left: 1cm;
        margin-top: 1cm;
        margin-right: 1cm;
        }
        @frame hr {
        -pdf-frame-content:page-hr;
        margin-left: 1cm;
margin-right: 1cm;
top: {{hr_margin|safe}}cm;
}
@frame details {
-pdf-frame-content:page-details;
margin-left: 1cm;
margin-top: 3cm;
}
@frame content_frame {
margin-bottom: 2.5cm;
margin-left: 1cm;
margin-right: 1cm;
top: {{content_margin|safe}}cm;
}
@frame footer {
-pdf-frame-content:page-footer;
bottom: 0cm;
height: 2cm;
margin-left: 1cm;
margin-right: 1cm;
}
}

@page innerpages {
  /* second page onwards */
  size: {{ pagesize|default:"A4" }};
  @frame content_frame {
margin-bottom: 2.5cm;
  margin-left: 1cm;
  margin-right: 1cm;
  top: 1cm;
}
@frame footer {
-pdf-frame-content:page-footer;
bottom: 0cm;
height: 2cm;
margin-left: 1cm;
margin-right: 1cm;
}
}

body {
  font-family: "Open Sans", sans-serif;
}

.classes {
  border-collapse: collapse;
  border-spacing: 0;
  text-align: left;
}

.classes table, th, td {
  border: none;
}

.classes th, td {
  padding: 4px 4px 0px 4px;
  vertical-align: top;
}

.classes th {
  font-weight: bold;
  text-align: left;
  font-size: 12px;
  text-decoration: underline;
}

#info {
  border: none;
  width: 100%;

}

#info td {
  border: none;
  padding: 0;
  width: auto;
  text-align: left;
}

#info td:first-child {
  text-align: left;
  padding-right: 5px;
  width: 130px;
}

.align-left {
  text-align: left;
}

.align-right {
  text-align: right;
}

.align-center {
  text-align: center;
}

#title td {
  border: none;
}

.logo {
  height: 1cm;
}

h2.normal-weight {
  font-weight: normal;
}

#page-contact {
  font-size: 8px;
}

.amount-in-card {
  font-size: 8px;
```

```
          color: #959595;
        }
      </style>
    {% endblock %}
  {% endblock %}
</head>
<body>
<div>
  <div id="page-title">
    <table id="title">
      <tr>
        <td class="align-center">
          <h2>{% trans "SCHEDULE OF
CLASSES" %}</h2>
        </td>
      </tr>
      <tr>
        <td class="align-center">
          <h2 class="normal-weight">{% trans
"University of the Philippines Manila"
%}</h2>
        </td>
      </tr>
      <tr>
        <td class="align-center">
          <h2 class="normal-weight">{{
college.name }}</h2>
        </td>
      </tr>
      {% if department %}
        <tr>
          <td class="align-center">
            <h2 class="normal-weight">{{
department.name }}</h2>
          </td>
        </tr>
      {% endif %}
      <tr>
        <td class="align-center">
          <h2 class="normal-weight">{{ semester
}}</h2>
        </td>
      </tr>
    </table>
  </div>

  <div id="page-hr">
    <hr/>
  </div>

  <div id="page-footer">
    <p class="align-center">
      {% blocktrans %}
        Page
        <pdf:pagenumber/> /
        <pdf:pagecount/>
      {% endblocktrans %}
    </p>
  </div>

  <!-- Remove header on subsequent pages -->
  <pdf:nexttemplate name="innerpages"/>

  {% if class_list %}
    <table class="classes">
      <thead>
      <tr>
        <th>{% trans "Subject" %}</th>
        <th>{% trans "Days" %}</th>
        <th>{% trans "Timeslot" %}</th>
        <th>{% trans "Room" %}</th>
        <th>{% trans "Instructor/s" %}</th>
      </tr>
      </thead>
      <tbody>
      {% for class in class_list %}
        <tr>
          <td style="width: 25%">{{
class.subject|default:"---" }}</td>
          <td style="width: 10%">{{
class.assigned_time_slot.days|default:"---"
}}</td>
          <td style="width: 20%">{{
class.assigned_time_slot.hours|default:"---"
}}</td>
          <td style="width: 15%">{{
class.assigned_room.abbrev|default:"---"
}}</td>
          <td style="width: 30%">{{
class.instructor_list|default:"---" }}</td>
        </tr>
      {% endfor %}
      </tbody>
    </table>
  {% else %}
    <p class="align-center" style="color:
#808080;">
      <h3>{% trans "No classes to show"
%}</h3>
    </p>

  {% endif %}

</div>
</body>
</html>
```

59. upmcts/templates/units/class_constraint_list.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}
```

```
<div class="row margin-bottom-20">
   <div class="col-sm-10 col-sm-push-1">
      {% if unit and not unit.name == '-----' %}
         <h2>{% trans "Class Constraints - " %}
{{ unit }}</h2>
      {% else %}
         <h2>{% trans "Class Constraints"
%}</h2>
      {% endif %}
   </div>
</div>

<div class="row margin-bottom-30">
   <div class="input-group col-xs-10 col-xs-
push-1">
      <div class="row">
         <form method="get" action='{{
request.path }}'>
            <span class="col-xs-4">
               {% crispy semester_form %}
            </span>
            <span class="col-xs-4 margin-left-
10">
               <button type="submit" class="btn
btn-primary">{% trans "Select Semester"
%}</button>
            </span>
         </form>
      </div>

   </div>
</div>

<div class="row">

   <div class="col-sm-10 col-sm-push-1">
      <div class="row">
         <div class="col-sm-12">
            <button type="button" class="popup-
button btn btn-primary"
                  data-url="{% url
'units:class_constraint_create' semester.pk %}"
                  >{% trans "Add New
Constraint" %}</button>
         </div>
      </div>
      <div class="row margin-top-20">
         <div class="col-sm-12">
            <table id="class-constraint-table"
class="display table table-bordered table-
striped">
               <thead>
                  <tr>
                     <th class="col-md-2">{%
trans "Constraint" %}</th>
                     <th class="col-md-4">{%
trans "Classes" %}</th>
                     <th class="col-md-2">{%
trans "Preference Level" %}</th>
                     <th class="col-md-2">{%
trans "Set By" %}</th>
                     <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                     <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                  </tr>
               </thead>
               <tbody></tbody>
            </table>
         </div>
      </div>
   </div>
</div>
{% endblock content %}

{% block javascript %}
   {{ block.super }}
   {% django_js %}
   {% datatables_js %}
   {% datatables_bootstrap_js %}

   {# Reinitialize to remove interference of
django_js with select2 #}
   <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>
   <script src="{% static
'bower_components/jQuery UI Sortable/jquery-
ui-sortable.min.js' %}"></script>
   <script type="text/javascript">
      var constraint_choice_list =
{{constraint_choice_list|safe}};
      var preference_level_list =
{{preference_level_list|safe}};
      var constraint_help_text_list =
{{constraint_help_text_list|safe}};
      var semester_pk = {{semester.pk|safe}};
      var unit_parameter =
"{{unit_parameter}}";
   </script>
   {% js "js/units/class_constraint.js" %}
{% endblock javascript %}
```

60. upmcts/templates/units/class_details.html

```
{% extends "base.html" %}
{% load js staticfiles eztables i18n %}

{% block content %}

<ul class="nav nav-tabs">
   {% for class in  class_list %}
      <li {% if class == selected_class
%}class="active"{% endif %}>
         <a data-toggle="tab" href="#class-{{
class.pk }}">{{ class.additional_name }}</a>
```

```
        </li>
    {% endfor %}
</ul>

<div class="tab-content">
    {% for class in  class_list %}
        <div id="class-{{ class.pk }}"
             class="tab-pane fade {% if class ==
selected_class %}in active{% endif %}">
          <h3>{{ class.course.name }} {{
class.additional_name }}</h3>

          <div class="row">
            <div class="col-sm-10 col-sm-push-1">
                <span>
                    <h4>{% trans "Slots: " %} <span
id="slots-{{class.pk}}">{{ class.slots
}}</span></h4>
                </span>
                <span>
                    <button type="button"
class="popup-button btn btn-primary"
                          data-url="{% url
'units:class_slots_update' class.pk %}"
                          data-pk="slots-{{class.pk}}"
>{% trans "Update Slots" %}</button>
                </span>
            </div>
          </div>

          <div class="row margin-top-30">
              <div class="col-sm-10 col-sm-push-
1">
                <div class="row">
                    <div class="col-sm-12">
                        <button type="button"
class="popup-button btn btn-primary"
                              data-url="{% url
'units:class_instructor_create' class.pk %}"
                              data-pk="instructor-
{{class.pk}}"\
                              >{% trans "Add
Instructor" %}</button>
                    </div>
                </div>
                <div class="row margin-top-20">
                    <div class="col-sm-12">
                      <table id="instructor-table-{{
class.pk }}"
                             class="display table table-
bordered table-striped">
                          <thead>
                              <tr>
                                  <th class="col-md-
6">{% trans "Name" %}</th>
                                  <th class="col-md-
2">{% trans "Faculty Code" %}</th>
                                  <th class="col-md-
3">{% trans "Email" %}</th>
                                  <th class="col-md-1
column-delete">{% trans "Delete" %}</th>
                              </tr>
                          </thead>
                          <tbody></tbody>
                      </table>
                    </div>
                </div>
              </div>
          </div>

          <div class="row">
              <div class="col-sm-10 col-sm-push-
1">
                  <div class="row">
                      <div class="col-sm-12">
                          <button type="button"
class="popup-button btn btn-primary"
                                data-url="{% url
'units:preferred_room_create' class.pk %}"
                                data-pk="room-
{{class.pk}}"
                                >{% trans "Add Preferred
Room" %}
                          </button>

                          <button type="button"
class="popup-button btn btn-primary"
                                data-url="{% url
'units:preferred_room_group_create' class.pk
%}"
                                data-pk="room-
{{class.pk}}"
                                >{% trans "Add Preferred
Room Group" %}
                          </button>
                      </div>
                  </div>
                  <div class="row margin-top-20">
                      <div class="col-sm-12">
                        <table id="preferred-room-
table-{{ class.pk }}"
                               class="display table table-
bordered table-striped">
                            <thead>
                                <tr>
                                    <th class="col-md-
2"></th>
                                    <th class="col-md-
4">{% trans "Room" %}</th>
                                    <th class="col-md-
1">{% trans "Capacity" %}</th>
                                    <th class="col-md-
2">{% trans "Preference Level" %}</th>
                                    <th class="col-md-
1">{% trans "Commited?" %}</th>
```

```
                        <th class="col-md-
1">{% trans "Edit" %}</th>
                        <th class="col-md-
1">{% trans "Delete" %}</th>
                      </tr>
                    </thead>
                    <tbody></tbody>
                  </table>
                </div>
              </div>
            </div>
          </div>


          <div class="row">
            <div class="col-sm-10 col-sm-push-
1">
              <div class="row">
                <div class="col-sm-12">
                  <button type="button"
class="popup-button btn btn-primary"
                      data-url="{% url
'units:preferred_time_create' class.pk %}"
                      data-pk="time-
{{class.pk}}"
                      >{% trans "Add Preferred
Time" %}</button>
                  <button type="button"
class="popup-button btn btn-primary"
                      data-url="{% url
'units:preferred_time_slot_group_create'
class.pk %}"
                      data-pk="time-
{{class.pk}}"
                      >{% trans "Add Preferred
Time Group" %}
                  </button>
                </div>
              </div>
              <div class="row margin-top-20">
                <div class="col-sm-12">
                  <table id="preferred-time-table-
{{ class.pk }}"
                    class="display table table-
bordered table-striped">
                    <thead>
                      <tr>
                        <th class="col-md-
3">{% trans "Days" %}</th>
                        <th class="col-md-
4">{% trans "Hours" %}</th>
                        <th class="col-md-
2">{% trans "Preference Level" %}</th>
                        <th class="col-md-
1">{% trans "Commited" %}</th>
                        <th class="col-md-1
column-edit">{% trans "Edit" %}</th>
                        <th class="col-md-1
column-delete">{% trans "Delete" %}</th>
                      </tr>
                    </thead>
                    <tbody></tbody>
                  </table>
                </div>
              </div>
            </div>
          </div>
        </div>
  {% endfor %}
</div>

{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  {# Reinitialize to remove interference with
django_js #}
  <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

  <script type="text/javascript">
    var classPKList = {{class_pk_list|safe}};
    var preference_level_list =
{{preference_level_list|safe}};
  </script>

  {% js "js/units/class_details.js" %}
{% endblock javascript %}
```

61. upmcts/templates/units/class _list.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
  <div class="col-sm-10 col-sm-push-1">
    {% if unit and not unit.name == '-----' %}
      <h2>{% trans "Classes - " %} {{ unit
}}</h2>
    {% else %}
      <h2>{% trans "Classes" %}</h2>
    {% endif %}
  </div>
</div>

<div class="row margin-bottom-30">
```

```
    <div class="input-group col-sm-10 col-sm-
push-1">
        <form method="get" action='{{
request.path }}'>
            {% if request.user.is_department_admin
%}
                <span class="col-xs-4">
                    {% crispy unit_form %}
                </span>
            {% endif %}
            <span class="col-xs-4">
                {% crispy semester_form %}
            </span>
            <span class="margin-left-10">
                <button type="submit" class="btn
btn-primary">{% trans "Select" %}</button>
            </span>
        </form>
    </div>
</div>

<div class="row">
    <div class="col-sm-10 col-sm-push-1">
        <div class="row">
        <div class="col-sm-6">
            <button type="button" class="popup-
button btn btn-primary"
                    data-url="{% url
'units:class_create' semester.pk %}{{
unit_parameter }}"
                    >{% trans "Add New Class"
%}</button>
        </div>
        <div class="col-sm-6">
            <button type="button" class="popup-
button btn btn-primary pull-right"
                    data-url="{% url
'units:copy_semester_classes' semester.pk
%}{{ unit_parameter }}"
                    >{% trans "Copy All Classes from
a Semester" %}</button>
        </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="class-table" class="display
table table-bordered table-striped">
                    <thead>
                        <tr>
                            <th class="col-md-5">{%
trans "Title" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>
    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    <script type="text/javascript">
        var semester_pk = {{semester.pk|safe}};
        var classes_with_lab =
{{classes_with_lab|safe}};
        var unit_parameter =
"{{unit_parameter}}";
    </script>

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js "js/units/class_list.js" %}
{% endblock javascript %}
```

62. upmcts/templates/units/course_list.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
    <div class="col-sm-10 col-sm-push-1">
        {% if unit and not unit.name == '-----' %}
            <h2>{% trans "Courses - " %} {{ unit
}}</h2>
        {% else %}
            <h2>{% trans "Courses" %}</h2>
        {% endif %}
    </div>
</div>

{% if request.user.is_department_admin %}
    <div class="row margin-bottom-20">
        <div class="input-group col-sm-10 col-
sm-push-1">
            <form method="get" action='{{
request.path }}'>
                <span class="col-xs-6">
                    {% crispy unit_form %}
                </span>
            <span>
                <button type="submit" class="btn
btn-primary btn-sm">{% trans "Select Unit"
%}</button>
            </span>
        </form>
    </div>
  </div>
{% endif %}

<div class="row">

  <div class="col-sm-10 col-sm-push-1">
    <div class="row">
        <div class="col-sm-12">
            <button type="button" class="popup-
button btn btn-primary"
                data-url="{% url
'units:course_create' %}{{ unit_parameter }}"
                >{% trans "Add New Course"
%}</button>
        </div>
    </div>
    <div class="row margin-top-20">
        <div class="col-sm-12">
            <table id="course-table"
class="display table table-bordered table-
striped">
                <thead>
                    <tr>
                        <th class="col-md-2">{%
trans "Name" %}</th>
                        <th class="col-md-5">{%
trans "Title" %}</th>
                        <th class="col-md-1">{%
trans "Units" %}</th>
                        <th class="col-md-1">{%
trans "Lecture" %}</th>
                        <th class="col-md-1">{%
trans "Lab" %}</th>
                        <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                        <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </div>

    </div>

  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  <script type="text/javascript">
    var unit_parameter =
"{{unit_parameter}}";
  </script>

  {# Reinitialize to remove interference of
django_js with select2 #}
  <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

  {% js "js/units/course.js" %}
{% endblock javascript %}
```

63. upmcts/templates/units/instructor_list.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
  <div class="col-sm-10 col-sm-push-1">
      {% if unit and not unit.name == '-----' %}
        <h2>{% trans "Instructors - " %} {{
unit }}</h2>
      {% else %}
        <h2>{% trans "Instructors" %}</h2>
      {% endif %}
  </div>
</div>


{% if request.user.is_department_admin %}
  <div class="row margin-bottom-20">
      <div class="input-group col-sm-10 col-
sm-push-1">
        <form method="get" action='{{
request.path }}'>
            <span class="col-sm-6">
                {% crispy unit_form %}
            </span>
            <span>
                <button type="submit" class="btn
btn-primary btn-sm">{% trans "Select Unit"
%}</button>
            </span>
        </form>
      </div>
  </div>
{% endif %}

<div class="row">
```

```
    <div class="col-sm-10 col-sm-push-1">
        <div class="row">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'units:instructor_create' %}{{ unit_parameter
}}"
                        >{% trans "Add New Instructor"
%}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="instructor-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-4">{%
trans "Name" %}</th>
                            <th class="col-md-2">{%
trans "Faculty Code" %}</th>
                            <th class="col-md-3">{%
trans "Email" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit Prohibited Time Slots"
%}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>

    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    <script type="text/javascript">
        var unit_parameter =
"{{unit_parameter}}";
    </script>

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js "js/units/instructor.js" %}
{% endblock javascript %}
```

64. upmcts/templates/units/
instructor_prohibited_time_slot_list.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Prohibited Time Slots - "
%}{{instructor}}</h2>
    </div>
</div>

<div class="row margin-bottom-30">
    <div class="input-group col-sm-10 col-sm-
push-1">
        <form method="get" action='{{
request.path }}'>
            <span class="col-sm-4">
                {% crispy semester_form %}
            </span>
            <span class="margin-left-10">
                <button type="submit" class="btn
btn-primary">{% trans "Select Semester"
%}</button>
            </span>
        </form>
    </div>
</div>


<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <div class="row">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'units:instructor_prohibited_time_slot_create'
instructor.pk semester.pk %}"
                        >{% trans "Add New Prohibited
Time Slot" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
            <table id="instructor-prohibited-
table" class="display table table-bordered
table-striped">
                <thead>
                    <tr>
                        <th class="col-md-5">{%
trans "Days" %}</th>
                        <th class="col-md-6">{%
trans "Hours" %}</th>
                        <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </div>

    </div>

  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  <script type="text/javascript">
  var instructor_pk = {{instructor.pk|safe}};
  var semester_pk = {{semester.pk|safe}};
  var unit_parameter = "{{unit_parameter}}";
  </script>

  {# Reinitialize to remove interference of
django_js with select2 #}
  <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

  {% js
"js/units/instructor_prohibited_time_slot.js" %}
{% endblock javascript %}
```

65. upmcts/templates/units/room_group_details.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
  <div class="col-sm-10 col-sm-push-1">
    <h2>{% trans "Room Group - "
%}{{room_group}}</h2>
  </div>
</div>

<div class="row">

  <div class="col-sm-10 col-sm-push-1">
    <div class="row">
      <div class="col-sm-12">
        <button type="button" class="popup-
button btn btn-primary"
                data-url="{% url
'units:room_group_member_create'
room_group.pk %}"
                >{% trans "Add New Room to
Group" %}</button>
      </div>
    </div>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <table id="room-group-member-
table" class="display table table-bordered
table-striped">
            <thead>
                <tr>
                    <th class="col-md-2"></th>
                    <th class="col-md-4">{%
trans "Room" %}</th>
                    <th class="col-md-1">{%
trans "Capacity" %}</th>
                    <th class="col-md-3">{%
trans "Preference Level" %}</th>
                    <th class="col-md-1">{%
trans "Edit" %}</th>
                    <th class="col-md-1">{%
trans "Delete" %}</th>
                </tr>
            </thead>
            <tbody></tbody>
        </table>
      </div>

    </div>

  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  <script type="text/javascript">
    var preference_level_list =
{{preference_level_list|safe}};
    var room_group_pk =
{{room_group.pk|safe}};
  </script>
```

```
   {# Reinitialize to remove interference of
django_js with select2 #}
   <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

   {% js "js/units/room_group_details.js" %}
{% endblock javascript %}
```

66. upmcts/templates/units/room_group_list.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
   <div class="col-sm-10 col-sm-push-1">
      {% if unit and not unit.name == '-----' %}
         <h2>{% trans "Room Groups - " %} {{
unit }}</h2>
      {% else %}
         <h2>{% trans "Room Groups"
%}</h2>
      {% endif %}
   </div>
</div>

<div class="row">

   <div class="col-sm-10 col-sm-push-1">
      <div class="row">
         <div class="col-sm-12">
            <button type="button" class="popup-
button btn btn-primary"
                    data-url="{% url
'units:room_group_create' %}"
                    >{% trans "Add New Room
Group" %}</button>
         </div>
      </div>
      <div class="row margin-top-20">
         <div class="col-sm-12">
            <table id="room-group-table"
class="display table table-bordered table-
striped">
               <thead>
                  <tr>
                     <th class="col-md-4">{%
trans "Name" %}</th>
                     <th class="col-md-5">{%
trans "Description" %}</th>
                     <th class="col-md-1 column-
edit">{% trans "Update Rooms" %}</th>
                     <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                     <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                  </tr>
               </thead>
               <tbody></tbody>
            </table>
         </div>

      </div>

   </div>
</div>
{% endblock content %}

{% block javascript %}
   {{ block.super }}
   {% django_js %}
   {% datatables_js %}
   {% datatables_bootstrap_js %}

   {# Reinitialize to remove interference of
django_js with select2 #}
   <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

   {% js "js/units/room_group_list.js" %}
{% endblock javascript %}
```

67. upmcts/templates/units/time_slot_group_details
.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}


<div class="row margin-bottom-20">
   <div class="col-sm-10 col-sm-push-1">
      <h2>{% trans "Time Slot Group - "
%}{{time_slot_group}}</h2>
   </div>
</div>

<div class="row">

   <div class="col-sm-10 col-sm-push-1">
      <div class="row">
         <div class="col-sm-12">
            <button type="button" class="popup-
button btn btn-primary"
                    data-url="{% url
'units:time_slot_group_member_create'
time_slot_group.pk %}"
                    >{% trans "Add New Time Slot
to Group" %}</button>
         </div>
```

```
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="time-slot-group-member-
table" class="display table table-bordered
table-striped">
                    <thead>
                        <tr>
                            <th class="col-md-3">{%
trans "Days" %}</th>
                            <th class="col-md-4">{%
trans "Hours" %}</th>
                            <th class="col-md-3">{%
trans "Preference Level" %}</th>
                            <th class="col-md-1">{%
trans "Edit" %}</th>
                            <th class="col-md-1">{%
trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>

        </div>

    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    <script type="text/javascript">
        var preference_level_list =
{{preference_level_list|safe}};
        var time_slot_group_pk =
{{time_slot_group.pk|safe}};
    </script>

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js "js/units/time_slot_group_details.js"
%}
{% endblock javascript %}
```

68. upmcts/templates/units/class_constraint_list.html

```
{% extends "base.html" %}
{% load crispy_forms_tags js staticfiles
eztables i18n %}

{% block content %}


<div class="row margin-bottom-20">
    <div class="col-sm-10 col-sm-push-1">
        {% if unit and not unit.name == '-----' %}
            <h2>{% trans "Time Slot Groups - "
%} {{ unit }}</h2>
        {% else %}
            <h2>{% trans "Time Slot Groups"
%}</h2>
        {% endif %}
    </div>
</div>

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <div class="row">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'units:time_slot_group_create' %}"
                        >{% trans "Add New Time Slot
Group" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="time-slot-group-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-4">{%
trans "Name" %}</th>
                            <th class="col-md-4">{%
trans "Description" %}</th>
                            <th class="col-md-1">{%
trans "Time Slot Duration" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Update Time Slots" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>

        </div>
    </div>
</div>
{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {# Reinitialize to remove interference of
django_js with select2 #}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    {% js "js/units/time_slot_group_list.js" %}
{% endblock javascript %}
```

69. upmcts/templates/university/college_admin_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "College Admins -
University of the Philippines Manila" %}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'university:college_admin_create' %}"
                        >{% trans "Add New College
Admin" %}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="college-admin-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-2">{%
trans "Username" %}</th>
                            <th class="col-md-1">{%
trans "College" %}</th>
                            <th class="col-md-2">{%
trans "Email" %}</th>
                            <th class="col-md-2">{%
trans "First Name" %}</th>
                            <th class="col-md-2">{%
trans "Last Name" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Details" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Password" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>

        </div>

    </div>
</div>

{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {% js "js/university/college_admin.js" %}
{% endblock javascript %}
```

70. upmcts/templates/university/college _list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">
    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Colleges - University of
the Philippines Manila" %}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'university:college_create' %}"
                        >{% trans "Add New College"
%}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="college-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                        <th class="col-md-3"></th>
                        <th class="col-md-7">{% 
trans "College" %}</th>
                        <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                        <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </div>

    </div>
  </div>
</div>

{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  {% js "js/university/college.js" %}
{% endblock javascript %}
```

71. upmcts/templates/university/college_timetable_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">
  <div class="col-sm-10 col-sm-push-1">
    <h2>{% trans "Timetables - " 
%}{{college}}</h2>
    <div class="row margin-top-20">
      <div class="col-sm-12">
        <table id="college-timetable-table" 
class="display table table-bordered table-
striped">
          <thead>
            <tr>
              <th class="col-md-6">{% 
trans "Semester" %}</th>
              <th class="col-md-3">{% 
trans "Status" %}</th>
              <th class="col-md-1">{% 
trans "Assigned Classes" %}</th>
              <th class="col-md-1">{% 
trans "Unassigned Classes" %}</th>
              <th class="col-md-1 column-
edit">{% trans "View Output" %}</th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>
    </div>
  </div>
</div>
{% endblock content %}

{% block javascript %}
  {{ block.super }}
  {% django_js %}
  {% datatables_js %}
  {% datatables_bootstrap_js %}

  {# Reinitialize to remove interference of 
django_js with select2 #}
  <script src="{% static 
'bower_components/select2/dist/js/select2.js' 
%}"></script>

  <script type="text/javascript">
    var college_pk = {{college.pk|safe}};
    var sem_list = {{sem_list|safe}};
    var status_list = {{status_list|safe}};
  </script>

  {% js 
"js/university/college_timetable_list.js" %}
{% endblock javascript %}
```

72. upmcts/templates/university/college_timetable_output.html

```
{% extends "base.html" %}

{% load crispy_forms_tags js staticfiles 
django_tables2 eztables i18n %}

{% block content %}

<div class="row margin-bottom-20">
  <div class="input-group col-xs-10 col-xs-
push-1">
    <h2>{% trans "Timetable " %} 
({{timetable.semester}}) - 
{{timetable.college}} </h2>
    <form method="get" action='{{ 
request.path }}' class="row margin-top-20">
      <span class="col-sm-4">
        {% crispy status_form %}
      </span>
      <span class="col-sm-4">
        {% crispy department_form %}
      </span>
      <span class="col-sm-2 pull-left">
        <button type="submit" class="btn 
btn-primary btn-sm">{% trans "Filter" 
%}</button>
```

```
            </span>
        </form>
    </div>
</div>

<div class="row">
    <div class="col-xs-10 col-xs-push-1">
        {% render_table table %}
    </div>
</div>
{% endblock content %}
```

73. upmcts/templates/university/semester_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">

    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Semesters - University of
the Philippines Manila" %}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <button type="button" class="popup-
button btn btn-primary"
                        data-url="{% url
'university:semester_create' %}"
                        >{% trans "Add New Semester"
%}</button>
            </div>
        </div>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="semester-table"
class="display table table-bordered table-
striped">
                    <thead>
                        <tr>
                            <th class="col-md-7">{%
trans "Semester" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "Edit" %}</th>
                            <th class="col-md-1 column-
delete">{% trans "Delete" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>
    </div>
</div>

{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {% js "js/university/semester.js" %}
{% endblock javascript %}
```

74. upmcts/templates/university/university_timetab
le_list.html

```
{% extends "base.html" %}

{% load js staticfiles eztables i18n %}

{% block content %}

<div class="row">
    <div class="col-sm-10 col-sm-push-1">
        <h2>{% trans "Timetables - University of
the Philippines Manila" %}</h2>
        <div class="row margin-top-20">
            <div class="col-sm-12">
                <table id="university-timetable-
table" class="display table table-bordered
table-striped">
                    <thead>
                        <tr>
                            <th class="col-md-3"></th>
                            <th class="col-md-7">{%
trans "College" %}</th>
                            <th class="col-md-1 column-
edit">{% trans "View Timetables" %}</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
                </table>
            </div>
        </div>
    </div>
</div>

{% endblock content %}

{% block javascript %}
    {{ block.super }}
    {% django_js %}
    {% datatables_js %}
    {% datatables_bootstrap_js %}

    {% js "js/university/university_timetable.js"
%}
{% endblock javascript %}
```

75. upmcts/templates/users/user_detail.html

```
{% extends "base.html" %}
{% load static %}
{% block title %}User: {{ object.username
}}{% endblock %}

{% block content %}
<div class="container">

  <div class="row">
    <div class="col-sm-12">

      <h2>{{ object.username }}</h2>
      {% if object.name %}
        <p>{{ object.name }}</p>
      {% endif %}
    </div>
  </div>

{% if object == request.user %}
<!-- Action buttons -->
<div class="row">

  <div class="col-sm-12 ">
    <a class="btn btn-primary" href="{% url
'users:update' %}">My Info</a>
    <a class="btn btn-primary" href="{% url
'account_change_password'
%}">Password</a>
    <!-- Your Stuff: Custom user template urls --
>
  </div>

</div>
<!-- End Action buttons -->
{% endif %}


</div>
{% endblock content %}
```

76. upmcts/templates/users/user_form.html

```
{% extends "base.html" %}
{% load crispy_forms_tags %}

{% block title %}{{ user.username }}{%
endblock %}

{% block content %}
    <div class="row">
      <div class="col-md-5">
        <h1>{{ user.username }}</h1>
        <form class="form-horizontal"
method="post" action="{% url 'users:update'
%}">
            {% csrf_token %}
            {{ form|crispy }}
            <div class="control-group">
              <div class="controls">
                <button type="submit" class="btn
btn-primary">Update</button>
              </div>
            </div>
        </form>
      </div>
    </div>
{% endblock %}
```

77. upmcts/templates/users/user_list.html

```
{% extends "base.html" %}
{% load static %}{% load i18n %}
{% block title %}Members{% endblock %}

{% block content %}

<div class="container">

    <h2>Users</h2>

    <div class="list-group">
        {% for user in user_list %}
          <a href="{% url 'users:detail'
user.username %}" class="list-group-item">
            <h4 class="list-group-item-heading">{{
user.username }}</h4>
          </a>
        {% endfor %}

    </div>

{% endblock content %}
```

78. upmcts/templates/utils/generic_message_page.html

```
{% extends "base.html" %}

{% load i18n %}

{% block content %}

<div class="row">
    <div class="col-sm-10 col-sm-push-1">
    {% if message_title %}
        <h1>{{ message_title }}</h1>
    {% endif %}
    {% if message_content %}
        <p>{{ message_content }}</p>
    {% endif %}
    </div>
</div>
{% endblock content %}
```

79. upmcts/templates/_navbar.html

```
{% load staticfiles i18n %}

<nav class="navbar navbar-default navbar-
static-top">
```

```
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="/">{% trans "UPMCTS" %}</a>
      </div>
      <div id="navbar" class="navbar-collapse collapse">
        <ul class="nav navbar-nav">

          {% if request.user.is_authenticated %}
           {% if request.user.is_superuser %}
            <li class="dropdown {% if active_tab_parent == 'colleges' %}active{% endif %}">
              <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">{% trans "Colleges" %} <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li class="{% if active_tab == 'colleges' %}active{% endif %}"><a class="nav-link" href="{% url 'university:college_list' %}">{% trans "Colleges" %}</a></li>
                <li class="{% if active_tab == 'college_admins' %}active{% endif %}"><a class="nav-link" href="{% url 'university:college_admin_list' %}">{% trans "College Admins" %}</a></li>
              </ul>
            </li>
            <li class="nav-item {% if active_tab == 'semesters' %}active{% endif %}">
              <a class="nav-link" href="{% url 'university:semester_list' %}">{% trans "Semesters" %}</a>
            </li>
            <li class="nav-item {% if active_tab == 'university_timetable_list' %}active{% endif %}">
              <a class="nav-link" href="{% url 'university:university_timetable_list' %}">{% trans "Timetables" %}</a>
            </li>
           {% endif %}
           {% if request.user.is_college_admin %}
            <li class="dropdown {% if active_tab_parent == 'departments' %}active{% endif %}">
              <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">{% trans "Departments" %} <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li class="{% if active_tab == 'departments' %}active{% endif %}"><a class="nav-link" href="{% url 'colleges:department_list' %}">{% trans "Departments" %}</a></li>
                <li class="{% if active_tab == 'department_admins' %}active{% endif %}"><a class="nav-link" href="{% url 'colleges:department_admin_list' %}">{% trans "Department Chairmans" %}</a></li>
              </ul>
            </li>
            <li class="dropdown {% if active_tab_parent == 'rooms' %}active{% endif %}">
              <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">{% trans "Rooms" %} <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li class="{% if active_tab == 'rooms' %}active{% endif %}"><a class="nav-link" href="{% url 'colleges:room_list' %}">{% trans "Rooms" %}</a></li>
                <li class="{% if active_tab == 'room_features' %}active{% endif %}"><a class="nav-link" href="{% url 'colleges:room_feature_list' %}">{% trans "Room Features" %}</a></li>
              </ul>
            </li>
            <li class="dropdown {% if active_tab_parent == 'timetables' %}active{% endif %}">
              <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">{% trans "Timetables" %} <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li class="{% if active_tab == 'generate_timetables' %}active{% endif %}"><a class="nav-link" href="{% url 'colleges:timetable_list' %}">{% trans "Generate Timetables" %}</a></li>
                <li class="{% if active_tab == 'all_timetables' %}active{% endif %}"><a class="nav-link" href="{% url
```

```
'university:university_timetable_list' %}">{% 
trans "All Timetables" %}</a></li>
          </ul>
         </li>
        {% endif %}
        {% if 
request.user.is_department_admin %}
         <li class="dropdown {% if 
active_tab_parent == 'units' %}active{% endif 
%}">
          <a href="#" class="dropdown-
toggle" data-toggle="dropdown" role="button" 
aria-haspopup="true" aria-
expanded="false">{% trans "Units" %} <span 
class="caret"></span></a>
          <ul class="dropdown-menu">
           <li class="{% if active_tab == 
'units' %}active{% endif %}"><a class="nav-
link" href="{% url 'departments:unit_list' 
%}">{% trans "Units" %}</a></li>
           <li class="{% if active_tab == 
'unit_admins' %}active{% endif %}"><a 
class="nav-link" href="{% url 
'departments:unit_admin_list' %}">{% trans 
"Unit Heads" %}</a></li>
          </ul>
         </li>
         <li class="dropdown {% if 
active_tab_parent == 'block_sections' 
%}active{% endif %}">
          <a href="#" class="dropdown-
toggle" data-toggle="dropdown" role="button" 
aria-haspopup="true" aria-
expanded="false">{% trans "Block Sections" 
%} <span class="caret"></span></a>
          <ul class="dropdown-menu">
           <li class="{% if active_tab == 
'block_sections' %}active{% endif %}"><a 
class="nav-link" href="{% url 
'departments:block_section_list' %}">{% trans 
"Block Sections" %}</a></li>
           <li class="{% if active_tab == 
'block_section_classes' %}active{% endif 
%}"><a class="nav-link" href="{% url 
'departments:block_section_class_list' %}">{% 
trans "Block Section Classes" %}</a></li>
          </ul>
         </li>
        {% endif %}
        {% if request.user.is_unit_admin or 
request.user.is_department_admin %}
         <li class="dropdown {% if 
active_tab_parent == 'classes' %}active{% 
endif %}">
          <a href="#" class="dropdown-
toggle" data-toggle="dropdown" role="button" 
aria-haspopup="true" aria-
expanded="false">{% trans "Classes" %} 
<span class="caret"></span></a>
          <ul class="dropdown-menu">
           <li class="{% if active_tab == 
'classes' %}active{% endif %}"><a class="nav-
link" href="{% url 'units:class_list' %}">{% 
trans "Classes" %}</a></li>
           <li class="{% if active_tab == 
'class_constraints' %}active{% endif %}"><a 
class="nav-link" href="{% url 
'units:class_constraint_list' %}">{% trans 
"Class Constraints" %}</a></li>
          </ul>
         </li>
         <li class="nav-item {% if active_tab 
== 'courses' %}active{% endif %}">
          <a class="nav-link" href="{% url 
'units:course_list' %}">{% trans "Courses" 
%}</a>
         </li>
         <li class="dropdown {% if 
active_tab_parent == 'groups' %}active{% 
endif %}">
          <a href="#" class="dropdown-
toggle" data-toggle="dropdown" role="button" 
aria-haspopup="true" aria-
expanded="false">{% trans "Groups" %} 
<span class="caret"></span></a>
          <ul class="dropdown-menu">
           <li class="{% if active_tab == 
'room_groups' %}active{% endif %}"><a 
class="nav-link" href="{% url 
'units:room_group_list' %}">{% trans "Room 
Groups" %}</a></li>
           <li class="{% if active_tab == 
'time_slot_groups' %}active{% endif %}"><a 
class="nav-link" href="{% url 
'units:time_slot_group_list' %}">{% trans 
"Time Slot Groups" %}</a></li>
          </ul>
         </li>
         <li class="nav-item {% if active_tab 
== 'instructors' %}active{% endif %}">
          <a class="nav-link" href="{% url 
'units:instructor_list' %}">{% trans 
"Instructors" %}</a>
         </li>
         <li class="nav-item {% if active_tab 
== 'timetables' %}active{% endif %}">
          <a class="nav-link" href="{% url 
'university:university_timetable_list' %}">{% 
trans "Timetables" %}</a>
         </li>
        {% endif %}
       {% else %}
       {% endif %}
      </ul>
```

```
        <ul class="nav navbar-nav navbar-
right">
          {% if request.user.is_authenticated %}
            <li class="nav-item">
              <p class="navbar-text">Hello, {{
request.user.username }}!</p>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="{% url
'users:detail' request.user.username  %}">{%
trans "My Profile" %}</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="{% url
'account_logout' %}">{% trans "Logout"
%}</a>
            </li>
          {% else %}
            {# <li class="nav-item" > #}
              {# <a class="nav-link" href="{% url
'account_login' %}">{% trans "Log In"
%}</a> #}
            {# </li> #}
          {% endif %}
        </ul>
      </div><!--/.nav-collapse -->
    </div>
  </nav>
```

80. upmcts/templates/base.html

```
{% load staticfiles i18n eztables %}
<!DOCTYPE html>
<html lang="en" ng-app>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="x-ua-compatible"
content="ie=edge">
    <title>{% block title %}{% trans
"UPMCTS" %}{% endblock title %}</title>
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">

    <!-- HTML5 shim, for IE6-8 support of
HTML5 elements -->
    <!--[if lt IE 9]>
      <script
src="https://html5shim.googlecode.com/svn/tru
nk/html5.js"></script>
    <![endif]-->

    {% block css %}
    <!-- Latest compiled and minified CSS -->
    <link href="{% static
'bower_components/bootstrap/dist/css/bootstra
p.css' %}" rel="stylesheet">
    <link href="{% static
'bower_components/font-awesome/css/font-
awesome.min.css' %}" rel="stylesheet">
    <link href="{% static
'bower_components/pickadate/lib/compressed/t
hemes/classic.css' %}" rel="stylesheet">
    <link href="{% static
'bower_components/pickadate/lib/compressed/t
hemes/classic.time.css' %}" rel="stylesheet">

    <!-- Your stuff: Third-party css libraries go
here -->
    {% datatables_bootstrap_css %}

    <!-- This file store project specific CSS -->
    <link href="{% static 'css/project.css' %}"
rel="stylesheet">
    {% endblock %}

    {% block header_javascript %}
      <!-- Latest JQuery -->
      <script src="{% static
'bower_components/jquery/dist/jquery.min.js'
%}"></script>
    {% endblock header_javascript %}


  </head>

  <body>

    <!-- Fixed navbar -->
    {% include "_navbar.html" %}
    <div class="container">

      {% if messages %}
          {% for message in messages %}
              <div class="alert {% if message.tags
%}alert-{{ message.tags }}"{% endif %}>{{
message }}</div>
          {% endfor %}
      {% endif %}

      {% block content %}
      {% endblock content %}

    </div> <!-- /container -->

    {% block modal %}
    <div id="popup" class="modal fade" data-
keyboard="false" role= "dialog">
      <div class="modal-dialog"
role="document">
        <div id="popup-content" class="modal-
content">
```

```
      </div><!-- /.modal-content -->
    </div><!-- /.modal-dialog -->
  </div><!-- /.modal -->
  {% endblock modal %}

  <!-- Le javascript
=================================
================ -->
  <!-- Placed at the end of the document so the
pages load faster -->
  {% block javascript %}

    <!-- Your stuff: Third-party javascript
libraries go here -->
    <script src="{% static
'bower_components/bootstrap/dist/js/bootstrap.
js' %}"></script>
    <script src="{% static
'bower_components/noty/js/noty/packaged/jque
ry.noty.packaged.min.js' %}"></script>
    <script src="{% static
'bower_components/pickadate/lib/compressed/
picker.js' %}"></script>
    <script src="{% static
'bower_components/pickadate/lib/compressed/
picker.time.js' %}"></script>
    <script src="{% static
'bower_components/moment/min/moment.min.
js' %}"></script>
    {% datatables_js %}
    {% datatables_bootstrap_js %}
    <script src="{% static
'bower_components/select2/dist/js/select2.js'
%}"></script>

    <!-- place project specific Javascript in this
file -->
    <script src="{% static 'js/project.js'
%}"></script>
  {% endblock javascript %}
 </body>
</html>
```

81. upmcts/templates/base_confirmation_form.ajax
.html

```
{% load i18n %}

<div class="modal-header">
  <button type="button" class="close" data-
dismiss="modal" aria-label="Close">
    <span aria-hidden="true">&times;</span>
  </button>
  <h4 id="popup-title" class="modal-title">{{
popup_title }}</h4>
</div>
<form method="post" class="modal-form"
action='{{ request.get_full_path }}'>
    {% csrf_token %}
    <div id="popup-body" class="modal-body">
      {% block message %}
        {{ message }}
      {% endblock message %}
    </div>
    <div class="modal-footer">
      <button type="button" class="btn btn-
secondary" data-dismiss="modal">{% trans
"Cancel" %}</button>
      <button type="submit" data-loading-
text="Please wait..." class="submit btn btn-
primary">{% block submit_button_name %}{{
submit_button_name }}{% endblock
submit_button_name %}</button>
    </div>
</form>
```

82. upmcts/templates/base_delete_form.ajax.html

```
{% extends
"base_confirmation_form.ajax.html" %}

{% load i18n %}
{% load crispy_forms_tags %}

{% block message %}
  {% trans "Are you sure you want to delete
this?" %}
{% endblock message %}

{% block submit_button_name %}
  {% trans "Delete" %}
{% endblock submit_button_name %}
```

83. upmcts/templates/base_form.ajax.html

```
{% load i18n %}
{% load crispy_forms_tags %}

<div class="modal-header">
  <button type="button" class="close" data-
dismiss="modal" aria-label="Close">
    <span aria-hidden="true">&times;</span>
  </button>
  <h4 id="popup-title" class="modal-title">{{
popup_title }}</h4>
</div>
<form method="post" class="modal-form"
action='{{ request.get_full_path }}' {{
aditional_attributes }}>
    {% csrf_token %}
    <div id="popup-body" class="modal-body">
        {% crispy form %}
    </div>
    <div class="modal-footer">
    <button type="button" class="btn btn-
secondary" data-dismiss="modal">{% trans
"Cancel" %}</button>
    <button type="submit" data-loading-
text="Please wait..." class="submit btn btn-
primary">{% trans "Submit" %}</button>
  </div>
</form>
```

84. upmcts/units/forms.py

```
import datetime

from django import forms
from django.conf import settings
from django.db.models import Q
from django.utils.translation import
ugettext_lazy as _

from crispy_forms.helper import FormHelper
from django_select2.forms import
Select2Widget, Select2MultipleWidget

from upmcts.colleges.models import Room
from upmcts.university.models import
Semester

from .models import (
  Unit, Course, Class, ClassRoomPreference,
ClassTimePreference, TimeSlot,
  ClassConstraint, ConstrainedClass,
RoomGroupMember, TimeSlotGroup,
TimeSlotGroupMember,
  RoomGroup, Instructor,
InstructorProhibitedTimeSlot)


class UnitDropdownForm(forms.Form):
  unit = forms.ModelChoiceField(
    queryset=Unit.objects.all(),
    required=True,
    empty_label=None,
    # widget=Select2Widget
  )

  def __init__(self, unit, *args, **kwargs):
    super(UnitDropdownForm,
self).__init__(*args, **kwargs)
    self.fields['unit'].initial = unit
    self.fields['unit'].label = ''
    self.fields['unit'].queryset =
Unit.objects.filter(department=unit.department)

    self.helper = FormHelper()
    self.helper.form_tag = False
    self.helper.disable_csrf = True
    self.helper.template_pack = 'bootstrap3'

class CourseForm(forms.ModelForm):

  class Meta:
    model = Course
    fields = (
      'name', 'title', 'units',
      'lec_duration', 'required_lec_features',
'has_lab', 'lab_units', 'lab_duration',
      'required_lab_features')
    widgets = {
      'required_lec_features':
Select2MultipleWidget,
      'required_lab_features':
Select2MultipleWidget,
    }

  def __init__(self, *args, **kwargs):
    self.unit = kwargs.pop('unit', None)
    super(CourseForm, self).__init__(*args,
**kwargs)
    self.fields['lab_duration'].required = False
    self.fields['lab_units'].required = False

    self.lec_feature_list = []
    self.lab_feature_list = []
    is_new = not self.instance or not
self.instance.pk
    if not is_new:
      self.lec_feature_list =
self.instance.required_lec_features.values_list('
id', flat=True)
      self.lab_feature_list =
self.instance.required_lab_features.values_list('
id', flat=True)

    self.helper = FormHelper()
    self.helper.form_tag = False
    self.helper.template_pack = 'bootstrap3'

  def clean_lab_duration(self):
    lab_duration =
self.cleaned_data.get('lab_duration', None)
    has_lab = self.cleaned_data.get('has_lab',
None)
    if has_lab and not lab_duration:
      raise forms.ValidationError('This field
is required.')
    return lab_duration

  def clean_lab_units(self):
    lab_units =
self.cleaned_data.get('lab_units', None)
    has_lab = self.cleaned_data.get('has_lab',
None)
    if has_lab and not lab_units:
      raise forms.ValidationError('This field
is required.')
    return lab_units
    def save(self, commit=True, *args,
**kwargs):
        course = super(
            CourseForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk

        lec_features =
self.cleaned_data.get('required_lec_features',
None)
        lec_feature_ids =
lec_features.values_list('id', flat=True)

        lab_features =
self.cleaned_data.get('required_lab_features',
None)
        lab_feature_ids =
lab_features.values_list('id', flat=True)

        if is_new and self.unit:
            course.unit = self.unit

        if commit:
            course.save()
            updated_lec = not
set(self.lec_feature_list) == set(lec_feature_ids)
            if not is_new and updated_lec:
                lec_preferences =
ClassRoomPreference.objects.filter(
                    course_class__course=course,
                    course_class__lecture=None,

).select_related('course_class__course')
                for pref in lec_preferences:
                    lec_class_room_features_id_list =
pref.room.features.values_list('id', flat=True)
                    if (not set(lec_feature_ids) <
set(lec_class_room_features_id_list) and
                            not set(lec_feature_ids) ==
set(lec_class_room_features_id_list)):
                        pref.delete()

            updated_lab = not
set(self.lab_feature_list) ==
set(lab_feature_ids)
            if not is_new and updated_lab:
                lab_preferences =
ClassRoomPreference.objects.filter(
                    course_class__course=course,
                ).exclude(
                    course_class__lecture=None,

).select_related('course_class__course')
                for pref in lab_preferences:
                    lab_class_room_features_id_list =
pref.room.features.values_list('id', flat=True)
                    if (not set(lab_feature_ids) <
set(lab_class_room_features_id_list) and
                            not set(lab_feature_ids) ==
set(lab_class_room_features_id_list)):
                        pref.delete()

            self.save_m2m()
        return course


class ClassForm(forms.ModelForm):
    lab_count = forms.IntegerField(
        _('number of lab classes'),
        required=False,
        min_value=1,)

    class Meta:
        model = Class
        fields = ('course', )
        widgets = {
            'course': Select2Widget,
        }

    def __init__(self, *args, **kwargs):

        initial = kwargs.get('initial', {})
        initial['lab_count'] = 1
        kwargs['initial'] = initial
        self.unit = kwargs.pop('unit', None)
        self.semester = kwargs.pop('semester',
None)
        super(ClassForm, self).__init__(*args,
**kwargs)
        self.fields['course'].queryset =
Course.objects.filter(unit=self.unit)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        class_lecture = super(
            ClassForm,
            self).save(commit=False, *args,
**kwargs)
        lecture_count = Class.objects.filter(
            course=class_lecture.course,
            semester=self.semester,
            lecture=None).count()
        class_lecture.additional_name = 'LEC ' +
str(lecture_count + 1)
        if commit:
            semester = self.semester
            class_lecture.semester = semester
            class_lecture.save()
            course = class_lecture.course

            if course.has_lab:
                lab_count =
self.cleaned_data.get('lab_count', 1)
                for count in xrange(lab_count):
                    Class.objects.create(
                        course=course,
                        lecture=class_lecture,
                        semester=semester,
                        additional_name='LAB ' +
str(lecture_count + 1) + chr(ord('A') + count),
                    )

class_lecture.create_class_constraints(semester
)

        return class_lecture


class ClassSlotsForm(forms.ModelForm):
    class Meta:
        model = Class
        fields = ('slots',)

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        super(ClassSlotsForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        course_class = super(
            ClassSlotsForm,
            self).save(commit=False, *args,
**kwargs)
        slots = self.cleaned_data.get('slots', None)
        if commit:
            course_class.save()
            invalid_room_preferences =
ClassRoomPreference.objects.filter(
                course_class=course_class,
                room__capacity__lt=slots,
            )
            invalid_room_preferences.delete()

        return course_class


class ClassInstructorForm(forms.Form):
    instructor = forms.ModelChoiceField(
        queryset=Instructor.objects.all(),
        widget=Select2Widget)

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.course_class =
kwargs.pop('course_class', None)
        super(ClassInstructorForm,
self).__init__(*args, **kwargs)

        instructor_qs =
Instructor.objects.filter(unit=self.unit)

        instructor_qs =
instructor_qs.exclude(pk__in=self.course_class
.instructors.all())

        self.fields['instructor'].queryset =
instructor_qs
        self.fields['instructor'].label_from_instance
= lambda obj: "{} (Assigned units: {})".format(
            str(obj),
obj.get_semester_units(self.course_class.semes
ter)
        )
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def clean_instructor(self):
        instructor =
self.cleaned_data.get('instructor', None)
        prohibited_time_slot_qs =
InstructorProhibitedTimeSlot.objects.filter(
            instructor=instructor,
            semester=self.course_class.semester,
        )
        prohibited_time_slots =
TimeSlot.objects.filter(id__in=prohibited_time
_slot_qs.values('time_slot__id'))

        preferred_time_slots =
self.course_class.preferred_time_slots.all()
        for time_slot in prohibited_time_slots:
            conflicting_time_slots =
preferred_time_slots.filter(
                (Q(mondays=True) &
Q(mondays=time_slot.mondays)) |
                (Q(tuesdays=True) &
Q(tuesdays=time_slot.tuesdays)) |
                (Q(wednesdays=True) &
Q(wednesdays=time_slot.wednesdays)) |
                (Q(thursdays=True) &
Q(thursdays=time_slot.thursdays)) |
                (Q(fridays=True) &
Q(fridays=time_slot.fridays)) |
                (Q(saturdays=True) &
Q(saturdays=time_slot.saturdays)) |
                (Q(sundays=True) &
Q(sundays=time_slot.sundays))
            ).filter(
                start__lt=time_slot.end,
                end__gt=time_slot.start,
            )
            if conflicting_time_slots:
                conflicts = ', '.join([str(i) for i in
conflicting_time_slots.all()])
                error = ("Instuctor's prohibited time:
{} is conflicting with "
                         "the following preferred time
slots of the class: {}").format(
                    str(time_slot), conflicts
                )
                self.add_error(None, error)

        return instructor

    def save(self, commit=True, *args,
**kwargs):
        instructor =
self.cleaned_data.get('instructor', None)
        if commit:

self.course_class.instructors.add(instructor)

        return instructor


class PreferredRoomForm(forms.ModelForm):

    class Meta:
        model = ClassRoomPreference
        fields = (
            'room', 'preference_level', 'commited', )
        widgets = {
            'room': Select2Widget,
            'preference_level': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.course_class =
kwargs.pop('course_class', None)
        super(PreferredRoomForm,
self).__init__(*args, **kwargs)

        if not self.instance or not self.instance.pk:
            if (self.course_class.lecture is None):
                required_features =
self.course_class.course.required_lec_features.
all()
            else:
                required_features =
self.course_class.course.required_lab_features.
all()
            room_qs = Room.objects.filter(
                college=self.unit.department.college,
).exclude(pk__in=self.course_class.preferred_r
ooms.all())
            room_qs =
room_qs.exclude(capacity__lt=self.course_clas
s.slots)
            if required_features:
                room_qs =
room_qs.filter(features__in=required_features)
            self.fields['room'].queryset = room_qs
        elif self.instance:
            del self.fields["room"]

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        preferred_room = super(
            PreferredRoomForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:
            preferred_room.course_class =
self.course_class

        commited =
self.cleaned_data.get('commited', False)

        if commit:
            if commited:
                previously_commited =
ClassRoomPreference.objects.filter(
                    course_class=self.course_class,
                    commited=True,
                )

previously_commited.update(commited=False)
            preferred_room.save()

        return preferred_room


class PreferredRoomGroupForm(forms.Form):

    room_group = forms.ModelChoiceField(
        queryset=RoomGroup.objects.all(),
        widget=Select2Widget,
        help_text=_(
            'A room in the group will be added to
the preference list if it '
            'satisfies the requirements for the class
and it is not added yet.'))

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.course_class =
kwargs.pop('course_class', None)
        super(PreferredRoomGroupForm,
self).__init__(*args, **kwargs)

        room_group_qs =
RoomGroup.objects.filter(unit__department=se
lf.unit.department)
        self.fields['room_group'].queryset =
room_group_qs

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        room_group =
self.cleaned_data.get('room_group', None)
        if commit:
            room_preferences =
room_group.members.all()

            if (self.course_class.lecture is None):
                class_room_features_id_list =
self.course_class.course.required_lec_features.
values_list('id', flat=True)
            else:
                class_room_features_id_list =
self.course_class.course.required_lab_features.
values_list('id', flat=True)

            for room_pref in room_preferences:
                room_pref_features_id_list =
room_pref.room.features.values_list('id',
flat=True)

                if (room_pref.room.capacity >=
self.course_class.slots and
                    (set(class_room_features_id_list) <
set(room_pref_features_id_list) or
                        set(class_room_features_id_list)
== set(room_pref_features_id_list))):

ClassRoomPreference.objects.get_or_create(
                        course_class=self.course_class,
                        room=room_pref.room,

preference_level=room_pref.preference_level,
                    )

        return room_group


class PreferredTimeForm(forms.ModelForm):
    mondays =
forms.BooleanField(required=False)
    tuesdays =
forms.BooleanField(required=False)
    wednesdays =
forms.BooleanField(required=False)
    thursdays =
forms.BooleanField(required=False)
    fridays =
forms.BooleanField(required=False)
    saturdays =
forms.BooleanField(required=False)
    start =
forms.TimeField(input_formats=settings.TIME
_INPUT_FORMATS)

    class Meta:
        model = ClassTimePreference
        fields = (
            'start', 'mondays', 'tuesdays',
'wednesdays', 'thursdays', 'fridays',
            'saturdays', 'preference_level',
'commited', )
        widgets = {
            'preference_level': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.course_class =
kwargs.pop('course_class', None)
        super(PreferredTimeForm,
self).__init__(*args, **kwargs)
        if self.instance and self.instance.pk:
            self.fields['mondays'].initial =
self.instance.time_slot.mondays
            self.fields['tuesdays'].initial =
self.instance.time_slot.tuesdays
            self.fields['wednesdays'].initial =
self.instance.time_slot.wednesdays
            self.fields['thursdays'].initial =
self.instance.time_slot.thursdays
            self.fields['fridays'].initial =
self.instance.time_slot.fridays
            self.fields['saturdays'].initial =
self.instance.time_slot.saturdays
            self.fields['start'].initial =
self.instance.time_slot.start
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def clean(self):
        cleaned_data = self.cleaned_data
        mondays = cleaned_data.get('mondays',
None)
        tuesdays = cleaned_data.get('tuesdays',
None)
        wednesdays =
cleaned_data.get('wednesdays', None)
```

```
        thursdays = cleaned_data.get('thursdays',
None)
        fridays = cleaned_data.get('fridays', None)
        saturdays = cleaned_data.get('saturdays',
None)
        start = cleaned_data.get('start', None)

        if not(mondays or tuesdays or wednesdays
or thursdays or fridays or saturdays):
            raise forms.ValidationError('Please
select at least one day.')
        is_new = not self.instance or not
self.instance.pk
        if is_new:
            end = self.course_class.get_end_time(
                start, mondays, tuesdays,
wednesdays, thursdays, fridays, saturdays
            )
            instructors =
self.course_class.instructors.all()
            semester = self.course_class.semester
        else:
            end =
self.instance.course_class.get_end_time(
                start, mondays, tuesdays,
wednesdays, thursdays, fridays, saturdays
            )
            instructors =
self.instance.course_class.instructors.all()
            semester =
self.instance.course_class.semester

        for instructor in instructors:
            prohibited_time_slot_qs =
InstructorProhibitedTimeSlot.objects.filter(
                instructor=instructor,
                semester=semester,
            )
            prohibited_time_slots =
TimeSlot.objects.filter(id__in=prohibited_time
_slot_qs.values('time_slot__id'))
            conflicting_time_slots =
prohibited_time_slots.filter(
                (Q(mondays=True) &
Q(mondays=mondays)) |
                (Q(tuesdays=True) &
Q(tuesdays=tuesdays)) |
                (Q(wednesdays=True) &
Q(wednesdays=wednesdays)) |
                (Q(thursdays=True) &
Q(thursdays=thursdays)) |
                (Q(fridays=True) &
Q(fridays=fridays)) |
                (Q(saturdays=True) &
Q(saturdays=saturdays))
            ).filter(
                start__lt=end,
                end__gt=start,
```

```
            )
            if conflicting_time_slots:
                conflicts = ', '.join([str(i) for i in
conflicting_time_slots.all()])
                error = ("Time slot is conflicting with
{}'s "
                         "prohibited time slots:
{}").format(
                    str(instructor), conflicts
                )
                self.add_error(None, error)

        return cleaned_data

    def save(self, commit=True, *args,
**kwargs):
        preferred_time = super(
            PreferredTimeForm,
            self).save(commit=False, *args,
**kwargs)

        cleaned_data = self.cleaned_data
        mondays = cleaned_data.get('mondays',
None)
        tuesdays = cleaned_data.get('tuesdays',
None)
        wednesdays =
cleaned_data.get('wednesdays', None)
        thursdays = cleaned_data.get('thursdays',
None)
        fridays = cleaned_data.get('fridays', None)
        saturdays = cleaned_data.get('saturdays',
None)
        start = cleaned_data.get('start', None)
        preference_level =
cleaned_data.get('preference_level', None)
        commited = cleaned_data.get('commited',
False)


        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:
            preferred_time.course_class =
self.course_class
        if commit:
            if is_new:
                end =
self.course_class.get_end_time(
                    start, mondays, tuesdays,
wednesdays, thursdays, fridays, saturdays
                )
            else:
                end =
self.instance.course_class.get_end_time(
                    start, mondays, tuesdays,
wednesdays, thursdays, fridays, saturdays
                )
            time_slot, created =
TimeSlot.objects.get_or_create(
                start=start, end=end,
mondays=mondays, tuesdays=tuesdays,
                wednesdays=wednesdays,
thursdays=thursdays, fridays=fridays,
                saturdays=saturdays, sundays=False,
            )

            if commited:
                previously_commited =
ClassTimePreference.objects.filter(
                    course_class=self.course_class,
                    commited=True,
                )

previously_commited.update(commited=False)

            try:
                time_preference =
ClassTimePreference.objects.get(
                    course_class=self.course_class,
                    time_slot=time_slot,
                    commited=commited,
                )
                time_preference.preference_level =
preference_level
                time_preference.save()
            except
ClassTimePreference.DoesNotExist:
                preferred_time.time_slot = time_slot
                preferred_time.save()

        return preferred_time


class
PreferredTimeSlotGroupForm(forms.Form):

    time_slot_group = forms.ModelChoiceField(
        queryset=TimeSlotGroup.objects.all(),
        widget=Select2Widget,
        help_text=_(
            'Only groups with the same duration as
the class duration are listed. '
            'A time slot in the group will be added
to the preference list if it '
            'does not have conflicts with a
prohibited time slot for an instructor'
            ' of the class, and it is not added yet.'))

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.course_class =
kwargs.pop('course_class', None)
        super(PreferredTimeSlotGroupForm,
self).__init__(*args, **kwargs)

        if (self.course_class.lecture is None):
            duration =
self.course_class.course.lec_duration
        else:
            duration =
self.course_class.course.lab_duration

        time_slot_qs =
TimeSlotGroup.objects.filter(time_slot_duratio
n=duration)

        self.fields['time_slot_group'].queryset =
time_slot_qs

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        time_slot_group =
self.cleaned_data.get('time_slot_group', None)
        if commit:
            time_slot_group_members =
time_slot_group.members.all()
            instructors =
self.course_class.instructors.all()

            for member in
time_slot_group_members:
                for instructor in instructors:
                    prohibited_time_slot_qs =
InstructorProhibitedTimeSlot.objects.filter(
                        instructor=instructor,

semester=self.course_class.semester,
                    )
                    prohibited_time_slots =
TimeSlot.objects.filter(id__in=prohibited_time
_slot_qs.values('time_slot__id'))
                    conflicting_time_slots =
prohibited_time_slots.filter(
                        (Q(mondays=True) &
Q(mondays=member.time_slot.mondays)) |
                        (Q(tuesdays=True) &
Q(tuesdays=member.time_slot.tuesdays)) |
                        (Q(wednesdays=True) &
Q(wednesdays=member.time_slot.wednesdays)
) |
                        (Q(thursdays=True) &
Q(thursdays=member.time_slot.thursdays)) |
                        (Q(fridays=True) &
Q(fridays=member.time_slot.fridays)) |
                        (Q(saturdays=True) &
Q(saturdays=member.time_slot.saturdays))
                    ).filter(
                        start__lt=member.time_slot.end,
end__gt=member.time_slot.start,
                )
                if not conflicting_time_slots:

ClassTimePreference.objects.get_or_create(

course_class=self.course_class,
                    time_slot=member.time_slot,

preference_level=member.preference_level,
                    )

        return time_slot_group


class ClassConstraintForm(forms.ModelForm):

    class_order =
forms.CharField(max_length=200,
widget=forms.HiddenInput(), required=False)
    classes =
forms.MultipleChoiceField(choices=())

    class Meta:
        model = ClassConstraint
        fields = (
            'constraint', 'preference_level',
'class_order')
        widgets = {
            'constraint': Select2Widget,
            'preference_level': Select2Widget,
            'classes': Select2MultipleWidget,
        }

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.semester = kwargs.pop('semester',
None)
        super(ClassConstraintForm,
self).__init__(*args, **kwargs)
        class_constraint_ids = []
        class_qs = Class.objects.filter(
            semester=self.semester,

course__unit__department__college=self.unit.d
epartment.college,
        )
        if self.instance and self.instance.pk and
self.instance.classes:
            class_constraints =
ConstrainedClass.objects.filter(
                class_constraint=self.instance
            ).order_by('index')
            class_constraint_ids =
list(class_constraints.values_list('course_class_
_id', flat=True))
            selected_classes =
Class.objects.filter(id__in=class_constraint_ids
)
            sorted_classes_all = sorted(
                class_qs,
                key=lambda p:
class_constraint_ids.index(p.pk) if p.pk in
class_constraint_ids else 999
            )
            class_choices = [(c.pk, str(c)) for c in
sorted_classes_all]

            self.fields['classes'].initial = [c.pk for c
in selected_classes]

        sorted_classes_all = sorted(
            class_qs,
            key=lambda p:
class_constraint_ids.index(p.pk) if p.pk in
class_constraint_ids else 999
        )
        class_choices = [(c.pk, str(c)) for c in
sorted_classes_all]
        self.fields['classes'].choices =
class_choices
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def clean(self):
        cleaned_data = self.cleaned_data
        constraint = cleaned_data.get('constraint',
None)
        classes = cleaned_data.get('classes', None)
        classes_count = len(classes)

        if classes_count == 1:
            raise forms.ValidationError('Please
select at least two classes.')

        # Check for class limit
        if constraint in
ClassConstraint.CONSTRAINT_CLASS_EXA
CT_LIMIT:
            limit =
ClassConstraint.CONSTRAINT_CLASS_EXA
CT_LIMIT[constraint]
            if not limit == classes_count:
                self.add_error(None, _(
                    'There must be exactly {} classes
selected'
                    ' when using the selected
constraint'.format(limit)))

        return cleaned_data

    def save(self, commit=True, *args,
**kwargs):
        class_constraint = super(
            ClassConstraintForm,
            self).save(commit=False, *args,
**kwargs)

        if commit:
            is_new = not self.instance or not
self.instance.pk
            if not is_new:

ConstrainedClass.objects.filter(class_constraint
=self.instance).delete()
            elif self.unit and self.semester:
                class_constraint.unit = self.unit
                class_constraint.semester =
self.semester
            class_constraint.constraint_type =
ClassConstraint.CUSTOM
            class_constraint.save()
            classes = self.cleaned_data.get('classes',
None)
            classes = map(int, classes)
            class_order =
self.cleaned_data.get('class_order', None)
            class_order = class_order.split(',')

            order_dict = dict((int(val), idx) for idx,
val in enumerate(class_order))

            classes_qs =
Class.objects.filter(id__in=classes)

            for class_obj in classes_qs:

ConstrainedClass.objects.get_or_create(
                    course_class=class_obj,
                    index=order_dict.get(class_obj.pk,
999),
                    class_constraint=class_constraint)

            class_constraints =
ConstrainedClass.objects.filter(class_constraint
=class_constraint).order_by('-index')
            sorted_classes = sorted(

Class.objects.filter(id__in=class_constraints.val
ues('course_class__id')),
                key=lambda p: classes.index(p.pk) if
p.pk in classes else 1111
            )
            classes_display = []
            for class_obj in sorted_classes:
                classes_display.append('{}
({})'.format(class_obj.course.name,
class_obj.additional_name))
            class_constraint.classes_display = ',
'.join(classes_display)
            class_constraint.save()

        return class_constraint


class SemesterClassesCopyForm(forms.Form):
    semester = forms.ModelChoiceField(
        queryset=Semester.objects.all(),
        widget=Select2Widget,
        help_text=_(
            'All your classes will be replaced with
the classes '
            'from the semester that you will
select.'))

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.semester = kwargs.pop('semester',
None)
        super(SemesterClassesCopyForm,
self).__init__(*args, **kwargs)
        semester_qs =
Semester.objects.exclude(id=self.semester.id)

        self.fields['semester'].queryset =
semester_qs

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        semester_reference =
self.cleaned_data.get('semester', None)
        if commit:
            Class.objects.filter(
                course__unit=self.unit,
                semester=self.semester,
            ).delete()
            lec_classes_to_be_copied =
Class.objects.filter(
                course__unit=self.unit,
                semester=semester_reference,
                lecture=None,
            )
            for class_obj in
lec_classes_to_be_copied:
                new_class = Class.objects.create(
                    course=class_obj.course,
                    semester=self.semester,
                    assigned_room=None,
                    assigned_time_slot=None,
                    lecture=class_obj.lecture,
                    slots=class_obj.slots,

additional_name=class_obj.additional_name,
                    commited=class_obj.commited,
                )
```

```
new_class.instructors.add(*class_obj.instructor
s.all())

            copied_preferred_rooms =
class_obj.classes_for_room.all()
            for preferred_room in
copied_preferred_rooms:
                preferred_room.course_class =
new_class
                preferred_room.pk = None


ClassRoomPreference.objects.bulk_create(copi
ed_preferred_rooms)

            copied_preferred_times =
class_obj.classes_for_time.all()
            for preferred_time in
copied_preferred_times:
                preferred_time.course_class =
new_class
                preferred_time.pk = None


ClassTimePreference.objects.bulk_create(copie
d_preferred_times)

            lab_classes_to_be_copied =
Class.objects.filter(
                lecture=class_obj,
            )

            for lab_class_obj in
lab_classes_to_be_copied:
                new_lab_class =
Class.objects.create(
                    course=lab_class_obj.course,
                    semester=self.semester,
                    assigned_room=None,
                    assigned_time_slot=None,
                    lecture=new_class,
                    slots=lab_class_obj.slots,

additional_name=lab_class_obj.additional_nam
e,

commited=lab_class_obj.commited,
                )


new_lab_class.instructors.add(*lab_class_obj.i
nstructors.all())

                copied_preferred_rooms =
lab_class_obj.classes_for_room.all()
                for preferred_room in
copied_preferred_rooms:
                    preferred_room.course_class =
new_lab_class
                    preferred_room.pk = None


ClassRoomPreference.objects.bulk_create(copi
ed_preferred_rooms)

                copied_preferred_times =
lab_class_obj.classes_for_time.all()
                for preferred_time in
copied_preferred_times:
                    preferred_time.course_class =
new_lab_class
                    preferred_time.pk = None


ClassTimePreference.objects.bulk_create(copie
d_preferred_times)


new_class.create_class_constraints(self.semest
er)

        return semester_reference


class RoomGroupForm(forms.ModelForm):
    class Meta:
        model = RoomGroup
        fields = ('name', 'description', )

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        super(RoomGroupForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        room_group = super(
            RoomGroupForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:
            room_group.unit = self.unit

        if commit:
            room_group.save()

        return room_group
```

```
class
RoomGroupMemberForm(forms.ModelForm):

    class Meta:
        model = RoomGroupMember
        fields = (
            'room', 'preference_level', )
        widgets = {
            'room': Select2Widget,
            'preference_level': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.room_group =
kwargs.pop('room_group', None)
        super(RoomGroupMemberForm,
self).__init__(*args, **kwargs)

        if not self.instance or not self.instance.pk:
            room_qs = Room.objects.filter(
                college=self.unit.department.college

).exclude(pk__in=self.room_group.rooms.all())
            self.fields['room'].queryset = room_qs
        elif self.instance:
            del self.fields["room"]

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        room_group_member = super(
            RoomGroupMemberForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:
            room_group_member.room_group =
self.room_group

        if commit:
            room_group_member.save()

        return room_group_member


class InstructorForm(forms.ModelForm):

    class Meta:
        model = Instructor
        fields = ('first_name', 'middle_initials',
'last_name', 'email', 'faculty_code')

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)

        super(InstructorForm, self).__init__(*args,
**kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        instructor = super(
            InstructorForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:
            instructor.unit = self.unit

        if commit:
            instructor.save()

        return instructor


class
InstructorProhibitedTimeSlotForm(forms.Mode
lForm):
    mondays =
forms.BooleanField(required=False)
    tuesdays =
forms.BooleanField(required=False)
    wednesdays =
forms.BooleanField(required=False)
    thursdays =
forms.BooleanField(required=False)
    fridays =
forms.BooleanField(required=False)
    saturdays =
forms.BooleanField(required=False)
    start =
forms.TimeField(input_formats=settings.TIME
_INPUT_FORMATS)
    end =
forms.TimeField(input_formats=settings.TIME
_INPUT_FORMATS)

    class Meta:
        model = InstructorProhibitedTimeSlot
        fields = (
            'start', 'end', 'mondays', 'tuesdays',
'wednesdays', 'thursdays', 'fridays',
            'saturdays',)

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.instructor = kwargs.pop('instructor',
None)
        self.semester = kwargs.pop('semester',
None)
        super(InstructorProhibitedTimeSlotForm,
self).__init__(*args, **kwargs)

        if self.instance and self.instance.pk:
            self.fields['mondays'].initial =
self.instance.time_slot.mondays
            self.fields['tuesdays'].initial =
self.instance.time_slot.tuesdays
            self.fields['wednesdays'].initial =
self.instance.time_slot.wednesdays
            self.fields['thursdays'].initial =
self.instance.time_slot.thursdays
            self.fields['fridays'].initial =
self.instance.time_slot.fridays
            self.fields['saturdays'].initial =
self.instance.time_slot.saturdays
            self.fields['start'].initial =
self.instance.time_slot.start
            self.fields['end'].initial =
self.instance.time_slot.end

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def clean(self):
        cleaned_data = self.cleaned_data
        mondays = cleaned_data.get('mondays',
None)
        tuesdays = cleaned_data.get('tuesdays',
None)
        wednesdays =
cleaned_data.get('wednesdays', None)
        thursdays = cleaned_data.get('thursdays',
None)
        fridays = cleaned_data.get('fridays', None)
        saturdays = cleaned_data.get('saturdays',
None)

        classes_qs = Class.objects.filter(
            semester=self.semester,
            instructors=self.instructor,
            ).distinct()
        if classes_qs:
            class_list = ', '.join([str(i) for i in
classes_qs])
            error = _('You cannot add more
prohibited time slots since the '
                'following classes are already
assinged to {}: {}. '
                'If you want to add prohibited time
slots for this semester, '
                'Unassign all the classes of the
instructor first.'.format(
                    str(self.instructor), class_list,
                )

            )
            raise forms.ValidationError(error)


        if not(mondays or tuesdays or wednesdays
or thursdays or fridays or saturdays):
            raise forms.ValidationError('Please
select at least one day.')
        return cleaned_data

    def save(self, commit=True, *args,
**kwargs):
        prohibited_time_slot = super(
            InstructorProhibitedTimeSlotForm,
            self).save(commit=False, *args,
**kwargs)

        is_new = not self.instance or not
self.instance.pk
        if is_new:
            prohibited_time_slot.instructor =
self.instructor
            prohibited_time_slot.semester =
self.semester
        if commit:
            cleaned_data = self.cleaned_data
            mondays = cleaned_data.get('mondays',
None)
            tuesdays = cleaned_data.get('tuesdays',
None)
            wednesdays =
cleaned_data.get('wednesdays', None)
            thursdays =
cleaned_data.get('thursdays', None)
            fridays = cleaned_data.get('fridays',
None)
            saturdays =
cleaned_data.get('saturdays', None)
            start = cleaned_data.get('start', None)
            end = cleaned_data.get('end', None)
            time_slot, created =
TimeSlot.objects.get_or_create(
                start=start, end=end,
mondays=mondays, tuesdays=tuesdays,
                wednesdays=wednesdays,
thursdays=thursdays, fridays=fridays,
                saturdays=saturdays, sundays=False,
            )
            try:

InstructorProhibitedTimeSlot.objects.get(
                    instructor=self.instructor,
                    semester=self.semester,
                time_slot=time_slot,
            )
        except
InstructorProhibitedTimeSlot.DoesNotExist:
            prohibited_time_slot.time_slot =
time_slot
        prohibited_time_slot.save()

        return prohibited_time_slot


class TimeSlotGroupForm(forms.ModelForm):
    class Meta:
        model = TimeSlotGroup
        fields = ('name', 'description',
'time_slot_duration')

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        super(TimeSlotGroupForm,
self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def save(self, commit=True, *args,
**kwargs):
        time_slot_group = super(
            TimeSlotGroupForm,
            self).save(commit=False, *args,
**kwargs)
        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:
            time_slot_group.unit = self.unit

        if commit:
            time_slot_group.save()

        return time_slot_group


class
TimeSlotGroupMemberForm(forms.ModelFor
m):

    mondays =
forms.BooleanField(required=False)
    tuesdays =
forms.BooleanField(required=False)
    wednesdays =
forms.BooleanField(required=False)
    thursdays =
forms.BooleanField(required=False)
    fridays =
forms.BooleanField(required=False)
    saturdays =
forms.BooleanField(required=False)
    start =
forms.TimeField(input_formats=settings.TIME
_INPUT_FORMATS)

    class Meta:
        model = TimeSlotGroupMember
        fields = (
            'start', 'mondays', 'tuesdays',
'wednesdays', 'thursdays', 'fridays',
            'saturdays', 'preference_level')
        widgets = {
            'preference_level': Select2Widget,
        }

    def __init__(self, *args, **kwargs):
        self.unit = kwargs.pop('unit', None)
        self.time_slot_group =
kwargs.pop('time_slot_group', None)
        super(TimeSlotGroupMemberForm,
self).__init__(*args, **kwargs)

        if self.instance and self.instance.pk:
            self.fields['mondays'].initial =
self.instance.time_slot.mondays
            self.fields['tuesdays'].initial =
self.instance.time_slot.tuesdays
            self.fields['wednesdays'].initial =
self.instance.time_slot.wednesdays
            self.fields['thursdays'].initial =
self.instance.time_slot.thursdays
            self.fields['fridays'].initial =
self.instance.time_slot.fridays
            self.fields['saturdays'].initial =
self.instance.time_slot.saturdays
            self.fields['start'].initial =
self.instance.time_slot.start

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'

    def clean(self):
        cleaned_data = self.cleaned_data
        mondays = cleaned_data.get('mondays',
None)
        tuesdays = cleaned_data.get('tuesdays',
None)
        wednesdays =
cleaned_data.get('wednesdays', None)
        thursdays = cleaned_data.get('thursdays',
None)
        fridays = cleaned_data.get('fridays', None)
        saturdays = cleaned_data.get('saturdays',
None)
```

```
        if not(mondays or tuesdays or wednesdays
or thursdays or fridays or saturdays):
            raise forms.ValidationError('Please
select at least one day.')

        return cleaned_data

    def save(self, commit=True, *args,
**kwargs):
        time_slot_group_member = super(
            TimeSlotGroupMemberForm,
            self).save(commit=False, *args,
**kwargs)
        cleaned_data = self.cleaned_data
        mondays = cleaned_data.get('mondays',
None)
        tuesdays = cleaned_data.get('tuesdays',
None)
        wednesdays =
cleaned_data.get('wednesdays', None)
        thursdays = cleaned_data.get('thursdays',
None)
        fridays = cleaned_data.get('fridays', None)
        saturdays = cleaned_data.get('saturdays',
None)
        start = cleaned_data.get('start', None)
        preference_level =
cleaned_data.get('preference_level', None)

        days_count = 0
        if mondays:
            days_count += 1
        if tuesdays:
            days_count += 1
        if wednesdays:
            days_count += 1
        if thursdays:
            days_count += 1
        if fridays:
            days_count += 1
        if saturdays:
            days_count += 1
        meeting_duration_in_mins =
int((self.time_slot_group.time_slot_duration /
days_count) * 60)
        start_datetime = datetime.datetime(100, 1,
1, start.hour, start.minute, start.second)
        end_datetime = start_datetime +
datetime.timedelta(minutes=meeting_duration_
in_mins)  # days, seconds, then other fields.

        end = end_datetime.time()

        is_new = not self.instance or not
self.instance.pk
        if is_new and self.unit:

time_slot_group_member.time_slot_group =
self.time_slot_group

        if commit:
            time_slot, created =
TimeSlot.objects.get_or_create(
                start=start, end=end,
mondays=mondays, tuesdays=tuesdays,
                wednesdays=wednesdays,
thursdays=thursdays, fridays=fridays,
                saturdays=saturdays, sundays=False,
            )
            try:
                time_preference =
TimeSlotGroupMember.objects.get(

time_slot_group=self.time_slot_group,
                    time_slot=time_slot,
                )
                time_preference.preference_level =
preference_level
                time_preference.save()
            except
TimeSlotGroupMember.DoesNotExist:
                time_slot_group_member.time_slot =
time_slot
            time_slot_group_member.save()

        return time_slot_group_member
```

85. upmcts/units/models.py

```
import datetime

from django.core.urlresolvers import reverse
from django.db import models
from django.db.models import F, Sum
from django.utils.translation import
ugettext_lazy as _
from django.db.models.signals import
post_save
from django.dispatch import receiver

from model_utils import Choices

from upmcts.colleges.models import Room,
RoomFeature
from upmcts.departments.models import
Department, BlockSection
from upmcts.university.models import
Semester
from upmcts.users.models import User


class UnitManager(models.Manager):
    def get_default_unit(self, department):
        default_unit, created =
super(UnitManager,
self).get_queryset().get_or_create(
            department=department,
            name='-----',
            abbrev=''
        )

        return default_unit


class Unit(models.Model):
    department =
models.ForeignKey(Department,
related_name='units')
    name = models.CharField(
        verbose_name=_(u'Departmental Unit
Name'),
        max_length=255)
    abbrev = models.CharField(
        verbose_name=_(u'Departmental Unit
Abbreviation'),
        max_length=20,
        blank=True)

    objects = UnitManager()

    class Meta:
        unique_together = ('name', 'department', )

    def __unicode__(self):
        return self.name


class UnitAdmin(models.Model):
    user = models.OneToOneField(User)
    unit = models.ForeignKey(
        Unit,
        related_name='unit_admins')

    def __unicode__(self):
        return '{} ({})'.format(self.user.username,
self.unit.abbrev)


class Course(models.Model):
    unit = models.ForeignKey(Unit,
related_name='courses')
    name = models.CharField(
        verbose_name=_(u'Course Name'),
        max_length=50,
        help_text=_(u'Ex. Bio 101'),
        unique=True)
    title = models.CharField(
        verbose_name=_(u'Course Title'),
        max_length=255,
        help_text=_(u'Ex. Introduction to
Biology'))
    units =
models.PositiveIntegerField(verbose_name=_(
u'Units'))
    lab_units =
models.PositiveIntegerField(verbose_name=_(
u'Laboratory Units'), null=True)
    lec_duration = models.DecimalField(
        verbose_name=_(u'Lecture Class Duration
(in hours)'),
        max_digits=4,
        decimal_places=2,
        help_text=_(u'Ex. 1.5'))
    has_lab = models.BooleanField(
        default=False,
        help_text=_(u'Does this class have
laboratory classes?'))
    lab_duration = models.DecimalField(
        verbose_name=_(u'Laboratory Class
Duration (in hours)'),
        max_digits=4,
        decimal_places=2,
        null=True,
        blank=True,
        help_text=_(u'Ex. 1.5'))
    required_lec_features =
models.ManyToManyField(
        RoomFeature,
        verbose_name=_(u'Room Features
Needed for Lecture'),
        related_name='required_lec_features',
        blank=True)
    required_lab_features =
models.ManyToManyField(
        RoomFeature,
        verbose_name=_(u'Room Features
Needed for Laboratory'),
        related_name='required_lab_features',
        blank=True)

    def __unicode__(self):
        return '{} ({})'.format(self.title, self.name)


class TimeSlotManager(models.Manager):
    def get_obj_from_xml_output(self,
timeslot_dict):
        days = timeslot_dict.get('@days',
'0000000')
        start = int(timeslot_dict.get('@start', '0'))
        length = int(timeslot_dict.get('@length',
'0'))

        base_datetime = datetime.datetime(100, 1,
1, 0, 0, 0)  # 12am
        start_datetime = base_datetime +
datetime.timedelta(minutes=start * 5)  # days,
seconds, then other fields.
```

```
        end_datetime = start_datetime +
datetime.timedelta(minutes=length * 5)  # days,
seconds, then other fields.

        mondays = days[0] == '1'
        tuesdays = days[1] == '1'
        wednesdays = days[2] == '1'
        thursdays = days[3] == '1'
        fridays = days[4] == '1'
        saturdays = days[5] == '1'
        sundays = days[6] == '1'
        time_slot, created =
super(TimeSlotManager,
self).get_queryset().get_or_create(
            mondays=mondays,
            tuesdays=tuesdays,
            wednesdays=wednesdays,
            thursdays=thursdays,
            fridays=fridays,
            saturdays=saturdays,
            sundays=sundays,
            start=start_datetime.time(),
            end=end_datetime.time(),
        )

        return time_slot


class TimeSlot(models.Model):

    start = models.TimeField(_(u'start time'),
auto_now=False, auto_now_add=False)
    end = models.TimeField(_(u'end time'),
auto_now=False, auto_now_add=False,
blank=True, null=True)
    mondays = models.BooleanField(_(u'occurs
Mondays'), default=False)
    tuesdays = models.BooleanField(_(u'occurs
Tuesdays'), default=False)
    wednesdays =
models.BooleanField(_(u'occurs Wednesdays'),
default=False)
    thursdays = models.BooleanField(_(u'occurs
Thursdays'), default=False)
    fridays = models.BooleanField(_(u'occurs
Fridays'), default=False)
    saturdays = models.BooleanField(_(u'occurs
Saturdays'), default=False)
    sundays = models.BooleanField(_(u'occurs
Sundays'), default=False)

    objects = TimeSlotManager()

    def __unicode__(self):
        days = ''
        if self.days:
            days = '[{}] '.format(self.days)
        return '{}{}'.format(days, self.hours)

    @property
    def days(self):
        days = ''
        days = days + 'M' if self.mondays else
days
        days = days + 'T' if self.tuesdays else days
        days = days + 'W' if self.wednesdays else
days
        days = days + 'Th' if self.thursdays else
days
        days = days + 'F' if self.fridays else days
        days = days + 'S' if self.saturdays else days
        days = days + 'Su' if self.sundays else days

        return days

    @property
    def hours(self):
        hours = self.start.strftime('%I:%M %p')
        hours = hours + ' - ' +
self.end.strftime('%I:%M %p')
        return hours


class Instructor(models.Model):
    unit = models.ForeignKey(Unit,
related_name='instructors')
    prohibited_time_slots =
models.ManyToManyField(
        TimeSlot,
        through='InstructorProhibitedTimeSlot',
        verbose_name=_(u'prohibited time slots'),
        related_name='prohibited_to',
        blank=True)
    first_name = models.CharField(_('first
name'), max_length=100)
    middle_initials =
models.CharField(_('middle initials'),
max_length=10)
    last_name = models.CharField(_('last name'),
max_length=100)
    email = models.EmailField(_('email'),
blank=True, null=True)
    faculty_code = models.CharField(
        _('faculty code'),
        max_length=20,
        blank=True,
        null=True,
        default=None
    )

    def get_semester_units(self, semester):
        assigned_units_dict =
self.classes.filter(semester=semester,
lecture=None).aggregate(
            semester_units=Sum('course__units')
        )
        assigned_units =
assigned_units_dict.get('semester_units', None)
        assigned_units = assigned_units if
assigned_units is not None else 0

        assigned_lab_units_dict =
self.classes.filter(semester=semester).exclude(
            lecture=None,
        ).aggregate(
            semester_units=Sum('course__units')
        )
        assigned_lab_units =
assigned_lab_units_dict.get('semester_units',
None)
        assigned_lab_units = assigned_lab_units if
assigned_lab_units is not None else 0
        return assigned_units +
assigned_lab_units

    def __unicode__(self):
        return '{} {}
{}'.format(self.first_name.encode('utf-8'),
self.middle_initials.encode('utf-8'),
self.last_name.encode('utf-8'))


class
InstructorProhibitedTimeSlot(models.Model):
    time_slot = models.ForeignKey(
        TimeSlot, on_delete=models.CASCADE,

related_name='time_slot_instructor_prohibited
_memberships')
    instructor = models.ForeignKey(
        Instructor,
        on_delete=models.CASCADE,

related_name='time_slot_instructor_prohibited
_memberships')
    semester = models.ForeignKey(
        Semester,
        on_delete=models.CASCADE,

related_name='time_slot_instructor_prohibited
_memberships')

    class Meta:
        unique_together = ('time_slot', 'semester',
'instructor')

    def __unicode__(self):
        return '{} ({})'.format(self.time_slot,
str(self.semester))


class ClassManager(models.Manager):
    def xml_dict(self, college, semester):
        classes = super(ClassManager,
self).get_queryset().filter(

course__unit__department__college=college,
            semester=semester,
        ).annotate(
            classLimit=F('slots'),

scheduler=F('course__unit__department__pk'),

department=F('course__unit__department__pk'
),
        ).values(
            'id', 'commited',
            'classLimit', 'scheduler',
            'department',
        )
        for class_dict in classes:
            room_pref =
ClassRoomPreference.objects.xml_dict(class_d
ict['id'])
            time_pref =
ClassTimePreference.objects.xml_dict(class_di
ct['id'])
            class_dict.update(room_pref)
            class_dict.update(time_pref)
        classes_dict = {
            'class': list(classes)
        }
        return classes_dict

    def reset_final_schedules(self, college,
semester):
        classes = super(ClassManager,
self).get_queryset().filter(

course__unit__department__college=college,
            semester=semester,
        )
        classes.update(assigned_room=None,
assigned_time_slot=None)

    def assign_final_schedules(self, classes):
        if isinstance(classes, dict):
            classes = [classes, ]
        for class_dict in classes:
            rooms = class_dict.get('room', [])
            if isinstance(rooms, dict):
                rooms = [rooms, ]
            times = class_dict.get('time', [])
            if isinstance(times, dict):
                times = [times, ]
            selected_room = None
            selected_time = None
            try:
                if rooms:
                    selected_room = next(d for i, d in
enumerate(rooms) if '@solution' in d)
```

```
            except Exception:
                pass

            try:
                if times:
                    selected_time = next(d for i, d in
enumerate(times) if '@solution' in d)
            except Exception:
                pass

            try:
                class_id = class_dict.get('@id', None)
                class_obj = super(ClassManager,
self).get_queryset().get(id=class_id)
            except Class.DoesNotExist:
                continue

            try:
                if selected_room:
                    room_id =
selected_room.get('@id', None)
                    assigned_room =
Room.objects.get(id=room_id)
                else:
                    assigned_room = None
                class_obj.assigned_room =
assigned_room
            except Room.DoesNotExist:
                class_obj.assigned_room = None

            try:
                if times:
                    if selected_time:
                        class_obj.assigned_time_slot =
TimeSlot.objects.get_obj_from_xml_output(sel
ected_time)
                    else:
                        class_obj.assigned_time_slot =
None
            except Exception:
                pass
            class_obj.save()


class Class(models.Model):
    course = models.ForeignKey(Course,
related_name='classes')
    semester = models.ForeignKey(Semester,
related_name='classes')
    assigned_room = models.ForeignKey(
        Room, related_name='classes_assigned',
        blank=True, null=True, default=None
    )
    assigned_time_slot = models.ForeignKey(
        TimeSlot,
related_name='classes_assigned',
        blank=True, null=True, default=None
    )
    instructors =
models.ManyToManyField(Instructor,
related_name='classes')
    lecture = models.ForeignKey(
        'self',
        on_delete=models.CASCADE,
        blank=True,
        null=True,
        default=None,
        related_name='labs')
    preferred_rooms =
models.ManyToManyField(
        Room,
        through='ClassRoomPreference',
        verbose_name=_(u'Room Preference'),
        related_name='preferred_by_class',
        blank=True)
    preferred_time_slots =
models.ManyToManyField(
        TimeSlot,
        through='ClassTimePreference',
        verbose_name=_(u'Time Preference'),
        related_name='preferred_by_class',
        blank=True)

    slots =
models.PositiveIntegerField(verbose_name=_(
u'Slots'), default=20)
    additional_name = models.CharField(
        verbose_name=_(u'Additional Name'),
        max_length=50)
    commited = models.BooleanField(
        default=False,
        help_text=_(u'Schedule commited?'))

    objects = ClassManager()

    def __unicode__(self):
        return '{} ({})'.format(self.course.name,
self.additional_name)

    def get_absolute_url(self):
        return reverse('units:class_details',
kwargs=dict(pk=self.pk))

    @property
    def subject(self):
        return '{} ({})'.format(self.course.name,
self.additional_name)

    @property
    def instructor_list(self):
        if self.instructors is not None:
            return ', '.join([str(instructors) for
instructors in self.instructors.all()])
        else:
            return None
    def create_class_constraints(self, semester):
        lab_classes =
Class.objects.filter(lecture=self)
        for lab in lab_classes:
            class_constraint_diff_time =
ClassConstraint.objects.create(

preference_level=ClassConstraint.REQUIRED,

constraint=ClassConstraint.DIFF_TIME,
                semester=semester,

constraint_type=ClassConstraint.IMPLIED,
                unit=self.course.unit,
            )
            ConstrainedClass.objects.get_or_create(
                course_class=lab,
                index=0,

class_constraint=class_constraint_diff_time)
            ConstrainedClass.objects.get_or_create(
                course_class=self,
                index=0,

class_constraint=class_constraint_diff_time)

            classes_display = []
            classes_display.append('{}
({})'.format(self.course.name,
self.additional_name))
            classes_display.append('{}
({})'.format(lab.course.name,
lab.additional_name))

class_constraint_diff_time.classes_display = ',
'.join(classes_display)
            class_constraint_diff_time.save()

            class_constraint_precedence =
ClassConstraint.objects.create(

preference_level=ClassConstraint.STRONGLY
_PREFERED,

constraint=ClassConstraint.PRECEDENCE,
                semester=semester,

constraint_type=ClassConstraint.IMPLIED,
                unit=self.course.unit,
            )
            ConstrainedClass.objects.get_or_create(
                course_class=lab,
                index=0,

class_constraint=class_constraint_precedence)
            ConstrainedClass.objects.get_or_create(
                course_class=self,
                index=0,

class_constraint=class_constraint_precedence)

            classes_display = []
            classes_display.append('{}
({})'.format(self.course.name,
self.additional_name))
            classes_display.append('{}
({})'.format(lab.course.name,
lab.additional_name))

class_constraint_precedence.classes_display =
', '.join(classes_display)
            class_constraint_precedence.save()

    def get_end_time(self, start_time, mondays,
tuesdays, wednesdays, thursdays,
                     fridays, saturdays):
        days_count = 0
        if mondays:
            days_count += 1
        if tuesdays:
            days_count += 1
        if wednesdays:
            days_count += 1
        if thursdays:
            days_count += 1
        if fridays:
            days_count += 1
        if saturdays:
            days_count += 1

        if self.lecture is None:
            weekly_duration =
self.course.lec_duration
        else:
            weekly_duration =
self.course.lab_duration

        meeting_duration_in_mins =
int((weekly_duration / days_count) * 60)

        start_datetime = datetime.datetime(100, 1,
1, start_time.hour, start_time.minute,
start_time.second)
        end_datetime = start_datetime +
datetime.timedelta(minutes=meeting_duration_
in_mins)  # days, seconds, then other fields.
        end = end_datetime.time()
        return end


class
ClassRoomPreferenceManager(models.Manage
r):
    def xml_dict(self, course_class_pk):
        class_room_preferences =
super(ClassRoomPreferenceManager,
self).get_queryset().filter(
            course_class__pk=course_class_pk
        ).annotate(
            room_id=F('room__pk'),
            pref=F('preference_level'),
            solution=F('commited'),
        ).values(
            'room_id', 'pref',
            'solution',
        )
        for pref in class_room_preferences:
            pref['id'] = pref.pop('room_id')
        class_room_preferences_dict = {
            'room': list(class_room_preferences)
        }
        return class_room_preferences_dict


class ClassRoomPreference(models.Model):

    STRONGLY_DISCOURAGED = 2
    DISCOURAGED = 1
    NEUTRAL = 0
    PREFERED = -1
    STRONGLY_PREFERED = -2
    PREFERENCE_LEVEL_CHOICES =
Choices(
        (STRONGLY_PREFERED, _(u'Strongly
Preferred')),
        (PREFERED, _(u'Preferred')),
        (NEUTRAL, _(u'Neutral')),
        (DISCOURAGED, _(u'Discouraged')),
        (STRONGLY_DISCOURAGED,
_(u'Strongly Discouraged')),
    )
    course_class = models.ForeignKey(Class,
on_delete=models.CASCADE,
related_name='classes_for_room')
    room = models.ForeignKey(Room,
on_delete=models.CASCADE,
related_name='rooms')
    preference_level = models.IntegerField(
        _(u'preference level'),

choices=PREFERENCE_LEVEL_CHOICES,
        default=NEUTRAL)

    commited = models.BooleanField(
        _(u'commited'), default=False,
        help_text=_('Assign this room to the class
as the final choice (preference level will be
disregarded)?')
    )

    objects = ClassRoomPreferenceManager()

class
ClassTimePreferenceManager(models.Manager
):
    def xml_dict(self, course_class_pk):
        class_time_preferences_qs =
super(ClassTimePreferenceManager,
self).get_queryset().filter(
            course_class__pk=course_class_pk
        )

        class_time_preferences_list = []

        for pref in class_time_preferences_qs:
            days = '1' if pref.time_slot.mondays else
'0'
            days += '1' if pref.time_slot.tuesdays
else '0'
            days += '1' if pref.time_slot.wednesdays
else '0'
            days += '1' if pref.time_slot.thursdays
else '0'
            days += '1' if pref.time_slot.fridays else
'0'
            days += '1' if pref.time_slot.saturdays
else '0'
            days += '1' if pref.time_slot.sundays
else '0'

            today = datetime.date.today()
            duration = datetime.datetime.combine(
                today, pref.time_slot.end) -
datetime.datetime.combine(today,
pref.time_slot.start)
            length = duration.seconds // 60 // 5

            start_duration =
datetime.datetime.combine(
                today, pref.time_slot.start) -
datetime.datetime(today.year, today.month,
today.day)
            start = start_duration.seconds // 60 // 5
            class_time_preferences_list.append({
                'days': days,
                'start': start,
                'length': length,
                'pref': pref.preference_level,
            })
            if pref.commited:
                class_time_preferences_list.append({
                    'solution': pref.commited,
                })

        class_time_preferences_dict = {
            'time': class_time_preferences_list
        }
        return class_time_preferences_dict
```

```
class ClassTimePreference(models.Model):

   STRONGLY_DISCOURAGED = 2
   DISCOURAGED = 1
   NEUTRAL = 0
   PREFERED = -1
   STRONGLY_PREFERED = -2
   PREFERENCE_LEVEL_CHOICES =
Choices(
      (STRONGLY_PREFERED, _(u'Strongly
Preferred')),
      (PREFERED, _(u'Preferred')),
      (NEUTRAL, _(u'Neutral')),
      (DISCOURAGED, _(u'Discouraged')),
      (STRONGLY_DISCOURAGED,
_(u'Strongly Discouraged')),
   )
   course_class = models.ForeignKey(Class,
on_delete=models.CASCADE,
related_name='classes_for_time')
   time_slot = models.ForeignKey(
      TimeSlot, on_delete=models.CASCADE,
      related_name='classes_for_time',
      blank=True, null=True, default=None)

   preference_level = models.IntegerField(
      _(u'preference level'),

choices=PREFERENCE_LEVEL_CHOICES,
      default=NEUTRAL)

   commited = models.BooleanField(
      _(u'commited'), default=False,
      help_text=_('Assign this time slot to the
class as the final choice (preference level will
be disregarded)?')
   )

   objects = ClassTimePreferenceManager()

   class Meta:
      unique_together = ('time_slot',
'course_class')


class
ClassConstraintManager(models.Manager):
   def xml_dict(self, college, semester):
      class_constraint =
super(ClassConstraintManager,
self).get_queryset().filter(
         semester=semester,
         unit__department__college=college,
      ).annotate(
         type=F('constraint'),
         pref=F('preference_level'),
      ).values(
         'type', 'pref', 'id'
      )
      for constraint in class_constraint:
         classes =
ConstrainedClass.objects.xml_dict(constraint['i
d'])
         constraint.update(classes)
      class_constraint_dict = {
         'constraint': list(class_constraint)
      }
      return class_constraint_dict


class ClassConstraint(models.Model):

   'SAME_ROOM', 'SAME_TIME',
'SAME_START', 'SAME_DAYS',
'BTB_TIME', 'BTB', 'NHB_GTE(1)',
'NHB_LT(6)', 'DIFF_TIME', 'BTB_DAY',
   'PRECEDENCE', 'MIN_ROOM_USE',
'NDB_GT_1', 'FOLLOWING_DAY',
'EVERY_OTHER_DAY'

   SAME_ROOM = 'SAME_ROOM'
   SAME_TIME = 'SAME_TIME'
   SAME_START = 'SAME_START'
   SAME_DAYS = 'SAME_DAYS'
   BTB_TIME = 'BTB_TIME'
   BTB = 'BTB'
   NHB_GTE_1 = 'NHB_GTE(1)'
   NHB_LT_6 = 'NHB_LT(6)'
   DIFF_TIME = 'DIFF_TIME'
   BTB_DAY = 'BTB_DAY'
   PRECEDENCE = 'PRECEDENCE'
   MIN_ROOM_USE = 'MIN_ROOM_USE'
   NDB_GT_1 = 'NDB_GT_1'
   FOLLOWING_DAY =
'FOLLOWING_DAY'
   EVERY_OTHER_DAY =
'EVERY_OTHER_DAY'
   CONSTRAINT_CHOICES = Choices(
      (DIFF_TIME, _(u'Different Time')),
      (SAME_ROOM, _(u'Same Room')),
      (SAME_TIME, _(u'Same Time')),
      (SAME_START, _(u'Same Start')),
      (SAME_DAYS, _(u'Same Days')),
      (BTB_TIME, _(u'Back to Back Time')),
      (BTB, _(u'Back to Back (and same
room)')),
      (NHB_GTE_1, _(u'At Least 1 Hour
Between')),
      (NHB_LT_6, _(u'Less Than 6 Hours
Between')),
      (BTB_DAY, _(u'Back-To-Back Day')),
      (PRECEDENCE, _(u'Precedence')),
      (MIN_ROOM_USE, _(u'Minimize
Number Of Rooms Used')),
        (NDB_GT_1, _(u'More Than 1 Day
Between')),
        (FOLLOWING_DAY, _(u'Next Day')),
        (EVERY_OTHER_DAY, _(u'Two Days
After')),
    )

    REQUIRED = 'R'
    STRONGLY_DISCOURAGED = '2'
    DISCOURAGED = '1'
    PREFERED = '-1'
    STRONGLY_PREFERED = '-2'
    PROHIBITED = 'P'
    PREFERENCE_LEVEL_CHOICES =
Choices(
        (REQUIRED, _(u'Required')),
        (STRONGLY_PREFERED, _(u'Strongly
Preferred')),
        (PREFERED, _(u'Preferred')),
        (DISCOURAGED, _(u'Discouraged')),
        (STRONGLY_DISCOURAGED,
_(u'Strongly Discouraged')),
        (PROHIBITED, _(u'Prohibited')),
    )

    CONSTRAINT_CLASS_EXACT_LIMIT =
{
        EVERY_OTHER_DAY: 2,
        FOLLOWING_DAY: 2,
    }

    CONSTRAINT_HELP_TEXT_LIST = {
        SAME_ROOM: _('Included classes must
be scheduled in the same room.'),
        SAME_TIME: _(
            'Included classes must occur at the same
time period (not necessarily '
            'same day). If the classes have different
lengths, All shorter classes '
            'must occur inside the time period of the
longest class.'),
        SAME_START: _('Included classes must
start at the same time.'),
        SAME_DAYS: _('Included classes must
be scheduled on the same days.'),
        BTB_TIME: _('Included classes must be
scheduled adjacent (time) to each other.'),
        BTB: _(
            'Included classes must be scheduled
adjacent (time) to each other'
            ' in the same room.'),
        NHB_GTE_1: _('Included classes must be
at least one hour apart from each other (on the
same day).'),
        NHB_LT_6: _('Included classes must not
be scheduled more than six hours apart on the
same day.'),
        DIFF_TIME: _('Included classes must not
overlap in time if scheduled on the same day.'),
        BTB_DAY: _('Included classes must be
scheduled on adjacent days.'),
        PRECEDENCE: _(
            'Included classes must be schedule in
the order you choose.'
            ' You can drag a class to reorder the
classes.'),
        MIN_ROOM_USE: _('Included classes
must be placed in as fewest rooms as
possible.'),
        NDB_GT_1: _('Included classes must be
scheduled at least one day between them.'),
        FOLLOWING_DAY: _(
            'The second class must be scheduled on
the following day '
            'that the first class is scheduled.'),
        EVERY_OTHER_DAY: _('The second
class must be scheduled two days after the first
class.'),
    }

    IMPLIED = 0
    CUSTOM = 1
    CONSTRAINT_TYPE_CHOICES =
Choices(
        (IMPLIED, _(u'Implied (created
automatically)')),
        (CUSTOM, _(u'Custom (created by
user)')),
    )
    unit = models.ForeignKey(Unit,
related_name='class_constraints',
on_delete=models.CASCADE)
    semester = models.ForeignKey(Semester,
related_name='class_constraints')
    classes = models.ManyToManyField(
        Class,
        through='ConstrainedClass',
        verbose_name=_(u'classes'),
        related_name='constraints')

    constraint = models.CharField(
        _(u'constraint'),
        max_length=20,
        choices=CONSTRAINT_CHOICES,
        default=DIFF_TIME,
        help_text='Select a constraint to view its
details.')

    classes_display =
models.TextField(verbose_name=_(u'classes
display'))

    preference_level = models.CharField(
        _(u'preference level'),
        max_length=2,
choices=PREFERENCE_LEVEL_CHOICES,
        default=REQUIRED)

    constraint_type = models.IntegerField(
        _(u'preference level'),

choices=CONSTRAINT_TYPE_CHOICES,
        blank=True,
        default=IMPLIED)

    objects = ClassConstraintManager()

    def __unicode__(self):
        return '[{}] {} ({})'.format(self.pk,
self.get_constraint_display(),
self.get_preference_level_display())


class
ConstrainedClassManager(models.Manager):
    def xml_dict(self, class_constraint_pk):
        constrained_classes =
super(ConstrainedClassManager,
self).get_queryset().filter(

class_constraint__pk=class_constraint_pk
        ).values(
            'course_class__id'
        )

        for constrained_class in
constrained_classes:
            constrained_class['id'] =
constrained_class.pop('course_class__id')
        constrained_classes_dict = {
            'class': list(constrained_classes)
        }
        return constrained_classes_dict


class ConstrainedClass(models.Model):
    course_class = models.ForeignKey(Class,
on_delete=models.CASCADE)
    class_constraint =
models.ForeignKey(ClassConstraint,
on_delete=models.CASCADE)
    index =
models.PositiveIntegerField(u'index',
default=0)

    objects = ConstrainedClassManager()

    def __unicode__(self):
        return '{} ({})'.format(self.course_class,
self.class_constraint)

class BlockSectionClasses(models.Model):
    block_section =
models.ForeignKey(BlockSection,
related_name='block_section_classes')
    semester = models.ForeignKey(Semester,
related_name='block_section_classes')
    classes = models.ManyToManyField(
        Class,
        verbose_name=_(u'classes'),
        related_name='block_section_classes')

    class Meta:
        unique_together = ('block_section',
'semester', )


class RoomGroup(models.Model):
    unit = models.ForeignKey(Unit,
related_name='room_groups')
    rooms = models.ManyToManyField(
        Room,
        through='RoomGroupMember',
        verbose_name=_(u'Rooms'),
        related_name='room_groups',
        blank=True)
    name = models.CharField(
        verbose_name=_(u'Room Group Name'),
        max_length=255,)
    description =
models.TextField(verbose_name=_(u'descriptio
n'), blank=True)

    def __unicode__(self):
        return self.name


class RoomGroupMember(models.Model):

    STRONGLY_DISCOURAGED = 2
    DISCOURAGED = 1
    NEUTRAL = 0
    PREFERED = -1
    STRONGLY_PREFERED = -2
    PREFERENCE_LEVEL_CHOICES =
Choices(
        (STRONGLY_PREFERED, _(u'Strongly
Preferred')),
        (PREFERED, _(u'Preferred')),
        (NEUTRAL, _(u'Neutral')),
        (DISCOURAGED, _(u'Discouraged')),
        (STRONGLY_DISCOURAGED,
_(u'Strongly Discouraged')),
    )
    room_group =
models.ForeignKey(RoomGroup,
on_delete=models.CASCADE,
related_name='members')
```

```
    room = models.ForeignKey(Room,
on_delete=models.CASCADE,
related_name='room_group_memberships')
    preference_level = models.IntegerField(
        _(u'preference level'),

choices=PREFERENCE_LEVEL_CHOICES,
        default=NEUTRAL)

    def __unicode__(self):
        return '{}
({})'.format(self.room_group.name,
str(self.room))


class TimeSlotGroup(models.Model):
    unit = models.ForeignKey(Unit,
related_name='time_slot_groups')
    time_slots = models.ManyToManyField(
        TimeSlot,
        through='TimeSlotGroupMember',
        verbose_name=_(u'time slots'),
        related_name='time_slot_groups',
        blank=True)
    name = models.CharField(
        verbose_name=_(u'Time Slot Group
Name'),
        max_length=255,)
    description =
models.TextField(verbose_name=_(u'descriptio
n'), blank=True)
    time_slot_duration = models.DecimalField(
        verbose_name=_(u'Time Slot Duration (in
hours)'),
        max_digits=4,
        decimal_places=2,
        help_text=_(u'Ex. 1.5'))

    def __unicode__(self):
        return self.name


class TimeSlotGroupMember(models.Model):

    STRONGLY_DISCOURAGED = 2
    DISCOURAGED = 1
    NEUTRAL = 0
    PREFERED = -1
    STRONGLY_PREFERED = -2
    PREFERENCE_LEVEL_CHOICES =
Choices(
        (STRONGLY_PREFERED, _(u'Strongly
Preferred')),
        (PREFERED, _(u'Preferred')),
        (NEUTRAL, _(u'Neutral')),
        (DISCOURAGED, _(u'Discouraged')),
        (STRONGLY_DISCOURAGED,
_(u'Strongly Discouraged')),
    )
    time_slot_group =
models.ForeignKey(TimeSlotGroup,
on_delete=models.CASCADE,
related_name='members')
    time_slot = models.ForeignKey(TimeSlot,
on_delete=models.CASCADE,
related_name='time_slot_group_memberships')
    preference_level = models.IntegerField(
        _(u'preference level'),

choices=PREFERENCE_LEVEL_CHOICES,
        default=NEUTRAL)

    class Meta:
        unique_together = ('time_slot',
'time_slot_group')

    def __unicode__(self):
        return '{}
({})'.format(self.time_slot_group.name,
str(self.time_slot))


@receiver(post_save, sender=Department)
def post_save_department(sender, **kwargs):
    created = kwargs.get('created', False)
    instance = kwargs.get('instance')

    # Create dafault unit for dept
    if created:
        Unit.objects.create(
            department=instance,
            name='-----',
            abbrev='')
```

86. upmcts/units/urls.py

```
from django.conf.urls import url

from . import views


urlpatterns = [

    url(
        r'^courses/$',
        views.CourseListView.as_view(),
        name='course_list'
    ),
    url(
        r'^courses/objects/$',

views.ObjectCourseDatatablesView.as_view(),
        name='DT_course_objects'
    ),
    url(
        r'^courses/new/$',
        views.CourseCreateView.as_view(),
        name='course_create'
    ),
    url(
        r'^courses/(?P<pk>\d+)/delete/$',
        views.CourseDeleteView.as_view(),
        name='course_delete'
    ),
    url(r'^courses/(?P<pk>\d+)/update/$',
        views.CourseUpdateView.as_view(),
        name='course_update'),

    url(
        r'^classes/$',
        views.ClassListView.as_view(),
        name='class_list'
    ),
    url(

r'^classes/(?P<semester_pk>\d+)/objects/$',

views.ObjectClassDatatablesView.as_view(),
        name='DT_class_objects'
    ),
    url(
        r'^classes/(?P<semester_pk>\d+)/new/$',
        views.ClassCreateView.as_view(),
        name='class_create'
    ),


    url(r'^classes/(?P<pk>\d+)/slots_update/$',
        views.ClassSlotsUpdateView.as_view(),
        name='class_slots_update'),

    url(
        r'^classes/(?P<pk>\d+)/delete/$',
        views.ClassDeleteView.as_view(),
        name='class_delete'
    ),


    url(r'^classes/(?P<pk>\d+)/details/$',
        views.ClassDetailView.as_view(),
        name='class_details'),

    url(

r'^classes/(?P<semester_pk>\d+)/replace/$',

views.SemesterClassesCopyView.as_view(),
        name='copy_semester_classes'
    ),


url(r'^classes/(?P<class_pk>\d+)/room_prefere
nce/$',

views.ObjectClassPreferredRoomsDatatablesVi
ew.as_view(),

name='DT_class_room_preference_objects'),


url(r'^classes/(?P<class_pk>\d+)/room_prefere
nce/create$',

views.PreferredRoomCreateView.as_view(),
        name='preferred_room_create'),


url(r'^classes/(?P<class_pk>\d+)/room_group_
preference/create$',

views.PreferredRoomGroupCreateView.as_vie
w(),
        name='preferred_room_group_create'),

    url(
        r'^room_preference/(?P<pk>\d+)/delete/$',

views.PreferredRoomDeleteView.as_view(),
        name='preferred_room_delete'
    ),

    url(

r'^room_preference/(?P<pk>\d+)/update/$',

views.PreferredRoomUpdateView.as_view(),
        name='preferred_room_update'
    ),


url(r'^classes/(?P<class_pk>\d+)/time_preferen
ce/$',

views.ObjectClassPreferredTimesDatatablesVi
ew.as_view(),

name='DT_class_time_preference_objects'),


url(r'^classes/(?P<class_pk>\d+)/time_preferen
ce/create$',

views.PreferredTimeCreateView.as_view(),
        name='preferred_time_create'),


url(r'^classes/(?P<class_pk>\d+)/time_slot_gro
up_preference/create$',

views.PreferredTimeSlotGroupCreateView.as_
view(),
name='preferred_time_slot_group_create'),

   url(
      r'^time_preference/(?P<pk>\d+)/delete/$',

views.PreferredTimeDeleteView.as_view(),
      name='preferred_time_delete'
   ),

   url(
      r'^time_preference/(?P<pk>\d+)/update/$',

views.PreferredTimeUpdateView.as_view(),
      name='preferred_time_update'
   ),


url(r'^classes/(?P<class_pk>\d+)/instructor/(?P
<instructor_pk>\d+)/delete$',

views.ClassInstructorDeleteView.as_view(),
      name='class_instructor_delete'),


url(r'^classes/(?P<class_pk>\d+)/instructor/$',

views.ObjectClassInstructorsDatatablesView.as
_view(),
      name='DT_class_instructor_objects'),


url(r'^classes/(?P<class_pk>\d+)/instructor/crea
te$',

views.ClassInstructorCreateView.as_view(),
      name='class_instructor_create'),

   url(
      r'^constraints/$',
      views.ClassConstraintListView.as_view(),
      name='class_constraint_list'
   ),
   url(

r'^constraints/(?P<semester_pk>\d+)/objects/$',

views.ObjectClassConstraintDatatablesView.as
_view(),
      name='DT_class_constraint_objects'
   ),
   url(

r'^constraints/(?P<semester_pk>\d+)/new/$',

views.ClassConstraintCreateView.as_view(),
      name='class_constraint_create'
   ),
   url(
      r'^constraints/(?P<pk>\d+)/delete/$',

views.ClassConstraintDeleteView.as_view(),
      name='class_constraint_delete'
   ),
   url(r'^constraints/(?P<pk>\d+)/update/$',

views.ClassConstraintUpdateView.as_view(),
      name='class_constraint_update'),

   url(
      r'^room_groups/$',
      views.RoomGroupListView.as_view(),
      name='room_group_list'
   ),
   url(
      r'^room_groups/objects/$',

views.ObjectRoomGroupDatatablesView.as_vi
ew(),
      name='DT_room_group_objects'
   ),
   url(
      r'^room_groups/new/$',
      views.RoomGroupCreateView.as_view(),
      name='room_group_create'
   ),

   url(
      r'^room_groups/(?P<pk>\d+)/update/$',
      views.RoomGroupUpdateView.as_view(),
      name='room_group_update'
   ),

   url(
      r'^room_groups/(?P<pk>\d+)/delete/$',
      views.RoomGroupDeleteView.as_view(),
      name='room_group_delete'
   ),

   url(r'^room_groups/(?P<pk>\d+)/details/$',
      views.RoomGroupDetailView.as_view(),
      name='room_group_details'),


url(r'^room_group/(?P<room_group_pk>\d+)/
member/$',

views.ObjectRoomGroupMemberDatatablesVi
ew.as_view(),

name='DT_room_group_member_objects'),


url(r'^room_group/(?P<room_group_pk>\d+)/
member/create$',
views.RoomGroupMemberCreateView.as_vie
w(),
        name='room_group_member_create'),

    url(

r'^room_group_member/(?P<pk>\d+)/delete/$',

views.RoomGroupMemberDeleteView.as_vie
w(),
        name='room_group_member_delete'
    ),

    url(

r'^room_group_member/(?P<pk>\d+)/update/$'
,

views.RoomGroupMemberUpdateView.as_vie
w(),
        name='room_group_member_update'
    ),


    url(
        r'^instructors/$',
        views.InstructorListView.as_view(),
        name='instructor_list'
    ),
    url(
        r'^instructors/objects/$',

views.ObjectInstructorDatatablesView.as_view
(),
        name='DT_instructor_objects'
    ),
    url(
        r'^instructors/new/$',
        views.InstructorCreateView.as_view(),
        name='instructor_create'
    ),
    url(
        r'^instructors/(?P<pk>\d+)/delete/$',
        views.InstructorDeleteView.as_view(),
        name='instructor_delete'
    ),
    url(r'^instructors/(?P<pk>\d+)/update/$',
        views.InstructorUpdateView.as_view(),
        name='instructor_update'),
    url(

r'^instructors/(?P<pk>\d+)/prohibited_time_slo
ts/$',

views.InstructorPohibitedTimeSlotListView.as
_view(),

name='instructor_prohibited_time_slot_list'
    ),

    url(

r'^instructors/(?P<pk>\d+)/prohibited_time_slo
ts//(?P<semester_pk>\d+)/objects$',

views.ObjectInstructorPohibitedTimeSlotDatat
ablesView.as_view(),

name='DT_instructor_prohibited_time_slot_obj
ects'
    ),
    url(

r'^instructors/(?P<pk>\d+)/prohibited_time_slo
ts/(?P<semester_pk>\d+)/create$',

views.InstructorPohibitedTimeSlotCreateView.
as_view(),

name='instructor_prohibited_time_slot_create'
    ),
    url(

r'^instructors/prohibited_time_slots/(?P<pk>\d
+)/delete$',

views.InstructorPohibitedTimeSlotDeleteView.
as_view(),

name='instructor_prohibited_time_slot_delete'
    ),

    url(
        r'^time_slot_groups/$',
        views.TimeSlotGroupListView.as_view(),
        name='time_slot_group_list'
    ),
    url(
        r'^time_slot_groups/objects/$',

views.ObjectTimeSlotGroupDatatablesView.as
_view(),
        name='DT_time_slot_group_objects'
    ),
    url(
        r'^time_slot_groups/new/$',

views.TimeSlotGroupCreateView.as_view(),
        name='time_slot_group_create'
    ),

    url(

r'^time_slot_groups/(?P<pk>\d+)/update/$',
views.TimeSlotGroupUpdateView.as_view(),
        name='time_slot_group_update'
    ),

    url(

r'^time_slot_groups/(?P<pk>\d+)/delete/$',

views.TimeSlotGroupDeleteView.as_view(),
        name='time_slot_group_delete'
    ),


url(r'^time_slot_groups/(?P<pk>\d+)/details/$',

views.TimeSlotGroupDetailView.as_view(),
        name='time_slot_group_details'),


url(r'^time_slot_group/(?P<time_slot_group_p
k>\d+)/member/$',

views.ObjectTimeSlotGroupMemberDatatables
View.as_view(),

name='DT_time_slot_group_member_objects'),


url(r'^time_slot_group/(?P<time_slot_group_p
k>\d+)/member/create$',

views.TimeSlotGroupMemberCreateView.as_v
iew(),
        name='time_slot_group_member_create'),

    url(

r'^time_slot_group_member/(?P<pk>\d+)/delet
e/$',

views.TimeSlotGroupMemberDeleteView.as_v
iew(),
        name='time_slot_group_member_delete'
    ),

    url(

r'^time_slot_group_member/(?P<pk>\d+)/upda
te/$',

views.TimeSlotGroupMemberUpdateView.as_
view(),
        name='time_slot_group_member_update'
    ),

]
```

87. upmcts/units/views.py

```
import json
import re

from django.contrib import messages
from django.core.exceptions import
PermissionDenied
from django.core.urlresolvers import reverse
from django.db.models import Q
from django.shortcuts import render, redirect,
get_object_or_404
from django.template import RequestContext
from django.utils.six import text_type
from django.utils.translation import
ugettext_lazy as _
from django.views.generic import (
    TemplateView, CreateView, DeleteView,
UpdateView, FormView)

from braces.views import
(JSONResponseMixin, LoginRequiredMixin)
from eztables.views import DatatablesView

from upmcts.units.models import Unit,
Instructor
from upmcts.users.views import
AjaxTemplateMixin
from upmcts.university.forms import
SemesterDropdownForm
from upmcts.university.models import
Semester
from upmcts.utils.utils import LazyEncoder

from .forms import (
    CourseForm, ClassForm,
PreferredRoomForm, PreferredTimeForm,
    ClassConstraintForm, ClassInstructorForm,
ClassSlotsForm,
    SemesterClassesCopyForm,
RoomGroupForm, RoomGroupMemberForm,
    PreferredRoomGroupForm,
UnitDropdownForm, InstructorForm,
    InstructorProhibitedTimeSlotForm,
    TimeSlotGroupForm,
TimeSlotGroupMemberForm,
    PreferredTimeSlotGroupForm)
from .models import (
    Course, Class, ClassRoomPreference,
    ClassTimePreference, ClassConstraint,
RoomGroup, RoomGroupMember,
    TimeSlotGroup, TimeSlotGroupMember,
InstructorProhibitedTimeSlot)


RE_FORMATTED = re.compile(r'\{(\w+)\}')
```

```
class DepartmentobjectMixin(object):
   def dispatch(self, request, *args, **kwargs):
      if ((request.user.is_unit_admin() or
request.user.is_department_admin()) and
            self.get_unit().department ==
self.get_related_unit().department):
         return super(
            UnitAdminMixin,
self).dispatch(request, *args, **kwargs)
      else:
         raise PermissionDenied


class SemesterObjectMixin(object):
   def dispatch(self, request, *args, **kwargs):
      semesters =
Semester.objects.order_by('academic_year')
      if semesters.count() == 0:
         context = {}
         messages.add_message(
            request, messages.ERROR,
            _('Sorry, the university admin has not
created any semesters yet.'))
         return render(request,
'utils/generic_message_page.html',
RequestContext(request, context))
      semester_id =
self.request.GET.get('semester', None)
      if semester_id:
         try:
            self.semester =
Semester.objects.get(pk=semester_id)
         except Semester.DoesNotExist:
            self.semester = semesters.last()
      else:
         self.semester = semesters.last()
      return super(SemesterObjectMixin,
self).dispatch(request, *args, **kwargs)

   def get_context_data(self, **kwargs):
      context = super(SemesterObjectMixin,
self).get_context_data(**kwargs)

      context['semester'] = self.semester
      context['semester_form'] =
SemesterDropdownForm(
         semester=self.semester
      )
      return context


class UnitAdminMixin(LoginRequiredMixin):
   def dispatch(self, request, *args, **kwargs):
      # Department admin is olso a unit head
      if request.user.is_unit_admin() or
request.user.is_department_admin():
         return super(
            UnitAdminMixin,
self).dispatch(request, *args, **kwargs)
      else:
         raise PermissionDenied

   def get_unit(self):
      if
self.request.user.is_department_admin():
         unit_id = self.request.GET.get('unit',
None)
         if unit_id is not None:
            try:
               unit = Unit.objects.get(pk=unit_id)
            except Unit.DoesNotExist:
               unit =
Unit.objects.get_default_unit(

self.request.user.departmentadmin.department)
         else:
            unit = Unit.objects.get_default_unit(

self.request.user.departmentadmin.department)
         return unit
      else:
         return self.request.user.unitadmin.unit

   def get_context_data(self, **kwargs):
      context = super(UnitAdminMixin,
self).get_context_data(**kwargs)
      unit = self.get_unit()
      context['unit'] = unit
      context['unit_form'] =
UnitDropdownForm(
         unit=unit
      )
      context['unit_parameter'] = '?unit=' +
str(unit.pk)
      return context


class
UnitAdminWithObjectMixin(UnitAdminMixin
):
   def dispatch(self, request, *args, **kwargs):
      if ((request.user.is_unit_admin() and
self.get_unit() == self.get_related_unit()) or
         (request.user.is_department_admin()
and self.get_unit().department ==
self.get_related_unit().department)
            ):
         return super(
            UnitAdminMixin,
self).dispatch(request, *args, **kwargs)
      else:
         raise PermissionDenied
```

```
class
UnitAdminFormMixin(UnitAdminMixin):
    def get_form_kwargs(self):
        kwargs = super(UnitAdminFormMixin,
self).get_form_kwargs()
        kwargs.update({'unit': self.get_unit()})
        return kwargs


class CourseListView(UnitAdminMixin,
TemplateView):
    template_name = 'units/course_list.html'

    def get_context_data(self, **kwargs):
        context = super(CourseListView,
self).get_context_data(**kwargs)
        context['active_tab'] = 'courses'
        return context


class
ObjectCourseDatatablesView(UnitAdminMixi
n, DatatablesView):
    model = Course
    fields = {
        'name': 'name',
        'title': 'title',
        'units': 'units',
        'lec_duration': 'lec_duration',
        'has_lab': 'has_lab',
        'lab_duration': 'lab_duration',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs = super(ObjectCourseDatatablesView,
self).get_queryset()
        return qs.filter(unit=self.get_unit())


class
CourseCreateView(UnitAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = CourseForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:course_list')
        return reverse('units:course_create')

    def get_context_data(self, **kwargs):
        context = super(CourseCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Course'
        return context


class
CourseDeleteView(UnitAdminWithObjectMixi
n, JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = Course

    def get_related_unit(self):
        course = self.get_object()
        return course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:course_list')
        return reverse('units:course_list')

    def get_context_data(self, **kwargs):
        context = super(CourseDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Course'
        return context


class
CourseUpdateView(UnitAdminWithObjectMix
in, JSONResponseMixin, AjaxTemplateMixin,
UpdateView):
    template_name = 'base_form.html'
    model = Course
    form_class = CourseForm

    def get_related_unit(self):
        course = self.get_object()
        return course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:course_list')
        return reverse('units:course_update')

    def get_context_data(self, **kwargs):
        context = super(CourseUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Course'
        return context


class ClassListView(SemesterObjectMixin,
UnitAdminMixin, TemplateView):
    template_name = 'units/class_list.html'

    def get_context_data(self, **kwargs):
        context = super(ClassListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'classes'
        context['active_tab'] = 'classes'
        context['classes_with_lab'] =
Course.objects.filter(
            unit=self.get_unit(),
            has_lab=True,
        ).values_list('pk', flat=True)
        return context


class
ObjectClassDatatablesView(UnitAdminMixin,
DatatablesView):
    model = Class
    fields = {
        'title': 'course__title',
        'name': 'course__name',
        'additional_name': 'additional_name',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs = super(ObjectClassDatatablesView,
self).get_queryset()
        self.semester =
get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        return qs.filter(
            course__unit=self.get_unit(),
            semester=self.semester,
            lecture=None
        ).order_by('course__title')


class ClassCreateView(UnitAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = ClassForm

    def dispatch(self, request, *args, **kwargs):
        self.semester =
get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        return super(
            ClassCreateView,
self).dispatch(request, *args, **kwargs)

    def get_form_kwargs(self):
        kwargs = super(ClassCreateView,
self).get_form_kwargs()
        kwargs.update({'semester': self.semester})
        return kwargs

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_list')
        return reverse('units:class_create')

    def get_context_data(self, **kwargs):
        context = super(ClassCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Class'
        return context


class
ClassDeleteView(UnitAdminWithObjectMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = Class

    def get_related_unit(self):
        class_obj = self.get_object()
        return class_obj.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_list')
        return reverse('units:class_list')

    def get_context_data(self, **kwargs):
        context = super(ClassDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Class'
        return context


class ClassDetailView(UnitAdminMixin,
TemplateView):
    template_name = 'units/class_details.html'

    def get_context_data(self, **kwargs):
        context = super(ClassDetailView,
self).get_context_data(**kwargs)
        selected_class = get_object_or_404(Class,
pk=self.kwargs['pk'])
        lecture_class = selected_class if
selected_class.lecture is None else
selected_class.lecture
        class_list = Class.objects.filter(
            Q(lecture=lecture_class) |
            Q(pk=lecture_class.pk)
        ).select_related('course')
        class_pk_list = class_list.values_list('pk',
flat=True)
        context['class_list'] = class_list
        context['class_pk_list'] = class_pk_list
        context['selected_class'] = lecture_class
        context['preference_level_list'] =
json.dumps(dict(ClassTimePreference.PREFE
RENCE_LEVEL_CHOICES),
cls=LazyEncoder)
        return context
```

```
class
SemesterClassesCopyView(UnitAdminFormM
ixin, JSONResponseMixin,
AjaxTemplateMixin, FormView):
    template_name = 'base_form.html'
    form_class = SemesterClassesCopyForm

    def get_form_kwargs(self):
        kwargs =
super(SemesterClassesCopyView,
self).get_form_kwargs()
        self.semester =
get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        kwargs.update({'semester': self.semester})
        return kwargs

    def form_valid(self, form):
        form.save()
        return redirect(self.get_success_url())

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_list')
        return reverse('units:class_list')

    def get_context_data(self, **kwargs):
        context =
super(SemesterClassesCopyView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Copy All Classes
From Semester'
        return context


class
ClassSlotsUpdateView(UnitAdminWithObject
Mixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = Class
    form_class = ClassSlotsForm

    def get_related_unit(self):
        self.course_class = self.get_object()
        return self.course_class.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context = super(ClassSlotsUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Class
Slots'
        return context


class
ObjectClassPreferredRoomsDatatablesView(U
nitAdminMixin, DatatablesView):
    model = ClassRoomPreference
    fields = {
        'name': 'room__name',
        'abbrev': 'room__abbrev',
        'capacity': 'room__capacity',
        'commited': 'commited',
        'preference_level': 'preference_level',
        'class_pk': 'course_class__pk',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectClassPreferredRoomsDatatablesVi
ew, self).get_queryset()
        return qs.filter(

course_class__pk=self.kwargs['class_pk'],
        )


class
PreferredRoomCreateView(UnitAdminFormMi
xin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = PreferredRoomForm

    def get_form_kwargs(self):
        kwargs =
super(PreferredRoomCreateView,
self).get_form_kwargs()
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        kwargs.update({'course_class':
self.course_class})
        return kwargs

    def get_related_unit(self):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return self.course_class

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
```

```
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredRoomCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Preferred Room'
        context['aditional_attributes'] = 'data-
pk=room-' + str(self.course_class.pk)
        return context


class
PreferredRoomGroupCreateView(UnitAdminF
ormMixin, JSONResponseMixin,
AjaxTemplateMixin, FormView):
    template_name = 'base_form.html'
    form_class = PreferredRoomGroupForm

    def get_form_kwargs(self):
        kwargs =
super(PreferredRoomGroupCreateView,
self).get_form_kwargs()
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        kwargs.update({'course_class':
self.course_class})
        return kwargs

    def form_valid(self, form):
        form.save()
        return redirect(self.get_success_url())

    def get_related_unit(self):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return self.course_class

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredRoomGroupCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add Room Group
to Preferred Rooms'
        context['aditional_attributes'] = 'data-
pk=room-' + str(self.course_class.pk)
        return context


class
PreferredRoomUpdateView(UnitAdminWithO
bjectMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = ClassRoomPreference
    form_class = PreferredRoomForm

    def get_form_kwargs(self):
        kwargs =
super(PreferredRoomUpdateView,
self).get_form_kwargs()
        kwargs.update({'course_class':
self.get_object().course_class})
        return kwargs

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.course_class.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.obj.course_class.pk})
        return
reverse('units:preferred_room_update',
kwargs={'pk': self.obj.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredRoomUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Room
Preference'
        context['aditional_attributes'] = 'data-
pk=room-' + str(self.obj.course_class.pk)
        return context


class
PreferredRoomDeleteView(UnitAdminWithOb
jectMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = ClassRoomPreference

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.course_class.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.obj.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.obj.course_class.pk})
```

```
    def get_context_data(self, **kwargs):
        context =
super(PreferredRoomDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Room
From Preference'
        context['aditional_attributes'] = 'data-
pk=room-' + str(self.obj.course_class.pk)
        return context


class
ObjectClassPreferredTimesDatatablesView(Un
itAdminMixin, DatatablesView):
    model = ClassTimePreference
    fields = {
        'start': 'time_slot__start',
        'end': 'time_slot__end',
        'mondays': 'time_slot__mondays',
        'tuesdays': 'time_slot__tuesdays',
        'wednesdays': 'time_slot__wednesdays',
        'thursdays': 'time_slot__thursdays',
        'fridays': 'time_slot__fridays',
        'saturdays': 'time_slot__saturdays',
        'sundays': 'time_slot__sundays',
        'preference_level': 'preference_level',
        'commited': 'commited',
        'class_pk': 'course_class__pk',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectClassPreferredTimesDatatablesVi
ew, self).get_queryset()
        return qs.filter(

course_class__pk=self.kwargs['class_pk'],
        )


class
PreferredTimeCreateView(UnitAdminFormMi
xin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = PreferredTimeForm

    def get_form_kwargs(self):
        kwargs =
super(PreferredTimeCreateView,
self).get_form_kwargs()
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        kwargs.update({'course_class':
self.course_class})
        return kwargs

    def get_related_unit(self):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return self.course_class

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredTimeCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Preferred Time'
        context['aditional_attributes'] = 'data-
pk=time-' + str(self.course_class.pk)
        return context


class
PreferredTimeSlotGroupCreateView(UnitAdmi
nFormMixin, JSONResponseMixin,
AjaxTemplateMixin, FormView):
    template_name = 'base_form.html'
    form_class = PreferredTimeSlotGroupForm

    def get_form_kwargs(self):
        kwargs =
super(PreferredTimeSlotGroupCreateView,
self).get_form_kwargs()
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        kwargs.update({'course_class':
self.course_class})
        return kwargs

    def form_valid(self, form):
        form.save()
        return redirect(self.get_success_url())

    def get_related_unit(self):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return self.course_class

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
```

```
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredTimeSlotGroupCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add Time Group
to Preferred Times'
        return context


class
PreferredTimeDeleteView(UnitAdminWithObj
ectMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = ClassTimePreference

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.course_class.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.obj.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.obj.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredTimeDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Time From
Preference'
        return context


class
PreferredTimeUpdateView(UnitAdminWithOb
jectMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = ClassTimePreference
    form_class = PreferredTimeForm

    def get_form_kwargs(self):
        kwargs =
super(PreferredTimeUpdateView,
self).get_form_kwargs()
        kwargs.update({'course_class':
self.get_object().course_class})
        return kwargs

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.course_class.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.obj.course_class.pk})
        return
reverse('units:preferred_time_update',
kwargs={'pk': self.obj.pk})

    def get_context_data(self, **kwargs):
        context =
super(PreferredTimeUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Time
Preference'
        context['aditional_attributes'] = 'data-
pk=time-' + str(self.obj.course_class.pk)
        return context


class
ClassConstraintListView(SemesterObjectMixin
, UnitAdminMixin, TemplateView):
    template_name =
'units/class_constraint_list.html'

    def get_context_data(self, **kwargs):
        context = super(ClassConstraintListView,
self).get_context_data(**kwargs)
        context['constraint_choice_list'] =
json.dumps(dict(ClassConstraint.CONSTRAIN
T_CHOICES), cls=LazyEncoder)
        context['preference_level_list'] =
json.dumps(dict(ClassConstraint.PREFERENC
E_LEVEL_CHOICES), cls=LazyEncoder)
        context['constraint_help_text_list'] =
json.dumps(dict(ClassConstraint.CONSTRAIN
T_HELP_TEXT_LIST), cls=LazyEncoder)
        context['active_tab_parent'] = 'classes'
        context['active_tab'] = 'class_constraints'
        return context


class
ObjectClassConstraintDatatablesView(UnitAd
minMixin, DatatablesView):
    model = ClassConstraint
    fields = {
        'constraint': 'constraint',
        'classes_display': 'classes_display',
        'preference_level': 'preference_level',
        'department': 'unit__department__abbrev',
        'unit': 'unit__abbrev',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectClassConstraintDatatablesView,
self).get_queryset()
        self.semester =
get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        return qs.filter(

unit__department__college=self.get_unit().dep
artment.college,
            semester=self.semester,
        )


class
ClassConstraintCreateView(UnitAdminFormM
ixin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = ClassConstraintForm

    def dispatch(self, request, *args, **kwargs):
        self.semester =
get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        return super(
            ClassConstraintCreateView,
self).dispatch(request, *args, **kwargs)

    def get_form_kwargs(self):
        kwargs =
super(ClassConstraintCreateView,
self).get_form_kwargs()
        kwargs.update({'semester': self.semester})
        return kwargs

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:class_constraint_list')
        return
reverse('units:class_constraint_create')

    def get_context_data(self, **kwargs):
        context =
super(ClassConstraintCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Class
Constraint'
        return context


class
ClassConstraintDeleteView(UnitAdminMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = ClassConstraint

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:class_constraint_list')
        return reverse('units:class_constraint_list')

    def get_context_data(self, **kwargs):
        context =
super(ClassConstraintDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Class
Constraint'
        return context


class
ClassConstraintUpdateView(UnitAdminMixin,
JSONResponseMixin, AjaxTemplateMixin,
UpdateView):
    template_name = 'base_form.html'
    model = ClassConstraint
    form_class = ClassConstraintForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:class_constraint_list')
        return
reverse('units:class_constraint_update')

    def get_context_data(self, **kwargs):
        context =
super(ClassConstraintUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Class
Constraint'
        return context


class
ObjectClassInstructorsDatatablesView(UnitAd
minMixin, DatatablesView):
    model = Instructor
    fields = {
        'first_name': 'first_name',
        'middle_initials': 'middle_initials',
        'last_name': 'last_name',
        'faculty_code': 'faculty_code',
        'email': 'email',
        'pk': 'pk',
    }

    def dispatch(self, request, *args, **kwargs):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return super(
        ObjectClassInstructorsDatatablesView,
self).dispatch(request, *args, **kwargs)

    def get_row(self, row):
        '''Add class pk'''
        row.update({
            'class_pk': self.course_class.pk
        })

        updated_fields = dict(self.fields)
        updated_fields.update({'class_pk':
'class_pk'})

        if isinstance(self.fields, dict):
            return dict([
                (key, text_type(value).format(**row)
if RE_FORMATTED.match(value) else
row[value])
                for key, value in
updated_fields.items()
            ])
        else:
            return [text_type(field).format(**row) if
RE_FORMATTED.match(field)
                    else row[field]
                    for field in updated_fields]

    def get_queryset(self):
        return self.course_class.instructors.all()


class
ClassInstructorCreateView(UnitAdminFormMi
xin, JSONResponseMixin,
AjaxTemplateMixin, FormView):
    template_name = 'base_form.html'
    form_class = ClassInstructorForm

    def get_form_kwargs(self):
        kwargs =
super(ClassInstructorCreateView,
self).get_form_kwargs()
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        kwargs.update({'course_class':
self.course_class})
        return kwargs

    def form_valid(self, form):
        form.save()
        return redirect(self.get_success_url())

    def get_related_unit(self):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return self.course_class.course.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(ClassInstructorCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Class
Instructor'
        return context


class
ClassInstructorDeleteView(UnitAdminWithOb
jectMixin, JSONResponseMixin,
AjaxTemplateMixin, TemplateView):
    template_name = 'base_delete_form.html'

    def get_related_unit(self):
        self.course_class =
get_object_or_404(Class,
pk=self.kwargs['class_pk'])
        return self.course_class.course.unit

    def get(self, request, *args, **kwargs):
        context = self.get_context_data(**kwargs)
        return self.render_to_response(context)

    def post(self, request, *args, **kwargs):
        instructor = get_object_or_404(Instructor,
pk=self.kwargs['instructor_pk'])

self.course_class.instructors.remove(instructor)
        return super(ClassInstructorDeleteView,
self).get(request, *args, **kwargs)

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})
        return reverse('units:class_details',
kwargs={'pk': self.course_class.pk})

    def get_context_data(self, **kwargs):
        context =
super(ClassInstructorDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Class
Instructor'
        context['aditional_attributes'] = 'data-
pk=instructor-' + str(self.course_class.pk)
        return context
```

```
class RoomGroupListView(UnitAdminMixin,
TemplateView):
    template_name =
'units/room_group_list.html'

    def get_context_data(self, **kwargs):
        context = super(RoomGroupListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'groups'
        context['active_tab'] = 'room_groups'
        return context


class
ObjectRoomGroupDatatablesView(UnitAdmin
Mixin, DatatablesView):
    model = RoomGroup
    fields = {
        'name': 'name',
        'description': 'description',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectRoomGroupDatatablesView,
self).get_queryset()
        return qs.filter(

unit__department=self.get_unit().department,
        )


class
RoomGroupCreateView(UnitAdminFormMixi
n, JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = RoomGroupForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:room_group_list')
        return reverse('units:room_group_create')

    def get_context_data(self, **kwargs):
        context = super(RoomGroupCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Room
Group'
        return context


class
RoomGroupUpdateView(DepartmentobjectMi
xin, UnitAdminMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = RoomGroup
    form_class = RoomGroupForm

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:room_group_list')
        return reverse('units:room_group_update')

    def get_context_data(self, **kwargs):
        context = super(RoomGroupUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Room
Group'
        return context


class
RoomGroupDeleteView(DepartmentobjectMix
in, UnitAdminMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = RoomGroup

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:room_group_list')
        return reverse('units:room_group_list')

    def get_context_data(self, **kwargs):
        context = super(RoomGroupDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Room
Group'
        return context


class
RoomGroupDetailView(UnitAdminMixin,
TemplateView):
    template_name =
'units/room_group_details.html'

    def get_context_data(self, **kwargs):
        context = super(RoomGroupDetailView,
self).get_context_data(**kwargs)
        room_group =
get_object_or_404(RoomGroup,
pk=self.kwargs['pk'])
        context['room_group'] = room_group
        context['preference_level_list'] =
json.dumps(dict(RoomGroupMember.PREFER
ENCE_LEVEL_CHOICES),
cls=LazyEncoder)
        return context


class
ObjectRoomGroupMemberDatatablesView(Un
itAdminMixin, DatatablesView):
    model = RoomGroupMember
    fields = {
        'name': 'room__name',
        'abbrev': 'room__abbrev',
        'capacity': 'room__capacity',
        'preference_level': 'preference_level',
        'room_group_pk': 'room_group__pk',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectRoomGroupMemberDatatablesVi
ew, self).get_queryset()
        return qs.filter(

room_group__pk=self.kwargs['room_group_pk
'],
        )


class
RoomGroupMemberCreateView(UnitAdminFo
rmMixin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = RoomGroupMemberForm

    def get_form_kwargs(self):
        kwargs =
super(RoomGroupMemberCreateView,
self).get_form_kwargs()
        self.room_group =
get_object_or_404(RoomGroup,
pk=self.kwargs['room_group_pk'])
        kwargs.update({'room_group':
self.room_group})
        return kwargs

    def get_related_unit(self):
        self.room_group =
get_object_or_404(Class,
pk=self.kwargs['room_group_pk'])
        return self.room_group.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:room_group_details',
kwargs={'pk': self.room_group.pk})
        return reverse('units:room_group_details',
kwargs={'pk': self.room_group.pk})

    def get_context_data(self, **kwargs):
        context =
super(RoomGroupMemberCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Room
to Group'
        return context


class
RoomGroupMemberUpdateView(UnitAdmin
WithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = RoomGroupMember
    form_class = RoomGroupMemberForm

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.room_group.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:room_group_details',
kwargs={'pk': self.obj.room_group.pk})
        return
reverse('units:room_group_member_update',
kwargs={'pk': self.obj.pk})

    def get_context_data(self, **kwargs):
        context =
super(RoomGroupMemberUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Room
Group'
        context['aditional_attributes'] = 'data-
pk=room-' + str(self.obj.room_group.pk)
        return context


class
RoomGroupMemberDeleteView(UnitAdminW
ithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = RoomGroupMember

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.room_group.unit
    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:room_group_details',
kwargs={'pk': self.obj.room_group.pk})
        return reverse('units:room_group_details',
kwargs={'pk': self.obj.room_group.pk})

    def get_context_data(self, **kwargs):
        context =
super(RoomGroupMemberDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Room
From Group'
        context['aditional_attributes'] = 'data-
pk=room-' + str(self.obj.room_group.pk)
        return context


class InstructorListView(UnitAdminMixin,
TemplateView):
    template_name = 'units/instructor_list.html'

    def get_context_data(self, **kwargs):
        context = super(InstructorListView,
self).get_context_data(**kwargs)
        context['active_tab'] = 'isntructors'
        return context


class
ObjectInstructorDatatablesView(UnitAdminMi
xin, DatatablesView):
    model = Instructor
    fields = {
        'first_name': 'first_name',
        'middle_initials': 'middle_initials',
        'last_name': 'last_name',
        'email': 'email',
        'faculty_code': 'faculty_code',
        'unit': 'unit__abbrev',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectInstructorDatatablesView,
self).get_queryset()
        return qs.filter(unit=self.get_unit())


class
InstructorCreateView(UnitAdminFormMixin,
JSONResponseMixin, AjaxTemplateMixin,
CreateView):
    template_name = 'base_form.html'
    form_class = InstructorForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:instructor_list')
        return reverse('units:instructor_create')

    def get_context_data(self, **kwargs):
        context = super(InstructorCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Instructor'
        return context


class
InstructorDeleteView(UnitAdminWithObjectM
ixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = Instructor

    def get_related_unit(self):
        instructor = self.get_object()
        return instructor.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:instructor_list')
        return reverse('units:instructor_list')

    def get_context_data(self, **kwargs):
        context = super(InstructorDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Instructor'
        return context


class
InstructorUpdateView(UnitAdminWithObject
Mixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = Instructor
    form_class = InstructorForm

    def get_related_unit(self):
        instructor = self.get_object()
        return instructor.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('units:instructor_list')
        return reverse('units:instructor_update')

    def get_context_data(self, **kwargs):
        context = super(InstructorUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Instructor'
        return context
```

```
class
InstructorPohibitedTimeSlotListView(Semester
ObjectMixin, UnitAdminMixin,
TemplateView):
    template_name =
'units/instructor_prohibited_time_slot_list.html'

    def get_context_data(self, **kwargs):
        context =
super(InstructorPohibitedTimeSlotListView,
self).get_context_data(**kwargs)
        instructor = get_object_or_404(Instructor,
pk=self.kwargs['pk'])
        context['instructor'] = instructor
        return context


class
ObjectInstructorPohibitedTimeSlotDatatablesV
iew(UnitAdminMixin, DatatablesView):
    model = InstructorProhibitedTimeSlot
    fields = {
        'start': 'time_slot__start',
        'end': 'time_slot__end',
        'mondays': 'time_slot__mondays',
        'tuesdays': 'time_slot__tuesdays',
        'wednesdays': 'time_slot__wednesdays',
        'thursdays': 'time_slot__thursdays',
        'fridays': 'time_slot__fridays',
        'saturdays': 'time_slot__saturdays',
        'sundays': 'time_slot__sundays',
        'pk': 'pk',
    }

    def get_queryset(self):
        instructor = get_object_or_404(Instructor,
pk=self.kwargs['pk'])
        semester = get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        prohibited_time_slot_qs =
InstructorProhibitedTimeSlot.objects.filter(
            instructor=instructor,
            semester=semester,
        )

        return prohibited_time_slot_qs


class
InstructorPohibitedTimeSlotCreateView(UnitA
dminFormMixin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class =
InstructorProhibitedTimeSlotForm

    def dispatch(self, request, *args, **kwargs):
        self.instructor =
get_object_or_404(Instructor,
pk=self.kwargs['pk'])
        self.semester =
get_object_or_404(Semester,
pk=self.kwargs['semester_pk'])
        return super(

InstructorPohibitedTimeSlotCreateView,
self).dispatch(request, *args, **kwargs)

    def get_form_kwargs(self):
        kwargs =
super(InstructorPohibitedTimeSlotCreateView,
self).get_form_kwargs()
        kwargs.update({
            'instructor': self.instructor,
            'semester': self.semester,
        })
        return kwargs

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:instructor_prohibited_time_slot_l
ist', kwargs={'pk': self.instructor.pk})
        return
reverse('units:instructor_prohibited_time_slot_
create', kwargs={'pk': self.instructor.pk})

    def get_context_data(self, **kwargs):
        context =
super(InstructorPohibitedTimeSlotCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Pohibited Time Slot for Instructor'
        return context


class
InstructorPohibitedTimeSlotDeleteView(UnitA
dminWithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, TemplateView):
    template_name = 'base_delete_form.html'

    def dispatch(self, request, *args, **kwargs):
        self.prohibited_time_slot =
get_object_or_404(InstructorProhibitedTimeSl
ot, pk=self.kwargs['pk'])
        return super(

InstructorPohibitedTimeSlotDeleteView,
self).dispatch(request, *args, **kwargs)

    def get_related_unit(self):
        return
self.prohibited_time_slot.instructor.unit
```

```
    def get(self, request, *args, **kwargs):
        context = self.get_context_data(**kwargs)
        return self.render_to_response(context)

    def post(self, request, *args, **kwargs):
        self.prohibited_time_slot.delete()
        return
super(InstructorPohibitedTimeSlotDeleteView,
self).get(request, *args, **kwargs)

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:instructor_prohibited_time_slot_l
ist', kwargs={'pk': self.instructor.pk})
        return
reverse('units:instructor_prohibited_time_slot_l
ist', kwargs={'pk': self.instructor.pk})

    def get_context_data(self, **kwargs):
        context =
super(InstructorPohibitedTimeSlotDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Pohibited
Time Slot for Instructor'
        return context


class
TimeSlotGroupListView(UnitAdminMixin,
TemplateView):
    template_name =
'units/time_slot_group_list.html'

    def get_context_data(self, **kwargs):
        context = super(TimeSlotGroupListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'groups'
        context['active_tab'] = 'time_slot_groups'
        return context


class
ObjectTimeSlotGroupDatatablesView(UnitAd
minMixin, DatatablesView):
    model = TimeSlotGroup
    fields = {
        'name': 'name',
        'description': 'description',
        'time_slot_duration': 'time_slot_duration',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectTimeSlotGroupDatatablesView,
self).get_queryset()
        return qs.filter(

unit__department=self.get_unit().department,
        )


class
TimeSlotGroupCreateView(UnitAdminFormM
ixin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = TimeSlotGroupForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:time_slot_group_list')
        return
reverse('units:time_slot_group_create')

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Time
Slot Group'
        return context


class
TimeSlotGroupUpdateView(Departmentobject
Mixin, UnitAdminMixin,
JSONResponseMixin, AjaxTemplateMixin,
UpdateView):
    template_name = 'base_form.html'
    model = TimeSlotGroup
    form_class = TimeSlotGroupForm

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:time_slot_group_list')
        return
reverse('units:time_slot_group_update')

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Time Slot
Group'
        return context
```

```
class
TimeSlotGroupDeleteView(Departmentobject
Mixin, UnitAdminMixin,
JSONResponseMixin, AjaxTemplateMixin,
DeleteView):
    template_name = 'base_delete_form.html'
    model = TimeSlotGroup

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:time_slot_group_list')
        return reverse('units:time_slot_group_list')

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Time Slot
Group'
        return context


class
TimeSlotGroupDetailView(UnitAdminMixin,
TemplateView):
    template_name =
'units/time_slot_group_details.html'

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupDetailView,
self).get_context_data(**kwargs)
        time_slot_group =
get_object_or_404(TimeSlotGroup,
pk=self.kwargs['pk'])
        context['time_slot_group'] =
time_slot_group
        context['preference_level_list'] =
json.dumps(dict(TimeSlotGroupMember.PREF
ERENCE_LEVEL_CHOICES),
cls=LazyEncoder)
        return context


class
ObjectTimeSlotGroupMemberDatatablesView(
UnitAdminMixin, DatatablesView):
    model = TimeSlotGroupMember
    fields = {
        'start': 'time_slot__start',
        'end': 'time_slot__end',
        'mondays': 'time_slot__mondays',
        'tuesdays': 'time_slot__tuesdays',
        'wednesdays': 'time_slot__wednesdays',
        'thursdays': 'time_slot__thursdays',
        'fridays': 'time_slot__fridays',
        'saturdays': 'time_slot__saturdays',
        'sundays': 'time_slot__sundays',
        'preference_level': 'preference_level',
        'time_slot_group_pk':
'time_slot_group__pk',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectTimeSlotGroupMemberDatatables
View, self).get_queryset()
        return qs.filter(

time_slot_group__pk=self.kwargs['time_slot_g
roup_pk'],
        )


class
TimeSlotGroupMemberCreateView(UnitAdmi
nFormMixin, JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = TimeSlotGroupMemberForm

    def get_form_kwargs(self):
        kwargs =
super(TimeSlotGroupMemberCreateView,
self).get_form_kwargs()
        self.time_slot_group =
get_object_or_404(TimeSlotGroup,
pk=self.kwargs['time_slot_group_pk'])
        kwargs.update({'time_slot_group':
self.time_slot_group})
        return kwargs

    def get_related_unit(self):
        self.time_slot_group =
get_object_or_404(Class,
pk=self.kwargs['time_slot_group_pk'])
        return self.time_slot_group.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:time_slot_group_details',
kwargs={'pk': self.time_slot_group.pk})
        return
reverse('units:time_slot_group_details',
kwargs={'pk': self.time_slot_group.pk})

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupMemberCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New Time
Slot to Group'
        return context


class
TimeSlotGroupMemberUpdateView(UnitAdmi
nWithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = TimeSlotGroupMember
    form_class = TimeSlotGroupMemberForm

    def get_form_kwargs(self):
        kwargs =
super(TimeSlotGroupMemberUpdateView,
self).get_form_kwargs()
        kwargs.update({'time_slot_group':
self.obj.time_slot_group})
        return kwargs

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.time_slot_group.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:time_slot_group_details',
kwargs={'pk': self.obj.time_slot_group.pk})
        return
reverse('units:time_slot_group_member_update
', kwargs={'pk': self.obj.pk})

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupMemberUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Time Slot
Group'
        context['aditional_attributes'] = 'data-
pk=time-' + str(self.obj.time_slot_group.pk)
        return context


class
TimeSlotGroupMemberDeleteView(UnitAdmi
nWithObjectMixin, JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = TimeSlotGroupMember

    def get_related_unit(self):
        self.obj = self.get_object()
        return self.obj.time_slot_group.unit

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('units:time_slot_group_details',
kwargs={'pk': self.obj.time_slot_group.pk})
        return
reverse('units:time_slot_group_details',
kwargs={'pk': self.obj.time_slot_group.pk})

    def get_context_data(self, **kwargs):
        context =
super(TimeSlotGroupMemberDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Time Slot
From Group'
        context['aditional_attributes'] = 'data-
pk=time-' + str(self.obj.time_slot_group.pk)
        return context
```

88. upmcts/university/forms.py

```
from django import forms
from django.utils.translation import
ugettext_lazy as _

from crispy_forms.helper import FormHelper
from django_select2.forms import
Select2Widget

from upmcts.colleges.models import College,
CollegeAdmin
from upmcts.users.forms import
UserCreateForm
from upmcts.users.models import User

from .models import Semester


class CollegeForm(forms.ModelForm):

    class Meta:
        model = College
        fields = ('name', 'abbrev',)

    def __init__(self, *args, **kwargs):
        super(CollegeForm, self).__init__(*args,
**kwargs)
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class SemesterForm(forms.ModelForm):

    class Meta:
        model = Semester
        fields = ('academic_year', 'sem',)
    def __init__(self, *args, **kwargs):
        super(SemesterForm, self).__init__(*args,
**kwargs)
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class
CollegeAdminCreateForm(UserCreateForm):
    college =
forms.ModelChoiceField(queryset=College.obj
ects.all())

    class Meta:
        model = User
        fields = (
            'username', 'password1', 'password2',
'college',
            'email', 'first_name', 'last_name', )


class
CollegeAdminUpdateForm(forms.ModelForm)
:
    username = forms.CharField(required=True,
max_length=30)
    first_name =
forms.CharField(required=False,
max_length=30)
    last_name =
forms.CharField(required=False,
max_length=30)
    email = forms.EmailField(required=False,
max_length=254)

    class Meta:
        model = CollegeAdmin
        fields = (
            'username', 'college', 'email',
'first_name', 'last_name',)

    def __init__(self, *args, **kwargs):
        super(CollegeAdminUpdateForm,
self).__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'
        self.fields['username'].initial =
self.instance.user.username
        self.fields['email'].initial =
self.instance.user.email
        self.fields['first_name'].initial =
self.instance.user.first_name
        self.fields['last_name'].initial =
self.instance.user.last_name

    def clean_username(self):
        username =
self.cleaned_data.get('username')
        try:
            existing_user =
User.objects.get(username=username)
            if not existing_user.pk ==
self.instance.user.pk:
                raise forms.ValidationError(
                    'Username already taken. Choose a
different one.')
        except User.DoesNotExist:
            pass
        return username

    def save(self, commit=True, *args,
**kwargs):
        admin = super(
            CollegeAdminUpdateForm,
            self).save(commit=False, *args,
**kwargs)
        admin.user.username =
self.cleaned_data.get('username')
        admin.user.email =
self.cleaned_data.get('email')
        admin.user.first_name =
self.cleaned_data.get('first_name')
        admin.user.last_name =
self.cleaned_data.get('last_name')

        if commit:
            admin.user.save()
            admin.save()
        return admin


class SemesterDropdownForm(forms.Form):
    semester = forms.ModelChoiceField(
        queryset=Semester.objects.all(),
        required=True,
        empty_label=None,
        # widget=Select2Widget
    )

    def __init__(self, semester, *args,
**kwargs):
        super(SemesterDropdownForm,
self).__init__(*args, **kwargs)
        self.fields['semester'].initial = semester
        self.fields['semester'].label = ''

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.disable_csrf = True
        self.helper.template_pack = 'bootstrap3'
```

```
class
AssignmentStatusDropdownForm(forms.Form)
:
    STATUS_CHOICES = (
        (0, _("Assigned and Unassigned
Classes")),
        (1, _("Assigned Classes")),
        (2, _("Unassigned Classes")),
    )

    status = forms.ChoiceField(
        choices=STATUS_CHOICES,
        required=True,
        widget=Select2Widget,
    )

    def __init__(self, status, *args, **kwargs):
        super(AssignmentStatusDropdownForm,
self).__init__(*args, **kwargs)
        self.fields['status'].initial = status
        self.fields['status'].label = ''

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.disable_csrf = True
        self.helper.template_pack = 'bootstrap3'
```

89. upmcts/university/models.py

```
from django.db import models
from django.utils.translation import
ugettext_lazy as _

from model_utils import Choices


class Semester(models.Model):

    SEM_CHOICES = Choices(
        ('First Semester', _(u'First Semester')),
        ('Second Semester', _(u'Second
Semester')),
        ('Short Term', _(u'Short Term')),
    )

    academic_year = models.CharField(
        max_length=9,
        help_text=_(u'ex. 2014-2015')
    )
    sem = models.CharField(
        _(u'semester'),
        max_length=15,
choices=SEM_CHOICES,
        default='First Semester')

    class Meta:
        unique_together = ('academic_year', 'sem',
)
        ordering = ('-academic_year', )

    def __unicode__(self):
        return '{},
{}'.format(self.get_sem_display(),
self.academic_year)
```

90. upmcts/university/urls.py

```
from django.conf.urls import url

from . import views


urlpatterns = [

    url(
        r'^colleges/$',
        views.CollegeListView.as_view(),
        name='college_list'
    ),
    url(
        r'^colleges/objects/$',

views.ObjectCollegeDatatablesView.as_view(),
        name='DT_college_objects'
    ),
    url(
        r'^colleges/new/$',
        views.CollegeCreateView.as_view(),
        name='college_create'
    ),
    url(
        r'^colleges/(?P<pk>\d+)/delete/$',
        views.CollegeDeleteView.as_view(),
        name='college_delete'
    ),
    url(r'^colleges/(?P<pk>\d+)/update/$',
        views.CollegeUpdateView.as_view(),
        name='college_update'),

    url(
        r'^semesters/$',
        views.SemesterListView.as_view(),
        name='semester_list'
    ),
    url(
        r'^semesters/objects/$',

views.ObjectSemesterDatatablesView.as_view(
),
        name='DT_semester_objects'
    ),
    url(
        r'^semesters/new/$',
        views.SemesterCreateView.as_view(),
        name='semester_create'
    ),
    url(
        r'^semesters/(?P<pk>\d+)/delete/$',
        views.SemesterDeleteView.as_view(),
        name='semester_delete'
    ),
    url(
        r'^semesters/(?P<pk>\d+)/update/$',
        views.SemesterUpdateView.as_view(),
        name='semester_update'
    ),

    url(
        r'^college_admins/$',
        views.CollegeAdminListView.as_view(),
        name='college_admin_list'
    ),
    url(
        r'^college_admins/objects/$',

views.ObjectCollegeAdminDatatablesView.as_
view(),
        name='DT_college_admin_objects'
    ),
    url(
        r'^college_admins/new/$',

views.CollegeAdminCreateView.as_view(),
        name='college_admin_create'
    ),
    url(
        r'^college_admins/(?P<pk>\d+)/delete/$',

views.CollegeAdminDeleteView.as_view(),
        name='college_admin_delete'
    ),
    url(
        r'^college_admins/(?P<pk>\d+)/update/$',

views.CollegeAdminUpdateView.as_view(),
        name='college_admin_update'
    ),

    url(

r'^timetables/college/(?P<college_pk>\d+)/$',

views.CollegeTimetableListView.as_view(),
        name='college_timetable_list'
    ),

    url(

r'^timetables/(?P<college_pk>\d+)/objects/$',

views.ObjectCollegeTimetableDatatablesView.
as_view(),
        name='DT_college_timetable_objects'
    ),

    url(

r'^timetables/(?P<timetable_pk>\d+)/output/$',

views.CollegeTimetableOutputView.as_view(),
        name='college_timetable_output'
    ),

    url(
        r'^timetables/output/pdf/$',

views.CollegeTimetablePDFOutputView.as_vi
ew(),
        name='college_timetable_pdf_output'
    ),

    url(
        r'^timetables/$',

views.UniversityTimetableListView.as_view(),
        name='university_timetable_list'
    ),

]
```

91. upmcts/university/views.py

```
import json
import os

from django.conf import settings
from django.contrib.auth.models import Group
from django.core.urlresolvers import reverse
from django.db.models import Q
from django.shortcuts import redirect,
get_object_or_404
from django.views.generic import
TemplateView, CreateView, DeleteView,
UpdateView

from braces.views import
(JSONResponseMixin, LoginRequiredMixin)
from django_tables2_reports.views import
ReportTableView
from easy_pdf.views import
PDFTemplateView
from eztables.views import DatatablesView

from upmcts.colleges.models import College,
CollegeAdmin, Timetable
from upmcts.colleges.tables import
TimetableOutputTable
from upmcts.departments.forms import
DepartmentDropdownForm
from upmcts.departments.models import
Department
from upmcts.users.views import
AjaxTemplateMixin
from upmcts.units.models import Class
from upmcts.utils.utils import LazyEncoder

from .forms import (
    CollegeForm, CollegeAdminCreateForm,
    CollegeAdminUpdateForm, SemesterForm,
AssignmentStatusDropdownForm)
from .models import Semester


class CollegeListView(TemplateView):
    template_name =
'university/college_list.html'

    def get_context_data(self, **kwargs):
        context = super(CollegeListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'colleges'
        context['active_tab'] = 'colleges'
        return context


class
ObjectCollegeDatatablesView(DatatablesView
):
    model = College
    fields = {
        'abbrev': 'abbrev',
        'name': 'name',
        'pk': 'pk',
    }


class
CollegeCreateView(JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = CollegeForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('university:college_list')
        return reverse('university:college_create')

    def get_context_data(self, **kwargs):
        context = super(CollegeCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New College'
        return context


class
CollegeDeleteView(JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = College

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('university:college_list')
        return reverse('university:college_list')

    def get_context_data(self, **kwargs):
        context = super(CollegeDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete College'
        return context


class
CollegeUpdateView(JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = College
    form_class = CollegeForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('university:college_list')
        return reverse('university:college_update')

    def get_context_data(self, **kwargs):
        context = super(CollegeUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update College'
        return context


class SemesterListView(TemplateView):
    template_name =
'university/semester_list.html'

    def get_context_data(self, **kwargs):
        context = super(SemesterListView,
self).get_context_data(**kwargs)
        context['active_tab'] = 'semesters'
        return context


class
ObjectSemesterDatatablesView(DatatablesVie
w):
    model = Semester
    fields = {
        'sem': '{sem}, {academic_year}',
        'pk': 'pk',
    }

    def sort_col_0(self, direction):
        return ('%sacademic_year' % direction,
'%ssem' % direction)


class
SemesterCreateView(JSONResponseMixin,
AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = SemesterForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('university:semester_list')
        return
reverse('university:semester_create')

    def get_context_data(self, **kwargs):
        context = super(SemesterCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New
Semester'
        return context


class
SemesterDeleteView(JSONResponseMixin,
AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = Semester

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('university:semester_list')
        return
reverse('university:semester_create')

    def get_context_data(self, **kwargs):
        context = super(SemesterDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete Semester'
        return context


class
SemesterUpdateView(JSONResponseMixin,
AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = Semester
    form_class = SemesterForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse('university:semester_list')
        return
reverse('university:semester_create')

    def get_context_data(self, **kwargs):
        context = super(SemesterUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update Semester'
        return context


class CollegeAdminListView(TemplateView):
    template_name =
'university/college_admin_list.html'

    def get_context_data(self, **kwargs):
        context = super(CollegeAdminListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'colleges'
        context['active_tab'] = 'college_admins'
        return context


class
ObjectCollegeAdminDatatablesView(Datatable
sView):
    model = CollegeAdmin
    fields = {
        'username': 'user__username',
        'college': 'college__abbrev',
        'email': 'user__email',
        'first_name': 'user__first_name',
        'last_name': 'user__last_name',
        'pk': 'pk',
        'user_pk': 'user__pk',
    }


class
CollegeAdminCreateView(JSONResponseMixi
n, AjaxTemplateMixin, CreateView):
    template_name = 'base_form.html'
    form_class = CollegeAdminCreateForm

    def form_valid(self, form):
        user = form.save()
        college = form.cleaned_data.get('college')
        g =
Group.objects.get(name='CollegeAdmins')
        g.user_set.add(user)

        CollegeAdmin.objects.create(user=user,
college=college)

        return redirect(self.get_success_url())

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('university:college_admin_list')
        return
reverse('university:college_admin_create')

    def get_context_data(self, **kwargs):
        context =
super(CollegeAdminCreateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Add New College
Admin'
        return context
```

```
class
CollegeAdminDeleteView(JSONResponseMixi
n, AjaxTemplateMixin, DeleteView):
    template_name = 'base_delete_form.html'
    model = CollegeAdmin

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('university:college_admin_list')
        return
reverse('university:college_admin_create')

    def get_context_data(self, **kwargs):
        context =
super(CollegeAdminDeleteView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Delete College
Admin'
        return context


class
CollegeAdminUpdateView(JSONResponseMix
in, AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = CollegeAdmin
    form_class = CollegeAdminUpdateForm

    def get_success_url(self):
        if self.request.is_ajax():
            return
reverse('university:college_admin_list')
        return
reverse('university:collegeadmin_create')

    def get_context_data(self, **kwargs):
        context =
super(CollegeAdminUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'Update College
Admin'
        return context


class
UniversityTimetableListView(LoginRequired
Mixin, TemplateView):
    template_name =
'university/university_timetable_list.html'

    def get_context_data(self, **kwargs):
        context =
super(UniversityTimetableListView,
self).get_context_data(**kwargs)
        context['active_tab_parent'] = 'timetables'
        context['active_tab'] =
'university_timetable_list'
        return context


class
CollegeTimetableListView(LoginRequiredMix
in, TemplateView):
    template_name =
'university/college_timetable_list.html'

    def get_context_data(self, **kwargs):
        context =
super(CollegeTimetableListView,
self).get_context_data(**kwargs)
        college = get_object_or_404(College,
pk=self.kwargs['college_pk'])
        context['college'] = college
        context['sem_list'] =
json.dumps(dict(Semester.SEM_CHOICES),
cls=LazyEncoder)
        context['status_list'] =
json.dumps(dict(Timetable.STATUS_CHOICE
S), cls=LazyEncoder)
        context['active_tab_parent'] = 'timetables'
        context['active_tab'] =
'university_timetable_list'
        return context


class
ObjectCollegeTimetableDatatablesView(Login
RequiredMixin, DatatablesView):
    model = Timetable
    fields = {
        'semester_sem': 'semester__sem',
        'semester_academic_year':
'semester__academic_year',
        'status': 'status',
        'assigned_classes_count':
'assigned_classes_count',
        'unassigned_classes_count':
'unassigned_classes_count',
        'pk': 'pk',
    }

    def get_queryset(self):
        qs =
super(ObjectCollegeTimetableDatatablesView,
self).get_queryset()
        college = get_object_or_404(College,
pk=self.kwargs['college_pk'])
        return qs.filter(college=college)


class
CollegeTimetableOutputView(LoginRequired
Mixin, ReportTableView):
    template_name =
'university/college_timetable_output.html'
```

```
    table_class = TimetableOutputTable
    model = Class

    def dispatch(self, request, *args, **kwargs):
        self.timetable =
get_object_or_404(Timetable,
pk=self.kwargs['timetable_pk'])
        self.college = self.timetable.college
        self.semester = self.timetable.semester
        return
super(CollegeTimetableOutputView,
self).dispatch(request, *args, **kwargs)

    def get_table(self, **kwargs):
        filename = self.semester.sem + '__' +
self.semester.academic_year
        filename = filename.replace(" ",
"_").lower()
        kwargs.update({
            'filename': filename,
        })
        return
super(CollegeTimetableOutputView,
self).get_table(**kwargs)

    def get_queryset(self):
        qs = super(CollegeTimetableOutputView,
self).get_queryset()
        qs = qs.filter(
            semester=self.semester,

course__unit__department__college=self.colle
ge,
        )
        department =
self.request.GET.get('department', None)
        if department:
            qs =
qs.filter(course__unit__department=department
)
        status = self.request.GET.get('status', 0)
        if status == '1':
            qs =
qs.exclude(Q(assigned_room=None) |
Q(assigned_time_slot=None))
        elif status == '2':
            qs = qs.filter(Q(assigned_room=None) |
Q(assigned_time_slot=None))
        return qs

    def get_context_data(self, **kwargs):
        context =
super(CollegeTimetableOutputView,
self).get_context_data(**kwargs)
        department =
self.request.GET.get('department', None)
        status = self.request.GET.get('status', 0)
        context['timetable'] = self.timetable
        context['department'] = department
        context['department_form'] =
DepartmentDropdownForm(
            college=self.college,
department=department
        )
        context['status_form'] =
AssignmentStatusDropdownForm(
            status=status
        )
        return context


class
CollegeTimetablePDFOutputView(PDFTempla
teView):
    template_name = "pdf/timetable.html"
    pdf_filename = "timetable.pdf"

    def post(self, request, *args, **kwargs):
        context = self.get_context_data(**kwargs)
        timetable_pk =
request.POST.get('timetable_pk', None)
        department_pk =
request.POST.get('department_pk', None)

        timetable = get_object_or_404(Timetable,
pk=int(timetable_pk))
        department = None
        if department_pk:
            try:
                department =
get_object_or_404(Department,
pk=int(department_pk))
            except Exception as e:
                pass
        class_list = Class.objects.filter(
            semester=timetable.semester,

course__unit__department__college=timetable.
college,
        )
        margin_offset = 0
        if department:
            class_list =
class_list.filter(course__unit__department=dep
artment)
            margin_offset = .5

        context['hr_margin'] = 4 + margin_offset
        context['content_margin'] = 4.5 +
margin_offset

        context['class_list'] = class_list
        context['college'] = timetable.college
        context['semester'] = timetable.semester
        context['department'] = department
        font_file_path = settings.STATIC_ROOT
+ 'fonts/OpenSans-Regular.ttf'
        context['font'] =
os.path.abspath(font_file_path)

        return self.render_to_response(context)
```

92. upmcts/users/admin.py

```
# -*- coding: utf-8 -*-
from __future__ import absolute_import,
unicode_literals

from django import forms
from django.contrib import admin
from django.contrib.auth.admin import
UserAdmin as AuthUserAdmin
from django.contrib.auth.forms import
UserChangeForm, UserCreationForm

from .models import User


class MyUserChangeForm(UserChangeForm):
    class Meta(UserChangeForm.Meta):
        model = User


class
MyUserCreationForm(UserCreationForm):

    error_message =
UserCreationForm.error_messages.update({
        'duplicate_username': 'This username has
already been taken.'
    })

    class Meta(UserCreationForm.Meta):
        model = User

    def clean_username(self):
        username = self.cleaned_data["username"]
        try:
            User.objects.get(username=username)
        except User.DoesNotExist:
            return username
        raise
forms.ValidationError(self.error_messages['dup
licate_username'])


@admin.register(User)
class UserAdmin(AuthUserAdmin):
    form = MyUserChangeForm
    add_form = MyUserCreationForm
```

93. upmcts/users/forms.py

```
# -*- coding: utf-8 -*-
from django import forms
from django.contrib.auth.forms import
UserCreationForm,
AdminPasswordChangeForm
from django.utils.translation import
ugettext_lazy as _

from crispy_forms.helper import FormHelper

from .models import User


class UserSignupForm(forms.Form):

    def __init__(self, *args, **kwargs):

        super(UserSignupForm,
self).__init__(*args, **kwargs)

        if 'password2' in self.fields:

self.fields['password2'].widget.attrs['placeholde
r'] = _('Confirm password')

    def signup(self, request, user):
        pass


class UserCreateForm(UserCreationForm):
    class Meta:
        model = User
        fields = (
            'username', 'first_name',
            'last_name', 'password1', 'password2')

    def __init__(self, *args, **kwargs):
        super(UserCreateForm,
self).__init__(*args, **kwargs)
        self.fields['first_name'].required = False
        self.fields['first_name'].max_length = 30
        self.fields['last_name'].max_length = 30
        self.fields['last_name'].required = False

        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'


class
UserPasswordUpdateForm(AdminPasswordCh
angeForm):
    def __init__(self, *args, **kwargs):
        user = kwargs.pop('instance')
        super(UserPasswordUpdateForm,
self).__init__(user, *args, **kwargs)
        self.helper = FormHelper()
        self.helper.form_tag = False
        self.helper.template_pack = 'bootstrap3'
```

94. upmcts/users/models.py
```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals,
absolute_import

from django.contrib.auth.models import
AbstractUser
from django.core.urlresolvers import reverse
from django.db import models
from django.utils.encoding import
python_2_unicode_compatible
from django.utils.translation import
ugettext_lazy as _


@python_2_unicode_compatible
class User(AbstractUser):

    # First Name and Last Name do not cover
name patterns
    # around the globe.
    # name = models.CharField(_("Name of
User"), blank=True, max_length=255)

    def __str__(self):
        return self.username

    def is_university_admin(self):
        return self.is_member('UniversityAdmins')

    def is_college_admin(self):
        try:
            self.collegeadmin
        except AttributeError:
            return False
        return self.is_member('CollegeAdmins')

    def is_department_admin(self):
        try:
            self.departmentadmin
        except AttributeError:
            return False
        return
self.is_member('DepartmentAdmins')

    def is_unit_admin(self):
        try:
            self.unitadmin
        except AttributeError:
            return False
        return self.is_member('UnitAdmins')


    def is_member(self, group_name):
        return
self.groups.filter(name=group_name).exists()
```

95. upmcts/users/urls.py
```
# -*- coding: utf-8 -*-
from __future__ import absolute_import,
unicode_literals

from django.conf.urls import url

from . import views

urlpatterns = [
    # URL pattern for the UserListView
    url(
        regex=r'^$',
        view=views.UserListView.as_view(),
        name='list'
    ),

    # URL pattern for the UserRedirectView
    url(
        regex=r'^~redirect/$',
        view=views.UserRedirectView.as_view(),
        name='redirect'
    ),

    # URL pattern for the UserDetailView
    url(
        regex=r'^(?P<username>[\w.@+-]+)/$',
        view=views.UserDetailView.as_view(),
        name='detail'
    ),

    # URL pattern for the UserUpdateView
    url(
        regex=r'^~update/$',
        view=views.UserUpdateView.as_view(),
        name='update'
    ),

    url(
        r'^password/(?P<pk>\d+)/update/$',

views.UserPasswordUpdateView.as_view(),
        name='password_update'
    ),
]
```

96. upmcts/users/views.py
```
# -*- coding: utf-8 -*-
from __future__ import absolute_import,
unicode_literals

from django.core.urlresolvers import reverse
from django.views.generic import DetailView,
ListView, RedirectView, UpdateView

from braces.views import
LoginRequiredMixin, JSONResponseMixin
```

```
from upmcts.utils.views import
AjaxTemplateMixin

from .forms import UserPasswordUpdateForm
from .models import User


class UserDetailView(LoginRequiredMixin,
DetailView):
    model = User
    # These next two lines tell the view to index
lookups by username
    slug_field = "username"
    slug_url_kwarg = "username"


class UserRedirectView(LoginRequiredMixin,
RedirectView):
    permanent = False

    def get_redirect_url(self):
        if self.request.user.is_superuser:
            return reverse('university:college_list')
        elif self.request.user.is_college_admin():
            return
reverse('colleges:department_list')
        elif
self.request.user.is_department_admin():
            return reverse('departments:unit_list')
        elif self.request.user.is_unit_admin():
            return reverse('units:class_list')
        return '/'  # User is staff


class UserUpdateView(LoginRequiredMixin,
UpdateView):

    fields = ['first_name', 'last_name']

    # we already imported User in the view code
above, remember?
    model = User

    # send the user back to their own page after a
successful update
    def get_success_url(self):
        return reverse("users:detail",
                       kwargs={"username":
self.request.user.username})

    def get_object(self):
        # Only get the User record for the user
making the request
        return
User.objects.get(username=self.request.user.us
ername)


class UserListView(LoginRequiredMixin,
ListView):
    model = User
    # These next two lines tell the view to index
lookups by username
    slug_field = "username"
    slug_url_kwarg = "username"


class
UserPasswordUpdateView(JSONResponseMix
in, AjaxTemplateMixin, UpdateView):
    template_name = 'base_form.html'
    model = User
    form_class = UserPasswordUpdateForm

    def get_success_url(self):
        if self.request.is_ajax():
            return reverse(
                'users:password_update',
                kwargs={'pk':
self.get_object().pk, }
            )

    def get_context_data(self, **kwargs):
        context =
super(UserPasswordUpdateView,
self).get_context_data(**kwargs)
        context['popup_title'] = 'New Password'
        return context
```

97. upmcts/utils/utils.py

```
import traceback
import datetime
from json import dumps, loads
import errno
import os
import shutil
import subprocess
import xml.dom.minidom

from django.conf import settings
from django.core.files.base import ContentFile
from django.core.serializers.json import
DjangoJSONEncoder
from django.utils import timezone
from django.utils.encoding import force_text
from django.utils.functional import Promise

from upmcts.units.models import Class,
ClassConstraint
from upmcts.colleges.models import Room,
Timetable


class LazyEncoder(DjangoJSONEncoder):
    def default(self, obj):
        if isinstance(obj, Promise):
            return force_text(obj)
        return super(LazyEncoder,
self).default(obj)


def dict2xml(d, root_node=None):
    """
        Simple xml serializer.
        Original author: Reimund Trost 2013
        Reference:
https://gist.github.com/reimund/5435343
    """

    wrap = False if root_node is None or
isinstance(d, list) else True
    root = 'objects' if root_node is None else
root_node
    root_singular = root[:-1] if 's' == root[-1] and
root_node is None else root
    xml = ''
    children = []

    if isinstance(d, dict):
        for key, value in dict.items(d):
            if isinstance(value, dict):
                children.append(dict2xml(value,
key))
            elif isinstance(value, list):
                children.append(dict2xml(value,
key))
            elif isinstance(value, bool):
                xml = xml + ' ' + key + '="' +
str(value).lower() + '"'
            else:
                xml = xml + ' ' + key + '="' +
str(value) + '"'
    else:
        for value in d:
            children.append(dict2xml(value,
root_singular))

    end_tag = '>' if 0 < len(children) else '/>'

    if wrap or isinstance(d, dict):
        xml = '<' + root + xml + end_tag

    if 0 < len(children):
        for child in children:
            xml = xml + child

        if wrap or isinstance(d, dict):
            xml = xml + '</' + root + '>'

    return xml
```

```
def
generate_timetabling_input_data(timetable_obj
):
    timetable_obj.status =
Timetable.GENERATING_INPUT
    timetable_obj.save()
    timetable_obj.assign_class_statuses_count()

    college = timetable_obj.college
    semester = timetable_obj.semester

    rooms = Room.objects.xml_dict(college)
    classes = Class.objects.xml_dict(college,
semester)
    constraints =
ClassConstraint.objects.xml_dict(college,
semester)

    timetable_dict = {
        'version': "2.4",
        'initiative': "UP Manila",
        'term': '20152nd',
        'created': datetime.datetime.now(),
        'nrDays': "7",
        'slotsPerDay': "288",
        'students': {},
    }
    timetable_dict.update({
        'rooms': rooms,
        'classes': classes,
        'groupConstraints': constraints,
    })

    xml_str = dict2xml(timetable_dict,
'timetable')
    xml_obj =
xml.dom.minidom.parseString(xml_str)

    timetable_obj.input_created_date =
timezone.now()
    timetable_obj.save()
    input_path =
timetable_obj.get_filename('input')

    media_path = settings.MEDIA_ROOT + '/' +
input_path

    if not
os.path.exists(os.path.dirname(media_path)):
        try:

os.makedirs(os.path.dirname(media_path))
        except OSError as exc:  # Guard against
race condition
            if exc.errno != errno.EEXIST:
                raise

    timetable_obj.input_timetable.delete()
timetable_obj.input_timetable.save(input_path,
ContentFile(''))
    xml_file = open(settings.MEDIA_ROOT +
'/' + timetable_obj.input_timetable.name, 'w+')
    xml_obj.writexml(
        xml_file,
        indent="  ",
        addindent="  ",
        newl='\n'
    )

    xml_obj.unlink()
    xml_file.close()


def
generate_timetabling_output_data(timetable_o
bj):

    try:
        timetable_obj.status =
Timetable.GENERATING_OUTPUT
        timetable_obj.save()
        input_path = settings.MEDIA_ROOT + '/'
+ timetable_obj.input_timetable.name

        relative_output_filename =
timetable_obj.get_filename('output')
        output_filename =
settings.MEDIA_ROOT + '/' +
relative_output_filename
        output_path =
os.path.splitext(output_filename)[0]

        if not
os.path.exists(os.path.dirname(output_path)):

os.makedirs(os.path.dirname(output_path))
        else:

shutil.rmtree(os.path.dirname(output_path))

os.makedirs(os.path.dirname(output_path))

        #
http://www.unitime.org/uct_execution.php
        subprocess.call(
            [
                'java',
                '-Xmx1g',
                '-cp',

settings.TIMETABLING_SOLVER_ROOT +
'/coursett-1.3.jar:' +
settings.TIMETABLING_SOLVER_ROOT +
'/lib/*',
                'org.cpsolver.coursett.Test',
settings.TIMETABLING_SOLVER_ROOT +
'/config.cfg',
                input_path,
                output_path
            ]
        )
        # Get output file

        output_file = None
        for root, dirs, files in
os.walk(output_path):
            if
settings.TIMETABLE_OUTPUT_FILE in
files:
                output_file = os.path.join(root,
settings.TIMETABLE_OUTPUT_FILE)
                break
        import xmltodict
        with open(output_file) as fd:
            doc = xmltodict.parse(fd.read())

        doc = loads(dumps(doc))
        classes = doc['timetable']['classes']['class']


Class.objects.assign_final_schedules(classes)
        timetable_obj.status =
Timetable.SUCCESSFUL
        timetable_obj.save()

timetable_obj.assign_class_statuses_count()
    except Exception as e:
        print traceback.format_exc()
        timetable_obj.status = Timetable.FAILED
        timetable_obj.save()

Class.objects.reset_final_schedules(timetable_o
bj.college, timetable_obj.semester)
        print 'An unexpected error was
encountered. Please try again.'
```

98. upmcts/utils/views.py

```
class AjaxTemplateMixin(object):

    def dispatch(self, request, *args, **kwargs):
        if not hasattr(self, 'ajax_template_name'):
            split = self.template_name.split('.html')
            split[-1] = '.ajax'
            split.append('.html')
            self.ajax_template_name = ''.join(split)
        if request.is_ajax():
            self.template_name =
self.ajax_template_name
        return super(
            AjaxTemplateMixin,
            self
).dispatch(request, *args, **kwargs)
```

XI. Acknowledgement

I finally made it! Thank you UP. I have learned a lot of things. I have met a lot of brilliant people. I have made a lot of unforgettable memories. It has been quite the experience, and for it, I will be forever grateful.

To my SP adviser, Ma'am Perl Gasmen, thank you. You accompanied me on my final journey in college. There were a lot of setbacks. There were a lot of struggles. But in the end, with your help, I finally made it. Thank you!

To my high school friends, thank you. High school was a very fun chapter in my life. To Biboy, who was my rival in academics and was also one of my best friends, thank you. We were always the ones who are aiming for the top. From the entrance exam, until our final year in high school, we were there, challenging ourselves and pushing our limits, in order to prove ourselves. But outside of that, we were just kids having fun, trying to make a memorable time. It helped us grow into what we are now. To all my other high school classmates, thank you. We had so much fun together, did a lot of things together, and went to a lot of places together. Most of the time, someone would go on our house and invite me to hang out. Those days were very memorable to me. During college, I was not able to hang out with you anymore. Now that we (most of us) have graduated, I wish to spend more time with you to catch up, and share stories about the different journeys we've had.

To my college friends, thank you. I never really had that much friends outside the block, but for me, it was enough. We never ran out of things to talk about. We came from different places, from different backgrounds, with a lot of stories to tell. I had so much fun sharing ideas and stories with you. I learned a lot from all of you, and I hope I also made an impact, even just a small one, on you. To Marx, Gio, and Lean, thank you for always listening to all my stupid ideas and theories, but yours are just as stupid, haha. We never ran out of crazy things to do (or plans to do), and I hope we can do all of it someday.

To my colleagues at Icannhas, thank you. To Eric, thank you for giving me the opportunity to work with these awesome people. Even though I have not graduated yet when I applied for the job, you still gave me a chance to prove myself. To GM, thank you for letting me be a part of your team. You are the best boss ever. Sorry for all the trouble that I have caused you. Thank you for understanding my situation. You have made a lot of compromises just to let me finish my thesis. I will definitely make it up to you.

Lastly, to my family, all of this was for us. All of the hard work, all the struggles, all of the challenges, I did it for us. I want to make you proud. To my sister, thank you for always being there for me. I was a terrible brother. When we were young, I would always make her cry. But now, after all this time, she is still here for me. To my parents, who were always there from the very start, and will always be there for me, thank you! As a child, I have had some ambitious dreams. They never doubted me. They never pressured me. They just did all they can to support me. Even when I got delayed for a year, all I heard from them were words of encouragement. Thank you for believing in me. I will continue to make you proud.