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

Secure Multiparty Computation for Generating Health Data Statistics

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

Submitted by:

Jairus Mari H. Navarro

June 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 "Secure Multiparty Computation for Generating Health Data Statistics" prepared and submitted by Jairus Mari H. Navarro in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Richard Bryann L. Chua, M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*candidate*)	_____	_____
2. Avegail D. Carpio, M.Sc.	_____	_____
3. Perlita E. Gasmen, M.Sc. (*candidate*)	_____	_____
4. Ma. Sheila A. Magboo, M.Sc.	_____	_____
5. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____
6. Geoffrey A. Solano, M.Sc.	_____	_____

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

Ma. Sheila A. Magboo, M.Sc.
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

Alex C. Gonzaga, Ph.D., Dr.Eng.
Dean
College of Arts and Sciences

Abstract

Sharing of information by various individuals and organizations can provide a way for unlocking important and beneficial knowledge for all the parties involved. However, these information may be inherently confidential. Thus, there is a need for securely pooling these data while keeping their privacy. An important beneficiary of this situation is the medical field. Health data are inherently private, yet they are needed for information gathering, and the advancement of researches across many fields. The health data needed may be collected from databases, surveys, and other input sources. In this study, a health data statistics generation system was developed. It is a web-based tool that uses secure multiparty computation to aggregate answers from surveys published in the system.

Keywords: secure multiparty computation, health data, Sharemind

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Sharing of information by various individuals and organizations can provide a way for unlocking important and beneficial knowledge for all the parties involved. Many different data mining algorithms can be applied to these collected information to achieve the desired results. An application of data mining can be found in health data statistics generation. In this case, hospitals and other medical institutions are to collaborate their data to obtain some statistics that could be significantly helpful in medical research or disease surveillance. However, the information held by these different health units are inherently private. This is mainly due to confidentiality of patient records, and privacy of hospital employees. To address the privacy and security concerns, the data mining techniques to be employed should observe keeping the health information provided by medical units to remain private.

As an illustration, consider the case where the Department of Health (DOH) wanted to strengthen the arguments for passing the Sin Tax Bill into a health law. To do this, they aim to prove that smoking is the leading cause of lung cancer. Also, they aim to show the public that lung cancer could be a possible pandemic in the Philippines in the next 10 years. A possible option would be to conduct a survey with various hospitals and clinics to get the needed data to support their claims. At the bare minimum, they could construct the survey to ask for the number of cases of lung cancer for the past 10 years, along with the causes. However, it is possible that some of these medical units are not allowed to disclose neither patient nor hospital records to an external entity. One issue that can arise is the exposure of patient and doctor information. Another could be intrusive marketing, wherein the collected data will be used to tailor advertisements of products that targets specific medical institutions. This could happen if the survey data will not be kept private, or will be stored without

measures for keeping the confidentiality of input. Lastly, another issue could be the inference of staff performance. Data mining algorithms can be used over the given input of a particular hospital to see the increase or decrease of lung cancer cases, which could potentially imply the effectiveness, or ineffectiveness, of their employees. [1] Therefore, to obtain the needed surveillance reports, participating medical units must first be assured of the confidentiality of their information, and the computations executed must give correct results.

B. Statement of the Problem

Computing for health data statistics over sensitive data collected from different medical institutions can ideally be done by a trusted third party (TTP). To ensure security and privacy, this TTP must be incorruptible. In other words, it must not be vulnerable to malicious attacks from the participating users nor from external users. However, this assumption is unrealistic. It is very hard, if not impossible, to find an external party that can be trusted by all the parties involved in the computation. [2, 3] As an alternative, the computations can be done in a multiparty setting. Secure multiparty computation (SMC) is a technique that deals with multiple parties wanting to jointly compute a function over distributed private information even while under attack against maliciously colluding parties. [3, 4] In the end, the only public knowledge that should be gained from the SMC are the correct results of the computation, and other information that can be inferred from them. SMC should be implemented without the members knowing each others inputs. [2, 3, 4] With this, security is a guarantee for the participants. Loosely speaking, a multiparty computation is secure if it exhibits the following properties: privacy, correctness, independence of inputs, guaranteed output delivery, and fairness. [3]

C. Objectives of the Study

To create a web-based Health Data Statistics Generation System (**HDSGS**) by conducting online surveys and computing the results through secure multiparty computation, with the following roles and their corresponding functionalities:

1. Allows the public user to
    (a) View health data surveys.
    (b) View the creator of a health data survey.
    (c) View results of health data surveys.
    (d) Download results of health data surveys.
2. Allows the contributor to
    (a) View health data surveys.
    (b) View the creator of a health data survey.
    (c) Answer health data surveys.
    (d) View health data surveys answered.
    (e) View results of health data surveys.
    (f) Download results of health data surveys.
    (g) Edit own account details.
    (h) Deactivate own account.
3. Allows the research group leader to
    (a) View health data surveys.
    (b) View the creator of a health data survey.
    (c) View results of health data surveys.

(d) Download results of health data surveys.

(e) Add new health data surveys.

(f) Manage created surveys.

i. Edit created health data surveys.

ii. Deactivate created health data surveys.

iii. Delete created health data surveys.

(g) Edit own account details.

(h) Deactivate own account.

4. Allows the admin to

5. Add new user accounts.

6. View details of any user account.

7. Manage user accounts.

(a) Edit details of any user account.

(b) Deactivate any user account except own.

(c) Reactivate any user account.

(d) Delete any user account except own.

D. Significance of the Project

This system will especially be beneficial to health researchers, institutions or individuals, as it provides a secure way of pooling data. This system aims to be an avenue for conducting privacy-preserving data collection regarding health data without sacrificing the confidentiality of the inputs. It can be used by medical laboratories to obtain

information regarding the effectiveness of prescribed drugs in treating a specific illness. It can also be used by health regulatory boards through conducting surveys for surveillance reports regarding various diseases. Moreover,

since the reports of the surveys are publicly available, this system can also be of help to information seekers with regard to the available health data statistics posted.

E. Scope and Limitations

1. The three miners in the system are virtually hosted in a single machine.
2. The system, which uses the Sharemind framework for implementing SMC, only accepts integers as valid inputs.
3. Auditing and interpretation of results is not part of the system.
4. Only the following statistics can be computed for each survey:
   (a) Total number of contributors who answered the survey.
   (b) Average answer per question.
   (c) Maximum answer per question.
   (d) Minimum answer per question.
   (e) Tally of answers per question.
5. Statistical tests cannot be performed by the system.
6. Mechanics for choosing a contributor is outside of the system.
7. Mechanics for choosing a research group leader is outside of the system.
8. Mechanics for approval of accounts is outside of the system.
9. Separate accounts are needed if a contributor is also a research group leader.

10. A research group leader can have several concurrently open surveys.

11. A contributor cannot edit his or her answers to a survey.

F. Assumptions

1. A contributor is a single official representative of a medical institution.

2. All three miner hosts are given the same set of scripts to execute.

3. No official patient record will be provided to answer the survey.

4. Surveys are automatically approved once constructed.

5. Surveys to be created are those whose results are to be publicly available.

II. Review of Related Literature

Secure multiparty computation has been a topic of research since 1980s. [5] In SMC, all participants compute for the output of a function together without revealing their inputs to the other participants. A first approach to SMC is to have a *purely multiparty* scenario wherein the participants themselves are the ones who are going to perform the joint computation. In [6], Yao introduced a technique for securely computing a two-party function implicitly using Boolean circuits. It was shown that given a function that can be computed in polynomial time, there is a corresponding logical circuit of polynomial size that can evaluate it. In this *circuit-based* approach, two parties are involved, namely, the circuit generator and the circuit evaluator. The circuit generator will build a garbled logical circuit for the given function. This garbled circuit will then be securely assessed by the circuit evaluator in a constant number of rounds. Therefore, any function computable in polynomial time can be securely computed in a constant number of rounds. Nonetheless, having all participants involved in SMC presents some problems. One is the inability of have all participants online throughout the entire duration of the computation. Another is the high communication and/or complexity cost of executing protocols with large number of participants involved. [2] As an alternative, the *representative-based* architecture [2] can be used. In this approach, the participants are divided into three, not necessarily mutually disjoint, groups: input parties, computing servers, and output servers. Input parties are the donors of the data. Each input party splits the input and distributes them among the computing servers. The computing servers are the ones who will participate in the SMC in computing the output of a function. This output is then passed to the output servers, which will then announce the output of the computation. With this approach, participants need not be online during the whole computation period. Hence, the system can be expanded to a larger scale, with a bigger number of input parties.

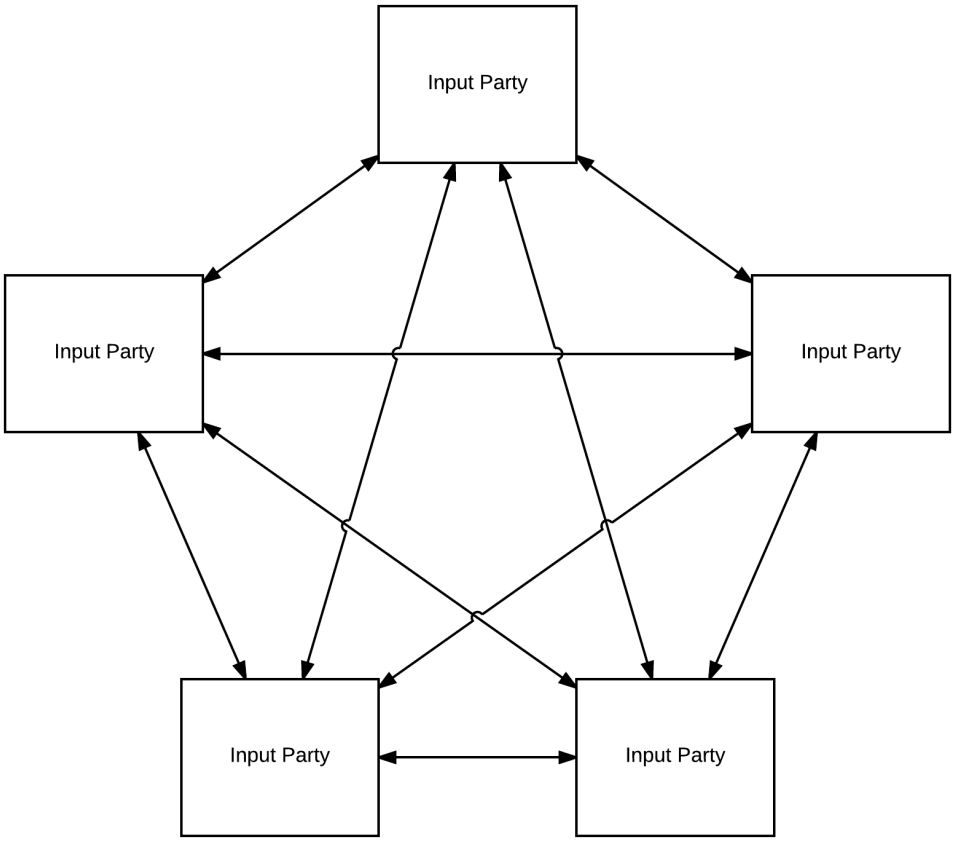


Figure 1: Purely Multiparty architecture

Different frameworks for implementing secure multiparty computation are publicly available. Each of which observes any one of the three approaches previously discussed. For the purely multiparty approach, there is VIFF [7] and Canon-MPC [8]. For the circuit-based approach, Fairplay [9] and FairplayMP [10] are available. For the representative-based approach, SEPIA [5] and Sharemind [11] can be used. These frameworks can be used to solve general privacy-preserving problems. [12]

VIFF (Virtual Ideal Functionality Framework) is a Python-based framework that provides a library for developing cryptographic protocols that will be used in an SMC environment. It aims to be usable in both asynchronous networks such as the Internet, and synchronous networks such as local area networks. Its library provides security until the number of corrupted parties is a third of the total participants.

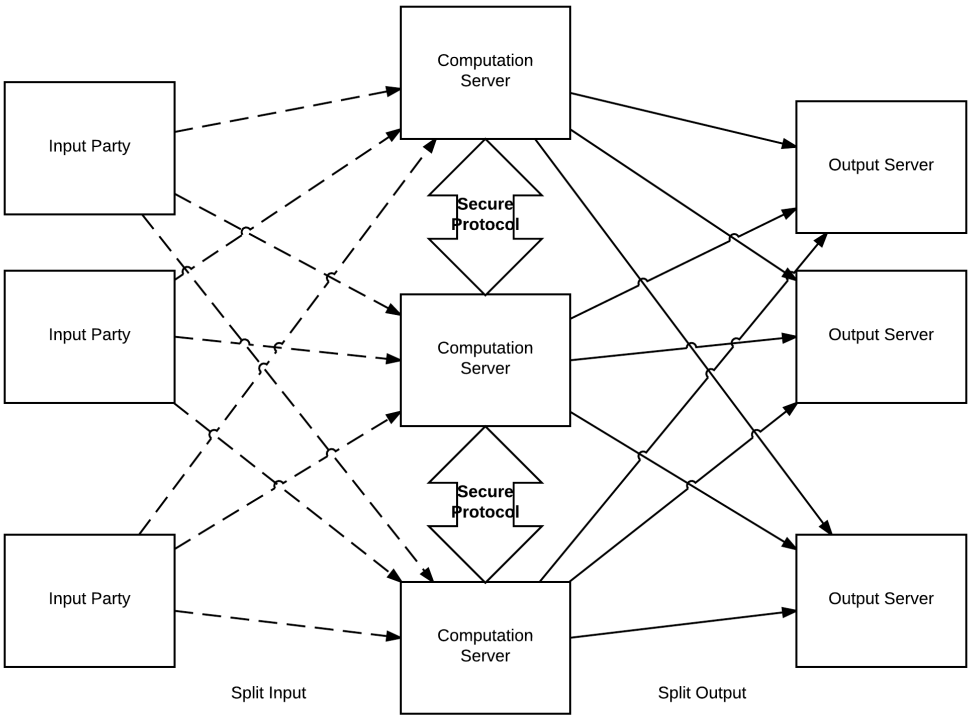


Figure 2: Representative-based architecture

[7, 12] VIFF utilizes the idea of automatic parallelization wherein an operation is readily executed once the needed inputs are acquired. Due to this, implementations of functions in VIFF can be reasonably fast, as was shown in the tests performed in [7] and [12]. Some applications that used VIFF include the sugar beet contracts double auction in Denmark that was based on the research in [13], implementation of a protocol for distributed RSA key generation [14], realization of a multiparty version of the Advanced Encryption Standard [15], and web-based voting applications in [16] and [17]. Furthermore, VIFF requires several dependencies before it can be usable. These many dependencies need to be setup manually which can be cumbersome for the developer.

Canon-MPC (CAsual NON-interactive secure Multi-Party Computation) is a browser-based system that enables non-interactive secure computation. This framework does not require any software to be installed on the participants' computers, except for a

web browser. It uses Google's Native Client (NaCl)[1] technology. It is an open-source technology that will be able to download and run native code in the web browser. The current implementation of Canon-MPC is available online[2]. Canon-MPC implements the protocol presented in the study of Halevi et. al. [18] for computing symmetric binary functions. In this framework, the participants themselves are the ones who will perform the secure multiparty computations. However, it observes the idea of non-interactive computation, meaning participants do not need to be online all at the same time while performing the computation. Nonetheless, all participants are expected to give their own inputs in order to go through the protocol. Realistically, there is a possibility that not all of them are going to participate. Consequently, the computation will be halted until all parties have participated. To address this problem, Jarrous [8] suggested a variant implementation of the protocol for handling absentee participants. Canon-MPC's protocol was also designed to be secure against malicious adversaries. However, the currently available implementation of the system was not coded securely. It is vulnerable to security attacks such as SQL injection, and it does not use TLS encryption since HTTPS was also not used, although the transmitted messages between client and server are encrypted using the El Gamal scheme.

Fairplay is an SMC framework that was specifically made for secure two-party computations. It is the predecessor of FairplayMP. FairplayMP was designed for secure computations distributed among three or more parties. Both systems use the logic circuit approach in solving multiparty computation problems which was based on Yao's garbled circuit system [6]. The scripts used are written in a language called SFDL (Secure Function Definition Language) which is also used to define the number of participants beforehand. [12] The functions defined are then converted to Boolean circuits. Because of the nature of Boolean circuits, both systems suffer from high

[1]https://developer.chrome.com/native-client
[2]http://canon-mpc.org/

runtime complexity. [10, 12]

SEPIA (Security through Private Information Aggregation) is a Java library that observes multiparty computation for aggregating data over multi-domain networks. [5] This library implements basic operations such as addition, multiplication, and comparison of secret shared values. [5] It utilizes Shamir's secret sharing scheme [19] for dividing inputs of parties to be distributed to privacy peers. These privacy peers may be a subset of the input peers, or external parties, who will be the ones to carry out the computations. The privacy of input data is guaranteed as long as less than half of the privacy peers are semi-honest. [5] Semi-honest parties are those who try to learn more information about the other members' inputs but still follow the protocol. SEPIA also offers a graphical user interface for setting up the initial configurations for input and privacy peers. However, these initial configurations may be cumbersome especially if there are many input and privacy peers involved. Moreover, in SEPIA, the developer must program the entire flow of the secure multiparty computation, from the input collection to the generating of shares, the computation proper, and ultimately the publishing of the output. The SEPIA protocol was used in an application in [20] which troubleshoots network outages by collaborating traffic monitoring data across multiple Internet service providers.

Sharemind is an SMC framework implemented in C++. Sharemind protocols can be programmed using Sharemind assembly [21], C++, or SecreC [12]. Sharemind has been used in various deployed applications such as a financial data analysis project in Estonia [22] which is the realization of Talviste's thesis [4]. Sharemind was also chosen in showing the feasibility of using SMC in a genome-wide association study [23] with databases, and surveys as data sources. The Sharemind framework has also been used in several prototype implementations such as for an online survey [4, 24, 25]. The aforementioned real-world applications, and feasible prototypes show that Sharemind is a sound framework to utilize.

III. Theoretical Framework

A. Privacy-Preserving Data Collection

Data mining is a technique employed to often large data that aims to find hidden patterns, discover relationships between variables, and summarize the data in such a way that it is understandable and useful to the owner/s. [26, 27] Due to the ever increasing amount of data that are gathered and stored [27, 28], data mining has become increasingly popular. [26, 27] The inputs in data mining algorithms may be from a single or multiple sources, owned by a single or multiple parties. Illustratively, data mining can be helpful in the corporate world when used in generating financial reports. A multinational company can collect financial statements from its different branches to determine which of them are doing well or not. This knowledge can help that company's executives decide on expanding or downsizing. Another is in the retail business; a store owner can use data mining on their transaction records to determine consumer trends. By discovering these trends, retailers will know which products to stock more (or less) of. The medical industry can also benefit greatly from data mining algorithms. Genetic databases can be analyzed to aid in medical researches such as drug manufacture. Patient records from different hospitals and clinics can also be mined to, for example, compute statistics of health cases in a city, among others.

A significant problem in data mining is preserving the privacy of data to be analyzed. [4, 26, 27] Keeping the privacy of data can be motivated by law—as for the case of hospital records—or by business interests—as for financial statements and transaction records. Consequently, the data to be processed should also be safe from exposure and corruption. To address these concerns concerns, privacy-preserving data mining (PPDM) is applied.

Privacy-preserving data mining concerns itself with executing data mining algo-

rithms on confidential data. Generally speaking, the privacy aspect in data mining should guarantee that accidental or purposeful disclosure cannot be used to determine the source of the exposed information. [26] To keep the confidentiality of the data, they should not only be kept from the public, but should also be kept secret from any of the contributors, and those running the algorithm. [3, 26]

B. Secure Multiparty Computation

In an ideal world, there exists an external and honest TTP, trusted by all the contributors involved, that will be the one to perform the mining of data. [3] In such a world, each of the contributors only needs to send their inputs to the TTP, and wait for the aggregated output. It is noted that the inputs are safe from exposure since the TTP is following the protocol, and that the contributors are not maliciously colluding with any other party. Also, the output is assumed to be correct because the TTP is honest. In the real world, however, these conditions are not easily satisfied. Firstly, finding an external TTP that is incorruptible, honest, and willing can be very difficult. Secondly, malicious attacks can be performed by the contributors themselves. It can happen that a contributor will collude with a subset of the parties involved–including the TTP–to uncover the inputs submitted by the honest contributors, manipulate the input or output, and even abnormally terminate the protocol. With that, finding a TTP all the more proves to be troublesome.

Secure multiparty computation aims to solve this issue. It was first introduced by Yao [6] wherein he posed the millionaire's problem. It is about two millionaires who want to know which of them is richer without letting the other know of their wealth. SMC was defined to be a technique that aims for multiple to parties to be able to jointly compute a function, given their respective inputs. These parties should be able to learn the correct output even if they are under a malicious attack, and nothing more. [3, 4, 6, 17] SMC minimizes the trust required from the contributors

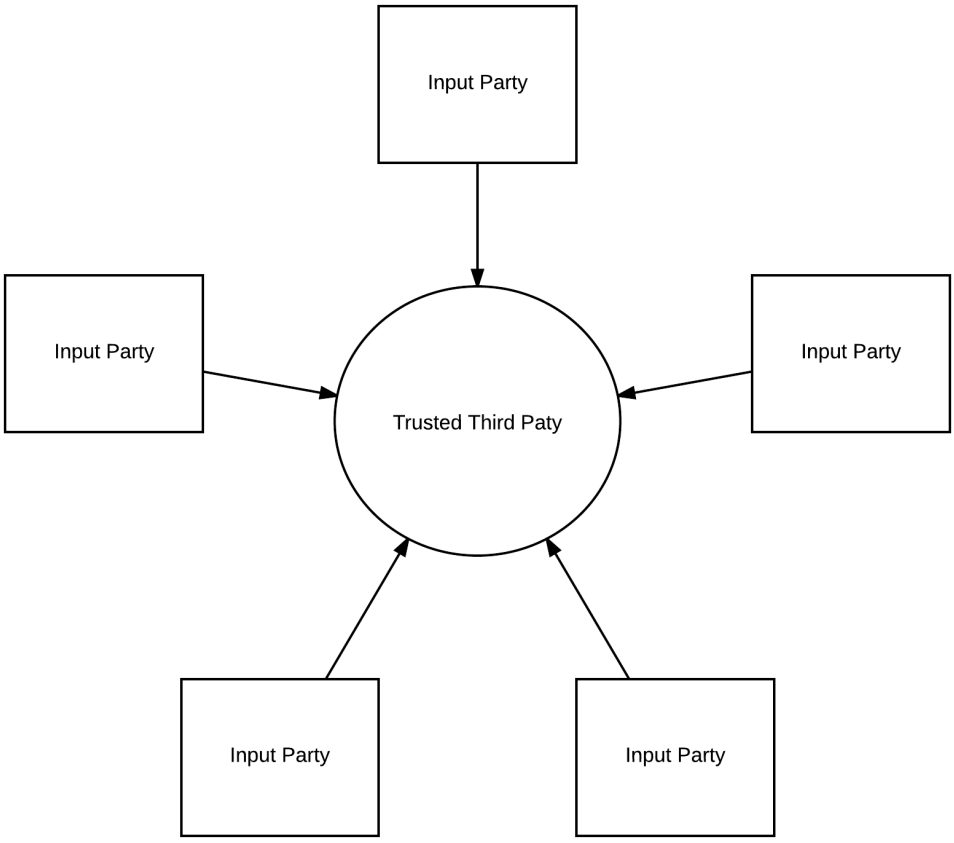


Figure 3: TTP-based architecture

since a TTP is no longer needed. [17]

C. Sharemind

Sharemind is an SMC framework developed by Bogdanov, et. al. [11]. It uses the additive secret sharing scheme over the ring of 32-bit integers. The functions executed for the secure multiparty computation are done by three dedicated nodes called *miners*. Having three miners is optimal as the overall communication complexity grows quadratically with the number of miners; and because guaranteeing an adequate level of security for many data storage facilities may be expensive. [11, 29] By using the representative-based approach, participants using the application need not be online

while the computation is executing. It can provide security even if one of the three miner nodes are corrupted during the computation. [11] Though secret sharing of the inputs may be automatic especially for standalone applications, the developer may also program the secret sharing protocol in a different language where the shares are directly sent to the miners. Programming in Sharemind can be done by using Sharemind assembly which was developed by Jagomägis [21], C++, or *SecreC* which was also developed by Jagomägis [12]. The Sharemind assembly language is a low-level language that can be executed by the virtual machine. It gives the programmer more control over the computations than the other two, since the details of the computation are to be specified, down to the registers to be used. On the other hand, using C++ also gives the developer control over the computations because the implementation details need to be coded explicitly. Writing functions in Sharemind assembly and C++ can be tedious considering that the programmer needs to know the architecture of the computer, and clearly define the security implementation of the functions. In contrast to this, the programmer does not need to know about underlying structures when coding in SecreC. SecreC is a high-level privacy-aware language that aims to abstract the cryptographic side of programming so developers can focus more on the business logic of the application. [12] It was developed specifically for the use in the Sharemind framework. SecreC codes are compiled into Sharemind assembly scripts. These scripts are then given to each of the miners to be executed by Sharemind. The advantage of using SecreC is that the programming style is similar to that of normal procedural code. There is no need to explicitly code the per-peer detailed instructions needed in SMC as this is already abstracted by the language. The three miners of the application is simulated virtually on a single machine during development.

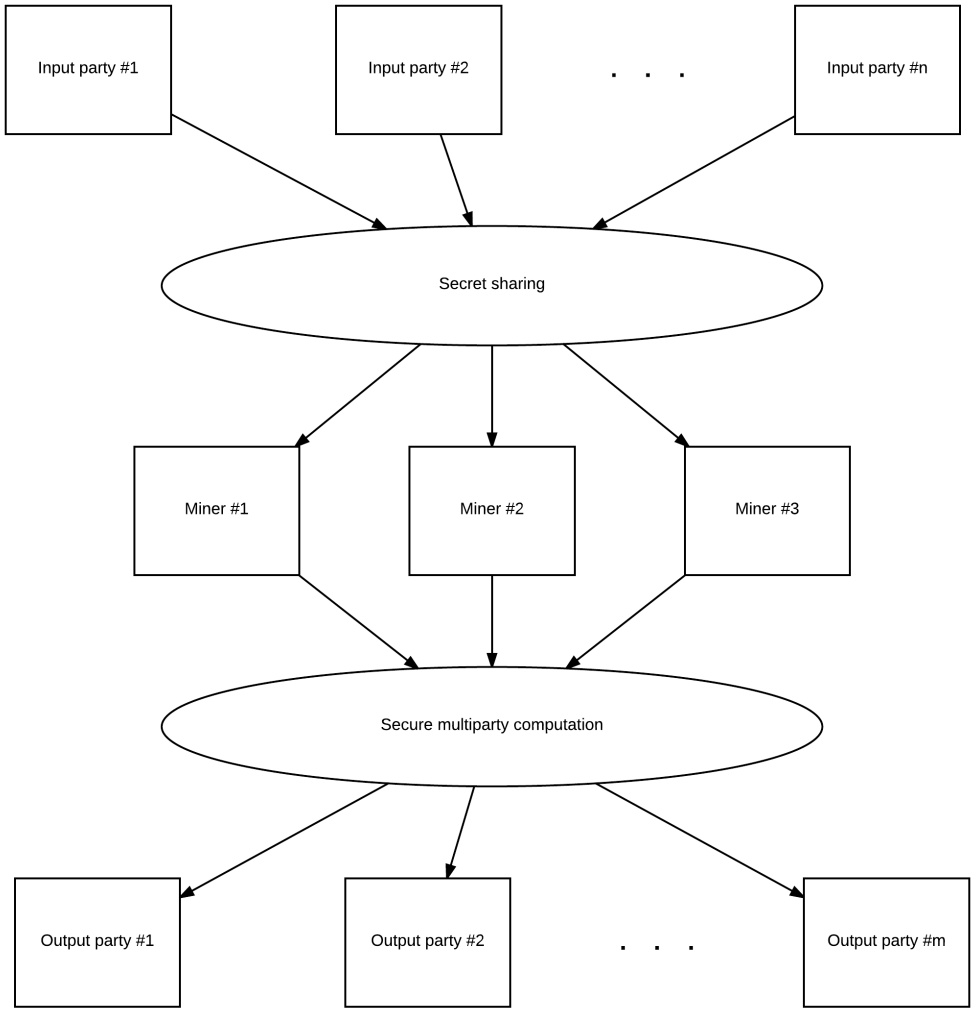


Figure 4: Overview of Sharemind's process

IV. Design and Implementation

A. Use Cases

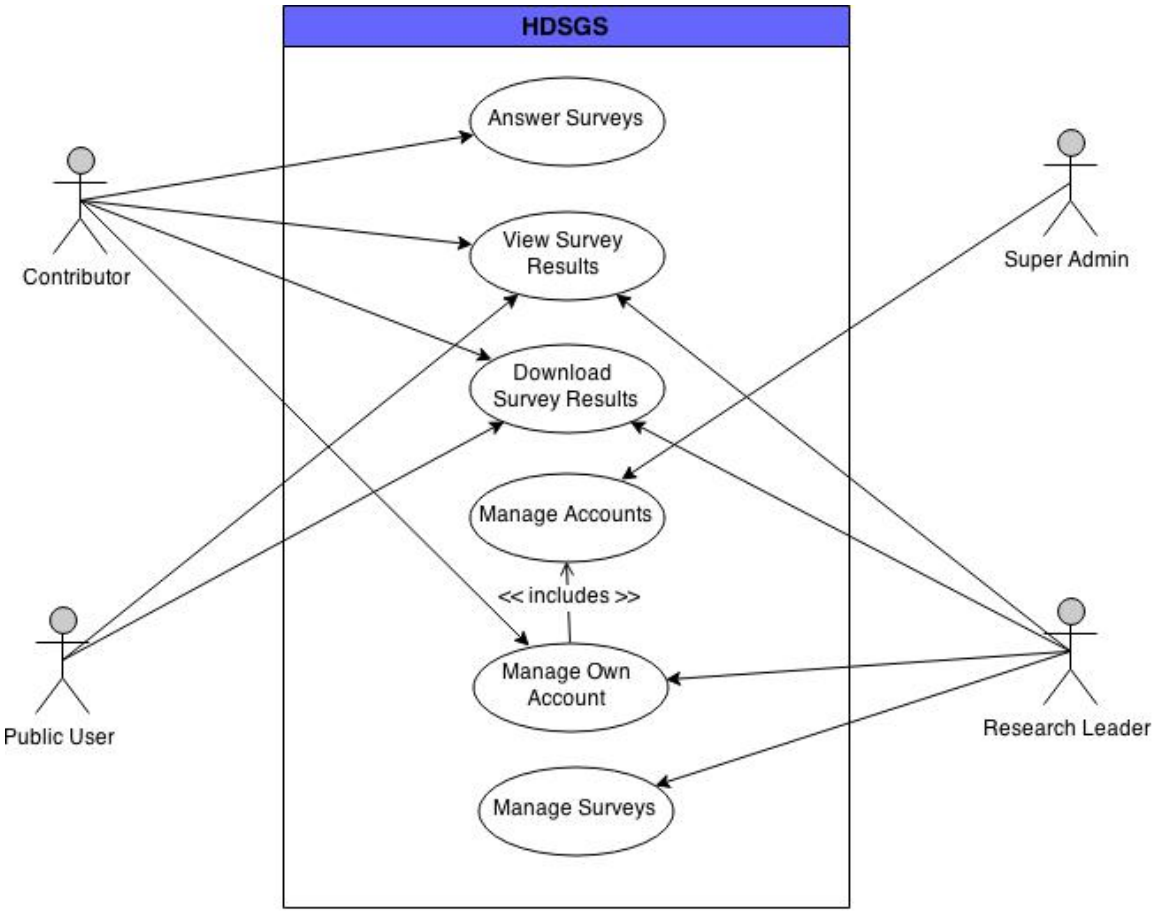


Figure 5: Use case diagram

The health data statistics generation system has four user roles: public user, contributor, research leader, and admin.

A public user, contributor, and research leader can view a survey along with its results. Moreover, a PDF copy of these results can be downloaded by any of the three mentioned user roles.

A research leader is the only one capable of creating surveys, while only a contributor can answer them. Furthermore, a research leader can also update, deactivate,

and delete any survey that he or she has created.

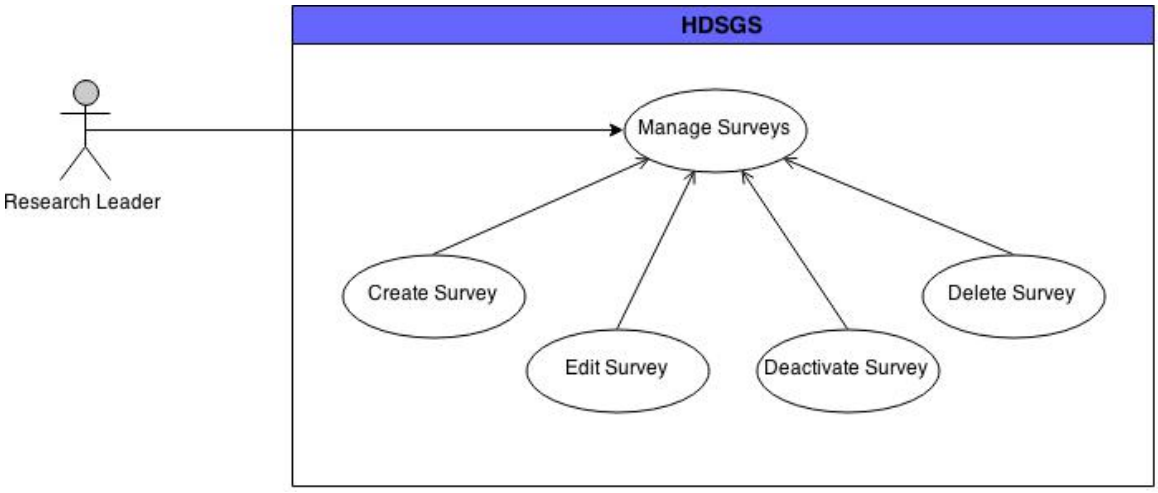


Figure 6: Use case diagram for managing surveys

Management of user accounts in the system is the sole responsibility of the admin. The admin can view all the users in the system. In contrast, a contributor or research leader can only view their own account, while a public user does not have one. Updating, and deactivating of a user account can be done by the admin, and also by the account owner. Additionally, deleting of a user account can only be performed by the admin.

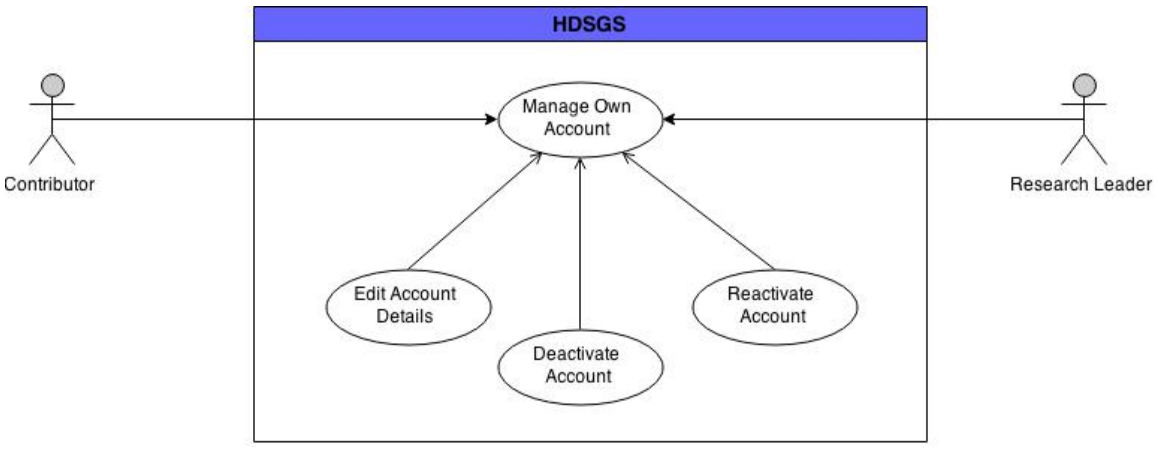


Figure 7: Use case diagram for managing own account

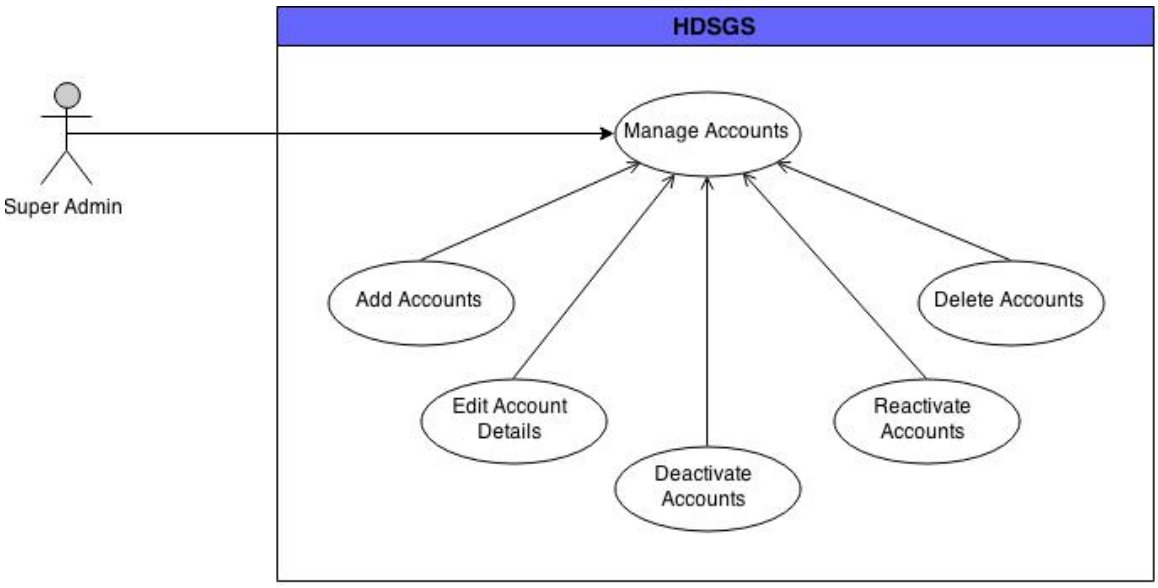


Figure 8: Use case diagram for managing user accounts

B. System Development

The **Spring** framework was used in developing this web application. HDSGS follows the three-tier application architecture. These three layers are namely, the database tier, the service tier, and the presentation tier.

1. Database Tier

The database tier is responsible for the creation and manipulation of the various data needed by the system. The information about the users and surveys are stored in the *HDSGS database* which is backed by MySQL. The passwords of the users are encrypted by Spring Security before storing them in the HDSGS database. In addition, only the survey details, questions, and computed results are stored there since they do not need to be kept private. Spring Data and Hibernate[3] are used for connecting the MySQL database to the HDSGS web application and vice-versa. For the survey answers, however, privacy of data is important so they are stored in the

[3]http://hibernate.org/orm

internal database of Sharemind. In Sharemind 2, only integer data types can be stored privately. Hence, survey answers have to be converted into an integer value when they are stored in the Sharemind database. Since we do not want a particular survey answer to be traced to a particular contributor, there is no column for storing the identifier of contributors. This implies that we cannot trace the particular respondent of a survey answer by inspecting the database records. On the other hand, this also means that contributors cannot edit their survey answers.

2. Service Tier

The service tier is where all the business logic is implemented. It contains the implementation details of all the functionalities provided by the application. This layer is the one who communicates with the database layer, and also with the presentation layer. In essence, the service layer protects the database information from the direct manipulation of the client. It is also in the service layer that the results of a survey are computed through secure multiparty computation. External programs, called *Sharemind controllers*, are called by HDSGS when computing for survey results. They run the appropriate SecreC scripts to achieve the desired output. The SecreC scripts are the ones that communicate with Sharemind's internal database. They connect to the miners who will then perform the secure multiparty computation.

To compute for survey results, four SecreC scripts were written. There is a script for computing the average answer, the minimum answer, the maximum answer, and for counting the number of times a certain question choice was answered. This last script is used for obtaining the tally of answers for a survey question. Since Sharemind can only store integers and booleans as secret-shared data[12], each question choice is given an equivalent integer value. This value is the one inserted into the Sharemind database, and is consequently used for the tally computations. In assigning the value of the choice, the first choice is 1, the second is 2, and so on.

3. Presentation Tier

The presentation tier is the one that interacts with the client through the various endpoints of the application via a Javacript-enabled browser. This is made possible by Spring MVC, Spring HATEAOS, Spring Security, and Thymeleaf[4]–an HTML template engine. HDSGS controllers, developed with Spring, handle the various requests made by the client. They call the correct service methods to fulfill the request then assembles the appropriate response. The web pages are HTML5 documents enriched with CSS and Bootstrap[5], and jQuery[6].

C. Database Design

Two databases are utilized for this application. The main database for this web application, the HDSGS database, is a MySQL database. It contains information about the users and the surveys. The internal database of Sharemind was used here for storing the survey answers.

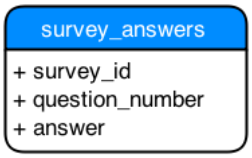


Figure 9: Database table in Sharemind

[4]http://thymeleaf.org
[5]http://getbootstrap.com/
[6]https://jquery.com/

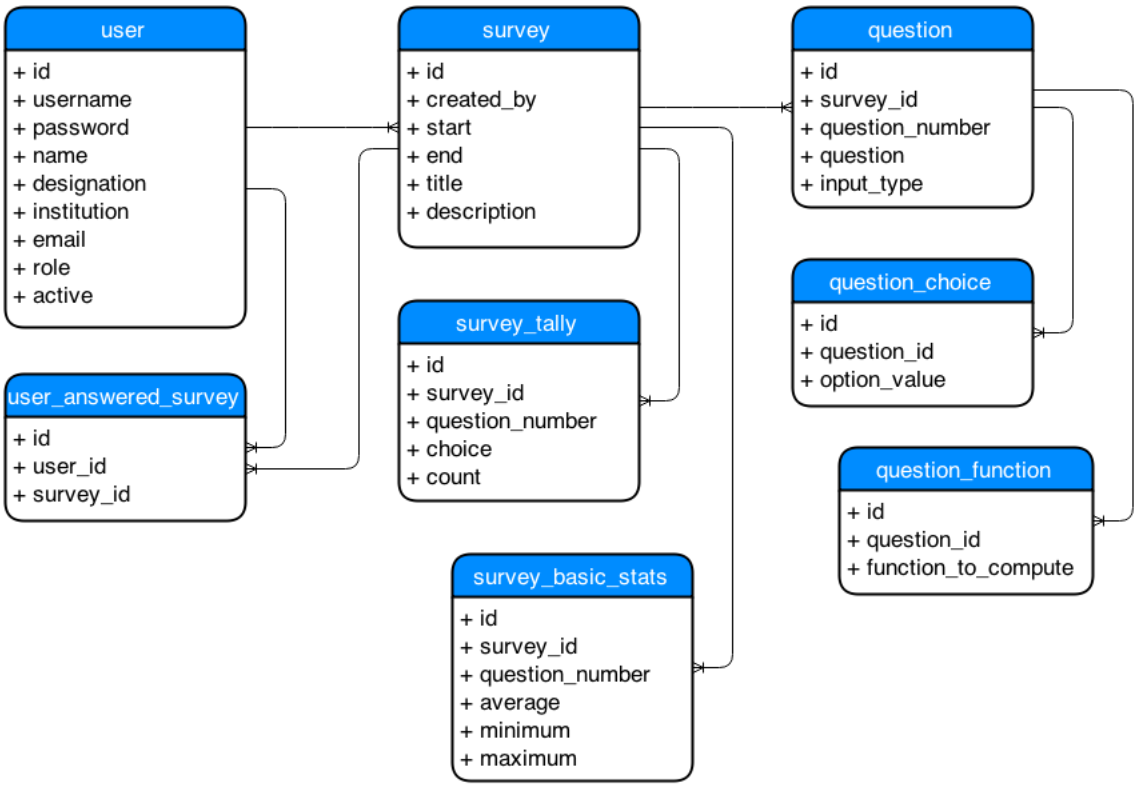


Figure 10: Database tables in MySQL

user			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of user
username	varchar(20), unique		Username of user
password	varchar(255)		Password of user
name	varchar(50)		Name of user
designation	varchar(50)		Position held by user
institution	varchar(50)		Institution where user belongs
email	varchar(30)		Email address of user
role	enum('CONTRIBUTOR', 'RESEARCH_LEADER', 'ADMIN')		Role of user in the system
active	bool		Status of account

Table 1: Data dictionary for **user** table

survey			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of survey
created_by	int(11)	Foreign	ID of creator; references *user.id*
start	date		Date when survey ends
end	date		Date when survey ends
title	varchar(50)		Survey title
description	varchar(255)		Survey description or message for the contributors who will answer

Table 2: Data dictionary for **survey** table

question			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of question
survey_id	int(11)	Foreign	ID of survey where question belongs; references *survey.id*
question_number	int(3)		Question number
question	varchar(255)		Question
input_type	enum('NUMBER', 'RADIO', 'CHECKBOX', 'DROPDOWN')		Form of choices or input to question

Table 3: Data dictionary for **question** table

question_choice			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of question choice
question_id	int(11)	Foreign	ID of question where choice belongs; references *question.id*
option_value	varchar(50)		Choice

Table 4: Data dictionary for **question_choice** table

question_function			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of question choice
question_id	int(11)	Foreign	ID of question where function belongs; references *question.id*
function_to_compute	enum('AVERAGE', 'MINIMUM', 'MAXIMUM', 'TALLY')		Result to compute for a question's answers

Table 5: Data dictionary for **question_function** table

survey_basic_stats			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of record
survey_id	int(11)	Foreign	ID of survey; references *survey.id*
question_number	int(3)		Question number
average	double		Average of answers to question, -1 if not computed
minimum	int(11)		Minimum answer to question, -1 if not computed
maximum	int(11)		Maximum answer to question, -1 if computed

Table 6: Data dictionary for **survey_basic_stats** table

survey_tally			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of record
survey_id	int(11)	Foreign	ID of survey; references *survey.id*
question_number	int(3)		Question number
choice	varchar(50)		A choice for the question
count	int(11)		Number of times the choice was answered to the question

Table 7: Data dictionary for **survey_tally** table

user_answered_survey			
Column	**Type**	**Key Type**	**Caption**
id	int(11), auto-incremented	Primary	ID of record
user_id	int(11)	Foreign	ID of contributor who answered the survey; references *user.id*
survey_id	int(11)	Foreign	ID of survey that the contributor answered; references *survey.id*

Table 8: Data dictionary for **user_answered_survey** table

D. System Architecture

The main framework used in developing this web application is Spring, along with Sharemind which provided the secure multiparty computation of survey results. Spring provides a comprehensive programming model for Java-based applications on any deployment platform, standalone, web, or otherwise. Under the Spring umbrella are many different projects to aid the development of these applications. For HDSGS, the Spring modules used are Spring Boot for the main structure, Spring Data for database interactions, Spring MVC for making the application web-based, Spring HATEAOS for making the application hypertext-driven, and Spring Security for authorization and authentication support.[7]

For the data storage, the MySQL and Sharemind databases are used. The database in MySQL is connected to HDSGS via Hibernate and Spring Data. Hibernate is concerned with data persistence in relational databases, while Spring Data is concerned with the data access methods of the application, and therefore also invokes the functionalities of Hibernate. The Sharemind database is used as a repository for survey answers. This is because the Sharemind framework handles the privacy-preserving storage of these data. This repository is connected to HDSGS via the Sharemind

[7] http://spring.io/projects

controllers. They are called upon by the service methods of the application which are developed with Spring. These service methods are invoked by the HDSGS controllers, also developed with Spring. These controllers are available to the client via HTTP requests made through a browser. The HTML pages of this application are created with Thymeleaf. Thymeleaf is a template engine whose goal is to create HTML page templates.

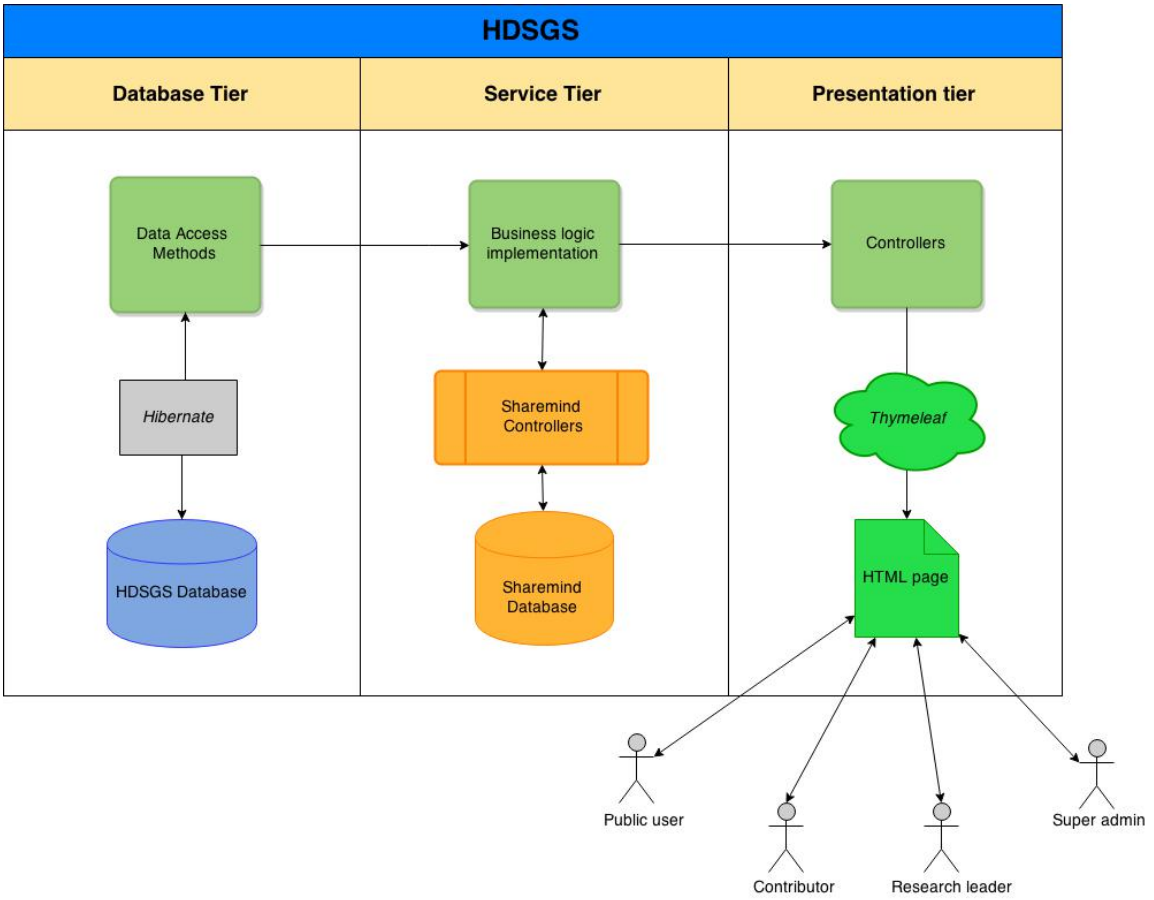


Figure 11: System architecture

E. Technical Architecture

The database used for the web information is MySQL, while the one used for storage of survey answers and aggregation of survey results is the internal database of

Sharemind. The Sharemind SDK used is contained in a virtual machine that is run on the host computer using VirtualBox.[8] The virtual machine was configured to have additional software such as as Tomcat, MySQL, and a JavaScript-enabled browser.

System Technical Components

1. Host OS
   (a) Windows 8.1, 64-bit
   (b) VirtualBox 4.3.20
2. Virtual Machine
   (a) Linux Debian 6.06, 64-bit
   (b) 1.92GB RAM
   (c) 1.80GHz processor
   (d) Sharemind 2 SDK
   (e) Tomcat 7
   (f) MySQL 5
   (g) Chromium 6.0.472.63 or Iceweasel 3.5.16 with enabled JavaScript

[8] https://www.virtualbox.org

V. Results

A. Home Page

Figure 12 shows the index page of HDSGS. This is the home page shown if the user is not logged in. There is a link in the navigation bar for going to the home page, and viewing the list of existing surveys in the system. There is also a link for viewing the login page, which can be seen in Figure 13.

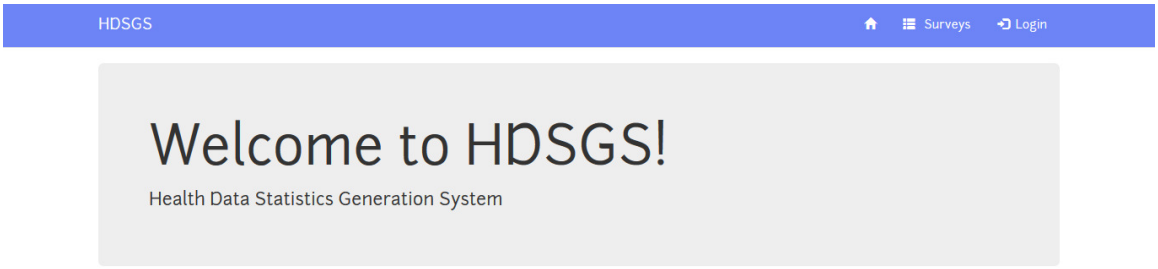


Figure 12: Home page for public users

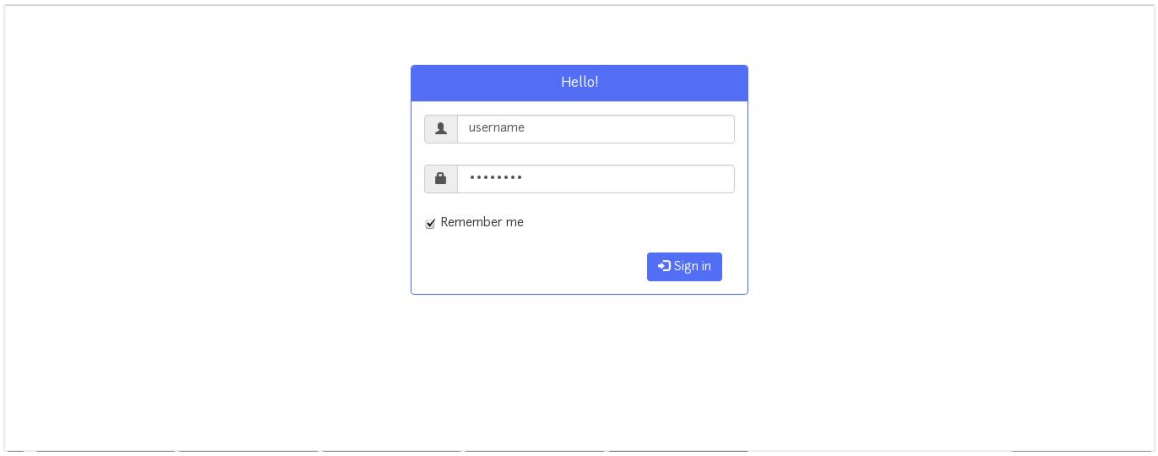


Figure 13: Login page

After successfully logging in to the system, the user will be redirected to the home page but with different navigation bar links depending on his or her role.

A contributor or a research leader will have links for viewing the home page, the list of surveys, their own profile, updating their own profile, and for logging out of the system as shown in Figures 14, and 15. On the other hand, Figure 16 shows that the admin will have a link for viewing the list of users instead of a link for viewing the list of surveys.

If a user has successfully logged out of the system, he or she will then be redirected to the index page as seen in Figure 12.

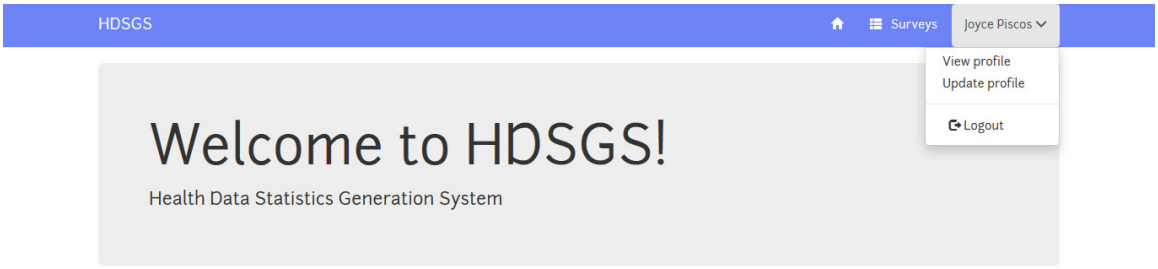


Figure 14: Home page of a contributor

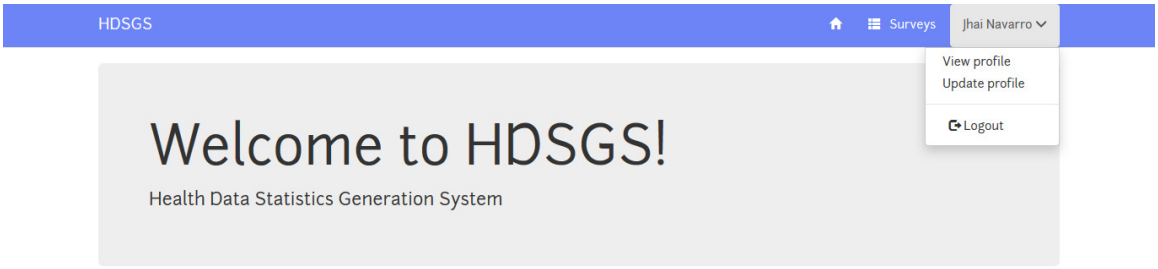


Figure 15: Home page of a research leader

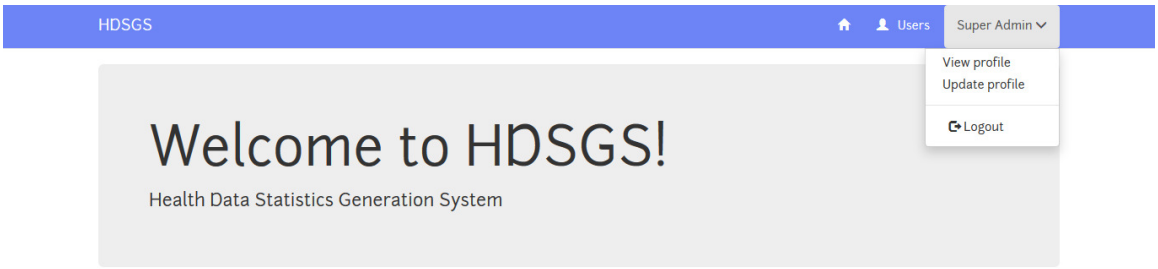


Figure 16: Home page of the admin

B. View All Surveys

Figures 17 and 18 show that a public user, a contributor, or a research leader can view the list of all surveys, however, only a research leader has a button for creating a new survey.

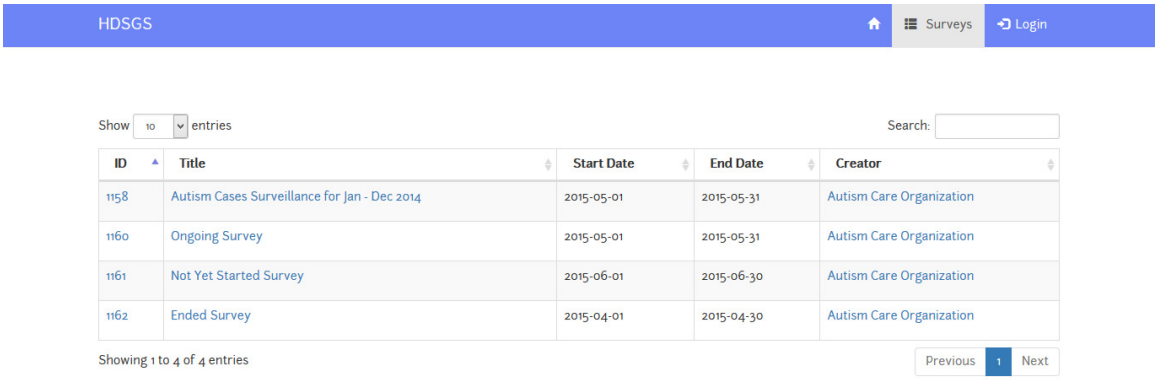


Figure 17: View all surveys page

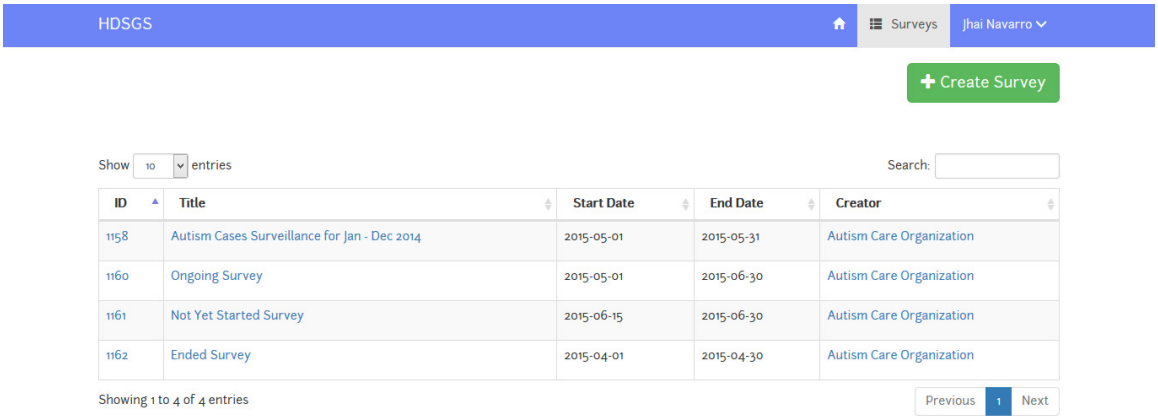


Figure 18: View all surveys page, accessed by a research leader

C. Create Survey

After a research leader has clicked the Create Survey button, he or she will be pointed to the page shown in Figure 19. If the form has errors, the appropriate messages will be shown upon clicking the Submit button. Else, the research leader will be redirected to the page for viewing the details of the newly created survey which can be seen in Figure 20. Note that this page will differ based on the status of the survey. Meanwhile, clicking the Cancel button will simply redirect back to viewing the list of all existing surveys.

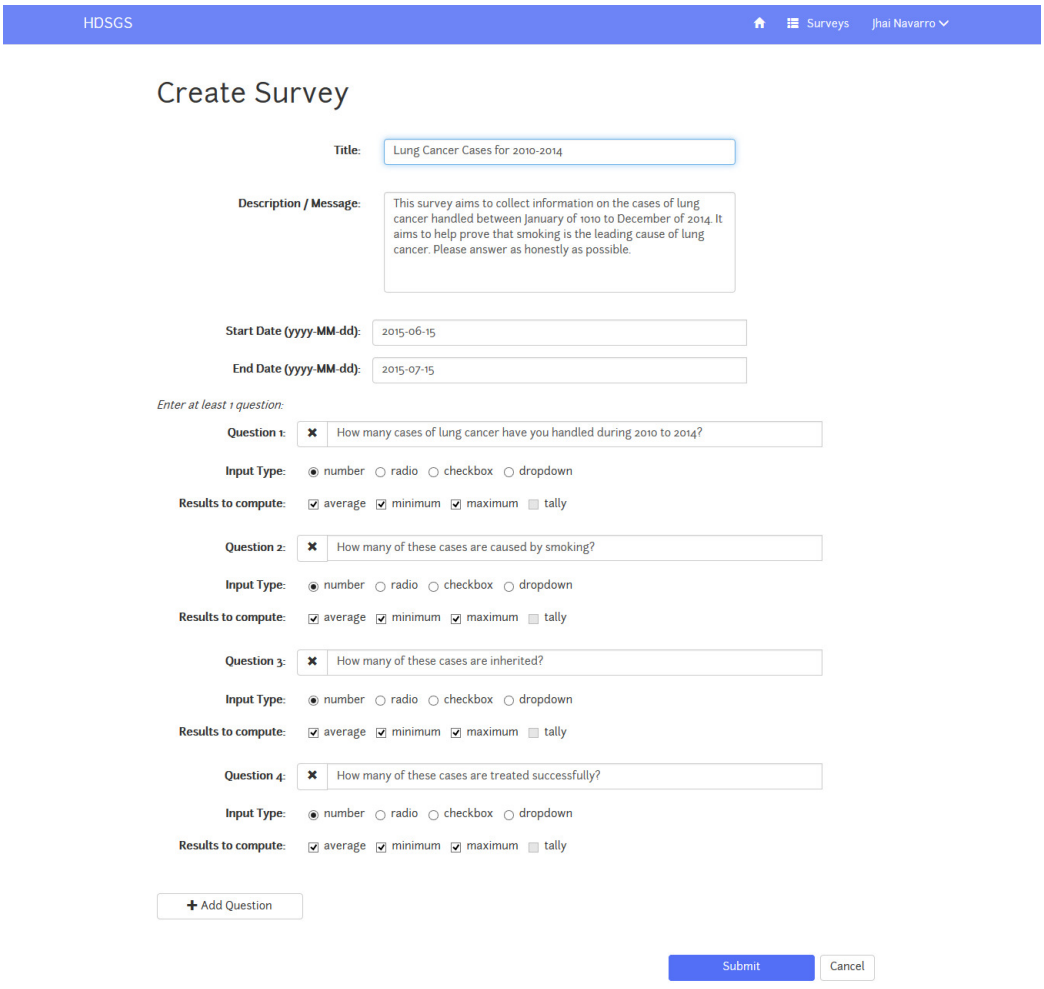


Figure 19: Create survey page

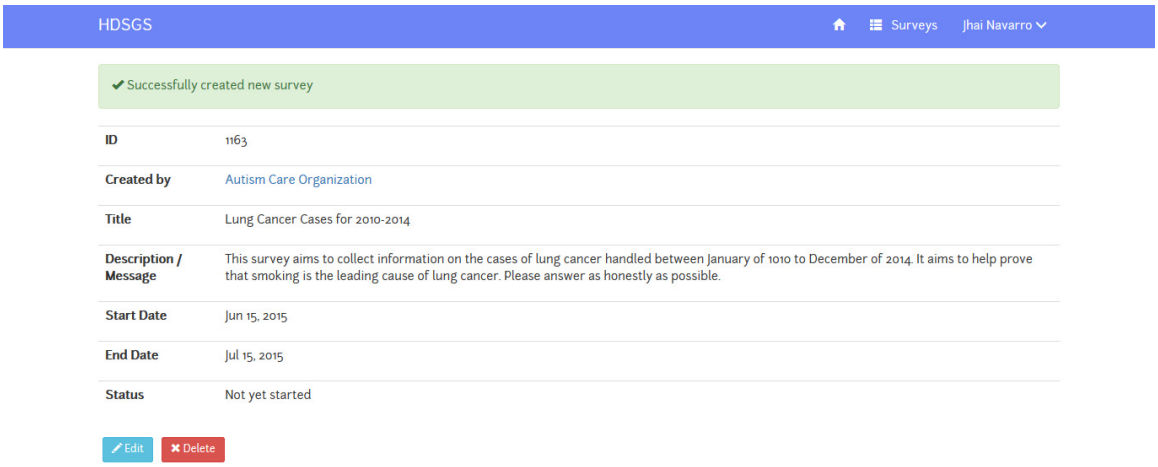


Figure 20: Successfully created survey page

D. View Survey Details

The details of a survey can be viewed by clicking its ID or title in the View Surveys page. Only public users, contributors or research leaders can access this page. The buttons available on a survey's page will depend on the role of the user accessing it, and the status of the survey.

For a survey that has not yet started, public users and contributors will see the page shown in Figure 21. Meanwhile, research leaders can view the page in Figure 22 which includes the Edit and Delete buttons.

Figure 21: View page of survey that has not yet started, accessed by a public user or contributor

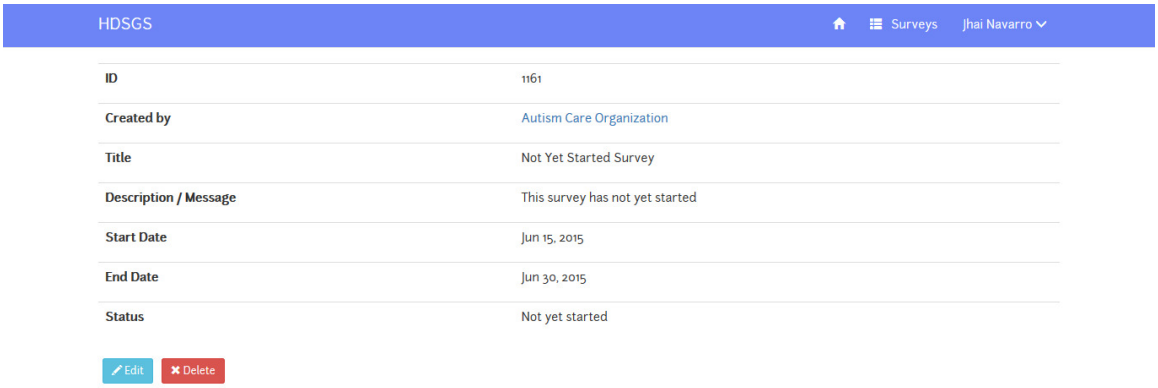


Figure 22: View page of survey that has not yet started, accessed by a research leader

For a survey that is ongoing, public users will see the page in Figure 23. Meanwhile, contributors will see the page shown in Figure 24 which has a button for answering the survey. Moreover, research leaders can view the page shown in Figure 25 which includes buttons to Deactivate or Delete the survey.

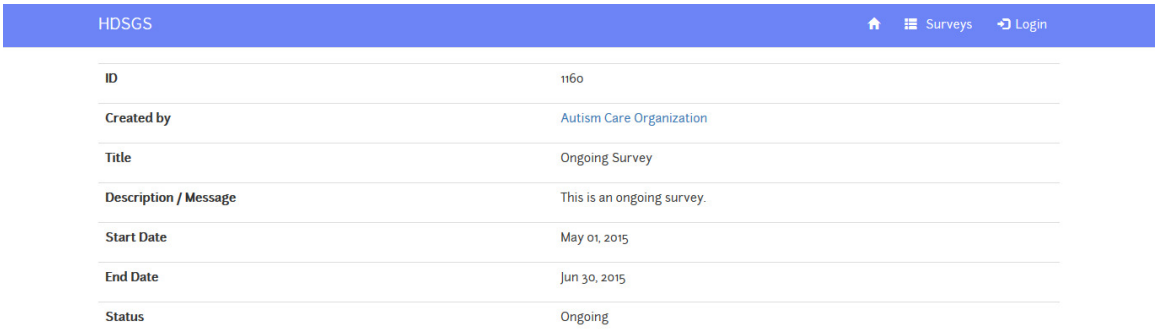


Figure 23: View page of survey that is ongoing, accessed by a public user

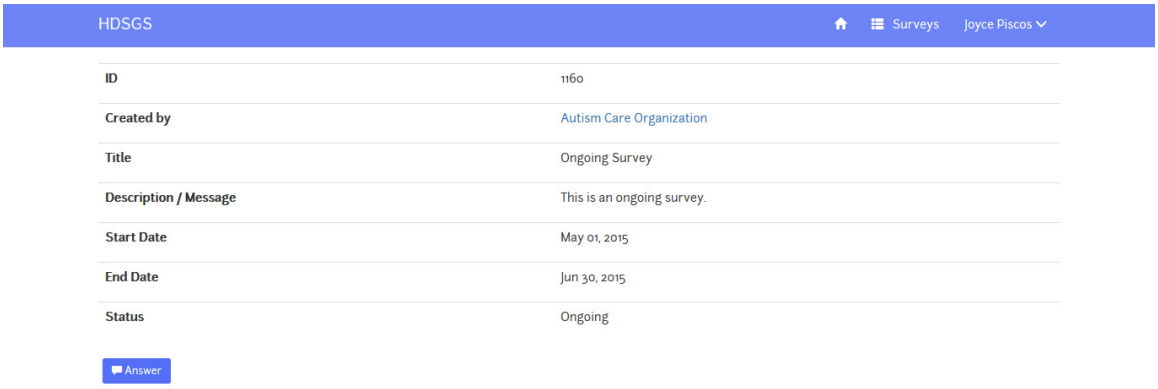


Figure 24: View page of survey that is ongoing, accessed by a contributor

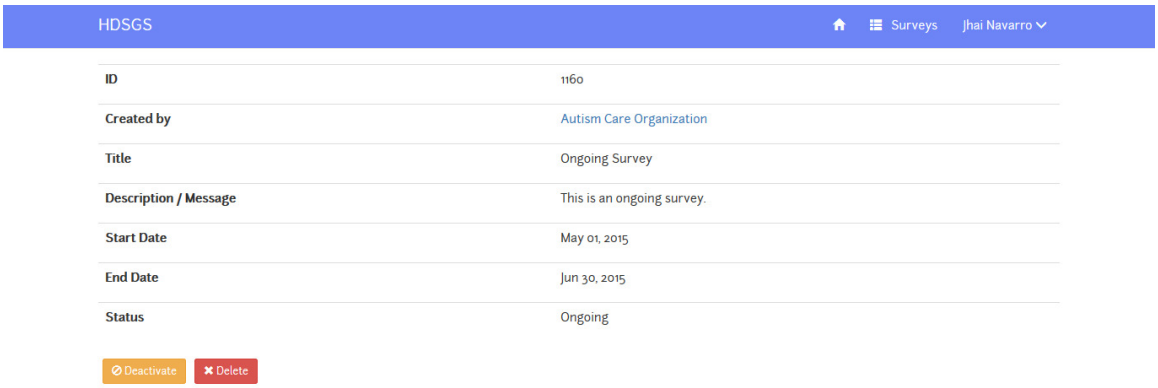


Figure 25: View page of survey that is ongoing, accessed by a research leader

For a survey that has already ended, public users and contributors will see the page shown in Figure 26. Included here is a button for viewing the results of the survey. On the other hand, research leaders can view the page seen in Figure 27 which also includes a button for viewing the results, and for deleting the survey.

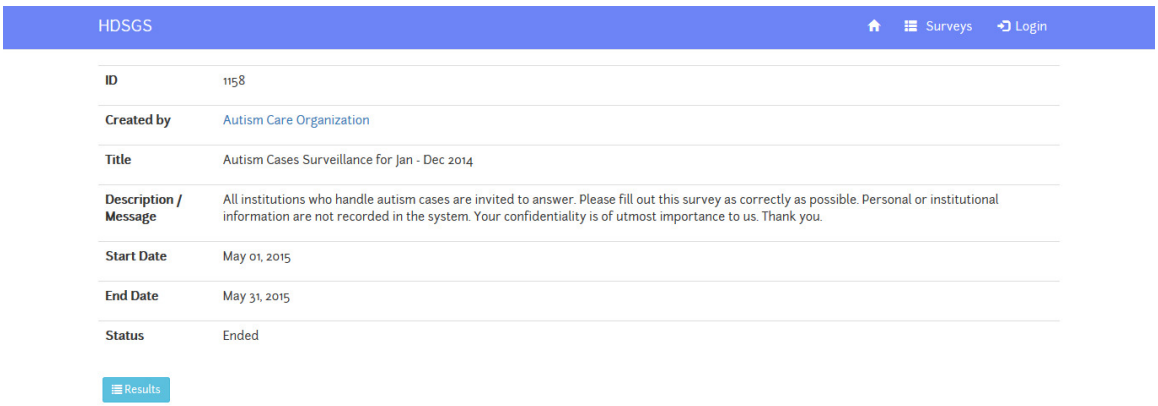


Figure 26: View page of survey that has ended, accessed by a public user or contributor

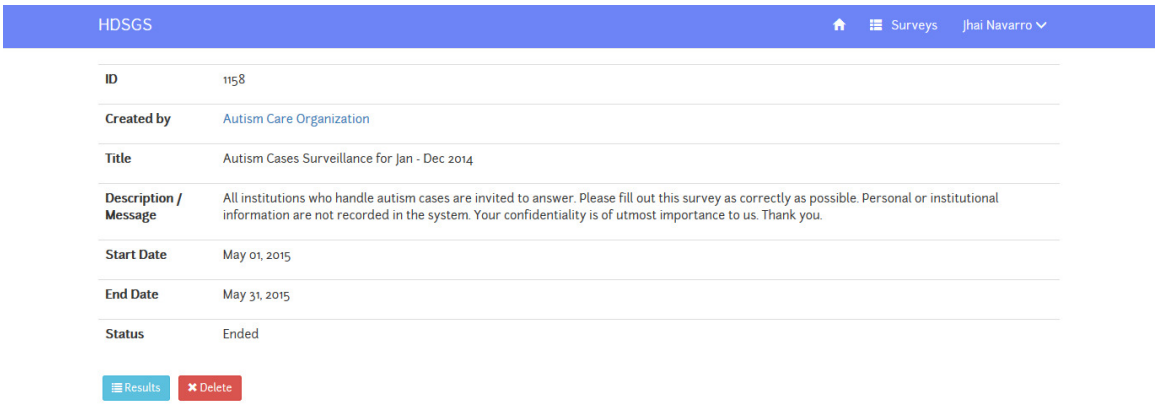


Figure 27: View page of survey that ended, accessed by a research leader

E. Update Survey

The page for updating a survey is only available to the creator of the survey. It can be reached by clicking the Edit button on the page for viewing the details of the survey. If the form has errors, the appropriate messages will be shown upon clicking the Submit button. Else, it will redirect back to survey's page which will then contain the message that the update was successful. This can be seen on Figure 29. On the other hand, clicking the Cancel button will simply redirect back to viewing the details page of the survey.

Figure 28: Update survey page

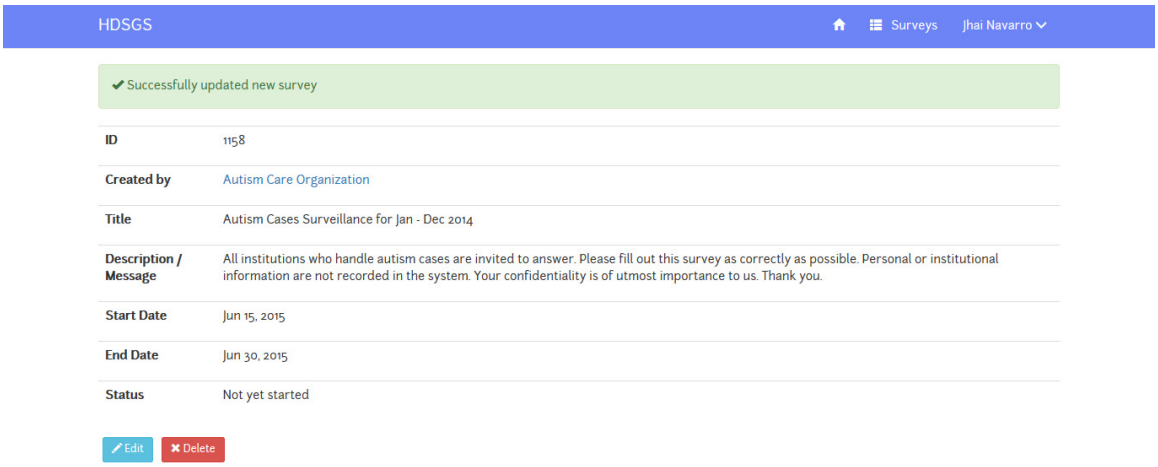


Figure 29: Successfully updated survey page

F. Answer Survey

Answering a survey can only be done by a contributor. It can be accessed by clicking the Answer button on the survey's page. Clicking the submit button will redirect back to survey's page which will contain the message that the answers have been recorded. This can be seen on Figure 31. Otherwise, clicking on the Cancel button will simply redirect back to viewing the details of the survey.

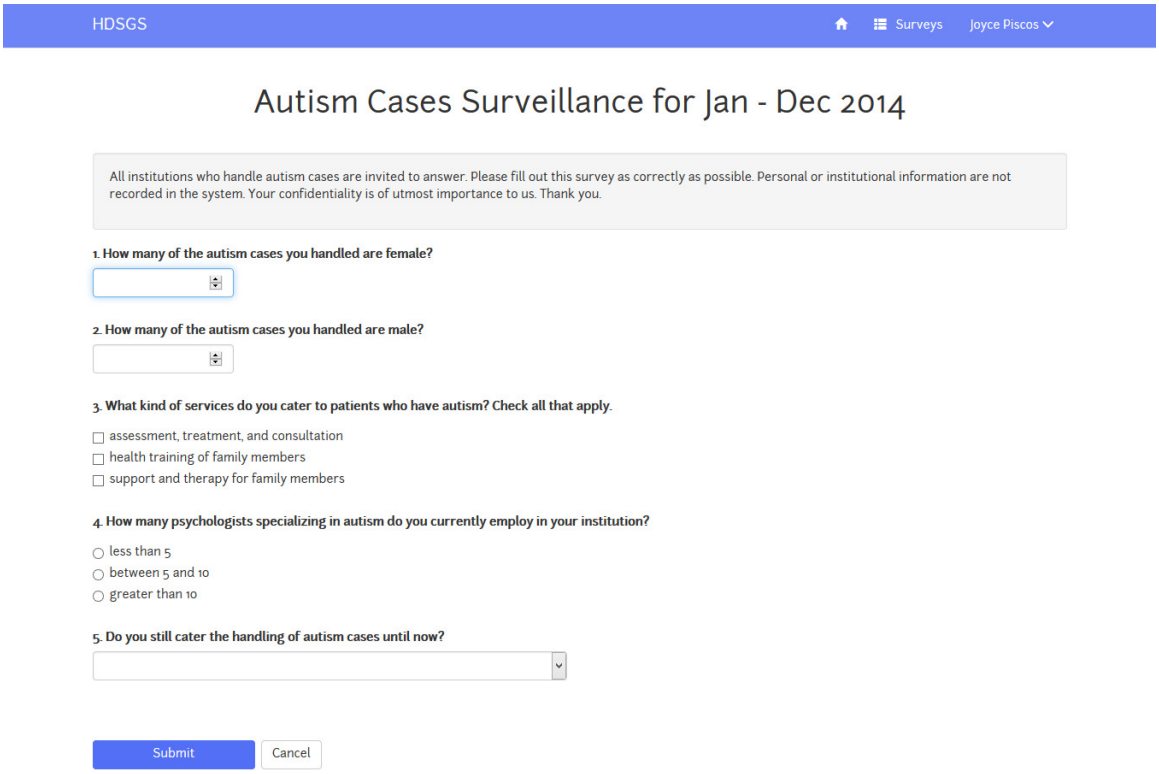


Figure 30: Answer survey page

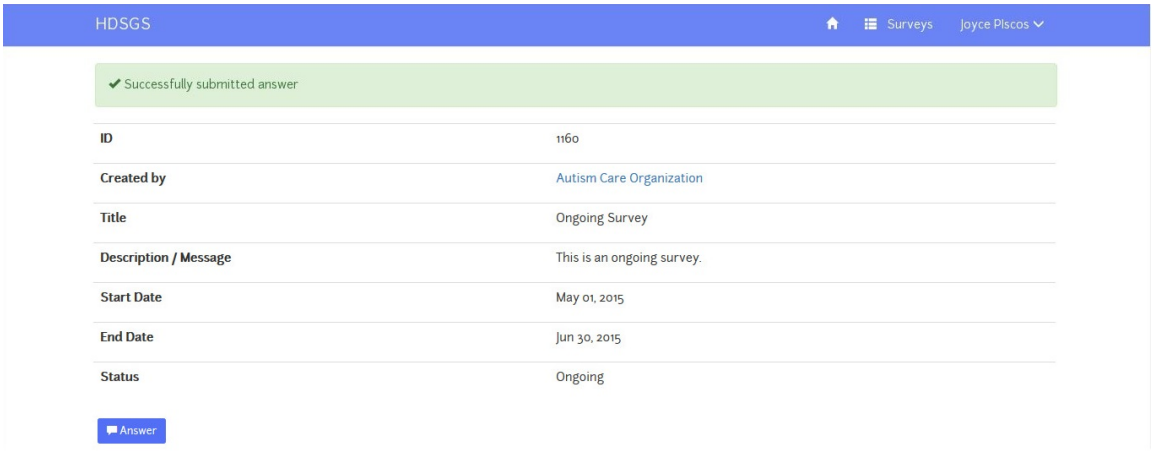


Figure 31: Successfully answered survey page

G. Deactivate Survey

Deactivation is done only by the creator of the survey. Clicking on the Deactivate button will open up a modal to confirm or cancel the action. Note that a survey cannot be reactivated once it has been deactivated or ended.

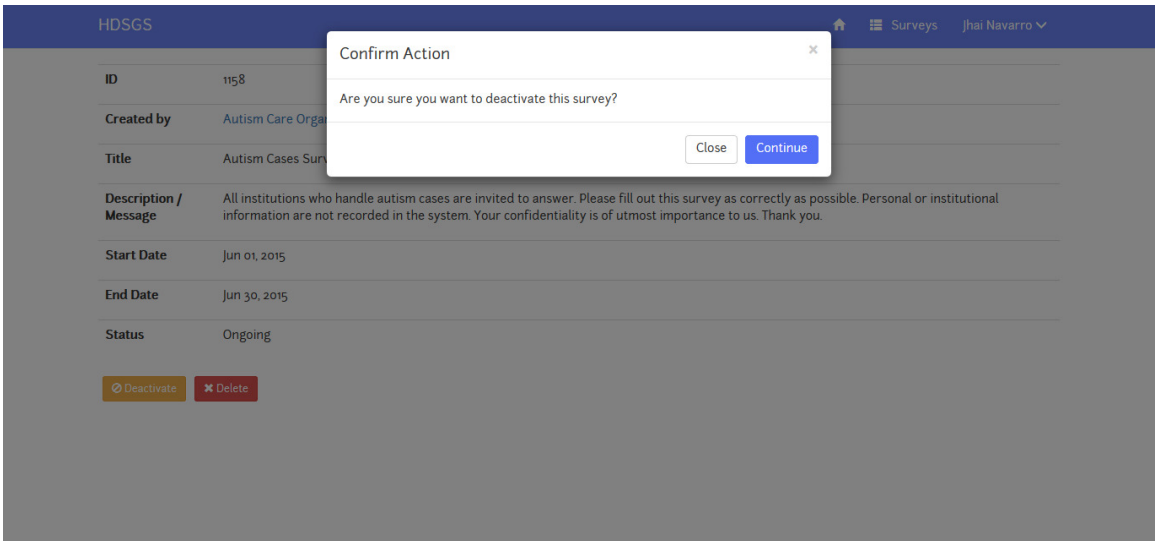


Figure 32: Confirm deactivation of survey

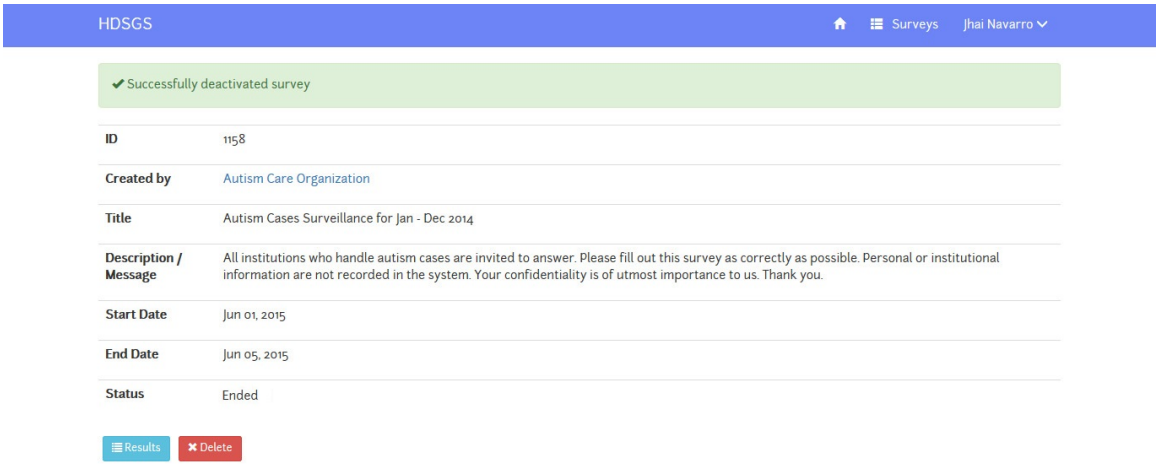


Figure 33: Successfully deactivated survey

H. Delete Survey

Similar with deactivation, deletion of a survey can only be performed by its creator. Clicking on the Delete button will also open a modal for the confirmation of the action. There is no recovery for a deleted survey.

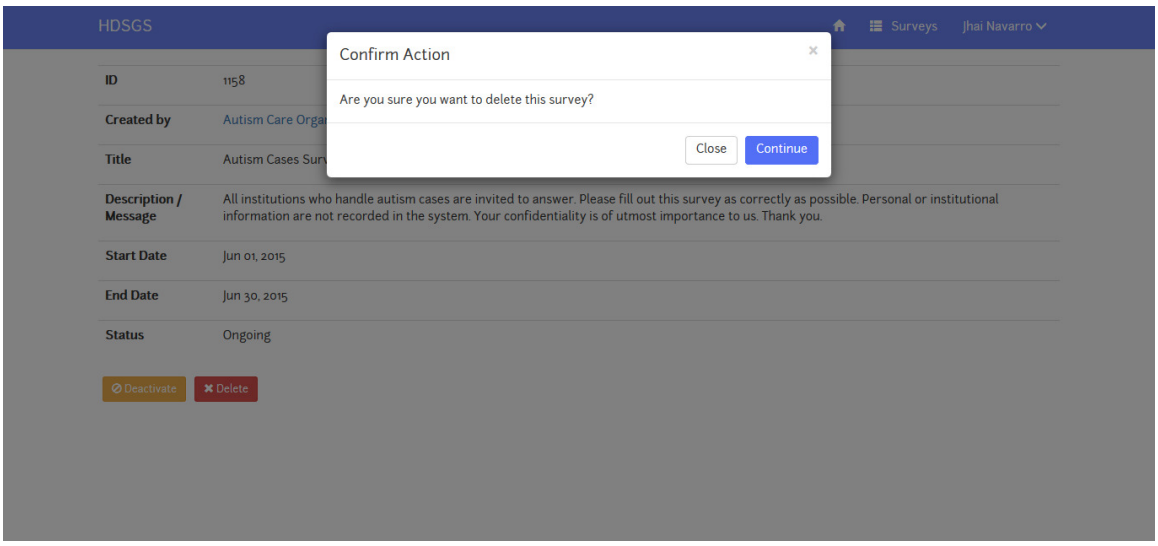


Figure 34: Confirm deletion of survey

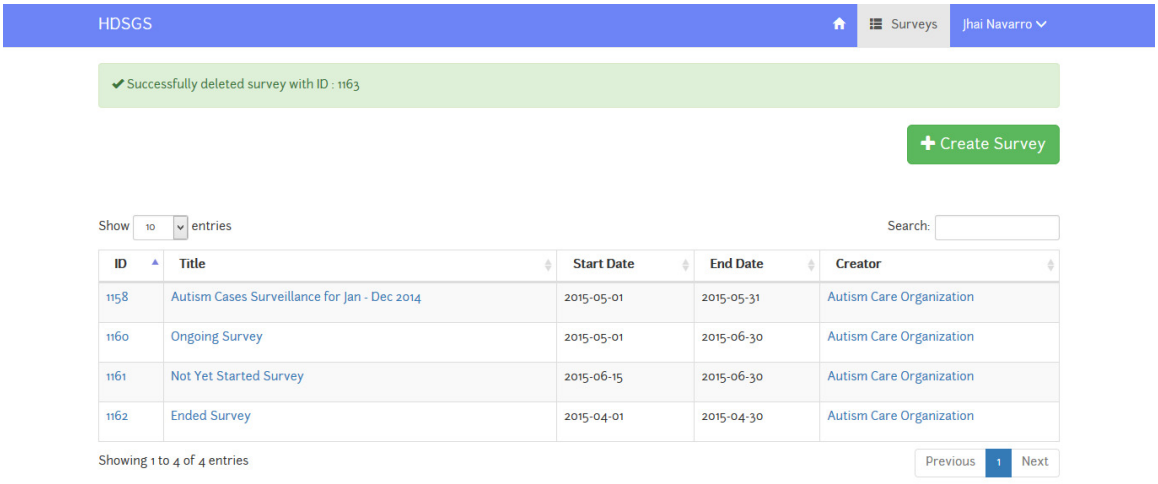
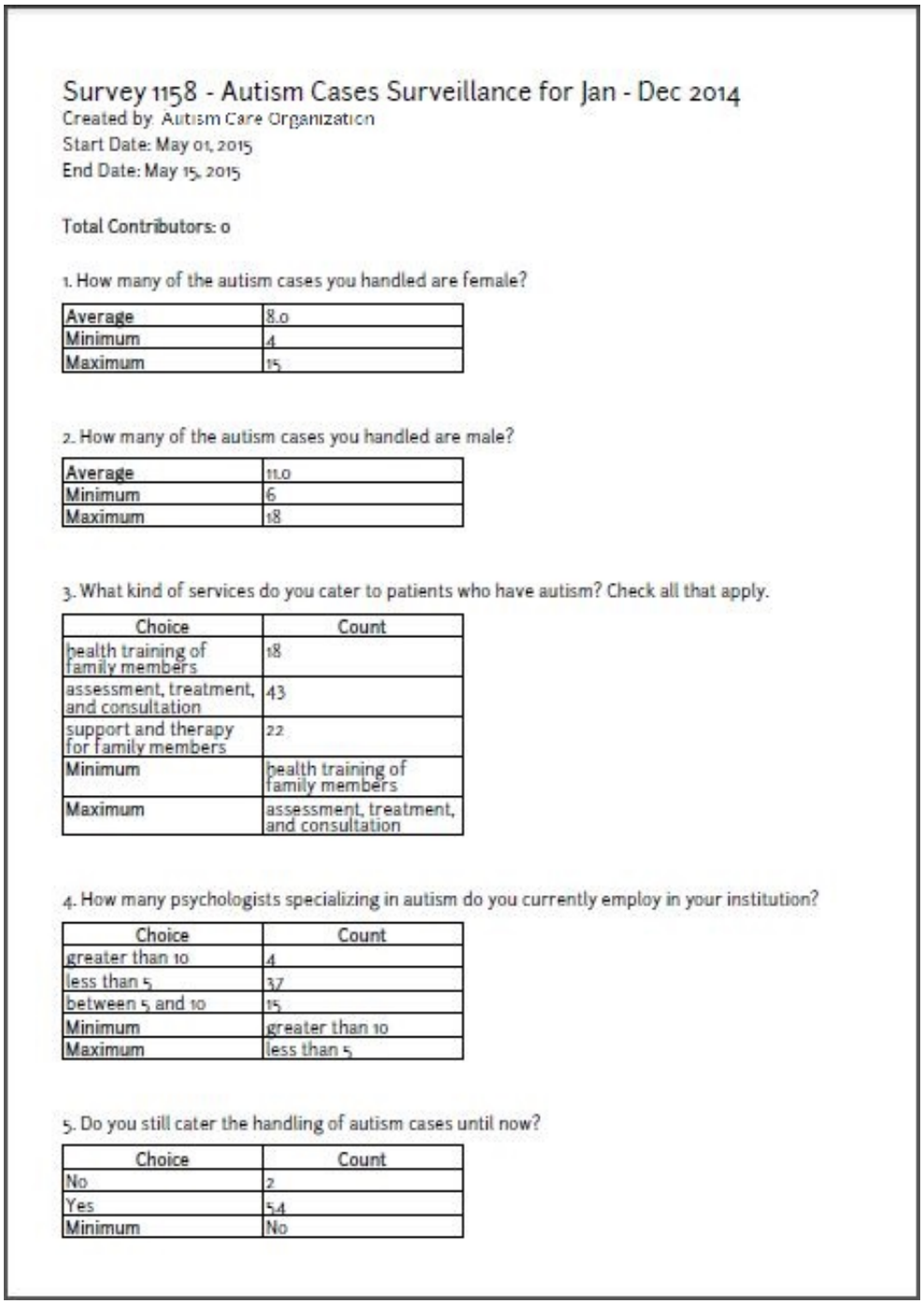


Figure 35: Successfully deleted survey

I. View Survey Results

Survey results are available to all users except for the admin. It can be viewed by clicking on the Results button that can be found on the survey's page. A PDF copy of the results can be obtained by clicking the Download Results button. An example of this can be seen in Figure 37.

Figure 36: Survey results page



Survey 1158 - Autism Cases Surveillance for Jan - Dec 2014

Created by: Autism Care Organization
Start Date: May 01, 2015
End Date: May 15, 2015

Total Contributors: 0

1. How many of the autism cases you handled are female?

Average	8.0
Minimum	4
Maximum	15

2. How many of the autism cases you handled are male?

Average	11.0
Minimum	6
Maximum	18

3. What kind of services do you cater to patients who have autism? Check all that apply.

Choice	Count
health training of family members	18
assessment, treatment, and consultation	43
support and therapy for family members	22
Minimum	health training of family members
Maximum	assessment, treatment, and consultation

4. How many psychologists specializing in autism do you currently employ in your institution?

Choice	Count
greater than 10	4
less than 5	37
between 5 and 10	15
Minimum	greater than 10
Maximum	less than 5

5. Do you still cater the handling of autism cases until now?

Choice	Count
No	2
Yes	54
Minimum	No

Figure 37: Sample PDF of a survey's results

J. View All Users

The page in Figure 38 is only available to the admin. Included here is a button for creating a new user. Also, clicking the username of a user in the table will redirect the admin to view that user's page.

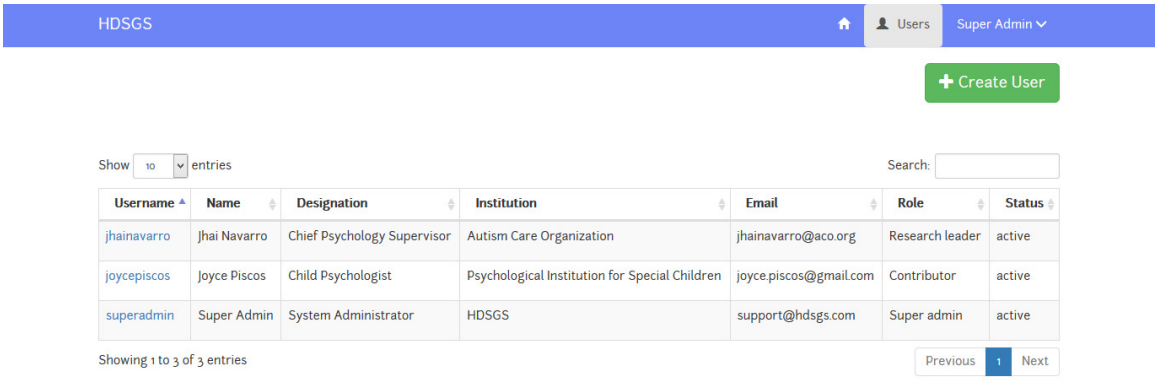


Figure 38: View all users page

K. Create User Account

User accounts are created by the admin. This page can be accessed by clicking the Create User button found on Figure 38. If the form has errors, the appropriate messages will be shown upon clicking the Submit button. Else, the admin will be redirected to the page for viewing the details of the newly created user which can be seen in Figure 40. Meanwhile, clicking the Cancel button will simply redirect back to viewing the list of all existing users.

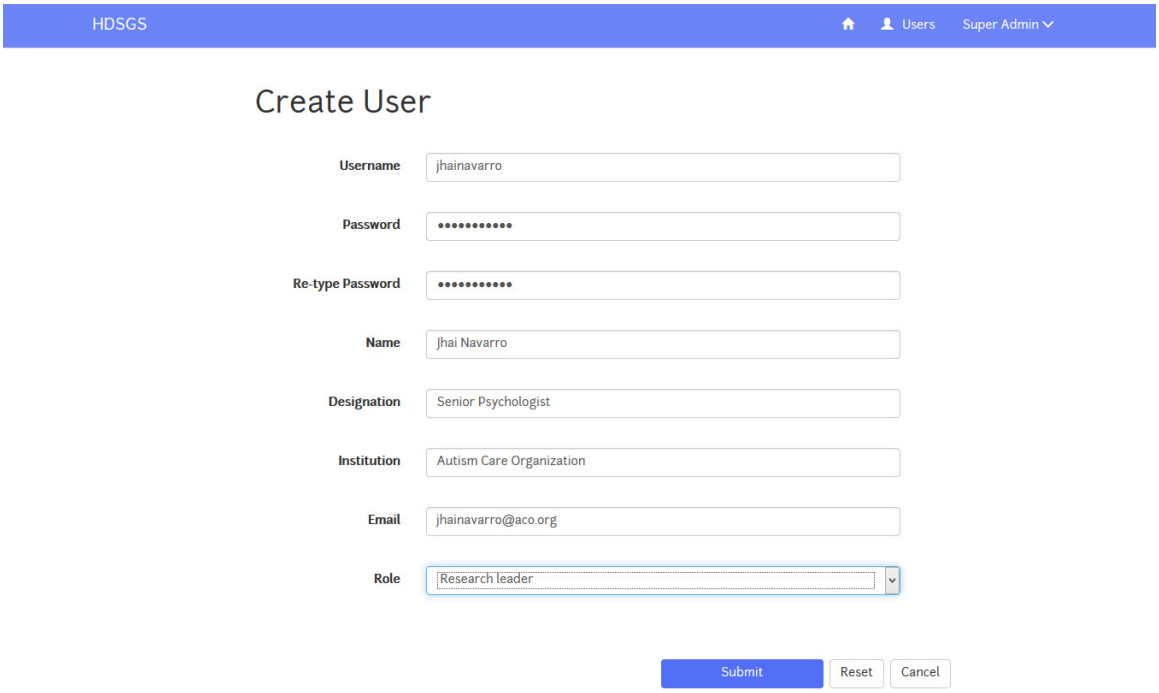


Figure 39: Create user page

Figure 40: Successfully created user page

L. View User Account

Contributors can view their own account page by clicking the View Profile link on the navigation bar. Included in this page are buttons for editing, and deactivating or reactivating their account as seen in Figure 41. Also, there is also a link for viewing the list of surveys which they have answered.

Figure 41: View account of contributor

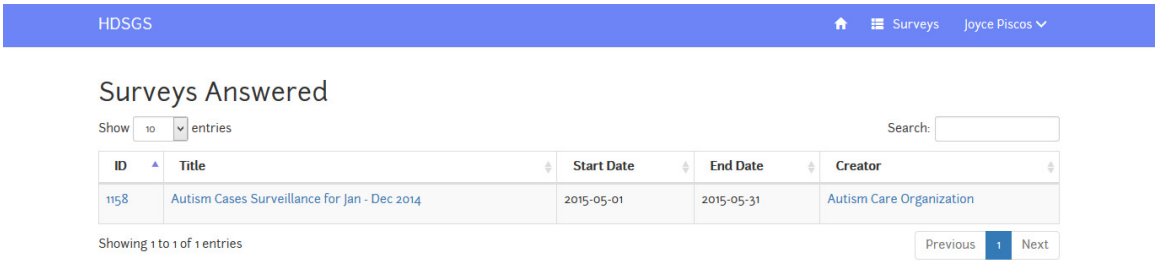


Figure 42: View answered surveys of a contributor

In the same fashion, research leaders can also view their own account by clicking the View Profile link on the navigation bar. Their account page also contains buttons for editing their profile, or toggling their account status. However, the link available to them is for viewing the surveys which they have created. These pages can be seen in Figures 43 and 44.

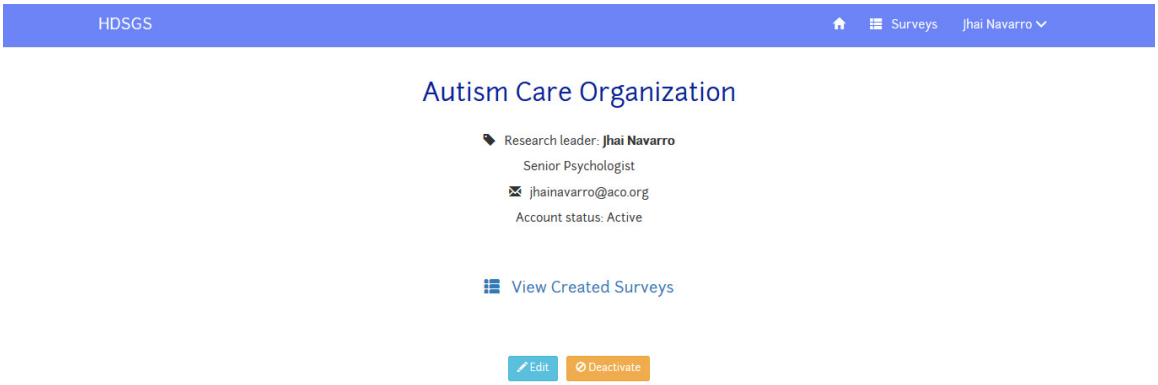


Figure 43: View account of research leader

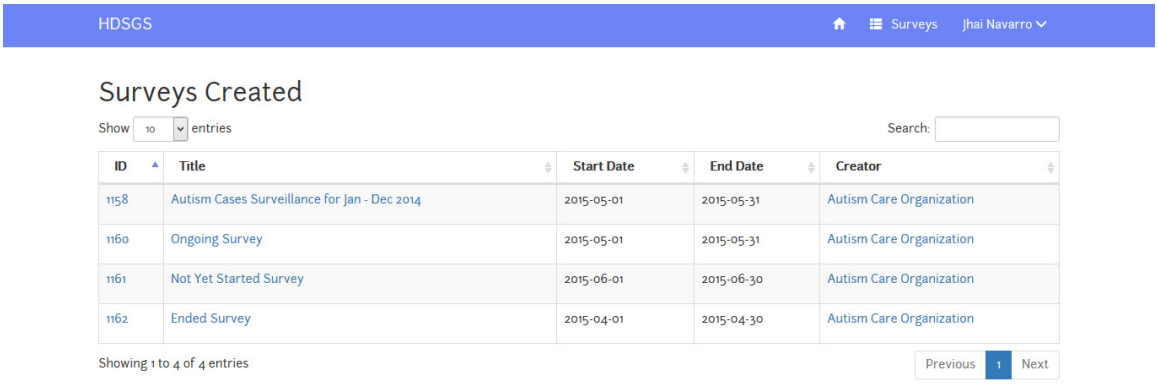


Figure 44: View created surveys of a research leader

Similar to contributors and research leaders, the admin can also view, update, deactivate or reactivate their own account. However, they have the additional button for deleting their account. Furthermore, the admin can view any of the user accounts available in the system. This means that they can also update, deactivate, reactivate, and delete any existing user account.

Figure 45: View account, accessed by the admin

M. Update User Account

This page can be viewed by either clicking on the Update Profile link on the navigation bar, or the Edit button on the user's page. The user can update their account details and their password separately. If the form has errors, the appropriate messages will be shown upon clicking the Submit button. Else, the user will be redirected to the page for viewing the updated details of the user which can be seen in Figure 48. Meanwhile, clicking the Cancel button will simply redirect back to viewing the user's page.

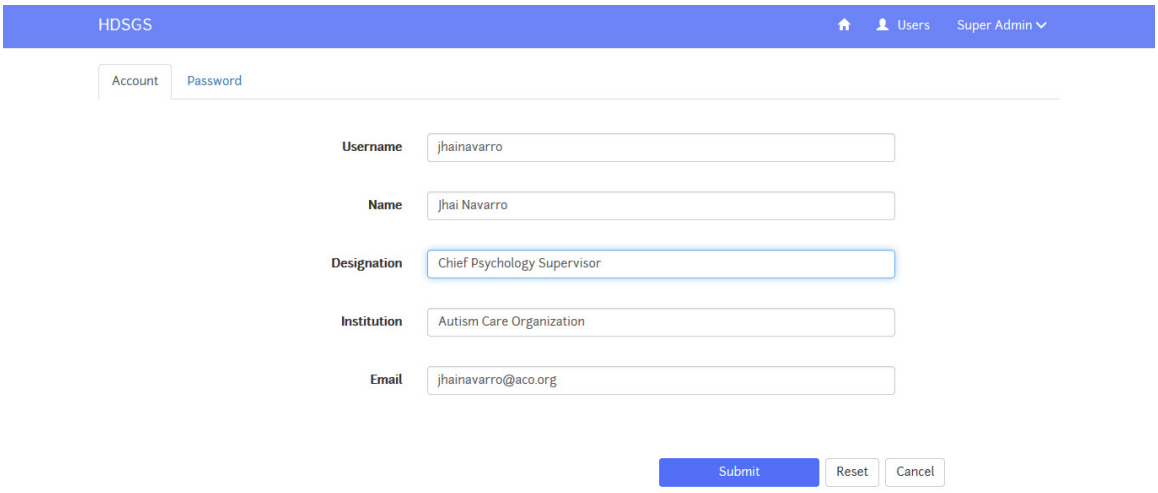


Figure 46: Update user account details

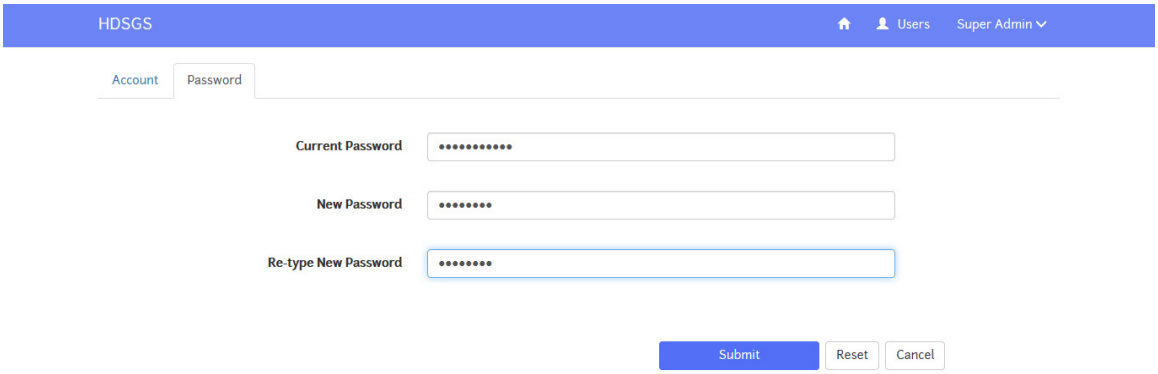


Figure 47: Update user password

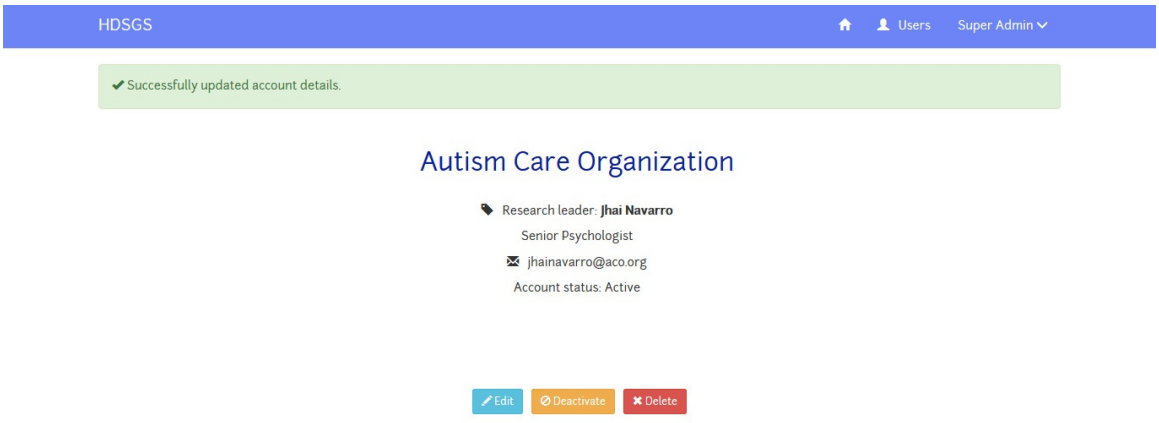


Figure 48: Successfully updated user account

N. Deactivate User Account

Clicking the Deactivate button will open a modal to confirm the action. Deactivating a contributor's account will hinder that user to answer any survey, or make any updates to their account details. Meanwhile, a deactivated research leader can neither create, update, deactivate nor delete any of the surveys he or she has created. Moreover, that research leader will also be unable to update, deactivate nor delete their account. Furthermore, the admin can only deactivate accounts other than his or her own.

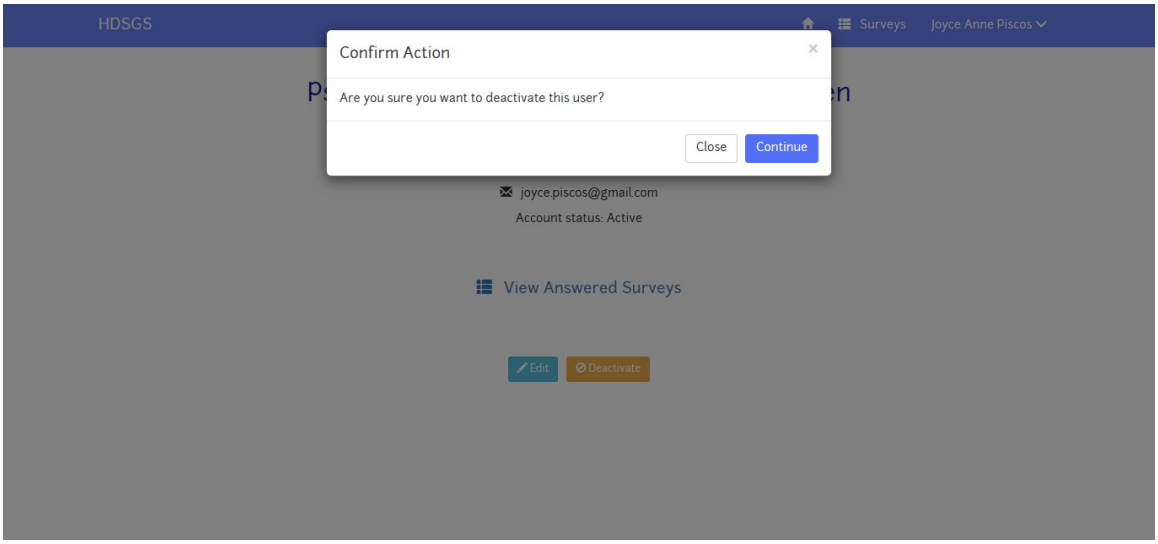


Figure 49: Confirm deactivation of account

Figure 50: Successfully deactivated account

O. Reactivate User Account

Similar to deactivation, reactivating an account will first need confirmation. On the other hand, this functionality is only available to the admin, unlike deactivation.

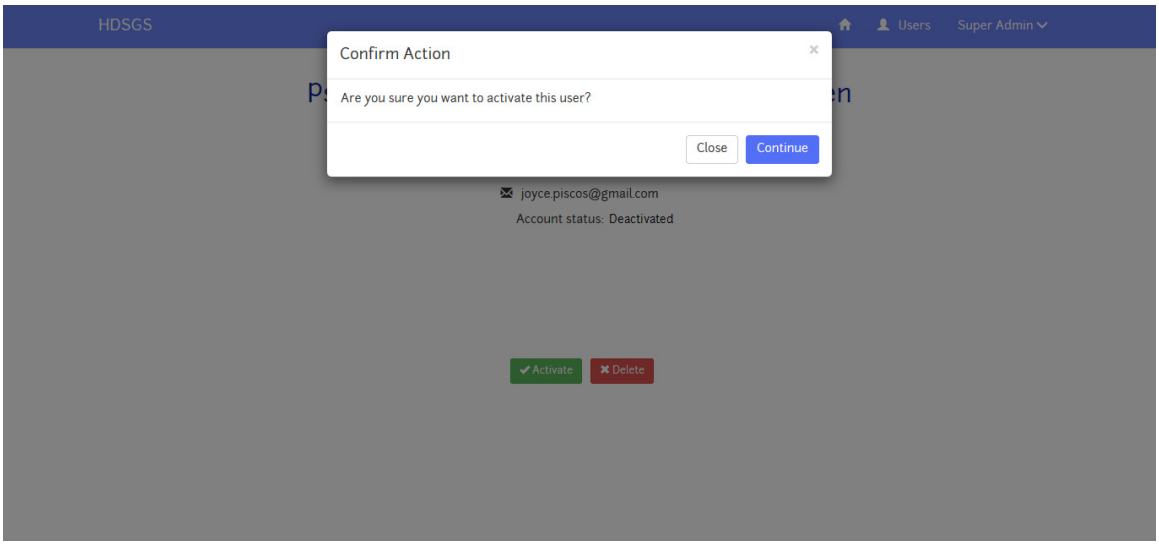


Figure 51: Confirm reactivation of account

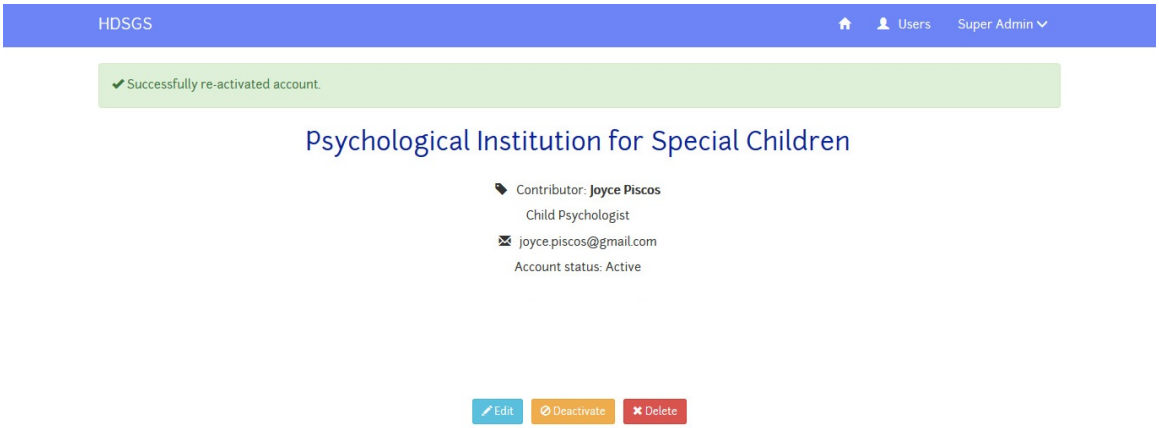


Figure 52: Successfully reactivated account

P. Delete User Account

The deleting of account is a functionality of the admin only, and can only be applied to accounts other than his or her own. Clicking the Delete button on user's page will open a modal for confirmation of the action. Note that if a research leader is deleted, all the surveys he or she has created will also be deleted. In contrast, deleting a contributor does not affect the results of any of the surveys he or she has answered.

After successful deletion of an account, the admin will be redirected to the list of all existing users.

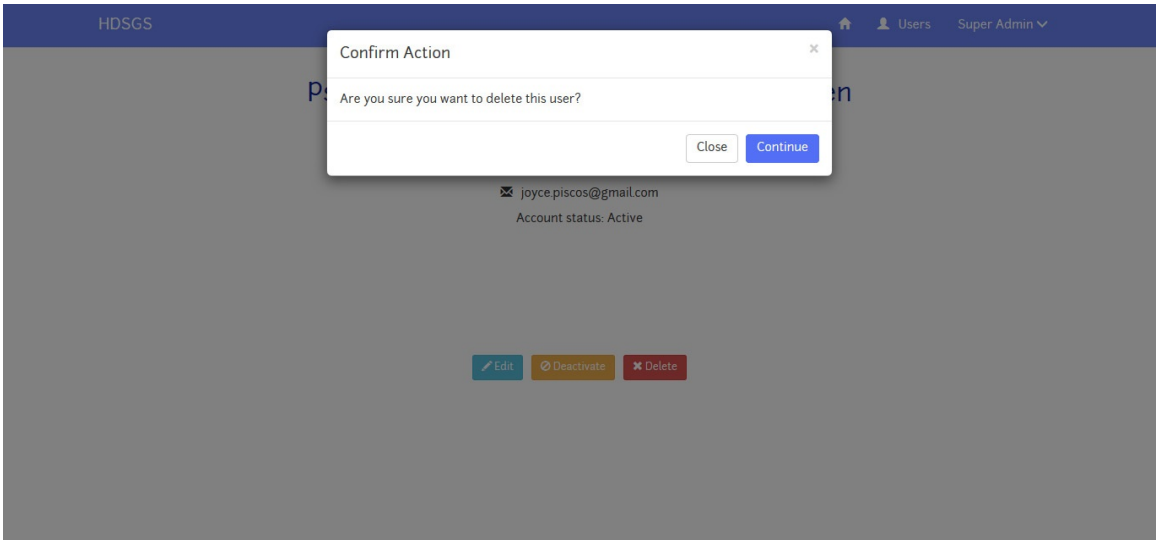


Figure 53: Confirm deletion of account

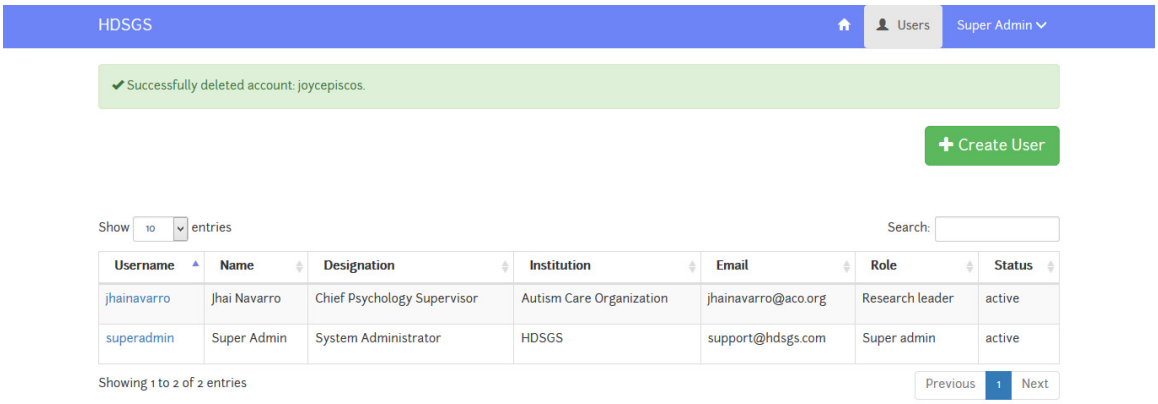


Figure 54: Successfully deleted account

VI. Discussions

HDSGS or Health Data Statistics Generation System is a web-based application tool that provides a way to securely pool and aggregate health data statistics. Specifically, it is an application used to host online surveys which need to maintain confidentiality on the responses of the respondents (e.g. health data information). The storage and computation of survey results is done by the Sharemind system. HDSGS has four roles: research leader, contributor, admin, and public user.

HDSGS provides the functionality of creating and managing surveys to research leaders. Research leaders can only manage the surveys they have created. However, they can still view the details and results of other surveys. Management of surveys includes updating, deactivating, and deleting of surveys. Once a survey has started, contributors will be able to answer it. Furthermore, after the survey has ended, the results will then be available for viewing. Research leaders, contributors, and public users all have the privilege of viewing the details of any existing survey. Additionally, they can also view and download the survey results.

For the admin, HDSGS gave it the exclusive authority to create and manage user accounts. User accounts are given to research leaders, and contributors. There is only one admin in the system; admin accounts cannot be created nor deleted by the system. In addition to creating new accounts, the admin can also deactivate, reactivate, and delete any account except their own. Nevertheless, the owner of an account also has the privilege to view, update, and deactivate it.

The storage of survey answers is hosted by the internal database of Sharemind. Sharemind provides the security of these information. To elaborate, these information to be stored are first secret-shared, and then these shares are distributed to the three miner hosts. Sharemind uses additive secret sharing over the ring of 32-bit integers which has been proven to be sure given that no two miners collude with each other throughout the computation. [11, 29, 12]

Given these points, HDSGS can be a helpful tool in securely pooling health information that is most beneficial for institutions and individuals in the medical field. They can conduct online surveys with HDSGS, which stores the survey answers and computes the results via secure multiparty computation. In essence, health data gathering can be made easier and more secure using HDSGS. Some of the possible uses of HDSGS can be for disease surveillance, drug effectiveness, and general health information collection, among others. Since the results of the surveys can be made publicly available without revealing the identities of respondents, HDSGS is not only useful to medical institutions, but also to information seekers as well.

The use of Spring made it much easier to develop HDSGS. Being a widely used framework, there are many references to aid in coding. Especially with the use of Spring Boot and Spring's in-depth documentations, the back-end of HDSGS was developed with little difficulty. However, the same cannot be said about Sharemind. Being a relatively new SMC framework, there are few available articles and applications using Sharemind. As a result, the development of HDSGS's secure multiparty protocols for computing the average, minimum, maximum, and tally of survey answers took much effort.

The primary issue when performing the secure multiparty computation for the survey results is its running time. Tables 9 and 10 show an approximation of the average time it takes when aggregating survey answers. Three runs are performed for each SMC task. Note that the time it takes to render the corresponding HTML pages, such as the survey results page, is not included in the time measurements.

Given 1 question with 10, 20, 50, and 100 answers per question:

It can be observed from Table 9 that the measurements increase almost linearly as the number of answers increase. Moreover, it can be expected that as the number of questions increases, the measurements will also increase as the aggregation of answers is done one question at a time, and not in parallel.

SMC Task	10 answers	20 answers	50 answers	100 answers
Record Answers	11.99 s	14.07 s	28.33 s	80.06 s
Average	1.50 s	1.08 s	0.90 s	1.01 s
Minimum	1.416 s	1.70 s	1.96 s	2.78 s
Maximum	2.785 s	3.74 s	8.16 s	15.43 s
Tally	9.846 s	19.04 s	47.97 s	98.02 s

Table 9: Time measurement (in seconds) of different SMC tasks for 1 question

Given 10 questions with 10, 20, 50, and 100 answers per question:

SMC Task	10 answers	20 answers	50 answers	100 answers
Record Answers	46.72 s	91.43 s	235.80 s	528.04 s
Average	7.17 s	9.25 s	8.80 s	10.51 s
Minimum	14.40 s	101.91 s	16.91 s	206.79 s
Maximum	17.46 s	16.80 s	15.73 s	1127.04 s
Tally	96.31 s	216.25 s	451.83 s	1257.21 s

Table 10: Time measurement (in seconds) of different SMC tasks for 10 questions

Predictably, increasing the number of questions gave longer computing times. The measurements in Table 10 are for 10 questions. Comparing with Table 9, we can see that there is a slightly over 10 times increase in the time measurements. These data show the slow execution of computing the survey results, especially when computing for the tally of answers.

From the performance results above, it can be observed that as the number of questions increases, the longer it takes for the results to be computed. To work around this issue, the survey results are computed only once—after deactivation, or on its end date. The results are then stored in the MySQL database for easier and shorter time of retrieval. Calculation of the survey results was implemented to be done only after the deactivation of a survey. In effect, it is only then that this issue should be encountered.

VII. Conclusions

Health Data Statistics Generation System (HDSGS) is a web-based tool that was developed to be an avenue for conducting privacy-preserving health data collection, and statistics aggregation through secure multiparty computation. It can be most helpful to institutions that want to compute a particular statistical function over sensitive health data input.

The statistics that can be generated by the system is the average, minimum, maximum, and tally of answers to a survey question. This is performed through secure multiparty computation to ensure the confidentiality of survey answers. The Sharemind framework was used in the development of these protocols. With the help of Sharemind, the correct results given are correct, under the assumption that there was no manipulation on any of the miners' databases.

HDSGS follows the three-tier web application architecture. These are the database, service, and presentation tiers. The main framework used in developing HDSGS is Spring. It is present all throughout the three layers.

For the data access methods, they were developed with Spring Data, and used Hibernate to communicate with the MySQL database. The MySQL database was used to store user information, survey information, and the generated survey results.

In addition to the service methods that were also created with Spring, the Sharemind controllers were integrated for executing the calculation of survey results. After the results are computed, they are to be stored in the MySQL database so that future viewing of results will have faster loading time. This is to opposed to computing the survey results every time the page is requested, which was shown in Table 9 and 10 to be very slow.

For the presentation layer, controller methods are similarly coded with the help of Spring. They are also coupled with Thymeleaf which is a template engine for creating the HTML pages.

Overall, the usage of the Spring framework enabled HDSGS to be developed with little difficulty. It also helped provide the functionalities needed by the system. In the same way, the Sharemind framework ensured the preservation of confidentiality that is inherent in the survey answers, at the same time providing the needed correct survey results.

VIII. Recommendations

HDSGS is a potentially very useful tool for securely generating of health data statistics. However, the system can still be enhanced by optimizing the algorithms used for the Sharemind controllers since they consume a lot of time, as was shown in the previous section. The release of a new version of Sharemind may be timely for this purpose. However, migration to the new version of Sharemind may require some effort because of the new syntax and functions implemented in Sharemind 3.

```
void main () {                  // main function
        private uint a, b, c;   // private data
        a = b + c;              // private computation
        public uint d;          // public data
        d = declassify (a);     // private -> public
        publish (d);            // send to client
}
```

Listing 1: Sharemind 2

```
kind additive3pp;               // declare PDK
domain pd_a3p additive3pp;      // declare PD
void main () {                  // main function
        pd_a3p uint a, b, c;    // private data
        a = b + c;              // private computation
        public uint d;          // public data
        d = declassify (a);     // private -> public
        publish (d);            // send to client
}
```

Listing 2: Sharemind 3

In Sharemind 2, variables can only either be one of the two privacy types: `public` or `private`. However, in Sharemind 3, they introduce protection domains (PD) and

protection domain kinds (PDK). According to the documentation of Sharemind 3, a PDK contains protocols, data representations and algorithms that are used for the storage and computation of private data. Furthermore, a PD is a set of protected data for which there are defined protocols used for calculations on that data while keeping them private. Morevover, each PD belongs to some kind of PDK. PDKs and PDs are mainly used in Sharemind 3 for polymorphism and function overloading, which are new additions over Sharemind 2. The `private` privacy type in Sharemind 2 can be thought of as a PD called "`private`" in an additive 3-party PDK. [30]

On the other hand, a particularly significant improvement is that Sharemind 3 already has built-in functions for calculating the minimum and maximum elements of a given matrix. These can replace the created functions developed for computing the minimum and maximum answer to a survey question in the current implementation of HDSGS.

In addition to using Sharemind 3, ranking of question choices could also be added for when creating or updating surveys to provide another option for gauging priorities or effectiveness, among others, and to better emulate manual surveys.

IX. Bibliography

[1] K. El Emam, J. Hu, J. Mercer, L. Peyton, M. Kantarcioglu, B. Malin, D. Buckeridge, S. Samet, and C. Earle, "A secure protocol for protecting the identity of providers when disclosing data for disease surveillance," *Journal of the American Medical Informatics Association*, vol. 18, pp. 212–217, May 2011.

[2] K. B. Frikken, "Secure Multiparty Computation," in *Algorithms and Theory of Computation Handbook*, ch. 14, pp. 14–1–14–16, Chapman & Hall/CRC, second ed., 2010.

[3] Y. Lindell and B. Pinkas, "Secure Multiparty Computation for Privacy-Preserving Data Mining," *The Journal of Privacy and Confidentiality*, vol. 1, no. 1, pp. 59–98, 2009.

[4] R. Talviste, *Deploying secure multiparty computation for joint data analysisa case study*. PhD thesis, University of Tartu, 2011.

[5] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "SEPIA : Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics," in *Proceedings of the 19th USENIX Conference on Security*, 2010.

[6] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science*, pp. 160–164, Ieee, Nov. 1982.

[7] J. Damgård, Ivan and Geisler, Martin and Krøigaard, Mikkel and Nielsen, "Asynchronous Multiparty Computation : Theory and Implementation," in *Public Key Cryptography*, pp. 160–179, Springer-Verlag, 2009.

[8] A. Jarrous and B. Pinkas, "Canon-MPC, a system for casual non-interactive secure multi-party computation using native client," in *Proceedings of the 12th*

ACM workshop on Workshop on privacy in the electronic society, (New York, New York, USA), pp. 155–166, ACM Press, 2013.

[9] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay A Secure Two-Party Computation System," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, pp. 20–20, 2004.

[10] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP A System for Secure Multi-Party Computation," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pp. 257–266, 2008.

[11] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, pp. 192–206, 2008.

[12] R. Jagomägis, *SecreC : a Privacy-Aware Programming Language with Applications in Data Mining*. PhD thesis, University of Tartu, 2010.

[13] P. Bogetoft, D. L. Christensen, I. Damgå rd, M. Geisler, T. Jakobsen, M. Krø igaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, "Secure Multiparty Computation Goes Live," in *Financial Cryptography and Data Security*, pp. 325–343, Springer-Verlag, 2009.

[14] A. Mauland, T. I. Reistad, and S. F. Mjø lsnes, "Realizing Distributed RSA Key Generation using VIFF," in *NISK 2009 Proceedings of the 2nd Norwegian Security Conference*, pp. 122–135, 2009.

[15] I. Damgå rd and M. Keller, "Secure Multiparty AES," in *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, pp. 367–374, 2010.

[16] A. Based, S. F. Mjø lsnes, and M. B. Stenbek, "Multiparty ballot counting in an Internet voting scheme," *Acta Electrotechnica et Informatica*, vol. 10, no. 4, pp. 10–15, 2010.

[17] H. v. Vegge, *Realizing Secure Multiparty Computations*. PhD thesis, Norwegian University of Science and Technology, 2009.

[18] S. Halevi and B. Pinkas, "Secure Computation on the Web : Computing without Simultaneous Interaction," in *Proceedings of the 31st Annual Conference on Advances in Cryptology*, pp. 132–150, 2011.

[19] A. Shamir, "How to Share A Secret," *Communications of the ACM*, vol. 22, pp. 612–613, Nov. 1979.

[20] M. Djatmiko, D. Schatzmann, X. Dimitropoulos, A. Friedman, and R. Boreli, "Collaborative Network Outage Troubleshooting with Secure Multiparty Computation," *Communications Magazine, IEEE*, vol. 51, no. 11, pp. 78–84, 2013.

[21] R. Jagomägis, *A programming language for creating privacy-preserving applications*. PhD thesis, University of Tartu, 2008.

[22] D. Bogdanov, R. Talviste, and J. Willemson, "Deploying secure multi-party computation for financial data analysis (extended version)," in *Financial Cryptography and Data Security*, pp. 57–64, Springer Berlin - Heidelberg, 2012.

[23] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo, "A New Way to Protect Privacy in Large-scale Genome-wide Association Studies," *Bioinformatics*, vol. 29, pp. 886–893, Apr. 2013.

[24] D. Bogdanov and R. Talviste, "A prototype of online privacy-preserving questionnaire system," tech. rep., Cybernetica, 2010.

[25] R. Talviste and D. Bogdanov, "An improved method for privacy-preserving web-based data collection," tech. rep., Cybernetica, 2009.

[26] R. Dong, *Secure Multiparty Computation*. PhD thesis, Bowling Green State University, 2009.

[27] M. Bramer, *Principles of Data Mining*. Springer-Verlag, 2007.

[28] A. McAfee and E. Brynjolfsson, "Big Data: The Management Revolution," 2012.

[29] D. Bogdanov and R. Jagomägis, "A universal toolkit for cryptographically secure privacy-preserving data mining," in *Proceedings of the 2012 Pacific Asia Conference on Intelligence and Security Informatics*, no. 8124 in PAISI'12, pp. 112–126, 2012.

[30] D. Bogdanov, P. Laud, and J. Randmets, "Domain-Polymorphic Programming of," in *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security*, pp. 1–24, 2014.

X. Appendix

A. Source Code

1. SecreC Scripts

1. survey_dataEntry.sc

```
/* Inserts the survey answer to the Sharemind database. */

void main(private uint32 answer, private uint32
     question_number, private uint32 survey_id)
{
  public string db;
  db = "hdsgs";

  public bool exists;

  exists = dbExists(db);
  if (!exists)
  {
    print("Database does not exist!");
    publish("error", "Database does not exist!")
     ;

    return;
  }

  dbLoad(db);

  public string tbl;
  tbl = "survey_answers";

  exists = dbTableExists(tbl);
  if (!exists)
  {
    print("Table does not exist!");
    publish("error", "Table does not exist!");

    return;
  }

  private uint32[3] data;
  data[0] = answer;
  data[1] = question_number;
  data[2] = survey_id;

  dbInsertRow(tbl, data);

  return;
}
```

2. survey_min.sc

```
/* Gets the minimum answer of a survey question. */

void main(public uint32 survey_id, public uint32
      question_number)
{
  public string db;
  db = "hdsgs";

  public bool exists;

  exists = dbExists(db);
  if (!exists)
  {
    print("Database does not exist!");
    publish("error", "Database does not exist!")
     ;

    return;
  }

  dbLoad(db);

  public string tbl;
  tbl = "survey_answers";

  exists = dbTableExists(tbl);
  if (!exists)
  {
    print("Table does not exist!");
    publish("error", "Table does not exist!");

    return;
  }

/*==========COMPUTE FOR THE "MINIMUM" OF A
   QUESTION IN A SURVEY==========*/

// Get the number of rows
public uint32 rows;
rows = dbRowCount (tbl);

// Extract data in the answer_column
private uint32[rows] answers;
answers = dbReadColumnPrivateUint32(tbl, "
     answer_column");

// Extract data in the survey_id_column
private uint32[rows] survey_ids;
survey_ids = dbReadColumnPrivateUint32(tbl, "
     survey_id_column");

// Extract data in the question_number_column
private uint32[rows] question_nums;
question_nums = dbReadColumnPrivateUint32(tbl,
      "question_number_column");

// Store the given survey_id as a vector, for
   comparison
private uint32[rows] survey_id_vector;
survey_id_vector = survey_id;

// Compare which rows have survey_id_column is
    equal to given survey_id
private bool[rows] belongs_to_given_survey_id;
belongs_to_given_survey_id = (survey_ids ==
     survey_id_vector);

// Create comparison vector for question
   number
private uint32[rows] curr_question_num;
curr_question_num = question_number;

// Create boolean vector of whether the row is
    for the given question number...
private bool[rows] is_question_i;
is_question_i = (question_nums ==
   curr_question_num);
// ...of the given survey
is_question_i = (is_question_i *
   belongs_to_given_survey_id);

// Extract answers belonging to given question
    number
private uint32 x;
x = vecSum(is_question_i);
public uint32 num_of_answers_to_question;
num_of_answers_to_question = declassify(x);
private uint32[num_of_answers_to_question]
   answers_to_question;

public uint32 i = 0;
public uint32 ind = 0;
public uint32 q = 0;
public uint32 s = 0;
while(i < rows)
{
  q = declassify(question_nums[i]);
  s = declassify(survey_ids[i]);
  if(q == question_number && s == survey_id)
  {
    answers_to_question[ind] = answers[i];
    ind = ind + 1;
  }
```

```
    i = i + 1;
  }

  // Get min answer
  private uint32 min_answer = 0;
  private bool[num_of_answers_to_question] comp;
  private uint32[num_of_answers_to_question]
     curr;
  private uint32 lessers;
  public uint32 lessers_publicized;
  public uint32 j;
  for (j = 0; j < num_of_answers_to_question; j
     = j + 1)
  {
    curr = answers_to_question[j];
    comp = (answers_to_question < curr);
    lessers = vecSum(comp);

    lessers_publicized = declassify(lessers);
    if (lessers_publicized == 0) // found min
    {
      min_answer = answers_to_question[j];
      break;
    }
  }

  // Print and publish min answer
  public uint32 min_publicized;
  min_publicized = declassify(min_answer);
  publish("min", min_publicized);
  public string message;
  message = "min" + " = " + min_publicized;
  print(message);

  return;
}
```

3. survey_max.sc

```
/* Gets the maximum answer of a survey question. */

void main(public uint32 survey_id, public uint32
     question_number)
{
  public string db;
  db = "hdsgs";

  public bool exists;

  exists = dbExists(db);
  if (!exists)
  {
    print("Database does not exist!");
    publish("error", "Database does not exist!")
    ;

    return;
  }

  dbLoad(db);

  public string tbl;
  tbl = "survey_answers";

  exists = dbTableExists(tbl);
  if (!exists)
  {
    print("Table does not exist!");
    publish("error", "Table does not exist!");

    return;
  }

  /*==========COMPUTE FOR THE "MAXIMUM" OF A
     QUESTION IN A SURVEY==========*/

  // Get the number of rows
  public uint32 rows;
  rows = dbRowCount (tbl);

  // Extract data in the answer_column
  private uint32[rows] answers;
  answers = dbReadColumnPrivateUint32(tbl, "
       answer_column");

  // Extract data in the survey_id_column
  private uint32[rows] survey_ids;
  survey_ids = dbReadColumnPrivateUint32(tbl, "
       survey_id_column");

  // Extract data in the question_number_column
  private uint32[rows] question_nums;
  question_nums = dbReadColumnPrivateUint32(tbl,
       "question_number_column");

  // Store the given survey_id as a vector, for
     comparison
  private uint32[rows] survey_id_vector;
  survey_id_vector = survey_id;

  // Compare which rows have survey_id_column is
      equal to given survey_id
  private bool[rows] belongs_to_given_survey_id;
  belongs_to_given_survey_id = (survey_ids ==
       survey_id_vector);

  // Create comparison vector for question
     number
  private uint32[rows] curr_question_num;
  curr_question_num = question_number; // needed
      to be a vector for matrix operations

  // Create boolean vector of whether the row is
      for the given question number...
  private bool[rows] is_question_i;
  is_question_i = (question_nums ==
     curr_question_num);
  // ...of the given survey
  is_question_i = (is_question_i *
     belongs_to_given_survey_id);

  // Convert boolean vector to int for future
     arithmetic operations
  private uint32[rows] int_is_question_i;
  int_is_question_i = boolToInt(is_question_i);

  // Get answer vector for the given question
     number of the given survey
  private uint32[rows] question_i_answers;
  question_i_answers = (int_is_question_i *
     answers);

  // Get max aswer
  private uint32 max_answer = 0;
  private bool[rows] comp;
  private uint32[rows] curr;
  private uint32 greaters;
  public uint32 greaters_publicized;
  public uint32 j;
  for (j = 0; j < rows; j = j + 1)
  {
    curr = question_i_answers[j];
    comp = (question_i_answers > curr);
    greaters = vecSum(comp);

    greaters_publicized = declassify(greaters);
    if (greaters_publicized == 0) // found max
    {
      max_answer = question_i_answers[j];
      break;
    }
  }

  // Print and publish max answer
  public uint32 max_publicized;
  max_publicized = declassify(max_answer);
  publish("max", max_publicized);
  public string message;
  message = "max" + " = " + max_publicized;
  print(message);

  return;
}
```

4. survey_average.sc

```
/* Gets the average answer of a survey question. */

void main(public uint32 survey_id, public uint32
     question_number)
{
  public string db;
  db = "hdsgs";

  public bool exists;

  exists = dbExists(db);
  if (!exists)
  {
```

```
  print("Database does not exist!");
  publish("error", "Database does not exist!")
   ;

  return;
}

dbLoad(db);

public string tbl;
tbl = "survey_answers";

exists = dbTableExists(tbl);
if (!exists)
{
  print("Table does not exist!");
  publish("error", "Table does not exist!");

  return;
}

/*==========COMPUTE FOR THE "AVERAGE" OF A
   QUESTION IN A SURVEY==========*/

// Get the number of rows
public uint32 rows;
rows = dbRowCount (tbl);

// Extract data in the answer_column
private uint32[rows] answers;
answers = dbReadColumnPrivateUint32(tbl, "
     answer_column");

// Extract data in the survey_id_column
private uint32[rows] survey_ids;
survey_ids = dbReadColumnPrivateUint32(tbl, "
     survey_id_column");

// Extract data in the question_number_column
private uint32[rows] question_nums;
question_nums = dbReadColumnPrivateUint32(tbl,
      "question_number_column");

// Store the given survey_id as a vector, for
   comparison
private uint32[rows] survey_id_vector;
survey_id_vector = survey_id;

// Compare which rows have survey_id_column is
    equal to given survey_id
private bool[rows] belongs_to_given_survey_id;
belongs_to_given_survey_id = (survey_ids ==
     survey_id_vector);

// Create comparison vector for question
   number
private uint32[rows] curr_question_num;
curr_question_num = question_number;

// Create boolean vector of whether the row is
    for the given question number...
private bool[rows] is_question_i;
is_question_i = (question_nums ==
   curr_question_num);
// ...of the given survey
is_question_i = (is_question_i *
   belongs_to_given_survey_id);

// Get number of contributors for question
private uint32 contributors;
contributors = vecSum(is_question_i);

// Print and publish number of contributors
public uint32 contributors_publicized;
contributors_publicized = declassify(
   contributors);
publish("contributors",
   contributors_publicized);
public string message;
message = "contributors" + " = " +
   contributors_publicized;
print(message);

// Convert boolean vector to int for future
   arithmetic operations
private uint32[rows] int_is_question_i;
int_is_question_i = boolToInt(is_question_i);

// Get answer vector for the given question
   number of the given survey
private uint32[rows] question_i_answers;
question_i_answers = (int_is_question_i *
    answers);

// Get sum of answers
private uint32 sum_to_question;
sum_to_question = vecSum(question_i_answers);

// Print and publish sum of answers
public uint32 sum_publicized;
sum_publicized = declassify(sum_to_question);
publish("sum", sum_publicized);
message = "sum" + " = " + sum_publicized;
print(message);

  return;
}
```

5. survey_tally.sc

```
/* Gets the number of times a certain answer to a survey question
was chosen. */

void main(public uint32 survey_id, public uint32
     question_number, public uint32 choice)
{
  public string db;
  db = "hdsgs";

  public bool exists;

  exists = dbExists(db);
  if (!exists)
  {
    print("Database does not exist!");
    publish("error", "Database does not exist!")
     ;

    return;
  }

  dbLoad(db);

  public string tbl;
  tbl = "survey_answers";

  exists = dbTableExists(tbl);
  if (!exists)
  {
    print("Table does not exist!");
    publish("error", "Table does not exist!");

    return;
  }

  /*==========COMPUTE FOR THE "OCCURENCES OF A
     CHOICE" OF A QUESTION IN A SURVEY
     ==========*/

  // Get the number of rows
  public uint32 rows;
  rows = dbRowCount (tbl);

  // Extract data in the answer_column
  private uint32[rows] answers;
  answers = dbReadColumnPrivateUint32(tbl, "
       answer_column");

  // Extract data in the survey_id_column
  private uint32[rows] survey_ids;
  survey_ids = dbReadColumnPrivateUint32(tbl, "
       survey_id_column");

  // Extract data in the question_number_column
  private uint32[rows] question_nums;
  question_nums = dbReadColumnPrivateUint32(tbl,
       "question_number_column");

  // Store the given survey_id as a vector, for
     comparison
  private uint32[rows] survey_id_vector;
  survey_id_vector = survey_id;

  // Compare which rows have survey_id_column is
      equal to given survey_id
  private bool[rows] belongs_to_given_survey_id;
  belongs_to_given_survey_id = (survey_ids ==
       survey_id_vector);

  // Create comparison vector for question
     number
  private uint32[rows] curr_question_num;
```

```
curr_question_num = question_number;

// Create boolean vector of whether the row is
    for the given question number...
private bool[rows] is_question_i;
is_question_i = (question_nums ==
   curr_question_num);
// ...of the given survey
is_question_i = (is_question_i *
   belongs_to_given_survey_id);

// Convert boolean vector to int for future
   arithmetic operations
private uint32[rows] int_is_question_i;
int_is_question_i = boolToInt(is_question_i);

// Get answer vector for the given question
   number of the given survey
private uint32[rows] question_i_answers;
question_i_answers = (int_is_question_i *
   answers);

// Create choice vector
private uint32[rows] choice_vector;
choice_vector = choice;

// Determine if answer is equal to given
   choice
private bool[rows] is_choice;
is_choice = (question_i_answers ==
   choice_vector);

// Get number of occurences of given choice
private uint32 tally;
tally = vecSum(is_choice);

// Print and publish sum of answers
public uint32 tally_publicized;
tally_publicized = declassify(tally);
publish("tally", tally_publicized);
public string message;
message = "tally" + " = " + tally_publicized;
print(message);

return;
}
```

2. Sharemind Controllers

1. Survey_DatabaseEntry.cpp

```
/* Controller for inserting survey answers into the Sharemind
database. */

#include "controller/ControllerLibrary.h"
#include <string>
#include <iostream>
#include <sstream>

using namespace std;

const string scriptName = "survey_dataEntry.sa";

ControllerInterface* StartConnection()
{
  ControllerInterface *controller = new
     ControllerInterface("Survey_DatabaseEntry.
     log");

  // Give miners' addresses
  controller->loadConfiguration ("controller.cfg
     ");

  // Wait until miners are OK
  if (!controller->connect ("testuser"))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
     "Failed to connect to the miners!")
    delete controller;
    controller = NULL;
  }

  return controller;
}

void EndConnection(ControllerInterface *
     controller)
{
  controller->shutdown();
  delete (controller);
}

bool SaveValue (const val_t answer, const val_t
     survey_id, const val_t question_number,
     ControllerInterface* const controller)
{
  WRITE_LOG_DEBUG (controller->getConsole(), "
     Running script '" << scriptName << "' on
     the miners.")

  vector<RegisterDescriptor> parameters;
  vector<RegisterDescriptor> results;

  RegisterDescriptor answer_param(PRIVATE_INT, "
     answer");
  answer_param.setValueAsInteger(answer);
  parameters.push_back(answer_param);

  RegisterDescriptor question_number_param(
     PRIVATE_INT, "question_number");
  question_number_param.setValueAsInteger(
     question_number);
  parameters.push_back(question_number_param);

  RegisterDescriptor survey_id_param(PRIVATE_INT
     , "survey_id");
  survey_id_param.setValueAsInteger(survey_id);
  parameters.push_back(survey_id_param);

  if (!controller->runScript (scriptName,
     parameters, results))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
     "Error: Failed to run the script!")
    return false;
  }

  return true;
}

int main(int argc, char* argv[])
{
  val_t answer;
  val_t survey_id;
  val_t question_number;

  // Connect to the miners
  ControllerInterface *controller =
     StartConnection();
  if (controller == NULL)
  {
    return EXIT_FAILURE;
  }

  stringstream(argv[1]) >> answer;
  stringstream(argv[2]) >> question_number;
  stringstream(argv[3]) >> survey_id;

  if (!SaveValue(answer, survey_id,
     question_number, controller))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
     "Error inserting values in the survey
     database. - FAILED!")

    EndConnection(controller);

    return EXIT_FAILURE;
  }
  else
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Success! Values " << answer << ", " <<
         question_number << ", " << survey_id
       << " inserted.")
  }

  EndConnection(controller);

  return EXIT_SUCCESS;
}
```

2. Survey_GetMinimum.cpp

```
/* Controller for getting the minimum answer of a survey question.
*/

#include "controller/ControllerLibrary.h"
#include <string>
#include <iostream>
#include <sstream>

using namespace std;

const string scriptName = "survey_min.sa";
string returnValueName = "min";

ControllerInterface* StartConnection()
{
  ControllerInterface *controller = new
      ControllerInterface("Survey_GetMinimum.
      log");

  // Give miners' addresses
  controller->loadConfiguration ("controller.cfg
      ");

  // Wait until miners are OK
  if (!controller->connect ("testuser"))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Failed to connect to the miners!");
    delete controller;
    controller = NULL;
  }

  return controller;
}

void EndConnection(ControllerInterface *
    controller)
{
  controller->shutdown();
  delete (controller);
}

bool getAverage(const val_t survey_id, const
    val_t question_number, uint32 &min,
    ControllerInterface *controller)
{
  WRITE_LOG_DEBUG (controller->getConsole(), "
      Running script '" << scriptName << "' on
      the miners.")

  vector<RegisterDescriptor> parameters;
  vector<RegisterDescriptor> results;

  RegisterDescriptor survey_id_param(PUBLIC_INT,
      "survey_id");
  survey_id_param.setValueAsInteger(survey_id);
  parameters.push_back(survey_id_param);

  RegisterDescriptor question_number_param(
      PUBLIC_INT, "question_number");
  question_number_param.setValueAsInteger(
      question_number);
  parameters.push_back(question_number_param);

  if (!controller->runScript (scriptName,
      parameters, results))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Error: Failed to run the script!");
    return false;
  }

  if (results.size() != 1 || results[0].getName
      () != returnValueName)
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Script returned wrong result!");
    return false;
  }

  if (!results[0].getValueAsInteger(min))
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Failed to read result returned
        by script!");
    return false;
  }

  return true;
}

int main(int argc, char* argv[])
{
  val_t survey_id;
  val_t question_number;
  uint32 min;

  // Connect to the miners
  ControllerInterface *controller =
      StartConnection();
  if (controller == NULL)
  {
    return EXIT_FAILURE;
  }

  stringstream(argv[1]) >> survey_id;
  stringstream(argv[2]) >> question_number;

  if (!getAverage(survey_id, question_number,
      min, controller))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
      "Error getting minimum for survey_id = " <<
       survey_id << " question_number = " <<
      question_number << ".");

    EndConnection(controller);

    return EXIT_FAILURE;
  }
  else
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Minimum = " << min);
  }

  EndConnection(controller);

  return EXIT_SUCCESS;
}
```

3. Survey_GetMaximum.cpp

```
/* Controller for getting the maximum answer of a survey ques-
tion. */

#include "controller/ControllerLibrary.h"
#include <string>
#include <iostream>
#include <sstream>

using namespace std;

const string scriptName = "survey_max.sa";
string returnValueName = "max";

ControllerInterface* StartConnection()
{
  ControllerInterface *controller = new
      ControllerInterface("Survey_GetMaximum.
      log");

  // Give miners' addresses
  controller->loadConfiguration ("controller.cfg
      ");

  // Wait until miners are OK
  if (!controller->connect ("testuser"))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Failed to connect to the miners!");
    delete controller;
    controller = NULL;
  }

  return controller;
}

void EndConnection(ControllerInterface *
    controller)
{
  controller->shutdown();
  delete (controller);
}

bool getAverage(const val_t survey_id, const
    val_t question_number, uint32 &max,
    ControllerInterface *controller)
{
```

```
  WRITE_LOG_DEBUG (controller->getConsole(), "
      Running script '" << scriptName << "' on
      the miners.")

  vector<RegisterDescriptor> parameters;
  vector<RegisterDescriptor> results;

  RegisterDescriptor survey_id_param(PUBLIC_INT,
      "survey_id");
  survey_id_param.setValueAsInteger(survey_id);
  parameters.push_back(survey_id_param);

  RegisterDescriptor question_number_param(
      PUBLIC_INT, "question_number");
  question_number_param.setValueAsInteger(
      question_number);
  parameters.push_back(question_number_param);

  if (!controller->runScript (scriptName,
      parameters, results))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Error: Failed to run the script!");
    return false;
  }

  if (results.size() != 1 || results[0].getName
      () != returnValueName)
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Script returned wrong result!");
    return false;
  }

  if (!results[0].getValueAsInteger(max))
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Failed to read result returned
        by script!");
    return false;
  }

  return true;
}

int main(int argc, char* argv[])
{
  val_t survey_id;
  val_t question_number;
  uint32 max;

  // Connect to the miners
  ControllerInterface *controller =
      StartConnection();
  if (controller == NULL)
  {
    return EXIT_FAILURE;
  }

  stringstream(argv[1]) >> survey_id;
  stringstream(argv[2]) >> question_number;

  if (!getAverage(survey_id, question_number,
      max, controller))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
     "Error getting maximum for survey_id = " <<
      survey_id << " question_number = " <<
     question_number << ".");

    EndConnection(controller);

    return EXIT_FAILURE;
  }
  else
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Maximum = " << max);
  }

  EndConnection(controller);

  return EXIT_SUCCESS;
}
```

4. Survey_GetAverage.cpp

```
/* Controller for getting the average answer of a survey question.
*/
#include "controller/ControllerLibrary.h"
#include <string>
#include <iostream>
#include <sstream>

using namespace std;

const string scriptName = "survey_average.sa";
string returnValueName1 = "contributors";
string returnValueName2 = "sum";

ControllerInterface* StartConnection()
{
  ControllerInterface *controller = new
      ControllerInterface("Survey_GetAverage.
      log");

  // Give miners' addresses
  controller->loadConfiguration ("controller.cfg
      ");

  // Wait until miners are OK
  if (!controller->connect ("testuser"))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Failed to connect to the miners!");
    delete controller;
    controller = NULL;
  }

  return controller;
}

void EndConnection(ControllerInterface *
    controller)
{
  controller->shutdown();
  delete (controller);
}

bool getAverage(const val_t survey_id, const
    val_t question_number, uint32 &contributors
    , uint32 &sum, ControllerInterface *
    controller)
{
  WRITE_LOG_DEBUG (controller->getConsole(), "
      Running script '" << scriptName << "' on
      the miners.")

  vector<RegisterDescriptor> parameters;
  vector<RegisterDescriptor> results;

  RegisterDescriptor survey_id_param(PUBLIC_INT,
      "survey_id");
  survey_id_param.setValueAsInteger(survey_id);
  parameters.push_back(survey_id_param);

  RegisterDescriptor question_number_param(
      PUBLIC_INT, "question_number");
  question_number_param.setValueAsInteger(
      question_number);
  parameters.push_back(question_number_param);

  if (!controller->runScript (scriptName,
      parameters, results))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Error: Failed to run the script!");
    return false;
  }

  if (results.size() != 2 || (results[0].getName
      () != returnValueName1 && results[1].
      getName() != returnValueName2 ) )
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Script returned wrong results!")
        ;
    return false;
  }

  if (!(results[0].getValueAsInteger(
      contributors) && results[1].
      getValueAsInteger(sum)))
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Failed to read results returned
        by script!");
    return false;
  }

  return true;
```

```
}

int main(int argc, char* argv[])
{
  val_t survey_id;
  val_t question_number;
  uint32 contributors;
  uint32 sum;
  double average = 0.0;

  // Connect to the miners
  ControllerInterface *controller =
      StartConnection();
  if (controller == NULL)
  {
    return EXIT_FAILURE;
  }

  stringstream(argv[1]) >> survey_id;
  stringstream(argv[2]) >> question_number;

  if (!getAverage(survey_id, question_number,
      contributors, sum, controller))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
     "Error getting average for survey_id = " <<
      survey_id << " question_number = " <<
     question_number << ".");

    EndConnection(controller);

    return EXIT_FAILURE;
  }
  else
  {
    if(contributors == 0)
    {
      average = 0;
    }
    else
    {
      average = (double) sum / contributors;
    }
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Average = " << average);
  }

  EndConnection(controller);

  return EXIT_SUCCESS;
}
```

5. Survey_GetTally.cpp

```
/* Controller for getting the number of occurrences of a certain
answer to a survey question. */

#include "controller/ControllerLibrary.h"
#include <string>
#include <iostream>
#include <sstream>

using namespace std;

const string scriptName = "survey_tally.sa";
string returnValueName = "tally";

ControllerInterface* StartConnection()
{
  ControllerInterface *controller = new
      ControllerInterface("Survey_GetTally.log"
      );

  // Give miners' addresses
  controller->loadConfiguration ("controller.cfg
      ");

  // Wait until miners are OK
  if (!controller->connect ("testuser"))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Failed to connect to the miners!");
    delete controller;
    controller = NULL;
  }

  return controller;
}

void EndConnection(ControllerInterface *
    controller)
{
  controller->shutdown();
  delete (controller);
}

bool getTally(const val_t survey_id, const val_t
     question_number, const val_t choice,
    uint32 &tally, ControllerInterface *
    controller)
{
  WRITE_LOG_DEBUG (controller->getConsole(), "
      Running script '" << scriptName << "' on
      the miners.")

  vector<RegisterDescriptor> parameters;
  vector<RegisterDescriptor> results;

  RegisterDescriptor survey_id_param(PUBLIC_INT,
      "survey_id");
  survey_id_param.setValueAsInteger(survey_id);
  parameters.push_back(survey_id_param);

  RegisterDescriptor question_number_param(
      PUBLIC_INT, "question_number");
  question_number_param.setValueAsInteger(
      question_number);
  parameters.push_back(question_number_param);

  RegisterDescriptor choice_param(PUBLIC_INT, "
      choice");
  choice_param.setValueAsInteger(choice);
  parameters.push_back(choice_param);

  if (!controller->runScript (scriptName,
      parameters, results))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Error: Failed to run the script!");
    return false;
  }

  if (results.size() != 1 || results[0].getName
      () != returnValueName)
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Script returned wrong result!");
    return false;
  }

  if (!results[0].getValueAsInteger(tally))
  {
    WRITE_LOG_NORMAL(controller->getConsole(), "
        Error: Failed to read result returned
        by script!");
    return false;
  }

  return true;
}

int main(int argc, char* argv[])
{
  val_t survey_id;
  val_t question_number;
  val_t choice;
  uint32 tally;

  // Connect to the miners
  ControllerInterface *controller =
      StartConnection();
  if (controller == NULL)
  {
    return EXIT_FAILURE;
  }

  stringstream(argv[1]) >> survey_id;
  stringstream(argv[2]) >> question_number;
  stringstream(argv[3]) >> choice;

  if (!getTally(survey_id, question_number,
      choice, tally, controller))
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
     "Error getting tally.");

    EndConnection(controller);

    return EXIT_FAILURE;
  }
  else
```

```
  {
    WRITE_LOG_NORMAL (controller->getConsole(),
        "Tally = " << tally);
  }
```

```
  EndConnection(controller);

  return EXIT_SUCCESS;
}
```

3. Web Application Components

1. HdsgsApplication.java

```
package com.hdsgs;

import org.springframework.boot.
    SpringApplication;
import org.springframework.boot.autoconfigure.
    SpringBootApplication;
import org.springframework.boot.builder.
    SpringApplicationBuilder;
import org.springframework.boot.context.web.
    SpringBootServletInitializer;
import org.springframework.scheduling.annotation
    .EnableScheduling;

/**
 * Main class of the application.
 */
@SpringBootApplication
@EnableScheduling
public class HdsgsApplication extends
    SpringBootServletInitializer
{
  /**
   * Makes the application executable.
   *
   * @param args
   */
  public static void main( String[] args )
  {
    SpringApplication.run( HdsgsApplication.
        class, args );
  }

  @Override
  protected SpringApplicationBuilder configure(
      SpringApplicationBuilder application )
  {
    return application.sources( HdsgsApplication
        .class );
  }
}
```

2. SecurityConfig.java

```
package com.hdsgs;

import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.boot.autoconfigure.
    security.SecurityProperties;
import org.springframework.context.annotation.
    Configuration;
import org.springframework.core.annotation.Order
    ;
import org.springframework.security.config.
    annotation.authentication.builders.
    AuthenticationManagerBuilder;
import org.springframework.security.config.
    annotation.method.configuration.
    EnableGlobalMethodSecurity;
import org.springframework.security.config.
    annotation.web.builders.HttpSecurity;
import org.springframework.security.config.
    annotation.web.builders.WebSecurity;
import org.springframework.security.config.
    annotation.web.configuration.
    WebSecurityConfigurerAdapter;
import org.springframework.security.core.
    userdetails.UserDetailsService;
import org.springframework.security.crypto.
    bcrypt.BCryptPasswordEncoder;

/**
 * Security configuration for the application.
 *
 */
@Configuration
@EnableGlobalMethodSecurity( prePostEnabled =
    true )
@Order( SecurityProperties.ACCESS_OVERRIDE_ORDER
    )
public class SecurityConfig extends
    WebSecurityConfigurerAdapter
{
  @Autowired
  private UserDetailsService userDetailsService;


  @Override
  protected void configure( HttpSecurity http )
      throws Exception
  {
    // @formatter:off

    http.authorizeRequests()
        .and()
      .formLogin()
        .loginPage( "/login" )
        .failureUrl( "/login?error" )
        .permitAll()
        .and()
      .logout()
        .logoutUrl( "/logout" )
        .deleteCookies( "remember-me" )
        .logoutSuccessUrl( "/" )
        .permitAll()
        .and()
      .rememberMe()
        .tokenValiditySeconds( 1800 ); // 30
            minutes

    // @formatter:on
  }


  @Override
  public void configure( WebSecurity web )
      throws Exception
  {
    web.ignoring().antMatchers( "/webjars/**" );
  }


  @Override
  public void configure(
      AuthenticationManagerBuilder auth )
      throws Exception
  {
    auth.userDetailsService( userDetailsService
        ).passwordEncoder( new
        BCryptPasswordEncoder() );
  }

}
```

3. User.java

```
package com.hdsgs.entity;

import static javax.persistence.EnumType.STRING;
import static javax.persistence.GenerationType.
    IDENTITY;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

import com.hdsgs.enums.UserRole;

/**
 * Represents a single user. Each user is
     classified as a
 * {@link UserRole#CONTRIBUTOR contributor}, a {
     @link UserRole#RESEARCH_LEADER
 * research leader}, or a {@link UserRole#ADMIN
     super administrator}. Only a
 * research leader can create zero or more {
     @link Survey surveys}. This class
 * was generated, and is used by Hibernate to
     persist user data.
 */
@Entity
@Table( name = "user", catalog = "hdsgs",
    uniqueConstraints = @UniqueConstraint(
    columnNames = "username" ) )
public class User implements java.io.
    Serializable
{
  private static final long serialVersionUID =
      -6914829311780412869L;

  private Integer id;

  private String username;

  private String password;

  private String name;

  private String designation;

  private String institution;

  private String email;

  private UserRole role;

  private boolean active;

  private List< Survey > surveys = new ArrayList
      < Survey >( 0 );


  /**
   * Constructs a new user with default values.
       Needed by Hibernate.
   */
  protected User()
  {

  }


  /**
   * Constructs a new user with the specified
       values.
   *
   * @param username
   * @param password
   * @param name
   * @param designation
   * @param institution
   * @param email
   * @param role
   * @param active
   */
  public User( String username, String password,
      String name, String designation, String
      institution, String email, UserRole role,
      boolean active )
  {
    this.username = username;
    this.password = password;
    this.name = name;
    this.designation = designation;
    this.institution = institution;
    this.email = email;
    this.role = role;
    this.active = active;
  }


  /**
   * Constructs a new user with the specified
       values.
   *
   * @param username
   * @param password
   * @param name
   * @param designation
   * @param institution
   * @param email
   * @param role
   * @param active
   * @param surveys
   */
  public User( String username, String password,
      String name, String designation, String
      institution, String email, UserRole role,
      boolean active,
      List< Survey > surveys )
  {
    this.username = username;
    this.password = password;
    this.name = name;
    this.designation = designation;
    this.institution = institution;
    this.email = email;
    this.role = role;
    this.active = active;
    this.surveys = surveys;
  }


  /**
   * Returns the id of this user.
   *
   * @return id
   */
  @Id
  @GeneratedValue( strategy = IDENTITY )
  @Column( name = "id", unique = true, nullable
      = false )
  public Integer getId()
  {
    return this.id;
  }


  /**
   * Sets the id of this user.
   *
   * @param id
   */
  public void setId( Integer id )
  {
    this.id = id;
  }


  /**
   * Returns the username of this user.
   *
   * @return username
   */
  @Column( name = "username", unique = true,
      nullable = false, length = 20 )
  public String getUsername()
  {
    return this.username;
  }


  /**
   * Sets the username of this user.
   *
   * @param username
   */
  public void setUsername( String username )
  {
    this.username = username;
  }


  /**
   * Returns the password of this user.
   *
   * @return password
   */
  @Column( name = "password", nullable = false,
      length = 255 )
  public String getPassword()
  {
    return this.password;
  }
```

```
/**
 * Sets the password of this user.
 *
 * @param password
 */
public void setPassword( String password )
{
  this.password = password;
}


/**
 * Returns the name of this user.
 *
 * @return name
 */
@Column( name = "name", nullable = false,
    length = 50 )
public String getName()
{
  return this.name;
}


/**
 * Sets the name of this user.
 *
 * @param name
 */
public void setName( String name )
{
  this.name = name;
}


/**
 * Returns the designation of this user.
 *
 * @return designation
 */
@Column( name = "designation", nullable =
    false, length = 50 )
public String getDesignation()
{
  return this.designation;
}


/**
 * Sets the designation of this user.
 *
 * @param designation
 */
public void setDesignation( String designation
    )
{
  this.designation = designation;
}


/**
 * Returns the name of the institution where
     this user belongs to.
 *
 * @return institution
 */
@Column( name = "institution", nullable =
    false, length = 50 )
public String getInstitution()
{
  return this.institution;
}


/**
 * Sets the name of the institution where this
     user belongs to.
 *
 * @param institution
 */
public void setInstitution( String institution
    )
{
  this.institution = institution;
}


/**
 * Returns the email address of this user.
 *
 * @return email address
 */
@Column( name = "email", nullable = false,
    length = 30 )
public String getEmail()
{
  return this.email;
}


/**
 * Sets the email address of this user.
 *
 * @param email
 */
public void setEmail( String email )
{
  this.email = email;
}


/**
 * Returns the role of this user.
 *
 * @return role
 */
@Column( name = "role", nullable = false,
    columnDefinition = "enum('CONTRIBUTOR', '
    RESEARCH_LEADER', 'ADMIN')" )
@Enumerated( STRING )
public UserRole getRole()
{
  return this.role;
}


/**
 * Sets the role of this user.
 *
 * @param role
 */
public void setRole( UserRole role )
{
  this.role = role;
}


/**
 * @return is this user active
 */
@Column( name = "active", nullable = false )
public boolean isActive()
{
  return this.active;
}


/**
 * Sets whether this user is active.
 *
 * @param active
 */
public void setActive( boolean active )
{
  this.active = active;
}


/**
 * Returns the list of surveys created by this
     user.
 *
 * @return surveys
 */
@OneToMany( fetch = FetchType.LAZY, mappedBy =
    "creator", cascade = CascadeType.ALL )
public List< Survey > getSurveys()
{
  return this.surveys;
}


/**
 * Sets the list of surveys created by this
     user.
 *
 * @param surveys
 */
public void setSurveys( List< Survey > surveys
    )
{
  this.surveys = surveys;
```

```
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( active ? 1231 :
       1237 );
  result = prime * result + ( ( designation ==
       null ) ? 0 : designation.hashCode() );
  result = prime * result + ( ( email == null
       ) ? 0 : email.hashCode() );
  result = prime * result + ( ( id == null ) ?
       0 : id.hashCode() );
  result = prime * result + ( ( institution ==
       null ) ? 0 : institution.hashCode() );
  result = prime * result + ( ( name == null )
       ? 0 : name.hashCode() );
  result = prime * result + ( ( password ==
       null ) ? 0 : password.hashCode() );
  result = prime * result + ( ( role == null )
       ? 0 : role.hashCode() );
  result = prime * result + ( ( surveys ==
       null ) ? 0 : surveys.hashCode() );
  result = prime * result + ( ( username ==
       null ) ? 0 : username.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  User other = ( User ) obj;

  if( active != other.active )
  {
    return false;
  }

  if( designation == null )
  {
    if( other.designation != null )
    {
      return false;
    }
  }
  else if( !designation.equals( other.
       designation ) )
  {
    return false;
  }

  if( email == null )
  {
    if( other.email != null )
    {
      return false;
    }
  }
  else if( !email.equals( other.email ) )
  {
    return false;
  }

  if( id == null )
  {
    if( other.id != null )
    {
      return false;
    }
  }
  else if( !id.equals( other.id ) )
  {
    return false;
  }

  if( institution == null )
  {
    if( other.institution != null )
    {
      return false;
    }
  }
  else if( !institution.equals( other.
       institution ) )
  {
    return false;
  }

  if( name == null )
  {
    if( other.name != null )
    {
      return false;
    }
  }
  else if( !name.equals( other.name ) )
  {
    return false;
  }

  if( password == null )
  {
    if( other.password != null )
    {
      return false;
    }
  }
  else if( !password.equals( other.password )
       )
  {
    return false;
  }

  if( role != other.role )
  {
    return false;
  }

  if( surveys == null )
  {
    if( other.surveys != null )
    {
      return false;
    }
  }
  else if( !surveys.equals( other.surveys ) )
  {
    return false;
  }

  if( username == null )
  {
    if( other.username != null )
    {
      return false;
    }
  }
  else if( !username.equals( other.username )
       )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "User [id=" + id + ", username=" +
       username + ", password=" + password + "
       , name=" + name + ", designation=" +
       designation
      + ", institution=" + institution + ",
           email=" + email + ", role=" + role
           + ", isActive=" + active + ",
           surveys=" + surveys + "]";
}
}
```

4. CurrentUser.java

```java
package com.hdsgs.entity;

import org.springframework.security.core.
    authority.AuthorityUtils;

import com.hdsgs.enums.UserRole;

/**
 * Represents the currently logged in user.
 */
public class CurrentUser extends org.
    springframework.security.core.userdetails.
    User
{
  private static final long serialVersionUID =
      -8492858710041037624L;

  private User user;


  /**
   * Wraps the currently logged in user entity
       with this object.
   *
   * @param user
   */
  public CurrentUser( User user )
  {
    super( user.getUsername(), user.getPassword
        (), AuthorityUtils.createAuthorityList(
         user.getRole().toString() ) );
    this.user = user;
  }


  /**
   * Returns the wrapped user entity.
   *
   * @return user
   */
  public User getUser()
  {
    return user;
  }


  /**
   * Returns the id of the currently logged in
       user.
   *
   * @return id
   */
  public Integer getId()
  {
    return user.getId();
  }


  /**
   * Returns the name of the currently logged in
       user.
   *
   * @return name
   */
  public String getName()
  {
    return user.getName();
  }


  /**
   * Returns the {@link UserRole role} of the
       currently logged in user.
   *
   * @return role
   */
  public UserRole getRole()
  {
    return user.getRole();
  }


  /**
   * @return {@code true} if current user is
       active, {@code false} otherwise
   */
  public boolean isActive()
  {
    return user.isActive();
  }
}
```

5. Survey.java

```java
package com.hdsgs.entity;

import static javax.persistence.GenerationType.
    IDENTITY;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

import org.hibernate.annotations.Type;
import org.joda.time.LocalDate;

import com.hdsgs.enums.UserRole;

/**
 * Represents a single survey. Each survey is
      created by a {@link UserRole#
      RESEARCH_LEADER research leader}, and has
      one
 * or more {@link Question questions}. This
      class was generated, and is used by
      Hibernate to persist survey data.
 */
@Entity
@Table( name = "survey", catalog = "hdsgs" )
public class Survey implements java.io.
    Serializable
{
  private static final long serialVersionUID =
      -6557921851689593O7L;

  private Integer id;

  private User creator;

  private String title;

  private LocalDate startDate;

  private LocalDate endDate;

  private String description;

  private List< Question > questions = new
      ArrayList< Question >( 0 );

  private List< BasicStats > basicStats = new
      ArrayList< BasicStats >( 0 );

  private List< Tally > tally = new ArrayList<
      Tally >( 0 );


  /**
   * Constructs a new survey with default values
       . Needed by Hibernate.
   */
  protected Survey()
  {

  }


  /**
   * Constructs a new survey with the specified
       values.
   *
   * @param user
   * @param title
   * @param startDate
   * @param endDate
   * @param description
   */
  public Survey( User user, String title,
      LocalDate startDate, LocalDate endDate,
      String description )
```

```
{
  this.creator = user;
  this.title = title;
  this.startDate = startDate;
  this.endDate = endDate;
  this.description = description;
}


/**
 * Constructs a new survey with the specified
     values.
 *
 * @param user
 * @param title
 * @param startDate
 * @param endDate
 * @param description
 * @param questions
 */
public Survey( User user, String title,
    LocalDate startDate, LocalDate endDate,
    String description, List< Question >
    questions )
{
  this.creator = user;
  this.title = title;
  this.startDate = startDate;
  this.endDate = endDate;
  this.questions = questions;
  this.description = description;
}


/**
 * Return the id of this survey.
 *
 * @return id
 */
@Id
@GeneratedValue( strategy = IDENTITY )
@Column( name = "id", unique = true, nullable
    = false )
public Integer getId()
{
  return this.id;
}


/**
 * Sets the id of this survey.
 *
 * @param id
 */
public void setId( Integer id )
{
  this.id = id;
}


/**
 * Returns the creator of this survey.
 *
 * @return creator
 */
@ManyToOne( fetch = FetchType.EAGER )
@JoinColumn( name = "created_by", nullable =
    false )
public User getCreator()
{
  return this.creator;
}


/**
 * Sets the creator of this survey
 *
 * @param creator
 */
public void setCreator( User creator )
{
  this.creator = creator;
}


/**
 * Returns the title of this survey.
 *
 * @return title
 */
@Column( name = "title", nullable = false,
    length = 50 )
public String getTitle()
{
  return this.title;
}


/**
 * Sets the title of this survey.
 *
 * @param title
 */
public void setTitle( String title )
{
  this.title = title;
}


/**
 * Returns the date when this survey will
     start.
 *
 * @return start date
 */
@Temporal( TemporalType.TIMESTAMP )
@Column( name = "start", nullable = false,
    length = 19 )
@Type( type = "org.jadira.usertype.dateandtime
    .joda.PersistentLocalDate" )
public LocalDate getStartDate()
{
  return this.startDate;
}


/**
 * Sets the date when this survey will start.
 *
 * @param startDate
 */
public void setStartDate( LocalDate startDate
    )
{
  this.startDate = startDate;
}


/**
 * Returns the date when this survey will end.
 *
 * @return end date
 */
@Temporal( TemporalType.TIMESTAMP )
@Column( name = "end", nullable = false,
    length = 19 )
@Type( type = "org.jadira.usertype.dateandtime
    .joda.PersistentLocalDate" )
public LocalDate getEndDate()
{
  return this.endDate;
}


/**
 * Sets the date when this survey will end.
 *
 * @param endDate
 */
public void setEndDate( LocalDate endDate )
{
  this.endDate = endDate;
}


/**
 * Returns the description of this survey.
 *
 * @return description
 */
@Column( name = "description", nullable =
    false, length = 255 )
public String getDescription()
{
  return this.description;
}


/**
 * Sets the description of this survey.
 *
 * @param description
 */
public void setDescription( String description
```

```
    )
{
  this.description = description;
}


/**
 * Returns the list of questions of this
     survey.
 *
 * @return questions
 */
@OneToMany( fetch = FetchType.LAZY, mappedBy =
     "survey", cascade = CascadeType.ALL,
    orphanRemoval = true )
public List< Question > getQuestions()
{
  return this.questions;
}


/**
 * Sets the list of questions of this survey.
     Needed by Hibernate but not used in this
     application.
 *
 * @param questions
 */
protected void setQuestions( List< Question >
    questions )
{
  this.questions = questions;
}


/**
 * Sets the list of questions of this survey.
     Replaced {@link Survey#setQuestions}.
 *
 * @param questions
 */
public void populateQuestions( List< Question
    > questions )
{
  this.questions.clear();
  this.questions.addAll( questions );
}


@OneToMany( fetch = FetchType.LAZY, mappedBy =
     "survey", cascade = CascadeType.ALL,
    orphanRemoval = true )
public List< BasicStats > getBasicStats()
{
  return basicStats;
}


protected void setBasicStats( List< BasicStats
    > basicStats )
{
  this.basicStats = basicStats;
}


public void populateBasicStats( List<
    BasicStats > basicStats )
{
  this.basicStats.clear();
  this.basicStats.addAll( basicStats );
}


@OneToMany( fetch = FetchType.LAZY, mappedBy =
     "survey", cascade = CascadeType.ALL,
    orphanRemoval = true )
public List< Tally > getTally()
{
  return tally;
}


protected void setTally( List< Tally > tally )
{
  this.tally = tally;
}


public void populateTally( List< Tally > tally
    )
{
  this.tally.clear();
  this.tally.addAll( tally );
}


/**
 * @return is today's date between this survey
     's start and end dates (inclusive)
 */
public boolean active()
{
  LocalDate now = LocalDate.now();

  return ( now.isEqual( this.startDate ) || (
      now.isAfter( this.startDate ) && now.
      isBefore( this.endDate ) ) ) && !now.
      equals( endDate );
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( ( creator ==
      null ) ? 0 : creator.hashCode() );
  result = prime * result + ( ( description ==
       null ) ? 0 : description.hashCode() );
  result = prime * result + ( ( endDate ==
      null ) ? 0 : endDate.hashCode() );
  result = prime * result + ( ( id == null ) ?
       0 : id.hashCode() );
  result = prime * result + ( ( questions ==
      null ) ? 0 : questions.hashCode() );
  result = prime * result + ( ( startDate ==
      null ) ? 0 : startDate.hashCode() );
  result = prime * result + ( ( title == null
      ) ? 0 : title.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  Survey other = ( Survey ) obj;

  if( creator == null )
  {
    if( other.creator != null )
    {
      return false;
    }
  }
  else if( !creator.equals( other.creator ) )
  {
    return false;
  }

  if( description == null )
  {
    if( other.description != null )
    {
      return false;
    }
  }
  else if( !description.equals( other.
      description ) )
  {
    return false;
  }

  if( endDate == null )
  {
    if( other.endDate != null )
```

```
      {
        return false;
      }
    }
    else if( !endDate.equals( other.endDate ) )
    {
      return false;
    }

    if( id == null )
    {
      if( other.id != null )
      {
        return false;
      }
    }
    else if( !id.equals( other.id ) )
    {
      return false;
    }

    if( questions == null )
    {
      if( other.questions != null )
      {
        return false;
      }
    }
    else if( !questions.equals( other.questions
        ) )
    {
      return false;
    }

    if( startDate == null )
    {
      if( other.startDate != null )
      {
        return false;
      }
    }
    else if( !startDate.equals( other.startDate
        ) )
    {
      return false;
    }

    if( title == null )
    {
      if( other.title != null )
      {
        return false;
      }
    }
    else if( !title.equals( other.title ) )
    {
      return false;
    }

    return true;
  }


  @Override
  public String toString()
  {
    return "Survey [id=" + id + ", creator=" +
        creator + ", title=" + title + ",
        startDate=" + startDate + ", endDate="
        + endDate
        + ", description=" + description + ",
            questions=" + questions + "]";
  }
}
```

6. Question.java

```
package com.hdsgs.entity;

import static javax.persistence.EnumType.STRING;
import static javax.persistence.GenerationType.
    IDENTITY;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import com.hdsgs.enums.InputType;

/**
 * Represents a single survey question. Each
     question belongs to a
 * {@link Survey}. A question can have zero or
     more {@link QuestionChoice
 * choices}, and one or more {@link
     QuestionFunction result functions}. This
 * class was generated, and is primarily used by
     Hibernate to persist survey
 * question data.
 */
@Entity
@Table( name = "question", catalog = "hdsgs" )
public class Question implements java.io.
    Serializable
{
  private static final long serialVersionUID =
      6686143293164549O6L;

  private Integer id;

  private Survey survey;

  private int questionNumber;

  private String question;

  private InputType inputType;

  private List< QuestionChoice > choices = new
      ArrayList< QuestionChoice >( 0 );

  private List< QuestionFunction > functions =
      new ArrayList< QuestionFunction >( 0 );


  /**
   * Constructs a new question with default
       values. Needed by Hibernate.
   */
  protected Question()
  {

  }


  /**
   * Constructs a new question with the
       specified values.
   *
   * @param survey
   * @param questionNumber
   * @param question
   * @param inputType
   */
  public Question( Survey survey, int
      questionNumber, String question,
      InputType inputType )
  {
    this.survey = survey;
    this.questionNumber = questionNumber;
    this.question = question;
    this.inputType = inputType;
  }


  /**
   * Constructs a new question with the
       specified values.
   *
   * @param survey
   * @param questionNumber
   * @param question
   * @param inputType
   * @param choices
   * @param functions
   */
  public Question( Survey survey, int
      questionNumber, String question,
      InputType inputType, List< QuestionChoice
      > choices,
```

```
    List< QuestionFunction > functions )
{
  this.survey = survey;
  this.questionNumber = questionNumber;
  this.question = question;
  this.inputType = inputType;
  this.choices = choices;
  this.functions = functions;
}


/**
 * Returns the id of this question.
 *
 * @return id
 */
@Id
@GeneratedValue( strategy = IDENTITY )
@Column( name = "id", unique = true, nullable
    = false )
public Integer getId()
{
  return this.id;
}


/**
 * Sets the id of this question.
 *
 * @param id
 */
public void setId( Integer id )
{
  this.id = id;
}


/**
 * Returns the survey to which this question
     belongs.
 *
 * @return survey
 */
@ManyToOne( fetch = FetchType.EAGER )
@JoinColumn( name = "survey_id", nullable =
    false )
public Survey getSurvey()
{
  return this.survey;
}


/**
 * Sets the survey to which this question
     belongs.
 *
 * @param survey
 */
public void setSurvey( Survey survey )
{
  this.survey = survey;
}


/**
 * Returns the question number of this
     question.
 *
 * @return question number
 */
@Column( name = "question_number", nullable =
    false )
public int getQuestionNumber()
{
  return this.questionNumber;
}


/**
 * Sets the question number of this question.
 *
 * @param questionNumber
 */
public void setQuestionNumber( int
    questionNumber )
{
  this.questionNumber = questionNumber;
}


/**
 * Returns the question string.
 *
 * @return question string
 */
@Column( name = "question", nullable = false )
public String getQuestion()
{
  return this.question;
}


/**
 * Sets the question string.
 *
 * @param question
 */
public void setQuestion( String question )
{
  this.question = question;
}


/**
 * Returns the input type of this question.
 *
 * @return input type
 */
@Column( name = "input_type", nullable = false
    , columnDefinition = "enum('NUMBER', '
    RADIO', 'CHECKBOX', 'DROPDOWN')" )
@Enumerated( STRING )
public InputType getInputType()
{
  return this.inputType;
}


/**
 * Sets the input type of this question.
 *
 * @param inputType
 */
public void setInputType( InputType inputType
    )
{
  this.inputType = inputType;
}


/**
 * Returns the list of choices for this
     question.
 *
 * @return choices
 */
@OneToMany( fetch = FetchType.LAZY, mappedBy =
    "question", cascade = CascadeType.ALL,
    orphanRemoval = true )
public List< QuestionChoice > getChoices()
{
  return this.choices;
}


/**
 * Sets the list of choices for this question.
     Needed by Hibernate but not
 * used in this application.
 *
 * @param choices
 */
protected void setChoices( List<
    QuestionChoice > choices )
{
  this.choices = choices;
}


/**
 * Sets the list of choices for this question.
     Replaced
 * {@link Question#setChoices}.
 *
 * @param choices
 */
public void populateChoices( List<
    QuestionChoice > choices )
{
  this.choices.clear();
  this.choices.addAll( choices );
}
```

```
/**
 * Returns the list of result functions to
      compute for this question.
 *
 * @return functions
 */
@OneToMany( fetch = FetchType.LAZY, mappedBy =
      "question", cascade = CascadeType.ALL,
     orphanRemoval = true )
public List< QuestionFunction > getFunctions()
{
  return this.functions;
}


/**
 * Sets the list of functions for this
      question. Needed by Hibernate but not
 * used by this application.
 *
 * @param functions
 */
protected void setFunctions( List<
     QuestionFunction > functions )
{
  this.functions = functions;
}


/**
 * Sets the list of functions for this
      question. Replaced
 * {@link Question#setFunctions}.
 *
 * @param functions
 */
public void populateFunctions( List<
     QuestionFunction > functions )
{
  this.functions.clear();
  this.functions.addAll( functions );
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( ( choices ==
       null ) ? 0 : choices.hashCode() );
  result = prime * result + ( ( functions ==
       null ) ? 0 : functions.hashCode() );
  result = prime * result + ( ( id == null ) ?
        0 : id.hashCode() );
  result = prime * result + ( ( inputType ==
       null ) ? 0 : inputType.hashCode() );
  result = prime * result + ( ( question ==
       null ) ? 0 : question.hashCode() );
  result = prime * result + questionNumber;
  result = prime * result + ( ( survey == null
        ) ? 0 : survey.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  Question other = ( Question ) obj;

  if( choices == null )
  {
    if( other.choices != null )
    {
      return false;
    }
  }
  else if( !choices.equals( other.choices ) )
  {
    return false;
  }

  if( functions == null )
  {
    if( other.functions != null )
    {
      return false;
    }
  }
  else if( !functions.equals( other.functions
       ) )
  {
    return false;
  }

  if( id == null )
  {
    if( other.id != null )
    {
      return false;
    }
  }
  else if( !id.equals( other.id ) )
  {
    return false;
  }

  if( inputType != other.inputType )
  {
    return false;
  }

  if( question == null )
  {
    if( other.question != null )
    {
      return false;
    }
  }
  else if( !question.equals( other.question )
       )
  {
    return false;
  }

  if( questionNumber != other.questionNumber )
  {
    return false;
  }

  if( survey == null )
  {
    if( other.survey != null )
    {
      return false;
    }
  }
  else if( !survey.equals( other.survey ) )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "Question [id=" + id + ", survey.id="
       + survey.getId() + ", questionNumber="
       + questionNumber + ", question=" +
      question
      + ", inputType=" + inputType + ",
           choices=" + choices + ", functions=
           " + functions + "]";
}
}
```

7. QuestionChoice.java

```
package com.hdsgs.entity;
```

```
import static javax.persistence.GenerationType.
    IDENTITY;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

/**
 * Represents a single survey question option.
     Each option belongs to a
 * {@link Question question}. This class was
     generated, and is primarily used by
 * Hibernate to persist survey question options
     data.
 */
@Entity
@Table( name = "question_choice", catalog = "
    hdsgs" )
public class QuestionChoice implements java.io.
    Serializable
{
  private static final long serialVersionUID =
      4611404058494930552L;

  private Integer id;

  private Question question;

  private String option;


  /**
   * Constructs a new question choice object
       with default values. Needed by
   * Hibernate.
   */
  protected QuestionChoice()
  {

  }


  /**
   * Constructs a new question choice object
       with the specified values.
   *
   * @param question
   * @param optionValue
   */
  public QuestionChoice( Question question,
      String optionValue )
  {
    this.question = question;
    this.option = optionValue;
  }


  /**
   * Returns the id of this object.
   *
   * @return id
   */
  @Id
  @GeneratedValue( strategy = IDENTITY )
  @Column( name = "id", unique = true, nullable
      = false )
  public Integer getId()
  {
    return this.id;
  }


  /**
   * Sets the id of this object.
   *
   * @param id
   */
  public void setId( Integer id )
  {
    this.id = id;
  }


  /**
   * Returns the question associated with this
       object.
   *
   * @return question
   */
  @ManyToOne( fetch = FetchType.EAGER )
  @JoinColumn( name = "question_id", nullable =
      false )
  public Question getQuestion()
  {
    return this.question;
  }


  /**
   * Sets the question associated with this
       object.
   *
   * @param question
   */
  public void setQuestion( Question question )
  {
    this.question = question;
  }


  /**
   * Returns the option.
   *
   * @return option
   */
  @Column( name = "option_value", nullable =
      false, length = 50 )
  public String getOption()
  {
    return this.option;
  }


  /**
   * Sets the option.
   *
   * @param option
   */
  public void setOption( String option )
  {
    this.option = option;
  }


  @Override
  public String toString()
  {
    return "QuestionChoice [id=" + id + ",
        questionId=" + question.getId() + ",
        optionValue=" + option + "]";
  }


  @Override
  public int hashCode()
  {
    final int prime = 31;
    int result = 1;

    result = prime * result + ( ( id == null ) ?
        0 : id.hashCode() );
    result = prime * result + ( ( option == null
        ) ? 0 : option.hashCode() );
    result = prime * result + ( ( question ==
        null ) ? 0 : question.hashCode() );

    return result;
  }


  @Override
  public boolean equals( Object obj )
  {
    if( this == obj )
    {
      return true;
    }

    if( obj == null )
    {
      return false;
    }

    if( getClass() != obj.getClass() )
    {
      return false;
    }

    QuestionChoice other = ( QuestionChoice )
```

```
        obj;

    if( id == null )
    {
      if( other.id != null )
      {
        return false;
      }
    }
    else if( !id.equals( other.id ) )
    {
      return false;
    }

    if( option == null )
    {
      if( other.option != null )
      {
        return false;
      }
    }
    else if( !option.equals( other.option ) )
    {
      return false;
    }

    if( question == null )
    {
      if( other.question != null )
      {
        return false;
      }
    }
    else if( !question.equals( other.question )
        )
    {
      return false;
    }

    return true;
  }
}
```

8. QuestionFunction.java

```
package com.hdsgs.entity;

import static javax.persistence.EnumType.STRING;
import static javax.persistence.GenerationType.
    IDENTITY;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

import com.hdsgs.enums.Function;

/**
 * Represents a single survey question result
     function. Each function belongs to
 * a {@link Question question}. Permissible
     values are {@link Function#AVERAGE
 * average}, {@link Function#MINIMUM minimum}, {
     @link Function#MAXIMUM maximum},
 * and {@link Function#TALLY tally}. This class
     represents the statistics that
 * will be generated for this question. This
     class was generated, and is
 * primarily used by Hibernate to persist survey
     question options data.
 */
@Entity
@Table( name = "question_function", catalog = "
    hdsgs" )
public class QuestionFunction implements java.io
    .Serializable
{
  private static final long serialVersionUID =
      286450609474918542L;

  private Integer id;

  private Question question;

  private Function functionToCompute;


  /**
   * Constructs a new question function with
       default values. Needed by
   * Hibernate.
   */
  protected QuestionFunction()
  {

  }


  /**
   * Constructs a new question function with the
       specified values.
   *
   * @param question
   * @param functionToCompute
   */
  public QuestionFunction( Question question,
      Function functionToCompute )
  {
    this.question = question;
    this.functionToCompute = functionToCompute;
  }


  /**
   * Returns the id of this question function.
   *
   * @return id
   */
  @Id
  @GeneratedValue( strategy = IDENTITY )
  @Column( name = "id", unique = true, nullable
      = false )
  public Integer getId()
  {
    return this.id;
  }


  /**
   * Sets the id of this question function.
   *
   * @param id
   */
  public void setId( Integer id )
  {
    this.id = id;
  }


  /**
   * Returns the question associated with this
       question function.
   *
   * @return question
   */
  @ManyToOne( fetch = FetchType.EAGER )
  @JoinColumn( name = "question_id", nullable =
      false )
  public Question getQuestion()
  {
    return this.question;
  }


  /**
   * Sets the question associated with this
       survey.
   *
   * @param question
   */
  public void setQuestion( Question question )
  {
    this.question = question;
  }


  /**
   * Returns the {@link Function function} of
       this question function.
   *
   * @return function
   */
  @Column( name = "function_to_compute",
      nullable = false, columnDefinition = "
      enum('AVERAGE', 'MINIMUM', 'MAXIMUM', '
```

```
    TALLY')" )
@Enumerated( STRING )
public Function getFunctionToCompute()
{
  return this.functionToCompute;
}


/**
 * Sets the {@link Function function} of this
     question function.
 *
 * @param functionToCompute
 */
public void setFunctionToCompute( Function
    functionToCompute )
{
  this.functionToCompute = functionToCompute;
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( (
      functionToCompute == null ) ? 0 :
      functionToCompute.hashCode() );
  result = prime * result + ( ( id == null ) ?
       0 : id.hashCode() );
  result = prime * result + ( ( question ==
      null ) ? 0 : question.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  QuestionFunction other = ( QuestionFunction
      ) obj;

  if( functionToCompute != other.
      functionToCompute )
  {
    return false;
  }

  if( id == null )
  {
    if( other.id != null )
    {
      return false;
    }
  }
  else if( !id.equals( other.id ) )
  {
    return false;
  }

  if( question == null )
  {
    if( other.question != null )
    {
      return false;
    }
  }
  else if( !question.equals( other.question )
      )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "QuestionFunction [id=" + id + ",
       question.id=" + question.getId() + ",
       functionToCompute=" + functionToCompute
       + "]";
}
}
```

9. UserAnsweredSurvey.java

```
package com.hdsgs.entity;

import static javax.persistence.GenerationType.
    IDENTITY;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

import com.hdsgs.enums.UserRole;

/**
 * Represents a relation between a {@link
     UserRole#CONTRIBUTOR contributor} and
 * a {@link Survey survey} s/he answered. This
     class was generated, and is used
 * by Hibernate to persist survey data.
 */
@Entity
@Table( name = "user_answered_survey", catalog =
     "hdsgs" )
public class UserAnsweredSurvey implements java.
    io.Serializable
{
  private static final long serialVersionUID =
      -5574469045234222959L;

  private int id;

  private Survey survey;

  private User user;


  /**
   * Constructs a new object with default values
       . Needed by Hibernate.
   */
  protected UserAnsweredSurvey()
  {

  }


  /**
   * Constructs a new object with the specified
       values.
   *
   * @param survey
   * @param user
   *     contributor
   */
  public UserAnsweredSurvey( Survey survey, User
      user )
  {
    this.survey = survey;
    this.user = user;
  }


  /**
   * Returns the id of this object.
   *
   * @return id
   */
  @Id
  @GeneratedValue( strategy = IDENTITY )
  @Column( name = "id", unique = true, nullable
      = false )
```

```
public int getId()
{
  return this.id;
}


/**
 * Sets the id of this object.
 *
 * @param id
 */
public void setId( int id )
{
  this.id = id;
}


/**
 * Returns the survey associated with this
     object.
 *
 * @return survey
 */
@ManyToOne( fetch = FetchType.LAZY )
@JoinColumn( name = "survey_id", nullable =
    false )
public Survey getSurvey()
{
  return this.survey;
}


/**
 * Sets the survey associated with this survey
     .
 *
 * @param survey
 */
public void setSurvey( Survey survey )
{
  this.survey = survey;
}


/**
 * Returns the user associated with this
     object.
 *
 * @return user
 */
@ManyToOne( fetch = FetchType.LAZY )
@JoinColumn( name = "user_id", nullable =
    false )
public User getUser()
{
  return this.user;
}


/**
 * Sets the user associated with this object.
 *
 * @param user
 */
public void setUser( User user )
{
  this.user = user;
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + id;
  result = prime * result + ( ( survey == null
      ) ? 0 : survey.hashCode() );
  result = prime * result + ( ( user == null )
      ? 0 : user.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  UserAnsweredSurvey other = (
      UserAnsweredSurvey ) obj;

  if( id != other.id )
  {
    return false;
  }

  if( survey == null )
  {
    if( other.survey != null )
    {
      return false;
    }
  }
  else if( !survey.equals( other.survey ) )
  {
    return false;
  }

  if( user == null )
  {
    if( other.user != null )
    {
      return false;
    }
  }
  else if( !user.equals( other.user ) )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "UserAnsweredSurvey [id=" + id + ",
      survey.id=" + survey.getId() + ", user.
      id=" + user.getId() + "]";
}
}
```

10. BasicStats.java

```
package com.hdsgs.entity;

import static javax.persistence.GenerationType.
    IDENTITY;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

/**
 * Represents the basic statistics - average,
     minimum, and maximum - for a
 * {@link Question survey question}.
 */
@Entity
@Table( name = "survey_basic_stats", catalog = "
    hdsgs" )
public class BasicStats implements java.io.
    Serializable
{
  private static final long serialVersionUID =
      -8477511118520638203L;

  private Integer id;
```

```
private Survey survey;

private int questionNumber;

private double average;

private int minimum;

private int maximum;


/**
 * Constructs a new object with default values
      . Needed by Hibernate.
 */
protected BasicStats()
{

}


/**
 * Constructs a new object with the specified
      values.
 *
 * @param survey
 * @param questionNumber
 * @param average
 * @param minimum
 * @param maximum
 */
public BasicStats( Survey survey, int
     questionNumber, double average, int
     minimum, int maximum )
{
  this.survey = survey;
  this.questionNumber = questionNumber;
  this.average = average;
  this.minimum = minimum;
  this.maximum = maximum;
}


/**
 * Returns the id of this object.
 *
 * @return id
 */
@Id
@GeneratedValue( strategy = IDENTITY )
@Column( name = "id", unique = true, nullable
     = false )
public Integer getId()
{
  return this.id;
}


/**
 * Sets the id of this object.
 *
 * @param id
 */
public void setId( Integer id )
{
  this.id = id;
}


/**
 * Returns the survey associated with this
      object.
 *
 * @return survey
 */
@ManyToOne( fetch = FetchType.EAGER )
@JoinColumn( name = "survey_id", nullable =
     false )
public Survey getSurvey()
{
  return this.survey;
}


/**
 * Sets the survey associated with this object
      .
 *
 * @param survey
 */
public void setSurvey( Survey survey )
{
  this.survey = survey;
}


/**
 * Returns the question number of this object.
 *
 * @return question number
 */
@Column( name = "question_number", nullable =
     false )
public int getQuestionNumber()
{
  return this.questionNumber;
}


/**
 * Sets the question number of this object.
 *
 * @param questionNumber
 */
public void setQuestionNumber( int
     questionNumber )
{
  this.questionNumber = questionNumber;
}


/**
 * Returns the average.
 *
 * @return average
 */
@Column( name = "average", nullable = false,
     precision = 22, scale = 0 )
public double getAverage()
{
  return this.average;
}


/**
 * Sets the average.
 *
 * @param average
 */
public void setAverage( double average )
{
  this.average = average;
}


/**
 * Returns the minimum.
 *
 * @return minimum
 */
@Column( name = "minimum", nullable = false )
public int getMinimum()
{
  return this.minimum;
}


/**
 * Sets the minimum.
 *
 * @param minimum
 */
public void setMinimum( int minimum )
{
  this.minimum = minimum;
}


/**
 * Returns the maximum.
 *
 * @return maximum
 */
@Column( name = "maximum", nullable = false )
public int getMaximum()
{
  return this.maximum;
}


/**
 * Sets the maximum.
 *
 * @param maximum
```

```
 */
public void setMaximum( int maximum )
{
  this.maximum = maximum;
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;
  long temp;

  temp = Double.doubleToLongBits( average );
  result = prime * result + ( int ) ( temp ^ (
       temp >>> 32 ) );
  result = prime * result + ( ( id == null ) ?
       0 : id.hashCode() );
  result = prime * result + maximum;
  result = prime * result + minimum;
  result = prime * result + questionNumber;
  result = prime * result + ( ( survey == null
       ) ? 0 : survey.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  BasicStats other = ( BasicStats ) obj;

  if( Double.doubleToLongBits( average ) !=
       Double.doubleToLongBits( other.average
       ) )
  {
    return false;
  }

  if( id == null )
  {
    if( other.id != null )
    {
      return false;
    }
  }
  else if( !id.equals( other.id ) )
  {
    return false;
  }

  if( maximum != other.maximum )
  {
    return false;
  }

  if( minimum != other.minimum )
  {
    return false;
  }

  if( questionNumber != other.questionNumber )
  {
    return false;
  }

  if( survey == null )
  {
    if( other.survey != null )
    {
      return false;
    }
  }
  else if( !survey.equals( other.survey ) )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "SurveyBasicStats [id=" + id + ",
       survey=" + survey + ", questionNumber="
        + questionNumber + ", average=" +
       average + ", minimum="
      + minimum + ", maximum=" + maximum + "]"
           ;
}
}
```

11. Tally.java

```
package com.hdsgs.entity;

import static javax.persistence.GenerationType.
    IDENTITY;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

/**
 * Represents the tally of answers for a {@link
      Question survey question}.
 */
@Entity
@Table( name = "survey_tally", catalog = "hdsgs"
      )
public class Tally implements java.io.
    Serializable
{
  private static final long serialVersionUID =
       7447712607285438307L;

  private Integer id;

  private Survey survey;

  private int questionNumber;

  private String choice;

  private int count;


  /**
   * Constructs a new tally object with default
       values. Needed by Hibernate.
   */
  protected Tally()
  {

  }


  /**
   * Constructs a new tally object with the
       specified values.
   *
   * @param survey
   * @param questionNumber
   * @param choice
   * @param count
   */
  public Tally( Survey survey, int
       questionNumber, String choice, int count
       )
  {
    this.survey = survey;
    this.questionNumber = questionNumber;
    this.choice = choice;
    this.count = count;
  }
```

```
/**
 * Returns the id of this object.
 *
 * @return id
 */
@Id
@GeneratedValue( strategy = IDENTITY )
@Column( name = "id", unique = true, nullable
    = false )
public Integer getId()
{
  return this.id;
}


/**
 * Sets the id of this object.
 *
 * @param id
 */
public void setId( Integer id )
{
  this.id = id;
}


/**
 * Returns the survey associated with this
     object.
 *
 * @return survey
 */
@ManyToOne( fetch = FetchType.EAGER )
@JoinColumn( name = "survey_id", nullable =
    false )
public Survey getSurvey()
{
  return this.survey;
}


/**
 * Sets the survey associated with this object
     .
 *
 * @param survey
 */
public void setSurvey( Survey survey )
{
  this.survey = survey;
}


/**
 * Returns the question number of this object.
 *
 * @return question number
 */
@Column( name = "question_number", nullable =
    false )
public int getQuestionNumber()
{
  return this.questionNumber;
}


/**
 * Sets the question number of this object
 *
 * @param questionNumber
 */
public void setQuestionNumber( int
    questionNumber )
{
  this.questionNumber = questionNumber;
}


/**
 * Returns the choice.
 *
 * @return choice
 */
@Column( name = "choice", nullable = false,
    length = 50 )
public String getChoice()
{
  return this.choice;
}


/**
 * Sets the choice.
 *
 * @param choice
 */
public void setChoice( String choice )
{
  this.choice = choice;
}


/**
 * Returns the number of times the choice was
     selected as an answer.
 *
 * @return count
 */
@Column( name = "count", nullable = false )
public int getCount()
{
  return this.count;
}


/**
 * Sets the number of times the choice was
     selected as an answer.
 *
 * @param count
 */
public void setCount( int count )
{
  this.count = count;
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( ( choice == null
      ) ? 0 : choice.hashCode() );
  result = prime * result + count;
  result = prime * result + ( ( id == null ) ?
      0 : id.hashCode() );
  result = prime * result + questionNumber;
  result = prime * result + ( ( survey == null
      ) ? 0 : survey.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  Tally other = ( Tally ) obj;

  if( choice == null )
  {
    if( other.choice != null )
    {
      return false;
    }
  }
  else if( !choice.equals( other.choice ) )
  {
    return false;
  }

  if( count != other.count )
  {
    return false;
  }

  if( id == null )
```

```
    {
      if( other.id != null )
      {
        return false;
      }
    }
    else if( !id.equals( other.id ) )
    {
      return false;
    }

    if( questionNumber != other.questionNumber )
    {
      return false;
    }

    if( survey == null )
    {
      if( other.survey != null )
      {
        return false;
      }
    }
    else if( !survey.equals( other.survey ) )
    {
      return false;
    }

    return true;
  }


  @Override
  public String toString()
  {
    return "SurveyTally [id=" + id + ", survey="
        + survey + ", questionNumber=" +
        questionNumber + ", choice=" + choice +
        ", count=" + count
        + "]";
  }
}
```

12. UserDTO.java

```
package com.hdsgs.dto;

import javax.validation.constraints.Size;

import org.springframework.hateoas.
    ResourceSupport;

import com.hdsgs.entity.User;
import com.hdsgs.enums.UserRole;

/**
 * The data transfer object for a {@link User
     user}.
 */
public class UserDTO extends ResourceSupport
{
  private int userId;

  @Size( min = 2, max = 20, message = "should be
      2-20 characters" )
  private String username;

  @Size( min = 2, max = 255, message = "should
      be greater than 2 characters" )
  private String oldPassword;

  @Size( min = 2, max = 255, message = "should
      be greater than 2 characters" )
  private String password;

  private String retypedPassword;

  @Size( min = 2, max = 50, message = "should be
      2-50 characters" )
  private String name;

  @Size( min = 2, max = 50, message = "should be
      2-50 characters" )
  private String designation;

  @Size( min = 2, max = 50, message = "should be
      2-50 characters" )
  private String institution;

  @Size( min = 2, max = 30, message = "should be
      2-30 characters" )
  private String email;

  private UserRole role;

  private boolean isActive;


  /**
   * Constructs a new object with default values
       .
   */
  public UserDTO()
  {
    this.userId = 0;
    this.username = null;
    this.password = null;
    this.retypedPassword = null;
    this.name = null;
    this.designation = null;
    this.institution = null;
    this.email = null;
    this.role = null;
    this.isActive = true;
  }


  /**
   * Returns the id of this user.
   *
   * @return id
   */
  public int getUserId()
  {
    return this.userId;
  }


  /**
   * Sets the id of this user.
   *
   * @param id
   */
  public void setUserId( int id )
  {
    this.userId = id;
  }


  /**
   * Returns the username of this user.
   *
   * @return username
   */
  public String getUsername()
  {
    return this.username;
  }


  /**
   * Sets the username of this user.
   *
   * @param username
   */
  public void setUsername( String username )
  {
    this.username = username;
  }


  /**
   * Returns the old password of this user.
   *
   * @return old password
   */
  public String getOldPassword()
  {
    return oldPassword;
  }


  /**
   * Sets the old password of this user.
   *
   * @param oldPassword
   */
  public void setOldPassword( String oldPassword
      )
  {
    this.oldPassword = oldPassword;
```

```
}

/**
 * Returns the password of this user.
 *
 * @return password
 */
public String getPassword()
{
  return this.password;
}

/**
 * Sets the password of this user.
 *
 * @param password
 */
public void setPassword( String password )
{
  this.password = password;
}

/**
 * Returns the re-typed password from the form
      involving this user.
 *
 * @return re-typed password
 */
public String getRetypedPassword()
{
  return retypedPassword;
}

/**
 * Sets the re-typed password for the form
     involving this user.
 *
 * @param retypedPassword
 */
public void setRetypedPassword( String
    retypedPassword )
{
  this.retypedPassword = retypedPassword;
}

/**
 * Returns the name of this user.
 *
 * @return name
 */
public String getName()
{
  return this.name;
}

/**
 * Sets the name of this user.
 *
 * @param name
 */
public void setName( String name )
{
  this.name = name;
}

/**
 * Returns the designation of this user.
 *
 * @return position
 */
public String getDesignation()
{
  return this.designation;
}

/**
 * Sets the designation of this user.
 *
 * @param designation
 */
public void setDesignation( String designation
    )
{
  this.designation = designation;
}

/**
 * Returns the name of the institution where
     this user belongs to.
 *
 * @return institution
 */
public String getInstitution()
{
  return this.institution;
}

/**
 * Sets the name of the institution where this
      user belongs to.
 *
 * @param institution
 */
public void setInstitution( String institution
    )
{
  this.institution = institution;
}

/**
 * Returns the email address of this user.
 *
 * @return email address
 */
public String getEmail()
{
  return this.email;
}

/**
 * Sets the email address of this user.
 *
 * @param email
 */
public void setEmail( String email )
{
  this.email = email;
}

/**
 * Returns the role of this user.
 *
 * @return role
 */
public UserRole getRole()
{
  return this.role;
}

/**
 * Sets the role of this user.
 *
 * @param role
 */
public void setRole( UserRole role )
{
  this.role = role;
}

/**
 * @return is this user active
 */
public boolean isActive()
{
  return this.isActive;
}

/**
 * Sets whether this user is active.
 *
 * @param isActive
 *     defaults to {@code true} during
     initialization
 */
public void setActive( boolean isActive )
{
  this.isActive = isActive;
}
```

```
@Override
public int hashCode()
{
  final int prime = 31;
  int result = super.hashCode();

  result = prime * result + ( ( designation ==
       null ) ? 0 : designation.hashCode() );
  result = prime * result + ( ( email == null
      ) ? 0 : email.hashCode() );
  result = prime * result + ( ( institution ==
       null ) ? 0 : institution.hashCode() );
  result = prime * result + ( isActive ? 1231
      : 1237 );
  result = prime * result + ( ( name == null )
       ? 0 : name.hashCode() );
  result = prime * result + ( ( password ==
      null ) ? 0 : password.hashCode() );
  result = prime * result + ( (
      retypedPassword == null ) ? 0 :
      retypedPassword.hashCode() );
  result = prime * result + ( ( role == null )
       ? 0 : role.hashCode() );
  result = prime * result + userId;
  result = prime * result + ( ( username ==
      null ) ? 0 : username.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( !super.equals( obj ) )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  UserDTO other = ( UserDTO ) obj;

  if( designation == null )
  {
    if( other.designation != null )
    {
      return false;
    }
  }
  else if( !designation.equals( other.
      designation ) )
  {
    return false;
  }

  if( email == null )
  {
    if( other.email != null )
    {
      return false;
    }
  }
  else if( !email.equals( other.email ) )
  {
    return false;
  }

  if( institution == null )
  {
    if( other.institution != null )
    {
      return false;
    }
  }
  else if( !institution.equals( other.
      institution ) )
  {
    return false;
  }

  if( isActive != other.isActive )
  {
    return false;
  }

  if( name == null )
  {
    if( other.name != null )
    {
      return false;
    }
  }
  else if( !name.equals( other.name ) )
  {
    return false;
  }

  if( password == null )
  {
    if( other.password != null )
    {
      return false;
    }
  }
  else if( !password.equals( other.password )
      )
  {
    return false;
  }

  if( retypedPassword == null )
  {
    if( other.retypedPassword != null )
    {
      return false;
    }
  }
  else if( !retypedPassword.equals( other.
      retypedPassword ) )
  {
    return false;
  }

  if( role != other.role )
  {
    return false;
  }

  if( userId != other.userId )
  {
    return false;
  }

  if( username == null )
  {
    if( other.username != null )
    {
      return false;
    }
  }
  else if( !username.equals( other.username )
      )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "UserDTO [userId=" + userId + ",
       username=" + username + ", password=" +
        password + ", name=" + name + ",
       designation=" + designation
      + ", institution=" + institution + ",
           email=" + email + ", role=" + role
           + ", isActive=" + isActive + ", " +
            super.toString() + "]";
}
}
```

13. SurveyDTO.java

```
package com.hdsgs.dto;

import java.util.ArrayList;
import java.util.List;
```

```java
import javax.validation.Valid;
import javax.validation.constraints.Future;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.joda.time.LocalDate;
import org.springframework.format.annotation.
    DateTimeFormat;
import org.springframework.format.annotation.
    DateTimeFormat.ISO;
import org.springframework.hateoas.
    ResourceSupport;

import com.hdsgs.entity.Survey;
import com.hdsgs.enums.SurveyStatus;

/**
 * The data transfer object for a {@link Survey
     survey}.
 */
public class SurveyDTO extends ResourceSupport
{
  private final static String datePattern = "MMM
      dd, yyyy";

  private int surveyId;

  private String creator;

  private String institutionOfCreator;

  @Size( min = 2, max = 50, message = "Should be
      2-50 characters" )
  private String title;

  @NotNull( message = "Should not be empty" )
  @DateTimeFormat( iso = ISO.DATE )
  private LocalDate startDate;

  @NotNull( message = "Should not be empty" )
  @Future( message = "Should be in the future" )
  @DateTimeFormat( iso = ISO.DATE )
  private LocalDate endDate;

  @Size( min = 1, max = 255, message = "Should
      be 1-255 characters" )
  private String description;

  @Valid
  private List< QuestionDTO > questions;


  /**
   * Constructs a new object with default values
       .
   */
  public SurveyDTO()
  {
    this.surveyId = 0;
    this.creator = null;
    this.title = null;
    this.startDate = null;
    this.endDate = null;
    this.description = null;
    this.questions = new ArrayList< QuestionDTO
        >( 0 );
  }


  /**
   * Returns the id of this survey.
   *
   * @return id
   */
  public int getSurveyId()
  {
    return this.surveyId;
  }


  /**
   * Sets the id of this survey.
   *
   * @param id
   */
  public void setSurveyId( int id )
  {
    this.surveyId = id;
  }


  /**
   * Returns the title of this survey.
   *
   * @return title
   */
  public String getTitle()
  {
    return this.title;
  }


  /**
   * Sets the title of this survey.
   *
   * @param title
   */
  public void setTitle( String title )
  {
    this.title = title;
  }


  /**
   * Returns the username of the creator of this
       survey.
   *
   * @return username of creator
   */
  public String getCreator()
  {
    return this.creator;
  }


  /**
   * Sets the username of the creator of this
       survey.
   *
   * @param creator
   *     username
   */
  public void setCreator( String creator )
  {
    this.creator = creator;
  }


  /**
   * Returns the institution of the creator of
       this survey.
   *
   * @return institution of creator
   */
  public String getInstitutionOfCreator()
  {
    return institutionOfCreator;
  }


  /**
   * Sets the institution of the creator of this
       survey.
   *
   * @param institutionOfCreator
   */

  public void setInstitutionOfCreator( String
      institutionOfCreator )
  {
    this.institutionOfCreator =
        institutionOfCreator;
  }


  /**
   * Returns the date when this survey starts.
   *
   * @return start date
   */
  public LocalDate getStartDate()
  {
    return this.startDate;
  }


  /**
   * Returns this survey's start date in
       readable format, e.g., Jan 1, 1970.
   *
   * @return readable start date
   */
  public String getReadableStartDate()
```

```
{
  return this.startDate.toString( datePattern
      );
}


/**
 * Sets the date when this survey starts.
 *
 * @param startDate
 */
public void setStartDate( LocalDate startDate
    )
{
  this.startDate = startDate;
}


/**
 * Returns the date when this survey ends.
 *
 * @return end date
 */
public LocalDate getEndDate()
{
  return this.endDate;
}


/**
 * Returns this survey's end date in readable
     format, e.g., Jan 1, 1970.
 *
 * @return readable end date
 */
public String getReadableEndDate()
{
  return this.endDate.toString( datePattern );
}


/**
 * Sets the date when this survey ends.
 *
 * @param endDate
 */
public void setEndDate( LocalDate endDate )
{
  this.endDate = endDate;
}


/**
 * Returns the description for this survey.
 *
 * @return description
 */
public String getDescription()
{
  return description;
}


/**
 * Sets the description of this survey.
 *
 * @param description
 */
public void setDescription( String description
    )
{
  this.description = description;
}


/**
 * Returns the list of questions for this
     survey.
 *
 * @return questions
 */
public List< QuestionDTO > getQuestions()
{
  return this.questions;
}


/**
 * Sets the list of questions for this survey.
 *
 * @param questions
 */
public void setQuestions( List< QuestionDTO >
    questions )
{
  this.questions = questions;
}


/**
 * @return is today's date between this survey
     's start and end dates
 *     (inclusive)
 */
public boolean isActive()
{
  LocalDate now = LocalDate.now();

  return ( now.isEqual( this.startDate ) || (
      now.isAfter( this.startDate ) && now.
      isBefore( this.endDate ) ) ) && !now.
      equals( endDate );
}


public SurveyStatus getStatus()
{
  LocalDate now = LocalDate.now();

  if( now.isBefore( this.startDate ) )
  {
    return SurveyStatus.NOT_YET_STARTED;
  }
  else if( now.equals( this.endDate ) || now.
      isAfter( this.endDate ) )
  {
    return SurveyStatus.ENDED;
  }
  else
  {
    return SurveyStatus.ONGOING;
  }
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = super.hashCode();

  result = prime * result + ( ( creator ==
      null ) ? 0 : creator.hashCode() );
  result = prime * result + ( ( description ==
       null ) ? 0 : description.hashCode() );
  result = prime * result + ( ( endDate ==
      null ) ? 0 : endDate.hashCode() );
  result = prime * result + ( ( questions ==
      null ) ? 0 : questions.hashCode() );
  result = prime * result + ( ( startDate ==
      null ) ? 0 : startDate.hashCode() );
  result = prime * result + surveyId;
  result = prime * result + ( ( title == null
      ) ? 0 : title.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( !super.equals( obj ) )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  SurveyDTO other = ( SurveyDTO ) obj;

  if( creator == null )
  {
    if( other.creator != null )
    {
      return false;
```

```
    }
  }
  else if( !creator.equals( other.creator ) )
  {
    return false;
  }

  if( description == null )
  {
    if( other.description != null )
    {
      return false;
    }
  }
  else if( !description.equals( other.
      description ) )
  {
    return false;
  }

  if( endDate == null )
  {
    if( other.endDate != null )
    {
      return false;
    }
  }
  else if( !endDate.equals( other.endDate ) )
  {
    return false;
  }

  if( questions == null )
  {
    if( other.questions != null )
    {
      return false;
    }
  }
  else if( !questions.equals( other.questions
      ) )
  {
    return false;
  }

  if( startDate == null )
  {
    if( other.startDate != null )
    {
      return false;
    }
  }
  else if( !startDate.equals( other.startDate
      ) )
  {
    return false;
  }

  if( surveyId != other.surveyId )
  {
    return false;
  }

  if( title == null )
  {
    if( other.title != null )
    {
      return false;
    }
  }
  else if( !title.equals( other.title ) )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "SurveyDTO [surveyId=" + surveyId + "
      , creator=" + creator + ", title=" +
      title + ", startDate=" + startDate + ",
       endDate=" + endDate
      + ", description=" + description + ",
          questions=" + questions + "]";
}
}
```

14. QuestionDTO.java

```
package com.hdsgs.dto;

import java.util.ArrayList;
import java.util.List;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.
    NotEmpty;
import org.hibernate.validator.constraints.Range
    ;

import com.hdsgs.entity.Question;
import com.hdsgs.enums.Function;
import com.hdsgs.enums.InputType;

/**
 * The data transfer object for a survey {@link
     Question question}.
 */
public class QuestionDTO
{
  private int id;

  private int surveyId;

  @Range( min = 1, max = 999, message = "
      Question number too large" )
  private int questionNumber;

  @Size( min = 2, max = 255, message = "Question
      should be 2-255 characters" )
  private String question;

  @NotNull( message = "Select an input type" )
  private InputType inputType;

  @NotEmpty( message = "Select at least 1 result
      function to compute" )
  private List< Function > functions;

  private List< String > choices;


  /**
   * Constructs a new object with default values
       .
   */
  public QuestionDTO()
  {
    this.id = 0;
    this.surveyId = 0;
    this.questionNumber = 0;
    this.question = "";
    this.inputType = null;
    this.functions = new ArrayList< Function >(
        0 );
    this.choices = new ArrayList< String >( 0 );
  }


  /**
   * Returns the id of this question.
   *
   * @return id
   */
  public int getId()
  {
    return id;
  }


  /**
   * Sets the id of this question.
   *
   * @param id
   */
  public void setId( int id )
  {
    this.id = id;
  }


  /**
   * Returns the question number of this
       question.
   *
```

```
 * @return question number
 */
public int getQuestionNumber()
{
  return this.questionNumber;
}


/**
 * Sets the question number of this question.
 *
 * @param questionNumber
 */
public void setQuestionNumber( int
    questionNumber )
{
  this.questionNumber = questionNumber;
}


/**
 * Returns the id of the survey to which this
     question is found.
 *
 * @return survey id
 */
public int getSurveyId()
{
  return this.surveyId;
}


/**
 * Sets the survey to which this question is
     found.
 *
 * @param surveyId
 */
public void setSurveyId( int surveyId )
{
  this.surveyId = surveyId;
}


/**
 * Returns the question string.
 *
 * @return question
 */
public String getQuestion()
{
  return this.question;
}


/**
 * Sets the question string.
 *
 * @param question
 */
public void setQuestion( String question )
{
  this.question = question;
}


/**
 * Returns the input type of this question.
 *
 * @return input type
 */
public InputType getInputType()
{
  return this.inputType;
}


/**
 * Sets the input type of the question.
 *
 * @param inputType
 */
public void setInputType( InputType inputType
    )
{
  this.inputType = inputType;
}


/**
 * Returns the list of choices for this
     question.
 *
 * @return choices
 */
public List< String > getChoices()
{
  return this.choices;
}


/**
 * Sets the list of choices for this question.
 *
 * @param questionChoices
 */
public void setChoices( List< String >
    questionChoices )
{
  this.choices = questionChoices;
}


/**
 * Returns the list of result functions to
     compute for this question.
 *
 * @return result functions
 */
public List< Function > getFunctions()
{
  return functions;
}


/**
 * Sets the list of result functions to
     compute for this question.
 *
 * @param functions
 */
public void setFunctions( List< Function >
    functions )
{
  this.functions = functions;
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( ( choices ==
      null ) ? 0 : choices.hashCode() );
  result = prime * result + ( ( functions ==
      null ) ? 0 : functions.hashCode() );
  result = prime * result + id;
  result = prime * result + ( ( inputType ==
      null ) ? 0 : inputType.hashCode() );
  result = prime * result + ( ( question ==
      null ) ? 0 : question.hashCode() );
  result = prime * result + questionNumber;
  result = prime * result + surveyId;

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  QuestionDTO other = ( QuestionDTO ) obj;

  if( choices == null )
  {
    if( other.choices != null )
    {
```

```
        return false;
      }
    }
    else if( !choices.equals( other.choices ) )
    {
      return false;
    }

    if( functions == null )
    {
      if( other.functions != null )
      {
        return false;
      }
    }
    else if( !functions.equals( other.functions
        ) )
    {
      return false;
    }

    if( id != other.id )
    {
      return false;
    }

    if( inputType != other.inputType )
    {
      return false;
    }

    if( question == null )
    {
      if( other.question != null )
      {
        return false;
      }
    }
    else if( !question.equals( other.question )
        )
    {
      return false;
    }

    if( questionNumber != other.questionNumber )
    {
      return false;
    }

    if( surveyId != other.surveyId )
    {
      return false;
    }

    return true;
  }


  @Override
  public String toString()
  {
    return "QuestionDTO [id=" + id + ", surveyId
        =" + surveyId + ", questionNumber=" +
        questionNumber + ", question=" +
        question + ", inputType="
        + inputType + ", functions=" + functions
            + ", choices=" + choices + "]";
  }
}
```

15. SurveyAnswer.java

```
package com.hdsgs.dto;

import java.util.ArrayList;
import java.util.List;

import org.hibernate.validator.constraints.
    NotEmpty;
import org.springframework.hateoas.
    ResourceSupport;

/**
 * The data transfer object for an answer for a
     survey.
 */
public class SurveyAnswer extends
    ResourceSupport
{
  private int surveyId;

  private String contributor;

  @NotEmpty( message = "answers list should not
      be empty" )
  private List< String > answers = new ArrayList
      < String >( 0 );


  /**
   * Constructs a new survey answer object.
   */
  public SurveyAnswer()
  {

  }


  /**
   * Returns the id of the survey to be answered
       .
   *
   * @return id
   */
  public int getSurveyId()
  {
    return surveyId;
  }


  /**
   * Sets the survey id associated with this
       survey answer.
   *
   * @param surveyId
   */
  public void setSurveyId( int surveyId )
  {
    this.surveyId = surveyId;
  }


  /**
   * Returns the username of the contributor who
       will answer the survey.
   *
   * @return username of contributor
   */
  public String getContributor()
  {
    return contributor;
  }


  /**
   * Sets the contributor for the survey who
       will answer the survey.
   *
   * @param contributor
   *     username
   */
  public void setContributor( String contributor
      )
  {
    this.contributor = contributor;
  }


  /**
   * Returns the list of answers
   *
   * @return answers
   */
  public List< String > getAnswers()
  {
    return answers;
  }


  /**
   * Sets the answers for a survey.
   *
   * @param answers
   */
  public void setAnswers( List< String > answers
      )
  {
    this.answers = answers;
  }
```

```
  @Override
  public String toString()
  {
    return "SurveyAnswer [surveyId=" + surveyId
        + ", contributor=" + contributor + ",
        answers=" + answers + "]";
  }
}
```

16. QuestionResults.java

```
package com.hdsgs.dto;

import java.util.LinkedHashMap;

import com.hdsgs.entity.QuestionFunction;

/**
 * The data transfer object for a question's {
     @link QuestionFunction result
 * function}.
 */
public class QuestionResults
{
  private int surveyId;

  private int questionNumber;

  private String question;

  private double average;

  private int minimum;

  private int maximum;

  private TallyDto tallyDto;


  /**
   * Initializes a new object with default
       values.
   */
  public QuestionResults()
  {
    surveyId = 0;
    questionNumber = 0;
    question = null;
    average = -1;
    minimum = -1;
    maximum = -1;
    tallyDto = new TallyDto( new LinkedHashMap<
        String, Integer >( 0 ) );
  }


  /**
   * Returns the survey id of the question.
   *
   * @return survey id
   */
  public int getSurveyId()
  {
    return surveyId;
  }


  /**
   * Sets the survey id of the question.
   *
   * @param surveyId
   */
  public void setSurveyId( int surveyId )
  {
    this.surveyId = surveyId;
  }


  /**
   * Returns the question number.
   *
   * @return question number
   */
  public int getQuestionNumber()
  {
    return questionNumber;
  }


  /**
   * Sets the question number.
   *
   * @param questionNumber
   */
  public void setQuestionNumber( int
      questionNumber )
  {
    this.questionNumber = questionNumber;
  }


  /**
   * Returns the question.
   *
   * @return question
   */
  public String getQuestion()
  {
    return question;
  }


  /**
   * Sets the question.
   *
   * @param question
   */
  public void setQuestion( String question )
  {
    this.question = question;
  }


  /**
   * Returns the average of the answers to the
       question.
   *
   * @return average answer
   */
  public double getAverage()
  {
    return average;
  }


  /**
   * Sets the average of the answers to the
       question.
   *
   * @param average
   */
  public void setAverage( double average )
  {
    this.average = average;
  }


  /**
   * Returns the minimum of the answers to the
       question.
   *
   * @return minimum answer
   */
  public int getMinimum()
  {
    return minimum;
  }


  /**
   * Sets the minimum of the answers to the
       question.
   *
   * @param minimum
   */
  public void setMinimum( int minimum )
  {
    this.minimum = minimum;
  }


  /**
   * Returns the maximum of the answers to the
       question.
   *
   * @return maximum answer
   */
  public int getMaximum()
  {
    return maximum;
  }
```

```
/**
 * Sets the maximum of the answers to the
      question.
 *
 * @param maximum
 */
public void setMaximum( int maximum )
{
  this.maximum = maximum;
}


/**
 * Returns the tally of answers to the
      question.
 *
 * @return tally of answers
 */
public TallyDto getTally()
{
  return tallyDto;
}


/**
 * Sets the tally of answers to the question
 *
 * @param tallyDto
 */
public void setTally( TallyDto tallyDto )
{
  this.tallyDto = tallyDto;
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;
  long temp;

  temp = Double.doubleToLongBits( average );
  result = prime * result + ( int ) ( temp ^ (
       temp >>> 32 ) );
  result = prime * result + maximum;
  result = prime * result + minimum;
  result = prime * result + ( ( question ==
       null ) ? 0 : question.hashCode() );
  result = prime * result + questionNumber;
  result = prime * result + surveyId;
  result = prime * result + ( ( tallyDto ==
       null ) ? 0 : tallyDto.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  QuestionResults other = ( QuestionResults )
       obj;

  if( Double.doubleToLongBits( average ) !=
       Double.doubleToLongBits( other.average
       ) )
  {
    return false;
  }

  if( maximum != other.maximum )
  {
    return false;
  }

  if( minimum != other.minimum )
  {
    return false;
  }

  if( question == null )
  {
    if( other.question != null )
    {
      return false;
    }
  }
  else if( !question.equals( other.question )
       )
  {
    return false;
  }

  if( questionNumber != other.questionNumber )
  {
    return false;
  }

  if( surveyId != other.surveyId )
  {
    return false;
  }

  if( tallyDto == null )
  {
    if( other.tallyDto != null )
    {
      return false;
    }
  }
  else if( !tallyDto.equals( other.tallyDto )
       )
  {
    return false;
  }

  return true;
}


@Override
public String toString()
{
  return "QuestionResults [surveyId=" +
       surveyId + ", questionNumber=" +
       questionNumber + ", question=" +
       question + ", average=" + average
      + ", minimum=" + minimum + ", maximum="
           + maximum + ", tally=" + tallyDto +
            "]";
}
}
```

17. TallyDTO.java

```
package com.hdsgs.dto;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.stream.Collectors;

/**
 * Helper class for holding the tally of answers
      to a question.
 */
public class TallyDto
{
  private Map< String, Integer > counts;

  private List< String > choicesWithMinCount;

  private List< String > choicesWithMaxCount;


  /**
   * Constructs a new tally with the given map
       of counts.
   *
   * @param counts
   *     where key is the choice, and value is
       the count
```

```
 */
public TallyDto( Map< String, Integer > counts
    )
{
  this.counts = new LinkedHashMap< String,
      Integer >( counts );
  choicesWithMinCount = new ArrayList< String
      >( 0 );
  choicesWithMaxCount = new ArrayList< String
      >( 0 );

  if( counts.size() > 0 )
  {
    setChoicesWithMinCount();
    setChoicesWithMaxCount();
  }
  else
  {
    choicesWithMinCount = new ArrayList<
        String >( 0 );
    choicesWithMaxCount = new ArrayList<
        String >( 0 );
  }
}


/**
 * Determines which choices have the least
     count.
 */
private void setChoicesWithMinCount()
{
  int minCount = counts.entrySet().iterator().
      next().getValue();
  for( Entry< String, Integer > entry : counts
      .entrySet() )
  {
    if( entry.getValue() < minCount )
    {
      minCount = entry.getValue();
    }
  }

  for( Entry< String, Integer > entry : counts
      .entrySet() )
  {
    if( entry.getValue() == minCount )
    {
      choicesWithMinCount.add( entry.getKey()
          );
    }
  }
}


/**
 * Determines which choices have the most
     count.
 */
private void setChoicesWithMaxCount()
{
  int maxCount = 0;
  for( Entry< String, Integer > entry : counts
      .entrySet() )
  {
    if( entry.getValue() > maxCount )
    {
      maxCount = entry.getValue();
    }
  }

  for( Entry< String, Integer > entry : counts
      .entrySet() )
  {
    if( entry.getValue() == maxCount )
    {
      choicesWithMaxCount.add( entry.getKey()
          );
    }
  }
}


/**
 * Returns the map of counts.
 *
 * @return counts
 */
public Map< String, Integer > getCounts()
{
  return counts;
}


/**
 * Returns the list of choices with the
     minimum count.
 *
 * @return choices with the minimum count
 */
public String getChoicesWithMinCount()
{
  return choicesWithMinCount.stream().collect(
      Collectors.joining( ", " ) );
}


/**
 * Returns the list of choices with the
     maximum count.
 *
 * @return choices with the maximum count
 */
public String getChoicesWithMaxCount()
{
  return choicesWithMaxCount.stream().collect(
      Collectors.joining( ", " ) );
}


@Override
public int hashCode()
{
  final int prime = 31;
  int result = 1;

  result = prime * result + ( (
      choicesWithMaxCount == null ) ? 0 :
      choicesWithMaxCount.hashCode() );
  result = prime * result + ( (
      choicesWithMinCount == null ) ? 0 :
      choicesWithMinCount.hashCode() );
  result = prime * result + ( ( counts == null
      ) ? 0 : counts.hashCode() );

  return result;
}


@Override
public boolean equals( Object obj )
{
  if( this == obj )
  {
    return true;
  }

  if( obj == null )
  {
    return false;
  }

  if( getClass() != obj.getClass() )
  {
    return false;
  }

  TallyDto other = ( TallyDto ) obj;

  if( choicesWithMaxCount == null )
  {
    if( other.choicesWithMaxCount != null )
    {
      return false;
    }
  }
  else if( !choicesWithMaxCount.equals( other.
      choicesWithMaxCount ) )
  {
    return false;
  }

  if( choicesWithMinCount == null )
  {
    if( other.choicesWithMinCount != null )
    {
      return false;
    }
  }
  else if( !choicesWithMinCount.equals( other.
      choicesWithMinCount ) )
  {
    return false;
  }
```

```
    if( counts == null )
    {
      if( other.counts != null )
      {
        return false;
      }
    }
    else if( !counts.equals( other.counts ) )
    {
      return false;
    }

    return true;
  }


  @Override
  public String toString()
  {
    return "TallyDto [counts=" + counts + ",
        choicesWithMinCount=" +
        choicesWithMinCount + ",
        choicesWithMaxCount=" +
        choicesWithMaxCount + "]";
  }
}
```

18. UserAssembler.java

```
package com.hdsgs.dto.assembler;

import static org.springframework.hateoas.mvc.
    ControllerLinkBuilder.linkTo;

import org.springframework.hateoas.mvc.
    ResourceAssemblerSupport;
import org.springframework.stereotype.Component;

import com.hdsgs.controller.UserController;
import com.hdsgs.dto.UserDTO;

/**
 * Resource assembler for {@link UserDTO}.
 */
@Component
public class UserAssembler extends
    ResourceAssemblerSupport< UserDTO, UserDTO
    >
{
  /**
   * Constructs a new UserAssembler object.
   */
  public UserAssembler()
  {
    super( UserController.class, UserDTO.class )
        ;
  }


  /**
   * Adds links navigate this user.
   */
  @Override
  public UserDTO toResource( UserDTO userDto )
  {
    userDto.add( linkTo( UserController.class ).
        slash( userDto.getUsername() ).
        withSelfRel() );
    userDto.add( linkTo( UserController.class ).
        slash( userDto.getUsername() ).withRel(
        "toggleStatus" ) );

    if( userDto.isActive() )
    {
      userDto.add( linkTo( UserController.class
          ).slash( userDto.getUsername() ).
          slash( "update" ).withRel( "update" )
          );
    }

    return userDto;
  }


  /**
   * Adds links to submit and cancel the create
       new user form.
   *
   * @param userDto
   * @return user DTO with the added links
   */
  public UserDTO toCreateFormResource( UserDTO
      userDto )
  {
    userDto.add( linkTo( UserController.class ).
        withRel( "cancel" ) );
    userDto.add( linkTo( UserController.class ).
        withRel( "submit" ) );

    return userDto;
  }


  /**
   * Adds links to submit and cancel the update
       user form.
   *
   * @param userDto
   * @return user DTO with the added links
   */
  public UserDTO toUpdateFormResource( UserDTO
      userDto )
  {
    userDto.add( linkTo( UserController.class ).
        slash( userDto.getUsername() ).withRel(
        "cancel" ) );
    userDto.add( linkTo( UserController.class ).
        slash( userDto.getUsername() ).withRel(
        "submit" ) );

    return userDto;
  }
}
```

19. SurveyAssembler.java

```
package com.hdsgs.dto.assembler;

import static org.springframework.hateoas.mvc.
    ControllerLinkBuilder.linkTo;

import org.joda.time.LocalDate;
import org.springframework.hateoas.mvc.
    ResourceAssemblerSupport;
import org.springframework.stereotype.Component;

import com.hdsgs.controller.SurveyController;
import com.hdsgs.controller.UserController;
import com.hdsgs.dto.SurveyDTO;

/**
 * Resource assembler for {@link SurveyDTO}.
 */
@Component
public class SurveyAssembler extends
    ResourceAssemblerSupport< SurveyDTO,
    SurveyDTO >
{
  /**
   * Constructs a new SurveyAssembler object.
   */
  public SurveyAssembler()
  {
    super( SurveyController.class, SurveyDTO.
        class );
  }


  /**
   * Adds links to navigate this survey.
   */
  @Override
  public SurveyDTO toResource( SurveyDTO
      surveyDto )
  {
    surveyDto.add( linkTo( SurveyController.
        class ).slash( surveyDto.getSurveyId()
        ).withSelfRel() );
    surveyDto.add( linkTo( UserController.class
        ).slash( surveyDto.getCreator() ).
        withRel( "creator" ) );

    if( LocalDate.now().isAfter( surveyDto.
        getEndDate() ) || LocalDate.now().
        equals( surveyDto.getEndDate() ) )
    {
```

```
        surveyDto.add( linkTo( SurveyController.
            class ).slash( surveyDto.getSurveyId
            () ).slash( "results" ).withRel( "
            results" ) );
        surveyDto.add( linkTo( SurveyController.
            class ).slash( surveyDto.getSurveyId
            () ).slash( "results" ).slash( "
            download" )
            .withRel( "downloadResults" ) );
      }

      return surveyDto;
    }


    /**
     * Adds links to submit and cancel the create
         new survey form.
     *
     * @param surveyDto
     * @return survey DTO with the added links
     */
    public SurveyDTO toCreateFormResource(
        SurveyDTO surveyDto )
    {
      surveyDto.add( linkTo( SurveyController.
          class ).withRel( "cancel" ) );
      surveyDto.add( linkTo( SurveyController.
          class ).withRel( "submit" ) );

      return surveyDto;
    }


    /**
     * Adds links to submit and cancel the update
         survey form.
     *
     * @param surveyDto
     * @return survey DTO with the added links
     */
    public SurveyDTO toUpdateFormResource(
        SurveyDTO surveyDto )
    {
      surveyDto.add( linkTo( SurveyController.
          class ).slash( surveyDto.getSurveyId()
          ).withRel( "cancel" ) );
      surveyDto.add( linkTo( SurveyController.
          class ).slash( surveyDto.getSurveyId()
          ).withRel( "submit" ) );

      return surveyDto;
    }
}
```

20. SurveyAnswerAssembler.java

```
package com.hdsgs.dto.assembler;

import static org.springframework.hateoas.mvc.
    ControllerLinkBuilder.linkTo;

import org.springframework.hateoas.
    ResourceAssembler;
import org.springframework.stereotype.Component;

import com.hdsgs.controller.SurveyController;
import com.hdsgs.dto.SurveyAnswer;

/**
 * Resource assembler of {@link SurveyAnswer}.
 */
@Component
public class SurveyAnswerAssembler implements
    ResourceAssembler< SurveyAnswer,
    SurveyAnswer >
{
  /**
   * Adds links to self, submit, and cancel
       answer survey form.
   *
   * @param surveyAnswer
   * @return survey answer object with the added
       links
   */
  @Override
  public SurveyAnswer toResource( SurveyAnswer
      surveyAnswer )
  {
    surveyAnswer.add( linkTo( SurveyController.
        class ).slash( surveyAnswer.getSurveyId
        () ).slash( "answer" ).withSelfRel() );
    surveyAnswer.add( linkTo( SurveyController.
        class ).slash( surveyAnswer.getSurveyId
        () ).slash( "answer" ).withRel( "submit
        " ) );
    surveyAnswer.add( linkTo( SurveyController.
        class ).slash( surveyAnswer.getSurveyId
        () ).withRel( "cancel" ) );

    return surveyAnswer;
  }
}
```

21. Mapper.java

```
package com.hdsgs.utility;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import com.hdsgs.dto.QuestionDTO;
import com.hdsgs.dto.QuestionResults;
import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.dto.TallyDto;
import com.hdsgs.dto.UserDTO;
import com.hdsgs.entity.BasicStats;
import com.hdsgs.entity.Question;
import com.hdsgs.entity.QuestionChoice;
import com.hdsgs.entity.QuestionFunction;
import com.hdsgs.entity.Survey;
import com.hdsgs.entity.Tally;
import com.hdsgs.entity.User;
import com.hdsgs.enums.Function;

/**
 * A mapping utility class used to convert from
     an entity to a DTO.
 */
public final class Mapper
{
  /**
   * Maps a {@link User user entity} to a {@link
       UserDTO user DTO}.
   *
   * @param userEntity
   * @return equivalent user DTO
   */
  public static UserDTO toDto( User userEntity )
  {
    UserDTO userDto = new UserDTO();
    userDto.setUserId( userEntity.getId() );
    userDto.setUsername( userEntity.getUsername
        () );
    userDto.setName( userEntity.getName() );
    userDto.setDesignation( userEntity.
        getDesignation() );
    userDto.setInstitution( userEntity.
        getInstitution() );
    userDto.setEmail( userEntity.getEmail() );
    userDto.setRole( userEntity.getRole() );
    userDto.setActive( userEntity.isActive() );

    return userDto;
  }


  /**
   * Maps a {@link Survey survey entity} to a {
       @link SurveyDTO survey DTO}.
   *
   * @param surveyEntity
   * @return equivalent survey DTO
   */
  public static SurveyDTO toDto( Survey
      surveyEntity )
  {
    List< QuestionDTO > questions = surveyEntity
        .getQuestions().stream().map( Mapper::
        toDto ).collect( Collectors.toList() );

    surveyEntity.getQuestions().stream().collect
        ( Collectors.toList() );

    SurveyDTO surveyDto = new SurveyDTO();
    surveyDto.setSurveyId( surveyEntity.getId()
        );
```

```
    surveyDto.setCreator( surveyEntity.
        getCreator().getUsername() );
    surveyDto.setInstitutionOfCreator(
        surveyEntity.getCreator().
        getInstitution() );
    surveyDto.setTitle( surveyEntity.getTitle()
        );
    surveyDto.setStartDate( surveyEntity.
        getStartDate() );
    surveyDto.setEndDate( surveyEntity.
        getEndDate() );
    surveyDto.setDescription( surveyEntity.
        getDescription() );
    surveyDto.setQuestions( questions );

    return surveyDto;
  }


  /**
   * Maps a {@link Question question entity} to
       a {@QuestionDTO
   * question DTO}.
   *
   * @param questionEntity
   * @return equivalent question DTO
   */
  public static QuestionDTO toDto( Question
      questionEntity )
  {
    List< String > choices = questionEntity.
        getChoices().stream().map(
        QuestionChoice::getOption ).collect(
        Collectors.toList() );

    List< Function > functions = questionEntity.
        getFunctions().stream().map(
        QuestionFunction::getFunctionToCompute
        )
        .collect( Collectors.toList() );

    QuestionDTO questionDto = new QuestionDTO();
    questionDto.setId( questionEntity.getId() );
    questionDto.setSurveyId( questionEntity.
        getSurvey().getId() );
    questionDto.setQuestionNumber(
        questionEntity.getQuestionNumber() );
    questionDto.setQuestion( questionEntity.
        getQuestion() );
    questionDto.setInputType( questionEntity.
        getInputType() );
    questionDto.setChoices( choices );
    questionDto.setFunctions( functions );

    return questionDto;
  }


  /**
   * Maps a survey's basic statistics and
       tallies of answers to a list of
   * question results.
   *
   * @param questions
   * @param basicStats
   * @param tally
   * @return list of question results
   */
  public static List< QuestionResults >
      toResults( List< Question > questions,
      List< BasicStats > basicStats, List<
      Tally > tally )
  {
    List< QuestionResults > results = new
        ArrayList< QuestionResults >();

    List< Integer > statsQuestions = basicStats.
        stream().map( bs -> bs.
        getQuestionNumber() ).collect(
        Collectors.toList() );

    for( Question q : questions )
    {
      int questionNumber = q.getQuestionNumber()
          ;
      QuestionResults res = new QuestionResults
          ();

      res.setSurveyId( q.getSurvey().getId() );
      res.setQuestion( q.getQuestion() );
      res.setQuestionNumber( questionNumber );

      if( statsQuestions.contains(
          questionNumber ) )
      {
        BasicStats stats = basicStats.get(
            statsQuestions.indexOf(
            questionNumber ) );

        res.setAverage( stats.getAverage() );
        res.setMinimum( stats.getMinimum() );
        res.setMaximum( stats.getMaximum() );
      }
      else
      {
        List< Tally > tallyOfQuestionAnswers =
            tally.stream().filter( t -> t.
            getQuestionNumber() ==
            questionNumber )
            .collect( Collectors.toList() );

        Map< String, Integer > counts =
            tallyOfQuestionAnswers.stream().
            collect( Collectors.toMap( t -> t.
            getChoice(), t -> t.getCount() ) );

        res.setTally( new TallyDto( counts ) );
      }

      results.add( res );
    }

    return results;
  }
}
```

22. UserRepository.java

```
package com.hdsgs.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.
    JpaRepository;
import org.springframework.stereotype.Repository
    ;

import com.hdsgs.entity.User;

/**
 * A JPA repository which contains data access
     methods involving the
 * {@link User users}. The implementation of
     this interface is provided by
 * Spring Data JPA.
 */
@Repository
public interface UserRepository extends
    JpaRepository< User, Integer >
{
  /**
   * Retrieves the user with the given id.
   *
   * @param userId
   * @return optional user
   */
  Optional< User > findById( Integer userId );


  /**
   * Retrieves the user with the given username.
   *
   * @param username
   * @return optional user
   */
  Optional< User > findByUsername( String
      username );
}
```

23. SurveyRepository.java

```
package com.hdsgs.repository;

import java.util.List;
import java.util.Optional;

import org.joda.time.LocalDate;
import org.springframework.data.jpa.repository.
    JpaRepository;
```

```
import org.springframework.stereotype.Repository
    ;

import com.hdsgs.entity.Survey;
import com.hdsgs.enums.UserRole;

/**
 * A JPA repository which contains data access
     methods that involves the
 * {@link Survey surveys}. The implementation of
     this interface is provided by
 * Spring Data JPA.
 */
@Repository
public interface SurveyRepository extends
    JpaRepository< Survey, Integer >
{
  /**
   * Retrieves a survey with the given id.
   *
   * @param surveyId
   * @return optional survey
   */
  Optional< Survey > findById( Integer surveyId
      );


  /**
   * If the user, determined by the given
     username, is a
   * {@link UserRole#CONTRIBUTOR contributor},
     this method returns the surveys
   * answered. If the user is a {@link UserRole#
     RESEARCH_LEADER research
   * leader}, this method returns the surveys
     created.
   *
   * @param username
   */
  List< Survey > findByCreatorUsername( String
      username );


  /**
   * Retrieves a list of surveys that will end
     at the specified date.
   *
   * @param endDate
   * @return list of surveys that will end at
     the specified date
   */
  List< Survey > findByEndDate( LocalDate
      endDate );
}
```

24. QuestionRepository.java

```
package com.hdsgs.repository;

import org.springframework.data.jpa.repository.
    JpaRepository;
import org.springframework.stereotype.Repository
    ;

import com.hdsgs.entity.Question;

/**
 * A JPA repository which contains data access
     methods that involves the
 * {@link Question questions} in a survey. The
     implementation of this interface
 * is provided by Spring Data JPA.
 */
@Repository
public interface QuestionRepository extends
    JpaRepository< Question, Integer >
{

}
```

25. QuestionChoiceRepository.java

```
package com.hdsgs.repository;

import org.springframework.data.jpa.repository.
    JpaRepository;
import org.springframework.stereotype.Repository
    ;

import com.hdsgs.entity.QuestionChoice;

/**
 * A JPA repository which contains data access
     methods that involves the
 * {@link QuestionChoice choices} of a question
     in a survey. The implementation
 * of this interface is provided by Spring Data
     JPA.
 */
@Repository
public interface QuestionChoiceRepository
    extends JpaRepository< QuestionChoice,
    Integer >
{
  /**
   * Retrieves a {@link QuestionChoice} that has
       the given question ID and
   * option value.
   *
   * @param questionId
   * @param option
   * @return entity having the given parameters
   */
  QuestionChoice findByQuestionIdAndOption( int
      questionId, String option );
}
```

26. QuestionFunctionRepository.java

```
package com.hdsgs.repository;

import org.springframework.data.jpa.repository.
    JpaRepository;
import org.springframework.stereotype.Repository
    ;

import com.hdsgs.entity.QuestionFunction;
import com.hdsgs.enums.Function;

/**
 * A JPA repository which contains data access
     methods that involves the
 * {@link QuestionFunction result functions} to
     compute per question in a
 * survey. The implementation of this interface
     is provided by Spring Data JPA.
 */
@Repository
public interface QuestionFunctionRepository
    extends JpaRepository< QuestionFunction,
    Integer >
{
  /**
   * Retrieves a {@link QuestionFunction} that
       has the given question ID and
   * function.
   *
   * @param questionId
   * @param function
   * @return entity having the given parameters
   */
  QuestionFunction
      findByQuestionIdAndFunctionToCompute( int
      questionId, Function function );
}
```

27. UserAnsweredSurveyRepository.java

```
package com.hdsgs.repository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.jpa.repository.
    JpaRepository;
import org.springframework.stereotype.Repository
    ;

import com.hdsgs.entity.UserAnsweredSurvey;

/**
```

```
 * A JPA repository which contains data access
      methods that involves the {@link
      UserAnsweredSurvey surveys answered by
 * users}. The implementation of this interface
      is provided by Spring Data JPA.
 */
@Repository
public interface UserAnsweredSurveyRepository
     extends JpaRepository< UserAnsweredSurvey,
     Integer >
{
  /**
   * Retrieves a list of surveys answered by the
        user.
   *
   * @param username
   * @return a list of surveys answered by the
        user
   */
  List< UserAnsweredSurvey > findByUserUsername(
       String username );


  /**
   * Returns the number of contributors for the
       survey.
   *
   * @param surveyId
   * @return number of contributors
   */
  int countBySurveyId( int surveyId );


  /**
   * Retrieves the record with the given user
       and survey id.
   *
   * @return optional record
   */
  Optional< UserAnsweredSurvey >
       findBySurveyIdAndUserUsername( int
       surveyId, String username );
}
```

28. BaseService.java

```
package com.hdsgs.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Base service that contains the logger to be
      used by subclasses.
 *
 */
public abstract class BaseService
{
  protected final Logger logger = LoggerFactory.
       getLogger( getClass() );
}
```

29. UserService.java

```
package com.hdsgs.service;

import java.util.List;

import com.hdsgs.dto.UserDTO;
import com.hdsgs.exception.
     ResourceNotFoundException;
import com.hdsgs.exception.
     ServiceOperationException;

/**
 * Caters user services such as creating,
      updating, retrieving, deactivating, and
      activating users.
 */
public interface UserService
{
  /**
   * Creates a new user.
   *
   * @param newUser
   * @return created user
   * @throws ServiceOperationException
   *      if the username of the new user already
       exists
   */
  UserDTO createNewUser( UserDTO newUserDto )
       throws ServiceOperationException;


  /**
   * Retrieves the user with the given username.
   *
   * @param username
   * @return found user, {@code null} if user
       does not exist
   */
  UserDTO findByUsername( String username )
       throws ResourceNotFoundException;


  /**
   * Returns a list of all the existing users.
   *
   * @return list of all users
   */
  List< UserDTO > getAllUsers();


  /**
   * Updates the details of a user.
   *
   * @param updatedUser
   * @return updated user
   * @throws ResourceNotFoundException
   *      if the user does not exist
   * @throws ServiceOperationException
   *      if the user is deactivated, or when the
        new username to assign already exists
       in the database (usernames
   *      are unique)
   */
  UserDTO updateUser( UserDTO updatedUser )
       throws ResourceNotFoundException,
       ServiceOperationException;


  /**
   * Returns the password of the user
   *
   * @param username
   * @return hashed password
   * @throws ResourceNotFoundException
   *      if the user does not exist
   */
  String getPassword( String username ) throws
       ResourceNotFoundException;


  /**
   * Activates the user with the given username.
   *
   * @param username
   * @throws ResourceNotFoundException
   *      if the user does not exist
   * @throws ServiceOperationException
   *      if the user is already activated
   */
  void activateUser( String username ) throws
       ResourceNotFoundException,
       ServiceOperationException;


  /**
   * Deactivates the user with the given
       username.
   *
   * @param username
   * @throws ResourceNotFoundException
   *      if the user does not exist
   * @throws ServiceOperationException
   *      if the user is already deactivated
   */
  void deactivateUser( String username ) throws
       ResourceNotFoundException,
       ServiceOperationException;


  /**
   * Deletes the user with the given username.
   *
   * @param username
   * @throws ResourceNotFoundException
   *      if user does not exist
```

```
    */
   void deleteUser( String username ) throws
        ResourceNotFoundException;
}
```

30. SurveyService.java

```
package com.hdsgs.service;

import java.util.List;

import com.hdsgs.dto.QuestionResults;
import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.enums.UserRole;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;

/**
 * Caters survey services.
 */
public interface SurveyService
{
   /**
    * Creates a new survey.
    *
    * @param newSurvey
    * @return created survey
    * @throws ServiceOperationException
    *       if the creator does not exist or is
        deactivated
    */
   SurveyDTO createNewSurvey( SurveyDTO newSurvey
        ) throws ServiceOperationException;


   /**
    * Retrieves the survey with the given id.
    *
    * @param surveyId
    * @return survey with the given id
    * @throws ResourceNotFoundException
    *       if the survey does not exist
    */
   SurveyDTO findById( int surveyId ) throws
        ResourceNotFoundException;


   /**
    * Returns a list of surveys that ended
        yesterday. Note that a survey is active
        from the start to the day before the
    * end dates.
    *
    * @return list of surveys
    */
   List< SurveyDTO > getEndedYesterday();


   /**
    * Returns a list of all the existing surveys.
    *
    * @return list of surveys
    */
   List< SurveyDTO > getAllSurveys();


   /**
    * Returns a list of surveys associated with
        the user.
    *
    * @param username
    * @return a list of answered surveys if the
        user specified is a {@link UserRole#
        CONTRIBUTOR contributor}, a list of
    *       created surveys if the user specified
        is a {@link UserRole#RESEARCH_LEADER
        research leader}
    * @throws ResourceNotFoundException
    *       if the user does not exist
    */
   List< SurveyDTO > getSurveys( String username
        ) throws ResourceNotFoundException;


   /**
    * Returns the survey to be updated.
    *
    * @param surveyId
    * @return survey to update
    * @throws ResourceNotFoundException
    *       if the survey does not exist
    * @throws ServiceOperationException
    *       if the survey is not active
    */
   SurveyDTO getSurveyToUpdate( int surveyId )
        throws ResourceNotFoundException,
        ServiceOperationException;


   /**
    * Updates the details of a user.
    *
    * @param updatedSurvey
    * @return updated user
    * @throws ResourceNotFoundException
    *       if the survey does not exist
    * @throws ServiceOperationException
    *       if the creator is deactivated
    */
   SurveyDTO updateSurvey( SurveyDTO
        updatedSurvey ) throws
        ResourceNotFoundException,
        ServiceOperationException;


   /**
    * Returns the survey to be answered.
    *
    * @param surveyId
    * @param contributor
    * @return survey to answer
    * @throws ResourceNotFoundException
    *       if the survey does not exist
    * @throws ServiceOperationException
    *       if the survey is not active, or
        contributor is deactivated
    */
   SurveyDTO getSurveyToAnswer( int surveyId,
        String contributor ) throws
        ResourceNotFoundException,
        ServiceOperationException;


   /**
    * Inserts the survey answers to Sharemind's
        database.
    *
    * @param surveyId
    * @param answers
    * @param contributor
    */
   void recordAnswers( int surveyId, List< String
        > answers, String contributor ) throws
        ResourceNotFoundException,
        ServiceOperationException;


   /**
    * Computes for the results of a survey
        through SMC in collaboration with the
        Sharemind framework.
    *
    * @param surveyId
    * @return list of results per question
    * @throws ResourceNotFoundException
    *       if survey does not exist
    * @throws ServiceOperationException
    *       if survey has not yet started or is on-
        going
    */
   List< QuestionResults > computeResults( int
        surveyId ) throws
        ResourceNotFoundException,
        ServiceOperationException;


   /**
    * Returns the total number of contributors
        who answered the survey.
    *
    * @param surveyId
    * @return total number of contributors
    */
   public int getTotalContributors( int surveyId
        );


   /**
    * Deactivates the user with the given id.
```

```
     *
     * @param surveyId
     * @throws ResourceNotFoundException
     *      if the survey does not exist
     * @throws ServiceOperationException
     *      if the survey is already deactivated
     */
    void deactivateSurvey( int surveyId ) throws
        ResourceNotFoundException,
        ServiceOperationException;


    /**
     * Deletes the survey with the given id.
     *
     * @param surveyId
     * @throws ResourceNotFoundException
     *      if the survey does not exist
     */
    void deleteSurvey( int surveyId ) throws
        ResourceNotFoundException;
}
```

31. ReportService.java

```
package com.hdsgs.service;

import java.io.OutputStream;

import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;

/**
 * Caters report services.
 */
public interface ReportService
{
    /**
     * Generates the PDF of the survey results.
     *
     * @param survey
     * @param outputStream
     */
    void generateReport( SurveyDTO survey,
        OutputStream outputStream ) throws
        ResourceNotFoundException,
        ServiceOperationException;
}
```

32. UserServiceImpl.java

```
package com.hdsgs.service.impl;

import static com.hdsgs.enums.Error.
    USERNAME_EXISTS;
import static com.hdsgs.enums.Error.
    USER_ACTIVATED;
import static com.hdsgs.enums.Error.
    USER_DEACTIVATED;
import static com.hdsgs.enums.Error.USER_DNE;

import java.util.List;
import java.util.stream.Collectors;

import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.security.
    authentication.
    UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context
    .SecurityContextHolder;
import org.springframework.security.core.
    userdetails.UserDetails;
import org.springframework.security.core.
    userdetails.UserDetailsService;
import org.springframework.security.crypto.
    bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.
    annotation.Transactional;

import com.hdsgs.dto.UserDTO;
import com.hdsgs.entity.User;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;
import com.hdsgs.repository.UserRepository;
import com.hdsgs.service.BaseService;
import com.hdsgs.service.UserService;
import com.hdsgs.utility.Mapper;

/**
 * Implementation of the {@link UserService}
     interface.
 */
@Service
@Transactional
public class UserServiceImpl extends BaseService
    implements UserService
{
  @Autowired
  private UserRepository userRepository;

  @Autowired
  private UserDetailsService userDetailsService;

  @Autowired
  private CurrentUserService currentUserService;


  @Override
  public UserDTO createNewUser( UserDTO
      newUserDto ) throws
      ServiceOperationException
  {
    if( userRepository.findByUsername(
        newUserDto.getUsername() ).isPresent()
        )
    {
      throw new ServiceOperationException(
          USERNAME_EXISTS );
    }

    String password = new BCryptPasswordEncoder
        ().encode( newUserDto.getPassword() );

    User newUser = new User( newUserDto.
        getUsername(), password, newUserDto.
        getName(), newUserDto.getDesignation(),
         newUserDto.getInstitution(),
        newUserDto.getEmail(), newUserDto.
            getRole(), newUserDto.isActive() );

    UserDTO created = Mapper.toDto(
        userRepository.save( newUser ) );

    logger.debug( "Created account of " +
        created.getUsername() );

    return created;
  }


  @Override
  public UserDTO findByUsername( String username
      ) throws ResourceNotFoundException
  {
    UserDTO found = userRepository.
        findByUsername( username ).map( Mapper
        ::toDto ).orElseThrow( ( ) -> new
        ResourceNotFoundException( USER_DNE ) )
        ;

    logger.debug( "Found account of " + found.
        getUsername() );

    return found;
  }


  @Override
  public String getPassword( String username )
      throws ResourceNotFoundException
  {
    User found = userRepository.findByUsername(
        username ).orElseThrow( ( ) -> new
        ResourceNotFoundException( USER_DNE ) )
        ;

    logger.debug( "Retrieved password of " +
        found.getUsername() );

    return found.getPassword();
  }
```

```
@Override
public List< UserDTO > getAllUsers()
{
  List< UserDTO > users = userRepository.
       findAll().stream().map( Mapper::toDto )
       .collect( Collectors.toList() );

  logger.debug( "Found " + users.size() + "
       users" );

  return users;
}


@Override
public UserDTO updateUser( UserDTO
     updatedUserDto ) throws
     ResourceNotFoundException,
     ServiceOperationException
{
  User userToUpdate = userRepository.findById(
       updatedUserDto.getUserId() ).
       orElseThrow( ( ) -> new
       ResourceNotFoundException( USER_DNE ) )
       ;
  String oldUsername = userToUpdate.
       getUsername();

  if( !userToUpdate.isActive() )
  {
    throw new ServiceOperationException(
         USER_DEACTIVATED );
  }

  User existingUser = userRepository.
       findByUsername( updatedUserDto.
       getUsername() ).orElse( null );

  if( existingUser != null && userToUpdate.
       getId() != existingUser.getId() )
  {
    throw new ServiceOperationException(
         USERNAME_EXISTS );
  }

  if( updatedUserDto.getPassword() == null )
       // from update account form, else from
       change password form
  {
    userToUpdate.setUsername( updatedUserDto.
         getUsername() );
    userToUpdate.setName( updatedUserDto.
         getName() );
    userToUpdate.setDesignation(
         updatedUserDto.getDesignation() );
    userToUpdate.setInstitution(
         updatedUserDto.getInstitution() );
    userToUpdate.setEmail( updatedUserDto.
         getEmail() );
  }
  else
  {
    userToUpdate.setPassword( new
         BCryptPasswordEncoder().encode(
         updatedUserDto.getPassword() ) );
  }

  UserDTO updated = Mapper.toDto(
       userRepository.save( userToUpdate ) );

  if( currentUserService.getCurrentUser().
       getUsername().equals( oldUsername ) )
  {
    reauthenticate( updated.getUsername() );
  }

  logger.debug( "Updated account of " +
       updated.getUsername() );

  return updated;
}


/**
 * Re-authenticates user after updating to
      reflect possible changes in
 * username or password.
 *
 * @param username
 */
private void reauthenticate( String username )
{
  UserDetails userDetails = userDetailsService
       .loadUserByUsername( username );
  SecurityContextHolder.getContext().
       setAuthentication(
      new UsernamePasswordAuthenticationToken(
           userDetails, userDetails.
           getPassword(), userDetails.
           getAuthorities() ) );

  logger.debug( "Re-authenticated user" );
}


@Override
public void activateUser( String username )
     throws ResourceNotFoundException,
     ServiceOperationException
{
  toggleUserStatus( username, true );

  if( currentUserService.getCurrentUser().
       getUsername().equals( username ) )
  {
    reauthenticate( username );
  }

  logger.debug( "Activated account of " +
       username );
}


@Override
public void deactivateUser( String username )
     throws ResourceNotFoundException,
     ServiceOperationException
{
  toggleUserStatus( username, false );

  if( currentUserService.getCurrentUser().
       getUsername().equals( username ) )
  {
    reauthenticate( username );
  }

  logger.debug( "Deactivated account of " +
       username );
}


private void toggleUserStatus( String username
     , boolean isActive ) throws
     ResourceNotFoundException,
     ServiceOperationException
{
  User userDto = userRepository.findByUsername
       ( username ).orElseThrow( ( ) -> new
       ResourceNotFoundException( USER_DNE ) )
       ;

  if( userDto.isActive() == isActive )
  {
    if( isActive )
    {
      throw new ServiceOperationException(
           USER_ACTIVATED );
    }
    else
    {
      throw new ServiceOperationException(
           USER_DEACTIVATED );
    }
  }

  userDto.setActive( isActive );

  userRepository.save( userDto );
}


@Override
public void deleteUser( String username )
     throws ResourceNotFoundException
{
  User user = userRepository.findByUsername(
       username ).orElseThrow( ( ) -> new
       ResourceNotFoundException( USER_DNE ) )
       ;

  userRepository.delete( user );
```

```
    logger.debug( "Deleted account of " +
        username );
  }
}
```

33. SurveyServiceImpl.java

```
package com.hdsgs.service.impl;

import static com.hdsgs.enums.Error.
    ALREADY_ANSWERED;
import static com.hdsgs.enums.Error.
    SURVEY_DEACTIVATED;
import static com.hdsgs.enums.Error.SURVEY_DNE;
import static com.hdsgs.enums.Error.
    SURVEY_NOT_STARTED;
import static com.hdsgs.enums.Error.
    SURVEY_ONGOING;
import static com.hdsgs.enums.Error.
    USER_DEACTIVATED;
import static com.hdsgs.enums.Error.USER_DNE;
import static com.hdsgs.enums.Function.AVERAGE;
import static com.hdsgs.enums.Function.MAXIMUM;
import static com.hdsgs.enums.Function.MINIMUM;
import static com.hdsgs.enums.Function.TALLY;
import static com.hdsgs.enums.InputType.NUMBER;
import static com.hdsgs.enums.UserRole.
    CONTRIBUTOR;
import static com.hdsgs.enums.UserRole.
    RESEARCH_LEADER;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import org.joda.time.LocalDate;
import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.
    annotation.Transactional;

import com.hdsgs.constants.SharemindController;
import com.hdsgs.dto.QuestionDTO;
import com.hdsgs.dto.QuestionResults;
import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.entity.BasicStats;
import com.hdsgs.entity.Question;
import com.hdsgs.entity.QuestionChoice;
import com.hdsgs.entity.QuestionFunction;
import com.hdsgs.entity.Survey;
import com.hdsgs.entity.Tally;
import com.hdsgs.entity.User;
import com.hdsgs.entity.UserAnsweredSurvey;
import com.hdsgs.enums.Function;
import com.hdsgs.enums.UserRole;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;
import com.hdsgs.repository.
    QuestionChoiceRepository;
import com.hdsgs.repository.
    QuestionFunctionRepository;
import com.hdsgs.repository.QuestionRepository;
import com.hdsgs.repository.SurveyRepository;
import com.hdsgs.repository.
    UserAnsweredSurveyRepository;
import com.hdsgs.repository.UserRepository;
import com.hdsgs.service.BaseService;
import com.hdsgs.service.SurveyService;
import com.hdsgs.utility.Mapper;

/**
 * Implementation of the {@link SurveyService}
     interface.
 */
@Service
@Transactional
public class SurveyServiceImpl extends
    BaseService implements SurveyService
{
  @Autowired
  private SurveyRepository surveyRepository;

  @Autowired
  private UserRepository userRepository;

  @Autowired
  private UserAnsweredSurveyRepository
      userAnsweredSurveyRepository;

  @Autowired
  private QuestionRepository questionRepository;

  @Autowired
  private QuestionFunctionRepository
      questionFunctionRepository;

  @Autowired
  private QuestionChoiceRepository
      questionChoiceRepository;

  private final int offsetDueToDatePrefix = 10;


  @Override
  public SurveyDTO createNewSurvey( SurveyDTO
      newSurveyDto ) throws
      ServiceOperationException
  {
    User creator = userRepository.findByUsername
        ( newSurveyDto.getCreator() ).
        orElseThrow( ( ) -> new
        ServiceOperationException( USER_DNE ) )
        ;

    if( !creator.isActive() )
    {
      throw new ServiceOperationException(
          USER_DEACTIVATED );
    }

    Survey newSurvey = new Survey( creator,
        newSurveyDto.getTitle(), newSurveyDto.
        getStartDate(), newSurveyDto.getEndDate
        (),
        newSurveyDto.getDescription() );

    List< Question > surveyQuestions = new
        ArrayList< Question >();
    int i = 1;
    for( QuestionDTO newQuestionDto :
        newSurveyDto.getQuestions() )
    {
      if( newQuestionDto.getQuestion().trim().
          length() != 0 )
      {
        newQuestionDto.setQuestionNumber( i++ );
        surveyQuestions.add( createNewQuestion(
            newSurvey, newQuestionDto ) );
      }
    }
    newSurvey.populateQuestions( surveyQuestions
        );

    SurveyDTO created = Mapper.toDto(
        surveyRepository.save( newSurvey ) );

    logger.debug( "Created new survey: " +
        created.getTitle() );

    return created;
  }


  /**
   * Helper method that creates a new question
       for a survey.
   *
   * @param survey
   * @param question
   * @return question entity
   */
  private Question createNewQuestion( Survey
      survey, QuestionDTO question )
  {
    Question newQuestion = new Question( survey,
        question.getQuestionNumber(), question
        .getQuestion(), question.getInputType()
        );

    if( !question.getInputType().equals( NUMBER
        ) )
    {
```

```
    List< QuestionChoice > choices = new
        ArrayList< QuestionChoice >();
    for( String choice : question.getChoices()
        )
    {
      if( choice.trim().length() != 0 )
      {
        choices.add( new QuestionChoice(
            newQuestion, choice ) );
      }
    }
    newQuestion.populateChoices( choices );
  }

  List< QuestionFunction > functions = new
      ArrayList< QuestionFunction >();
  for( Function fxn : question.getFunctions()
      )
  {
    functions.add( new QuestionFunction(
        newQuestion, fxn ) );
  }
  newQuestion.populateFunctions( functions );

  return newQuestion;
}


@Override
public SurveyDTO findById( int surveyId )
    throws ResourceNotFoundException
{
  SurveyDTO found = surveyRepository.findById(
      surveyId ).map( Mapper::toDto ).
      orElseThrow( ( ) -> new
      ResourceNotFoundException( SURVEY_DNE )
      );

  logger.debug( "Found survey: " + found.
      getTitle() );

  return found;
}


@Override
public List< SurveyDTO > getEndedYesterday()
{
  List< SurveyDTO > surveys = surveyRepository
      .findByEndDate( LocalDate.now().
      minusDays( 1 ) ).stream().map( Mapper::
      toDto )
      .collect( Collectors.toList() );

  logger.debug( "Found " + surveys.size() + "
      surveys that ended yesterday" );

  return surveys;
}


@Override
public List< SurveyDTO > getAllSurveys()
{
  List< SurveyDTO > surveys = surveyRepository
      .findAll().stream().map( Mapper::toDto
      ).collect( Collectors.toList() );

  logger.debug( "Found " + surveys.size() + "
      surveys" );

  return surveys;
}


@Override
public List< SurveyDTO > getSurveys( String
    username ) throws
    ResourceNotFoundException
{
  User user = userRepository.findByUsername(
      username ).orElseThrow( ( ) -> new
      ResourceNotFoundException( USER_DNE ) )
      ;

  UserRole role = user.getRole();
  List< Survey > surveys = new ArrayList<
      Survey >();

  if( role == RESEARCH_LEADER )
  {
    surveys = surveyRepository.
        findByCreatorUsername( username );
  }
  else if( role == CONTRIBUTOR )
  {
    List< UserAnsweredSurvey > records =
        userAnsweredSurveyRepository.
        findByUserUsername( username );

    surveys = records.stream().map(
        UserAnsweredSurvey::getSurvey ).
        collect( Collectors.toList() );
  }

  List< SurveyDTO > surveyDtos = surveys.
      stream().map( Mapper::toDto ).collect(
      Collectors.toList() );

  logger.debug( "Found " + surveyDtos.size() +
      " surveys associated with " + username
      );

  return surveyDtos;
}


@Override
public SurveyDTO getSurveyToUpdate( int
    surveyId ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  Survey survey = surveyRepository.findById(
      surveyId ).orElseThrow( ( ) -> new
      ResourceNotFoundException( SURVEY_DNE )
      );

  if( survey.active() )
  {
    throw new ServiceOperationException(
        SURVEY_ONGOING );
  }
  else if( LocalDate.now().isAfter( survey.
      getEndDate() ) )
  {
    throw new ServiceOperationException(
        SURVEY_DEACTIVATED );
  }

  SurveyDTO surveyToUpdate = Mapper.toDto(
      survey );

  logger.debug( "Survey to update: " +
      surveyToUpdate.getTitle() );

  return surveyToUpdate;
}


@Override
public SurveyDTO updateSurvey( SurveyDTO
    updatedSurveyDto ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  Survey surveyToUpdate = surveyRepository.
      findById( updatedSurveyDto.getSurveyId
      () ).orElseThrow(
      ( ) -> new ResourceNotFoundException(
          SURVEY_DNE ) );

  if( !userRepository.findByUsername(
      surveyToUpdate.getCreator().getUsername
      () ).get().isActive() )
  {
    throw new ServiceOperationException(
        USER_DEACTIVATED );
  }

  List< Question > questions = new ArrayList<
      Question >();
  for( QuestionDTO q : updatedSurveyDto.
      getQuestions() )
  {
    questions.add( updateQuestion(
        surveyToUpdate, q ) );
  }

  surveyToUpdate.setTitle( updatedSurveyDto.
      getTitle() );
  surveyToUpdate.setStartDate(
      updatedSurveyDto.getStartDate() );
  surveyToUpdate.setEndDate( updatedSurveyDto.
```

```
        getEndDate() );
    surveyToUpdate.populateQuestions( questions
        );

    SurveyDTO updated = Mapper.toDto(
        surveyRepository.save( surveyToUpdate )
         );

    logger.debug( "Updated survey: " + updated.
        getTitle() );

    return updated;
  }


  /**
   * Helper method that creates or updates a
       question of a survey to update.
   *
   * @param surveyToUpdate
   * @param question
   * @return question entity
   */
  private Question updateQuestion( Survey
      surveyToUpdate, QuestionDTO question )
  {
    int questionId = question.getId();
    Question questionToUpdate =
        questionRepository.findOne( questionId
        );

    if( questionToUpdate != null )
    {
      List< QuestionFunction >
          updatedQuestionFunctions = new
          ArrayList< QuestionFunction >();
      for( Function fxn : question.getFunctions
          () )
      {
        QuestionFunction qf =
            questionFunctionRepository.
            findByQuestionIdAndFunctionToCompute
            ( questionId, fxn );

        if( qf != null )
        {
          updatedQuestionFunctions.add( qf );
        }
        else
        {
          updatedQuestionFunctions.add( new
              QuestionFunction(
              questionToUpdate, fxn ) );
        }

      }

      List< QuestionChoice >
          updatedQuestionChoices = new
          ArrayList< QuestionChoice >();
      for( String choice : question.getChoices()
          )
      {
        if( choice.trim().length() != 0 )
        {
          QuestionChoice qc =
              questionChoiceRepository.
              findByQuestionIdAndOption(
              questionId, choice );

          if( qc != null )
          {
            updatedQuestionChoices.add( qc );
          }
          else
          {
            updatedQuestionChoices.add( new
                QuestionChoice(
                questionToUpdate, choice ) );
          }
        }
      }

      questionToUpdate.setInputType( question.
          getInputType() );
      questionToUpdate.setQuestion( question.
          getQuestion() );
      questionToUpdate.populateFunctions(
          updatedQuestionFunctions );
      questionToUpdate.populateChoices(
          updatedQuestionChoices );
    }
    else
    {
      questionToUpdate = createNewQuestion(
          surveyToUpdate, question );
    }

    return questionToUpdate;
  }


  @Override
  public SurveyDTO getSurveyToAnswer( int
      surveyId, String contributor ) throws
      ResourceNotFoundException,
      ServiceOperationException
  {
    Survey survey = surveyRepository.findById(
        surveyId ).orElseThrow( ( ) -> new
        ResourceNotFoundException( SURVEY_DNE )
         );

    if( !survey.active() )
    {
      throw new ServiceOperationException(
          SURVEY_DEACTIVATED );
    }

    Optional< UserAnsweredSurvey > record =
        userAnsweredSurveyRepository.
        findBySurveyIdAndUserUsername( surveyId
        , contributor );

    if( record.isPresent() )
    {
      throw new ServiceOperationException(
          ALREADY_ANSWERED );
    }

    SurveyDTO surveyToAnswer = Mapper.toDto(
        survey );

    logger.debug( "Survey to answer: " +
        surveyToAnswer.getTitle() );

    return surveyToAnswer;
  }


  @Override
  public void recordAnswers( int surveyId, List<
      String > answers, String contributor )
      throws ResourceNotFoundException,
      ServiceOperationException
  {
    List< Integer > questionNumbers = new
        ArrayList< Integer >();
    List< String > submittedAnswers = new
        ArrayList< String >();

    //parse; for checkbox answers
    for( int qNum = 1; qNum <= answers.size();
        qNum++ )
    {
      if( answers.get( qNum - 1 ) == null ||
          answers.get( qNum - 1 ).isEmpty() )
      {
        continue;
      }

      List< String > ans = Arrays.asList(
          answers.get( qNum - 1 ).split( "," )
          );
      submittedAnswers.addAll( ans );

      for( int i = 0; i < ans.size(); i++ )
      {
        questionNumbers.add( qNum );
      }
    }

    ProcessBuilder pb = null;
    for( int i = 0; i < submittedAnswers.size();
        i++ )
    {
      pb = new ProcessBuilder(
          SharemindController.INSERT,
          submittedAnswers.get( i ),
          questionNumbers.get( i ).toString(),
          String.valueOf( surveyId ) );
      pb.directory( new File(
          SharemindController.DIRECTORY ) );
```

```
    try
    {
      Process p = pb.start();
      p.waitFor();
    }
    catch( InterruptedException e )
    {
      logger.error( e.getMessage() );

      return;
    }
    catch( IOException e )
    {
      logger.error( e.getMessage() );

      return;
    }
  }

  recordAnsweredSurvey( contributor, surveyId
      );

  logger.debug( "Recorded survey answers of "
      + contributor + " to survey " +
      surveyId );
}


/**
 * Records that a contributor answered a
     survey.
 *
 * @param username
 *     of contributor
 * @param surveyId
 * @throws ResourceNotFoundException
 *     if contributor or survey does not exist
 * @throws ServiceOperationException
 *     if contributor or survey is deactivated
 */
private void recordAnsweredSurvey( String
    username, int surveyId ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  Survey survey = surveyRepository.findById(
      surveyId ).orElseThrow( ( ) -> new
      ResourceNotFoundException( SURVEY_DNE )
      );

  if( !survey.active() )
  {
    throw new ServiceOperationException(
        SURVEY_DEACTIVATED );
  }

  User contributor = userRepository.
      findByUsername( username ).orElseThrow(
      ( ) -> new ResourceNotFoundException(
      USER_DNE ) );

  if( !contributor.isActive() )
  {
    throw new ServiceOperationException(
        USER_DEACTIVATED );
  }

  UserAnsweredSurvey record = new
      UserAnsweredSurvey( survey, contributor
      );

  userAnsweredSurveyRepository.save( record );
}


@Override
public List< QuestionResults > computeResults(
    int surveyId ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  Survey survey = surveyRepository.findById(
      surveyId ).orElseThrow( ( ) -> new
      ResourceNotFoundException( SURVEY_DNE )
      );

  if( LocalDate.now().isBefore( survey.
      getStartDate() ) )
  {
    throw new ServiceOperationException(
        SURVEY_NOT_STARTED );
  }
  else if( survey.active() )
  {
    throw new ServiceOperationException(
        SURVEY_ONGOING );
  }

  List< QuestionResults > results = new
      ArrayList< QuestionResults >();

  List< BasicStats > basicStats = survey.
      getBasicStats();
  List< Tally > tally = survey.getTally();

  if( basicStats.size() > 0 || tally.size() >
      0 )
  {
    results = Mapper.toResults( survey.
        getQuestions(), basicStats, tally );
  }
  else
  {
    results = saveResults( survey );
  }

  logger.debug( "Computed results for survey:
      " + survey.getTitle() );

  return results;
}


/**
 * Computes the desired results per question
     and saves them to the database.
 *
 * @param survey
 * @return list of saved question results
 */
private List< QuestionResults > saveResults(
    Survey survey )
{
  List< BasicStats > basicStats = new
      ArrayList< BasicStats >();
  List< Tally > tallyList = new ArrayList<
      Tally >();

  int surveyId = survey.getId();

  for( Question question : survey.getQuestions
      () )
  {
    int questionNumber = question.
        getQuestionNumber();

    boolean hasBasicStats = false;
    double average = -1;
    int minimum = -1;
    int maximum = -1;

    String choice = null;
    int count = -1;

    for( QuestionFunction qf : question.
        getFunctions() )
    {
      Function fxn = qf.getFunctionToCompute()
          ;

      if( fxn == AVERAGE )
      {
        hasBasicStats = true;
        average = getStatistic( surveyId,
            questionNumber,
            SharemindController.AVERAGE,
            SharemindController.
            AVERAGE_MARKER );
      }
      else if( fxn == MINIMUM )
      {
        hasBasicStats = true;
        minimum = ( int ) getStatistic(
            surveyId, questionNumber,
            SharemindController.MINIMUM,
            SharemindController.
            MINIMUM_MARKER );
      }
      else if( fxn == MAXIMUM )
      {
        hasBasicStats = true;
        maximum = ( int ) getStatistic(
            surveyId, questionNumber,
            SharemindController.MAXIMUM,
```

```
                SharemindController.
                MAXIMUM_MARKER );
        }
        else if( fxn == TALLY )
        {
          List< QuestionChoice > choices =
              question.getChoices();

          for( int i = 0; i < choices.size(); i
              ++ )
          {
            choice = choices.get( i ).getOption
                ();
            count = getCount( surveyId,
                questionNumber, ( i + 1 ) );

            tallyList.add( new Tally( survey,
                questionNumber, choice, count )
                );
          }
        }
      }

      if( hasBasicStats )
      {
        basicStats.add( new BasicStats( survey,
            questionNumber, average, minimum,
            maximum ) );
      }
    }

    survey.populateBasicStats( basicStats );
    survey.populateTally( tallyList );

    Survey updated = surveyRepository.save(
        survey );

    return Mapper.toResults( survey.getQuestions
        (), updated.getBasicStats(), updated.
        getTally() );
  }


  /**
   * Calls the appropriate Sharemind controller
       to compute for the needed
   * statistic (average, minimum, or maximum).
   *
   * @param surveyId
   * @param questionNumber
   * @param controllerName
   * @param marker
   * @return average, minimum, or maximum answer
       for a question
   */
  private double getStatistic( int surveyId, int
      questionNumber, String controllerName,
      String marker )
  {
    double statistic = 0.0;

    ProcessBuilder pb = null;
    Process p = null;
    BufferedReader reader = null;
    String line = "";

    pb = new ProcessBuilder( controllerName,
        String.valueOf( surveyId ), String.
        valueOf( questionNumber ) );
    pb.directory( new File( SharemindController.
        DIRECTORY ) );

    try
    {
      p = pb.start();

      reader = new BufferedReader( new
          InputStreamReader( p.getInputStream()
          ) );

      while( ( line = reader.readLine() ) !=
          null )
      {
        if( line.contains( marker ) )
        {
          statistic = Double.parseDouble( line.
              substring( marker.length() +
              offsetDueToDatePrefix ) );
        }
      }

      p.waitFor();
    }
    catch( IOException e )
    {
      logger.error( e.getMessage() );

      return -1;
    }
    catch( InterruptedException e )
    {
      logger.error( e.getMessage() );

      return -1;
    }

    return statistic;
  }


  /**
   * Calls the appropriate Sharemind controller
       to compute for the number of
   * times a certain question choice was
       answered.
   *
   * @param surveyId
   * @param questionNumber
   * @param choices
   * @return tally of answers for a question
   * @throws InterruptedException
   * @throws IOException
   */
  private int getCount( int surveyId, int
      questionNumber, int choiceValue )
  {
    int count = 0;

    ProcessBuilder pb = null;
    Process p = null;
    BufferedReader reader = null;
    String line = "";

    pb = new ProcessBuilder( SharemindController
        .TALLY, String.valueOf( surveyId ),
        String.valueOf( questionNumber ),
        String.valueOf( choiceValue ) );
    pb.directory( new File( SharemindController.
        DIRECTORY ) );

    try
    {
      p = pb.start();

      reader = new BufferedReader( new
          InputStreamReader( p.getInputStream()
          ) );

      while( ( line = reader.readLine() ) !=
          null )
      {
        if( line.contains( SharemindController.
            TALLY_MARKER ) )
        {
          count = Integer.parseInt( line.
              substring( SharemindController.
              TALLY_MARKER.length() +
              offsetDueToDatePrefix ) );
        }
      }

      p.waitFor();
    }
    catch( IOException e )
    {
      logger.error( e.getMessage() );

      return -1;
    }
    catch( InterruptedException e )
    {
      logger.error( e.getMessage() );

      return -1;
    }

    return count;
  }


  @Override
  public int getTotalContributors( int surveyId
      )
  {
```

```
    int totalContributors =
        userAnsweredSurveyRepository.
        countBySurveyId( surveyId );

    logger.debug( "Survey " + surveyId + " has "
        + totalContributors + " contributors"
        );

    return totalContributors;
  }


  @Override
  public void deactivateSurvey( int surveyId )
      throws ResourceNotFoundException,
      ServiceOperationException
  {
    Survey surveyToDeactivate = surveyRepository
        .findById( surveyId ).orElseThrow( ( )
        -> new ResourceNotFoundException(
        SURVEY_DNE ) );

    if( !surveyToDeactivate.active() )
    {
      throw new ServiceOperationException(
          SURVEY_DEACTIVATED );
    }

    surveyToDeactivate.setEndDate( LocalDate.now
        () );

    surveyRepository.save( surveyToDeactivate );

    logger.debug( "Deactivated survey " +
        surveyId );

    computeResults( surveyId );
  }


  @Override
  public void deleteSurvey( int surveyId )
      throws ResourceNotFoundException
  {
    Survey surveyToDelete = surveyRepository.
        findById( surveyId ).orElseThrow( ( )
        -> new ResourceNotFoundException(
        SURVEY_DNE ) );

    surveyRepository.delete( surveyToDelete );

    logger.debug( "Deleted survey " + surveyId )
        ;
  }
}
```

34. ReportServiceImpl.java

```
package com.hdsgs.service.impl;

import java.io.IOException;
import java.io.OutputStream;
import java.util.List;
import java.util.Map.Entry;

import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.stereotype.Service;

import com.hdsgs.dto.QuestionResults;
import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.dto.TallyDto;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;
import com.hdsgs.service.BaseService;
import com.hdsgs.service.ReportService;
import com.hdsgs.service.SurveyService;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.BaseFont;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;

/**
 * Implementation of the {@link ReportService}
     interface.
 */
@Service
public class ReportServiceImpl extends
    BaseService implements ReportService
{
  @Autowired
  private SurveyService surveyService;

  private Font titleFont;

  private Font boldFont;

  private Font font;

  private final Paragraph space = new Paragraph(
      20, " " );


  /**
   * Sets the values of the class variables.
   */
  public ReportServiceImpl()
  {
    try
    {
      BaseFont base = BaseFont.createFont( "/
          static/css/Junction-webfont.ttf",
          BaseFont.IDENTITY_H, true );

      titleFont = new Font( base, 16f );
      boldFont = new Font( base, 11f, Font.BOLD
          );
      font = new Font( base, 11f );
    }
    catch( DocumentException | IOException e )
    {
      logger.error( e.getMessage() );
    }

  }


  @Override
  public void generateReport( SurveyDTO survey,
      OutputStream outputStream ) throws
      ResourceNotFoundException,
      ServiceOperationException
  {
    logger.debug( "Creating report..." );

    int surveyId = survey.getSurveyId();
    List< QuestionResults > results =
        surveyService.computeResults( surveyId
        );

    Document document = new Document();

    try
    {
      PdfWriter writer = PdfWriter.getInstance(
          document, outputStream );

      document.open();

      // set attributes
      document.addAuthor( "HDSGS" );
      document.addCreator( "HDSGS" );
      document.addCreationDate();
      document.addTitle( survey.getTitle() + " -
          Results" );

      // start document
      document.add( new Paragraph( "Survey " +
          surveyId + " - " + survey.getTitle(),
          titleFont ) );
      document.add( new Paragraph( "Created by:
          " + survey.getInstitutionOfCreator(),
          font ) );
      document.add( new Paragraph( "Start Date:
          " + survey.getReadableStartDate(),
          font ) );
      document.add( new Paragraph( "End Date: "
          + survey.getReadableEndDate(), font )
          );
      document.add( space );
      document.add( new Paragraph( "Total
          Contributors: " + surveyService.
          getTotalContributors( surveyId ),
```

```
            boldFont ) );
        document.add( space );

        for( QuestionResults result : results )
        {
          document.add( new Paragraph( result.
              getQuestionNumber() + ". " + result
              .getQuestion(), font ) );

          if( result.getTally().getCounts().
              isEmpty() )
          {
            document.add( createBasicStatsTable(
                result ) );
          }
          else
          {
            document.add( createTallyTable( result
                .getTally() ) );
          }

          document.add( space );
        }

        document.close();
        writer.close();
      }
      catch( DocumentException e )
      {
        logger.error( e.getMessage() );
      }

      logger.debug( "Report created" );
    }


    /**
     * Creates the table for the tally of answers
         to be added to the PDF.
     *
     * @param tally
     * @return tally table
     */
    private Element createTallyTable( TallyDto
        tally ) throws DocumentException
    {
      PdfPTable tallyTable = new PdfPTable( 2 );
      float[] tallyColumnWidths = { 1f, 1f };

      tallyTable.setWidthPercentage( 50 );
      tallyTable.setHorizontalAlignment( Element.
          ALIGN_LEFT );
      tallyTable.setSpacingBefore( 10f );
      tallyTable.setSpacingAfter( 10f );
      tallyTable.setWidths( tallyColumnWidths );

      PdfPCell choiceHeader = new PdfPCell( new
          Phrase( "Choice", boldFont ) );
      choiceHeader.setHorizontalAlignment( Element
          .ALIGN_CENTER );

      PdfPCell countHeader = new PdfPCell( new
          Phrase( "Count", boldFont ) );
      countHeader.setHorizontalAlignment( Element.
          ALIGN_CENTER );

      tallyTable.addCell( choiceHeader );
      tallyTable.addCell( countHeader );

      for( Entry< String, Integer > entry : tally.
          getCounts().entrySet() )
      {
        PdfPCell choice = new PdfPCell( new Phrase
            ( entry.getKey(), font ) );
        PdfPCell count = new PdfPCell( new Phrase(
            String.valueOf( entry.getValue() ),
            font ) );

        tallyTable.addCell( choice );
        tallyTable.addCell( count );
      }

      tallyTable.addCell( new PdfPCell( new Phrase
          ( "Minimum", boldFont ) ) );
      tallyTable.addCell( new PdfPCell( new Phrase
          ( tally.getChoicesWithMinCount().
          toString(), font ) ) );

      tallyTable.addCell( new PdfPCell( new Phrase
          ( "Maximum", boldFont ) ) );
      tallyTable.addCell( new PdfPCell( new Phrase
          ( tally.getChoicesWithMaxCount().
              toString(), font ) ) );

      return tallyTable;
    }


    /**
     * Creates the table for the average, minimum,
          and maximum answers to be
     * added to the PDF.
     *
     * @param result
     * @return table for basic statistics
     */
    private PdfPTable createBasicStatsTable(
        QuestionResults result ) throws
        DocumentException
    {
      PdfPTable basicStatsTable = new PdfPTable( 2
          );
      float[] basicStatsColumnWidths = { 1f, 1f };

      basicStatsTable.setWidthPercentage( 50 );
      basicStatsTable.setHorizontalAlignment(
          Element.ALIGN_LEFT );
      basicStatsTable.setSpacingBefore( 10f );
      basicStatsTable.setSpacingAfter( 10f );
      basicStatsTable.setWidths(
          basicStatsColumnWidths );

      if( result.getAverage() >= 0 )
      {
        basicStatsTable.addCell( new PdfPCell( new
            Phrase( "Average", boldFont ) ) );
        basicStatsTable.addCell( new PdfPCell( new
            Phrase( String.valueOf( result.
            getAverage() ), font ) ) );
      }

      if( result.getMinimum() >= 0 )
      {
        basicStatsTable.addCell( new PdfPCell( new
            Phrase( "Minimum", boldFont ) ) );
        basicStatsTable.addCell( new PdfPCell( new
            Phrase( String.valueOf( result.
            getMinimum() ), font ) ) );
      }

      if( result.getMaximum() >= 0 )
      {
        basicStatsTable.addCell( new PdfPCell( new
            Phrase( "Maximum", boldFont ) ) );
        basicStatsTable.addCell( new PdfPCell( new
            Phrase( String.valueOf( result.
            getMaximum() ), font ) ) );
      }

      return basicStatsTable;
    }
}
```

35. LoginService.java

```
package com.hdsgs.service.impl;

import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.security.core.
    userdetails.UserDetailsService;
import org.springframework.security.core.
    userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import com.hdsgs.entity.CurrentUser;
import com.hdsgs.entity.User;
import com.hdsgs.repository.UserRepository;
import com.hdsgs.service.BaseService;

/**
 * Implementation of the {@link
     UserDetailsService} in Spring Security for
     authenticating login credentials.
 */
@Service
public class LoginService extends BaseService
    implements UserDetailsService
{
  @Autowired
  private UserRepository userRepository;
```

```
  @Override
  public CurrentUser loadUserByUsername( String
       username ) throws
       UsernameNotFoundException
  {
    logger.debug( "Authenticating user..." );

    User user = userRepository.findByUsername(
         username ).orElseThrow(
         ( ) -> new UsernameNotFoundException(
              String.format( "User with username
              = %s was not found", username ) ) )
              ;

    return new CurrentUser( user );
  }
}
```

36. CurrentUserService.java

```
package com.hdsgs.service.impl;

import org.springframework.beans.factory.
     annotation.Autowired;
import org.springframework.security.core.context
     .SecurityContextHolder;
import org.springframework.stereotype.Service;

import com.hdsgs.entity.CurrentUser;
import com.hdsgs.repository.SurveyRepository;
import com.hdsgs.service.BaseService;

/**
 * Caters services for the current user.
 */
@Service
public class CurrentUserService extends
     BaseService
{
  @Autowired
  private SurveyRepository surveyRepository;


  /**
   * @param username
   * @param surveyId
   *
   * @return is current user the creator of the
        survey
   */
  public boolean isSurveyCreator( String
       username, int surveyId )
  {
    boolean isSurveyCreator = surveyRepository.
         getOne( surveyId ).getCreator().
         getUsername().equals( username );

    logger.debug( username + ( isSurveyCreator ?
         " is " : " is not " ) + " the creator
         of survey " + surveyId );

    return isSurveyCreator;
  }


  /**
   * Returns the currently logged in user.
   *
   * @return current user
   */
  public CurrentUser getCurrentUser()
  {
    return ( CurrentUser ) SecurityContextHolder
         .getContext().getAuthentication().
         getPrincipal();
  }
}
```

37. BaseController.java

```
package com.hdsgs.controller;

import static org.springframework.hateoas.mvc.
     ControllerLinkBuilder.linkTo;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.hateoas.Link;
import org.springframework.hateoas.
     ResourceSupport;
import org.springframework.http.HttpStatus;
import org.springframework.security.access.
     AccessDeniedException;
import org.springframework.security.core.
     Authentication;
import org.springframework.web.bind.annotation.
     ExceptionHandler;
import org.springframework.web.bind.annotation.
     ModelAttribute;
import org.springframework.web.bind.annotation.
     ResponseStatus;
import org.springframework.web.servlet.
     ModelAndView;

import com.hdsgs.entity.CurrentUser;
import com.hdsgs.enums.Error;
import com.hdsgs.exception.
     ResourceNotFoundException;
import com.hdsgs.exception.
     ServiceOperationException;

/**
 * Base controller with common model attributes
      for all controllers, logger, and
 * methods for exception handling.
 */
public abstract class BaseController
{
  protected final Logger logger = LoggerFactory.
       getLogger( getClass() );

  private final String ERROR_VIEW_NAME = "error"
       ;


  /**
   * Returns the model attribute for navigation
        links.
   *
   * @param authentication
   * @return navigation links
   */
  @ModelAttribute( "navigation" )
  public ResourceSupport getNavbarLinks(
       Authentication authentication )
  {
    Link selfLink = linkTo( HdsgsController.
         class ).withSelfRel();
    Link login = linkTo( HdsgsController.class )
         .slash( "/login" ).withRel( "login" );
    Link logout = linkTo( HdsgsController.class
         ).slash( "/logout" ).withRel( "logout"
         );
    Link usersLink = linkTo( UserController.
         class ).withRel( "users" );
    Link surveysLink = linkTo( SurveyController.
         class ).withRel( "surveys" );

    ResourceSupport navigation = new
         ResourceSupport();
    navigation.add( selfLink );
    navigation.add( login );
    navigation.add( logout );
    navigation.add( usersLink );
    navigation.add( surveysLink );

    if( authentication != null )
    {
      CurrentUser currentUser = ( CurrentUser )
           authentication.getPrincipal();

      Link currentUserPage = linkTo(
           UserController.class ).slash(
           currentUser.getUsername() ).withRel(
           "profile" );
      Link updateCurrentUserPage = linkTo(
           UserController.class ).slash(
           currentUser.getUsername() ).slash( "
           update" )
          .withRel( "updateProfile" );

      navigation.add( currentUserPage );
      navigation.add( updateCurrentUserPage );
    }

    return navigation;
```

```
}


/**
 * Returns the model attribute for the
      currently logged in user.
 *
 * @param authentication
 * @return
 */
@ModelAttribute( "currentUser" )
public CurrentUser getCurrentUser(
     Authentication authentication )
{
  return ( authentication == null ) ? null : (
       CurrentUser ) authentication.
      getPrincipal();
}


/**
 * Returns a ModelAndView of the custom 401
      error page.
 *
 * @param e
 * @param authentication
 * @return error 401 page
 */
@ResponseStatus( HttpStatus.UNAUTHORIZED )
@ExceptionHandler( AccessDeniedException.class
     )
public ModelAndView handleUnauthorized(
     AccessDeniedException e, Authentication
     authentication )
{
  return getModelAndView( "Unauthorized", "
      Unauthorized", Error.ACCESS_DENIED.
      getMessage(), authentication );
}


/**
 * Returns a ModelAndView of the custom 404
      error page.
 *
 * @param e
 * @param authentication
 * @return error 404 page
 */
@ResponseStatus( HttpStatus.NOT_FOUND )
@ExceptionHandler( ResourceNotFoundException.
     class )
public ModelAndView handleResourceNotFound(
     ResourceNotFoundException e,
     Authentication authentication )
{
  return getModelAndView( "Page not found", "
      Page not found", e.getMessage(),
      authentication );
}


/**
 * Returns a ModelAndView of the custom 409
      error page.
 *
 * @param e
 * @param authentication
 * @return error 409 page
 */
@ResponseStatus( HttpStatus.CONFLICT )
@ExceptionHandler( ServiceOperationException.
     class )
public ModelAndView handleConflict(
     ServiceOperationException e,
     Authentication authentication )
{
  return getModelAndView( "Conflict", "", e.
      getMessage(), authentication );
}


/**
 * Returns a ModelAndView of the custom 500
      error page.
 *
 * @return error 500 page
 */
@ResponseStatus( HttpStatus.
     INTERNAL_SERVER_ERROR )
@ExceptionHandler( Exception.class )
public ModelAndView handleException(
     Authentication authentication )
{
  return getModelAndView( "Error", "An error
      has occurred", "Uh-oh! An error has
      occurred. :(", authentication );
}


/**
 * Returns a ModelAndView of the error page.
 *
 * @param title
 *     title of the page
 * @param header
 *     header message
 * @param errorMessage
 * @param authentication
 * @return model and view of the error page
 */
private ModelAndView getModelAndView( String
     title, String header, String errorMessage
     , Authentication authentication )
{
  logger.debug( "ERROR: " + errorMessage );

  ModelAndView mav = new ModelAndView(
      ERROR_VIEW_NAME );

  mav.addObject( "title", title );
  mav.addObject( "header", header );
  mav.addObject( "message", errorMessage );
  mav.addObject( "home", linkTo(
      HdsgsController.class ).withRel( "home"
      ) );
  mav.addObject( "navigation", getNavbarLinks(
       authentication ) );
  mav.addObject( "currentUser", getCurrentUser
      ( authentication ) );

  return mav;
}
}
```

38. HdsgsController.java

```
package com.hdsgs.controller;

import static org.springframework.hateoas.mvc.
     ControllerLinkBuilder.linkTo;
import static org.springframework.web.bind.
     annotation.RequestMethod.GET;

import org.springframework.stereotype.Controller
     ;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.
     RequestMapping;
import org.springframework.web.bind.annotation.
     RequestParam;

/**
 * Main controller of the application that
      handles requests not directly
 * involving the users, surveys, or reports.
 */
@Controller
@RequestMapping( value = "/" )
public class HdsgsController extends
     BaseController
{
  /**
   * Handles request to view the index page.
   */
  @RequestMapping( method = GET )
  public String getIndex()
  {
    logger.debug( "Getting home page..." );

    return "index";
  }


  /**
   * Handles request to view the login page.
   *
   * @param error
   *     if there are login errors such as wrong
        credentials
   */
```

```
    @RequestMapping( value = "/login", method =
        GET )
    public String getLoginPage( @RequestParam(
        required = false ) String error, Model
        model )
    {
      logger.debug( "Getting login form..." );

      model.addAttribute( "error", error );
      model.addAttribute( "login", linkTo(
          HdsgsController.class ).slash( "/login"
          ).withRel( "login" ) );

      return "login";
    }
}
```

39. UserController.java

```
package com.hdsgs.controller;

import static org.springframework.hateoas.mvc.
    ControllerLinkBuilder.linkTo;
import static org.springframework.web.bind.
    annotation.RequestMethod.DELETE;
import static org.springframework.web.bind.
    annotation.RequestMethod.GET;
import static org.springframework.web.bind.
    annotation.RequestMethod.POST;
import static org.springframework.web.bind.
    annotation.RequestMethod.PUT;

import java.util.Arrays;
import java.util.stream.Collectors;

import javax.validation.Valid;

import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.security.access.
    prepost.PreAuthorize;
import org.springframework.security.core.
    Authentication;
import org.springframework.stereotype.Controller
    ;
import org.springframework.ui.Model;
import org.springframework.validation.
    BindingResult;
import org.springframework.web.bind.
    WebDataBinder;
import org.springframework.web.bind.annotation.
    InitBinder;
import org.springframework.web.bind.annotation.
    ModelAttribute;
import org.springframework.web.bind.annotation.
    PathVariable;
import org.springframework.web.bind.annotation.
    RequestMapping;
import org.springframework.web.servlet.mvc.
    support.RedirectAttributes;

import com.hdsgs.dto.UserDTO;
import com.hdsgs.dto.assembler.SurveyAssembler;
import com.hdsgs.dto.assembler.UserAssembler;
import com.hdsgs.entity.CurrentUser;
import com.hdsgs.enums.UserRole;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;
import com.hdsgs.service.SurveyService;
import com.hdsgs.service.UserService;
import com.hdsgs.validator.UserValidator;

/**
 * Controller for requests involving the users.
 */
@Controller
@RequestMapping( "/users" )
public class UserController extends
    BaseController
{
  @Autowired
  private UserService userService;

  @Autowired
  private SurveyService surveyService;

  @Autowired
  private UserAssembler userAssembler;

  @Autowired
  private SurveyAssembler surveyAssembler;

  @Autowired
  private UserValidator validator;


  /**
   * Binds {@link UserValidator} to validate the
       data in the Create User and
   * Update User forms.
   */
  @InitBinder( "user" )
  protected void initUserBinder( WebDataBinder
      binder )
  {
    binder.addValidators( validator );
  }


  /**
   * Handles request to view all users by page.
       This can only be accessed by a
   * {@link UserRole#Admin super admin}.
       Redirects to a 403 (Forbidden) page
   * if accessed by a contributor or research
       leader.
   */
  @PreAuthorize( "hasAuthority('Admin')" )
  @RequestMapping( method = GET )
  public String getUsers( Model model )
  {
    model.addAttribute( "users", userAssembler.
        toResources( userService.getAllUsers()
        ) );
    model.addAttribute( "createUser", linkTo(
        UserController.class ).slash( "create"
        ).withRel( "createUser" ) );

    return "users_view_all";
  }


  /**
   * Handles request to view the details of a
       single user. This can be
   * accessed by the owner of the account and by
       a {@link UserRole#Admin super
   * admin}. Redirects to a 403 (Forbidden) page
       if not the owner of the
   * account or a super administrator. Redirects
       to a 404 (Not Found) page if
   * the user does not exist.
   *
   * @param username
   */
  @RequestMapping( value = "/{username}", method
      = GET )
  public String getUser( @PathVariable( "
      username" ) String username, Model model
      ) throws ResourceNotFoundException
  {
    logger.debug( "Getting user page of " +
        username + "..." );

    UserDTO user = userAssembler.toResource(
        userService.findByUsername( username )
        );
    CurrentUser currentUser = ( CurrentUser )
        model.asMap().get( "currentUser" );

    if( currentUser != null )
    {
      if( currentUser.getUsername().equals(
          username ) && currentUser.getRole()
          != UserRole.ADMIN )
      {
        user.add( linkTo( UserController.class )
            .slash( user.getUsername() ).slash(
            "surveys" ).withRel( "surveys" ) )
            ;
      }

      if( currentUser.getRole() == UserRole.
          ADMIN )
      {
        user.add( linkTo( UserController.class )
            .slash( user.getUsername() ).
            withRel( "delete" ) );
      }
```

```
  }

  model.addAttribute( "user", user );

  return "user_view";
}


/**
 * Handles request to view the Create User
     form. This can only be accessed
 * by a {@link UserRole#Admin super admin}.
     Redirects to a 403 (Forbidden)
 * page if not a super administrator.
 */
@PreAuthorize( "hasAuthority('Admin')" )
@RequestMapping( value = "/create", method =
    GET )
public String getCreateUserForm( Model model )
{
  logger.debug( "Getting create user form..."
      );

  if( !model.containsAttribute( "user" ) )
  {
    model.addAttribute( "user", userAssembler.
        toCreateFormResource( new UserDTO() )
        );
  }

  model.addAttribute( "roles", Arrays.stream(
      UserRole.values() ).filter( r -> r !=
      UserRole.ADMIN ).collect( Collectors.
      toList() ) );
  model.addAttribute( "title", "Create User" )
      ;
  model.addAttribute( "method", "post" );

  return "user_create_form";
}


/**
 * Handles the submission of the Create User
     form. If there are validation
 * errors, the client will be redirected back
     to the form that will display
 * the appropriate error messages. Else, the
     new user will be created, and
 * the client will be redirected to the page
     for viewing the newly created
 * user. This can only be accessed by a {@link
     UserRole#Admin super admin}.
 * Redirects to a 403 (Forbidden) page if not
     a super administrator.
 * Redirects to a 409 (Conflict) page if
     username already exists.
 *
 * @param newUser
 */
@PreAuthorize( "hasAuthority('Admin')" )
@RequestMapping( method = POST )
public String createNewUser( @ModelAttribute(
    "user" ) @Valid UserDTO newUser,
    BindingResult result, RedirectAttributes
    redirectAttributes )
    throws ServiceOperationException
{
  if( result.hasErrors() )
  {
    logger.debug( "Create user form has errors
        . Redirecting back to form..." );

    redirectAttributes.addFlashAttribute( "
        user", userAssembler.
        toCreateFormResource( newUser ) );
    redirectAttributes.addFlashAttribute( "org
        .springframework.validation.
        BindingResult.user", result );

    return "redirect:/users/create";
  }

  UserDTO created = userService.createNewUser(
      newUser );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully created
      new account." );

  return "redirect:/users/" + created.
      getUsername();
}


/**
 * Handles request to view the Update User
     form. This can only be accessed
 * by the owner of the account and by a {@link
     UserRole#Admin super admin}.
 * Redirects to a 403 (Forbidden) page if not
     the owner of the account or a
 * super administrator. Redirects to a 404 (
     Not Found) page if user does not
 * exist. Redirects to a 409 (Conflict) page
     if user to update is
 * deactivated.
 *
 * @param username
 */
@PreAuthorize( "isAuthenticated() and (
    hasAuthority('Admin') or principal.
    username == #username )" )
@RequestMapping( value = "/{username}/update",
    method = GET )
public String getUpdateUserForm( @PathVariable
    ( "username" ) String username, Model
    model ) throws ResourceNotFoundException
{
  logger.debug( "Getting update user form of "
      + username + "..." );

  if( !model.containsAttribute( "user" ) )
  {
    model.addAttribute( "user", userAssembler.
        toUpdateFormResource( userService.
        findByUsername( username ) ) );
  }

  model.addAttribute( "title", "Update User" )
      ;
  model.addAttribute( "method", "put" );

  return "user_update_form";
}


/**
 * Handles the submission of the Update User
     form. If there are validation
 * errors, the client will be redirected back
     to the form that will display
 * the appropriate error messages. Else, the
     user details will be updated,
 * and the client will be redirected to the
     page for viewing the updated
 * user. This can only be accessed by the
     owner of the account and by a
 * {@link UserRole#Admin super admin}.
     Redirects to a 403 (Forbidden) page
 * if not the owner of the account or a super
     administrator. Redirects to a
 * 404 (Not Found) page if user does not exist
     . Redirects to a 409
 * (Conflict) page if user to update is
     deactivated, or the new username
 * already exists.
 *
 * @param updatedUser
 */
@PreAuthorize( "isAuthenticated() and (
    hasAuthority('Admin') or principal.id ==
    #updatedUser.userId )" )
@RequestMapping( value = "/{username}", method
    = PUT )
public String updateUser( @ModelAttribute( "
    user" ) @Valid UserDTO updatedUser,
    BindingResult result, RedirectAttributes
    redirectAttributes )
    throws ResourceNotFoundException,
        ServiceOperationException
{
  if( result.hasErrors() )
  {
    logger.debug( "Update user form has errors
        . Redirecting back to form..." );

    redirectAttributes.addFlashAttribute( "
        user", userAssembler.
        toUpdateFormResource( updatedUser ) )
        ;
    redirectAttributes.addFlashAttribute( "org
        .springframework.validation.
        BindingResult.user", result );
```

```
    if( result.hasFieldErrors( "oldPassword" )
          || result.hasFieldErrors( "
        retypedPassword" ) )
    {
      redirectAttributes.addFlashAttribute( "
          hasChangePasswordError", true );
    }

    return "redirect:/users/" + updatedUser.
        getUsername() + "/update";
  }

  UserDTO updated = userService.updateUser(
      updatedUser );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully updated
      account details." );

  return "redirect:/users/" + updated.
      getUsername();
}


/**
 * Handles request to deactivate a user. This
     can only be accessed by owner
 * of the account and by a {@link UserRole#
     Admin super admin}. Redirects to
 * a 403 (Forbidden) page if not the owner of
     the account or a super
 * administrator. Redirects to a 404 (Not
     Found) page if user does not
 * exist. Redirects to a 409 (Conflict) page
     if user is already deactivated.
 *
 * @param username
 */
@PreAuthorize( "isAuthenticated() and (
    hasAuthority('Admin') or principal.
    username == #username )" )
@RequestMapping( value = "/{username}", params
    = "deactivate=true", method = PUT )
public String deactivateUser( @PathVariable( "
    username" ) String username,
    RedirectAttributes redirectAttributes,
    Authentication authentication )
    throws ResourceNotFoundException,
        ServiceOperationException
{
  userService.deactivateUser( username );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully
      deactivated account." );

  return "redirect:/users/" + username;
}


/**
 * Handles request to activate a user with the
     given username. This can only
 * be accessed by a {@link UserRole#Admin
     super admin}. Redirects to a 403
 * (Forbidden) page if not a super
     administrator. Redirects to a 404 (Not
 * Found) page if user does not exist.
     Redirects to a 409 (Conflict) page if
 * user is already active.
 *
 * @param username
 */
@PreAuthorize( "isAuthenticated() and
    hasAuthority('Admin')" )
@RequestMapping( value = "/{username}", params
    = "activate=true", method = PUT )
public String activateUser( @PathVariable( "
    username" ) String username,
    RedirectAttributes redirectAttributes )
    throws ResourceNotFoundException,
        ServiceOperationException
{
  userService.activateUser( username );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully re-
      activated account." );

  return "redirect:/users/" + username;
}


/**
 * Handles request to delete a user. After
     successful deletion, the client
 * will be redirected to view existing users.
     This can only be accessed by a
 * {@link UserRole#Admin super admin}.
     Redirects to a 403 (Forbidden) page
 * if not a super administrator. Redirects to
     a 404 (Not Found) page if user
 * does not exist.
 *
 * @see UserController#getUsers
 * @param username
 */
@PreAuthorize( "hasAuthority('Admin')" )
@RequestMapping( value = "/{username}", method
    = DELETE )
public String deleteUser( @PathVariable( "
    username" ) String username,
    RedirectAttributes redirectAttributes )
    throws ResourceNotFoundException
{
  userService.deleteUser( username );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully deleted
      account: " + username + "." );

  return "redirect:/users";
}


/**
 * Handles request to view the surveys
     associated with the given user. If
 * the user requested is a {@link UserRole#
     CONTRIBUTOR contributor}, the
 * client will be able to view the surveys
     that user answered. If the user
 * requested is a {@link UserRole#
     RESEARCH_LEADER research leader}, the
 * client will be able to view the surveys
     that user created. This can be
 * accessed by the owner of the account and by
     a {@link UserRole#Admin super
 * admin}. Redirects to a 403 (Forbidden) page
     if not the owner of the
 * account or a super administrator. Redirects
     to a 404 (Not Found) page if
 * user does not exist.
 *
 * @param username
 */
@PreAuthorize( "isAuthenticated() and (
    hasAuthority('Admin') or principal.
    username == #username )" )
@RequestMapping( value = "/{username}/surveys"
    , method = GET )
public String getSurveys( @PathVariable( "
    username" ) String username, Model model
    ) throws ResourceNotFoundException
{
  model.addAttribute( "username", username );
  model.addAttribute( "role", userService.
      findByUsername( username ).getRole().
      name() );
  model.addAttribute( "surveys",
      surveyAssembler.toResources(
      surveyService.getSurveys( username ) )
      );

  return "user_surveys";
 }
}
```

40. SurveyController.java

```
package com.hdsgs.controller;

import static org.springframework.hateoas.mvc.
    ControllerLinkBuilder.linkTo;
import static org.springframework.web.bind.
    annotation.RequestMethod.DELETE;
import static org.springframework.web.bind.
    annotation.RequestMethod.GET;
import static org.springframework.web.bind.
    annotation.RequestMethod.POST;
```

```
import static org.springframework.web.bind.
    annotation.RequestMethod.PUT;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

import javax.servlet.http.HttpServletResponse;
import javax.validation.Valid;

import org.joda.time.LocalDate;
import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.security.access.
    prepost.PreAuthorize;
import org.springframework.stereotype.Controller
    ;
import org.springframework.ui.Model;
import org.springframework.validation.
    BindingResult;
import org.springframework.web.bind.
    WebDataBinder;
import org.springframework.web.bind.annotation.
    InitBinder;
import org.springframework.web.bind.annotation.
    ModelAttribute;
import org.springframework.web.bind.annotation.
    PathVariable;
import org.springframework.web.bind.annotation.
    RequestMapping;
import org.springframework.web.bind.annotation.
    RequestParam;
import org.springframework.web.servlet.mvc.
    support.RedirectAttributes;

import com.hdsgs.dto.QuestionDTO;
import com.hdsgs.dto.SurveyAnswer;
import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.dto.assembler.
    SurveyAnswerAssembler;
import com.hdsgs.dto.assembler.SurveyAssembler;
import com.hdsgs.entity.CurrentUser;
import com.hdsgs.enums.Function;
import com.hdsgs.enums.InputType;
import com.hdsgs.enums.UserRole;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;
import com.hdsgs.service.ReportService;
import com.hdsgs.service.SurveyService;
import com.hdsgs.service.UserService;
import com.hdsgs.service.impl.CurrentUserService
    ;
import com.hdsgs.validator.SurveyValidator;

/**
 * Controller for requests involving the surveys
    .
 */
@Controller
@RequestMapping( "/surveys" )
public class SurveyController extends
    BaseController
{
  @Autowired
  private SurveyService surveyService;

  @Autowired
  private UserService userService;

  @Autowired
  private CurrentUserService currentUserService;

  @Autowired
  private ReportService reportService;

  @Autowired
  private SurveyAssembler assembler;

  @Autowired
  private SurveyAnswerAssembler answerAssembler;

  @Autowired
  private SurveyValidator validator;


  /**
   * Binds {@link SurveyValidator} to validate
       the data in the Create Survey
   * and Update Survey forms.
   */
@InitBinder( "survey" )
protected void initSurveyBinder( WebDataBinder
    binder )
{
  binder.addValidators( validator );
}


/**
 * Handles request to view all surveys. This
     can be accessed by all users
 * except {@link UserRole#Admin super admin}.
     Redirects to a 403 (Forbidden)
 * page if accessed by a super admin.
 */
@PreAuthorize( "!hasAuthority('Admin')" )
@RequestMapping( method = GET )
public String getSurveys( Model model )
{
  logger.debug( "Getting surveys..." );

  CurrentUser currentUser = ( CurrentUser )
      model.asMap().get( "currentUser" );

  if( currentUser != null && currentUser.
      getRole() == UserRole.RESEARCH_LEADER )
  {
    model.addAttribute( "createSurvey", linkTo
        ( SurveyController.class ).slash( "
        create" ).withRel( "createSurvey" ) )
        ;
  }

  model.addAttribute( "surveys", assembler.
      toResources( surveyService.
      getAllSurveys() ) );

  return "surveys_view_all";
}


/**
 * Handles request to view details of a single
     survey. This can be accessed
 * by all users except {@link UserRole#Admin
     super admin}. Redirects to a
 * 403 (Forbidden) page if accessed by a super
     administrator. Redirects to a
 * 404 (Not Found) page if the survey
     requested does not exist.
 *
 * @param surveyId
 */
@PreAuthorize( "!hasAuthority('Admin')" )
@RequestMapping( value = "/{surveyId}", method
    = GET )
public String getSurvey( @PathVariable( "
    surveyId" ) int surveyId, Model model )
    throws ResourceNotFoundException
{
  logger.debug( "Getting survey page of survey
      " + surveyId + "..." );

  SurveyDTO survey = assembler.toResource(
      surveyService.findById( surveyId ) );
  CurrentUser currentUser = ( CurrentUser )
      model.asMap().get( "currentUser" );

  if( currentUser != null )
  {
    if( currentUserService.isSurveyCreator(
        currentUser.getUsername(), surveyId )
        )
    {
      if( LocalDate.now().isBefore( survey.
          getStartDate() ) )
      {
        survey.add( linkTo( SurveyController.
            class ).slash( survey.getSurveyId
            () ).slash( "update" ).withRel( "
            update" ) );
      }
      else if( survey.isActive() )
      {
        survey.add( linkTo( SurveyController.
            class ).slash( survey.getSurveyId
            () ).withRel( "deactivate" ) );
      }

      survey.add( linkTo( SurveyController.
          class ).slash( survey.getSurveyId()
          ).withRel( "delete" ) );
```

```
    }
    else if( currentUser.getRole() == UserRole
        .CONTRIBUTOR && survey.isActive() )
    {
      survey.add( linkTo( SurveyController.
          class ).slash( survey.getSurveyId()
          ).slash( "answer" ).withRel( "
          answer" ) );
    }
  }

  model.addAttribute( "survey", survey );

  return "survey_view";
}


/**
 * Handles request to view the Create Survey
     form. This can only be accessed
 * by a {@link UserRole#RESEARCH_LEADER
     research leader}. Redirects to a 403
 * (Forbidden) page if accessed by a user that
     is not a research leader.
 * Redirects to a 409 (Conflict) page if
     creator does not exist, or is
 * deactivated.
 */
@PreAuthorize( "hasAuthority('Research leader
    ')" )
@RequestMapping( value = "/create", method =
    GET )
public String getCreateSurveyForm( Model model
    )
{
  logger.debug( "Getting create survey form...
      " );

  if( !model.containsAttribute( "survey" ) )
  {
    SurveyDTO survey = new SurveyDTO();
    survey.setCreator( ( ( CurrentUser ) model
        .asMap().get( "currentUser" ) ).
        getUsername() );
    survey.setQuestions( Arrays.asList( new
        QuestionDTO() ) );

    model.addAttribute( "survey", assembler.
        toCreateFormResource( survey ) );
  }

  model.addAttribute( "inputTypes", InputType.
      values() );
  model.addAttribute( "functions", Function.
      values() );
  model.addAttribute( "title", "Create Survey"
      );
  model.addAttribute( "method", "post" );

  return "survey_form";
}


/**
 * Handles the submission of the Create Survey
     form. If there are validation
 * errors, the client will be redirected back
     to the form that will display
 * the appropriate error messages. Else, the
     new survey will be created, and
 * the client will be redirected to the page
     for viewing the details of the
 * newly created survey. This can only be
     accessed by a
 * {@link UserRole#RESEARCH_LEADER research
     leader}. Redirects to a 403
 * (Forbidden) page if not a research leader.
     Redirects to a 409 (Conflict)
 * page if creator does not exist, or is
     deactivated.
 *
 * @param newSurvey
 */
@PreAuthorize( "hasAuthority('Research leader
    ')" )
@RequestMapping( method = POST )
public String createNewSurvey( @ModelAttribute
    ( "survey" ) @Valid SurveyDTO newSurvey,
    BindingResult result, RedirectAttributes
    redirectAttributes )
    throws ServiceOperationException
{
  if( result.hasErrors() )
  {
    logger.debug( "Create survey form has
        errors. Redirecting back to form..."
        );

    redirectAttributes.addFlashAttribute( "
        survey", assembler.
        toCreateFormResource( newSurvey ) );
    redirectAttributes.addFlashAttribute( "org
        .springframework.validation.
        BindingResult.survey", result );

    return "redirect:/surveys/create";
  }

  SurveyDTO created = surveyService.
      createNewSurvey( newSurvey );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully created
      new survey" );

  return "redirect:/surveys/" + created.
      getSurveyId();
}


/**
 * Handles request to add a question to the
     new survey to create.
 *
 * @param newSurvey
 */
@PreAuthorize( "hasAuthority('Research leader
    ')" )
@RequestMapping( params = { "addQuestion" },
    method = POST )
public String addQuestionInCreateForm(
    @ModelAttribute( "survey" ) SurveyDTO
    newSurvey, RedirectAttributes
    redirectAttributes )
{
  newSurvey.getQuestions().add( new
      QuestionDTO() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toCreateFormResource
      ( newSurvey ) );

  return "redirect:/surveys/create";
}


/**
 * Handles request to remove a question from
     the new survey to create.
 * Successful only if survey currently has
     more than one question.
 *
 * @param newSurvey
 * @param questionIndex
 */
@PreAuthorize( "hasAuthority('Research leader
    ')" )
@RequestMapping( params = { "removeQuestion"
    }, method = POST )
public String removeQuestionInCreateForm(
    @RequestParam( "removeQuestion" ) int
    questionIndex, @ModelAttribute( "survey"
    ) SurveyDTO newSurvey,
    RedirectAttributes redirectAttributes )
{
  removeQuestion( questionIndex, newSurvey.
      getQuestions() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toCreateFormResource
      ( newSurvey ) );

  return "redirect:/surveys/create";
}


/**
 * Helper method to remove a question in the
     Create Survey or Update Survey
 * forms.
 *
 * @param questionIndex
 * @param questions
 */
```

```
private void removeQuestion( int questionIndex
    , List< QuestionDTO > questions )
{
  if( questions.size() > 1 )
  {
    questions.remove( questionIndex );
  }
}


/**
 * Handles request to add a choice to a
     question in the new survey to
 * create.
 *
 * @param newSurvey
 * @param questionIndex
 */
@PreAuthorize( "hasAuthority('Research leader
    ')" )
@RequestMapping( params = { "addChoice", "
    questionIndex" }, method = POST )
public String addChoiceInCreateForm(
    @RequestParam( "questionIndex" ) int
    questionIndex, @ModelAttribute( "survey"
    ) SurveyDTO newSurvey,
    RedirectAttributes redirectAttributes )
{
  newSurvey.getQuestions().get( questionIndex
      ).getChoices().add( new String() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toCreateFormResource
      ( newSurvey ) );

  return "redirect:/surveys/create";
}


/**
 * Handles request to remove a choice from a
     question in the new survey to
 * create. Successful only if survey question
     currently has more than 2
 * choices.
 *
 * @param newSurvey
 * @param questionIndex
 * @param choiceIndex
 */
@PreAuthorize( "hasAuthority('Research leader
    ')" )
@RequestMapping( params = { "removeChoice", "
    questionIndex" }, method = POST )
public String removeChoiceInCreateForm(
    @RequestParam( "removeChoice" ) int
    choiceIndex, @RequestParam( "
    questionIndex" ) int questionIndex,
    @ModelAttribute( "survey" ) SurveyDTO
        newSurvey, RedirectAttributes
        redirectAttributes )
{
  removeChoice( choiceIndex, newSurvey.
      getQuestions().get( questionIndex ).
      getChoices() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toCreateFormResource
      ( newSurvey ) );

  return "redirect:/surveys/create";
}


/**
 * Helper method to remove a choice from a
     question in the Create Survey or
 * Update Survey forms.
 *
 * @param choiceIndex
 * @param choices
 */
private void removeChoice( int choiceIndex,
    List< String > choices )
{
  if( choices.size() > 2 )
  {
    choices.remove( choiceIndex );
  }
}
```

```
/**
 * Handles request to view the Update Survey
     form. This can only be accessed
 * by the creator of the survey. Redirects to
     a 403 (Forbidden) page if not
 * the creator. Redirects to a 404 (Not Found)
     page if survey does not
 * exist. Redirects to a 409 (Conflict) page
     if creator does not exist.
 *
 * @param surveyId
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #surveyId )" )
@RequestMapping( value = "/{surveyId}/update",
    method = GET )
public String getUpdateSurveyForm(
    @PathVariable( "surveyId" ) int surveyId,
    Model model ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  logger.debug( "Getting update survey form of
      survey " + surveyId + "..." );

  if( !model.containsAttribute( "survey" ) )
  {
    model.addAttribute( "survey", assembler.
        toUpdateFormResource( surveyService.
        getSurveyToUpdate( surveyId ) ) );
  }

  model.addAttribute( "inputTypes", InputType.
      values() );
  model.addAttribute( "functions", Function.
      values() );
  model.addAttribute( "title", "Update Survey"
      );
  model.addAttribute( "method", "put" );

  return "survey_form";
}


/**
 * Handles the submission of the Update Survey
     form. If there are validation
 * errors, the client will be redirected back
     to the form that will display
 * the appropriate error messages. Else, the
     survey will be updated, and the
 * client will be redirected to the page for
     viewing the details of the
 * updated survey. This can only be accessed
     by the creator of the survey.
 * Redirects to a 403 (Forbidden) page if not
     the creator. Redirects to a
 * 404 (Not Found) page if survey does not
     exist. Redirects to a 409
 * (Conflict) page if creator is deactivated,
     or if survey has already
 * started / ended.
 *
 * @param updatedSurvey
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #updatedSurvey.
    surveyId )" )
@RequestMapping( value = "/{surveyId}", method
    = PUT )
public String updateSurvey( @ModelAttribute( "
    survey" ) @Valid SurveyDTO updatedSurvey,
    BindingResult result,
    RedirectAttributes redirectAttributes )
        throws ResourceNotFoundException,
        ServiceOperationException
{
  if( result.hasErrors() )
  {
    logger.debug( "Update survey form has
        errors. Redirecting back to form..."
        );

    redirectAttributes.addFlashAttribute( "
        survey", assembler.
        toUpdateFormResource( updatedSurvey )
        );
    redirectAttributes.addFlashAttribute( "org
```

```
        .springframework.validation.
        BindingResult.survey", result );

    return "redirect:/surveys/" +
        updatedSurvey.getSurveyId() + "/
        update";
  }

  SurveyDTO updated = surveyService.
      updateSurvey( updatedSurvey );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully updated
      new survey" );

  return "redirect:/surveys/" + updated.
      getSurveyId();
}


/**
 * Handles request to add a question to the
     survey to update.
 *
 * @param survey
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #survey.surveyId )" )
@RequestMapping( value = "/{surveyId}", params
    = { "addQuestion" }, method = PUT )
public String addQuestionInUpdateForm(
    @ModelAttribute( "survey" ) SurveyDTO
    survey, RedirectAttributes
    redirectAttributes )
{
  survey.getQuestions().add( new QuestionDTO()
      );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toUpdateFormResource
      ( survey ) );

  return "redirect:/surveys/" + survey.
      getSurveyId() + "/update";
}


/**
 * Handles request to remove a question from
     the survey to update.
 *
 * @param survey
 * @param questionIndex
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #survey.surveyId )" )
@RequestMapping( value = "/{surveyId}", params
    = { "removeQuestion" }, method = PUT )
public String removeQuestionInUpdateForm(
    @RequestParam( "removeQuestion" ) int
    questionIndex, @ModelAttribute( "survey"
    ) SurveyDTO survey,
    RedirectAttributes redirectAttributes )
{
  removeQuestion( questionIndex, survey.
      getQuestions() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toUpdateFormResource
      ( survey ) );

  return "redirect:/surveys/" + survey.
      getSurveyId() + "/update";
}


/**
 * Handles request to add a choice to a
     specific question in the survey to
 * update.
 *
 * @param survey
 * @param questionIndex
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #survey.surveyId )" )
@RequestMapping( value = "/{surveyId}", params
    = { "addChoice", "questionIndex" },
    method = PUT )
public String addChoiceInUpdateForm(
    @RequestParam( "questionIndex" ) int
    questionIndex, @ModelAttribute( "survey"
    ) SurveyDTO survey,
    RedirectAttributes redirectAttributes )
{
  survey.getQuestions().get( questionIndex ).
      getChoices().add( new String() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toUpdateFormResource
      ( survey ) );

  return "redirect:/surveys/" + survey.
      getSurveyId() + "/update";
}


/**
 * Handles request to remove a choice from a
     question in the survey to
 * update.
 *
 * @param survey
 * @param questionIndex
 * @param choiceIndex
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #survey.surveyId )" )
@RequestMapping( value = "/{surveyId}", params
    = { "removeChoice", "questionIndex" },
    method = PUT )
public String removeChoiceInUpdateForm(
    @RequestParam( "removeChoice" ) int
    choiceIndex, @RequestParam( "
    questionIndex" ) int questionIndex,
    @ModelAttribute( "survey" ) SurveyDTO
        survey, RedirectAttributes
        redirectAttributes )
{
  removeChoice( choiceIndex, survey.
      getQuestions().get( questionIndex ).
      getChoices() );

  redirectAttributes.addFlashAttribute( "
      survey", assembler.toUpdateFormResource
      ( survey ) );

  return "redirect:/surveys/" + survey.
      getSurveyId() + "/update";
}


/**
 * Handles request to answer a survey. This
     can only be accessed by a
 * {@code UserRole#CONTRIBUTOR}. Redirects to
     a 403 (Forbidden) page if not
 * a contributor. Redirects to a 404 (Not
     Found) page if survey does not
 * exist. Redirects to a 409 (Conflict) page
     if survey is deactivated.
 *
 * @param surveyId
 */
@PreAuthorize( "hasAuthority('Contributor')" )
@RequestMapping( value = "/{surveyId}/answer",
    method = GET )
public String getAnswerSurveyForm(
    @PathVariable( "surveyId" ) int surveyId,
    Model model ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  final String contributor = ( ( CurrentUser )
      model.asMap().get( "currentUser" ) ).
      getUsername();

  model.addAttribute( "survey", surveyService.
      getSurveyToAnswer( surveyId,
      contributor ) );

  if( !model.containsAttribute( "surveyAnswer"
      ) )
  {
    SurveyAnswer surveyAnswer = new
        SurveyAnswer();
```

```
    surveyAnswer.setSurveyId( surveyId );
    surveyAnswer.setContributor( contributor )
        ;

    model.addAttribute( "surveyAnswer",
        answerAssembler.toResource(
        surveyAnswer ) );
  }

  return "survey_answer";
}


/**
 * Handles request to submit answer to the
     survey. If there are validation
 * errors, the client will be redirected to
     the form with the corresponding
 * error messages. Else, the answer will be
     inserted in the database for
 * computation of results, then redirected to
     page for viewing the details
 * of the survey. This can only be accessed by
     a
 * {@code UserRole#CONTRIBUTOR contributor}.
     Redirects to a 403 (Forbidden)
 * page if not a contributor. Redirects to a
     404 (Not Found) page if survey
 * does not exist. Redirects to a 409 (
     Conflict) page if survey or
 * contributor is deactivated.
 *
 * @param surveyAnswer
 */
@PreAuthorize( "hasAuthority('Contributor')" )
@RequestMapping( value = "/{surveyId}/answer",
     method = POST )
public String answerSurvey( @ModelAttribute( "
    surveyAnswer" ) @Valid SurveyAnswer
    surveyAnswer, BindingResult result,
    RedirectAttributes redirectAttributes )
        throws ResourceNotFoundException,
        ServiceOperationException
{
  int surveyId = surveyAnswer.getSurveyId();

  if( result.hasErrors() )
  {
    redirectAttributes.addFlashAttribute( "
        surveyAnswer", answerAssembler.
        toResource( surveyAnswer ) );
    redirectAttributes.addFlashAttribute( "org
        .springframework.validation.
        BindingResult.surveyAnswer", result )
        ;

    return "redirect:/surveys/" + surveyId + "
        /answer";
  }

  surveyService.recordAnswers( surveyId,
      surveyAnswer.getAnswers(), surveyAnswer
      .getContributor() );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully
      submitted answer" );

  return "redirect:/surveys/" + surveyId;
}


/**
 * Handles request to deactivate a survey.
     This can only be accessed by the
 * creator of the survey. Redirects to a 403 (
     Forbidden) page if not the
 * creator. Redirects to a 404 (Not Found)
     page if survey does not exist.
 * Redirects to a 409 (Conflict) page if
     survey is already deactivated.
 *
 * @param surveyId
 */
@PreAuthorize( "isAuthenticated() and hasRole
    ('Research leader') and
    @currentUserService.isSurveyCreator(
    principal.username, #surveyId )" )
@RequestMapping( value = "/{surveyId}", params
     = "deactivate=true", method = PUT )
public String deactivateSurvey( @PathVariable(
     "surveyId" ) int surveyId,
    RedirectAttributes redirectAttributes )
    throws ResourceNotFoundException,
        ServiceOperationException
{
  surveyService.deactivateSurvey( surveyId );

  redirectAttributes.addFlashAttribute( "
      successMessage", "Successfully
      deactivated survey" );

  return "redirect:/surveys/" + surveyId;
}


/**
 * Handles request to view the results of a
     survey. This can be viewed by
 * all user types except {@code UserRole#Admin
     super admin}. Redirects to a
 * 403 (Forbidden) page if super administrator
     . Redirects to a 404 (Not
 * Found) page if survey does not exist.
     Redirects to a 409 (Conflict) page
 * if survey has not yet started or is on-
     going.
 *
 * @param surveyId
 */
@PreAuthorize( "!hasAuthority('Admin')" )
@RequestMapping( value = "/{surveyId}/results"
    , method = GET )
public String viewResults( @PathVariable( "
    surveyId" ) int surveyId, Model model )
    throws ResourceNotFoundException,
    ServiceOperationException
{
  model.addAttribute( "results", surveyService
      .computeResults( surveyId ) );
  model.addAttribute( "totalContributors",
      surveyService.getTotalContributors(
      surveyId ) );
  model.addAttribute( "survey", assembler.
      toResource( surveyService.findById(
      surveyId ) ) );

  return "survey_results";
}


/**
 * Handles request to download a PDF of the
     survey results. This can be
 * viewed by all user types except {@code
     UserRole#Admin super admin}.
 * Redirects to a 403 (Forbidden) page if
     super administrator. Redirects to
 * a 404 (Not Found) page if survey does not
     exist. Redirects to a 409
 * (Conflict) page if survey has not yet
     started or is on-going.
 *
 * @param surveyId
 */
@PreAuthorize( "!hasAuthority('Admin')" )
@RequestMapping( value = "/{surveyId}/results/
    download", method = GET )
public void downloadResults( @PathVariable( "
    surveyId" ) int surveyId,
    HttpServletResponse response ) throws
    ResourceNotFoundException,
    ServiceOperationException
{
  SurveyDTO survey = surveyService.findById(
      surveyId );

  try
  {
    response.setContentType( "application/pdf"
        );
    response.setHeader( "Content-disposition",
        "inline; filename=" + survey.
        getTitle() + " - Results.pdf" );

    reportService.generateReport( survey,
        response.getOutputStream() );
  }
  catch( IOException e )
  {
    logger.error( e.getMessage() );
  }
}
```

```
  /**
   * Handles request to delete a survey. After
       successful deletion, the client
   * will be redirected to view existing surveys
       . This can only be accessed by
   * the creator of the survey. Redirects to a
       403 (Forbidden) page if not the
   * creator. Redirects to a 404 (Not Found)
       page if survey does not exist.
   *
   * @see SurveyController#getSurveys
   * @param surveyId
   */
  @PreAuthorize( "isAuthenticated() and hasRole
      ('Research leader') and
      @currentUserService.isSurveyCreator(
      principal.username, #surveyId )" )
  @RequestMapping( value = "/{surveyId}", method
      = DELETE )
  public String deleteSurvey( @PathVariable( "
      surveyId" ) int surveyId,
      RedirectAttributes redirectAttributes )
      throws ResourceNotFoundException
  {
    surveyService.deleteSurvey( surveyId );

    redirectAttributes.addFlashAttribute( "
        successMessage", "Successfully deleted
        survey with ID : " + surveyId );

    return "redirect:/surveys";
  }
}
```

41. SharemindController.java

```
package com.hdsgs.constants;

/**
 * Constants for the directory paths of
     Sharemind controllers, and string
 * markers to be used for filtering the output
     of those controllers.
 */
public final class SharemindController
{
  /**
   * Folder that contains the Sharemind
       controllers.
   */
  public final static String DIRECTORY = "/home/
      sharemind/DevTools/bin/";

  /**
   * For inserting survey answers to the
       Sharemind database.
   */
  public final static String INSERT = "./
      Survey_DatabaseEntry";

  /**
   * For computing the total number of
       contributors to a particular survey.
   */
  public final static String NUM_OF_CONTRIBUTORS
      = "./Survey_GetNumberOfContributors";

  /**
   * Marks the line that contains the total
       number of contributors.
   */
  public final static String
      NUM_OF_CONTRIBUTORS_MARKER = "
      Contributors = ";

  /**
   * For computing the average of answers to a
       particular question on a
   * survey.
   */
  public final static String AVERAGE = "./
      Survey_GetAverage";

  /**
   * Marks the line that contains the average to
       a question.
   */
  public final static String AVERAGE_MARKER = "
      Average = ";

  /**
   * For computing the minimum answer to a
       particular question on a survey.
   */
  public final static String MINIMUM = "./
      Survey_GetMinimum";

  /**
   * Marks the line that contains the minimum
       answer to a question.
   */
  public final static String MINIMUM_MARKER = "
      Minimum = ";

  /**
   * For computing the maximum answer to a
       particular question on a survey.
   */
  public final static String MAXIMUM = "./
      Survey_GetMaximum";

  /**
   * Marks the line that contains the maximum
       answer to a question.
   */
  public final static String MAXIMUM_MARKER = "
      Maximum = ";

  /**
   * For counting the number of times a specific
       choice of a question on a
   * survey was answered by the contributors.
   */
  public final static String TALLY = "./
      Survey_GetTally";

  /**
   * Marks the line that contains the count of
       the number of times a specific
   * choice of a question on a survey was
       answered by the contributors.
   */
  public final static String TALLY_MARKER = "
      Tally = ";
}
```

42. UserRole.java

```
package com.hdsgs.enums;

/**
 * Represents the different roles a user can
     have.
 */
public enum UserRole
{
  CONTRIBUTOR( "Contributor" ),

  RESEARCH_LEADER( "Research leader" ),

  ADMIN( "Admin" );

  String role;


  /**
   * Sets the text associated with this enum.
   *
   * @param role
   */
  private UserRole( String role )
  {
    this.role = role;
  }


  @Override
  public String toString()
  {
    return role;
  }
}
```

43. SurveyStatus.java

```
package com.hdsgs.enums;

/**
 * Represents the different survey status.
 */
public enum SurveyStatus
{
  NOT_YET_STARTED( "Not yet started" ),

  ONGOING( "Ongoing" ),

  ENDED( "Ended" );

  private String status;


  /**
   * Sets the status associated with this enum.
   *
   * @param status
   */
  private SurveyStatus( String status )
  {
    this.status = status;
  }


  @Override
  public String toString()
  {
    return status;
  }
}
```

44. Function.java

```
package com.hdsgs.enums;

/**
 * Represents the different result functions
      that a question may have.
 */
public enum Function
{
  AVERAGE,

  MINIMUM,

  MAXIMUM,

  TALLY;
}
```

45. InputType.java

```
package com.hdsgs.enums;

/**
 * Represents the different input types a
      question can have.
 */
public enum InputType
{
  NUMBER,

  RADIO,

  CHECKBOX,

  DROPDOWN;
}
```

46. Error.java

```
package com.hdsgs.enums;

/**
 * Represents the different error messages for
      the application.
 */
public enum Error
{
  USERNAME_EXISTS( "Username already exists" ),

  USER_DNE( "User does not exist" ),

  USER_DEACTIVATED( "User is deactivated" ),

  USER_ACTIVATED( "User is activated" ),

  SURVEY_DNE( "Survey does not exist" ),

  SURVEY_NOT_STARTED( "Survey has not yet
       started" ),

  SURVEY_ONGOING( "Survey is on-going" ),

  SURVEY_DEACTIVATED( "Survey is deactivated" ),

  ALREADY_ANSWERED( "You have already answered
       the survey" ),

  ACCESS_DENIED( "You do not have access to this
       page" );

  String message;


  /**
   * Initializes this enum with the given error
       message.
   *
   * @param message
   *     error message
   */
  private Error( String message )
  {
    this.message = message;
  }


  /**
   * Returns the error message associated with
       this enum.
   *
   * @return error message
   */
  public String getMessage()
  {
    return this.message;
  }
}
```

47. ServiceOperationException.java

```
package com.hdsgs.exception;

import com.hdsgs.enums.Error;

/**
 * An exception that is thrown when there is an
      error in executing services.
 * This exception primarily means there is a
      conflict between the needed
 * information and the one provided by the
      client.
 */
public class ServiceOperationException extends
     Exception
{
  private static final long serialVersionUID =
       2675943545022072299L;


  /**
   * Constructs a new ServiceOperationException
       with the given message.
   *
   * @param error
   *     error message
   */
  public ServiceOperationException( Error error
       )
  {
    super( error.getMessage() );
  }
}
```

48. ResourceNotFoundException.java

```java
package com.hdsgs.exception;

import com.hdsgs.enums.Error;

/**
 * An exception that is thrown when the
      requested resource cannot be found.
 */
public class ResourceNotFoundException extends
     Exception
{
  private static final long serialVersionUID =
       9638464235377637 33L;


  /**
   * Constructs a new ResourceNotFoundException
        with the given error.
   *
   * @param error
   */
  public ResourceNotFoundException( Error error
       )
  {
    super( error.getMessage() );
  }
}
```

49. UserValidator.java

```java
package com.hdsgs.validator;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.
     annotation.Autowired;
import org.springframework.context.MessageSource
     ;
import org.springframework.security.crypto.
     bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import com.hdsgs.dto.UserDTO;
import com.hdsgs.exception.
     ResourceNotFoundException;
import com.hdsgs.service.UserService;

/**
 * Used for validating input from a Create User
      or Update User form.
 */
@Component
public class UserValidator implements Validator
{
  protected final Logger logger = LoggerFactory.
       getLogger( getClass() );

  @Autowired
  private MessageSource messageSource;

  @Autowired
  private UserService userService;


  @Override
  public boolean supports( Class< ? > clazz )
  {
    return UserDTO.class.isAssignableFrom( clazz
         );
  }


  @Override
  public void validate( Object target, Errors
       errors )
  {
    UserDTO userDto = ( UserDTO ) target;
    String username = userDto.getUsername();
    UserDTO userInDb;

    try
    {
      userInDb = userService.findByUsername(
           username );
    }
    catch( ResourceNotFoundException e )
    {
      return;
    }

    if( userInDb != null && userDto.getUserId()
         != userInDb.getUserId() )
    {
      errors.rejectValue( "username", "username.
           exists.error" );
    }

    BCryptPasswordEncoder encoder = new
         BCryptPasswordEncoder();

    try
    {
      if( userDto.getPassword() != null &&
           userDto.getPassword().length() >= 2
          && !encoder.matches( userDto.
               getOldPassword(), userService.
               getPassword( username ) ) )
      {
        errors.rejectValue( "oldPassword", "old.
             password.error" );
      }
      else if( userDto.getPassword() != null &&
           !userDto.getPassword().equals(
           userDto.getRetypedPassword() ) )
      {
        errors.rejectValue( "retypedPassword", "
             retyped.password.error" );
      }
    }
    catch( ResourceNotFoundException e )
    {
      logger.error( e.getMessage() );
    }
  }
}
```

50. SurveyValidator.java

```java
package com.hdsgs.validator;

import java.util.List;

import org.joda.time.LocalDate;
import org.springframework.beans.factory.
     annotation.Autowired;
import org.springframework.context.MessageSource
     ;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.
     ValidationUtils;
import org.springframework.validation.Validator;

import com.hdsgs.dto.QuestionDTO;
import com.hdsgs.dto.SurveyDTO;

/**
 * Used for validating input from a Create
      Survey or Update Survey form.
 */
@Component
public class SurveyValidator implements
     Validator
{
  @Autowired
  private MessageSource messageSource;


  @Override
  public boolean supports( Class< ? > clazz )
  {
    return SurveyDTO.class.isAssignableFrom(
         clazz );
  }


  @Override
  public void validate( Object target, Errors
       errors )
  {
    SurveyDTO survey = ( SurveyDTO ) target;

    LocalDate endDate = survey.getEndDate();
    LocalDate startDate = survey.getStartDate();

    // validate end date is after start date
```

```
    if( startDate != null && endDate != null &&
        startDate.isAfter( endDate ) )
    {
      errors.rejectValue( "endDate", "end.date.
          error" );
    }

    // validate start date is today or in the
        future
    if( startDate != null && startDate.isBefore(
        LocalDate.now() ) )
    {
      errors.rejectValue( "startDate", "start.
          date.error" );
    }

    // validate choices
    List< QuestionDTO > questions = survey.
        getQuestions();
    for( int i = 0; i < questions.size(); i++ )
    {
      QuestionDTO q = questions.get( i );

      List< String > choices = q.getChoices();
      for( int j = 0; j < choices.size(); j++ )
      {
        String field = "questions[" + i + "].
            choices[" + j + "]";

        ValidationUtils.
            rejectIfEmptyOrWhitespace( errors,
            field, "empty.choice.error" );
      }
    }
  }
}
```

51. ComputeResultsTask.java

```
package com.hdsgs.task;

import java.text.SimpleDateFormat;
import java.util.Date;

import org.springframework.beans.factory.
    annotation.Autowired;
import org.springframework.scheduling.annotation
    .Scheduled;
import org.springframework.stereotype.Component;
import org.springframework.transaction.
    annotation.Transactional;

import com.hdsgs.dto.SurveyDTO;
import com.hdsgs.exception.
    ResourceNotFoundException;
import com.hdsgs.exception.
    ServiceOperationException;
import com.hdsgs.service.BaseService;
import com.hdsgs.service.SurveyService;

/**
 * Contains the task for computing survey
     results.
 */
@Component
public class ComputeResultsTask extends
    BaseService
{
  @Autowired
  private SurveyService surveyService;

  private static final SimpleDateFormat
      dateFormat = new SimpleDateFormat( "HH:mm
      :ss" );


  /**
   * Runs the task of computing the results of
       surveys which ended yesterday. This is
       scheduled to run at the start of
   * every day, i.e., 12:00 AM.
   */
  @Scheduled( cron = "0 0 0 * * *" )
  @Transactional
  public void computeResults()
  {
    logger.debug( "Starting computing results
        task - the time is now " + dateFormat.
        format( new Date() ) );

    try
    {
      for( SurveyDTO endedYesterday :
          surveyService.getEndedYesterday() )
      {
        surveyService.computeResults(
            endedYesterday.getSurveyId() );
      }
    }
    catch( ResourceNotFoundException e )
    {
      logger.error( e.getMessage() );

      return;
    }
    catch( ServiceOperationException e )
    {
      logger.error( e.getMessage() );

      return;
    }

    logger.debug( "Computing results task
        finished" );
  }
}
```

52. index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1" />
  <title>Home</title>
  <link th:replace="imports" />
</head>
<body role="document">
  <th:block th:if="${currentUser}">
    <div th:replace="navbar :: navbar(role='__${
        currentUser.getRole().toString()}__',
        name='__${currentUser.getName()}__')"
        ></div>
  </th:block>
  <th:block th:if="${currentUser == null}">
    <div th:replace="navbar :: navbar(role='',
        name='')"></div>
  </th:block>

  <div class="container">
    <div class="jumbotron">
      <h1>Welcome to HDSGS!</h1>
      <p>Health Data Statistics Generation
       System</p>
    </div>
  </div>
</body>
</html>
```

53. imports.html

```
<link th:href="@{/myfavicon.ico}" rel="icon"
    type="image/x-icon" />
<link rel="stylesheet" th:href="@{/webjars/
    bootstrap/3.3.4/css/bootstrap.min.css}" />
<link rel="stylesheet" th:href="@{/css/style.css
    }" />
<script th:src="@{/webjars/jquery/1.11.1/jquery.
    min.js}"></script>
<script th:src="@{/webjars/bootstrap/3.3.4/js/
    bootstrap.min.js}"></script>
<script th:src="@{/js/script.js}"></script>
```

54. navbar.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
</head>
<body>
  <th:block th:fragment="navbar(role, name)">
    <nav class="navbar navbar-default navbar-
        fixed-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" th:href="${
      navigation.getLink('self').href}"
      >HDSGS</a>
</div>
<div id="navbar" class="navbar-collapse
    collapse navbar-right">
  <ul class="nav navbar-nav">
    <li>
      <a th:href="${navigation.getLink('
          self').href}">
        <span class="glyphicon glyphicon
            -home" aria-hidden="true"
            ></span>
      </a>
    </li>
    <li id="usersNav" th:if="${role == '
        Admin'}">
      <a th:href="${navigation.getLink('
          users').href}">
        <span class="glyphicon glyphicon
            -user" aria-hidden="true"
            ></span>   Users
      </a>
    </li>
    <li id="surveysNav" th:unless="${
        role == 'Admin'}">
      <a th:href="${navigation.getLink('
          surveys').href}">
        <span class="glyphicon glyphicon
            -th-list" aria-hidden="true
            "></span>   Surveys
      </a>
    </li>
    <li th:if="${name != ''}">
      <a href="#" class="dropdown-toggle
          " data-toggle="dropdown" aria
          -expanded="false">
        <th:block th:text="${name}"></th
            :block> <span class="
            glyphicon glyphicon-menu-
            down" aria-hidden="true"></
            span>
      </a>
      <ul class="dropdown-menu" role="
          menu">
        <li><a th:href="${navigation.
            getLink('profile').href}">
            View profile</a></li>
        <li><a th:href="${navigation.
            getLink('updateProfile').
            href}">Update profile</a></
            li>
        <li class="divider"></li>
        <li>
          <form th:action="@{/logout}"
              method="post" role="form"
              >
            <div class="logoutBtn-
                container">
                 
              <button type="submit"
                  class="btn" id="
                  logoutBtn">
                <span class="glyphicon
                    glyphicon-log-out"
                    aria-hidden="true"
                    ></span> Logout
              </button>
            </div>
          </form>
        </li>
      </ul>
    </li>
    <li th:if="${name == ''}">
      <a th:href="${navigation.getLink('
          login').href}">
        <span class="glyphicon glyphicon
            -log-in" aria-hidden="true"
            ></span>  Login
      </a>
    </li>
          </ul>
        </div>
      </div>
    </nav>
  </th:block>
</body>
</html>
```

55. login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title>Login</title>
  <link th:replace="imports" />
  <script th:src="@{/js/login.js}"></script>
</head>
<body role="document">
  <div class="container login-form-container">
    <div class="panel panel-default" >
      <div class="panel-heading">
        <div class="panel-title text-center">
    Hello!</div>
      </div>

      <div class="panel-body" >
        <div class="alert alert-danger" role="
            alert" th:if="${error}">
          <span class="glyphicon glyphicon-
              remove" aria-hidden="true"></span
              > Wrong credentials
        </div>

        <form role="form" id="loginForm" th:
    action="${login.href}" method="post">
          <div class="form-group">
            <div class="input-group">
              <span class="input-group-addon"><i
     class="glyphicon glyphicon-user"></i></
    span>
              <input id="username" type="text"
     class="form-control" name="username"
     placeholder="Username" />
            </div>
          </div>

          <div class="form-group">
            <div class="input-group">
              <span class="input-group-addon"><i
     class="glyphicon glyphicon-lock"></i></
    span>
              <input id="password" type="
    password" class="form-control" name="
    password" placeholder="Password" />
            </div>
          </div>

          <div class="form-group">
            <label class="checkbox-inline">
              <input type="checkbox" name="
              remember-me" th:text="|Remember me|
              " />
            </label>
          </div>

          <div class="form-group">
            <div class="col-sm-12 controls">
              <button type="submit" class="btn
    btn-primary pull-right" id="submitBtn"><i
    class="glyphicon glyphicon-log-in"></i>
    Sign in</button>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</body>
</html>
```

56. users_view_all.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
```

```
  <meta http-equiv="content-type" content="text/
      html; charset=utf-8" />
  <title>View Users</title>
  <link th:replace="imports" />
  <link th:replace="datatableImports" />
</head>
<body role="document">
  <div th:replace="navbar :: navbar(role='__${
      currentUser.getRole().toString()}__',
      name='__${currentUser.getName()}__')"></
      div>

  <div class="container">
    <div class="alert alert-success" role="alert
        " th:if="${successMessage}">
      <span class="glyphicon glyphicon-ok" aria-
          hidden="true"></span> <span th:text="
          ${successMessage}"></span>
    </div>

    <div class="text-right">
      <a th:href="${createUser.href}" class="btn
          btn-success btn-lg">
        <span class="glyphicon glyphicon-plus"
            aria-hidden="true"></span> Create
            User
      </a>
    </div><br /><br /><br />

    <table class="table table-bordered table-
        striped" id="myDataTable">
      <thead>
        <tr>
          <th>Username</th>
          <th>Name</th>
          <th>Designation</th>
          <th>Institution</th>
          <th>Email</th>
          <th>Role</th>
          <th>Status</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="user : ${users}" th:object=
            "${user}">
          <td><a th:href="${user.getLink('self')
              .href}" th:text="*{username}"></a
              ></td>
          <td><p th:text="*{name}" /></td>
          <td><p th:text="*{designation}" /></td
              >
          <td><p th:text="*{institution}" /></td
              >
          <td><p th:text="*{email}" /></td>
          <td><p th:text="*{role.toString()}"
              /></td>
          <td><p th:text="*{isActive()} ? '
              active' : deactivated" /></td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

57. user_view.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title>View User</title>
  <link th:replace="imports" />
</head>
<body role="document">
  <th:block th:if="${currentUser}">
    <div th:replace="navbar :: navbar(role='__${
        currentUser.getRole().toString()}__',
        name='__${currentUser.getName()}__')"
        ></div>
  </th:block>
  <th:block th:if="${currentUser == null}">
    <div th:replace="navbar :: navbar(role='',
        name='')"></div>
  </th:block>

  <div class="container">
    <div class="alert alert-success" role="alert
        " th:if="${successMessage}">
      <span class="glyphicon glyphicon-ok" aria-
          hidden="true"></span> <span th:text="
          ${successMessage}"></span>
</div>

<div class="text-center">
  <div class="h1">
    <span th:text="${user.institution}"></
        span>
  </div><br />

  <p>
    <span class="glyphicon glyphicon-tag"
        aria-hidden="true"></span>  
    <span th:text="|${user.role.toString()
        }:|"></span>
    <strong th:text="${user.name}"></strong>
  </p>
  <p th:text="${user.designation}" />
  <p><span class="glyphicon glyphicon-
      envelope" aria-hidden="true"></span>
        <span th:text="${user.email}"
      ></span></p>
  <p th:text="|Account status: ${user.
      isActive() ? 'Active' : 'Deactivated
      '}|" /><br /><br />

  <h4>
    <a th:if="${user.getLink('surveys')}" th
        :href="${user.getLink('surveys').
        href}">
      <span class="glyphicon glyphicon-th-
          list" aria-hidden="true"></span>
           
      <th:block th:if="${user.role.toString
          () == 'Research leader'}">View
          Created Surveys</th:block>
      <th:block th:if="${user.role.toString
          () == 'Contributor'}">View
          Answered Surveys</th:block>
    </a>
  </h4>

  <br /><br /><br />

  <th:block th:if="${currentUser}">
    <div class="buttonsWrapper" th:if="${
        currentUser.getRole().toString() ==
        'Admin' or user.username ==
        currentUser.user.username}">
      <div class="buttons">
        <a th:href="${user.getLink('update')
            .href}" th:if="${user.isActive
            ()}" class="btn btn-info btn-sm
            ">
          <span class="glyphicon glyphicon-
              pencil" aria-hidden="true"></
              span> Edit
        </a>
      </div>

      <div th:if="${user.isActive() and (
          currentUser.getRole().toString()
          == 'Admin' or user.username ==
          currentUser.username)}" class="
          buttons">
        <form th:action="${user.getLink('
            toggleStatus').href}" id="
            deactivateUserForm" th:method="
            put">
          <button type="button" class="btn
              btn-warning btn-sm"
            data-toggle="modal" data-target=
                "#confirmActionModal" data-
                action-name="deactivate"
            data-message="Are you sure you
                want to deactivate this
                user?">
            <span class="glyphicon glyphicon
                -ban-circle" aria-hidden="
                true"></span> Deactivate
          </button>
        </form>
      </div>

      <div th:if="${!user.isActive() and
          currentUser.getRole().toString()
          == 'Admin'}" class="buttons">
        <form th:action="${user.getLink('
            toggleStatus').href}" id="
            activateUserForm" th:method="
            put">
```

```
                <button type="button" class="btn
                    btn-success btn-sm"
                  data-toggle="modal" data-target=
                      "#confirmActionModal" data-
                      action-name="activate"
                  data-message="Are you sure you
                      want to activate this user?
                      ">
                  <span class="glyphicon glyphicon
                      -ok" aria-hidden="true"></
                      span> Activate
                </button>
              </form>
            </div>

            <div class="buttons" th:if="${
                currentUser.getRole().toString()
                == 'Admin'}">
              <form th:if="${user.getLink('delete
                  ')}" th:action="${user.getLink
                  ('delete').href}" th:method="
                  delete">
                <button type="button" class="btn
                    btn-danger btn-sm"
                  data-toggle="modal" data-target=
                      "#confirmActionModal" data-
                      action-name="delete"
                  data-message="Are you sure you
                      want to delete this user?">
                  <span class="glyphicon glyphicon
                      -remove" aria-hidden="true"
                      ></span> Delete
                </button>
              </form>
            </div>
          </div>
        </th:block>
      </div>
    </div>

    <div th:replace="confirm_modal :: modal"></div
        >
  </body>
  </html>
```

58. user_surveys.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title th:text="|${username} - Surveys|"></
      title>
  <link th:replace="imports" />
  <link th:replace="datatableImports" />
</head>
<body role="document">
  <div th:replace="navbar :: navbar(role='__${
      currentUser.getRole().toString()}__',
      name='__${currentUser.getName()}__')"></
      div>

  <div class="container">
    <h2 th:if="${role == 'CONTRIBUTOR'}">Surveys
        Answered</h2>
    <h2 th:if="${role == 'RESEARCH_LEADER'}">
        Surveys Created</h2>

    <div th:replace="surveys_table :: surveys"
        ></div>
  </div>
</body>
</html>
```

59. user_create_form.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title>Create User</title>
  <link th:replace="imports" />
  <script th:src="@{/js/user_form.js}"></script>
</head>
<body role="document">

<div th:replace="navbar :: navbar(role='__${
    currentUser.getRole().toString()}__',
    name='__${currentUser.getName()}__')"></
    div>

<div class="container">
  <div class="form-control-static col-sm-
      offset-2">
    <h1>Create User</h1>
  </div><br />

  <form id="userForm" th:method="${method}"
      role="form" class="form-horizontal" th:
      action="${user.getLink('submit').href}"
       th:object="${user}">
    <input type="hidden" th:value="*{userId}"
        name="userId" id="userId" />

    <div class="form-group">
      <label for="username" class="control-
          label col-sm-4">Username</label>
      <div class="col-sm-6">
        <input type="text" class="form-control
            " th:field="*{username}" />
        <span th:if="${#fields.hasErrors('
            username')}" class="error" th:
            errors="*{username}"></span><br
            />
      </div>
    </div>

    <div class="form-group">
      <label for="password" class="control-
          label col-sm-4">Password</label>
      <div class="col-sm-6">
        <input type="password" class="form-
            control" th:field="*{password}"/>
        <span th:if="${#fields.hasErrors('
            password')}" class="error" th:
            errors="*{password}"></span><br
            />
      </div>
    </div>

    <div class="form-group">
      <label for="retypedPassword" class="
          control-label col-sm-4">Re-type
          Password</label>
      <div class="col-sm-6">
        <input type="password" class="form-
            control" name="retypedPassword"
            id="retypedPassword" />
        <span th:if="${#fields.hasErrors('
            retypedPassword')}" class="error"
             th:errors="*{retypedPassword}"
            ></span><br />
      </div>
    </div>

    <div class="form-group">
      <label for="name" class="control-label
          col-sm-4">Name of Representative</
          label>
      <div class="col-sm-6">
        <input type="text" class="form-control
            " th:field="*{name}"/>
        <span th:if="${#fields.hasErrors('name
            ')}" class="error" th:errors="*{
            name}"></span><br />
      </div>
    </div>

    <div class="form-group">
      <label for="designation" class="control-
          label col-sm-4">Designation</label>
      <div class="col-sm-6">
        <input type="text" class="form-control
            " th:field="*{designation}"/>
        <span th:if="${#fields.hasErrors('
            designation')}" class="error" th:
            errors="*{designation}"></span><
            br />
      </div>
    </div>

    <div class="form-group">
      <label for="institution" class="control-
          label col-sm-4">Institution</label>
      <div class="col-sm-6">
        <input type="text" class="form-control
            " th:field="*{institution}"/>
        <span th:if="${#fields.hasErrors('
```

```
                institution')}" class="error" th:
                errors="*{institution}"></span><
                br />
            </div>
          </div>

          <div class="form-group">
            <label for="email" class="control-label
                col-sm-4">Email</label>
            <div class="col-sm-6">
              <input type="email" class="form-
                  control" th:field="*{email}"/>
              <span th:if="${#fields.hasErrors('
                  email')}" class="error" th:errors
                  ="*{email}"></span><br />
            </div>
          </div>

          <div class="form-group">
            <label for="role" class="control-label
                col-sm-4">Role</label>
            <div class="col-sm-6">
              <select id="role" th:field="*{role}"
                  class="form-control">
                <option th:each="role : ${roles}" th
                    :value="${#strings.toUpperCase(
                    role)}" th:text="${role.text}"
                    ></option>
              </select>
              <span th:if="${#fields.hasErrors('role
                  ')}" class="error" th:errors="*{
                  role}"></span><br />
            </div>
          </div><br /><br />

          <button type="submit" class="btn btn-
              primary col-sm-2 col-sm-offset-7" id=
              "submitBtn">Submit</button> 
          <button type="reset" class="btn btn-
              default col-sm">Reset</button> 
          <a th:href="${user.getLink('cancel').href}
              "><button type="button" class="btn
              btn-default col-sm">Cancel</button></
              a>
        </form>
      </div>
    </body>
    </html>
```

60. user_update_form.html

```
    <!DOCTYPE html>
    <html xmlns:th="http://www.thymeleaf.org">
    <head>
      <meta charset="ISO-8859-1"></meta>
      <title>Update User</title>
      <link th:replace="imports" />
      <script src="@{/js/user_form.js}"></script>
    </head>
    <body role="document">
      <div th:replace="navbar :: navbar(role='__${
          currentUser.getRole().toString()}__',
          name='__${currentUser.getName()}__')"></
          div>

      <div class="container">
        <span th:if="${hasChangePasswordError}" id="
            hasChangePasswordError"></span>
        <div role="tabpanel">
          <ul class="nav nav-tabs" role="tablist">
            <li role="presentation" class="active"
              id="accountTab"><a href="#account"
              aria-controls="account" role="tab"
              data-toggle="tab">Account</a></li>
            <li role="presentation" id="passwordTab"
              ><a href="#password" aria-controls="
              password" role="tab" data-toggle="tab
              ">Password</a></li>
          </ul>

          <div class="tab-content">
            <div role="tabpanel" class="tab-pane
              fade in active" id="account">
              <br /><br />

              <form id="userForm" th:method="${
                  method}" role="form" class="form-
                  horizontal" th:action="${user.
                  getLink('submit').href}" th:
                  object="${user}">
                <input type="hidden" th:field="*{
                    userId}" />
                <input type="hidden" th:field="*{
                    role}" />

                <div class="form-group">
                  <label for="username" class="
                      control-label col-sm-4">
                      Username</label>
                  <div class="col-sm-6">
                    <input type="text" class="form-
                        control" th:field="*{
                        username}" />
                    <span th:if="${#fields.hasErrors
                        ('username')}" class="error
                        " th:errors="*{username}"
                        ></span><br />
                  </div>
                </div>

                <div class="form-group">
                  <label for="name" class="control-
                      label col-sm-4">Name</label>
                  <div class="col-sm-6">
                    <input type="text" class="form-
                        control" th:field="*{name}"
                        />
                    <span th:if="${#fields.hasErrors
                        ('name')}" class="error" th
                        :errors="*{name}"></span><
                        br />
                  </div>
                </div>

                <div class="form-group">
                  <label for="designation" class="
                      control-label col-sm-4">
                      Designation</label>
                  <div class="col-sm-6">
                    <input type="text" class="form-
                        control" th:field="*{
                        designation}"/>
                    <span th:if="${#fields.hasErrors
                        ('designation')}" class="
                        error" th:errors="*{
                        designation}"></span><br />
                  </div>
                </div>

                <div class="form-group">
                  <label for="institution" class="
                      control-label col-sm-4">
                      Institution</label>
                  <div class="col-sm-6">
                    <input type="text" class="form-
                        control" th:field="*{
                        institution}"/>
                    <span th:if="${#fields.hasErrors
                        ('institution')}" class="
                        error" th:errors="*{
                        institution}"></span><br />
                  </div>
                </div>

                <div class="form-group">
                  <label for="email" class="control-
                      label col-sm-4">Email</label>
                  <div class="col-sm-6">
                    <input type="email" class="form-
                        control" th:field="*{email}
                        "/>
                    <span th:if="${#fields.hasErrors
                        ('email')}" class="error"
                        th:errors="*{email}"></span
                        ><br />
                  </div>
                </div><br /><br />

                <button type="submit" class="btn btn
                    -primary col-sm-2 col-sm-offset
                    -7" id="submitBtn">Submit</
                    button> 
                <button type="reset" class="btn btn-
                    default col-sm">Reset</button>&
                    nbsp;
                <a th:href="${user.getLink('cancel')
                    .href}"><button type="button"
                    class="btn btn-default col-sm">
                    Cancel</button></a>
              </form>
            </div>
            <div role="tabpanel" class="tab-pane
```

```
 fade" id="password">
<br /><br />

<form id="passwordForm" th:method="${
     method}" role="form" class="form-
     horizontal" th:action="${user.
     getLink('submit').href}" th:
     object="${user}">
  <input type="hidden" th:field="*{
       userId}" />
  <input type="hidden" th:field="*{
       username}" />
  <input type="hidden" th:field="*{
       name}" />
  <input type="hidden" th:field="*{
       designation}" />
  <input type="hidden" th:field="*{
       institution}" />
  <input type="hidden" th:field="*{
       email}" />
  <input type="hidden" th:field="*{
       role}" />

  <div class="form-group">
    <label for="password" class="
         control-label col-sm-4">
         Current Password</label>
    <div class="col-sm-6">
      <input type="password" class="
           form-control" th:field="*{
           oldPassword}"/>
      <span th:if="${#fields.hasErrors
           ('oldPassword')}" class="
           error" th:errors="*{
           oldPassword}"></span><br />
    </div>
  </div>

  <div class="form-group">
    <label for="password" class="
         control-label col-sm-4">New
         Password</label>
    <div class="col-sm-6">
      <input type="password" class="
           form-control" th:field="*{
           password}"/>
      <span th:if="${#fields.hasErrors
           ('password')}" class="error
           " th:errors="*{password}"
           ></span><br />
    </div>
  </div>

  <div class="form-group">
    <label for="retypedPassword" class
         ="control-label col-sm-4">Re-
         type New Password</label>
    <div class="col-sm-6">
      <input type="password" class="
           form-control" name="
           retypedPassword" id="
           retypedPassword" />
      <span th:if="${#fields.hasErrors
           ('retypedPassword')}" class
           ="error" th:errors="*{
           retypedPassword}"></span><
           br />
    </div>
  </div><br /><br />

  <button type="submit" class="btn btn
       -primary col-sm-2 col-sm-offset
       -7" id="submitBtn">Submit</
       button> 
  <button type="reset" class="btn btn-
       default col-sm">Reset</button>&
       nbsp;
  <a th:href="${user.getLink('cancel')
       .href}"><button type="button"
       class="btn btn-default col-sm">
       Cancel</button></a>
</form>
</div>
</div>
</div>
</div>
</body>
</html>
```

61. surveys_view_all.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title>View Surveys</title>
  <link th:replace="imports" />
  <link th:replace="datatableImports" />
</head>
<body role="document">
  <th:block th:if="${currentUser}">
    <div th:replace="navbar :: navbar(role='__${
         currentUser.getRole().toString()}__',
         name='__${currentUser.getName()}__')"
         ></div>
  </th:block>
  <th:block th:if="${currentUser == null}">
    <div th:replace="navbar :: navbar(role='',
         name='')"></div>
  </th:block>

  <div class="container">
    <div class="alert alert-success" role="alert
         " th:if="${successMessage}">
      <span class="glyphicon glyphicon-ok" aria-
           hidden="true"></span> <span th:text="
           ${successMessage}"></span>
    </div>

    <div class="text-right">
      <a th:href="${createSurvey.href}" class="
           btn btn-success btn-lg" th:if="${
           createSurvey != null and currentUser.
           active == true}">
        <span class="glyphicon glyphicon-plus"
             aria-hidden="true"></span> Create
             Survey
      </a>
      <a class="btn btn-success btn-lg disabled"
           th:if="${createSurvey != null and
           currentUser.active == false}">
        <span class="glyphicon glyphicon-plus"
             aria-hidden="true"></span> Create
             Survey
      </a>
    </div><br /><br /><br />

    <div th:replace="surveys_table :: surveys"
         ></div>
  </div>
</body>
</html>
```

62. surveys_table.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
</head>
<body>
  <div th:fragment="surveys">
    <table class="table table-bordered table-
         striped" id="myDataTable">
      <thead>
        <tr>
          <th>ID</th>
          <th>Title</th>
          <th>Start Date</th>
          <th>End Date</th>
          <th>Creator</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="survey : ${surveys}" th:
             object="${survey}">
          <td><a th:href="*{getLink('self').href
               }" th:text="*{surveyId}"></a></td
               >
          <td><a th:href="*{getLink('self').href
               }" th:text="*{title}"></a></td>
          <td><p th:text="*{startDate}" /></td>
          <td><p th:text="*{endDate}" /></td>
          <td><a th:href="*{getLink('creator').
               href}" th:text="*{
               institutionOfCreator}"></a></td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

63. survey_view.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title>View Survey</title>
  <link th:replace="imports" />
</head>
<body role="document">
  <th:block th:if="${currentUser}">
    <div th:replace="navbar :: navbar(role='__${
        currentUser.getRole().toString()}__',
        name='__${currentUser.getName()}__')"
        ></div>
  </th:block>
  <th:block th:if="${currentUser == null}">
    <div th:replace="navbar :: navbar(role='',
        name='')"></div>
  </th:block>

  <div class="container">
    <div class="alert alert-success" role="alert
        " th:if="${successMessage}">
      <span class="glyphicon glyphicon-ok" aria-
          hidden="true"></span> <span th:text="
          ${successMessage}"></span>
    </div>

    <table th:object="${survey}" class="table">
      <tr>
        <th>ID</th>
        <td><p th:text="*{surveyId}" /></td>
      </tr>
      <tr>
        <th>Created by</th>
        <td><p><a th:href="*{getLink('creator').
            href}" th:text="*{
            institutionOfCreator}"></a></p></td
            >
      </tr>
      <tr>
        <th>Title</th>
        <td><p th:text="*{title}" /></td>
      </tr>
      <tr>
        <th>Description / Message</th>
        <td><p th:text="*{description}" /></td>
      </tr>
      <tr>
        <th>Start Date</th>
        <td><p th:text="*{getReadableStartDate()
            }" /></td>
      </tr>
      <tr>
        <th>End Date</th>
        <td><p th:text="*{getReadableEndDate()}"
             /></td>
      </tr>
      <tr>
        <th>Status</th>
        <td><p th:text="*{getStatus()}" /></td>
      </tr>
    </table>


    <div class="buttons-left">
      <a class="btn btn-info btn-sm" th:if="${
          survey.getLink('update') != null and
          currentUser.active == true}" th:href=
          "${survey.getLink('update').href}">
        <span class="glyphicon glyphicon-pencil"
            aria-hidden="true"></span> Edit
      </a>
      <a class="btn btn-info btn-sm disabled" th
          :if="${survey.getLink('update') !=
          null and currentUser.active == false}
          ">
        <span class="glyphicon glyphicon-pencil"
            aria-hidden="true"></span> Edit
      </a>
    </div>

    <div class="buttons-left">
      <a class="btn btn-primary btn-sm" th:if="$
          {survey.getLink('answer') != null and
          currentUser.active == true}" th:href
          ="${survey.getLink('answer').href}">
        <span class="glyphicon glyphicon-comment
            " aria-hidden="true"></span> Answer
      </a>
      <a class="btn btn-primary btn-sm disabled"
          th:if="${survey.getLink('answer') !=
          null and currentUser.active == false
          }">
        <span class="glyphicon glyphicon-comment
            " aria-hidden="true"></span> Answer
      </a>
    </div>

    <div class="buttons-left">
      <a class="btn btn-info btn-sm" th:if="${
          survey.getLink('results')}" th:href="
          ${survey.getLink('results').href}">
        <span class="glyphicon glyphicon-list"
            aria-hidden="true"></span> Results
      </a>
    </div>

    <div class="buttons-left">
      <form th:if="${survey.getLink('deactivate
          ')}" th:action="${survey.getLink('
          deactivate').href}" th:method="put"
          id="deactivateSurvey">
        <button type="button" class="btn btn-
            warning btn-sm" th:if="${survey.
            getLink('deactivate')}"
          data-toggle="modal" data-target="#
              confirmActionModal" data-action-
              name="deactivate"
          data-message="Are you sure you want to
               deactivate this survey?"
          th:classappend="${currentUser.active
              == false ? 'disabled' : '' }">
          <span class="glyphicon glyphicon-ban-
              circle" aria-hidden="true"></span
              > Deactivate
        </button>
      </form>
    </div>

    <div class="buttons-left">
      <form th:if="${survey.getLink('delete')}"
          th:action="${survey.getLink('delete')
          .href}" th:method="delete">
        <button type="button" class="btn btn-
            danger btn-sm"
          data-toggle="modal" data-target="#
              confirmActionModal" data-action-
              name="delete"
          data-message="Are you sure you want to
               delete this survey?"
          th:classappend="${currentUser.active
              == false ? 'disabled' : '' }">
          <span class="glyphicon glyphicon-
              remove" aria-hidden="true"></span
              > Delete
        </button>
      </form>
    </div>
  </div>


  <div th:replace="confirm_modal :: modal"></div
      >
</body>
</html>
```

64. survey_form.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta http-equiv="Content-Type" content="text/
      html; charset=UTF-8" />
  <title th:text="${title}"></title>
  <link th:replace="imports" />
  <link rel="stylesheet" th:href="@{/webjars/
      jquery-ui-themes/1.11.4/smoothness/jquery
      -ui.css}" />
  <script th:src="@{/webjars/jquery-ui/1.11.4/
      jquery-ui.js}"></script>
  <script th:src="@{/js/survey_form.js}"></
      script>
</head>
<body role="document">
  <div th:replace="navbar :: navbar(role='__${
      currentUser.getRole().toString()}__',
      name='__${currentUser.getName()}__')"></
      div>
```

```
<div class="container">
  <div class="form-control-static col-sm-
       offset-1">
    <h1 th:text="${title}"></h1>
  </div><br />

  <form id="surveyForm" th:method="${method}"
       role="form" class="form-horizontal" th:
       action="${survey.getLink('submit').href
       }" th:object="${survey}">
    <input type="hidden" th:value="*{creator}"
         th:field="*{creator}" />

    <div class="form-group">
      <label for="title" class="control-label
           col-sm-4">Title:</label>
      <div class="col-sm-5">
        <input type="text" class="form-control
             " th:field="*{title}" />
        <span th:if="${#fields.hasErrors('
             title')}" class="error" th:errors
             ="*{title}"></span><br />
      </div>
    </div>

    <div class="form-group">
      <label for="description" class="control-
           label col-sm-4">Description /
           Message:</label>
      <div class="col-sm-5">
        <textarea class="form-control" rows="5
             " th:field="*{description}"></
             textarea>
        <span th:if="${#fields.hasErrors('
             description')}" class="error" th:
             errors="*{description}"></span><
             br />
      </div>
    </div>

    <div class="form-group">
      <label for="startDate" class="control-
           label col-sm-4">Start Date (yyyy-MM
           -dd):</label>
      <div class="col-sm-5 input-group">
        <input type="text" class="form-control
             datepicker" th:field="*{
             startDate}" autocomplete="off" />
        <span th:if="${#fields.hasErrors('
             startDate')}" class="error" th:
             errors="*{startDate}"></span><br
             />
      </div>
    </div>

    <div class="form-group">
      <label for="endDate" class="control-
           label col-sm-4">End Date (yyyy-MM-
           dd):</label>
      <div class="col-sm-5 input-group">
        <input type="text" class="form-control
             datepicker" th:field="*{endDate}
             " autocomplete="off" />
        <span th:if="${#fields.hasErrors('
             endDate')}" class="error" th:
             errors="*{endDate}"></span><br />
      </div>
    </div>

    <div class="questionsWrapper">
      <h5 class="form-control-static col-sm-
           offset-1"><i>Enter at least 1
           question:</i></h5>
      <div class="question" th:each="q : *{
           questions}">
        <input type="hidden" th:id="|
             questions${qStat.index}.id|" th:
             name="|questions[${qStat.index}].
             id|" th:value="${q.id}" />
        <input type="hidden" th:id="|
             questions${qStat.index}.surveyId|
             " th:name="|questions[${qStat.
             index}].surveyId|" th:value="*{
             surveyId}" />
        <input type="hidden" th:id="|
             questions${qStat.index}.
             questionNumber|" th:name="|
             questions[${qStat.index}].
             questionNumber|" th:value="${
             qStat.count}" />
        <div class="form-group">
          <label class="control-label col-sm-3
               " th:text="|Question ${qStat.
               count}: |" th:for="|questions${
               qStat.index}.question|"></label
               >
          <div class="input-group col-sm-7">
            <span class="input-group-btn">
              <button type="button" name="
               removeQuestion" class="btn btn-
               default removeQuestionBtn" th:
               value="${qStat.index}" th:if="$
               {qStat.size > 1}">
                <span class="glyphicon
                     glyphicon-remove" aria-
                     hidden="true"></span>
              </button>
            </span>
            <input type="text" class="form-
               control" th:field="*{questions[
               __${qStat.index}__].question}"
               />
            <span th:if="${#fields.hasErrors('
                 questions[__${qStat.index}__
                 ].question')}" class="error"
                 th:errors="*{questions[__${
                 qStat.index}__].question}"></
                 span><br />
          </div>
        </div>

        <div class="form-group">
          <label class="control-label col-sm-3
               ">Input Type:</label>
          <div class="inputType col-sm-9">
            <label class="radio-inline" th:
                 each="inputType : ${
                 inputTypes}">
              <input autocomplete="off" type="
                   radio" th:field="*{
                   questions[__${qStat.index}
                   __].inputType}" th:value="$
                   {inputType}" th:text="${#
                   strings.toLowerCase(
                   inputType)}" />
            </label>
            <span th:if="${#fields.hasErrors('
                 questions[__${qStat.index}__
                 ].inputType')}" class="error"
                  th:errors="*{questions[__${
                 qStat.index}__].inputType}"
                 ></span><br />
          </div>
        </div>

        <div class="choicesWrapper">
          <h5 class="form-control-static col-
               sm-offset-2" th:if="${#lists.
               size(survey.questions[__${qStat
               .index}__].choices) > 0}"><i>
               Enter at least 2 choices:</i></
               h5>
          <div class="choice form-group" th:
               each="choice : *{questions[__${
               qStat.index}__].choices}">
            <label class="control-label col-sm
                 -3" th:for="|questions${qStat
                 .index}.choices${choiceStat.
                 index}|">Choice:</label>
            <div class="input-group col-sm-5">
              <span class="input-group-btn">
                <button type="button" name="
                     removeChoice" class="btn
                     btn-default
                     removeChoiceBtn" th:value
                     ="${choiceStat.index}" th
                     :if="${choiceStat.size >
                     2}">
                  <span class="glyphicon
                       glyphicon-remove" aria-
                       hidden="true"></span>
                </button>
              </span>
              <input type="text" class="form-
                   control" th:field="*{
                   questions[__${qStat.index}
                   __].choices[__${choiceStat.
                   index}__]}" />
              <span th:if="${#fields.hasErrors
                   ('questions[__${qStat.index
                   }__].choices[__${choiceStat
                   .index}__]')}" class="error
```

```
                    " th:errors="*{questions[
                    __${qStat.index}__].choices
                    [__${choiceStat.index}__]}"
                    ></span><br />
                </div>
              </div>
            </div>
            <button type="button" name="addChoice"
                  class="btn btn-sm btn-default
                  addChoiceBtn col-sm-offset-2">
              <span class="glyphicon glyphicon-
                  plus" aria-hidden="true"></span
                  > Add Choice
            </button>

            <div class="form-group resultsWrapper"
                >
              <label class="control-label col-sm-3
                  ">Results to compute: </label>
              <div class="results col-sm-9">
                <label class="checkbox-inline" th:
                    each="function : ${functions}
                    ">
                  <input type="checkbox" th:field=
                      "*{questions[__${qStat.
                      index}__].functions}" th:
                      value="${function}" th:text
                      ="${#strings.toLowerCase(
                      function)}" />
                </label>
                <span class="error" th:errors="*{
                    questions[__${qStat.index}__
                    ].functions}"><br /></span>
              </div>
            </div>
          </div>
        </div>
        <br />
        <button type="button" name="addQuestion"
            class="btn btn-default addQuestionBtn
              col-sm-offset-1 col-sm-2">
          <span class="glyphicon glyphicon-plus"
              aria-hidden="true"></span> Add
              Question
        </button>
        <br /><br /><br /><br />

        <button type="submit" class="btn btn-
            primary col-sm-2 col-sm-offset-8" id=
            "submitBtn">Submit</button> 
        <a th:href="${survey.getLink('cancel').
            href}"><button type="button" class="
            btn btn-default col-sm">Cancel</
            button></a>
      </form>
    </div>
  </body>
  </html>
```

65. survey_answer.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta http-equiv="Content-Type" content="text/
      html; charset=UTF-8" />
  <title>Answer Survey</title>
  <link th:replace="imports" />
  <script th:src="@{/js/survey_answer.js}"></
      script>
</head>
<body role="document">
  <div th:replace="navbar :: navbar(role='__${
      currentUser.getRole().toString()}__',
      name='__${currentUser.getName()}__')"></
      div>

  <div class="container">
    <div class="form-control-static text-center"
        >
      <h1 th:text="${survey.title}"></h1>
    </div><br />

    <div class="well">
      <p th:text="${survey.description}" />
    </div>

    <div id="alert-container"></div>

    <form method="post" id="answerSurveyForm"
        role="form" th:action="${surveyAnswer.
        getLink('submit').href}" th:object="${
        surveyAnswer}" >
      <input type="hidden" th:value="${survey.
          surveyId}" th:field="*{surveyId}" />
      <input type="hidden" th:value="*{
          contributor}" th:field="*{contributor
          }" />

      <div class="question" th:each="q : ${
          survey.questions}">

        <div class="form-group" th:if="${q.
            inputType.toString() == 'NUMBER'}">
          <label th:text="|${qStat.count}. ${q.
              question}|" th:for="|answers${
              qStat.index}|"></label>
          <div class="row">
            <div class="col-sm-2">
              <input type="number" min="0" class
                  ="form-control" th:field="*{
                  answers[__${qStat.index}__]}"
                  />
            </div>
            <span class="errorMessage" th:errors
                ="*{answers[__${qStat.index}__
                ]}"><br /></span>
          </div>
        </div>

        <div class="form-group" th:if="${q.
            inputType.toString() == 'RADIO'}">
          <label th:text="|${qStat.count}. ${q.
              question}|"></label>
          <div class="inputRadio">
            <div class="radio" th:each="choice :
                ${q.choices}" >
              <label>
                <input type="radio" th:text="${
                    choice}" th:value="${
                    choiceStat.count}" th:field
                    ="*{answers[__${qStat.index
                    }__]}" />
              </label>
            </div>
          </div>
          <span class="errorMessage" th:errors="
              *{answers[__${qStat.index}__]}"><
              br /></span>
        </div>

        <div class="form-group" th:if="${q.
            inputType.toString() == 'CHECKBOX'}
            ">
          <label th:text="|${qStat.count}. ${q.
              question}|"></label>
          <div class="inputCheckbox">
            <div class="checkbox" th:each="
                choice : ${q.choices}" >
              <label>
                <input type="checkbox" th:text="
                    ${choice}" th:value="${
                    choiceStat.count}" th:field
                    ="*{answers[__${qStat.index
                    }__]}" />
              </label>
            </div>
          </div>
          <span class="errorMessage" th:errors="
              *{answers[__${qStat.index}__]}"><
              br /></span>
        </div>

        <div class="form-group" th:if="${q.
            inputType.toString() == 'DROPDOWN'}
            ">
          <label th:text="|${qStat.count}. ${q.
              question}|" th:for="|answers${
              qStat.index}|"></label>
          <div class="row">
            <div class="col-sm-6">
              <select class="form-control" th:
                  field="*{answers[__${qStat.
                  index}__]}">
                <option value="" selected="
                    selected"></option>
                <option th:each="choice : ${q.
                    choices}" th:value="${
                    choiceStat.count}" th:text=
                    "${choice}" th:field="*{
                    answers[__${qStat.index}__
```

```
                        ]}"></option>
                    </select>
                  </div>
                  <span class="errorMessage" th:errors
                       ="*{answers[__${qStat.index}__
                       ]}"><br /></span>
                </div>
              </div>

            </div>
            <br /><br />
            <button type="submit" class="btn btn-
                 primary col-sm-2" id="submitFormBtn">
                 Submit</button> 
            <a th:href="${surveyAnswer.getLink('cancel
                 ').href}"><button type="button" class
                 ="btn btn-default">Cancel</button></a
                 >
          </form>
        </div>
      </body>
      </html>
```

66. survey_results.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta http-equiv="Content-Type" content="text/
       html; charset=UTF-8" />
  <title>Survey Results</title>
  <link th:replace="imports" />
</head>
<body role="document">
  <th:block th:if="${currentUser}">
    <div th:replace="navbar :: navbar(role='__${
         currentUser.getRole().toString()}__',
         name='__${currentUser.getName()}__')"
         ></div>
  </th:block>
  <th:block th:if="${currentUser == null}">
    <div th:replace="navbar :: navbar(role='',
         name='')"></div>
  </th:block>

  <div class="container">
    <div class="text-right">
      <a th:href="${survey.getLink('
           downloadResults').href}" class="btn
           btn-success btn-lg">
        <span class="glyphicon glyphicon-save"
             aria-hidden="true"></span> Download
              Results
      </a>
    </div>

    <h2><a th:href="${survey.getLink('self').
         href}" th:text="${survey.title}"></a></
         h2>
    <h4 th:text="|Creator: ${survey.
         institutionOfCreator}|"></h4>
    <h4 th:text="|Total Contributors: ${
         totalContributors}|"></h4><br />

    <div th:each="result : ${results}" th:object
         ="${result}" class="question">
      <h4 th:text="|*{questionNumber}. *{
           question}|"></h4>
      <div class="row" th:if="*{average} >= 0 or
           *{minimum} >= 0 or *{maximum} >= 0">
        <div class="col-sm-3">
          <div class="panel panel-default">
            <div class="panel-heading"><strong>
                 RESULTS</strong></div>
            <table class="table table-bordered">
              <tr th:if="*{average} >= 0">
                <th>Average</th>
                <td th:text="*{average}"></td>
              </tr>
              <tr th:if="*{minimum} >= 0">
                <th>Minimum</th>
                <td th:text="*{minimum}"></td>
              </tr>
              <tr th:if="*{maximum} >= 0">
                <th>Maximum</th>
                <td th:text="*{maximum}"></td>
              </tr>
            </table>
          </div>
        </div>
        <div class="row" th:if="*{tally.counts.
             size() > 0}">
          <div class="col-sm-6">
            <div class="panel panel-default">
              <div class="panel-heading"><strong>
                   TALLY</strong></div>
              <table class="table table-bordered">
                <tr>
                  <th>Choice</th>
                  <th>Count</th>
                </tr>
                <tr th:each="entry : *{tally.
                     counts}">
                  <td th:text="${entry.key}"></td>
                  <td th:text="${entry.value}"></
                       td>
                </tr>
                <tr>
                  <td>Minimum</td>
                  <td th:text="*{tally.
                       choicesWithMinCount}"></td>
                </tr>
                <tr>
                  <td>Maximum</td>
                  <td th:text="*{tally.
                       choicesWithMaxCount}"></td>
                </tr>
              </table>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
  </html>
```

67. datatableImports.html

```
<link rel="stylesheet" th:href="@{/webjars/
     datatables-plugins/9dcbecd42ad/integration/
     bootstrap/3/dataTables.bootstrap.css}" />
<script th:src="@{/webjars/datatables/1.10.6/js/
     jquery.dataTables.min.js}"></script>
<script th:src="@{/webjars/datatables-plugins/9
     dcbecd42ad/integration/bootstrap/3/
     dataTables.bootstrap.min.js}"></script>
<script th:src="@{/js/toDataTable.js}"></script>
```

68. confirm_modal.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
</head>
<body>
  <div th:fragment="modal">
    <div class="modal fade" id="
         confirmActionModal" tabindex="-1" role=
         "dialog" aria-labelledby="
         confirmActionModalLabel" aria-hidden="
         true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <button type="button" class="close"
                 data-dismiss="modal" aria-label
                 ="Close"><span aria-hidden="
                 true">&times;</span></button>
            <h4 class="modal-title" id="
                 confirmActionModalLabel">
                 Confirm Action</h4>
          </div>
          <div class="modal-body">
            ...
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn
                 -default" data-dismiss="modal">
                 Close</button>
            <button type="button" class="btn btn
                 -primary" id="doActionBtn" >
                 Continue</button>
          </div>
        </div>
```

```
      </div>
    </div>
  </div>
</body>
</html>
```

69. error.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="ISO-8859-1"></meta>
  <title th:text="${title}"></title>
  <link th:replace="imports" />
</head>
<body role="document">
  <th:block th:if="${currentUser}">
    <div th:replace="navbar :: navbar(role='__${
        currentUser.getRole().toString()}__',
        name='__${currentUser.getName()}__')"
        ></div>
  </th:block>
  <th:block th:if="${currentUser == null}">
    <div th:replace="navbar :: navbar(role='',
        name='')"></div>
  </th:block>

  <div class="container">
    <h1 th:text="${header}"></h1>

    <h2 th:text="${message}"></h2>

    <a th:href="${home.href}">Back to home</a><
        br />
  </div>
</body>
</html>
```

70. style.css

```
@font-face {
  font-family: 'Junction';
  src: url('Junction-webfont.eot');
  src: url('Junction-webfont.eot?#iefix') format
     ('embedded-opentype'),
     url('Junction-webfont.woff') format('woff')
     ,
     url('Junction-webfont.ttf') format('
     truetype'),
     url('Junction-webfont.svg#
     junction_regularregular') format('svg');
  font-weight: normal;
  font-style: normal;
}

body {
  font-family: 'Junction', sans-serif;
  padding-top: 70px;
  padding-bottom: 30px;
}

*[hidden] {
  display: none;
}

.btn-primary {
  background-color: rgb( 82, 111, 245);
  border-color: rgb( 82, 111, 245);
}

.panel {
  border-color: rgb( 82, 111, 245);
}

.panel-default > .panel-heading {
  background-color: rgb( 82, 111, 245);
  color: #fff;
}

.panel-title {
  color: #fff;
}

.navbar-default {
  background-color: rgba( 82, 111, 245, 0.85);
}

.navbar-default .navbar-brand {
  color: #fff;
}

.navbar-default .navbar-nav > li > a {
  color: #fff;
}

.login-form-container {
  max-width: 430px;
}

#loginForm > .form-group {
  margin: 0px 0px 25px;
}

.h1 {
  font-size: 30px;
  color: #09239d;
}

textarea {
  resize: none;
}

.label {
  padding-top: 5px;
  padding-bottom: 1px;
}

.question {
  margin-bottom: 30px;
}

.buttonsWrapper
{
  position:relative;
  float:left;
  left:50%;
}

.buttons
{
  position:relative;
  float:left;
  left:-50%;
  margin: 0 5px;
}

.buttons-left {
  float: left;
  margin: 0 5px;
}

#logoutBtn {
  background-color: transparent;
}

.logoutBtn-container:hover {
  background-color: #f5f5f5;
}
```

71. script.js

```
$(document).ready( function()
{
  /* Style active navbar link */
  var url = window.location;

  var element = $( "ul.nav a" ).filter( function
     ()
  {
    return url.href == this.href && ( this.href
     != ( url.origin + "/" ) );
  }).parent();

  if ( element.is( "li" ) )
  {
    element.addClass( "active" );
  }

  /* Confirm action modals */
  $( "#confirmActionModal" ).on( "show.bs.modal"
     , function( e )
  {
    var button = $( e.relatedTarget );
    var message = button.data( "message" );
    var action = button.data( "action-name" );
```

```
    var form = button.closest( "form" );
    var modal = $( this );

    modal.find( ".modal-body" ).text( message );
    modal.find( ".modal-footer #doActionBtn" ).
        on( "click", function()
    {
      var input = $( "<input>", { type: "hidden"
          , name: action, value: true });

      $( form ).append( $( input ) ).submit();
    });
  });

});
```

72. login.js

```
$(document).ready(function()
{
  /* Focus on username field */
  $( "#username" ).focus();

  /* Validate form */
  $( "#submitBtn" ).on( "click", function( e ){
    var hasErrors = false;

    // remove previous error messages
    $( "#loginForm" ).find( "h5 > span.label-
        danger" ).parent().remove();
    $( "#loginForm" ).find( ".form-group.has-
        error,.has-feedback" ).removeClass( "
        has-error has-feedback" );
    $( "#loginForm" ).find( ".form-control-
        feedback" ).remove();

    // username must be 2-20 characters long
    if( $( "#username" ).val().length < 2 || $(
        "#username" ).val().length > 20 )
    {
      viewError( $( "#username" ).parent(), "
          please fill out this field" );
      hasErrors = true;
    }

    // password must be > 2 characters
    if( $( "#password" ).is( ":password" ) && (
        $( "#password" ).val().length < 2 || $(
        "#password" ).val().length > 255 ) )
    {
      viewError( $( "#password" ).parent(), "
          please fill out this field" );
      hasErrors = true;
    }

    if( hasErrors )
    {
      e.preventDefault();
    }
  });

  /* When text field has error */
  function viewError( selector, message )
  {
    $( selector ).after( '<h5><span class="label
        label-danger">' + message + '</span></
        h5>' );
    $( selector ).closest( ".form-group" ).
        addClass( "has-error has-feedback" );
    $( selector ).after( '<span class="glyphicon
        glyphicon-remove form-control-feedback
        "></span>' );
  }
});
```

73. toDataTable.js

```
$(document).ready( function()
{
  $( "#myDataTable" ).dataTable();
});
```

74. user_form.js

```
$(document).ready(function()
{
  /* Set initial active tab */
  var a = $( "#hasChangePasswordError" ).length
  if( $( "#hasChangePasswordError" ).length > 0
      )
  {
    $( "#accountTab" ).removeClass( "active" );
    $( "#account" ).removeClass( "in active" );

    $( "#passwordTab" ).addClass( "active" );
    $( "#password" ).addClass( "in active" );
  }

  /* Focus on username field */
  $( "#username" ).focus();

  /* Validate form */
  $( "#submitBtn" ).on( "click", function( e ){
    var hasErrors = false;

    // remove previous error messages
    $( "#userForm" ).find( "h5 > span.label-
        danger" ).parent().remove();
    $( "#userForm" ).find( ".form-group.has-
        error,.has-feedback" ).removeClass( "
        has-error has-feedback" );
    $( "#userForm" ).find( ".form-control-
        feedback" ).remove();

    // username must be 2-20 characters long
    if( $( "#username" ).val().length < 2 || $(
        "#username" ).val().length > 20 )
    {
      viewError( "#username", "should be 2-20
          characters" );
      hasErrors = true;
    }

    // current password must be > 2 characters
    if( $( "#currentPassword" ).is( ":password"
        ) && ( $( "#currentPassword" ).val().
        length < 2 || $( "#currentPassword" ).
        val().length > 255 ) )
    {
      viewError( "#currentPassword", "should be
          greater than 2 characters" );
      hasErrors = true;
    }

    // password must be > 2 characters
    if( $( "#password" ).is( ":password" ) && (
        $( "#password" ).val().length < 2 || $(
        "#password" ).val().length > 255 ) )
    {
      viewError( "#password", "should be greater
          than 2 characters" );
      hasErrors = true;
    }

    // retyped password must be same as password
    if( $( "#retypedPassword" ).is( ":password"
        ) && ( $( "#retypedPassword" ).val() !=
        $( "#password" ).val() ) )
    {
      viewError( "#retypedPassword", "passwords
          do not match" );
      hasErrors = true;
    }

    // name must be 2-50 characters long
    if( $( "#name" ).val().length < 2 || $( "#
        name" ).val().length > 50 )
    {
      viewError( "#name", "should be 2-50
          characters" );
      hasErrors = true;
    }

    // designation must be 2-50 characters long
    if( $( "#designation" ).val().length < 2 ||
        $( "#designation" ).val().length > 50 )
    {
      viewError( "#designation", "should be 2-50
          characters" );
      hasErrors = true;
    }

    // institution must be 2-50 characters long
    if( $( "#institution" ).val().length < 2 ||
        $( "#institution" ).val().length > 50 )
    {
```

```
      viewError( "#institution", "should be 2-50
          characters" );
      hasErrors = true;
    }

    // email must be 2-30 characters long
    if( $( "#email" ).val().length < 2 || $( "#
        email" ).val().length > 30 )
    {
      viewError( "#email", "should be 2-30
          characters" );
      hasErrors = true;
    }

    // a role must be selected
    if( $( "#role" ).val().length <= 0 )
    {
      $( "#role" ).closest( ".form-group" ).
          append( '<h5 class="form-control-
          static"><span class="label label-
          danger">select one</span></h5>' );
      $( "#role" ).closest( ".form-group" ).
          addClass( "has-error" );
    }

    if( hasErrors )
    {
      e.preventDefault();

      /* Focus on first element with error */
      var withError = $( "#userForm" ).find( ".
          has-error:first" ).find( "input" );
      var navbarOffset = parseInt( $( "body" ).
          css( "padding-top" ) );
      $('html, body').animate(
      {
        scrollTop: ( withError.offset().top -
    navbarOffset )
      }, 'slow');
      withError.focus();
    }
  });

  /* When text field has error */
  function viewError( selector, message )
  {
    $( selector ).after( '<h5><span class="label
        label-danger">' + message + '</span></
        h5>' );
    $( selector ).closest( ".form-group" ).
        addClass( "has-error has-feedback" );
    $( selector ).after( '<span class="glyphicon
        glyphicon-remove form-control-feedback
        "></span>' );
  }
});
```

75. survey_form.js

```
$(document).ready(function()
{
  /* Set intial focus */
  var empty = $( "#surveyForm" ).find( "input[
      value='']" ).first();

  if( empty.size() > 0 )
  {
    var navbarOffset = parseInt( $( "body" ).css
        ( "padding-top" ) );
    var headerOffset = parseInt( $( "h1" ).
        parent().css( "height" ) );
    $('html, body').animate(
    {
      scrollTop: ( empty.offset().top -
    navbarOffset - headerOffset )
    }, 'slow');

    empty.focus();
  }
  else
  {
    $( "#title" ).focus();
  }

  /* Set initial state of form */
  $( ".questionsWrapper > .question" ).each(
      function()
  {
    var element = $(this).find( ":radio:checked"
        );

    if( element.size() == 0 || element.val().
        toUpperCase() == 'NUMBER' )
    {
      $(element).children( ".choicesWrapper" ).
          hide();
      $(this).children( ".addChoiceBtn" ).prop("
          disabled", true).hide();
    }

    /* Set initial states of checkboxes */
    if( element.size() > 0 && element.val().
        toUpperCase() == 'NUMBER' )
    {
      /* 'tally' is not valid */
      $(element).closest( ".form-group" ).
          siblings( ".resultsWrapper" ).find( "
          :checkbox" ).each( function()
      {
        if( $(this).val() == 'TALLY' )
        {
          $(this).prop( "checked", false );
          $(this).prop( "disabled", true );
        }
        else
        {
          $(this).prop( "disabled", false );
        }
      });
    }
    else
    {
      /* only 'tally' is valid */
      $(element).closest( ".form-group" ).
          siblings( ".resultsWrapper" ).find( "
          :checkbox" ).each( function()
      {
        if( $(this).val() != 'TALLY' )
        {
          $(this).prop( "disabled", true );
        }

      });
    }
  });

  /* Add aditional parameters of addQuestion to
      add a question */
  $( ".addQuestionBtn" ).on( "click", function()
      {
    var input = $( "<input>", { type: "hidden",
        name: "addQuestion" });

    $('#surveyForm').append( $( input ) ).submit
        ();
  });

  /* Add aditional parameter of questionIndex
      before sending request to remove a
      question */
  $( ".removeQuestionBtn" ).on( "click",
      function(){
    var questionIndex = this.value;

    var input = $( "<input>", { type: "hidden",
        name: "removeQuestion", value:
        questionIndex });

    $('#surveyForm').append( $( input ) ).submit
        ();
  });

  /* Add aditional parameters of addChoice and
      questionIndex to add a choice */
  $( ".addChoiceBtn" ).on( "click", function(){
    var questionIndex = $(this).closest( ".
        question" ).index( ".question" );

    var input = $( "<input>", { type: "hidden",
        name: "addChoice" });
    var qIndex = $( "<input>", { type: "hidden",
        name: "questionIndex", value:
        questionIndex });

    $('#surveyForm').append( $( input ) ).append
        ( $( qIndex ) ).submit();
  });

  /* Add aditional parameters of choiceIndex and
      questionIndex to remove a choice */
  $( ".removeChoiceBtn" ).on( "click", function
      (){
```

```
  var questionIndex = $(this).closest( ".
      question" ).index( ".question" );

  var cIndex = $( "<input>", { type: "hidden",
      name: "removeChoice", value: this.
      value });
  var qIndex = $( "<input>", { type: "hidden",
      name: "questionIndex", value:
      questionIndex });

  $('#surveyForm').append( $( cIndex ) ).
      append( $( qIndex ) ).submit();
});

/* When input type choice is changed,
 * 1. Toggle visibility of Add Choice button
 * 2. Empty or populate choicesWrapper
 */
$( ".questionsWrapper > .question").find( ":
    radio" ).on( "change", function(){
  if( $(this).val().toUpperCase() == "NUMBER"
      )
  {
    $(this).closest( ".form-group" ).siblings(
        ".choicesWrapper" ).empty();
    $(this).closest( ".form-group" ).siblings(
        ".addChoiceBtn" ).prop("disabled",
        true).hide();

    /* 'tally' is not valid */
    $(this).closest( ".form-group" ).siblings(
        ".resultsWrapper" ).find( ":checkbox
        " ).each( function()
    {
      if( $(this).val() == 'TALLY' )
      {
        $(this).prop( "checked", false );
        $(this).prop( "disabled", true );
      }
      else
      {
        $(this).prop( "disabled", false );
      }
    });
  }
  else
  {
    var wrapper = $(this).closest( ".form-
        group" ).siblings( ".choicesWrapper"
        );

    if( wrapper.children( ".choice" ).size()
        == 0 )
    {
      var questionIndex = wrapper.parent().
          index( ".question" );

      wrapper.append( '<h5 class="form-control
          -static col-sm-offset-2"><i>Enter
          at least 2 choices:</i></h5>' +
              '<div class="choice form-group">
                  ' +
                '<label class="control-label
                    col-sm-3" for="questions'
                     + questionIndex + '.
                    choices0">Choice:</label>
                    ' +
                '<div class="col-sm-5">' +
                  '<input type="text" class="
                      form-control" id="
                      questions' +
                      questionIndex + '.
                      choices0" name="
                      questions[' +
                      questionIndex + '].
                      choices[0]" /> ' +
                '</div>' +
              '</div>' +
              '<div class="choice form-group">
                  ' +
                '<label class="control-label
                    col-sm-3" for="questions'
                     + questionIndex + '.
                    choices1">Choice:</label>
                    ' +
                '<div class="col-sm-5">' +
                  '<input type="text" class="
                      form-control" id="
                      questions' +
                      questionIndex + '.
                      choices1" name="
                      questions[' +
                      questionIndex + '].
                      choices[1]" /> ' +
                '</div>' +
              '</div>' );
    }

    wrapper.show();
    $(this).closest( ".form-group" ).siblings(
        ".addChoiceBtn" ).prop("disabled",
        false).show();

    /* only 'tally' is valid */
    $(this).closest( ".form-group" ).siblings(
        ".resultsWrapper" ).find( ":checkbox
        " ).each( function()
    {
      if( $(this).val() == 'TALLY' )
      {
        $(this).prop( "disabled", false );
      }
      else
      {
        $(this).prop( "checked", false );
        $(this).prop( "disabled", true );
      }
    });
  }
});

/* Validate form */
$( "#submitBtn" ).on( "click", function( e ){
  var hasErrors = false;

  // remove previous error messages
  $( "#surveyForm" ).find( "h5 > span.label-
      danger" ).parent().remove();
  $( "#surveyForm" ).find( ".form-group.has-
      error,.has-feedback" ).removeClass( "
      has-error has-feedback" );
  $( "#surveyForm" ).find( ".form-control-
      feedback" ).remove();

  // survey title must be 2-50 characters long
  if( $( "#title" ).val().length < 2 || $( "#
      title" ).val().length > 50 )
  {
    viewError( "#title", "should be 2-50
        characters" );
    hasErrors = true;
  }

  // description must be 1-255 characters long
  if( $( "#description" ).val().length < 1 ||
      $( "#description" ).val().length > 255
      )
  {
    viewError( "#description", "should be
        1-255 characters" );
    hasErrors = true;
  }

  // start date should follow the correct
      format (yyyy-MM-dd), and be today or in
      the future
  var startDate = $( "#startDate" ).val();
  var yesterday = new Date();
  yesterday.setDate( yesterday.getDate() - 1 )
      ;
  if( !isValidDate( startDate ) )
  {
    viewError( "#startDate", "invalid date" );
    hasErrors = true;
  }
  else if( !
      isFirstParamDateBeforeSecondParamDate(
      yesterday, new Date(startDate) ) )
  {
    {
      viewError( "#startDate", "should be in
          the future" );
      hasErrors = true;
    }
  }

  // end date should follow the correct format
      (yyyy-MM-dd), be in the future, and
      after startDate
  var endDate = $( "#endDate" ).val();
  if( !isValidDate( endDate ) )
  {
    viewError( "#endDate", "invalid date" );
    hasErrors = true;
```

```
}
else if( !
    isFirstParamDateBeforeSecondParamDate(
    yesterday, new Date(endDate) ) )
{
  viewError( "#endDate", "should be in the
      future" );
  hasErrors = true;
}
else if( !
    isFirstParamDateBeforeSecondParamDate(
    new Date(startDate), new Date(endDate)
    ) )
{
  viewError( "#endDate", "should be after
      start date" );
  hasErrors = true;
}


// each question must be 2-255 characters
    long
$( ".questionsWrapper" ).children( ".
    question" ).each( function( index,
    element ){
  $(this).find( ":text" ).each(function(){
    if( $(this).attr("id") == ("questions" +
        index + ".question") )
    {
      if( $(this).val().length < 2 || $(this
          ).val().length > 255 )
      {
        $( this ).closest( ".form-group" ).
            append( '<h5 class="form-
            control-static col-sm-offset
            -2"><span class="label label-
            danger">should be 2-255
            characters</span></h5>' );
        $( this ).closest( ".form-group" ).
            addClass( "has-error has-
            feedback" );
        $( this ).after( '<span class="
            glyphicon glyphicon-remove form
            -control-feedback"></span>' );
        hasErrors = true;
      }
    }
  });
});

// at least 1 input type
$( ".questionsWrapper" ).children( ".
    question" ).each( function( index,
    element ){
  if( $(this).find( ":radio:checked" ).size
      () == 0)
  {
    $(this).find( ":radio:last" ).closest( "
        .inputType" ).append( '<h5><span
        class="label label-danger">select
        an input type</span></h5>' );
    $(this).find( ":radio:last" ).closest( "
        .form-group" ).addClass( "has-error
        " );
    hasErrors = true;
  }
});

// each choice must 1-50 characters long
$( ".questionsWrapper" ).children( ".
    question" ).each( function(
    questionIndex, question ){
  $(this).children( ".choicesWrapper").
      children( ".choice" ).each( function(
      choiceIndex, choice ){
    $(this).find( ":text" ).each( function()
        {
      if( $(this).attr("id") == ( "questions
          " + questionIndex + ".choices" +
          choiceIndex ) )
      {
        if( $(this).val().length < 1 || $(
            this).val().length > 50 )
        {
          $( this ).closest( ".form-group" )
              .append( '<h5 class="form-
              control-static col-sm-offset
              -2"><span class="label label-
              danger">should be 1-50
              characters</span></h5>' );
          $( this ).closest( ".form-group" )
              .addClass( "has-error has-
              feedback" );
          $( this ).after( '<span class="
              glyphicon glyphicon-remove
              form-control-feedback"></span
              >' );
          hasErrors = true;
        }
      }
    });
  });
});

// each question must have at least 1 result
    function
$( ".questionsWrapper" ).children( ".
    question" ).each( function( index,
    element ){
  if( $(this).find( ":checkbox:checked" ).
      size() == 0)
  {
    $(this).find( ":checkbox:last" ).closest
        ( ".results" ).append( '<h5><span
        class="label label-danger">select
        at least 1 result to compute</span
        ></h5>' );
    $(this).find( ":checkbox:last" ).closest
        ( ".form-group" ).addClass( "has-
        error" );
    hasErrors = true;
  }
});

if( hasErrors )
{
  e.preventDefault();

  /* Focus on first element with error */
  var withError = $( "#surveyForm" ).find( "
      .has-error:first" ).find( "input,
      textarea" );
  var navbarOffset = parseInt( $( "body" ).
      css( "padding-top" ) );
  $('html, body').animate(
  {
    scrollTop: ( withError.offset().top -
 navbarOffset )
  }, 'slow');
  withError.focus();
}
});

/* When text field has error */
function viewError( selector, message )
{
  $( selector ).after( '<h5><span class="label
      label-danger">' + message + '</span></
      h5>' );
  $( selector ).closest( ".form-group" ).
      addClass( "has-error has-feedback" );
  $( selector ).after( '<span class="glyphicon
      glyphicon-remove form-control-feedback
      "></span>' );
}

/* Checks if date is in correct format (yyyy-
    MM-dd) */
function isValidDate( dateValue )
{
  var currVal = dateValue;
  if(currVal == '')
  {
    return false;
  }

  var rxDatePattern =
      /^(19[0-9][0-9]|20[0-9][0-9])
      -(0[1-9]|1[012])
      -(0[1-9]|[12][0-9]|3[01])$/;
  var dtArray = currVal.match(rxDatePattern);

  if(dtArray == null)
  {
    return false;
  }

  dtYear = dtArray[1];
  dtMonth = dtArray[2];
  dtDay= dtArray[3];

  if(dtMonth < 1 || dtMonth > 12)
  {
    return false;
```

```
    }
    else if(dtDay < 1 || dtDay > 31)
    {
      return false;
    }
    else if( ( dtMonth == 4 || dtMonth == 6 ||
         dtMonth == 9 || dtMonth == 11) && dtDay
          == 31)
    {
      return false;
    }
    else if(dtMonth == 2)
    {
      var isleap = (dtYear % 4 == 0 && (dtYear %
            100 != 0 || dtYear % 400 == 0));

      if(dtDay > 29 || (dtDay == 29 && !isleap))
      {
        return false;
      }
    }

    return true;
  }

  /* Checks if first parameter is before second
       parameter */
  function isFirstParamDateBeforeSecondParamDate
       ( before, after )
  {
    if( after.getFullYear() < before.getFullYear
         () )
    {
      return false;
    }
    else if( after.getFullYear() > before.
         getFullYear() )
    {
      return true;
    }
    else if( after.getMonth() < before.getMonth
         () )
    {
      return false;
    }
    else if( after.getMonth() > before.getMonth
         () )
    {
      return true;
    }
    else if( after.getDate() <= before.getDate()
         )
    {
      return false;
    }

    return true;
  }

  /* View date picker */
  $( ".datepicker" ).datepicker(
  {
    dateFormat: "yy-mm-dd",
    minDate: 0
  });
});
```

76. survey_answer.js

```
$(document).ready(function()
{
  /* Focus on first question */
  $( ".question:first" ).find( "input:first" ).
       focus();

  // Form validation
  $( "#submitFormBtn" ).on( "click", function( e
       )
  {
    var hasAnswer = false;

    // remove previous error message
    $( "#alert-container" ).empty();

    $( ".question" ).each( function()
    {
      var input = $(this).find( "input" );
      var inputType = $(input).prop( 'type' );
      var dropdown = $(this).find( "select" );

      if( inputType === 'number' )
      {
        if( $(input).val().length > 0 )
        {
          hasAnswer = true;
        }

      }
      else if( inputType === 'radio' )
      {
        if( $(input).parent().find( ":radio:
             checked" ).size() > 0 )
        {
          hasAnswer = true;
        }
      }
      else if( inputType === 'checkbox' )
      {
        if( $(input).parent().find( ":checkbox:
             checked" ).size() > 0 )
        {
          hasAnswer = true;
        }
      }
      else if( dropdown != null )
      {
        if( $(dropdown).val() > 0 )
        {
          hasAnswer = true;
        }
      }
    });

    if( hasAnswer )
    {
      $( "#answerSurveyForm" ).submit();
    }
    else
    {
      e.preventDefault();

      $( "#alert-container" ).append( '<div
           class="alert alert-danger" role="
           alert">' +
                        '<span class="glyphicon
                             glyphicon-remove"
                             aria-hidden="true
                             "></span> ' +
                        'Please answer at least
                             1 question before
                             submitting.' +
                      '</div>' )

      $('html, body').animate(
      {
        scrollTop: 0
      }, 'slow');

      $( ".question:first" ).find( "input:first"
           ).focus();
    }
  });
});
```

77. application.properties

```
# Logging
logging.level.com.hdsgs=DEBUG
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR
logging.file=hdsgs.log

# DataSource settings
spring.datasource.url=jdbc:mysql://localhost
     :3306/hdsgs
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driverClassName=com.mysql.jdbc
     .Driver

# Specify the DBMS
spring.jpa.database=MYSQL

# JPA (Hibernate) settings
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.hibernate.dialect=org.hibernate.
     dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.naming_strategy=org.
     hibernate.cfg.ImprovedNamingStrategy
```

```
spring.jpa.properties.hibernate.
    current_session_context_class=org.
    springframework.orm.hibernate4.
    SpringSessionContext

# Thymeleaf
spring.thymeleaf.cache=false
```

78. messages.properties

```
# User form validation error messages
old.password.error=incorrect old password
retyped.password.error=passwords should match
username.exists.error=username already exists

# Survey form binding error messages
typeMismatch.startDate=Invalid date format
typeMismatch.endDate=Invalid date format

# Survey form validation error messages
empty.choice.error=Choice cannot be empty
end.date.error=End date should be after start
    date
start.date.error=Start date should be today or
    in the future
```

XI. Acknowledgement

Una sa lahat, gusto kong pasalamatan nang lubos ang Panginoon na siyang tumulong, at patuloy na tumutulong, sa pag-abot ko sa aking mga pangarap. Wala ako dito ngayon kung hindi dahil sa inyo, Lord. Sobrang bait ninyo sa pagbibigay sa akin ng mga opportunities para maging successful ako. Sinasamahan niyo ako sa bawat hakbang sa buhay ko, gumagabay, nagbibigay ng pag-asa. Habambuhay akong magpapasalamat sa inyong walang-hanggang awa at pagmamahal sa akin.

Pangalawa, para kay Sir Chua, maraming, maraming, maraming salamat po sa pagpili sa akin bilang SP advisee. Talagang umasa ako noon na kukunin niyo ako kasi sabi nila 'yung mga advisees niyo ung kadalasang guma-graduate on time. Hahaha! Maraming salamat talaga, Sir, kahit toxic minsan. Hahaha. Pero nag-payoff naman lahat ng efforts at trabaho kasi nakaya kong tapusin itong SP dahil sa tulong niyo. Salamat po sa pagtitiyagang i-proofread ang mga SP documents ko. Salamat po sa patuloy na pag-check niyo noon sa progress ng code ko. Salamat po sa pagsalo ng mga tanong sa proposal at defense ko. At higit sa lahat, salamat po sa marami niyong crticisms kasi nagkaroon ako ng motivation para ayusin at pagbutihin pa 'yung ginagawa ko para mapatunayan ko sa inyo, at sa sarili ko, na kaya ko.

Para sa SP buddy ko na si Joyce, salamat kasi ginawa mo akong masipag. Hahaha. Ang sipag at galing mong mag-code, kaya madalas napipilitan na din akong maging ganun para hindi ako insultuhin ni Sir Chua sa susunod na consultation. Hahaha! Salamat kasi natitiis mo 'yung mga rants ko. Salamat kasi tinulungan mo ko sa code. Salamat kasi ikaw karamay ko kapag naiistress sa dami ng gagawin. Hahaha. Pero maraming salamat talaga kasi nagkaroon ako ng bagong kaibigan. Lol, drama.

Para sa HTHT, kamusta na 'yung plano natin sa outing? May kulay na ba? Puro drawing lang e. Hahahaha! Lizette, Sam, Arianne, Cara, Hana, KatDat, Jodie, JC, Dwight, at siyempre si Meemoo (Franz), kayo naging pamilya ko dito sa UP. Salamat sa maraming tawanan. Salamat sa chismisan. Salamat sa pagtanggap niyo sa

akin. Salamat sa pagmamahal. Salamat sa pag-unawa. Salamat sa walang-hanggang pagkakaibigan. Sana mapatunayan natin sa kanilang lahat na may forever. I ♡ you guys 5evz!

At syempre, para sa pamilya ko, kayo ang tanging dahilan kaya ako nagsisikap. Para sa inyo lahat ng paghihirap ko. Para sa inyo lahat ng pangarap ko. Sana patuloy na 'yung pag-ayos ng buhay natin. Mama, Daddy, konting tiis na lang! Maibibigay ko din lahat ng ibinigay niyo sa akin. Mahal na mahal ko kayo ni Jared.