UNIVERSITY OF THE PHILIPPINES MANILA

COLLEGE OF ARTS AND SCIENCES

DEPARTMENT OF PHYSICAL SCIENCES AND MATHEMATICS

MegaPixel Allocation Management System (MPAMS)

A special problem in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Ethan Jarvey C. Ocampo

August 2023

ACCEPTANCE SHEET

The Special Problem entitled “Mega Pixel Allocation System (MPAMS)” prepared and submitted by Ethan Jarvey C. Ocampo in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____
Avegail D. Carpio, M.Sc.
Adviser

EXAMINERS:

	Approved	Disapproved
1. Richard Bryann L. Chua, Ph.D. (cand)	____________	______________
2. Perlita E. Gasmen, M.Sc. (cand)	____________	______________
3. Ma. Sheila A. Magboo, Ph.D. (cand)	____________	______________
4. Vincent Peter C. Magboo, M.D., M.Sc.	____________	______________
5. Marbert John C. Marasigan, M.Sc.	____________	______________
6. Geoffrey A. Solano, Ph.D.	____________	______________

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

____
Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

____
Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

____
Maria Constancia O. Carillo, P
Dean
College of Arts and Sciences

Abstract

The project aims to solve the client's need to calculate order allocations, as their current use of multiple spreadsheets typically requires more time and human intervention in doing so. These are generally due to two reasons, the use of ratio and proportion formulas and the specific needs of a client's order. There are specific scenarios where their client may specify which variants they wish to have, which would require further thought by the user. The developed system allows standard users to manage the inventory of supported products, to allocate these inventories based on the needs of their clients, and to print related reports for review and for preparation in deliveries. For admin users, they have access to user management, client management and control for certain choice-based fields, in addition to the functionalities made available to standard users. This project was made with the following aims: to reduce the use of spreadsheets for the client, to provide a unified and easy to use platform for this aspect of the business, and to potentially reduce the amount of time and human intervention needed for order allocation. The developer was able to develop the system as specified by the client. The system allows for the order allocation process to be potentially faster, as it can calculate the units to be allocated automatically, allowing for less time and intervention needed. Future developments or attempts at the project could implement alternative input methods, performance-related improvements and additional features as seen fit by the client.

Keywords: Django, order allocation system, clothing inventory system, Python

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Megapixels Printing Services Inc is a company that provides digital printing services to its clients (Yellow.Place, n.d.). Founded in September 2012, they use the latest direct-to-garment technology for cotton and cotton blended materials for their products. These include standard printing tasks such as for business cards and catalogs, sublimation printing, direct to garments printing, and the application of studs and rhinestones. The company manages their available inventory and orders using Excel applications, primarily Google Sheets.

They use these applications to systematically organize all business matters present to them, including the management of inventory and of orders requested to them by their clients. Each type of garment is assigned its own stock code, and may come in different colors and sizes. The amount for each variant is shown in the spreadsheet, along with the total of all available inventory. Each spreadsheet is assigned to a particular garment and client, who may have multiple stores to receive clothes, and wherein order management also occurs. The total request amount is inserted into the spreadsheet and a formula is used to calculate the amount of each variant to be given to each store included in the spreadsheet.

B. Statement of the Problem

The client's current method of choice in managing inventory and orders involves the use of multiple spreadsheet applications and of ratio and proportion formulas. Each product has a spreadsheet to manage the number of units available to the company. Another spreadsheet is then used when an order is made by one of the company's clients, where they then use ratio and proportion formulas to determine the amounts to be allocated for each of the client's stores. This has proven to be useful to a certain degree but has certain limitations.

Their use of the technologies has helped to a certain degree, as it has helped to identify the proportion needed to be allocated from their pre-existing inventory, but it was shown to the researcher by the client that it has been inaccurate in the calculations and thus would need further manual manipulations for proper distribution. Additionally, the management of clothing of differing levels of quality, colors, and sizes has been shown to be very decentralized through the use of multiple spreadsheets. The current method in which they process and manage their orders is still relatively manual as the client is often required to make multiple manipulations on their part to properly balance the numbers shown in the spreadsheets.

The study wishes to provide an easy-to-use and effective way for the client to monitor orders and to assist in making decisions concerning the distribution of available inventory to multiple clients and stores.

C. Objectives of the Study

The project aims to develop a garment allocation system that may potentially serve as an effective replacement for the client's use of multiple spreadsheets in managing inventory allocation. The implementation of the project will aim to provide the client with a system that aids in distribution decisions in various scenarios. The following are the expected functionalities to be implemented:

1. Employees
    a. Inventory Management
        i. Add to Inventory
        ii. Editing/Reclassifying Inventory
        iii. Receive alert messages for low inventory levels.

    b. Allocate garments based on the available inventory
        i. Provide suggestions on garment distribution based on request quantity and other limiting factors
        ii. Print allocation breakdown

2. Admin Privileges
    a. User Management
        i. Add user
        ii. Deactivate user
        iii. Reactivate user
    b. Inventory Management
        i. Add to Inventory
        ii. Editing/Reclassifying Inventory
        iii. Receive alert messages for low inventory levels
    c. Add client
        i. Add stores of client
        ii. Deactivate client account
        iii. Reactivate client account
    d. Allocate garments based on the available inventory
        i. Provide suggestions on garment distribution based on request quantity and other limiting factors
        ii. Print allocation breakdown

D. Significance of the Study

The new system will help the client by providing a centralized system in handling inventory distributions. The system will allow for the client to lessen the need to use excel spreadsheets in his line of work, allowing for a slightly decluttered way of handling tasks. The client often accesses multiple spreadsheets at the same time when managing the inventory available and in allocating garments for his clients, this system will allow for all tasks to be done in one place. This is done as first the client must separate good quality products from those deemed to be inferior quality, as lower quality products are to be sold to flea markets (*tiangge*). Afterwards, good quality products are to be offered and sold to more established clients.

Decision making on inventory allocation could be executed in less time, as the system can provide suggestions that could satisfy the conditions that are provided by the user (i.e. use only a certain array of colors, only supply a certain set of sizes, etc.). In using spreadsheet applications, the client would often find scenarios where the allocation of inventory is not equally distributed across variants of the clothing item to be allocated. This would require further manual manipulation by the user, so that all of the allocations will match the total requested amount. Additionally, this may be a result of the stores having a particular condition in requesting products. The system will be able to provide and print a quick summary of the allocation for the user, allowing for easier notification for the stores when receiving these allocations as they would be able to quickly identify and recognize the clothing delivered and the amount provided by the client.

E. Scope and Limitations

1. No financial information or related functionalities is to be implemented or present in the application by client's request
2. The system allows for multiple simultaneous users
3. The system provides real time update by any user
4. The system requires internet connection to be operational

II. Review of Related Literature

This section of the paper provides a brief review on the progress and development of information management systems in multiple areas that are implemented through the use of multiple programming languages and technologies. This will provide insight on what possible technologies are available for the implementation of a new system. But first, it should be defined on what an information system is and what has been done in the development of these systems, particularly in the business and manufacturing industries.

As defined by Concordia St. Paul (n.d.), an information system is a “platform or set of platforms that exist to manage a set of information or a technology product.” This kind of technology has always been important in all sectors of society since its inception. Notably, information systems have been implemented and used in business sectors, in healthcare, and in the academic setting. In healthcare, the Electronic Health Record (EHR) allows for the digitization of the patient record, and the information system allows for multiple patient records to be quickly and readily accessible to the necessary users (ECPI University, n.d.). In academia, these systems are integral to maintaining accurate and up-to-date data regarding multiple facets of the academic ecosystem, such as student records and management of the educational facility’s various resources. One such example is that of the development of a postgraduate student support system by Aisar et. al (2020), as a means of improving on pre-existing use of Excel in maintaining records. The team was able to successfully implement all the requested functionalities through a modified waterfall method and provide a generally good rate of user satisfaction. The system allowed for easy communication, data security, and simple report generation. The study provided possible technologies that could be used for the implementation of the new system later in the research. In particular, the study used NetBeans IDE, Apache Tomcat web server, and MySQL server in the implementation of the project, providing generally good results and full implementation of functionalities. This shows the viability of Java in implementing such a system, along with other languages and/or associated frameworks that are present in other pieces of literature mentioned in the paper.

In the business sector, companies conduct their data recording activities using an inventory management system or an Enterprise Resource Planning (ERP) system. These systems allow companies to improve productivity and efficiency by having a unified system that allows for multiple departments to interact with one another, and this also allows them to be globally competitive (University of Scranton, 2020).Prior to the utilization of these systems, businesses mainly operated with paper-based systems, which can be subjected to human error, and order location status was a menial task, as people are required to manually inquire about their status. This has allowed for businesses to expand their prospects to the global market, as needs such as web-based ordering and customer relations management can be readily accommodated with these systems. In this paper, the focus is on the development of information systems in the business and manufacturing setting. Before attempting to implement an information system, it is pertinent to examine what has been done, what considerations there should be for inventory systems, and what database management system can be used to handle the data storage.

Saha, Basumatary, Senapati, and Maity (2021) conducted a study regarding what aspects or considerations should be made when designing a modern website for better efficiency. Their study involved their test users to fulfill ten tasks in differing websites, using the time taken to conduct each task and a short survey conducted afterwards as their metrics in determining trends on efficient website design. It was shown that interfaces should be made as simple as possible, as these take the least amount of time to complete tasks and these score generally high in their surveys. This could extend as well as to the design of prospective inventory systems, as this could allow prospective users to have ease in using the system and to complete their tasks in an efficient and satisfactory manner.

Database Management System (DBMS)

Sinha (2019) details the importance of the development of the database management system and different aspects concerning its implementation. They discuss the purpose of DBMS, the improvements it has made over other file systems, its features and its capabilities in servicing any business or organization. Furthermore, they discuss the different methods and techniques in reducing data redundancy and in mapping records to each other. Notably, they differentiated the use of relational databases from hierarchical and network type databases. Normalization was also shown as a means to minimize data redundancy and to potentially eliminate certain anomalies from occurring. Lastly, they provided both the advantages and disadvantages in using a DBMS in general and summarized what a DBMS is. Some of the advantages include data sharing capabilities, data integrity and security, and concurrent access. Disadvantages include lack of standardization, size, and costs in acquiring and maintaining the system.

Rawat, Purnama and Mulyati (2021) showcased the use of MySQL database in their paper documenting its use on the FTP Site LAPAN Bandung. LAPAN Bandung is a site that stores observational data from the LAPAN Aerospace Observer Center. They highlighted the nature of MySQL and its history, and cited some of the reasons and advantages of using this technology. Here they present that it allows for researchers and other users of the system to be able to access data more easily and more readily rather than the traditional method of obtaining data on-site. Furthermore, this allows for users and activities to be monitored and regulated more easily, allowing for protection and security of data from unauthorized and other malicious users. This may provide reason as to choose MySQL as the DBMS of choice for developers.

Hasan (n.d.) presented in his paper the differences between using NoSQL databases and using a relational database management system (RDBMS). One of the main points he presents is the scenario where requirements may change overtime, wherein using a NoSQL database would be more preferable than using a RDBMS. They illustrate this further by providing scenarios and examples which showcase the advantages of NoSQL over RDBMS. Most of these scenarios do posit data structure changes and availability being some of the main advantages of NoSQL. However, they do mention that RDBMS still has some benefits over NoSQL, mainly its ACID properties and queries that cannot be matched by NoSQL. Here this presents the potential decision between two database systems, where the developer may have to choose between flexibility and reliability. This is something to consider when developing any modern system.

One facet that may be important to consider in choosing the DBMS to use in one’s application is the speed at which operations or procedures are executed inside the database. This was noted in Hartono and Erfina’s 2021 article “Comparison of Stored Procedures on Relational Database Management Systems”, as they compared two open-source RDBMS, namely: MariaDB and PostgreSQL. In their study, they conducted an experimental and descriptive analysis, as both RDBMS are tested on insert, update and delete operations. They conducted them with 5000, 15000, and 25000 data entry tests, in order to compare the performance of each RDBMS, in terms of execution speed. It has been found that PostgreSQL performs better than MariaDB, as it takes less time to perform for all three noted operations. Notably, it takes significantly less time in deleting records than MariaDB, as it takes less than 0.05ms to perform for all three test loads, compared to a minimum 0.15ms for MariaDB. This could be a significant factor in choosing the DBMS for a project, as this performance may be highly sought for by most businesses and organizations.

Previous Works

Ilias et.al (2018) developed an inventory system for the shop At-Thoyyib, a small grocery store operating in Malaysia. The business recorded their sales in books and generated their reports manually, thereby running the risk of misregistering information and the potential waste of resources. The system was developed to provide an easier method of inventory management for the employees and overall store management for the administrators. The researchers have studied other inventory systems made by others in the past, as to further find other functionalities that may be necessary to implement and to serve as a guide and reference when developing the system. They’ve used the Waterfall methodology in developing the project, as this allows for them to properly understand the system and it provides a systematic approach to developing the system. They provided multiple diagrams, such as data flow diagrams and role flowcharts, to properly illustrate how data is to be treated in the system and to clarify on how the functionalities of the different users are shown in the system. The researchers used PHP and MySQL in the implementation of the project, and they were able to provide all the functionalities outlined in the initial portion of the paper.

Aru and Priyadharshini (n.d.) developed their own inventory management system, which demonstrates some proposed functionalities and capabilities based on the types of users that will be using the system. Some of these include manager's and admin's modules, a dynamic billing system, damage analysis, and placing orders from the store's suppliers. Both manager and admin could update the shelf stock and update any sales, but only the admin may alter the products which the store offers. The manager may also place orders to their suppliers through a module that has been developed into the system. Interestingly, the researchers have also added a damage analysis module into their system. This module allows for the store to identify which products have arrived damaged and how they came to be so, thus allowing them to make proper countermeasures as to prevent recurrences. Additionally, they propose that future development may be undertaken to this module, allowing for an ML based analysis to identify all possibilities and predictions of potential future damages based on gathered data. This could provide significant benefit to the store using this system, as it could allow for mitigation of damage to assets and for an increase to its profits.

Misu (2019) developed STASH, an online inventory management system, as part of their thesis and in assistance of an unnamed company. Developed using PHP, MySQL database via XAMPP, and the CodeIgniter framework, they have been able to develop a system that provides multiple functionalities such as purchases, user management, sourcing from local markets, sales, and report generation among them. All data records and processes have been extensively defined using multiple diagrams such as use-case, sequence, and dataflow diagrams, with each data class being properly specified of its properties. Results had shown that the system has been a success, as the company that used the system has been able to maximize its efficiency through using the system.

Srivastava, Choubey, & Kumar (2020) presented their own inventory management system, consisting of multiple stakeholders such as the department, warehouse, raw materials, suppliers, and employees. They reviewed multiple sources, in order to properly assess which set of technologies might be the most useful in implementing their system, taking note of the advantages and disadvantages of each technology with respect to a possible alternative and the discoveries made in the papers they have selected. Some of these technologies include databases (NoSQL, MongoDB, SQLite, MySQL), languages (PHP, JSP), Cloud Technology, and the use of Regression in forecasting. In this project, they have decided to use a combination of MongoDB, Java, and NetBeans to develop the system. They reasoned that MongoDB's Sharding and Replication capabilities and its way of storing data shows that it is a useful tool in storing the company's data, as it can allow for them to store duplicates in separate servers. This would prevent possible issues arising from any possible disruption that may occur when in operation. In future, they have posited to implement Machine Learning Prediction algorithms to predict possible requirements and the use of NewSQL to handle structured and unstructured data simultaneously in the system.

Shah, Kedia, and Jha (2021) presented their own take on producing an inventory system fit for medium sized manufacturing ventures. Here they noted that they went for an iterative enhancement model in developing the system, as this allows for more flexibility and feedback per stage as compared to the traditional SDLC. They've provided sufficient diagrams to properly explain the different processes and functions that were implemented in the system. Here they have noted that they used .NET Framework 4.5, Microsoft Visual Studio Code, and Microsoft SQL Server in developing the system. The .NET Framework was chosen as it provides sufficient tools and libraries in developing both front end and back end components of the system, and it allows for interactions with multiple servers easier. Microsoft SQL Server was chosen as the database for the application, as it provides encryption capabilities inside the database. This work illustrates further options for researchers and developers to use when developing any form of application and system, and may be taken into consideration when doing so.

The works mentioned in this section of the paper, addressed the need of automating the data management process through multiple technologies and methodologies. The researcher was able to identify technologies that could be used for the development of the online inventory system and the associated database management system in storing the data. Possible technologies that I could use include PHP, MySQL, NoSQL, MongoDB, frontend resources, and possibly Java, with the associated servers.

III. Theoretical Framework

A. Megapixels Printing Services, Inc.

The company was founded in September 2012 and is currently based in Caloocan City. The company provides multiple types of printing services to their customers. These include standard printing services (i.e., catalogues and business cards), sublimation printing, and direct-to-garment printing. Recently, they have added clothes trading as part of their business operations and source of revenue. For this portion of the business, they use multiple spreadsheets and ratio and proportion formulas to manage the logistics including inventory management and order allocation. This is the main source of the problem that the project aims to resolve.

B. Inventory Management and Order Allocation

The company sources their garments from different vendors they have encountered. After receiving the garments, they undergo an inspection process and separate them into two categories: damaged and well made. Damaged garments are then to be sold to flea markets (*tiangge*). Well-made garments are to be distributed to established clients upon request and negotiations are made. Each of these well-made garments are given its own stock code. These findings are noted down in spreadsheets, taking note of the amount for color and size variations of the garment.

Orders are obtained either through contact (calls or emails) from clients or through offerings to prospective clients. Once an order has been obtained, a spreadsheet will be created for each requested item with the client’s stock code attached for identification. Ratio/proportion formulas are used to identify an equal distribution of the inventory, based on the amount requested by the client. Further manual manipulations may be required based on the nature of the order (request for specific colors, number of colors, etc.) and/or the computed total of items to be allocated does not match the requested amount given by the user.

C. Database Management System (DBMS)

A Database Management System or DBMS is a set of programs that allows for the user to manage their data by having it in a database. (OmniSci, n.d.) This is possible as it mainly serves as an interface between the users and the database, allowing for a more organized and structured approach to managing data. Some of the functions and features include activity logging, data security, data abstraction, remote access, and data integrity, all of which is done to properly store data in a safe and manageable way. DBMS comes in multiple forms including cloud based database management systems, columnar database management systems, and NoSQL databases. Compared to traditional file systems, a DBMS would allow for larger file systems, easier data sharing, data integrity and a more complex backup system. Most DBMS would have multiple security features that could prove to be much more secure in comparison with traditional file systems.

One of the more popular types of databases to use currently is the relational database management system (RDBMS). (Brush, 2019) It is a set of programs that allows users to interact with a relational database. It provides storing and retrieving capabilities for large amounts of data and generally good system performance and ease of implementation. Compared to a DBMS, RDBMS can allow for multiple simultaneous users, larger storage capacity, and has an ACID implementation. This is done by storing data in a table format, using keys to identify certain entries in the database table. Additional advantages of RDBMS include flexibility in terms of updating and changing data, and an understandable data structure. Disadvantages may include cost in implementation, tedious set-up processes, and massive inserts of data may be problematic. This form of DBMS is often found in manufacturing, human resources, and banking among other fields, with some examples of RDBMS include MySQL, Oracle, IBM, Microsoft SQL Server, and PostgreSQL.

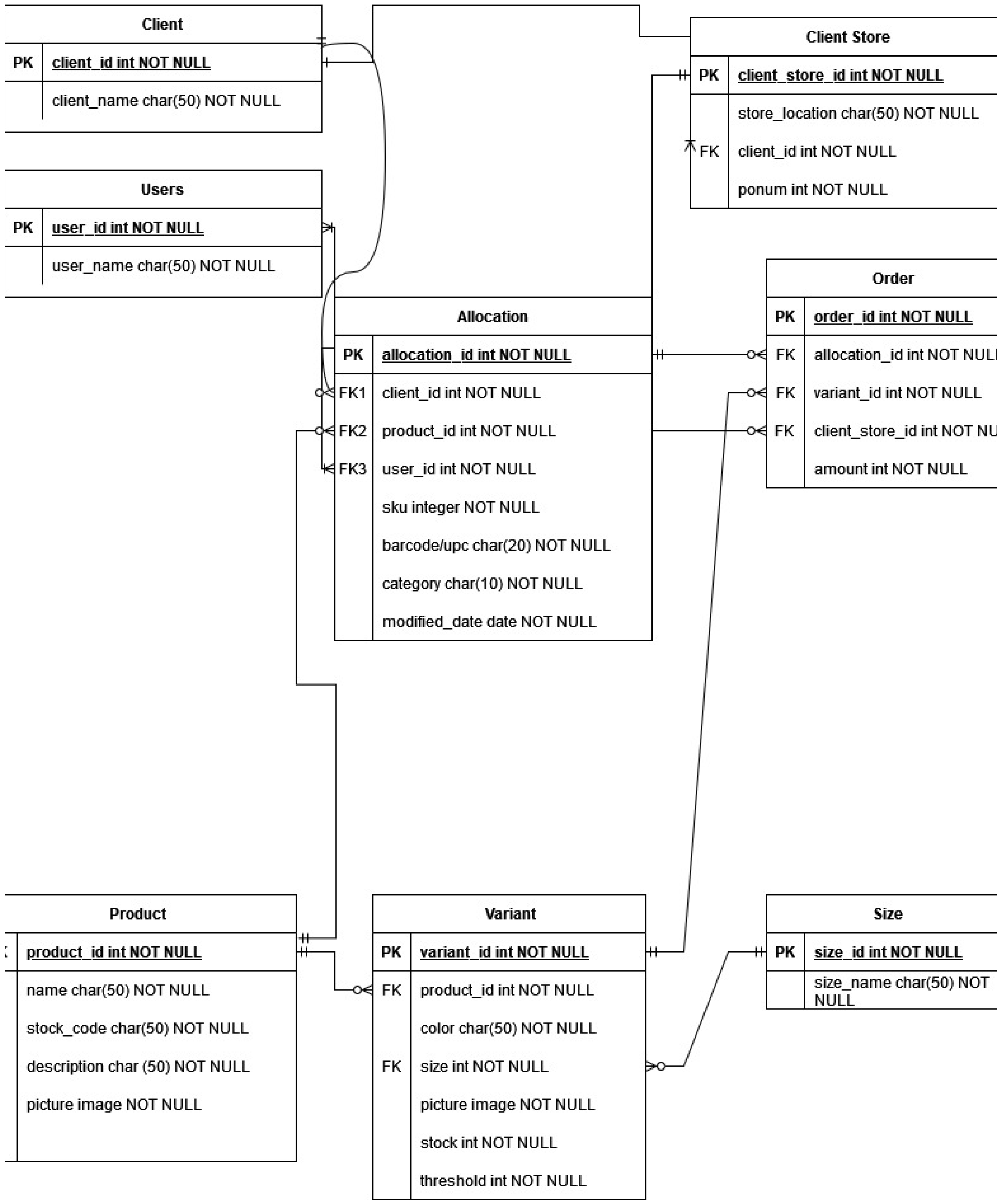
D. Efficient Inventory System

In modern times, having an efficient inventory management system can be crucial to being successful in any business or organization, as it can allow them to maximize the resources available and minimize any wastage of them. Some of the benefits to having an efficient inventory system include order accuracy, boosts in productivity, and more organized warehouse facilities. Overall, having an efficient inventory would allow owners to save time and money and use it to further improve their business. Some of the ways this could be done is by having accurate forecasting measures, having proper relations management and many other potential solutions can be available to these entities. (Katre, 2021) Forecasting measures can allow organizations to reasonably predict when they would have to restock certain resources or to consider what changes may need to be made depending on what is currently available. This could be useful as this could allow the organization to prevent instances of insufficient stock when needed.

Other methods and tactics may include using a realtime database management system, automated data entry, taking a holistic approach to examining inventory, using proper software in managing inventory, and using the best tools available to manage inventory. (Xero, n.d.) A holistic approach may refer to considering the cost in resources of moving inventory as a whole rather than focusing on just the cost of the inventory itself. Depending on the needs of the organization, some of the more traditional methods and tools available may be insufficient in handling its inventory. Methods, such as written records and sometimes spreadsheet applications, may present limitations in inventory management, as they could be prone to human error and/or it may limit the number of users possible in manipulation. The use of available tools such as automated data entry and realtime databases can provide flexibility and ease to viewing and manipulating inventory for the organization and its users. Automating data entry can allow for minimal error to occur in inputting the data in the system, preventing loss or inaccurate records to be included in the system. A realtime database system could allow for multiple users to view and manage inventory simultaneously, reducing gaps in changes in the system.

IV. Design and Implementation

A. Entity Relationship Diagram (ERD)


Client
PK client_id int NOT NULL
client_name char(50) NOT NULL
Users
PK user_id int NOT NULL
user_name char(50) NOT NULL
Client Store
PK client_store_id int NOT NULL
store_location char(50) NOT NULL
FK client_id int NOT NULL
ponum int NOT NULL
Order
PK order_id int NOT NULL
FK allocation_id int NOT NULL
FK variant_id int NOT NULL
FK client_store_id int NOT NU
amount int NOT NULL
Allocation
PK allocation_id int NOT NULL
FK1 client_id int NOT NULL
FK2 product_id int NOT NULL
FK3 user_id int NOT NULL
sku integer NOT NULL
barcode/upc char(20) NOT NULL
category char(10) NOT NULL
modified_date date NOT NULL
Product
product_id int NOT NULL
name char(50) NOT NULL
stock_code char(50) NOT NULL
description char (50) NOT NULL
picture image NOT NULL
Variant
PK variant_id int NOT NULL
FK product_id int NOT NULL
color char(50) NOT NULL
FK size int NOT NULL
picture image NOT NULL
stock int NOT NULL
threshold int NOT NULL
Size
PK size_id int NOT NULL
size_name char(50) NOT NULL
Figure 1: Entity Relationship Diagram

B. Data Dictionary

Client					
Attribute	Type	Description	Nullable	Pattern	Example
Client ID	Integer (Primary Key)	ID of client in the system	No	n	4
Client Name	Character (50)	Name of client to be displayed in the system	No	sssss	John Smith

Table 1: Client Data

User					
Attribute	Type	Description	Nullable	Pattern	Example
User ID	Integer (Primary Key)	ID of user in the system	No	n	7
User Name	Character (50)	Name of user in	No	sssss	Manuel Jose

		the system			

Table 2: User Data

Client Store					
Attribute	Type	Description	Nullable	Pattern	Example
Client Store ID	Integer (Primary Key)	ID of client's store in the system	No	n	10
Client ID	Integer (Foreign Key)	ID of the client which owns the store	No	n	13
Store Location	Character (50)	Location of the particular store	No	sssss	Pedro's Clothes North Caloocan
PO Number	Integer	Post Office number of a particular store	No	n	10293817

Table 3: Client Store Data

Product					
Attribute	Type	Description	Nullable	Pattern	Example
Product ID	Integer (Primary Key)	ID of the product in the system	No	n	2
Name	Character (50)	Name of the Product	No	ssss	Willson's Striped Polo shirt
Description	Character (50)	Description of the product offering	No	sssss	Acquired from seller in Malabon
Stock Code	Character (50)	Stock code of the product; for use within the company	No	ssssss	JM2901S
Picture	Image	Picture of the product for easy reference	No		…..some_product.jpg

Table 4: Product Data

Variant					
Attribute	Type	Descripti	Nullable	Pattern	Example

		on			
Variant ID	Integer (Primary Key)	ID of the variant in the system	No	n	2
Product ID	Integer (Foreign Key)	ID of the product the variant belongs to	No	n	6
Color	Character (50)	Color(s) of the variant	No	sssss	Yellow/Green/Black
Size	Integer (Foreign Key)	Size of the variant	No	n	2
Picture	Image	Picture of the variant for easy reference	No		…..some_product_redvariant.jpg
Stock	Integer	Number of variant units available	No	nnn	450
Threshold	Integer	Minimum number required to alert users of low stock levels	No	nn	20

Inches	Integer	Measurement in inches of the variant	No	n	15

Table 5: Variant Data

Size					
Attribute	Type	Description	Nullable	Pattern	Example
Size ID	Integer (Primary Key)	ID of the size in the system	No	n	2
Size Name	Character (10)	Name label of the size	No	ssss	XXL

Table 6: Size Data

Order					
Attribute	Type	Description	Nullable	Pattern	Example
Order ID	Integer (Primary Key)	ID of the order in the system	No	n	40
Allocation ID	Integer (Foreign	ID of the allocation	No	n	3

	Key)	the order belongs to			
Variant ID	Integer (Foreign Key)	ID of the variant requested	No	n	15
Client Store ID	Integer (Foreign Key)	ID of the store for the requested variant	No	n	6
Amount	Integer	Amount allocated of a particular variant to a particular store	No	n	35

Table 7: Order Data

Allocation					
Attribute	Type	Description	Nullable	Pattern	Example
Allocation ID	Integer (Primary Key)	ID of the allocation in the system	No	n	4
Client ID	Integer (Foreign Key)	ID of customer requesting the	No	n	3

		order			
Product ID	Integer (Foreign Key)	ID of the product requested	No	n	15
User ID	Integer (Foreign Key)	ID of the user who last modified the allocation	No	n	6
Modified Date	Date	Date of last modificat ion for allocation	No	M d,YYYY	June 4, 2022
SKU	Integer	SKU of the allocation	No	nnnnnnnn	10292012
Barcode/ UPC	Character (20)	Barcode/ UPC of the allocation	No	ssssssssss	920-201920183
Category	Character (10)	Category given by the client	No	sssssss	JH - R

Table 8: Allocation Data

C. Use Case Diagrams/Data Flow Diagrams

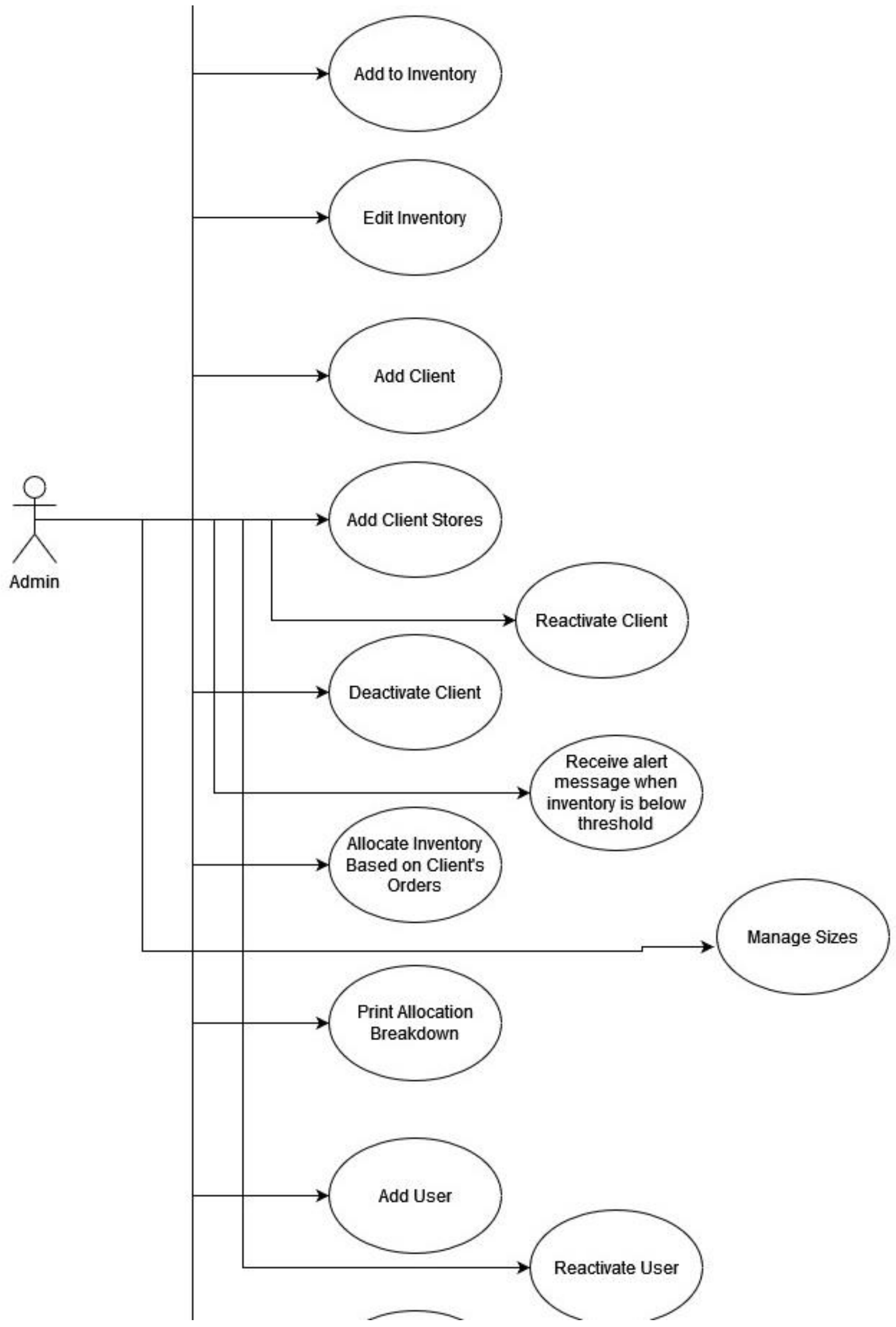


Figure 2: Admin User Use Case Diagram

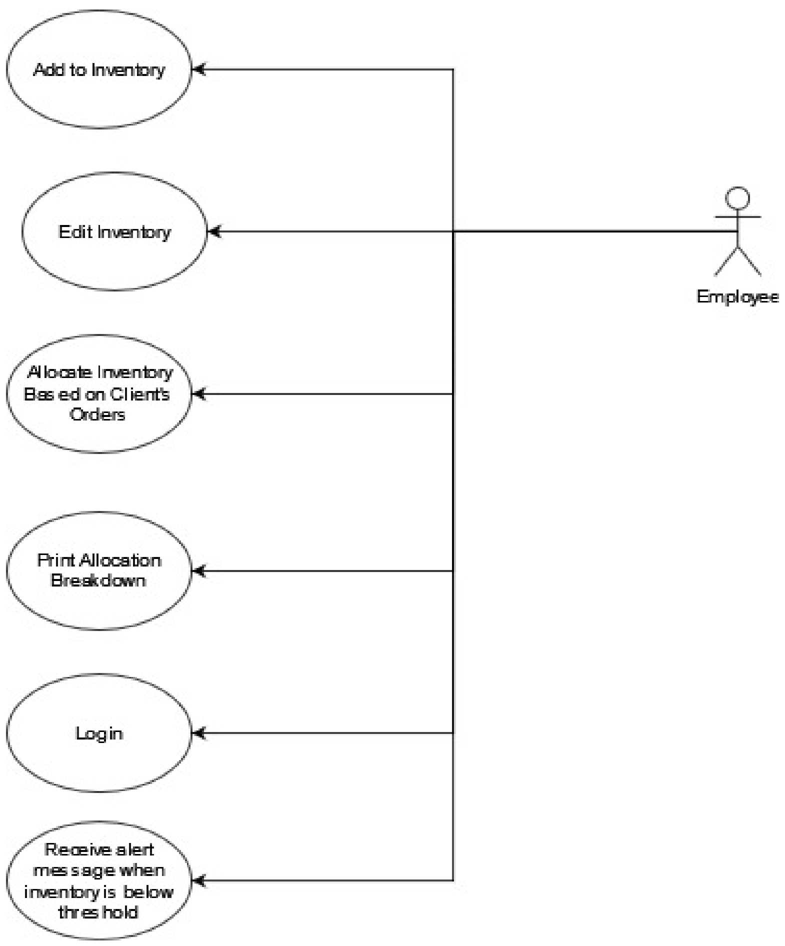

Add to Inventory
Edit Inventory
Allocate Inventory Based on Client's Orders
Print Allocation Breakdown
Login
Receive alert message when inventory is below threshold
Employee

Figure 3: Standard User Use Case Diagram

D.Technical Architecture

The application will be made using the following technologies: Django, Python, and Bootstrap. Django and XAMPP will be used to build a secure, easy-to-develop inventory management system. The proposed system will use Python to handle requests and to deliver requested content to the end user. The system’s front-end will be developed using BootStrap and jQuery, while the backend will be made using Python and built-in Django functionalities.

The application will be run on a server having the following technologies installed:

- Django
- Python

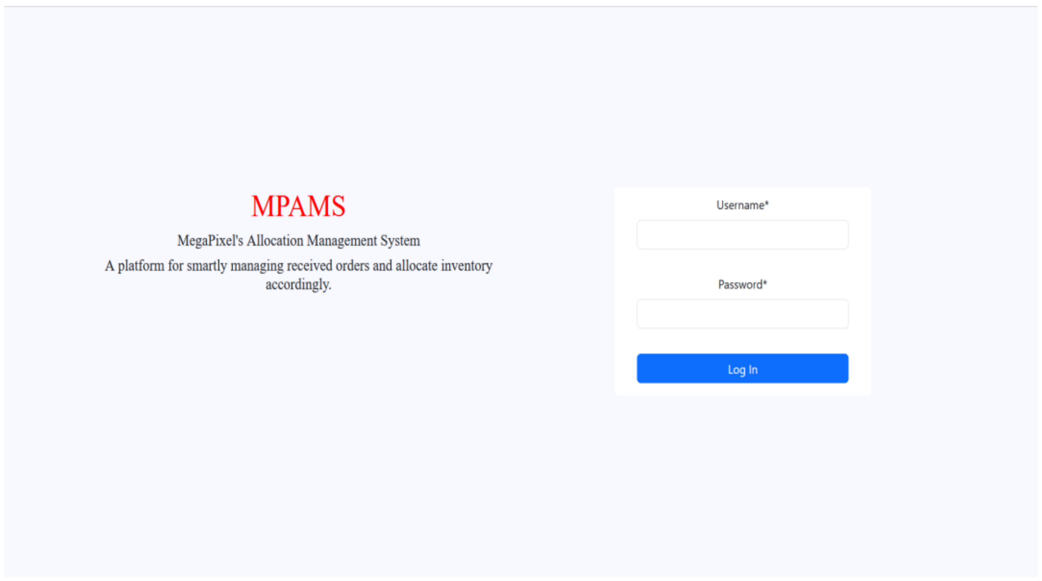
The web application should be accessible on the following browsers:

- Google Chrome
- Mozilla Firefox
- Safari
- Microsoft Edge

V. Results

A. User Login Page

The Login Page is the first page that users will encounter when accessing the website.


MPAMS

MegaPixel's Allocation Management System

A platform for smartly managing received orders and allocate inventory accordingly.

Figure 4: User Login Page

B. User Landing Page

This is the first page that users will encounter when logged into the website. It has a taskbar that contains search functionality, notifications (bell), and the logout (door with arrow) button. Notifications will remain in the notification area until they are dealt with. The page will welcome the user with their username and respective user type. Lastly, there are two buttons corresponding to the two main sections of the system, *Inventory Management* and *Order Allocation*. Clicking on the *MPAMS* button will return the user to this page.

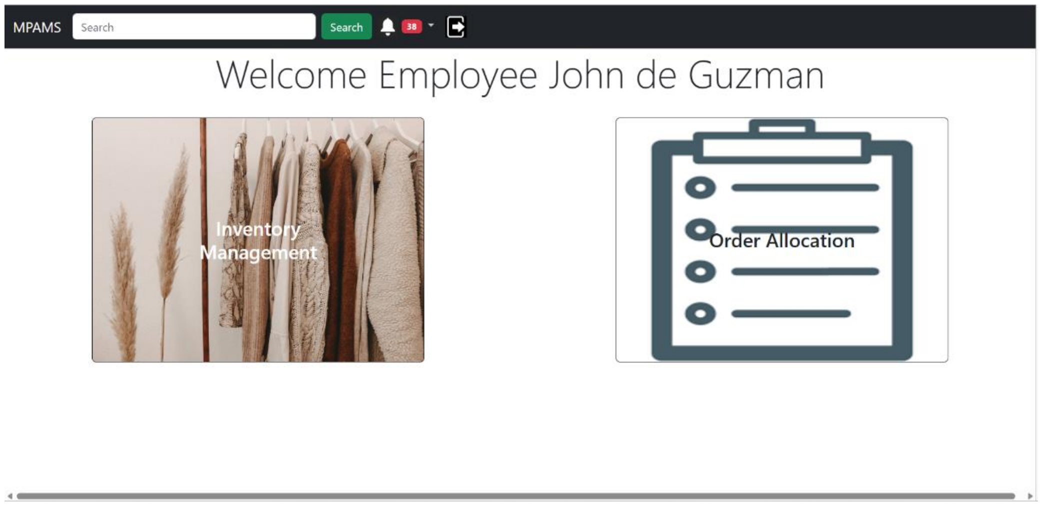
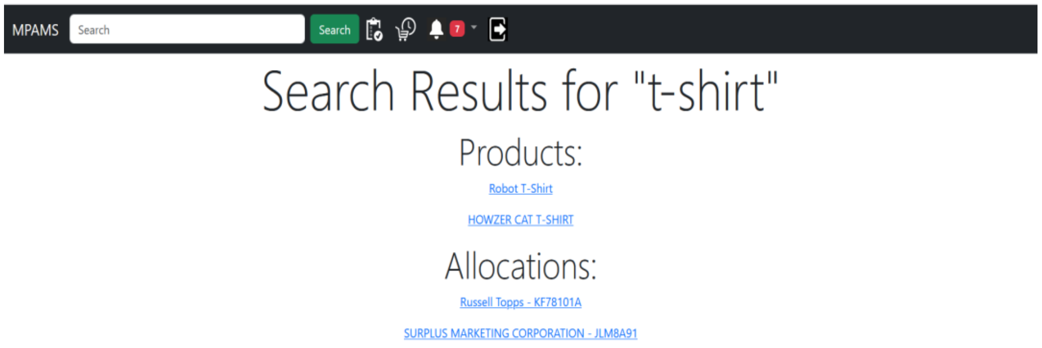


Figure 5: User Landing Page

C. General Search Page

This page is accessed by users after using the search bar found on the taskbar. Here the results of the provided search term will be displayed. The search results apply for products, variants and allocations. Clicking on a result will direct the user to the respective display page of each entity (i.e. product search result to product display page, etc.).



Search Results for "t-shirt"

Products:

Robot T-Shirt

HOWZER CAT T-SHIRT

Allocations:

Russell Topps - KF78101A

SURPLUS MARKETING CORPORATION - JLM8A91

Figure 6: General Search Result Page

D. Inventory Management Product List Page

This page can be accessed by users after clicking the *Inventory Management* button from the landing page or the equivalent button (clipboard) found on the taskbar. Here the user can view all of the products that are currently available in the system, can manage product information, and can access their variations. The page also contains a taskbar similar to the one found in the landing page, with two additional buttons corresponding to Inventory Management and Order Allocation list pages. This taskbar is also found on all other user pages, with the sole exception of the landing page mentioned earlier. Product details such as their picture, name, code, category and description can be seen on this page. Found on the page is an additional search bar, for quickly finding a specific product in lieu of manually scrolling through and finding the target product. If the user wishes to view all of the products again after searching, then they may press the *Show All* button. Clicking on the *Add New Product*, *Edit*, and *Variations* buttons will take the user to the Inventory Add Product page, Inventory Edit Product page and Inventory

Variation List Page respectively. The latter two mentioned buttons are made available for each product listed on the page.

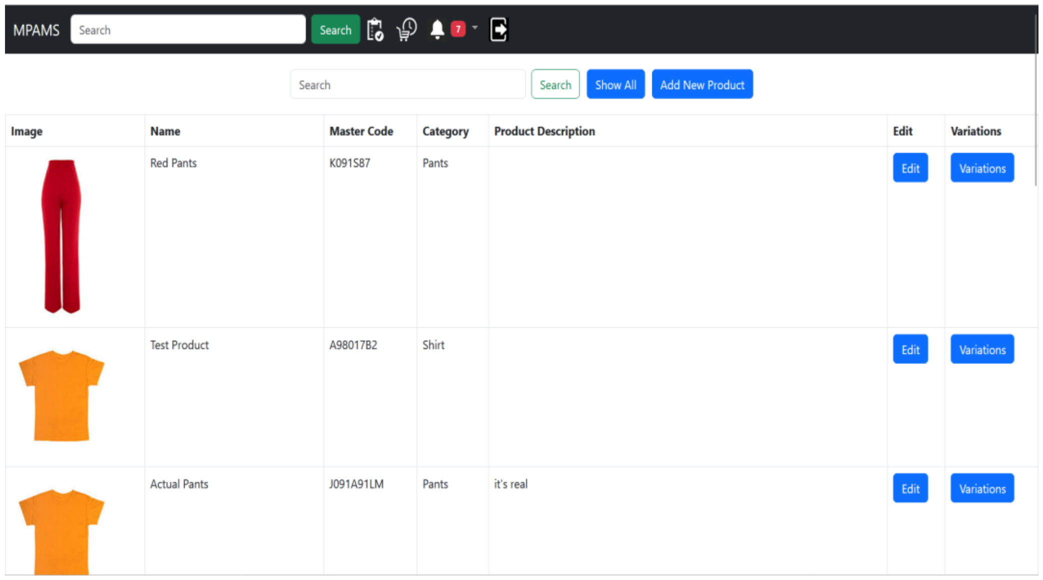
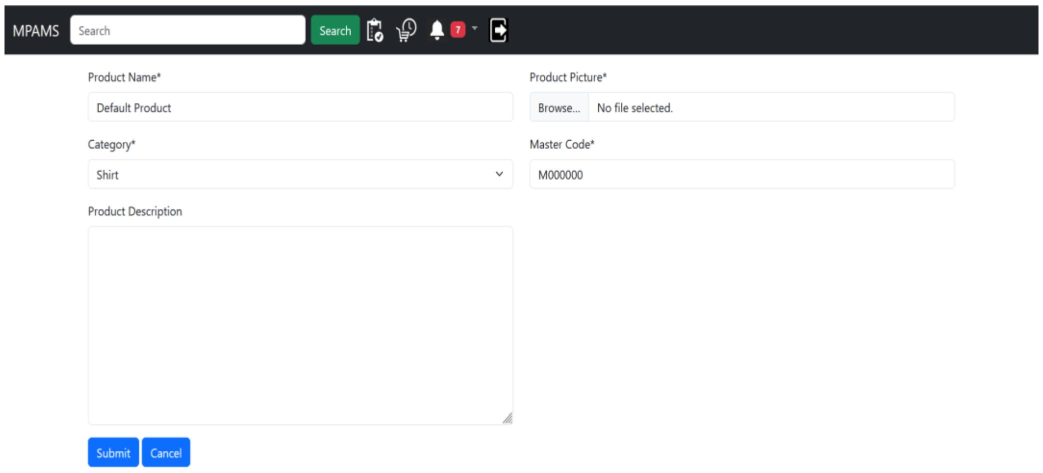


Image	Name	Master Code	Category	Product Description	Edit	Variations
	Red Pants	K091S87	Pants		Edit	Variations
	Test Product	A98017B2	Shirt		Edit	Variations
	Actual Pants	J091A91LM	Pants	it's real	Edit	Variations

Figure 7: Inventory Management Product List Page

E. Inventory Management Add Product Page

This page is accessed after clicking the *Add New Product* button from the Product List Page shown above. This allows the user to add new products to the system when needed. The page contains input fields for the name, picture, category, code, and description of the product, as these attributes are used to fully describe the product, as instructed by the client. Upon completion of the fields, the user may click the *Submit* button to add the product to the database. For any reason the user may not wish to add the product into the database, they may click the *Cancel* button.

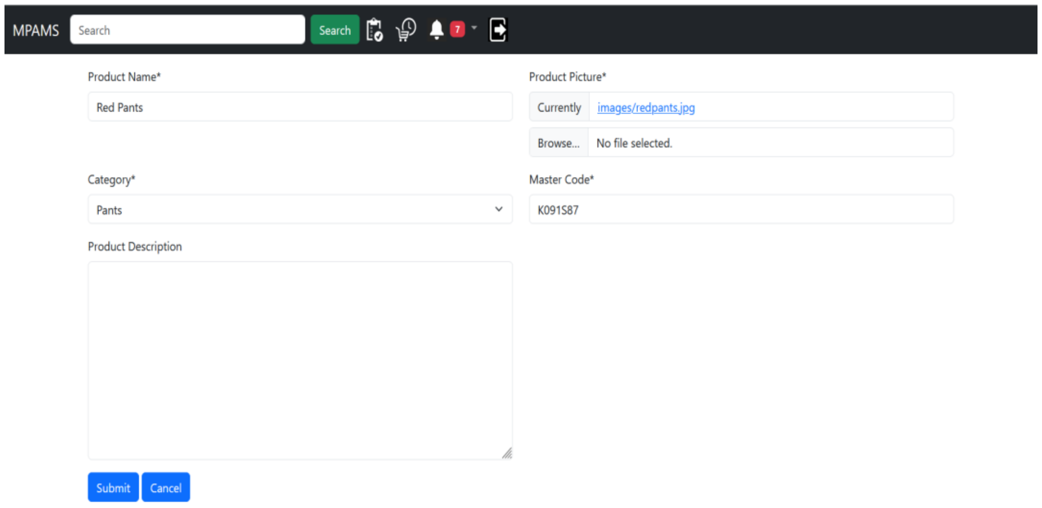


Product Name*
Default Product
Product Picture*
Browse... No file selected.
Category*
Shirt
Master Code*
M000000
Product Description

Figure 8: Inventory Management Product Add Page

F. Inventory Management Edit Product Page

This page is accessed after clicking the *Edit* button of one of the products listed on the Product List Page shown previously. This allows the user to edit the details of a particular product, in case of human error. The page contains the same content as shown in the add product page, with the input fields pre-filled with the information previously inputted. The user may click the *Submit* button once they have made the necessary changes to the product details. For any reason the user may not wish to change the product details, they may click the *Cancel* button.

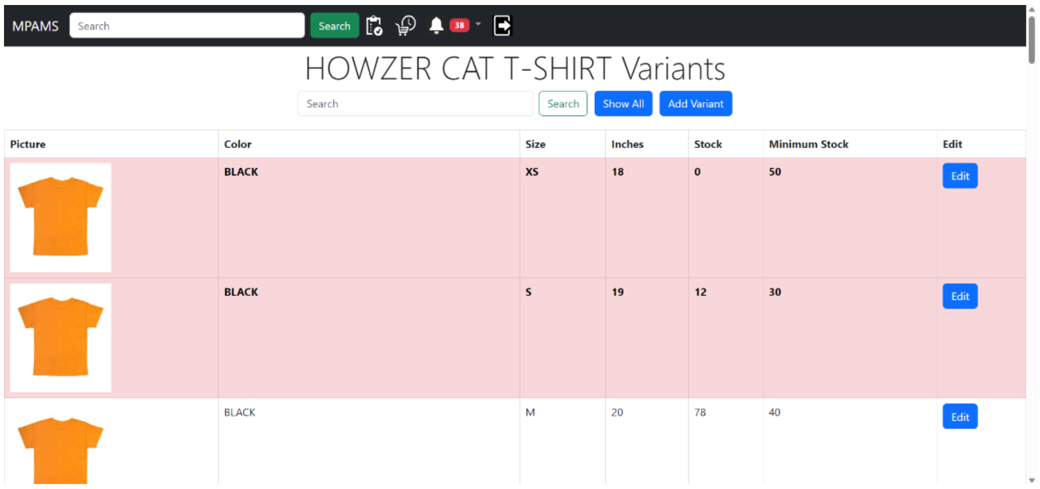


Product Name*	Red Pants
Product Picture*	Currently images/redpants.jpg
	Browse... No file selected.
Category*	Pants
Master Code*	K091S87
Product Description	

Figure 9: Inventory Management Product Edit Page

G. Inventory Management Variations List Page

This page is viewable after clicking the *Variations* button of a particular product on the Inventory Listing Page shown previously. Here the user can view all of the variants of a particular product. When a variant's stock is equal to or less than the minimum stock, the row will be colored red to indicate that it should be dealt with. Similar to the product list page, it contains a search bar for finding specific variants and a *Show All* button once the user wishes to view all the variants again after searching. The user may click the *Add Variant* button to add variants to the system. For each variant, there is a corresponding *Edit* button for editing details.



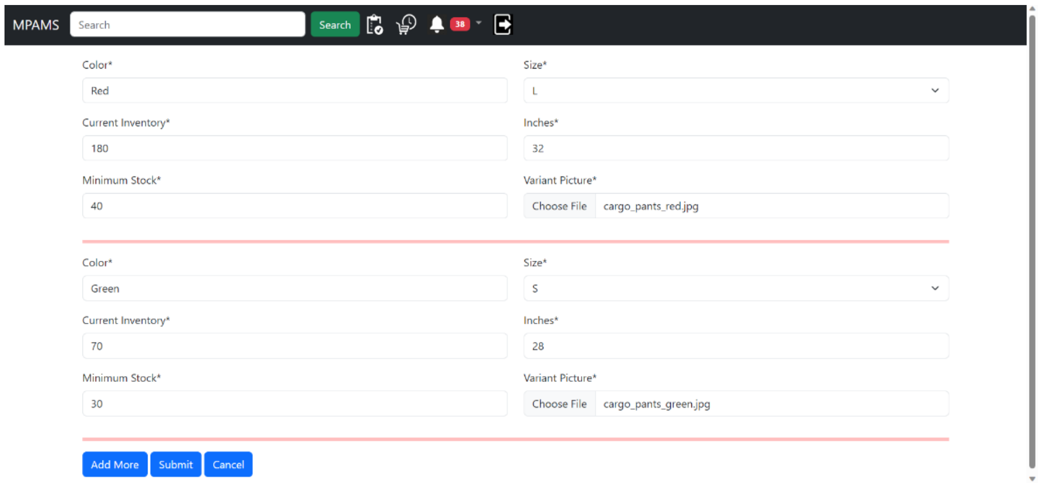
HOWZER CAT T-SHIRT Variants

Picture	Color	Size	Inches	Stock	Minimum Stock	Edit
	BLACK	XS	18	0	50	Edit
	BLACK	S	19	12	30	Edit
	BLACK	M	20	78	40	Edit

Figure 10: Inventory Management Variations List Page

H. Inventory Management Add Variant Page

The page is accessible after clicking the *Add Variant* button in the Variations List Page mentioned above. Here the user may add as many variants as needed. It includes input fields that describe a variant such as color, size, inches, current inventory, picture and minimum stock. If the user wants to add another variant, they may click the *Add More* button. When all of the variant details have been inputted, the user can click the *Submit* button to save all of the variants to the database. Lastly, the user may cancel and return to the variations list page by clicking the *Cancel* button.



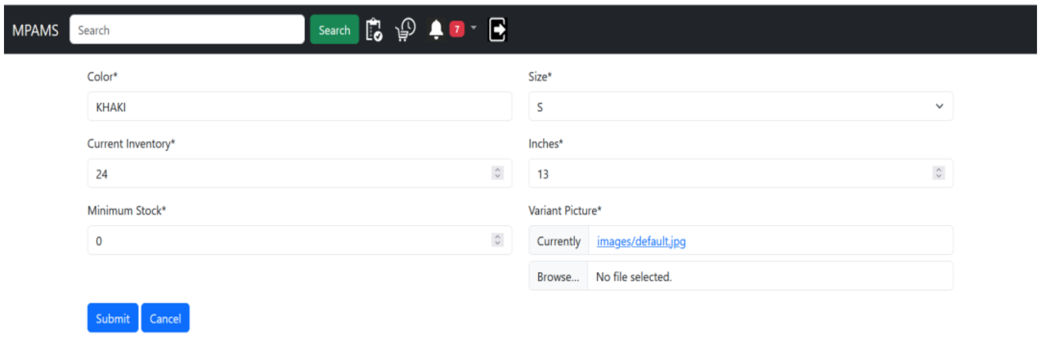
Color* Red
Size* L
Current Inventory* 180
Inches* 32
Minimum Stock* 40
Variant Picture* Choose File cargo_pants_red.jpg

Color* Green
Size* S
Current Inventory* 70
Inches* 28
Minimum Stock* 30
Variant Picture* Choose File cargo_pants_green.jpg

Figure 11: Inventory Management Add Variant Page

I. Inventory Management Edit Variant Page

The page is accessible after clicking an *Edit* button of one of the variants in the Variations List Page. Here the user may edit the details of a particular variant and add more to the current inventory attribute when more units are made available. It includes the same input fields mentioned in the add variant page, but pre-filled with the existing information in the database. When all of the necessary changes have been made, the user can click the *Submit* button to save the changes to the database. Lastly, the user may cancel and return to the variations list page by clicking the *Cancel* button.

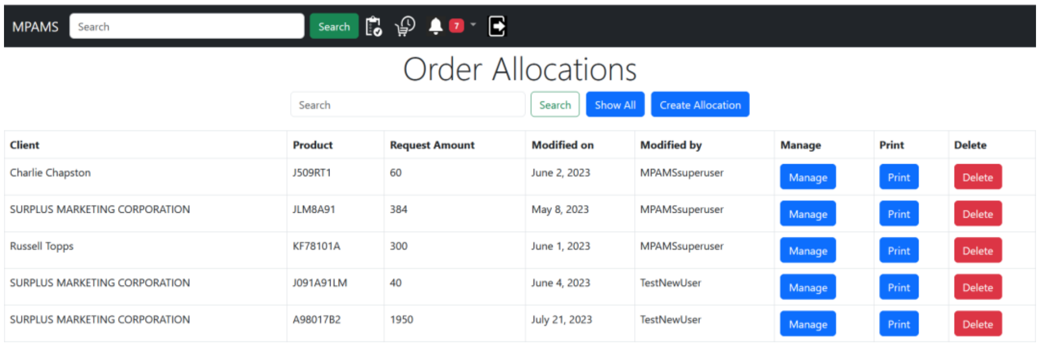


Color*	KHAKI
Size*	S
Current Inventory*	24
Inches*	13
Minimum Stock*	0
Variant Picture*	Currently images/default.jpg
Browse...	No file selected.

Figure 12: Inventory Management Edit Variant Page

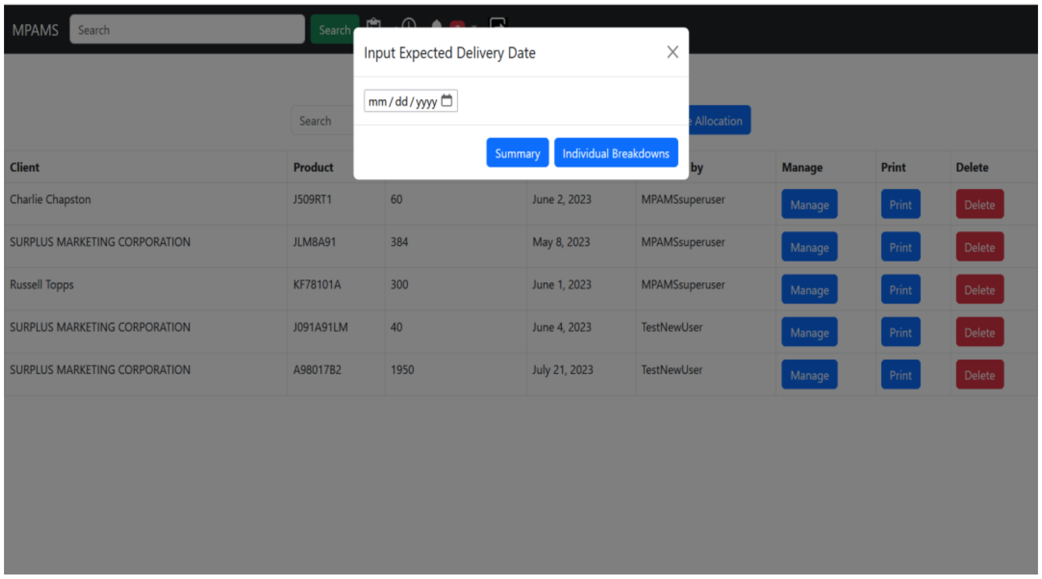
J. Order Allocation List Page

This page can be accessed by users after clicking the *Order Allocation* button from the landing page or the equivalent button (clipboard) found on the taskbar. Here the user can view and manage the orders made in the system. Order details such as the client, product code, request amount, modified by, and modified on attributes can be seen on this page. Found on the page is an additional search bar, for quickly finding a specific order allocation in lieu of manually scrolling through and finding the target allocation. If the user wishes to view all of the allocations again after searching, then they may press the *Show All* button. Clicking on the *Create Allocation* will take the user to the Order Allocation Create Allocation page. For each allocation, there are *Manage*, *Print*, and *Delete* buttons for managing/updating the allocation, printing the corresponding reports, and to cancel the allocation respectively. Clicking the Print button will open a pop-up that allows users to print a summary of the allocation and individual breakdowns for each store. This is done by selecting the expected date of delivery, then clicking either the *Summary* or *Individual Breakdowns* button for the respective reports. If the user is done with acquiring the requested documents, they can click the cross button on the upper right of the pop-up. Clicking on the *Delete* button will open a pop-up asking if the user wishes to delete the allocation. If the user is certain of deleting the allocation, they may click the *Delete* button found on the lower right corner of the pop-up. Otherwise, they may click on the cross button on the upper right corner to cancel the action. Upon deletion of the allocation, any allocated units would be returned to the variant's inventory.



Client	Product	Request Amount	Modified on	Modified by	Manage	Print	Delete
Charlie Chapston	J509RT1	60	June 2, 2023	MPAMSsuperuser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	JLM8A91	384	May 8, 2023	MPAMSsuperuser	Manage	Print	Delete
Russell Topps	KF78101A	300	June 1, 2023	MPAMSsuperuser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	J091A91LM	40	June 4, 2023	TestNewUser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	A98017B2	1950	July 21, 2023	TestNewUser	Manage	Print	Delete

Figure 13: Order Allocation List Page



MPAMS Search Search

Input Expected Delivery Date

mm / dd / yyyy

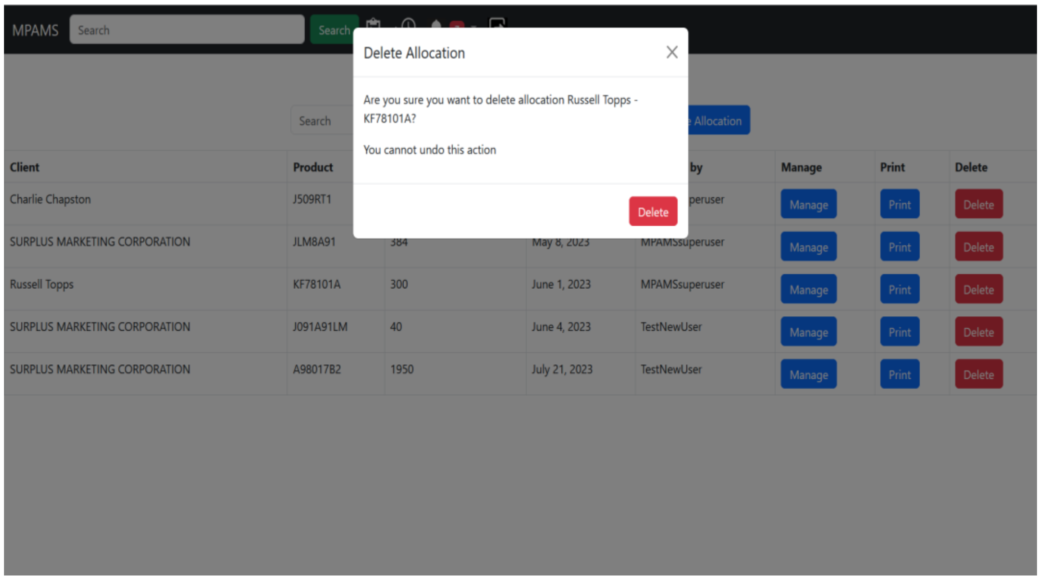
Summary Individual Breakdowns

Search

Allocation

Client	Product			by	Manage	Print	Delete
Charlie Chapston	J509RT1	60	June 2, 2023	MPAMSsuperuser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	JLM8A91	384	May 8, 2023	MPAMSsuperuser	Manage	Print	Delete
Russell Topps	KF78101A	300	June 1, 2023	MPAMSsuperuser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	J091A91LM	40	June 4, 2023	TestNewUser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	A98017B2	1950	July 21, 2023	TestNewUser	Manage	Print	Delete

Figure 14: Order Allocation List Page Print Function



MPAMS Search Search

Delete Allocation

Are you sure you want to delete allocation Russell Topps - KF78101A?

You cannot undo this action

Delete

Search

Allocation

Client	Product			by	Manage	Print	Delete
Charlie Chapston	J509RT1			peruser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	JLM8A91	384	May 8, 2023	MPAMSsuperuser	Manage	Print	Delete
Russell Topps	KF78101A	300	June 1, 2023	MPAMSsuperuser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	J091A91LM	40	June 4, 2023	TestNewUser	Manage	Print	Delete
SURPLUS MARKETING CORPORATION	A98017B2	1950	July 21, 2023	TestNewUser	Manage	Print	Delete

Figure 15: Order Allocation List Page Delete Function

K. Order Allocation Create Allocation Page

This page can be accessed by users after clicking the *Create Allocation* found on the Order Allocation List page. The user can create the allocation here, setting details such as the client, the product requested, the amount requested, barcode/UPC, SKU and category in their respective input fields. Once all of the details have been completed, the user can click the *Submit* button to create and save the allocation to the database. If they wish to cancel the creation page, they can click the *Cancel* button.

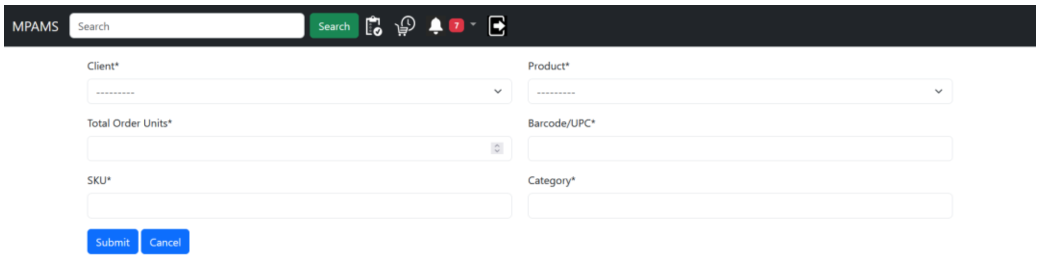
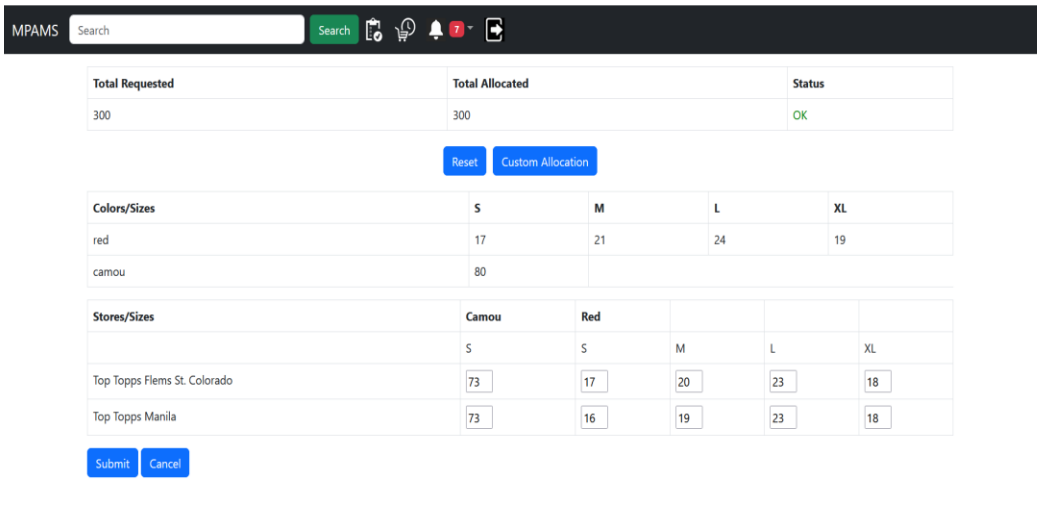


Figure 16: Order Allocation Create Allocation Page

L. Order Allocation Manage Allocation Page

This page can be accessed by users after clicking the *Manage* button of one of the allocations found on the Order Allocation List page, given that the allocation has not been managed yet. Here the user can specify the amounts for each variant of the product to each store that the client has. The first table provides a check on if the total currently allocated matches the total requested amount, and provides a corresponding message as described by the client. The second table provides information on how many units are remaining for each product variant, changing as the amounts allocated changes as well.

The final table is where the user can change the amounts to be allocated for each variant to each store. By default, the system will calculate the amounts such that each store will get proportionally equal amounts of each variant, totalling to the requested amount or allocating all available units if it is less than the requested amount. If after making changes the user wishes to return the previously calculated amount, they may click the *Reset* button. For more specific allocations, the user can click the *Custom Allocation* button, where they can specify which color variants to be allocated and/or the specific distribution of the total request amount. Once satisfied with the distribution of allocated units, they can click the *Submit* button. If they wish to not proceed, then they can click the *Cancel* button.

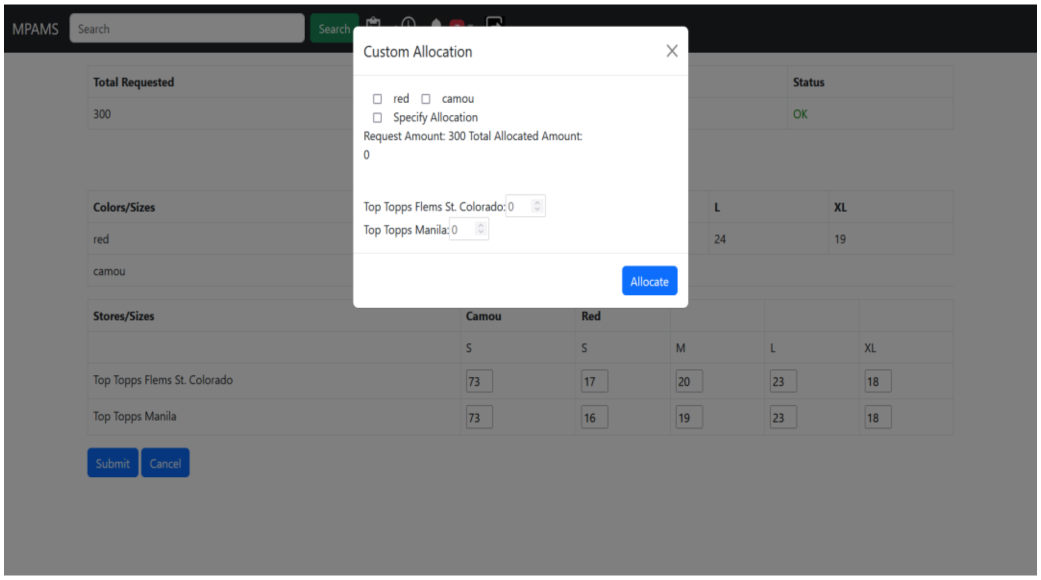


Total Requested	Total Allocated	Status
300	300	OK

Colors/Sizes	S	M	L	XL
red	17	21	24	19
camou	80			

Stores/Sizes	Camou	Red			
	S	S	M	L	XL
Top Topps Flems St. Colorado	73	17	20	23	18
Top Topps Manila	73	16	19	23	18

Figure 17: Order Allocation Manage Allocation Page



Custom Allocation

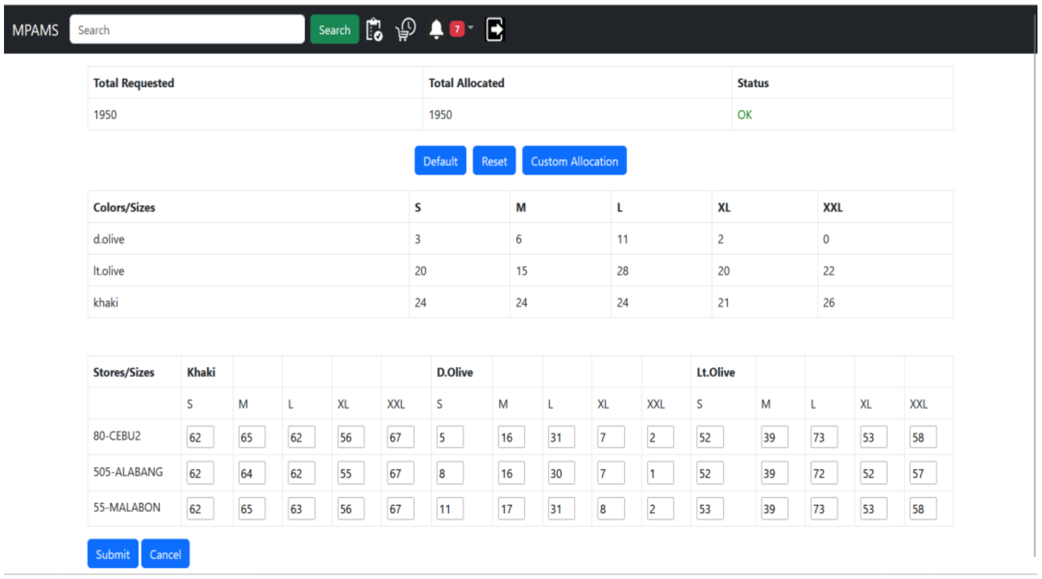
☐ red ☐ camou
☐ Specify Allocation
Request Amount: 300 Total Allocated Amount: 0

Top Topps Flems St. Colorado: 0
Top Topps Manila: 0

Figure 18: Order Allocation Manage Allocation Page Custom Allocation

M. Order Allocation Update Allocation Page

This page can be accessed by users after clicking the *Manage* button of one of the allocations found on the Order Allocation List page, given that the allocation has already been managed. The page is similarly structured to the Order Allocation Manage Allocation (OAMA) page, with a few key differences. Similar to the OAMA page, the user may specify the allocated units for each variant for each client store, with the sole exception being the input fields are pre-filled with the amounts allocated beforehand. Unlike the OAMA page, another option given to the user is the *Default* button. Clicking this button will calculate the units to be allocated of each variant for each store to be proportionally equal, similar to the default mentioned for the OAMA page. Once satisfied with the distribution of allocated units, they can click the *Submit* button to save the changes to the database. If they wish to not proceed, then they can click the *Cancel* button.



Total Requested	Total Allocated	Status
1950	1950	OK

Colors/Sizes	S	M	L	XL	XXL
d.olive	3	6	11	2	0
lt.olive	20	15	28	20	22
khaki	24	24	24	21	26

Stores/Sizes	Khaki					D.Olive					Lt.Olive				
	S	M	L	XL	XXL	S	M	L	XL	XXL	S	M	L	XL	XXL
80-CEBU2	62	65	62	56	67	5	16	31	7	2	52	39	73	53	58
505-ALABANG	62	64	62	55	67	8	16	30	7	1	52	39	72	52	57
55-MALABON	62	65	63	56	67	11	17	31	8	2	53	39	73	53	58

Figure 19: Order Allocation Update Allocation Page

N. Admin Login Page

This can be accessed after adding '/admin' to the base url.

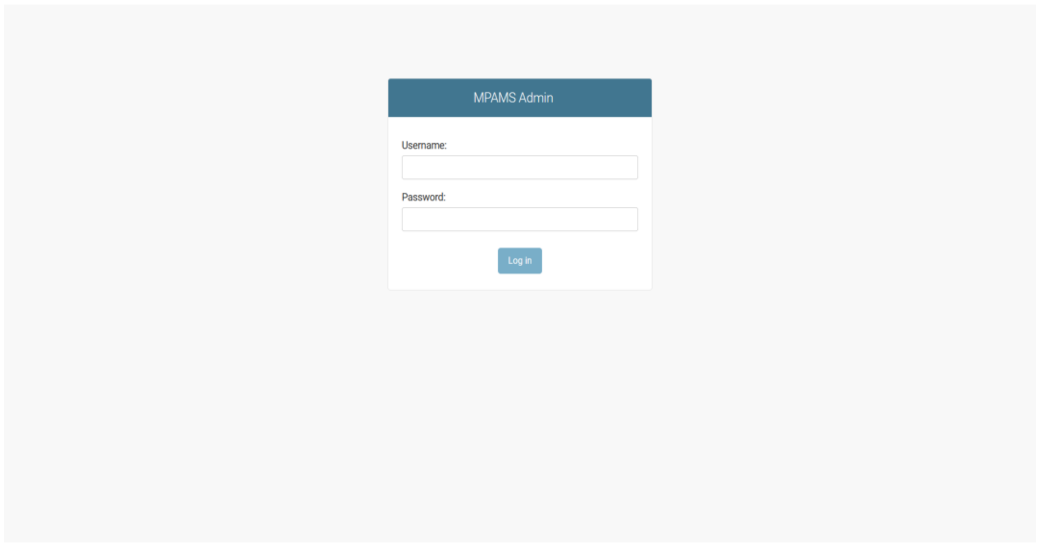
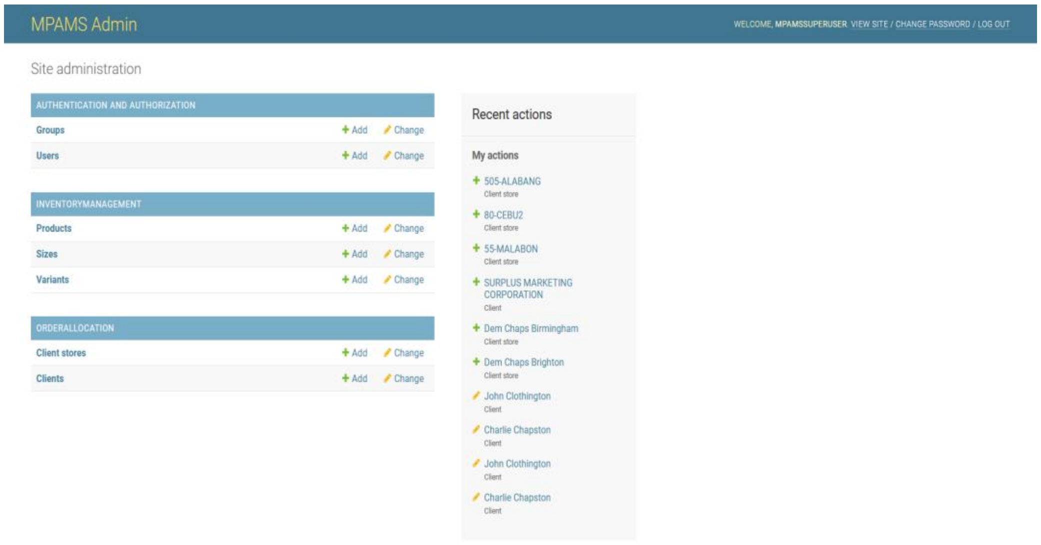


Figure 20: Admin Login Page

O. Admin Landing Page

This can be accessed after a successful login by an admin user. Here the admin can manage user access, manage client-related information, and control option-field choices (i.e. Size attribute) by clicking on the corresponding link (i.e. Client, Client Store, Size). Additionally, the latest additions and changes can be seen on the *Recent actions* panel found on the right side of the page. On the taskbar above, the admin can change their password, access the user pages of the website, and log out.



MPAMS Admin

WELCOME, **MPAMSSUPERUSER**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

INVENTORYMANAGEMENT		
Products	+ Add	Change
Sizes	+ Add	Change
Variants	+ Add	Change

ORDERALLOCATION		
Client stores	+ Add	Change
Clients	+ Add	Change

Recent actions

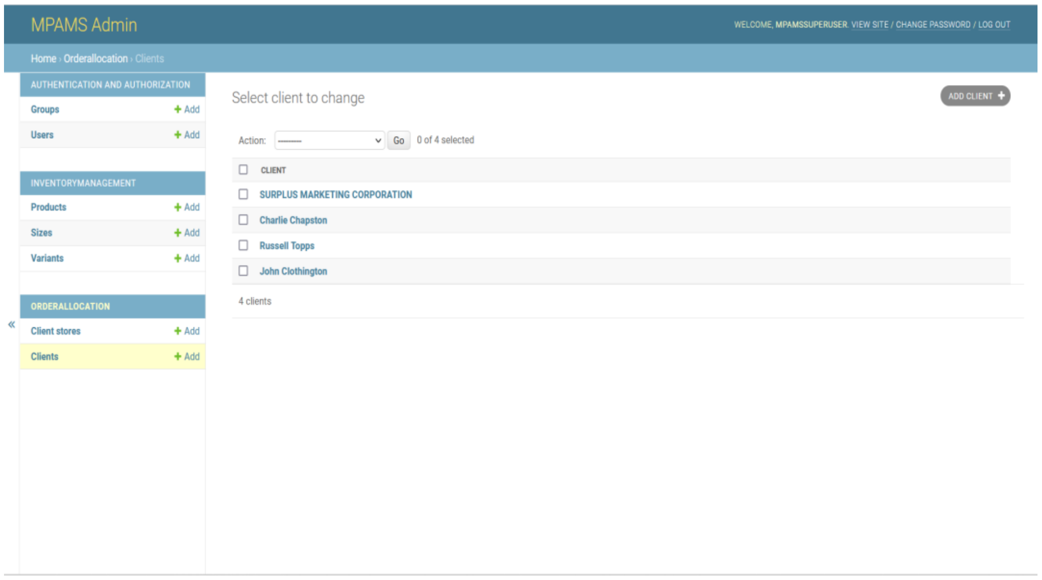
My actions

+ 505-ALABANG
  Client store
+ 80-CEBU2
  Client store
+ 55-MALABON
  Client store
+ SURPLUS MARKETING CORPORATION
  Client
+ Dem Chaps Birmingham
  Client store
+ Dem Chaps Brighton
  Client store
- John Clothington
  Client
- Charlie Chapston
  Client
- John Clothington
  Client
- Charlie Chapston
  Client

Figure 21: Admin Landing Page

P. Admin Client Management

This can be accessed by an admin user after clicking on the *Clients* link found on the panel on the left side of the webpage. The user may add or change the details of the company's client. Details include their name and active status, for if the company's client may discontinue correspondence with the company.



MPAMS Admin

WELCOME, MPAMSSUPERUSER. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Orderallocation › Clients

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add
Users	+ Add

INVENTORYMANAGEMENT	
Products	+ Add
Sizes	+ Add
Variants	+ Add

ORDERALLOCATION	
Client stores	+ Add
Clients	+ Add

Select client to change

ADD CLIENT +

Action: --------- Go 0 of 4 selected

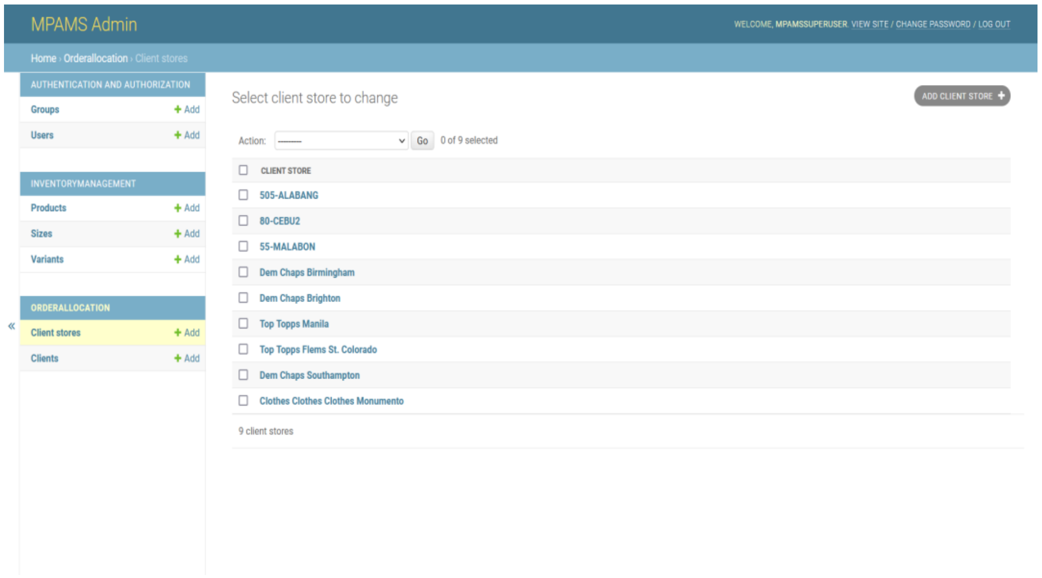
☐ CLIENT
☐ SURPLUS MARKETING CORPORATION
☐ Charlie Chapston
☐ Russell Topps
☐ John Clothington

4 clients

Figure 22: Admin Client Management

Q. Admin Client Store Management

This can be accessed by an admin user after clicking on the *Client stores* link found on the panel on the left side of the webpage. The user may add or change the details of the client's stores. Details include their name, the client it belongs to, and its P.O. box number.



MPAMS Admin

WELCOME, MPAMSSUPERUSER. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Orderallocation › Client stores

AUTHENTICATION AND AUTHORIZATION
- Groups + Add
- Users + Add

INVENTORYMANAGEMENT
- Products + Add
- Sizes + Add
- Variants + Add

ORDERALLOCATION
- Client stores + Add
- Clients + Add

Select client store to change

ADD CLIENT STORE +

Action: ——— Go 0 of 9 selected

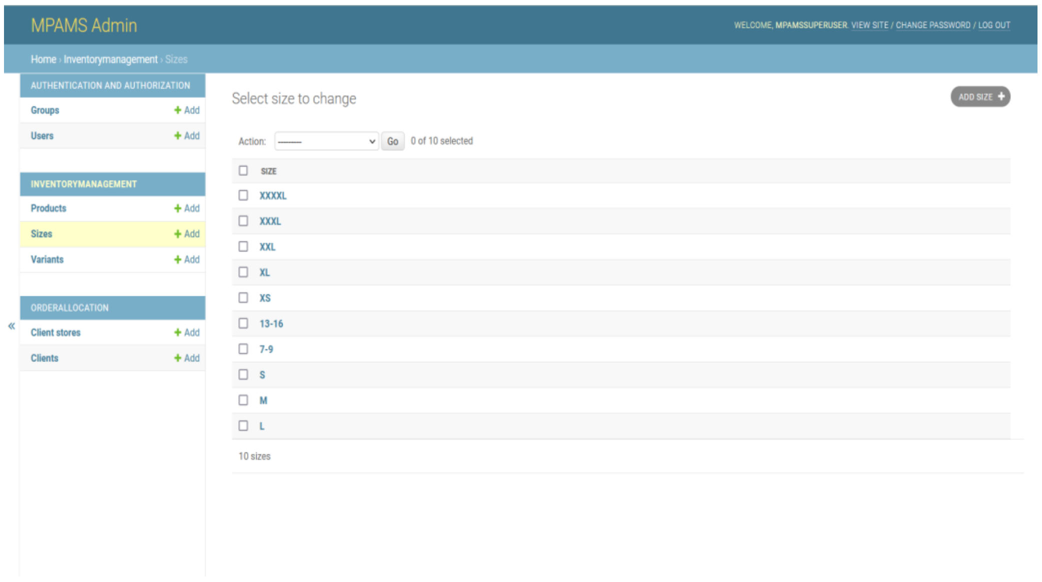
CLIENT STORE
505-ALABANG
80-CEBU2
55-MALABON
Dem Chaps Birmingham
Dem Chaps Brighton
Top Topps Manila
Top Topps Flems St. Colorado
Dem Chaps Southampton
Clothes Clothes Clothes Monumento

9 client stores

Figure 23: Admin Client Store Management

R. Admin Size Management

This can be accessed by an admin user after clicking on the *Sizes* link found on the panel on the left side of the webpage. The user may add or change the sizes available for use in the variant management process. This is done to reduce human error in the system, by controlling the input options for this field. The only attribute included is the name of the size.



Select size to change

Action: ——— Go 0 of 10 selected

SIZE
XXXXL
XXXL
XXL
XL
XS
13-16
7-9
S
M
L

10 sizes

Figure 24: Admin Size Management

VI. Discussions

For both inventory management and order allocation purposes, the client originally has to use multiple spreadsheets to manage their inventories and to allocate units for clients' orders. The developed system solves this by introducing a centralized and uniform system where the company can do all of these actions. The system allows its users to manage their inventory and order allocations of the company in a timely and orderly manner. The regular users can add and change the products and their variants in the system with an intuitive user interface. It informs the users of any shortages in the inventory, in case of any requests from future clients.

Regarding order allocations, the user can create an allocation and specify the amount of units to be allocated for the order. This is done automatically by the system, by default proportionally equal to for each variant to all stores. However, the user may also specify a specific distribution of the total request amount and specify the specific colors to be allocated for the order. This can potentially save time by automating the process and by potentially limiting the amount of human intervention required. This may be the case, as the original method uses ratio and proportion formulas in the aforementioned spreadsheet applications to determine the number of units for each variant to be allocated. This provides a limitation as these outputs float number values, which would then require further intervention by the user. The system does this internally with the additional step of distributing the remaining calculated units, based on the ratio of units already allocated at that point of time. As a result, the distribution of allocated units are made to be as even as possible, both in the case of proportionally equal distribution or in a more specific distribution made by the user.

The system also makes the required reports that the company needs to operate, specifically the summary report of the allocation and the individual breakdowns for each store of the client. The previous method's implementation of this functionality was sufficient for their purposes. This has been made possible as well in the developed system, allowing for ease of use for the users. These documents are required as it allows the company to review and to reassess easily that the distribution is done correctly. For the individual breakdowns, it serves as a label for the warehouse workers to pack the allocated units into a box easily and accurately, and for the target store’s employees to easily determine the contents of the box. Lastly in case of order cancellation, the system allows for the user to easily remove it from the system, and automatically returns allocated units to the inventory.

There have been some issues encountered during the development of this project. The search functionality provided some initial challenge to the developer, as it would be required that it is capable of searching across multiple fields for multiple entities. An initially conceived solution was to call for multiple filter functions, but this proved to be inefficient and a waste of server resources. This was later resolved through the use of Django’s native Q() function, as it allows for multiple filters to be applied to a single function, reducing the number of function and database calls needed.

Another consideration made was on how to present the order allocation data to the user. Initially, this was done by listing all of the orders in a vertical table. This proved to be inconvenient as the number of variants and stores increases, this drastically lengthens the webpage, making it inconvenient for user viewing. The client requested that the data presentation mimic the tabular format that they already use in their spreadsheet implementation. After multiple days and various attempts, the student was able to recreate this approach using extensive use of Django’s formset classes. Compared to a standard web form, the formset allows for multiple forms to be present in a webpage at the same time. In the current implementation, each store was given its own formset with all of the orders for all variants present in the formset. This allows data to be rendered on the web page in a concise and organized manner, allowing for simpler viewing for the user, limiting the amount of scrolling required to a minimum. However, this proves to be ineffective as this renders the page very slowly.

With the solution presented in the previous paragraph, it solves the problem of presenting the data in a more concise manner. However, the page rendering takes a considerable amount of time to do. In a sample dataset provided by the client, it contains 37 variants and 3 client stores, giving a total number of 111 orders. With the earlier referenced implementation, it takes around 40s to render the page. This seems to be consistent with related functions (i.e. Custom Allocation and Default Allocation), as they take similar amounts of time to complete rendering. After much consideration by the developer, it was decided to reconfigure the implementation, and use a more basic approach by manually defining the webpage and reducing the use of Django's template language and related functions. The use of formsets was abandoned in this implementation and resulted in a faster rendering time. As observed in the same dataset, this resulted in a reduction in rendering time by as much as 1200 percent. This is due to Django's implementation of Querysets in the field choices for foreign key fields. This applies to every choice field, With each choice field, a call to the database is made, which increases the rendering time of the webpage.

Another challenge encountered is the change of data type from a text-based field to a foreign key field, specifically the "size" attribute of the variant object. This was due to a human error issue raised by the student's adviser during a consultation session. This proved to be difficult as there was no straightforward method to change the attribute between the two types of input field. Ultimately, the solution came as a result of multiple migrations to the database. First, the developer created a new field to serve as the foreign key field implementation to the database. Afterwards, another migration was created in order to populate the new foreign key table of the database, with the data already existing in the variant object database table. Then the developer applied the two recently made migrations, creating and populating the new field. This avoided the potential issue of having None as a value for any pre-existing record. Lastly, the developer shifted all references to the 'size' attribute in forms and display pages to the newly created attribute. These steps are essential in performing this action, but for completeness and efficiency purposes the old attribute was deleted from the database table. For admin users, they are given access to the ‘size’ attribute choices, allowing for more control in the system and avoids as much human error as possible.

VII. Conclusions

The researcher has been able to complete their SP. The project has been able to solve the problem proposed in the study's statement of the problem, specifically solving the client's need to use multiple spreadsheets to manage their inventory and orders made by their customers. The system allows for the client to manage these in an easy and intuitive manner, to provide accurate distributions of inventory for multiple stores, and to provide a summary for review and individual labels for deliveries.

The researcher has been able to fulfill the objectives of the project, as outlined in the objectives of the study in the thesis proposal. Regular users are able to manage the inventory of the company and to allocate these to orders made by the company's clients. Additionally, they can print the summary and individual reports for an order. The summary report is made available for review by a supervisor and upon approval, this is then provided to the warehouse manager, so they may begin preparations for the order's delivery.

For admin users, they are capable of performing the same functionalities provided to regular users, in addition to their exclusive administrative actions. These include user access, control of the certain attributes, and control of client-related information in the system. This control has been given as it allows for the admin to manage certain inputs into the system, potentially limiting the possibility of human error. Managing access to the client-related information has also been given to administrators for similar reasons. Additionally, this allows for regular users to focus more on the allocation of inventory to the company's orders.

Upon consultation with the client, they find the project to perform suitably and it meets the requirements initially set in the thesis proposal. However, they find that there are improvements that can be made to the project that can further enhance user experience of the system.

VIII. Recommendations

The system does have some limitations after development, which may be sources of improvement. One of these could be the improvement in rendering web pages, particularly in the allocation management related pages. These currently contain multiple elements for processing and manipulating records in the database, at the potential cost of increasing the time needed for rendering. Upon consultation with the client, the current implementation is considered sufficient for their use case. For larger orders, however, this may prove to be inconvenient, as render time may consume a large portion of the time spent in managing the order. In this case, reconfiguring the webpage to render less fields while maintaining the functionalities present in the JavaScript files, could prove to be useful in improving the response time. Another potential consideration could be a review and improvement of the server code.

Another consideration that could be made is the implementation of alternative methods for data entry. The current way of inputting data is simple and generally sufficient for its purpose. However, a possible alternative to this could be the use of a file with a specific format. This method could potentially provide a more sensible and convenient way for inputting data for the users, as these files could be prepared easily using a spreadsheet program, then it could be quickly loaded into the system, without the need for repeated addition by the user.

There are some other considerations that could be made to make the processes within the company more efficient. One such suggestion could be the inclusion of certain information in the system, such as supplier information for each product. These pieces of data could be included in order for the company to easily reference potential sources of the product and make sound decisions in regard to procuring units for additional orders. Regarding the presentation of the product to the user, a photo gallery can be implemented for both products and variants. Similar to ones found on some e-commerce platforms, this could provide multiple angles of the product, allowing for easier reference to the user. Lastly, the reports can be improved to handle allocations dealing with multiple variants and multiple stores. The current implementation of this feature is sufficient for the company's use, but when dealing with larger orders this may prove to be inconvenient. This is due to the implementation attempting to fit the whole table in an A4 size paper on a landscape orientation. The larger orders would force the tables elements to shrink, in order to fit these within the page limits.

The current implementation of the code successfully manages to develop the system described by the client. However, this may not be the optimal solution to the problem posed, as there may still be sections of code that can be improved. There may be optimizations in the code that could still be done, given more time to do so. These may include sections for looping through objects, minimizing database calls, and others. The current implementation could contain unnecessary database calls, potentially due to it being overlooked and/or a lack of understanding from the developer. Other than reducing the amount of time required to render the Order Allocation webpages, the client has no other recommendations for this project.

IX. Bibliography

[1] Aisar, M. M. I., Fauzi, S. S. M., Nabil, F. J. M., Gining, R. J., Suali, A. J., & Sobri, W. A. W. M. (2020, April). The Development of a Web-Based Student Support System Using Java Server Pages and MySQL. In Journal of Physics: Conference Series (Vol. 1529, No. 3, p. 032082). IOP Publishing.

[2] Aru, H., & Priyadharshini, I. (n.d.). Web Based Inventory Management System. *International Research Journal of Engineering and Technology, 08*(04), 140–144.

[3] Brush, K. (2019, November 8). *What is a RDBMS (relational database management system)?* SearchDataManagement. Retrieved January 13, 2022, from https://searchdatamanagement.techtarget.com/definition/RDBMS-relational-database-management-system

[4] Concordia St.Paul. (2021, October 21). *Information systems vs. Information Technology*. CSP Online. Retrieved November 9, 2021, from https://online.csp.edu/resources/article/information-systems-vs-information-technology/.

[5] ECPI University. (n.d.). The Importance of Information Systems in Healthcare [web log]. Retrieved November 10, 2021, from https://www.ecpi.edu/blog/the-importance-of-information-systems-in-healthcare.

[6] Hartono, N., & Erfina, E. (2021). Comparison of stored procedures on Relational Database Management System. *Tech-E, 4*(2), 8–15. https://doi.org/10.31253/te.v4i2.529

[7] Hasan, W. U. (n.d.). Exploring NoSQL (Key-Value) DB vs Relational Database Management System.

[8] *How to create an efficient inventory management system*. Xero. (n.d.). Retrieved January 14, 2022, from https://www.xero.com/ph/resources/small-business-guides/business-management/inventory-management-system/

[9] Ilias, J., Kasim, S., Hassan, R., Mahdin, H., Ramli, A. A., Fudzee, M. F. M., & Aizi, M. (2018). At-Thoyyib Shop Inventory Management System. *Acta Informatica Malaysia (AIM), 2*(2), 12-16.

[10] Katre, H. (2021, November 22). *Tips for setting up Efficient Inventory Management System*. ProfitBooks.net. Retrieved January 14, 2022, from https://www.profitbooks.net/how-to-setup-inventory-management-system/

[11] Misu, M. A. (2019). *Stash - An Inventory Management System* (thesis). Retrieved November 14, 2021, from http://dspace.uiu.ac.bd/handle/52243/1509.

[12] Rawat, B., Purnama, S., & Mulyati (2021). MySQL Database Management System (DBMS) On FTP Site LAPAN Bandung. International Journal of Cyber and IT Service Management (IJCITSM), 1(2), 173-179. Retrieved fromhttps://iiast-journal.org/ijcitsm/index.php/IJCITSM/article/view/47

[13] Saha, S., Basumatary, D., Senapati, A., & Maity, R. (2021). Is there any further scope for improving the efficiency of modern websites? *2021 6th International*

Conference for Convergence in Technology (I2CT). https://doi.org/10.1109/i2ct51068.2021.9418141

[14] Shah, B. K., Kedia, V., & Jha, R. K. (2021). Integrated vendor-managed time efficient application to production of inventory systems. *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, 275–280. https://doi.org/10.1109/icict50816.2021.9358504

[15] Sinha, R. (2019). A Comparative Analysis on different aspects of Database Management System. *Journal of Applied Science and Computations, VI*(II), 2650–2667.

[16] Srivastava, K., Kumar Choubey, D., & Kumar, J. (2020, March). Implementation of Inventory Management System. In *Proceedings of the International Conference on Innovative Computing & Communications (ICICC)*.

[17] The University of Scranton. (2021, October 21). *Role of information systems in the 21st Century organization: Scranton*. The University of Scranton. Retrieved November 9, 2021, from https://elearning.scranton.edu/resources/article/the-role-of-information-systems-in-increasing-productivity/.

[18] *What is DBMS? definition and faqs*. OmniSci. (n.d.). Retrieved January 13, 2022, from https://www.omnisci.com/technical-glossary/dbms

[19] Yellow.Place. (n.d.). *Megapixels Printing Services Inc - Caloocan, Philippines*. Yellow.Place. Retrieved November 10, 2021, from https://yellow.place/en/megapixels-printing-services-inc-caloocan-philippines.

X. Appendix

A. Source Code

MPAMS/settings.py

```
from pathlib import Path

import os


# Build paths inside the project like this:
BASE_DIR / 'subdir'.

BASE_DIR =
Path(__file__).resolve().parent.parent

# SECURITY WARNING: don't run with debug
turned on in production!

DEBUG = True


ALLOWED_HOSTS = []


# Application definition


INSTALLED_APPS = [

'InventoryManagement.apps.Inventorymanagem
entConfig',

    'OrderAllocation.apps.OrderallocationConfig',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'crispy_forms',

    'crispy_bootstrap5',

    'debug_toolbar',

    ]


CRISPY_ALLOWED_TEMPLATE_PACK =
"bootstrap5"


CRISPY_TEMPLATE_PACK = "bootstrap5"


MIDDLEWARE = [


'django.middleware.security.SecurityMiddlewar
e',


'django.contrib.sessions.middleware.SessionMid
dleware',


'django.middleware.common.CommonMiddlew
are',


'django.middleware.csrf.CsrfViewMiddleware',


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


'django.contrib.messages.middleware.MessageM
iddleware',


'django.middleware.clickjacking.XFrameOption
sMiddleware',


#'debug_toolbar.middleware.DebugToolbarMid
dleware',

]


ROOT_URLCONF = 'MPAMS.urls'
```

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTempl
ates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [

'django.template.context_processors.debug',

'django.template.context_processors.request',

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

'django.contrib.messages.context_processors.me
ssages',
            ],
        },
    },


]


WSGI_APPLICATION =
'MPAMS.wsgi.application'


# Database
#
https://docs.djangoproject.com/en/3.2/ref/setting
s/#databases


DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mpams_dev_db',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET
sql_mode='STRICT_TRANS_TABLES'",
        },
    }
}


# Password validation
#
https://docs.djangoproject.com/en/3.2/ref/setting
s/#auth-password-validators


AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAt
tributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.Minim
umLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
#
https://docs.djangoproject.com/en/3.2/topics/i18n/


LANGUAGE_CODE = 'en-us'


TIME_ZONE = 'UTC'


USE_I18N = True


USE_L10N = True


USE_TZ = True


# Static files (CSS, JavaScript, Images)
#
https://docs.djangoproject.com/en/3.2/howto/static-files/


STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR,
'staticfiles')


STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]


# Default primary key field type
#
https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field


DEFAULT_AUTO_FIELD =
'django.db.models.BigAutoField'


LOGIN_REDIRECT_URL = 'main'


MEDIA_ROOT = os.path.join(BASE_DIR,
'media')
MEDIA_URL = '/media/'


INTERNAL_IPS = [
    "127.0.0.1",
]
```

MPAMS/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings
from . import views
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.AppLogInView.as_view(), name="index_page"),
    path('next-page', views.new_page, name="new_page"),
    path('main', views.main_page, name="main"),
    path('logout', views.AppLogOutView.as_view(), name="logout"),
    path('Inventory/', include('InventoryManagement.urls')),
    path('Order/', include('OrderAllocation.urls')),
    path('search', views.search, name="general_search"),
    #path('__debug__/', include("debug_toolbar.urls")),
]


if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

MPAMS/views.py

```
from audioop import reverse
from django.shortcuts import render, redirect
from django.contrib.auth.views import LoginView, LogoutView
from django.contrib.auth.mixins import LoginRequiredMixin
from InventoryManagement.models import Variant, Product
from OrderAllocation.models import Order
from OrderAllocation.models import Allocation
from django.db.models import Q


def index_page(request):
    if request.user.is_authenticated:
        return redirect('main')
    else:
        return render(request, 'base/index.html')


def new_page(request):
    return render(request, 'base/new_page.html')


class AppLogInView(LoginView):
    template_name = 'base/index.html'
    redirect_authenticated_user = True


def main_page(request):
    if request.user.is_authenticated:
        product_notifications = get_product_notifications()
        variant_notifications = get_variation_notifications()
        allocation_notifications = get_allocation_notifications()
        total_notifications = get_notifications_count(product_notifications, variant_notifications, allocation_notifications)


        return render(request, 'base/main.html', {'productnotifications': product_notifications, 'variantnotifications': variant_notifications,
'allocationnotifications': allocation_notifications,
'notificationscount': total_notifications})
    else:
        return redirect('index_page')


class AppLogOutView(LoginRequiredMixin,
LogoutView):
    next_page = '/'


def search(request):
    if request.user.is_authenticated:
        if request.method == "GET":
            searchterm =
request.GET.get('searchterm')
            if searchterm == '':
                searchterm == 'None'
            product_results =
Product.objects.filter(Q(name__icontains=searc
hterm) | Q(stock_code__icontains=searchterm) |
Q(description__icontains=searchterm) |
Q(category__icontains=searchterm) )
            variant_results =
Variant.objects.filter(Q(color__icontains=search
term) |
Q(size__sizename__icontains=searchterm) )
            allocation_results =
Allocation.objects.filter(Q(product__name__ico
ntains=searchterm) |
Q(client__name__icontains=searchterm) )


            product_notifications =
get_product_notifications()
            variant_notifications =
get_variation_notifications()
            allocation_notifications =
get_allocation_notifications()
            total_notifications =
get_notifications_count(product_notifications,
variant_notifications, allocation_notifications)


        return render(request,
'base/searchresults.html', {'product_results':
product_results, 'variant_results': variant_results,
'allocation_results': allocation_results,
'searchterm': searchterm, 'productnotifications':
product_notifications, 'variantnotifications':
variant_notifications, 'allocationnotifications':
allocation_notifications, 'notificationscount':
total_notifications})
    else:
        return redirect(reverse('index_page'))


def get_product_notifications():


    products = Product.objects.all()
    productnotificationlist = []
    for product in products:
        if not product.children.count():
            productnotificationlist.append(product)
    return productnotificationlist


def get_variation_notifications():


    variants = Variant.objects.all()
    variantnotificationlist = []
    for variant in variants:
        if variant.stock <= variant.threshold:
            variantnotificationlist.append(variant)
    return variantnotificationlist
```

```
def get_allocation_notifications():

    allocations = Allocation.objects.all()
    allocationnotificationlist = []
    for allocation in allocations:
        if not
Order.objects.filter(allocation_id=allocation.id):

allocationnotificationlist.append(allocation)
    return allocationnotificationlist


def
get_notifications_count(productnotificationlist,
variantnotificationlist, allocationnotificationlist):
    return len(productnotificationlist) +
len(variantnotificationlist) +
len(allocationnotificationlist)
```

templates/admin/base_site.html

```
{% extends "admin/base.html" %}

{% block title %}{% if subtitle %}{{ subtitle }}
| {% endif %}{{ title }} | {{
site_title|default:_('Django site admin') }}{%
endblock %}

{% block branding %}
<h1 id="site-name"><a href="{% url
'admin:index' %}">MPAMS Admin </a></h1>
{% endblock %}

{% block nav-global %}{% endblock %}
```

templates/base/index.html

```
{% extends "base/loggedoutbase.html" %}

{% load crispy_forms_tags %}

{% block extrascript %}

<style>

    body {

        background-color: ghostwhite;

    }

    #textbox{
        font-family: 'Times New Roman', Times,
serif;
    }

    h1 {
        color: red;
    }

    #contentbox{
        margin-top: 15%;
    }
    #loginbox{
        background-color: white;
        margin-left: 20%;
        margin-right: 20%;
        border-radius: 5px;
        padding-top: 10px;
    }

    pre{
        align-content: center;
    }
</style>

{% endblock %}

{% block content %}

    <div class="container" id="contentbox">
        <div class="row">
            <div class="col-6" id="textbox">
                <h1>MPAMS</h1>
                <h5>MegaPixel's Allocation
Management System</h5>
                <h5>
                    A platform for smartly managing
received orders
                    and allocate inventory
accordingly.
                </h5>
            </div>
            <div class="col-6">
                {% if form.errors %}
                    <p>Your username and password
didn't match. Please try again.</p>
                {% endif %}

                {% if next %}
                    {% if user.is_authenticated %}
                        <p>Your account doesn't have
access to this page. To proceed,
                        please login with an account that
has access.</p>
                    {% else %}
                        <p>Please login to see this
page.</p>
                    {% endif %}
                {% endif %}

                <div id="loginbox">

                    <form method="post" action="{%
url 'index_page' %}">
                        {% csrf_token %}

                        <div class="row mb-3">
                            <div class="col-1"></div>
                            <div class="col-10">
                                {{
form.username|as_crispy_field }}
                            </div>
                            <div class="col-1"></div>
                        </div>
                        <div class="row mb-3">
                            <div class="col-1"></div>
                            <div class="col-10">
                                {{
form.password|as_crispy_field }}
                            </div>
                            <div class="col-1"></div>
                        </div>

                        <div class="row mb-3">
                            <div class="col-1"></div>
                            <div class="col-10">
                                <button type="submit"
class="btn btn-primary col-12">Log
In</button>
                                <input type="hidden"
name="next" value="{{ next }}">
                            </div>
                            <div class="col-1"></div>
                        </div>

                        <div class="row"></div>

                    </form>

                </div>

            </div>
        </div>
    </div>
{% endblock %}
```

templates/base/loggedoutbase.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible"
content="IE=edge">
    <meta name="viewport"
content="width=device-width, initial-scale=1">
    <title>{% block title %}MPAMS{%
endblock %}</title>
    {% load static %}
    <link href="{% static 'css/bootstrap.min.css'
%}" rel="stylesheet">
    <script src="{% static 'js/bootstrap.min.js'
%}"></script>
  </head>

    {% block extrascript %}{% endblock %}
  </head>
  <body>
    <center>
        {% block content %}
        {% endblock %}
    </center>
  </body>
</html>
```

templates/base/loggedinbase.html

```
<!DOCTYPE html>
<html lang="en">

{% load static %}

<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible"
content="IE=edge">
   <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
   <title>{% block title %}MPAMS Main
Page{% endblock %}</title>
   <link href="{% static 'css/bootstrap.min.css'
%}" rel="stylesheet">
   <script src="{% static
'js/bootstrap.bundle.min.js' %}"></script>

   <style>
     #notificon {
       width: 30px;
       height: 30px;
     }

     #logouticon {
       width: 30px;
       height: 30px;
     }

     #invmngicon {
       width: 30px;
       height: 30px;
     }

     #ordericon {
       width: 30px;
       height: 30px;
     }

     #searchbox {
       width: 350px;
     }

     .dropdown-menu {
       overflow-y: scroll;
       height: 300px;
       width: 500px;
     }

   </style>

   {% block extrascript %}
   {% endblock %}
  </head>
<body>
   <nav class="navbar navbar-expand-lg navbar-
dark bg-dark">
      {% block navbarcontent %}
        <div class="container-fluid">
        <a class="navbar-brand" href="{% url
'index_page' %}">MPAMS</a>

        <button class="navbar-toggler"
type="button" data-bs-toggle="collapse" data-
bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle
navigation">

          <span class="navbar-toggler-
icon"></span>

        </button>

        <div class="collapse navbar-collapse"
id="navbarSupportedContent">

          <form class="d-flex" role="search"
action="/search">

            <input class="form-control me-2"
type="search" placeholder="Search" aria-
label="Search" name="searchterm"
id="searchbox">

            <button class="btn btn-success"
type="submit">Search</button>

          </form>

          <ul class="navbar-nav me-auto mb-2
mb-lg-0 nav-fill">

            <li class="nav-item">

              <a class="nav-link active" aria-
current="page" href="{% url
'product_index_page' %}"><img src="{% static
'images/inventory.png' %}" alt="Inventory
Management" id="invmngicon"></a>

            </li>

            <li class="nav-item">

              <a class="nav-link" href="{% url
'order_allocation_index_page' %}"><img
src="{% static 'images/order2.png'%}"
alt="Order Allocation" id="ordericon"></a>

            </li>

            <li class="nav-item dropdown">

              <a class="nav-link dropdown-toggle"
href="#" id="navbarDropdown" role="button"
data-bs-toggle="dropdown" aria-
expanded="false">

                <img src="{% static
'images/bell6.png' %}" id="notificon"
alt="Notifications"> <span class="badge text-
bg-danger">{{notificationscount}}</span>

              </a>

              <ul class="dropdown-menu" aria-
labelledby="navbarDropdown">

                {% if productnotifications or
variantnotifications or allocationnotifications %}

                  {% for productnotification in
productnotifications %}

                    <li><a class="dropdown-item"
href="{% url 'showproduct'
productnotification.id %}">{{
productnotification.name }} has no
variants!</a> </li>

                  {% endfor %}


                  {% for variantnotification in
variantnotifications %}

                    <li><a class="dropdown-item"
href="{% url 'showvariant' variantnotification.id
%}">{{variantnotification}} supply is running
low</a> </li>

                  {% endfor %}


                  {% for allocationnotification in
allocationnotifications %}

                    <li><a class="dropdown-item"
href="{% url 'show_allocation'
allocationnotification.id %}">Allocation
{{allocationnotification}} has not been managed
yet </a> </li>

                  {% endfor %}
                {% else %}
                    <li class="dropdown-item">There
are no notifications currently.</li>
                {% endif %}
              </ul>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="{% url
'logout' %}"><img src="{% static
'images/logout2.png' %}" alt="Log Out"
id="logouticon"></a>
            </li>
          </ul>


        </div>
      </div>
      {% endblock %}
    </nav>
  {% block content %}


  {% endblock %}
  </body>


</html>
```

templates/base/main.html

```
{% extends "base/loggedinbase.html" %}


{% load static %}
```

```
{% block extrascript %}


  <style>


    img {
      width: 495px;
      height: 350px;
    }


    #invmngcardtext{
      color: white;
    }


    #invsuptxt{
      top: 20px;
    }


  </style>


{% endblock %}


{% block navbarcontent %}
<div class="container-fluid">
  <a class="navbar-brand" href="{% url
'index_page' %}">MPAMS</a>
  <button class="navbar-toggler" type="button"
data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle
navigation">
```

```
        <span class="navbar-toggler-
icon"></span>

    </button>

    <div class="collapse navbar-collapse"
id="navbarSupportedContent">

        <form class="d-flex" role="search"
action="/search">

            <input class="form-control me-2"
type="search" placeholder="Search" aria-
label="Search" name="searchterm"
id="searchbox">

            <button class="btn btn-success"
type="submit">Search</button>

        </form>

    <ul class="navbar-nav me-auto mb-2 mb-lg-0
nav-fill">

        <li class="nav-item dropdown">

          <a class="nav-link dropdown-toggle"
href="#" id="navbarDropdown" role="button"
data-bs-toggle="dropdown" aria-
expanded="false">

            <img src="{% static 'images/bell6.png'
%}" alt="Notifications" id="notificon"> <span
class="badge text-bg-
danger">{{notificationscount}}</span>

          </a>

          <ul class="dropdown-menu" aria-
labelledby="navbarDropdown">

            {% if productnotifications or
variantnotifications or allocationnotifications %}

              {% for productnotification in
productnotifications %}

                <li><a class="dropdown-item"
href="{% url 'showproduct'
productnotification.id %}">{{
productnotification.name }} has no
variants!</a> </li>

              {% endfor %}

              {% for variantnotification in
variantnotifications %}

                <li><a class="dropdown-item"
href="{% url 'showvariant' variantnotification.id
%}">{{variantnotification}} supply is running
low</a> </li>

              {% endfor %}


              {% for allocationnotification in
allocationnotifications %}

                <li><a class="dropdown-item"
href="{% url 'show_allocation'
allocationnotification.id %}">Allocation
{{allocationnotification}} has not been managed
yet </a> </li>

              {% endfor %}

            {% else %}

              <li class="dropdown-item">There are
no notifications currently.</li>

            {% endif %}

          </ul>

        </li>

        <li class="nav-item">

          <a class="nav-link" href="{% url 'logout'
%}"><img src="{% static 'images/logout2.png'
%}" alt="Log Out" id="logouticon"></a>

        </li>

    </ul>



    </div>

  </div>

{% endblock %}
{% block content %}

    <h1 class="display-4 text-center">Welcome

            {% if user.is_staff or user.is_superuser %}
                Admin
            {% else %}
                Employee
            {% endif %}
            {{ user.first_name }} {{ user.last_name }}
    </h1><br>

    <div class="row align-items-center">

            <div class="col-1"></div>

            <div class="col-4">
                <a href="{% url 'product_index_page' %}">
                    <div class="card text-bg-dark">
                        <img src="{% static '/images/clothes.jpg' %}" class="card-img" alt="Picture 1" id="invmngpic">
                        <div class="card-img-overlay text-center" id="invmngcardtext">
                            <h3 class="card-title position-absolute top-50 start-50 translate-middle">Inventory Management</h3>
                        </div>
                    </div>
              </a>
            </div>

            <div class="col-2"></div>

            <div class="col-4">
              <a href="{% url 'order_allocation_index_page' %}">
                    <div class="card text-bg-dark text-dark text-center">
                        <img src="{% static 'images/clipboard5.png' %}" class="card-img" alt="Picture 2" id="ordallpic">
                        <div class="card-img-overlay text-center">
                            <h3 class="card-title position-absolute top-50 start-50 translate-middle">Order Allocation</h3>
                        </div>
                    </div>
              </a>
            </div>

            <div class="col-1"></div>

        </div>

{% endblock %}
```

templates/base/searchresults.html

```
{% extends "base/loggedinbase.html" %}
{% block title %} Search Results {% endblock
%}

{% block content %}

    <center>

        <h1 class="display-2">Search Results for
"{{ searchterm }}"</h1>


        {% if product_results or variant_results or
allocation_results %}

        {% if product_results %}
            <h2 class="display-5">Products: </h2>
                {% for result in product_results %}

                    <p><a href="{% url 'showproduct'
result.id %}">{{result.name}}</a> </p>

                {% endfor %}

        {% endif %}

        {% if variant_results %}
            <h2 class="display-5">Variants: </h2>
                {% for result in variant_results %}

                    <p> <a href="{% url 'showvariant'
result.id %}">{{result}}</a> </p>
                {% endfor %}

        {% endif %}

        {% if allocation_results %}
            <h2 class="display-5">Allocations:
</h2>
                {% for result in allocation_results %}

                    <p> <a href="{% url
'show_allocation' result.id %}">{{result}}</a>
</p>

                {% endfor %}

        {% endif %}

        {% else %}

            <h2 class="display-5">There are no
results</h2>

        {% endif %}

    </center>

{% endblock %}
```

InventoryManagement/migrations/0001_initial.py

```
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Product',
            fields=[
                ('id', models.CharField(max_length=50, primary_key=True, serialize=False, verbose_name='SKU Code')),
                ('name', models.CharField(max_length=50, verbose_name='Product Name')),
                ('description', models.TextField(blank=True, verbose_name='Product Description')),
            ],
        ),
        migrations.CreateModel(
            name='Variant',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('color', models.CharField(max_length=50, verbose_name='Color')),
                ('size', models.CharField(max_length=10, verbose_name='Size')),
                ('threshold', models.IntegerField(verbose_name='Minimum Stock')),
                ('product', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='InventoryManagement.product')),
            ],
        ),
    ]
```

InventoryManagement/migrations/0002_auto_20220416_1424.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0001_initial'),
    ]

    operations = [
        migrations.RemoveField(
            model_name='product',
            name='name',
        ),
        migrations.AddField(
            model_name='product',
            name='stock_code',
field=models.CharField(default='M000000', max_length=50, verbose_name='SKU Code'),
        ),
        migrations.AlterField(
            model_name='product',
            name='id',
field=models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID'),
        ),
    ]
```

InventoryManagement/migrations/0003_variant_stock.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0002_auto_20220416_1424'),
    ]

    operations = [
        migrations.AddField(
            model_name='variant',
            name='stock',
            field=models.IntegerField(default='0', verbose_name='Current Inventory'),
        ),
    ]
```

InventoryManagement/migrations/0004_auto_20220608_1113.py

```
from django.db import migrations, models

class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0003_variant_stock'),
    ]

    operations = [
        migrations.AddField(
            model_name='product',
            name='category',
            field=models.CharField(choices=[('SH', 'Shirt'), ('P', 'Pants'), ('AC', 'Accesory'), ('DR', 'Dress'), ('SK', 'Skirt'), ('SR', 'Shorts')], default='SH', max_length=50, verbose_name='Category'),
        ),
        migrations.AddField(
            model_name='product',
            name='name',
            field=models.CharField(default='Default
Product', max_length=50,
verbose_name='Product Name'),
        ),
        migrations.AddField(
            model_name='product',
            name='picture',

field=models.ImageField(default='images/defaul
t.jpg', upload_to='images',
verbose_name='Product Picture'),
        ),
        migrations.AddField(
            model_name='variant',
            name='picture',

field=models.ImageField(default='images/defaul
t.jpg', upload_to='images',
verbose_name='Variant Picture'),
        ),
    ]
```

InventoryManagement/migrations/0005_alter_variant_picture.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement',
'0004_auto_20220608_1113'),
    ]

    operations = [
        migrations.AlterField(
            model_name='variant',
            name='picture',

field=models.ImageField(default='images/defaul
t.jpg', upload_to='variants',
verbose_name='Variant Picture'),
        ),
    ]
```

InventoryManagement/migrations/0006_auto_20220706_2243.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement',
'0005_alter_variant_picture'),
    ]

    operations = [
        migrations.AddField(
            model_name='variant',
            name='inches',
            field=models.IntegerField(default='12',
verbose_name='Inches'),
        ),
        migrations.AlterField(
            model_name='variant',
            name='threshold',
            field=models.IntegerField(default='0',
verbose_name='Minimum Stock'),
        ),
    ]
```

InventoryManagement/migrations/0007_alter_product_stock_code.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement',
'0006_auto_20220706_2243'),
    ]

    operations = [
        migrations.AlterField(
            model_name='product',
            name='stock_code',

field=models.CharField(default='M000000',
max_length=50, verbose_name='Master Code'),
        ),
    ]
```

InventoryManagement/migrations/0008_auto_20230703_1616.py

```
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement',
'0007_alter_product_stock_code'),
    ]

    operations = [
        migrations.CreateModel(
            name='Size',
            fields=[
                ('id',
models.BigAutoField(auto_created=True,
primary_key=True, serialize=False,
verbose_name='ID')),
                ('sizename',
models.CharField(max_length=10,
verbose_name='Size')),
            ],
        ),
        migrations.AddField(
            model_name='variant',
            name='size_link',
            field=models.ForeignKey(null=True,
on_delete=django.db.models.deletion.CASCAD
E, to='InventoryManagement.size'),
        ),
    ]
```

InventoryManagement/migrations/0009_transfer_size.py

```
from django.db import migrations


def link_sizes(apps, schema_editor):
    Variant = apps.get_model('InventoryManagement', 'Variant')
    Size = apps.get_model('InventoryManagement', 'Size')
    for variant in Variant.objects.all():
        size, created = Size.objects.get_or_create(sizename=variant.size)
        variant.size_link = size
        variant.save()


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0008_auto_20230703_1616'),
    ]

    operations = [
        migrations.RunPython(link_sizes),
    ]
```

InventoryManagement/migrations/0010_remove_variant_size.py

```
from django.db import migrations


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0009_transfer_size'),
    ]

    operations = [
        migrations.RemoveField(
            model_name='variant',
            name='size',
        ),
    ]
```

InventoryManagement/migrations/0011_alter_variant_size_link.py

```
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0010_remove_variant_size'),
    ]

    operations = [
        migrations.AlterField(
            model_name='variant',
            name='size_link',

field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='InventoryManagement.size'),
        ),
    ]
```

InventoryManagement/migrations/0012_rename_size_link_variant_size.py

```
from django.db import migrations


class Migration(migrations.Migration):

    dependencies = [
        ('InventoryManagement', '0011_alter_variant_size_link'),
    ]

    operations = [
        migrations.RenameField(
            model_name='variant',
            old_name='size_link',
            new_name='size',
        ),
    ]
```

InventoryManagement/templates/inventory/index_page.html

```
{% extends "base/loggedinbase.html" %}

{% load static %}

{% block title %} MPAMS Inventory Management {% endblock %}

{% block extrascript %}

    <style>
        #actionbox{
            margin-top: 20px;
            margin-bottom: 20px;
        }

        #productsearch{
            width: 350px;
        }

        #showallbtn{
            margin-left: 10px;
            margin-right: 10px;
        }
    </style>

{% endblock %}
{% block content %}


    <div class="d-flex justify-content-center"
id="actionbox">


        <form class="d-flex" role="search"
action="searchproduct">

          <input class="form-control me-2"
type="search" placeholder="Search" aria-
label="Search" name="searchterm"
id="productsearch">

          <button class="btn btn-outline-
success" type="submit">Search</button>

        </form>


        <a href="{% url 'product_index_page'
%}"><button class="btn btn-primary"
id="showallbtn">Show All</button></a>


        <a href="{% url 'add-product'
%}"><button type="button" class="btn btn-
primary">Add New Product</button></a>


    </div>


    {% if product_list %}


      <table class="table table-bordered">


        <thead>


          <th>Image</th>

          <th>Name</th>

          <th>Master Code</th>

          <th>Category</th>

          <th>Product Description</th>

          <th>Edit</th>

          <th>Variations</th>


        </thead>


        <tbody class>

        {% for product in product_list %}


          <tr>


            <td> <img src="{{
product.picture.url }}" alt="Something's wrong"
class="img-fluid rounded-float-start"
width="150" height="150"> </td>

            <td> {{ product.name }} </td>

            <td> {{ product.stock_code }}
</td>

            <td> {{
product.get_category_display }} </td>

            <td> {{ product.description }}
</td>

            <td><a href="{% url 'edit-
product' product.id %}"><button type="button"
class="btn btn-
primary">Edit</button></a></td>

            <td><a href="{% url
'variation_list' product.id %}"><button
type="button" class="btn btn-
primary">Variations</button></a></td>


          </tr>
```

```
                {% endfor %}
                </tbody>

            </table>

        {% else %}

            <p>There are no products in our warehouse.</p>

        {% endif %}

{% endblock %}
```

InventoryManagement/templates/inventory/product.html

```
{% extends "base/loggedinbase.html" %}

{% load crispy_forms_tags %}

{% block title %} Manage Product {% endblock %}



{% block content %}

    <div class="container p-3">

    <form method="post" enctype="multipart/form-data">

{% csrf_token %}

<div class="row">
    <div class="col-lg-6">
        {{form.name|as_crispy_field}}
    </div>

    <div class="col-lg-6">
        {{form.picture|as_crispy_field}}
    </div>
</div>

<div class="row">
    <div class="col-lg-6">
        {{form.category|as_crispy_field}}
    </div>

    <div class="col-lg-6">
        {{form.stock_code|as_crispy_field}}
    </div>
</div>

<div class="row">
    <div class="col-lg-6">
        {{form.description|as_crispy_field}}
    </div>
</div>
```

```
        <button type="submit" class="btn btn-
primary">Submit</button>

        <a class="btn btn-primary" href="{% url
'product_index_page' %}">Cancel</a>


    </form>


    </div>


{% endblock %}
```

InventoryManagement/templates/inventory/variant_list.html

```
{% extends "base/loggedinbase.html" %}


{% block title %} {{ product.stock_code }}
Variants {% endblock %}


{% block extrascript %}


    <style>

        #actionbox{

            margin-bottom: 20px;

        }


        #variantsearch{

            width: 350px;

        }


        #showallbtn{

            margin-left: 10px;

            margin-right: 10px;

        }


    </style>


{% endblock %}


{% block content %}

        <h1 class="display-5 text-center">{{
product.name }} Variants</h1>


        <div class="d-flex justify-content-center"
id="actionbox">


            <form class="d-flex" role="search"
action="{% url 'searchvariant' product.id %}">

                <input class="form-control me-2"
type="search" placeholder="Search" aria-
label="Search" name="searchterm"
id="variantsearch">

                <button class="btn btn-outline-
success" type="submit">Search</button>

            </form>


            <a href="{% url 'variation_list'
product.id %}"><button class="btn btn-primary"
id="showallbtn">Show All</button></a>


            <a href="{% url 'add-variant' product.id
%}"><button type="button" class="btn btn-
primary">Add Variant</button></a>


        </div>
        {% if variant_list %}

          <table class="table table-bordered">

            <tr>

              <th>Picture</th>
              <th>Color</th>
              <th>Size</th>
              <th>Inches</th>
              <th>Stock</th>
              <th>Minimum Stock</th>
              <th>Edit</th>

            </tr>

            {% for variant in variant_list %}

              {% if variant.stock <=
variant.threshold %}

                <tr class="table-danger">

                  <td><img src="{{
variant.picture.url }}" class="img-fluid rounded-
float-start" width="150" height="150"></td>
                  <td> <p class="fw-bold">  {{
variant.color }} </p> </td>
                  <td> <p class="fw-bold"> {{
variant.size }} </p> </td>
                  <td> <p class="fw-bold"> {{
variant.inches }} </p></td>
                  <td> <p class="fw-bold"> {{
variant.stock }} </p> </td>
                  <td> <p class="fw-bold"> {{
variant.threshold }} </p> </td>
                  <td><a href="{% url 'edit-
variant' product.id variant.id %}"><button
type="button" class="btn btn-
primary">Edit</button></a></td>

                </tr>

              {% else %}

                <tr>

                  <td><img src="{{
variant.picture.url }}" class="img-fluid rounded-
float-start" width="150" height="150"></td>
                  <td> {{ variant.color }} </td>
                  <td> {{ variant.size }} </td>
                  <td> {{ variant.inches }} </td>
                  <td> {{ variant.stock }} </td>
                  <td> {{ variant.threshold }}
</td>
                  <td><a href="{% url 'edit-
variant' product.id variant.id %}"><button
type="button" class="btn btn-
primary">Edit</button></a></td>

                </tr>

              {% endif %}

            {% endfor %}
            </table>

        {% else %}

        <center>
            <p>There are no variants for this
product</p>
        </center>

        {% endif %}

{% endblock %}
```

InventoryManagement/templates/inventory/variant_update.html

```
{% extends "base/loggedinbase.html" %}

{% load crispy_forms_tags %}

{% block title %} Update Variant {% endblock
%}

{% block content %}

    <div class="container p-3">

    <form method="post"
enctype="multipart/form-data" id="form-
container">

{% csrf_token %}

<div class="row">

    <div class="col-lg-6">
        {{form.color|as_crispy_field}}
    </div>

    <div class="col-lg-6">
        {{form.size|as_crispy_field}}
    </div>

</div>

<div class="row">

    <div class="col-lg-6">
        {{form.stock|as_crispy_field}}
    </div>

    <div class="col-lg-6">
        {{form.inches|as_crispy_field}}
    </div>

</div>

<div class="row">

    <div class="col-lg-6">
```

```
            {{form.threshold|as_crispy_field}}
        </div>

        <div class="col-lg-6">
            {{form.picture|as_crispy_field}}
        </div>

    </div>

    <button type="submit" class="btn btn-primary">Submit</button>
    <a class="btn btn-primary" href="{% url 'variation_list' product_id %}">Cancel</a>
  </form>

  </div>

{% endblock %}
```

InventoryManagement/templates/inventory/variant.html

```
{% extends "base/loggedinbase.html" %}

{% load crispy_forms_tags %}

{% block title %} Create Variant {% endblock %}

{% block extrascript %}

<style>

  hr{
      color: red;
      border-width: 5px;
  }

</style>

{% endblock %}

{% block content %}

  <div class="container p-3">

  <form method="post" enctype="multipart/form-data" id="form-container">

      {% csrf_token %}

      {{ variantformset.management_form }}
      {% for form in variantformset %}

      <div class="variant-form">
          <div class="row">
              <div class="col-lg-6">
                  {{form.color|as_crispy_field}}
              </div>

              <div class="col-lg-6">
```

```
            {{form.size|as_crispy_field}}
        </div>
    </div>

    <div class="row">
        <div class="col-lg-6">
            {{form.stock|as_crispy_field}}
        </div>

        <div class="col-lg-6">
            {{form.inches|as_crispy_field}}
        </div>
    </div>

    <div class="row">
        <div class="col-lg-6">
            {{form.threshold|as_crispy_field}}
        </div>

        <div class="col-lg-6">
            {{form.picture|as_crispy_field}}
        </div>
    </div>

    <hr>
</div>

{% endfor %}
<button type="button" id="AddMore"
class="btn btn-primary">Add More</button>
    <button type="submit" class="btn btn-
primary">Submit</button>
    <a class="btn btn-primary" href="{% url
'variation_list' product_id %}">Cancel</a>
</form>


</div>


<script>

    let variantForm =
document.querySelectorAll(".variant-form")
    let formset =
document.querySelector("#form-container")
    let addButton =
document.querySelector("#AddMore")
    let totalForms =
document.querySelector("#id_form-
TOTAL_FORMS")
    let formNum = variantForm.length - 1

    addButton.addEventListener('click',
addMoreForm)

    function addMoreForm(e){
        e.preventDefault()

        let newForm =
variantForm[0].cloneNode(true)
        let formRegex = RegExp(`form-(\\d){1}-
`, 'g')

        formNum++
            newForm.innerHTML = 
newForm.innerHTML.replace(formRegex, 
`form-${formNum}-`)

            formset.insertBefore(newForm, 
addButton)


totalForms.setAttribute('value',`${formNum+1}`
)


        }


    </script>

{% endblock %}
```

InventoryManagement/admin.py

```
from django.contrib import admin

from .models import Product, Variant, Size


# Register your models here.

admin.site.register(Product)

admin.site.register(Variant)

admin.site.register(Size)
```

InventoryManagement/apps.py

```
from django.apps import AppConfig


class InventorymanagementConfig(AppConfig):

    default_auto_field = 
'django.db.models.BigAutoField'

    name = 'InventoryManagement'
```

InventoryManagement/forms.py

```
from django.forms import ModelForm, 
modelformset_factory, BaseFormSet

from .models import Variant


class VariantForm(ModelForm):

    class Meta:

        model = Variant

        fields = ['color', 'size' ,'stock', 'inches', 
'threshold', 'picture']


class BaseVariantFormSet(BaseFormSet):


    def clean(self):

        if any(self.errors):

            return


VariantFormSet = 
modelformset_factory(model=Variant, 
form=VariantForm, 
formset=BaseVariantFormSet)
```

InventoryManagement/models.py

```
from django.db import models


SHIRT = "SH"

PANTS = "P"

ACCESSORIES = "AC"

DRESS = "DR"
```

```
SKIRT = "SK"
SHORTS = "SR"


CLOTHES_CATEGORIES = (
    (SHIRT, "Shirt"),
    (PANTS, "Pants"),
    (ACCESSORIES, "Accesory"),
    (DRESS, "Dress"),
    (SKIRT, "Skirt"),
    (SHORTS, "Shorts")
)


# Create your models here.


class Size(models.Model):
    sizename =
models.CharField(verbose_name="Size",
max_length=10)


    def __str__(self):
        return self.sizename


class Product(models.Model):
    name =
models.CharField(verbose_name="Product
Name", default="Default Product",
max_length=50)
    stock_code =
models.CharField(verbose_name="Master
Code", max_length=50, default="M000000")
    description =
models.TextField(verbose_name="Product
Description", blank=True)
    picture =
models.ImageField(upload_to="images",
default="images/default.jpg",
verbose_name="Product Picture")
    category = models.CharField(max_length=50,
verbose_name="Category",
choices=CLOTHES_CATEGORIES,
default=SHIRT)


    def __str__(self):
        return self.stock_code


class Variant(models.Model):
    product = models.ForeignKey(Product,
on_delete=models.CASCADE,
related_name="children")
    picture =
models.ImageField(upload_to="variants",
default="images/default.jpg",
verbose_name="Variant Picture")
    color =
models.CharField(verbose_name="Color",
max_length=50)
    size = models.ForeignKey('Size',
on_delete=models.CASCADE)
    stock =
models.IntegerField(verbose_name="Current
Inventory", default="0")
    threshold =
models.IntegerField(verbose_name="Minimum
Stock", default="0")
    inches =
models.IntegerField(verbose_name="Inches",
default="12")


    def __str__(self):
        showstring = '{} - Size {} {}
variant'.format(self.product, self.size, self.color)
```

```
        return showstring
```

InventoryManagement/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.ProductListView.as_view(), name="product_index_page"),
    path('add-product', views.ProductCreateView.as_view(), name="add-product"),
    path('edit-product/<int:pk>', views.ProductEditView.as_view(), name="edit-product"),
    path('searchproduct', views.ProductFilterDisplayView.as_view(), name="searchproduct"),
    path('variations/<int:product_id>', views.VariationListView.as_view(), name="variation_list"),

path('variations/<int:product_id>/searchvariant', views.VariationUpdateListView.as_view(), name="searchvariant"),
    path('variations/<int:product_id>/add-variant', views.VariationCreateView.as_view(), name="add-variant"),
    path('variations/<int:product_id>/edit-variant/<int:pk>', views.VariationUpdateView.as_view(), name="edit-variant"),
    path('showproduct/<int:product_id>', views.ProductSpecificView.as_view(), name="showproduct"),
    path('showvariation/<int:variant_id>', views.VariantSpecificView.as_view(), name="showvariant"),
]
```

InventoryManagement/views.py

```
from django.forms.models import BaseModelForm
from django.http import HttpResponse, HttpResponseRedirect
from django.views.generic.edit import CreateView, UpdateView
from django.views.generic.list import ListView
from django.contrib.auth.mixins import LoginRequiredMixin
from InventoryManagement.models import Product, Variant
from django.db.models import Q
from MPAMS import views as mainviews
from .forms import VariantFormSet, VariantForm


# Create your views here.
class ProductListView(LoginRequiredMixin, ListView):
    model = Product
    template_name = 'inventory/index_page.html'
    context_object_name = 'product_list'
    queryset = Product.objects.all().order_by('-id')


    def get_context_data(self, **kwargs):
        context =
super().get_context_data(**kwargs)

        variant_list = Variant.objects.all()

        variant_shortages = []

        for variant in variant_list:

            if variant.stock <= variant.threshold:

                product =
Product.objects.get(id=variant.product_id)

                stock_code = product.stock_code

                variant_shortage = {

                    'variant': variant,

                    'stock_code': stock_code

                }


variant_shortages.append(variant_shortage)

        context['shortages'] = variant_shortages

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist

        context['variantnotifications'] =
variantnotificationlist

        context['allocationnotifications'] =
allocationnotificationlist

        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)

        return context
```

```
class ProductCreateView(LoginRequiredMixin,
CreateView):

    model = Product

    template_name = 'inventory/product.html'

    success_url = '/Inventory/'

    fields = ['name', 'picture', 'category',
'stock_code', 'description']


    def form_valid(self, form: BaseModelForm) -
> HttpResponse:

        formname = form.cleaned_data['name']

        formstock_code =
form.cleaned_data['stock_code']

        matches =
Product.objects.filter(Q(name__iexact =
formname) | Q(stock_code__iexact =
formstock_code))

        if matches:

            form.add_error('name', 'THIS
PRODUCT ALREADY EXISTS')

            return self.form_invalid(form)

        return super(ProductCreateView,
self).form_valid(form)


    def get_context_data(self, **kwargs):

        context =
super().get_context_data(**kwargs)

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist
```

```
        context['variantnotifications'] = variantnotificationlist

        context['allocationnotifications'] = allocationnotificationlist

        context['notificationscount'] = mainviews.get_notifications_count(productnotificationlist, variantnotificationlist, allocationnotificationlist)

        return context


class ProductEditView(LoginRequiredMixin, UpdateView):

    model = Product

    template_name = 'inventory/product.html'

    success_url = '/Inventory/'

    fields = ['name', 'picture', 'category', 'stock_code', 'description']


    def form_valid(self, form: BaseModelForm) -> HttpResponse:

        formname = form.cleaned_data['name']

        formstock_code = form.cleaned_data['stock_code']

        matches = Product.objects.filter(Q(name__iexact = formname) | Q(stock_code__iexact = formstock_code))

        if matches:

            currProduct = Product.objects.get(pk = self.kwargs['pk'])

            if currProduct not in matches:

                form.add_error('name', 'THIS PRODUCT ALREADY EXISTS')

                return self.form_invalid(form)

        return super(ProductEditView, self).form_valid(form)


    def get_context_data(self, **kwargs):

        context = super().get_context_data(**kwargs)

        productnotificationlist = mainviews.get_product_notifications()

        variantnotificationlist = mainviews.get_variation_notifications()

        allocationnotificationlist = mainviews.get_allocation_notifications()

        context['productnotifications'] = productnotificationlist

        context['variantnotifications'] = variantnotificationlist

        context['allocationnotifications'] = allocationnotificationlist

        context['notificationscount'] = mainviews.get_notifications_count(productnotificationlist, variantnotificationlist, allocationnotificationlist)

        return context


class VariationListView(LoginRequiredMixin, ListView):

    model = Variant

    template_name = 'inventory/variant_list.html'

    context_object_name = 'variant_list'


    def get_context_data(self, **kwargs):

        context = super().get_context_data(**kwargs)

        # Can ignore Class Product error, still works in implementation
        context['product'] = Product.objects.get(pk=self.kwargs['product_id'])
        productnotificationlist = mainviews.get_product_notifications()
        variantnotificationlist = mainviews.get_variation_notifications()
        allocationnotificationlist = mainviews.get_allocation_notifications()
        context['productnotifications'] = productnotificationlist
        context['variantnotifications'] = variantnotificationlist
        context['allocationnotifications'] = allocationnotificationlist
        context['notificationscount'] = mainviews.get_notifications_count(productnotificationlist, variantnotificationlist, allocationnotificationlist)
        return context


    def get_queryset(self):
        return Variant.objects.filter(product_id=self.kwargs['product_id']).order_by('color', 'inches')


class VariationCreateView(LoginRequiredMixin, CreateView):
    model = Variant
    template_name = 'inventory/variant.html'
    form_class = VariantForm
    #fields = ['color', 'size', 'stock', 'inches', 'threshold', 'picture']


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['product_id'] = self.kwargs['product_id']
        productnotificationlist = mainviews.get_product_notifications()
        variantnotificationlist = mainviews.get_variation_notifications()
        allocationnotificationlist = mainviews.get_allocation_notifications()
        context['variantformset'] = VariantFormSet()
        context['productnotifications'] = productnotificationlist
        context['variantnotifications'] = variantnotificationlist
        context['allocationnotifications'] = allocationnotificationlist
        context['notificationscount'] = mainviews.get_notifications_count(productnotificationlist, variantnotificationlist, allocationnotificationlist)
        return context


    def post(self, request, *args, **kwargs):
        formset = VariantFormSet(request.POST, request.FILES)
        if formset.is_valid():
            for form in formset:
                form.instance.product_id = self.kwargs['product_id']
            return self.form_valid(formset)


    def form_valid(self, formset):
        #form.instance.product_id =
self.kwargs['product_id']

        for form in formset:

            form.instance_product_id =
self.kwargs['product_id']

            form.instance.save()

        url = '/Inventory/variations/{}'

        success_url =
url.format(self.kwargs['product_id'])

        return HttpResponseRedirect(success_url)


    def get_success_url(self):

        url = '/Inventory/variations/{}'

        success_url =
url.format(self.kwargs['product_id'])

        return success_url


class
VariationUpdateView(LoginRequiredMixin,
UpdateView):

    model = Variant

    template_name =
'inventory/variant_update.html'

    fields = ['color', 'size', 'stock', 'inches',
'threshold', 'picture']


    def get_context_data(self, **kwargs):

        context =
super().get_context_data(**kwargs)

        context['product_id'] =
self.kwargs['product_id']

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist

        context['variantnotifications'] =
variantnotificationlist

        context['allocationnotifications'] =
allocationnotificationlist

        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)

        return context


    def form_valid(self, form):

        form.instance.product_id =
self.kwargs['product_id']

        matches =
Variant.objects.filter(Q(color__iexact=form.clea
ned_data['color']) &
Q(size=form.cleaned_data['size']) &
Q(inches=form.cleaned_data['inches']) &
Q(product=self.kwargs['product_id']))

        if matches:

            currVariant = Variant.objects.get(pk =
self.kwargs['pk'])

            if currVariant not in matches:

                form.add_error('color', 'THIS
VARIANT ALREADY EXISTS')

                return self.form_invalid(form)

        return super().form_valid(form)


    def get_success_url(self):

        url = '/Inventory/variations/{}'
        success_url = 
url.format(self.kwargs['product_id'])
        return success_url


class 
ProductFilterDisplayView(LoginRequiredMixin, 
ListView):
    model = Product
    template_name = 'inventory/index_page.html'
    context_object_name = 'product_list'


    def get_queryset(self):
        searchterm = 
self.request.GET.get('searchterm')
        if searchterm == '':
            searchterm = 'None'
        results = 
Product.objects.filter(Q(name__icontains=searc
hterm) | Q(stock_code__icontains=searchterm) | 
Q(description__icontains=searchterm) | 
Q(category__icontains=searchterm) )
        return results


    def get_context_data(self, **kwargs):
        context = 
super().get_context_data(**kwargs)
        variant_list = Variant.objects.all()
        variant_shortages = []
        for variant in variant_list:
            if variant.stock <= variant.threshold:
                product = 
Product.objects.get(id=variant.product_id)
                stock_code = product.stock_code
                variant_shortage = {
                    'variant': variant,
                    'stock_code': stock_code
                }

variant_shortages.append(variant_shortage)
        context['shortages'] = variant_shortages
        productnotificationlist = 
mainviews.get_product_notifications()
        variantnotificationlist = 
mainviews.get_variation_notifications()
        allocationnotificationlist = 
mainviews.get_allocation_notifications()
        context['productnotifications'] = 
productnotificationlist
        context['variantnotifications'] = 
variantnotificationlist
        context['allocationnotifications'] = 
allocationnotificationlist
        context['notificationscount'] = 
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist, 
allocationnotificationlist)
        return context


class 
VariationUpdateListView(LoginRequiredMixin, 
ListView):
    model = Variant
    template_name = 'inventory/variant_list.html'
    context_object_name = 'variant_list'


    def get_context_data(self, **kwargs):
        context = 
super().get_context_data(**kwargs)
```

```
        # Can ignore Class Product error, still
works in implementation

        context['product'] =
Product.objects.get(pk=self.kwargs['product_id']
)

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist

        context['variantnotifications'] =
variantnotificationlist

        context['allocationnotifications'] =
allocationnotificationlist

        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)

        return context


    def get_queryset(self):

        searchterm =
self.request.GET.get('searchterm')

        if searchterm == ":

            searchterm = 'None'

        variants =
Variant.objects.filter(Q(product_id=self.kwargs[
'product_id']), Q(size__icontains=searchterm) |
Q(color__icontains=searchterm) )

        return variants
```

```
class
ProductSpecificView(LoginRequiredMixin,
ListView):

    model = Product

    template_name = 'inventory/index_page.html'

    context_object_name = 'product_list'


    def get_queryset(self):

        product =
Product.objects.filter(pk=self.kwargs['product_i
d'])

        return product


    def get_context_data(self, **kwargs):

        context =
super().get_context_data(**kwargs)

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist

        context['variantnotifications'] =
variantnotificationlist

        context['allocationnotifications'] =
allocationnotificationlist

        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)

        return context
```

```
class
VariantSpecificView(LoginRequiredMixin,
ListView):

    model = Variant

    template_name = 'inventory/variant_list.html'

    context_object_name = 'variant_list'


    def get_queryset(self):

        variant =
Variant.objects.filter(pk=self.kwargs['variant_id'
])

        return variant


    def get_context_data(self, **kwargs):

        context =
super().get_context_data(**kwargs)

        # Can ignore Class Product error, still
works in implementation

        variant =
Variant.objects.get(pk=self.kwargs['variant_id'])

        context['product'] =
Product.objects.get(pk=variant.product_id)

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist

        context['variantnotifications'] =
variantnotificationlist

        context['allocationnotifications'] =
allocationnotificationlist

        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)

        return context
```

OrderAllocation/migrations/0001_initial.py

```
import datetime

from django.conf import settings

from django.db import migrations, models

import django.db.models.deletion


class Migration(migrations.Migration):


    initial = True


    dependencies = [

migrations.swappable_dependency(settings.AU
TH_USER_MODEL),

        ('InventoryManagement',
'0003_variant_stock'),

    ]


    operations = [

        migrations.CreateModel(

            name='Allocation',

            fields=[

                ('id',
models.BigAutoField(auto_created=True,
primary_key=True, serialize=False,
verbose_name='ID')),

                ('request_amount',
models.IntegerField(verbose_name='Total Order
Amount')),

            ],

        ),

        migrations.CreateModel(

            name='Client',

            fields=[

                ('id',
models.BigAutoField(auto_created=True,
primary_key=True, serialize=False,
verbose_name='ID')),

                ('name',
models.CharField(max_length=100,
verbose_name='Client Name')),

            ],

        ),

        migrations.CreateModel(

            name='Order',

            fields=[

                ('id',
models.BigAutoField(auto_created=True,
primary_key=True, serialize=False,
verbose_name='ID')),

                ('amount',
models.IntegerField(verbose_name='Allocation
Amount')),

                ('order_date',
models.DateField(default=datetime.date.today,
verbose_name='Modified At')),

                ('allocation_id',
models.ForeignKey(on_delete=django.db.model
s.deletion.CASCADE,
to='OrderAllocation.allocation')),

                ('user_id',
models.ForeignKey(on_delete=django.db.model
s.deletion.CASCADE,
to=settings.AUTH_USER_MODEL)),

                ('variant_id',
models.ForeignKey(on_delete=django.db.model
s.deletion.CASCADE,
to='InventoryManagement.variant')),

            ],

        ),

        migrations.CreateModel(

            name='ClientStore',

            fields=[

                ('id',
models.BigAutoField(auto_created=True,
primary_key=True, serialize=False,
verbose_name='ID')),

                ('location',
models.CharField(max_length=100,
verbose_name='Store Name')),

                ('client_id',
models.ForeignKey(on_delete=django.db.model
s.deletion.CASCADE,
to='OrderAllocation.client')),

            ],

        ),

        migrations.AddField(

            model_name='allocation',

            name='client_id',


field=models.ForeignKey(on_delete=django.db.
models.deletion.CASCADE,
to='OrderAllocation.client'),

        ),

        migrations.AddField(

            model_name='allocation',
            name='product_id',
field=models.ForeignKey(on_delete=django.db.
models.deletion.CASCADE,
to='InventoryManagement.product'),
        ),
    ]
```

OrderAllocation/migrations/0002_auto_20220520_2230.py

```
from django.db import migrations


class Migration(migrations.Migration):

    dependencies = [
        ('OrderAllocation', '0001_initial'),
    ]

    operations = [
        migrations.RenameField(
            model_name='allocation',
            old_name='client_id',
            new_name='client',
        ),
        migrations.RenameField(
            model_name='allocation',
            old_name='product_id',
            new_name='product',
        ),
        migrations.RenameField(
            model_name='clientstore',
            old_name='client_id',
            new_name='client',
        ),
        migrations.RenameField(
            model_name='order',
            old_name='allocation_id',
            new_name='allocation',
        ),
        migrations.RenameField(
            model_name='order',
            old_name='user_id',
            new_name='user',
        ),
        migrations.RenameField(
            model_name='order',
            old_name='variant_id',
            new_name='variant',
        ),
    ]
```

OrderAllocation/migrations/0003_auto_20220521_1057.py

```
import datetime
from django.conf import settings
from django.db import migrations, models
import django.db.models.deletion
```

```
class Migration(migrations.Migration):


    dependencies = [

migrations.swappable_dependency(settings.AUTH_USER_MODEL),
        ('OrderAllocation', '0002_auto_20220520_2230'),
    ]


    operations = [
        migrations.RemoveField(
            model_name='order',
            name='order_date',
        ),
        migrations.RemoveField(
            model_name='order',
            name='user',
        ),
        migrations.AddField(
            model_name='allocation',
            name='order_date',

field=models.DateField(default=datetime.date.today, verbose_name='Modified At'),
        ),
        migrations.AddField(
            model_name='allocation',
            name='user',
            field=models.ForeignKey(default=1, on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL, verbose_name='Modified by'),
        ),
        migrations.AddField(
            model_name='client',
            name='is_active',

field=models.BooleanField(default=True, verbose_name='Active Status'),
        ),
        migrations.AlterField(
            model_name='clientstore',
            name='client',

field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='OrderAllocation.client', verbose_name='Owner'),
        ),
    ]
```

OrderAllocation/migrations/0004_order_client_store.py

```
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):


    dependencies = [
        ('OrderAllocation', '0003_auto_20220521_1057'),
    ]
    operations = [
        migrations.AddField(
            model_name='order',
            name='client_store',
            field=models.ForeignKey(default=1,
on_delete=django.db.models.deletion.CASCAD
E, to='OrderAllocation.clientstore'),
        ),
    ]
```

OrderAllocation/migrations/0005_alter_allocation_request_amount.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('OrderAllocation',
'0004_order_client_store'),
    ]

    operations = [
        migrations.AlterField(
            model_name='allocation',
            name='request_amount',

field=models.IntegerField(verbose_name='Total
Order Units'),
        ),
    ]
```

OrderAllocation/migrations/0006_auto_20230425_1116.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('OrderAllocation',
'0005_alter_allocation_request_amount'),
    ]

    operations = [
        migrations.AddField(
            model_name='allocation',
            name='barupc',
            field=models.CharField(default=-
20211157749, max_length=20,
verbose_name='Barcode/UPC'),
            preserve_default=False,
        ),
        migrations.AddField(
            model_name='allocation',
            name='category',
            field=models.CharField(default='JOH',
max_length=10, verbose_name='Category'),
            preserve_default=False,
        ),
        migrations.AddField(
            model_name='allocation',
            name='sku',
field=models.CharField(default=10024843,
max_length=10, verbose_name='SKU'),
            preserve_default=False,
        ),
        migrations.AddField(
            model_name='clientstore',
            name='ponum',

field=models.CharField(default=1418031,
max_length=10, verbose_name='PO Number'),
            preserve_default=False,
        ),
    ]
```

OrderAllocation/templates/allocation/allocationcreateview.html

```
{% extends "base/loggedinbase.html" %}

{% load crispy_forms_tags %}

{% block title %} Create Allocation {%
endblock %}

{% block content %}

    <div class="container p-3">

        <form method="post">
            {% csrf_token %}

            <div class="row">

                <div class="col-lg-6">
                    {{form.client|as_crispy_field}}
                </div>

                <div class="col-lg-6">
                    {{form.product|as_crispy_field}}
                </div>

            </div>

            <div class="row align-items-center">

                <div class="col-lg-6">

{{form.request_amount|as_crispy_field}}
                </div>

                <div class="col-lg-6">
                    {{form.barupc|as_crispy_field}}
                </div>

            </div>

            <div class="row align-items-center">

                <div class="col-lg-6">
                    {{form.sku|as_crispy_field}}
                </div>
                <div class="col-lg-6">
                    {{form.category|as_crispy_field}}
                </div>

            </div>

            <button type="submit" class="btn btn-primary">Submit</button>
            <a class="btn btn-primary" href="{% url 'order_allocation_index_page' %}">Cancel</a>
        </form>
    </div>

{% endblock %}
```

OrderAllocation/templates/allocation/allocationmanage_test.html

```
{% extends "base/loggedinbase.min.html" %}

{% load static %}

{% block title %} Manage Allocation {% endblock %}

{% block extrascript %}

<link rel="stylesheet" href="{% static 'css/allocationupdate.css' %}">

{% endblock %}

{% block content %}
<div class="container p-3">
    <table class="table table-bordered">
        <thead>
            <th>
                Total Requested
            </th>
            <th>
                Total Allocated
            </th>
            <th>
                Status
            </th>
        </thead>
        <tbody>
            <tr>
                <td id="total_request_amount">
                    {{ allocation.request_amount }}
                </td>
                <td id="running_total">
                    {{ total }}
                </td>
                {% if status == 'ok' %}
                    <td id="notiftext" style="color: green;">OK</td>
                {% elif status == 'notallowed' %}
                    <td id="notiftext" style="color: red;">NOT ALLOWED</td>
                {% elif status == 'slightlyok' %}
                <td id="notiftext" style="color: orange;">OK WITH WARNING</td>

              {% else %}

                <td id="notiftext" style="color: red;">NOT OK</td>

              {% endif %}

            </tr>

          </tbody>

        </table>


        <div class="d-flex justify-content-center" id="actionbox">

          <a href="{% url 'allocation_page' primarykey %}"><button type="button" class="btn btn-primary" id="resetbutton">Reset</button></a>

          <button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#exampleModal">

            Custom Allocation

          </button>

        </div>


        <a id="RemainingStock" href="#Orders" class="section-header">Remaining Stock</a>

        <table class="table table-bordered" id="statsTable" hidden>


            <thead>


              <th>

                Colors/Sizes

              </th>

              {% for size in uniqueSizes %}


                <th>{{ size }}</th>


              {% endfor %}


            </thead>


            <tbody>


              {% for group in variantStats %}


                <tr>


                  {% for entry in group %}


                    {% if forloop.first %}


                      <td>{{ entry }}</td>


                    {% else %}


                      <p name="variant" hidden>{{ entry.displayVariant }}</p>

                      <p id="{{entry.displayVariant}}-total" hidden>{{ entry.remainingTotal }}</p>

                      <td id="{{entry.displayVariant}}">{{ entry.remainingTotal }}</td>


                    {% endif %}
                    {% endfor %}

                  </tr>

                {% endfor %}

              </tbody>

            </table>

        </div>

        <div class="container p-3">

        <div class="modal fade"
id="exampleModal" tabindex="-1" aria-
labelledby="exampleModalLabel" aria-
hidden="true">

            <div class="modal-dialog">

            <div class="modal-content">

              <div class="modal-header">

              <h5 class="modal-title"
id="exampleModalLabel">Custom
Allocation</h5>

              <button type="button" class="btn-
close" data-bs-dismiss="modal" aria-
label="Close"></button>

              </div>

              <form method="post" action="{% url
'allocatebycolor' primarykey %}">

                <div class="modal-body">

                    {% csrf_token %}

                    {% for color in colors %}

                      <input type="checkbox"
id="{{color}}" name="{{color}}"
value="{{color}}">

                      <label
for="{{color}}">{{color}}</label>

                    {% endfor %}

                    <br>

                    <input type="checkbox"
id="specific_allocation_toggle"
onclick="toggleStoreSpecificAllocations()"
value="toggle" name="toggle">

                    <label
for="specific_allocation_toggle">Specify
Allocation</label>

                    <br>Request Amount: {{
allocation.request_amount }} Total Allocated
Amount: <p
id="total_allocate_amount">0</p><br>

                    {% for store in stores %}

                      <label for="{{store}}-
number"> {{store}}: </label>

                      <input type="number"
id="{{store}}-number" name="{{store}}-
number" value="0" size="5"
class="store_specific_allocations"
disabled><br>

                    {% endfor %}
                    </div>

                    <div class="modal-footer">
                        <button type="submit"
class="btn btn-primary">Allocate</button>
                    </div>
                </form>
            </div>
            </div>
        </div>

        <form method="post" action="{% url
'allocation_page' primarykey %}">

            {% csrf_token %}

            <a id="Orders" href="#RemainingStock"
class="section-header">Orders</a>

            <table class="table table-bordered"
id="formsTable" hidden>

                <thead>
                    <th>Stores/Sizes</th>

                    {% for color in displayColors %}

                        {% for entry in color %}

                            {% if forloop.first %}
                                <th>{{ entry }}</th>
                            {% else %}

                                <th></th>

                            {% endif %}

                        {% endfor %}

                    {% endfor %}

                </thead>

                <tbody>

                    <tr>

                        <td></td>

                        {% for size in sizes %}
                            <td>{{ size }}</td>
                        {% endfor %}

                    </tr>

                    {% for entry in form_data %}

                        <tr>

                            {% for subentry in entry %}
                    {% if forloop.first %}

                        <td>{{ subentry }}</td>

                    {% else %}

                        {% for elements in
subentry %}

                            <td>
                                <!--
                                <select name="{{
elements.client_store }}-{{ elements.variant }}-
allocation">
                                    <option value="{{
elements.allocation }}" selected>{{
elements.allocation }}</option>
                                </select>
                                -->
                                <select name="{{
elements.client_store }}-{{ elements.variant }}-
variant" id="{{ elements.client_store }}-{{
elements.variant }}-variant">
                                    <option value="{{
elements.variant.id }}" selected>{{
elements.variant }}</option>
                                </select>
                                <select name="{{
elements.client_store }}-{{ elements.variant }}-
clientStore" id="{{ elements.client_store }}-{{
elements.variant }}-clientStore">
                                    <option value="{{
elements.client_store.id }}" id="{{
elements.client_store }}" name="{{
elements.client_store }}" selected>{{
elements.client_store }}</option>
                                </select>
                                <input type="text"
value="{{ elements.amount }}" id="{{
elements.client_store }}-{{ elements.variant }}-
formValue" name="{{ elements.client_store }}-
{{ elements.variant }}-formValue">
                                <p id="{{
elements.client_store }}-{{ elements.variant }}-
formValueOld" hidden>{{ elements.amount
}}</p>
                            </td>

                        {% endfor %}

                    {% endif %}

                {% endfor %}

            </tr>

        {% endfor %}

    </tbody>

</table>
<button type="submit" class="btn btn-
primary">Submit</button>
<a class="btn btn-primary" href="{% url
'order_allocation_index_page' %}">Cancel</a>
</form>
```

```
        <script src="{% static
'js/allocationmanage_test.js' %}"></script>

    </div>

{% endblock %}
```

OrderAllocation/templates/allocation/allocationupdate_test.html

```
{% extends "base/loggedinbase.min.html" %}

{% load static %}

{% block title %} Update Allocation {%
endblock %}

{% block extrascript %}

    <link rel="stylesheet" href="{% static
'css/allocationupdate.css' %}">

{% endblock %}

{% block content %}
    <div class="container p-3">
        <table class="table table-bordered">
            <thead>
                <th>
                    Total Requested
                </th>
                <th>
                    Total Allocated
                </th>
                <th>
                    Status
                </th>
            </thead>
            <tbody>
                <tr>
                    <td id="total_request_amount">
                        {{ allocation.request_amount }}
                    </td>
                    <td id="running_total">
                        {{ total }}
                    </td>
                    {% if status == 'ok' %}
                        <td id="notiftext" style="color:
green;">OK</td>
                    {% elif status == 'notallowed' %}
                        <td id="notiftext" style="color:
red;">NOT ALLOWED</td>
                    {% elif status == 'slightlyok' %}
                        <td id="notiftext" style="color:
orange;">OK WITH WARNING</td>
                    {% else %}
                        <td id="notiftext" style="color:
red;">NOT OK</td>
                    {% endif %}
                </tr>
            </tbody>
        </table>
        <div class="d-flex justify-content-center"
id="actionbox">

            <a href="{% url
'update_allocation_default' primarykey
%}"><button type="button" class="btn btn-
primary" id="defbutton">Default</button></a>

            <a href="{% url 'allocation_page'
primarykey %}"><button type="button"
class="btn btn-primary"
id="resetbutton">Reset</button></a>

            <button type="button" class="btn btn-
primary" data-bs-toggle="modal" data-bs-
target="#exampleModal">

                Custom Allocation

            </button>

        </div>


        <a href="#Orders" id="RemainingStock"
class="section-header">Remaining Stock</a>

        <table class="table table-bordered"
id="statsTable" hidden>


                <thead>


                    <th>

                        Colors/Sizes

                    </th>


                    {% for size in uniqueSizes %}


                        <th>{{ size }}</th>


                    {% endfor %}

                </thead>


                <tbody>


                    {% for group in variantStats %}


                        <tr>


                            {% for entry in group %}


                                {% if forloop.first %}


                                    <td>{{ entry }}</td>


                                {% else %}


                                    <p name="variant"
hidden>{{ entry.displayVariant }}</p>

                                    <p
id="{{entry.displayVariant}}-total" hidden>{{
entry.remainingTotal }}</p>

                                    <td
id="{{entry.displayVariant}}">{{
entry.remainingTotal }}</td>


                                {% endif %}


                            {% endfor %}


                        </tr>
                {% endfor %}


              </tbody>


            </table>


        </div>


          <div class="container p-3">

          <div class="modal fade"
id="exampleModal" tabindex="-1" aria-
labelledby="exampleModalLabel" aria-
hidden="true">

            <div class="modal-dialog">

            <div class="modal-content">

              <div class="modal-header">

              <h5 class="modal-title"
id="exampleModalLabel">Custom
Allocation</h5>

              <button type="button" class="btn-
close" data-bs-dismiss="modal" aria-
label="Close"></button>

              </div>

              <form method="post" action="{% url
'allocatebycolor' primarykey %}"
onsubmit="return
submitCheckCustomAllocation()">

                <div class="modal-body">


                  {% csrf_token %}


                  {% for color in colors %}

                      <input type="checkbox"
id="{{color}}" name="{{color}}"
value="{{color}}">

                      <label
for="{{color}}">{{color}}</label>

                    {% endfor %}


                    <br>


                    <input type="checkbox"
id="specific_allocation_toggle"
onclick="toggleStoreSpecificAllocations()"
value="toggle" name="toggle">

                    <label
for="specific_allocation_toggle">Specify
Allocation</label>



                    <br>Request Amount: {{
allocation.request_amount }} Total Allocated
Amount: <p
id="total_allocate_amount">0</p><br>


                    {% for store in stores %}


                      <label for="{{store}}-
number"> {{store}}: </label>

                      <input type="number"
id="{{store}}-number" name="{{store}}-
number" value="0" size="5"
class="store_specific_allocations"
disabled><br>


                    {% endfor %}


                </div>
```

```
                    <div class="modal-footer">
                        <button type="submit"
class="btn btn-primary">Allocate</button>
                    </div>
                </form>
            </div>
            </div>
        </div>

        <form method="post" action="{% url
'update_allocation' primarykey %}">

            {% csrf_token %}

            <a id="Orders" href="#RemainingStock"
class="section-header">Orders</a>

            <table class="table table-bordered"
id="formsTable" hidden>

                <thead>
                    <th>Stores/Sizes</th>

                    {% for color in displayColors %}

                        {% for entry in color %}

                            {% if forloop.first %}
                                <th>{{ entry }}</th>
                            {% else %}
                                <th></th>

                            {% endif %}

                        {% endfor %}

                    {% endfor %}

                </thead>

                <tbody>

                    <tr>

                        <td></td>

                        {% for size in sizes %}
                            <td>{{ size }}</td>
                        {% endfor %}

                    </tr>

                    {% for entry in form_data %}

                        <tr>

                            {% for subentry in entry %}

                                {% if forloop.first %}
                    <td>{{ subentry }}</td>

                  {% else %}

                    {% for elements in subentry %}

                      <p id="{{ elements.order }}-{{ elements.order.client_store }}-variant" hidden>{{ elements.variant }}</p>
                      <td>
                        <select>
                          <option value="{{ elements.order }}" id="{{ elements.order }}" name="{{ elements.order }}" selected>{{ elements.order }}</option>
                        </select>
                        <input type="text" value="{{ elements.amount }}" id="{{ elements.order }}-{{ elements.order.client_store }}-formValue" name="{{ elements.order }}-{{ elements.order.client_store }}-formValue">
                        <p id="{{ elements.order }}-{{ elements.order.client_store }}-formValueOld" hidden>{{ elements.amount }}</p>
                      </td>

                    {% endfor %}

                  {% endif %}

                {% endfor %}
                </tr>

              {% endfor %}

            </tbody>

          </table>
          <button type="submit" class="btn btn-primary">Submit</button>
          <a class="btn btn-primary" href="{% url 'order_allocation_index_page' %}">Cancel</a>
        </form>

        <script src="{% static 'js/allocationupdate_test.js' %}"></script>

    </div>

{% endblock %}
```

OrderAllocation/templates/allocation/order_allocation_index_page.html

```
{% extends "base/loggedinbase.html" %}

{% block title %} Order Allocation Main Page {% endblock %}

{% block extrascript %}
```

```
<style>
    #actionbox{
        margin-bottom: 20px;
    }

    #allocationsearch{
        width:350px;
    }

    #showallbtn{
        margin-left: 10px;
        margin-right: 10px;
    }

    .loader{
        border: 16px solid #f3f3f3;
        border-top: 16px solid #3498db;
        border-radius: 50%;
        width: 120px;
        height: 120px;
        animation: spin 2s linear infinite;
        position: absolute;
        top: 50%;
        left: 50%;
    }

    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
    </style>

{% endblock %}

{% block content %}

        <h1 class="display-5 text-center">Order
Allocations</h1>

        <div class="d-flex justify-content-center"
id="actionbox">
            <form class="d-flex" role="search"
action="{% url 'searchallocation'%}">
                <input class="form-control me-2"
type="search" placeholder="Search" aria-
label="Search" name="searchterm"
id="allocationsearch">
                <button class="btn btn-outline-
success" type="submit">Search</button>
            </form>

            <a href="{% url
'order_allocation_index_page' %}"><button
class="btn btn-primary" id="showallbtn">Show
All</button></a>

            <a href="{% url
'order_allocation_create_view' %}"><button
type="button" class="btn btn-primary">Create
Allocation</button></a>

        </div>
            {% if allocation_list %}
                <table class="table table-bordered">
                    <thead>
```

```
<th>Client</th>
<th>Product</th>
<th>Request Amount</th>
<th>Modified on</th>
<th>Modified by</th>
<th>Manage</th>
<th>Print</th>
<th>Delete</th>
</thead>

<tbody>

{% for allocation in allocation_list %}
<tr>
<td> {{ allocation.client }} </td>
<td> {{ allocation.product }} </td>
<td> {{ allocation.request_amount }} </td>
<td> {{ allocation.order_date }} </td>
<td> {{ allocation.user }} </td>
<td> <a href="{% url 'allocation_page' allocation.id %}"> <button type="button" class="btn btn-primary" name="managebutton">Manage</button> </a> </td>
<td>
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#printPDF{{allocation.id}}">
Print
</button>

<div class="modal fade" id="printPDF{{allocation.id}}" data-bs-keyboard="false" data-bs-backdrop="static" tabindex="-1" aria-labelledby="printPDFLabel" aria-hidden="true">
<div class="modal-dialog">
<div class="modal-content">
<div class="modal-header">
<h1 class="modal-title fs-5" id="staticBackdropLabel">Input Expected Delivery Date</h1>
<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
</div>
<form method="post" action="{% url 'printpdf' allocation.id %}">
<div class="modal-body">

{% csrf_token %}

<input type="date" id="edd" name="edd">

</div>
<div class="modal-footer">
```

```
                                        <input
type="submit" class="btn btn-primary"
name="printaction" value="Summary"></input>

                                        <input
type="submit" class="btn btn-primary"
name="printaction" value="Individual
Breakdowns"></input>



                                        </div>

                                    </form>

                                  </div>

                                </div>

                            </div>



                        </td>



                        <td>

                            <button type="button"
class="btn btn-danger" data-bs-toggle="modal"
data-bs-
target="#deleteAllocation{{allocation.id}}">

                                Delete

                            </button>



                            <div class="modal fade"
id="deleteAllocation{{allocation.id}}" data-bs-
keyboard="false" data-bs-backdrop="static"
tabindex="-1" aria-
labelledby="deleteAllocationLabel" aria-
hidden="true">

                                <div class="modal-
dialog">

                                  <div class="modal-
content">

                                    <div class="modal-
header">

                                      <h1 class="modal-title
fs-5" id="staticBackdropLabel">Delete
Allocation</h1>

                                      <button type="button"
class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>

                                    </div>

                                    <form method="post"
action="{% url 'deleteAllocation' allocation.id
%}">

                                      <div class="modal-
body">


                                        {% csrf_token %}



                                        <p>Are you sure
you want to delete allocation {{ allocation
}}?</p>

                                        <p>You cannot
undo this action</p>



                                      </div>

                                      <div class="modal-
footer">

                                      <input
type="submit" class="btn btn-danger"
name="deleteAllocation"
value="Delete"></input>

                                      </div>

                                    </form>

                                  </div>

                                </div>

                            </div>

                        </td>

                    </tr>
```

```
                {% endfor %}


                </tbody>

            </table>

        {% else %}

            <h2>There are no allocations
yet</h2>

        {% endif %}


    <div class="loader" id="loader" hidden>


    </div>


    <script>


        let managebuttons =
document.getElementsByName('managebutton')
;

        for(let i = 0; i<managebuttons.length;
i++){

managebuttons[i].addEventListener('click',
loadSpinner);

        }


        function loadSpinner(){

document.getElementById('loader').hidden =
false;

        }


    </script>

{% endblock %}
```

OrderAllocation/admin.py

```
from django.contrib import admin

from OrderAllocation.models import Client,
ClientStore

# Register your models here.
admin.site.register(Client)
admin.site.register(ClientStore)
```

OrderAllocation/apps.py

```
from django.apps import AppConfig


class OrderallocationConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'OrderAllocation'
```

OrderAllocation/forms.py

```
from typing import Any, Dict, Mapping,
Optional, Type, Union
from django.core.files.base import File
from django.db.models.base import Model
from django.forms import ModelForm

from django.forms.utils import ErrorList

from OrderAllocation.models import Order,
Allocation, Client

from InventoryManagement.models import
Variant

from django import forms


class AllocationForm (ModelForm):

    class Meta:

        model = Allocation

        fields = ['client', 'product', 'request_amount',
'barupc', 'sku', 'category']


    def __init__(self, *args, **kwargs):

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

        self.fields['client'].queryset =
Client.objects.filter(is_active = True)


class OrderForm (ModelForm):

    def __init__(self, *args, **kwargs):

        targetAllocation =
kwargs.pop('targetAllocation')

        listOfClientStores =
kwargs.pop('listOfClientStores')

        listOfVariants =
kwargs.pop('listOfVariants')

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

        self.fields['allocation'] =
forms.ModelChoiceField(queryset=targetAllocat
ion)

        self.fields['client_store'] =
forms.ModelChoiceField(queryset=listOfClient
Stores)

        self.fields['variant'] =
forms.ModelChoiceField(queryset=listOfVarian
ts)


    class Meta:

        model = Order

        fields = ['allocation', 'client_store', 'variant',
'amount']

        widgets = {

            'amount': forms.TextInput

        }


class OrderUpdateForm (forms.Form):


    order =
forms.ModelChoiceField(queryset=None)

    #allocation =
forms.ModelChoiceField(queryset=None)

    #client_store =
forms.ModelChoiceField(queryset=None)

    variant =
forms.ModelChoiceField(queryset=None)

    amount =
forms.IntegerField(widget=forms.TextInput)


    def __init__(self, listOfOrders, listOfVariants,
*args, **kwargs):

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

        #self.fields['allocation'].queryset =
targetAllocation

        self.fields['order'].queryset = listOfOrders
        #self.fields['client_store'].queryset =
listOfClientStores

        self.fields['variant'].queryset =
listOfVariants

        self.__name__ = 'Update'
```

OrderAllocation/models.py

```
from datetime import date

from django.db import models

from InventoryManagement.models import
Product, Variant

from django.contrib.auth.models import User


# Create your models here.

class Client(models.Model):

    name =
models.CharField(verbose_name='Client Name',
max_length=100)

    is_active =
models.BooleanField(verbose_name='Active
Status', default=True)


    def __str__(self):

        return self.name


class ClientStore(models.Model):

    location =
models.CharField(verbose_name='Store Name',
max_length=100)

    client = models.ForeignKey(Client,
on_delete=models.CASCADE,
verbose_name='Owner')

    ponum =
models.CharField(verbose_name="PO
Number", max_length=10)


    def __str__(self):

        return self.location


class Allocation(models.Model):

    client = models.ForeignKey(Client,
on_delete=models.CASCADE)

    product = models.ForeignKey(Product,
on_delete=models.CASCADE)

    request_amount =
models.IntegerField(verbose_name='Total Order
Units')

    user = models.ForeignKey(User,
on_delete=models.CASCADE,
verbose_name='Modified by', default=1)

    order_date =
models.DateField(verbose_name='Modified At',
default=date.today)

    sku =
models.CharField(verbose_name="SKU",
max_length=10)

    barupc =
models.CharField(verbose_name="Barcode/UP
C", max_length=20)

    category =
models.CharField(verbose_name="Category",
max_length=10)


    def __str__(self):

        showstring = '{} -
{}'.format(self.client.name,
self.product.stock_code)

        return showstring
```

```
class Order(models.Model):

    allocation = models.ForeignKey(Allocation,
on_delete=models.CASCADE)

    client_store =
models.ForeignKey(ClientStore,
on_delete=models.CASCADE, default=1)

    variant = models.ForeignKey(Variant,
on_delete=models.CASCADE)

    amount =
models.IntegerField(verbose_name='Allocation
Amount')


    def __str__(self):

        return str(self.id)
```

OrderAllocation/urls.py

```
from django.urls import path

from . import views


urlpatterns = [

    path('', views.AllocationListView.as_view(),
name="order_allocation_index_page"),

    path('searchallocation',
views.AllocationUpdateListView.as_view(),
name="searchallocation"),

    path('create-allocation',
views.AllocationCreateView.as_view(),
name="order_allocation_create_view"),

    path('manage-allocation/<int:pk>',
views.manage_allocation_test,
name="allocation_page"),

    path('update-allocation/<int:pk>',
views.update_allocation_test,
name="update_allocation"),

    path('showallocation/<int:allocation_id>',
views.AllocationSpecificView.as_view(),
name="show_allocation"),

    path('print-allocation/<int:allocation_id>',
views.printpdf, name="printpdf"),

    path('manage-
allocation/<int:pk>/allocate_by_color',
views.allocatebycolor_test,
name="allocatebycolor"),

    path('update-allocation/<int:pk>/default',
views.updateallocationdefault_test,
name="update_allocation_default"),

    path('delete-allocation/<int:pk>',
views.deleteAllocation,
name="deleteAllocation"),

]
```

OrderAllocation/views.py

```
from django.contrib.staticfiles import finders

from django.http import HttpResponse

from django.shortcuts import render, redirect

from django.urls import reverse

from InventoryManagement.models import
Product, Variant

from OrderAllocation.models import Allocation,
ClientStore, Order

from django.views.generic.edit import
CreateView

from django.views.generic.list import ListView

from django.contrib.auth.mixins import
LoginRequiredMixin

from django.forms import formset_factory
from OrderAllocation.forms import OrderForm,
OrderUpdateForm, AllocationForm

from reportlab.platypus import
SimpleDocTemplate, Image, Paragraph, Spacer,
KeepInFrame

from reportlab.lib.styles import ParagraphStyle

from reportlab.platypus.tables import Table,
TableStyle

from reportlab.lib.units import inch

from reportlab.lib.pagesizes import LEGAL

from reportlab.lib import colors

from django.db.models import Q

from datetime import date

import io, math

from MPAMS import views as mainviews

from multiprocessing import Pool

import time

from django.contrib.auth.decorators import
login_required


# Create your views here.

def index_page(request):

    if request.user.is_authenticated:

        allocation_list = Allocation.objects.all()

        product_notifications =
mainviews.get_product_notifications()

        variant_notifications =
mainviews.get_variation_notifications()

        allocation_notifications =
mainviews.get_allocation_notifications()

        total_notifications =
mainviews.get_notifications_count(product_noti
fications, variant_notifications,
allocation_notifications)

        return render(request,
'allocation/order_allocation_index_page.html',
{'allocation_list': allocation_list,
'productnotifications': product_notifications,
'variantnotifications': variant_notifications,
'allocationnotifications': allocation_notifications,
'notificationscount': total_notifications})

    else:

        return redirect(reverse('index_page'))


class AllocationListView(LoginRequiredMixin,
ListView):

    model = Allocation

    template_name =
'allocation/order_allocation_index_page.html'

    context_object_name = 'allocation_list'


    def get_context_data(self, **kwargs):

        context =
super().get_context_data(**kwargs)

        productnotificationlist =
mainviews.get_product_notifications()

        variantnotificationlist =
mainviews.get_variation_notifications()

        allocationnotificationlist =
mainviews.get_allocation_notifications()

        context['productnotifications'] =
productnotificationlist

        context['variantnotifications'] =
variantnotificationlist

        context['allocationnotifications'] =
allocationnotificationlist

        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)
```

```
        return context


class
AllocationUpdateListView(LoginRequiredMixi
n, ListView):
    model = Allocation
    template_name =
'allocation/order_allocation_index_page.html'
    context_object_name = 'allocation_list'


    def get_queryset(self):
        searchterm =
self.request.GET.get('searchterm')
        if searchterm == '':
            searchterm = 'None'
        allocations =
Allocation.objects.filter(Q(client__name__icont
ains=searchterm) |
Q(product__name__icontains=searchterm) |
Q(product__stock_code__icontains=searchterm)
| Q(user__username__icontains=searchterm))
        return allocations


    def get_context_data(self, **kwargs):
        context =
super().get_context_data(**kwargs)
        productnotificationlist =
mainviews.get_product_notifications()
        variantnotificationlist =
mainviews.get_variation_notifications()
        allocationnotificationlist =
mainviews.get_allocation_notifications()
        context['productnotifications'] =
productnotificationlist
        context['variantnotifications'] =
variantnotificationlist
        context['allocationnotifications'] =
allocationnotificationlist
        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)
        return context


class
AllocationCreateView(LoginRequiredMixin,
CreateView):
    template_name =
'allocation/allocationcreateview.html'
    form_class = AllocationForm
    success_url = '/Order/'


    def form_valid(self, form):
        form.instance.user = self.request.user
        return super().form_valid(form)


    def get_context_data(self, **kwargs):
        context =
super().get_context_data(**kwargs)
        productnotificationlist =
mainviews.get_product_notifications()
        variantnotificationlist =
mainviews.get_variation_notifications()
        allocationnotificationlist =
mainviews.get_allocation_notifications()
        context['productnotifications'] =
productnotificationlist
        context['variantnotifications'] =
variantnotificationlist
```

```
        context['allocationnotifications'] = allocationnotificationlist

        context['notificationscount'] = mainviews.get_notifications_count(productnotificationlist, variantnotificationlist, allocationnotificationlist)

        return context


def get_variant_stats (listofvariants, listofallocations, listofcolors):


    variant_stats = []


    for color in listofcolors:

        new_entry = [color]

new_entry.extend(get_variant_stats_by_color(color, listofvariants, listofallocations))

        variant_stats.append(new_entry)


    return variant_stats


def get_variant_stats_update (listofvariants, listofcolors):

    variant_stats = []


    for color in listofcolors:

        new_entry = [color]

new_entry.extend(get_variant_stats_by_color_update(color, listofvariants))

        variant_stats.append(new_entry)


    return variant_stats


def get_variant_stats_by_color (targetColor, variantsList, allocationsList):


    variant_stats_color = []

    variants = [variant for variant in variantsList if variant.color.lower() == targetColor]

    for entry in variants:

        variantAllocations = [allocation for allocation in allocationsList if entry.id == allocation[0]]

        if len(variantAllocations) != 0:


            remainingTotal = entry.stock


            for allocation in variantAllocations:

                remainingTotal -= allocation[1]


            listentry = {'displayVariant': entry, 'remainingTotal': remainingTotal}

            variant_stats_color.append(listentry)


    return variant_stats_color


def get_variant_stats_by_color_update (targetColor, variantsList):


    variant_stats_color = []

    variants = [variant for variant in variantsList if variant.color.lower() == targetColor]


    for entry in variants:
        listentry = {'displayVariant': entry,
'remainingTotal': entry.stock}
        variant_stats_color.append(listentry)


    return variant_stats_color


def getSizes(listOfAllocations,
listOfDisplayColors):


    store_order = listOfAllocations[0]


    store_sizes = [entry['variant'].size for entry in
store_order]
    color = 'testcolor'
    for order in store_order:
        if order['variant'].color.title() != color:
            color = order['variant'].color.title()
            numofcolor = len([entry for entry in
store_order if color ==
entry['variant'].color.title()])
            templist = [x for x in range(numofcolor)]
            listOfDisplayColors.append([color for x
in templist])


    return store_sizes


def getUniqueSizes(listOfStats):
    key = 0
    maxSizes = 0
    sizes = []


    for i, entry in enumerate(listOfStats):
        if len(entry) > maxSizes:
            key = i
            maxSizes = len(entry)


    targetEntry = listOfStats[key]
    for i in range(1,len(targetEntry)):
        variant = targetEntry[i]['displayVariant']
        sizes.append(variant.size)


    return sizes


#Returns the total amount of variants in stock
def gettotalvariants(variantslist):
    return sum([variant.stock for variant in
variantslist])


def collectTotalVariants(variantlist, orderlist):
    for variant in variantlist:
        for order in orderlist:
            if variant == order.variant:
                variant.stock += order.amount


def collectTotalVariants_test(orderlist,
listOfVariantsValues):
    for order in orderlist:
        for variant in listOfVariantsValues:
            if variant['id'] == order.variant_id:
                variant['stock'] += order.amount
                break


#Returns the total of the allocated units
```

```
def gettotalallocationunits(allocationunitslist):
  total = 0
  for entry in allocationunitslist:
    total += entry[1]
  return total


def
getTotalAllocationUnitsUpdate(allocationUnits
List):
  total = 0
  for entry in allocationUnitsList:
    total += entry["amount"]
  return total


#Returns a list of the proportions of each variant
wrt the total amount in stock
def getproportions(variantslist,
totalvariantscount):
  listofproportions = []
  for variant in variantslist:
    proportion = variant.stock /
totalvariantscount
    listofproportions.append([variant.id,
proportion])
  return listofproportions


#Returns a list of allocated units for each variant
given its proportion
def getallocationunits(proportions):
  allocationunitslist = []
  for proportion in proportions:
    variation = Variant.objects.get(pk =
proportion[0])
    variationunits = math.floor(variation.stock
* proportion[1])
    allocationunitslist.append([variation.id,
variationunits])
  return allocationunitslist


def getAllocationUnitsUpdate(proportions,
variants, allocation):
  allocationunitslist = []


  for proportion in proportions:
    targetVariant = [variant for variant in
variants if variant.id == proportion[0]]
    calculatedvariationunits =
math.floor(allocation.request_amount *
proportion[1])
    if calculatedvariationunits >
targetVariant[0].stock:
      variationunits = targetVariant[0].stock
    else:
      variationunits = calculatedvariationunits

allocationunitslist.append([targetVariant[0].id,
variationunits])
  return allocationunitslist


def getAllocationUnitsUpdate_test(proportions,
variants, allocation):


  allocationunitslist = []


  for proportion in proportions:
    targetVariant = [variant for variant in
variants if variant['id'] == proportion[0]]
```

```
        calculatedvariationunits = math.floor(allocation.request_amount * proportion[1])

        if calculatedvariationunits > targetVariant[0]['stock']:

            variationunits = targetVariant[0]['stock']

        else:

            variationunits = calculatedvariationunits


        allocationunitslist.append([targetVariant[0]['id'], variationunits])

    return allocationunitslist




#Finds the entry in the provided list that has the most stock available

def findMostAvailableStock(listofallocatedunits):

    key = 0

    variant = Variant.objects.get(pk = listofallocatedunits[0][0])

    keyproportion = listofallocatedunits[0][1] / variant.stock


    for i in range(1, len(listofallocatedunits)):

        variant = Variant.objects.get(pk = listofallocatedunits[i][0])

        proportion = listofallocatedunits[i][1] / variant.stock


        if proportion < keyproportion:

            key = i

            keyproportion = proportion

    return key


def findMostAvailableStockUpdate(listofallocatedunits, variantslist):

    key = 0

    variant = getMatchVariant(variantslist, listofallocatedunits[0][0])

    keyproportion = listofallocatedunits[0][1] / variant.stock

    for i in range(1, len(listofallocatedunits)):

        searchkey = listofallocatedunits[i][0]

        variant = getMatchVariant(variantslist, searchkey)

        proportion = listofallocatedunits[i][1] / variant.stock


        if proportion < keyproportion:

            key = i

            keyproportion = proportion


    return key


def findMostAvailableStockCustom(listofallocatedunits, variantslist, filteredvariantslist):

    key = 0

    variant = getMatchVariant(variantslist, listofallocatedunits[0][0])

    keyproportion = listofallocatedunits[0][1] / variant.stock

    for i in range(1, len(listofallocatedunits)):

        searchkey = listofallocatedunits[i][0]
        variant = getMatchVariant(variantslist, searchkey)

        proportion = listofallocatedunits[i][1] / variant.stock


        if proportion < keyproportion:

            if variant in filteredvariantslist:

                key = i

                keyproportion = proportion


    return key


def getMatchVariant(listofvariants, key):

    matchvariant = 0

    for variant in listofvariants:

        if variant.id == key:

            matchvariant = variant

            break

    return matchvariant


#Finds the entry in the provided list that has the most stock allocated

def findMostAllocatedStock(listofallocatedunits):

    key = 0

    variant = Variant.objects.get(pk = listofallocatedunits[0][0])

    keyproportion = listofallocatedunits[0][1] / variant.stock


    for i in range(1, len(listofallocatedunits)):

        variant = Variant.objects.get(pk = listofallocatedunits[i][0])

        proportion = listofallocatedunits[i][1] / variant.stock


        if proportion > keyproportion:

            key = i

            keyproportion = proportion


    return key


def findMostAllocatedStockUpdate(listofallocatedunits, variantslist):

    key = 0

    variant = getMatchVariant(variantslist, listofallocatedunits[0][0])

    keyproportion = listofallocatedunits[0][1] / variant.stock

    for i in range(1, len(listofallocatedunits)):

        searchkey = listofallocatedunits[i][0]

        variant = getMatchVariant(variantslist, searchkey)

        proportion = listofallocatedunits[i][1] / variant.stock


        if proportion > keyproportion:

            key = i

            keyproportion = proportion


    return key
```

```
def findMostAllocatedStockCustom(listofallocatedunits, variantslist, filteredvariantslist):
    key = 0
    variant = getMatchVariant(variantslist, listofallocatedunits[0][0])
    keyproportion = listofallocatedunits[0][1] / variant.stock
    for i in range(1, len(listofallocatedunits)):
        searchkey = listofallocatedunits[i][0]
        variant = getMatchVariant(variantslist, searchkey)
        proportion = listofallocatedunits[i][1] / variant.stock

        if proportion > keyproportion:
            if variant in filteredvariantslist:
                key = i
                keyproportion = proportion

    return key

#Divides the allocations among stores
def divideAllocations(allocationlist, allocation):

    dividedAllocations = []
    allocationlistcopy = allocationlist.copy()
    stores = ClientStore.objects.filter(client_id = allocation.client_id)
    numofstores = len(stores)
    for allocation in allocationlistcopy:
        stock = allocation[1]
        equaldivisionofstock = stock // numofstores
        listofmadeallocations = []
        for store in stores:
            if stock >= equaldivisionofstock:
                orderentry = [store, allocation[0], equaldivisionofstock]
                stock -= equaldivisionofstock
                listofmadeallocations.append(orderentry)
            else:
                orderentry = [store, allocation[0], stock]
                stock -= stock
                listofmadeallocations.append(orderentry)
        counter = 0
        while stock != 0:
            pickIndex = counter % numofstores
            listofmadeallocations[pickIndex][2] += 1
            counter += 1
            stock -= 1
        for allocation in listofmadeallocations:
            dividedAllocations.append(allocation)
    return dividedAllocations



#Converts list of lists to list of dicts
def listToDict(allocation, listOfProportions):
    newList = []
    client_stores = {entry[0] for entry in listOfProportions}
    for store in client_stores:
        store_orders = []

        orders = [entry for entry in
listOfProportions if store == entry[0]]

        for order in orders:

            dictentry = {'allocation': allocation,
'client_store': order[0], 'variant':
Variant.objects.get(pk=order[1]), 'amount':
order[2]}

            store_orders.append(dictentry)

        newList.append(store_orders)


    return newList


def listToDictUpdate(orderlist, clientStoreList):

    newList= []


    for store in clientStoreList:

        store_entries = []

        orders = [order for order in orderlist if
order.client_store == store]

        for order in orders:

            dictentry = {'order': order, 'variant':
order.variant, 'amount': order.amount}

            store_entries.append(dictentry)

        newList.append(store_entries)


    return newList


def listToDictUpdate_test(orderlist,
clientStoreList):

    newList= []

    for store in clientStoreList:

        store_entries = []

        orders = [order for order in orderlist if
order.client_store == store]

        for order in orders:

            dictentry = {'order': order, 'variant':
order.variant, 'amount': order.amount}

            store_entries.append([store, dictentry])

        newList.append(store_entries)


    return newList


def listToDictUpdateColor(allocation,
listOfProportions, ordersList):

    listOfRevisedOrders = []

    client_stores = {entry[0] for entry in
listOfProportions}

    for store in client_stores:

        store_orders = []

        orders = [entry for entry in
listOfProportions if store == entry[0]]

        for order in orders:

            client_store = order[0]

            variant =
Variant.objects.get(pk=order[1])

            targetOrder = [tOrder for tOrder in
ordersList if tOrder.allocation == allocation and
tOrder.client_store == client_store and
tOrder.variant == variant]

            dictentry = {'order': targetOrder[0],
'variant': variant, 'amount': order[2]}

            store_orders.append(dictentry)

        listOfRevisedOrders.append(store_orders)
    return listOfRevisedOrders


def getComputedTotal(calculatedAllocationsList):
    overallTotal = 0
    for entry in calculatedAllocationsList:
        formsetAmounts = [item['amount'] for item in entry[1]]
        overallTotal += sum(formsetAmounts)

    return overallTotal


def listToDictDeleteOrders(orderlist):
    newList= []
    for order in orderlist:
        allocation = Allocation.objects.get(pk=order["allocation_id"])
        client_store = ClientStore.objects.get(pk=order["client_store_id"])
        variant = Variant.objects.get(pk=order["variant_id"])
        dictentry = {'order': order["id"], 'allocation': allocation, 'client_store': client_store, 'variant': variant, 'amount': order["amount"]}
        newList.append(dictentry)
    return newList


class OrderListView(LoginRequiredMixin, ListView):
    model = Order
    template_name = 'allocation/allocationpdfview.html'
    context_object_name = 'entries'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        # Can ignore Class Product error, still works in implementation
        context['allocation'] = Allocation.objects.get(pk=self.kwargs['allocation_id'])
        productnotificationlist = mainviews.get_product_notifications()
        variantnotificationlist = mainviews.get_variation_notifications()
        allocationnotificationlist = mainviews.get_allocation_notifications()
        context['productnotifications'] = productnotificationlist
        context['variantnotifications'] = variantnotificationlist
        context['allocationnotifications'] = allocationnotificationlist
        context['allocationscount'] = mainviews.get_notifications_count(productnotificationlist, variantnotificationlist, allocationnotificationlist)
        return context

    def get_queryset(self):
        return Order.objects.filter(allocation_id=self.kwargs['allocation_id'])


def printpdf(request, allocation_id):
    if request.method == 'POST':

        edd = request.POST['edd']
        action = request.POST['printaction']

        pdf_buffer = io.BytesIO()

        allocation = Allocation.objects.get(id=allocation_id)
        product = Product.objects.get(pk = allocation.product_id)
        client_stores = ClientStore.objects.filter(client_id = allocation.client_id)
        orders = Order.objects.filter(allocation_id=allocation_id)

        elements = []

        if action == "Individual Breakdowns":

            doc = SimpleDocTemplate(pdf_buffer, pagesize=(LEGAL[1], LEGAL[0]), topMargin=0, bottomMargin=0, leftMargin=0, rightMargin=0)

            for store in client_stores:
                infoTable = allocationInfoTable(allocation, product, store, edd)
                breakdownTable = allocationBreakdownTable(product, store, orders)
                spacer = Spacer(20, 20)
                elements.append(infoTable)
                content = [breakdownTable,]
                breakdownTableToAdd = KeepInFrame(0,0, content, mode='shrink', fakeWidth=False)

elements.append(breakdownTableToAdd)
                elements.append(spacer)
            doc.build(elements)

            pdf_value = pdf_buffer.getvalue()
            pdf_buffer.close()

            response = HttpResponse(content_type='application/pdf')
            response_content = 'attachment; filename="{} Individual Breakdowns.pdf"'.format(allocation)
            response['Content-Disposition'] = response_content

            response.write(pdf_value)
            return response

        elif action == "Summary":

            doc = SimpleDocTemplate(pdf_buffer, pagesize=(LEGAL[1], LEGAL[0]), topMargin=0, bottomMargin=0, leftMargin=0, rightMargin=0)
            summaryTable = makeSummary(allocation, product, client_stores, orders, edd)
            content = [summaryTable,]
            tableToAdd = KeepInFrame(0,0, content, mode='shrink', fakeWidth=False)

            elements.append(tableToAdd)

            doc.build(elements)


            pdf_value = pdf_buffer.getvalue()

            pdf_buffer.close()


            response = HttpResponse(content_type='application/pdf')

            response_content = 'attachment; filename="{}: Summary.pdf"'.format(allocation)

            response['Content-Disposition'] = response_content


            response.write(pdf_value)

            return response


def allocationInfoTable(allocation, product, clientStore, expected_delivery_date):

    text_style = ParagraphStyle(name='text_style', alignment=1, textColor=colors.blue)

    paragraph_style = ParagraphStyle(name='paragraph_style',alignment=1)

    date_text = Paragraph('''<strong>{}</strong>'''.format(str(expected_delivery_date)), style=text_style)

    barcodetext = 'Barcode/UPC: {}'.format(allocation.barupc)

    categorytext = Paragraph('''<strong>{}</strong>'''.format('CATEGORY'), style=text_style)

    titleBar = [categorytext ,barcodetext, '', date_text]

    style = "STYLE: {}".format(product.name)

    imgpath = finders.find('images/SideUp.jpg')

    image = Image(imgpath)

    location = clientStore.location

    orders = Order.objects.filter(allocation_id = allocation.id)

    totalAllocations = 0

    for order in orders:

        if order.client_store_id == clientStore.id:

            totalAllocations += order.amount

    sku_code = allocation.sku

    actcatstyle = ParagraphStyle(name='actcatstyle', alignment=1, textColor=colors.blue, fontSize=18)

    actualcategorytext = Paragraph('''<strong>{}</strong>'''.format(allocation.category), style=actcatstyle)

    style_bar = [actualcategorytext ,style, '', image]

    location_text = Paragraph('''

        <strong>{}</strong>

    '''.format(str(location)), style=paragraph_style)

    total_allocation_style = ParagraphStyle(name='total_allocation_style', alignment=1, fontSize=14)

    ponum_style = ParagraphStyle(name='ponum_style', fontSize=14)

    ponum_text = Paragraph('''<strong>{}</strong>'''.format(clientStore.ponum), style=ponum_style)

    location_bar = ['',ponum_text, location_text, '']

    totalAllocationsText = Paragraph('''
        <strong>{}</strong>

    '''.format(str(totalAllocations)),
style=total_allocation_style)

    totalAllocationsText

    some_bar = ['' , allocation.client,
totalAllocationsText, '']

    sku_text = 'SKU: {}'.format(sku_code)

    sku_bar = ['', sku_text, '', '']

    table_values = [titleBar, style_bar,
location_bar, some_bar, sku_bar]

    table = Table(table_values,
colWidths=[2*inch, 2*inch, 3*inch, 2.5*inch])

    table_style = TableStyle([

        ('ALIGN', (1,0), (-1,-1), 'LEFT'),
('VALIGN', (0,0), (-1,-1), 'MIDDLE'), ('GRID',
(0,0), (-1,-1), 1, colors.black), ('SPAN', (0,1),
(0,-1)), ('SPAN', (1,0), (2,0)), ('SPAN', (1,1),
(2,1)), ('SPAN', (2, 3), (2,-1)), ('SPAN', (0,1),
(0,-1)), ('SPAN', (-1, 1), (-1, -1)), ('ALIGN',
(0,0), (0,-1), 'CENTER'),

        ])

    table.setStyle(table_style)

    return table



def allocationBreakdownTable(product,
clientStore, listOfOrders):

    text_style = ParagraphStyle(name='text_style',
textColor=colors.blue)

    style2 = ParagraphStyle(name='style2',
alignment=1)

    style = "STYLE: {}".format(product.name)

    style_text = Paragraph(style, style=text_style)

    master_code_text =
Paragraph(product.stock_code, style=text_style)

    listofInches = [master_code_text, '']

    listofColors = list({order.variant.color for
order in listOfOrders})

    totalAllocationEntry = sum([order.amount for
order in listOfOrders if order.client_store_id ==
clientStore.id])

    listofInchesBack = list({order.variant.inches
for order in listOfOrders})

    listofInchesBack.sort()

    listofInchesDisplay = ['{}"'.format(inch) for
inch in listofInchesBack]

    listofInches.extend(listofInchesDisplay)

    totalAllocationEntryText =
Paragraph('''<strong>{}</strong>'''.format(str(tot
alAllocationEntry)), style=style2)

    clientStore_text =
Paragraph('''<strong>{}</strong>'''.format(client
Store.location))

    listofTotals = [clientStore_text,
totalAllocationEntryText]

    for i in range(2,len(listofInches)):

        curSize = listofInches[i]

        sizeTotal = 0

        for order in listOfOrders:

            curVariant = Variant.objects.get(pk =
order.variant_id)

            check =
'{}"'.format(str(curVariant.inches))

            if curSize == check and
order.client_store_id == clientStore.id:

                sizeTotal += order.amount

        listofTotals.append(sizeTotal)



    table_values = [[style_text], listofInches,
listofTotals]
```

```
    for color in listofColors:
        listOfAllocationAmounts = [color.upper()]
        for i in range(1, len(listofInches)):
            listOfAllocationAmounts.append(0)
        for order in listOfOrders:
            curVariant = Variant.objects.get(pk =
order.variant_id)
            if curVariant.color == color:
                for i in range(2, len(listofInches)):
                    check =
'{}"'.format(str(curVariant.inches))
                    if check == listofInches[i] and
order.client_store_id == clientStore.id:
                        listOfAllocationAmounts[i] +=
order.amount
            else:
                continue


        totalAllocationAmount = 0
        for x in range(2,
len(listOfAllocationAmounts)):
            totalAllocationAmount +=
listOfAllocationAmounts[x]
        listOfAllocationAmounts[1] =
totalAllocationAmount



table_values.append(listOfAllocationAmounts)


    table = Table(table_values,
colWidths=1.5*inch)
    table_style = TableStyle([
        ('ALIGN', (0,0), (-1,-1), 'CENTER'),
('VALIGN', (0,0), (-1,-1), 'MIDDLE'), ('GRID',
(0,0), (-1,-1), 1, colors.black), ('SPAN', (0,0), (-
1,0)), ('BACKGROUND', (1,3), (1,-1),
colors.aliceblue), ('BACKGROUND', (2,1), (-
1,1), colors.HexColor('#f5a989')),
('BACKGROUND', (2,2), (-1,2),
colors.HexColor("#fadda7")), ('SPAN', (0,1),
(1,1)), ('ALIGN', (0,2), (0,-1), 'RIGHT'),
#('LEFTPADDING', (0,0), (-1,-1), 0),
('RIGHTPADDING', (0,0), (-1,-1), 0)
        ])
    table.setStyle(table_style)
    return table


def makeSummary(chosenAllocation,
targetProduct, listOfClientStores, listOfOrders,
expectedDeliveryDate):


    barcodeText =
'Barcode/UPC:{}'.format(chosenAllocation.baru
pc)
    eddStyle = ParagraphStyle(name='eddStyle',
alignment=1, textColor=colors.red)
    eddText =
Paragraph('''<strong>{}</strong>'''.format(str(ex
pectedDeliveryDate)), style=eddStyle)
    topBar = [barcodeText,
chosenAllocation.client, '', '', eddText]


    colorsBar = [chosenAllocation.sku, '', '','',]
    listOfColorTotals =
getColorTotals(listOfOrders)
    listOfInches = getListOfInches(listOfOrders)
    for colorTotal in listOfColorTotals:
        colorText =
Paragraph('''<strong>{}</strong>'''.format(color
Total[1]))
        colorsBar.append(colorTotal[0])
        colorsBar.append(colorText)

        diff = abs(len(listOfInches) -
len(colorTotal))

        for _ in range(diff):

            colorsBar.append('')


    styleText = targetProduct.name

    styleBar = [styleText, '']


    categoryText = 'CATEGORY
{}'.format(chosenAllocation.category)

    categoryBar = [categoryText, '', 'INCHES', '']

    listOfInchesDisplay = []

    inchesDisplayStyle =
ParagraphStyle(name='inchesDisplayStyle',
textColor=colors.blue, alignment=1)

    for inch in listOfInches:

        inchText =
Paragraph('''<strong>{}"</strong>'''.format(inch
), style=inchesDisplayStyle)

        listOfInchesDisplay.append(inchText)

    for _ in range(len(listOfColorTotals)):

        categoryBar.extend(listOfInchesDisplay)


    totalAllocated = 0

    for order in listOfOrders:

        totalAllocated += order.amount

    listOfColors = [entry[1] for entry in
listOfColorTotals]

    listOfAllocatedTotals =
getListOfAllocatedTotals(listOfColors,
listOfOrders, listOfInches)

    totalAllocatedStyle =
ParagraphStyle(name='totalAllocatedStyle',
alignment=1)

    totalAllocText =
Paragraph('{}'.format(totalAllocated),
style=totalAllocatedStyle)

    variantTotalsBar = ['', '', totalAllocText, '']

    for entry in listOfAllocatedTotals:

        variantTotalsBar.extend(entry)


    status = getstatus(totalAllocated,
chosenAllocation.request_amount)

    statusStyle =
ParagraphStyle(name='statusStyle',
textColor=colors.red, alignment=1)

    statusText =
Paragraph('''<strong>{}</strong>'''.format(status
), style=statusStyle)

    colorsBar[1] = statusText


    mastercodeText =
Paragraph('''<strong>{}</strong>'''.format(target
Product.stock_code))

    totalAllocTextBold =
Paragraph('''<strong>{}</strong>'''.format(str(tot
alAllocated)), style=totalAllocatedStyle)

    ponumBar = ['PO#', mastercodeText,
totalAllocTextBold, '']


    table_values = [topBar, colorsBar, styleBar,
categoryBar, variantTotalsBar, ponumBar]


    listOfLocationAllocations =
getListOfLocationAllocations(listOfClientStores
, listOfOrders, listOfColors, listOfInches)
```

```
    for locationAllocation in
listOfLocationAllocations:

        table_values.append(locationAllocation)



    table = Table(table_values)

    table_style = TableStyle([

        ('GRID', (0,0), (-1,-1), 1, colors.black),
('SPAN', (1,0), (2,0)), ('SPAN', (4,0), (-1,0)),
('ALIGN', (2,6), (2,-1), 'CENTER'), ('ALIGN',
(4,3), (-1,-1), 'CENTER'), ('BACKGROUND',
(2,5), (2,-1), colors.aliceblue),
('BACKGROUND', (1,5), (1,5),
colors.HexColor("#fadda7")), ('SPAN', (0,3),
(1,3)), ('SPAN', (2,1), (3,2)),
#('LEFTPADDING', (0,0), (-1,-1), 0),
('RIGHTPADDING', (0,0), (-1,-1), 0)

    ])

    table.setStyle(table_style)

    return table



def getColorTotals(ordersList):

    listOfColorTotals = []

    totalStyle = ParagraphStyle(name='totalStyle',
alignment=1)

    listOfUniqueVariants = list({order.variant for
order in ordersList})

    setOfColors = {variant.color.upper() for
variant in listOfUniqueVariants}

    for color in setOfColors:

        totalCount = 0

        for order in ordersList:

            if order.variant.color.upper() == color:

                totalCount += order.amount

        totalText =
Paragraph('{}'.format(str(totalCount)),
style=totalStyle)

        listOfColorTotals.append([totalText,
color])



    return listOfColorTotals



def getListOfInches(ordersList):

    listOfInches = list({order.variant.inches for
order in ordersList})

    listOfInches.sort()

    return listOfInches



def getListOfAllocatedTotals(colorsList,
ordersList, inchesList):

    listOfAllocatedTotals = []

    for color in colorsList:

        listOfColorAllocated = []

        for inch in inchesList:

            listOfAllocations = [order.amount for
order in ordersList if order.variant.color.upper()
== color and order.variant.inches == inch]

            if listOfAllocations:


listOfColorAllocated.append(sum(listOfAllocati
ons))

            else:

                listOfColorAllocated.append(0)


listOfAllocatedTotals.append(listOfColorAllocat
ed)

    return listOfAllocatedTotals
```

```
def
getListOfLocationAllocations(clientStoresList,
ordersList, colorsList, inchesList):


    listOfLocationAllocations = []


    for clientstore in clientStoresList:
        total = 0
        locationAllocation = [clientstore.ponum,
clientstore.location, total, '']
        for color in colorsList:
            filteredOrders = [order for order in
ordersList if order.client_store == clientstore
and order.variant.color.upper() == color]
            for inch in inchesList:
                allocMatch = [order for order in
filteredOrders if order.variant.inches == inch]
                if allocMatch:

locationAllocation.append(allocMatch[0].amoun
t)
                    total += allocMatch[0].amount
                else:
                    locationAllocation.append(0)
        locationAllocation[2] = total

listOfLocationAllocations.append(locationAlloc
ation)


    return listOfLocationAllocations
```

```
class
AllocationSpecificView(LoginRequiredMixin,
ListView):
    model = Allocation
    template_name =
'allocation/order_allocation_index_page.html'
    allocation_list = 'allocation_list'


    def get_context_data(self, **kwargs):
        context =
super().get_context_data(**kwargs)
        productnotificationlist =
mainviews.get_product_notifications()
        variantnotificationlist =
mainviews.get_variation_notifications()
        allocationnotificationlist =
mainviews.get_allocation_notifications()
        context['productnotifications'] =
productnotificationlist
        context['variantnotifications'] =
variantnotificationlist
        context['allocationnotifications'] =
allocationnotificationlist
        context['notificationscount'] =
mainviews.get_notifications_count(productnotifi
cationlist, variantnotificationlist,
allocationnotificationlist)
        return context


    def get_queryset(self):
        allocation =
Allocation.objects.filter(pk=self.kwargs['allocati
on_id'])
        return allocation
```

```
def getuniquecolors(listofvariants):
    return {variant.color.lower() for variant in
listofvariants}


def getProportionsFiltered(variantslist,
variantstotal, filteredvariantslist):
    listofproportions = []
    for variant in variantslist:
        if variant in filteredvariantslist:
            print(variant.stock)
            proportion = variant.stock / variantstotal
        else:
            proportion = 0
        listofproportions.append([variant.id,
proportion])
    return listofproportions


def getProportionsFiltered_test(variantslist,
variantstotal, filteredvariantslist):
    listofproportions = []
    for variant in variantslist:
        matches = [x for x in filteredvariantslist if
variant['id'] == x.id]
        if matches:
            proportion = variant['stock'] /
variantstotal
        else:
            proportion = 0
        listofproportions.append([variant['id'],
proportion])
    return listofproportions

def getVariantUnits(listofvariants,
listoffilteredvariants):
    listofVariantUnits = []
    for variant in listofvariants:
        amount = 0
        if variant in listoffilteredvariants:
            amount = variant.stock
        new_entry = [variant.id, amount]
        listofVariantUnits.append(new_entry)
    return listofVariantUnits


def filter_variants_by_colors(listofvariants,
listofdesiredcolors):


    return [x for x in listofvariants if
x.color.lower() in listofdesiredcolors]


def
sortAllocations(listofCalculatedAllocationsDict,
listofOrders, listofUpdatedOrders,
listofCreatedOrders, listofDeletedOrders):
    copyOfListOfOrders = listofOrders.values()
    listofexistingOrders = []
    for entry in listofCalculatedAllocationsDict:
        entryVariant = entry["variant"]
        entryStore = entry["client_store"]
        foundRecord = False


        for order in copyOfListOfOrders:
            orderVariant =
Variant.objects.get(pk=order.variant)
            orderStore =
ClientStore.objects.get(pk=order.client_store)
```

```
            if entryVariant == orderVariant and
entryStore == orderStore:


                #TODO DO CODE TO REPLACE
PRE-EXISTING AMOUNT BY
CALCULATED AMOUNT (SOLVED)

                entryAllocation =
Allocation.objects.get(pk=order["allocation_id"]
)

                newEntry = {'order': order['id'],
'allocation': entryAllocation, 'client_store':
entryStore, 'variant': entryVariant, 'amount':
entry["amount"]}


listofUpdatedOrders.append(newEntry)

                listofexistingOrders.append(order)

                foundRecord = True

                continue


        if not foundRecord:

            listofCreatedOrders.append(entry)


    for order in copyOfListOfOrders:

        if order not in listofexistingOrders:

            listofDeletedOrders.append(order)


def getstatus(total_allocated_units,
request_amount):

    status = ''

    ratio = total_allocated_units / request_amount


    if ratio == 1:

        status = 'ok'
```

```
    elif ratio >= 0.5:

        status = 'slightlyok'

    else:

        status = 'notallowed'


    return status


def divideSpecificAllocations(allocationList,
storeSpecificAmounts):

    dividedAllocations = []

    allocationListCopy = allocationList.copy()

    storeSpecificAmountsTotal = sum([m[1] for
m in storeSpecificAmounts])

    currentTotals = []

    listOfStoreSpecificProportions = []

    for item in storeSpecificAmounts:

        store_proportion =
item[1]/storeSpecificAmountsTotal


        new_entry = (item[0], store_proportion)


listOfStoreSpecificProportions.append(new_entr
y)

        currentTotals.append([0, item[1]])


    for alloc in allocationListCopy:

        stock = alloc[1]

        remainingStock = alloc[1]

        listOfMadeAllocations = []

        for i in
range(len(listOfStoreSpecificProportions)):
```

```
            amount = math.floor(stock *
listOfStoreSpecificProportions[i][1])
            remainingStock -= amount
            orderentry =
[listOfStoreSpecificProportions[i][0], alloc[0],
amount]
            currentTotals[i][0] += amount

listOfMadeAllocations.append(orderentry)


        print(str(alloc[0]) + " remaining: " +
str(remainingStock))


        while remainingStock != 0:
            index =
findMostAvailable(currentTotals)
            currentTotals[index][0] += 1
            listOfMadeAllocations[index][2] += 1
            remainingStock -= 1



dividedAllocations.extend(listOfMadeAllocation
s)


    return dividedAllocations


def findMostAvailable(listOfCurrentTotals):
    key = 0
    proportion = listOfCurrentTotals[0][0] /
listOfCurrentTotals[0][1]


    for i in range(len(listOfCurrentTotals)):
        currProportion = listOfCurrentTotals[i][0] /
listOfCurrentTotals[i][1]
        if currProportion < proportion:
            key = i
            proportion = currProportion


    return key


@login_required
def deleteAllocation(request, pk):


    if request.method == 'POST':


        allocation = Allocation.objects.get(id = pk)
        orders = Order.objects.filter(allocation_id =
pk)


        if len(orders) > 0:
            for order in orders:
                variant = Variant.objects.get(id =
order.variant_id)
                variant.stock += order.amount
                variant.save()


            orders.delete()
        allocation.delete()


        return
redirect('order_allocation_index_page')


@login_required
def update_allocation_test(request,pk):
    if request.method == 'GET':
        start = time.perf_counter()
        allocation = Allocation.objects.get(id = pk)
        orders = Order.objects.filter(allocation_id =
pk)
        stores =
ClientStore.objects.filter(client=allocation.client
)
        variants = Variant.objects.filter(product_id
=
allocation.product_id).defer('picture','threshold')
        selectcolors = getuniquecolors(variants)
        listofdictorders = listToDictUpdate(orders,
stores)
        totalallocations = sum([order.amount for
order in orders])
        displayColors = []
        sizes = getSizes(listofdictorders,
displayColors)
        variant_stats_before = time.perf_counter()
        variant_stats = get_variant_stats_update
(variants, selectcolors)
        variant_stats_after = time.perf_counter()
        print("Variant Stats Time:
{}".format(variant_stats_after-
variant_stats_before))
        uniqueSizes =
getUniqueSizes(variant_stats)
        product_notifications =
mainviews.get_product_notifications()
        variant_notifications =
mainviews.get_variation_notifications()
        allocation_notifications =
mainviews.get_allocation_notifications()
        finaldictlist = []
        for entry in listofdictorders:
            store = entry[0]['order'].client_store
            finaldictlist.append([store, entry])
        total_notifications =
mainviews.get_notifications_count(product_noti
fications, variant_notifications,
allocation_notifications)
        status = getstatus(totalallocations,
allocation.request_amount)
        end = time.perf_counter()
        print("Total Processing Time: " + str(end-
start))
        return
render(request,'allocation/allocationupdate_test.
html', {'form_data': finaldictlist,
'productnotifications': product_notifications,
'variantnotifications': variant_notifications,
'allocationnotifications': allocation_notifications,
'notificationscount': total_notifications,
'allocation': allocation, 'total': totalallocations,
"variantStats": variant_stats, 'primarykey': pk,
'status': status, 'sizes': sizes, 'uniqueSizes':
uniqueSizes, 'displayColors': displayColors,
'stores': stores, 'colors': selectcolors})
    else:
        allocation = Allocation.objects.get(pk = pk)
        orders = Order.objects.filter(allocation_id =
pk)
        variants = Variant.objects.filter(product_id
= allocation.product_id)
        for order in orders:
            targetString = '{}-{}-
formValue'.format(order.id, order.client_store)
            form_order_amount =
request.POST.get(targetString)
            affectVariant = Variant.objects.get(pk =
order.variant_id)
```

```
            difference = int(form_order_amount) -
order.amount

            affectVariant.stock -= difference

            order.amount = int(form_order_amount)

            affectVariant.save()

            order.save()


        allocation.user = request.user

        allocation.order_date = date.today()

        allocation.save()

        return
redirect(reverse('order_allocation_index_page'))


@login_required

def updateallocationdefault_test(request, pk):

    allocation = Allocation.objects.get(pk=pk)

    orders = Order.objects.filter(allocation_id =
allocation.id).prefetch_related()

    stores = ClientStore.objects.filter(client =
allocation.client)

    variants = Variant.objects.filter(product_id =
allocation.product_id).prefetch_related()

    variant_colors = getuniquecolors(variants)

    collectTotalVariants(variants, orders)

    totalvariantunits = gettotalvariants(variants)


    if allocation.request_amount >
totalvariantunits:

        variant_units = [[variant.id, variant.stock]
for variant in variants]

        divideallocationunits =
divideAllocations(variant_units, allocation)

        computatedtotal = sum(var[1] for var in
variant_units)

        updateOrders =
listToDictUpdateColor(allocation, variant_units,
orders)

        finaldictlist = []

        for entry in updateOrders:

            store = entry[0]['order'].client_store

            finaldictlist.append([store, entry])

        product_notifications =
mainviews.get_product_notifications()

        variant_notifications =
mainviews.get_variation_notifications()

        allocation_notifications =
mainviews.get_allocation_notifications()

        total_notifications =
mainviews.get_notifications_count(product_noti
fications, variant_notifications,
allocation_notifications)

        variant_stats = get_variant_stats(variants,
variant_units, variant_colors)

        status = getstatus(computatedtotal,
allocation.request_amount)

        displayColors = []

        sizes = getSizes(updateOrders,
displayColors)

        uniqueSizes =
getUniqueSizes(variant_stats)

        return render(request,
'allocation/allocationupdate.html', {'form_data':
finaldictlist, 'total': computatedtotal,
'productnotifications': product_notifications,
'variantnotifications': variant_notifications,
'allocationnotifications': allocation_notifications,
'notificationscount': total_notifications,
'primarykey': pk, 'colors': variant_colors,
'variantStats': variant_stats, 'allocation':
allocation, 'status': status, 'sizes': sizes,
'uniqueSizes': uniqueSizes, 'displayColors':
displayColors, 'stores': stores})


    proportionlist = getproportions(variants,
totalvariantunits)

    allocationunits =
getAllocationUnitsUpdate(proportionlist,
variants, allocation)

    totalallocationunits =
gettotalallocationunits(allocationunits)


    while totalallocationunits !=
allocation.request_amount:

        if totalallocationunits >
allocation.request_amount:

            chosenIndex =
findMostAllocatedStockUpdate(allocationunits,
variants)

            allocationunits[chosenIndex][1] -= 1

            totalallocationunits -= 1

        else:

            chosenIndex =
findMostAvailableStockUpdate(allocationunits,
variants)

            allocationunits[chosenIndex][1] += 1

            totalallocationunits += 1


    divideallocationunits =
divideAllocations(allocationunits, allocation)

    computatedtotal = sum([var[1] for var in
allocationunits])

    updateOrders =
listToDictUpdateColor(allocation,
divideallocationunits, orders)

    finaldictlist = []

    for entry in updateOrders:

        store = entry[0]['order'].client_store

        finaldictlist.append([store, entry])

    product_notifications =
mainviews.get_product_notifications()

    variant_notifications =
mainviews.get_variation_notifications()

    allocation_notifications =
mainviews.get_allocation_notifications()

    total_notifications =
mainviews.get_notifications_count(product_noti
fications, variant_notifications,
allocation_notifications)

    variant_stats = get_variant_stats(variants,
allocationunits, variant_colors)

    status = getstatus(computatedtotal,
allocation.request_amount)

    displayColors = []

    sizes = getSizes(updateOrders, displayColors)

    uniqueSizes = getUniqueSizes(variant_stats)

    return render(request,
'allocation/allocationupdate_test.html',
{'form_data': finaldictlist, 'total':
computatedtotal, 'productnotifications':
product_notifications, 'variantnotifications':
variant_notifications, 'allocationnotifications':
allocation_notifications, 'notificationscount':
total_notifications, 'primarykey': pk, 'colors':
variant_colors, 'variantStats': variant_stats,
'allocation': allocation, 'status': status, 'sizes':
sizes, 'uniqueSizes': uniqueSizes,
'displayColors': displayColors, 'stores': stores})


@login_required

def allocatebycolor_test(request, pk):


    allocation = Allocation.objects.get(pk=pk)
    orders = Order.objects.filter(allocation_id =
allocation.id).prefetch_related('variant',
'client_store')

    variants = Variant.objects.filter(product_id =
allocation.product_id).defer('picture', 'threshold')

    variants_test =
Variant.objects.filter(product_id =
allocation.product_id).defer('picture', 'threshold',
'inches').values()

    variant_colors = getuniquecolors(variants)

    filtered_variant_colors = []

    clientStores =
ClientStore.objects.filter(client=allocation.client
)

    storeSpecificList = []


    if "toggle" in request.POST:

        for store in clientStores:

            key = store.location + "-number"

            storeSpecificList.append((store,
int(request.POST[key])))


    for color in variant_colors:

        if color in request.POST:

            filtered_variant_colors.append(color)

    if len(filtered_variant_colors) == 0:


filtered_variant_colors.extend(variant_colors)

    filtered_variants =
filter_variants_by_colors(variants,
filtered_variant_colors)


    if len(orders) > 0:

        for variant in variants:

            print("Variant {} before:
{}".format(variant, variant.stock))

        collectTotalVariants(variants, orders)

        for variant in variants:

            print("Variant {} after:
{}".format(variant, variant.stock))

        totalvariantsunits =
gettotalvariants(filtered_variants)


        if allocation.request_amount >
totalvariantsunits:

            variantunits = getVariantUnits(variants,
filtered_variants)

            if len(storeSpecificList)>0:

                dividedallocationunits =
divideSpecificAllocations(variantunits,
storeSpecificList)

            else:

                dividedallocationunits =
divideAllocations(variantunits, allocation)


            computedtotal = sum([variant[1] for
variant in variantunits])

            updateOrders =
listToDictUpdateColor(allocation,
dividedallocationunits, orders)

            finaldictlist = []

            for entry in updateOrders:

                store = entry[0]['order'].client_store

                finaldictlist.append([store, entry])

            product_notifications =
mainviews.get_product_notifications()

            variant_notifications =
mainviews.get_variation_notifications()
```

```
    allocation_notifications = mainviews.get_allocation_notifications()

    total_notifications = mainviews.get_notifications_count(product_notifications, variant_notifications, allocation_notifications)

    variant_stats = get_variant_stats(variants, variantunits, variant_colors)

    status = getstatus(computedtotal, allocation.request_amount)

    displayColors = []

    sizes = getSizes(updateOrders, displayColors)

    uniqueSizes = getUniqueSizes(variant_stats)

    return render(request, 'allocation/allocationupdate_test.html', {'form_data': finaldictlist, 'total': computedtotal, 'productnotifications': product_notifications, 'variantnotifications': variant_notifications, 'allocationnotifications': allocation_notifications, 'notificationscount': total_notifications, 'primarykey': pk, 'colors': variant_colors, 'variantStats': variant_stats, 'allocation': allocation, 'status': status, 'sizes': sizes, 'uniqueSizes': uniqueSizes, 'displayColors': displayColors, 'stores': clientStores})



collectTotalVariants_test(orders, variants_test)

proportionlist = getProportionsFiltered_test(variants_test, totalvariantsunits, filtered_variants)

allocationunits = getAllocationUnitsUpdate_test(proportionlist, variants_test, allocation)

totalallocationunits = gettotalallocationunits(allocationunits)


while totalallocationunits != allocation.request_amount:

    if totalallocationunits > allocation.request_amount:

        chosenIndex = findMostAllocatedStockCustom_test(allocationunits, filtered_variants, variants_test)

        allocationunits[chosenIndex][1] -= 1

        totalallocationunits -= 1

    else:

        chosenIndex = findMostAvailableStockCustom_test(allocationunits, filtered_variants, variants_test)

        allocationunits[chosenIndex][1] += 1

        totalallocationunits += 1


if len(storeSpecificList) > 0:

    dividedallocationunits = divideSpecificAllocations(allocationunits, storeSpecificList)

else:

    dividedallocationunits = divideAllocations(allocationunits, allocation)


updateOrders = listToDictUpdateColor(allocation, dividedallocationunits, orders)

computedtotal = sum([allocationunit[1] for allocationunit in allocationunits])

finaldictlist = []

for entry in updateOrders:

    store = entry[0]['order'].client_store
```

```
            finaldictlist.append([store, entry])

        product_notifications = mainviews.get_product_notifications()

        variant_notifications = mainviews.get_variation_notifications()

        allocation_notifications = mainviews.get_allocation_notifications()

        total_notifications = mainviews.get_notifications_count(product_notifications, variant_notifications, allocation_notifications)

        variant_stats = get_variant_stats(variants, allocationunits, variant_colors)

        status = getstatus(computedtotal, allocation.request_amount)

        displayColors = []

        sizes = getSizes(updateOrders, displayColors)

        uniqueSizes = getUniqueSizes(variant_stats)

        return render(request, 'allocation/allocationupdate_test.html', {'form_data': finaldictlist, 'total': computedtotal, 'productnotifications': product_notifications, 'variantnotifications': variant_notifications, 'allocationnotifications': allocation_notifications, 'notificationscount': total_notifications, 'primarykey': pk, 'colors': variant_colors, 'variantStats': variant_stats, 'allocation': allocation, 'status':status, 'sizes': sizes, 'uniqueSizes': uniqueSizes, 'displayColors': displayColors, 'stores': clientStores})



    else:


        if request.method == 'POST':

            totalvariantsunits = gettotalvariants(filtered_variants)


            if allocation.request_amount > totalvariantsunits:

                variantunits = getVariantUnits(variants, filtered_variants)

                if storeSpecificList:

                    dividedallocationunits = divideSpecificAllocations(variantunits, storeSpecificList)

                else:

                    dividedallocationunits = divideAllocations(variantunits, allocation)

                listofDictAllocations = listToDict(allocation, dividedallocationunits)

                computedtotal = sum([variant[1] for variant in variantunits])

                finaldictlist = []

                for entry in listofDictAllocations:

                    store = entry[0]['client_store']

                    finaldictlist.append([store, entry])

                product_notifications = mainviews.get_product_notifications()

                variant_notifications = mainviews.get_variation_notifications()

                allocation_notifications = mainviews.get_allocation_notifications()

                total_notifications = mainviews.get_notifications_count(product_notifications, variant_notifications, allocation_notifications)

                variant_stats = get_variant_stats(variants, variantunits, variant_colors)
            status = getstatus(computedtotal, allocation.request_amount)
            displayColors = []
            sizes = getSizes(listofDictAllocations, displayColors)
            uniqueSizes = getUniqueSizes(variant_stats)
            return render(request, 'allocation/allocationmanage_test.html', {'form_data': finaldictlist, 'total': computedtotal, 'productnotifications': product_notifications, 'variantnotifications': variant_notifications, 'allocationnotifications': allocation_notifications, 'notificationscount': total_notifications, 'primarykey': pk, 'colors': variant_colors, 'variantStats': variant_stats, 'allocation': allocation, 'status':status, 'sizes': sizes, 'uniqueSizes': uniqueSizes, 'displayColors': displayColors, 'stores': clientStores})


        proportionlist = getProportionsFiltered(variants, totalvariantsunits, filtered_variants)
        print(proportionlist)
        allocationunits = getallocationunits(proportionlist)
        totalallocationunits = gettotalallocationunits(allocationunits)

        while totalallocationunits != allocation.request_amount:
            if totalallocationunits > allocation.request_amount:
                chosenIndex = findMostAllocatedStock_test(allocationunits, filtered_variants)
                allocationunits[chosenIndex][1] -= 1
                totalallocationunits -= 1
            else:
                chosenIndex = findMostAvailableStock_test(allocationunits, filtered_variants)
                allocationunits[chosenIndex][1] += 1
                totalallocationunits += 1

        if storeSpecificList:
            dividedallocationunits = divideSpecificAllocations(allocationunits, storeSpecificList)
        else:
            dividedallocationunits = divideAllocations(allocationunits, allocation)

        listofDictAllocations = listToDict(allocation, dividedallocationunits)
        computedtotal = sum([allocation[1] for allocation in allocationunits])
        finaldictlist = []
        for entry in listofDictAllocations:
            store = entry[0]['client_store']
            finaldictlist.append([store, entry])
        product_notifications = mainviews.get_product_notifications()
        variant_notifications = mainviews.get_variation_notifications()
        allocation_notifications = mainviews.get_allocation_notifications()
        total_notifications = mainviews.get_notifications_count(product_notifications, variant_notifications, allocation_notifications)
        variant_stats = get_variant_stats(variants, allocationunits, variant_colors)

        status = getstatus(totalallocationunits, allocation.request_amount)

        displayColors = []

        sizes = getSizes(listofDictAllocations, displayColors)

        uniqueSizes = getUniqueSizes(variant_stats)

        return render(request, 'allocation/allocationmanage_test.html', {'form_data': finaldictlist, 'total': computedtotal, 'productnotifications': product_notifications, 'variantnotifications': variant_notifications, 'allocationnotifications': allocation_notifications, 'notificationscount': total_notifications, 'primarykey': pk, 'colors': variant_colors, 'variantStats': variant_stats, 'allocation': allocation, 'status':status, 'sizes': sizes, 'uniqueSizes': uniqueSizes, 'displayColors': displayColors, 'stores': clientStores})


@login_required

def manage_allocation_test(request, pk):

  allocation = Allocation.objects.get(pk=pk)

  orders = Order.objects.filter(allocation_id = allocation.id)

  stores = ClientStore.objects.filter(client=allocation.client)


  if len(orders) > 0:

    return redirect('update_allocation', pk = pk)


  if request.method == 'GET':

      variants = Variant.objects.filter(product_id = allocation.product_id)

      totalvariantsunits = gettotalvariants(variants)

      selectcolors = getuniquecolors(variants)


      if allocation.request_amount > totalvariantsunits:

        variantunits = [[variant.id, variant.stock] for variant in variants]

        dividedallocationunits = divideAllocations(variantunits, allocation)

        listofDictAllocations = listToDict(allocation, dividedallocationunits)

        computedtotal = sum([variant[1] for variant in variantunits])

        finaldictlist = []

        for entry in listofDictAllocations:

          store = entry[0]['client_store']

          finaldictlist.append([store, entry])

        product_notifications = mainviews.get_product_notifications()

        variant_notifications = mainviews.get_variation_notifications()

        allocation_notifications = mainviews.get_allocation_notifications()

        total_notifications = mainviews.get_notifications_count(product_notifications, variant_notifications, allocation_notifications)

        variant_stats = get_variant_stats(variants, variantunits, selectcolors)

        status = getstatus(computedtotal, allocation.request_amount)

        displayColors = []
```

```
            sizes = getSizes(listofDictAllocations,
displayColors)

            uniqueSizes =
getUniqueSizes(variant_stats)

            return render(request,
'allocation/allocationmanage_test.html',
{'form_data': finaldictlist, 'total': computedtotal,
'productnotifications': product_notifications,
'variantnotifications': variant_notifications,
'allocationnotifications': allocation_notifications,
'notificationscount': total_notifications,
'primarykey': pk, 'colors': selectcolors,
'variantStats': variant_stats, 'allocation':
allocation, 'status':status, 'sizes': sizes,
'uniqueSizes': uniqueSizes, 'displayColors':
displayColors, 'stores': stores})



        proportionlist = getproportions(variants,
totalvariantsunits)

        allocationunits =
getallocationunits(proportionlist)

        totalallocationunits =
gettotalallocationunits(allocationunits)



        while totalallocationunits !=
allocation.request_amount:

            if totalallocationunits >
allocation.request_amount:

                chosenIndex =
findMostAllocatedStock(allocationunits)

                allocationunits[chosenIndex][1] -= 1

                totalallocationunits -= 1

            else:

                chosenIndex =
findMostAvailableStock(allocationunits)

                allocationunits[chosenIndex][1] += 1

                totalallocationunits += 1


        variant_stats = get_variant_stats(variants,
allocationunits, selectcolors)

        uniqueSizes =
getUniqueSizes(variant_stats)

        dividedallocationunits =
divideAllocations(allocationunits, allocation)

        listofDictAllocations =
listToDict(allocation, dividedallocationunits)

        displayColors = []

        sizes = getSizes(listofDictAllocations,
displayColors)

        computedtotal = sum([entry[1] for entry in
allocationunits])

        finaldictlist = []

        for entry in listofDictAllocations:

            store = entry[0]['client_store']

            finaldictlist.append([store, entry])

        product_notifications =
mainviews.get_product_notifications()

        variant_notifications =
mainviews.get_variation_notifications()

        allocation_notifications =
mainviews.get_allocation_notifications()

        total_notifications =
mainviews.get_notifications_count(product_noti
fications, variant_notifications,
allocation_notifications)

        status = getstatus(computedtotal,
allocation.request_amount)

        return render(request,
'allocation/allocationmanage_test.html',
{'form_data': finaldictlist, 'total': computedtotal,
'productnotifications': product_notifications,
'variantnotifications': variant_notifications,
'allocationnotifications': allocation_notifications,
'notificationscount': total_notifications,
'primarykey': pk, 'colors': selectcolors,
'variantStats': variant_stats, 'allocation':
allocation, 'status': status, 'sizes': sizes,
'uniqueSizes': uniqueSizes, 'displayColors':
displayColors, 'stores': stores})


    else:

        targetAllocation =
Allocation.objects.get(pk=pk)

        client = allocation.client

        clientStores =
ClientStore.objects.filter(client = client.id)

        variants = Variant.objects.filter(product_id
= allocation.product_id)

        for store in clientStores:

            for variant in variants:

                baseString = '{}-{}-'.format(store,
variant)

                formVariant =
request.POST.get(baseString + 'variant')

                formStore =
request.POST.get(baseString + 'clientStore')

                formAmount =
request.POST.get(baseString + 'formValue')

                targetVariant = Variant.objects.get(pk
= formVariant)

                newOrder =
Order(allocation=targetAllocation, client_store
= ClientStore.objects.get(pk = formStore),
variant = targetVariant, amount =
int(formAmount))

                targetVariant.stock -=
int(formAmount)

                targetVariant.save()

                newOrder.save()

        allocation.user = request.user

        allocation.order_date = date.today()

        allocation.save()

        return
redirect(reverse('order_allocation_index_page'))


def
findMostAllocatedStock_test(listOfAllocationU
nits, listOfFilteredVariants):

    key = 0

    is_in_filtered = False

    keyproportion = 0

    counter = 0

    while not is_in_filtered:


        variant = Variant.objects.get(pk =
listOfAllocationUnits[counter][0])

        keyproportion =
listOfAllocationUnits[counter][1] / variant.stock

        if variant in listOfFilteredVariants:

            key = counter

            is_in_filtered = True

        counter += 1



    for i in range(1, len(listOfAllocationUnits)):

        variant = Variant.objects.get(pk =
listOfAllocationUnits[i][0])

        proportion = listOfAllocationUnits[i][1] /
variant.stock



        if proportion > keyproportion:
            if variant in listOfFilteredVariants:
                key = i
                keyproportion = proportion

    return key

def
findMostAllocatedStockCustom_test(listOfAllo
cationUnits, listOfFilteredVariants,
listOfVariants):
    key = 0
    is_in_filtered = False
    keyproportion = 0
    counter = 0
    while not is_in_filtered:

        targetVariant = [variant for variant in
listOfVariants if variant['id'] ==
listOfAllocationUnits[counter][0]]
        keyproportion =
listOfAllocationUnits[counter][1] /
targetVariant['stock']
        for entry in listOfFilteredVariants:
            if targetVariant[0]['id'] == entry.id:
                key = counter
                is_in_filtered = True
        counter += 1


    for i in range(1, len(listOfAllocationUnits)):
        targetVariant = [variant for variant in
listOfVariants if variant['id'] ==
listOfAllocationUnits[i][0]]
        proportion = listOfAllocationUnits[i][1] /
targetVariant['stock']


        if proportion > keyproportion:
            for entry in listOfFilteredVariants:
                if entry.id == targetVariant[0]['id']:
                    key = i
                    keyproportion = proportion
                    break

    return key

def
findMostAvailableStock_test(listofallocatedunit
s, listOfFilteredVariants):
    key = 0
    is_in_filtered = False
    counter = 0
    keyproportion = 0
    while not is_in_filtered:
        variant = Variant.objects.get(pk =
listofallocatedunits[counter][0])
        keyproportion =
listofallocatedunits[counter][1] / variant.stock
        if variant in listOfFilteredVariants:
            key = counter
            is_in_filtered = True
        counter += 1

    for i in range(1, len(listofallocatedunits)):
        variant = Variant.objects.get(pk = listofallocatedunits[i][0])

        proportion = listofallocatedunits[i][1] / variant.stock


        if proportion < keyproportion:

            if variant in listOfFilteredVariants:

                key = i

                keyproportion = proportion


    return key


def findMostAvailableStockCustom_test(listofallocatedunits, listOfFilteredVariants, listOfVariants):

    key = 0

    is_in_filtered = False

    counter = 0

    keyproportion = 0

    while not is_in_filtered:

        targetVariant = [variant for variant in listOfVariants if variant['id'] == listofallocatedunits[counter][0]]

        keyproportion = listofallocatedunits[counter][1] / targetVariant[0]['stock']

        for entry in listOfFilteredVariants:

            if entry.id == targetVariant[0]['id']:

                key = counter

                is_in_filtered = True

        counter += 1


    for i in range(1, len(listofallocatedunits)):

        targetVariant = [variant for variant in listOfVariants if variant['id'] == listofallocatedunits[i][0]]

        proportion = listofallocatedunits[i][1] / targetVariant[0]['stock']


        if proportion < keyproportion:

            for entry in listOfFilteredVariants:

                if targetVariant[0]['id'] == entry.id:

                    key = i

                    keyproportion = proportion

                    break


    return key
```

requirements.txt

```
# This file may be used to create an environment using:

# $ conda create --name <env> --file <this file>

# platform: win-64

asgiref=3.4.1=pyhd3eb1b0_0

ca-certificates=2023.5.7=h56e8100_0

certifi=2023.5.7=pyhd8ed1ab_0

crispy-bootstrap5=0.7=pypi_0

django=3.2.5=pyhd3eb1b0_0

django-crispy-forms=2.0=pyhd8ed1ab_0

django-debug-toolbar=4.1.0=pyhd8ed1ab_0

freetype=2.10.4=h546665d_1

jpeg=9d=h8ffe710_0

krb5=1.17.1=hc04afaa_0
```

```
lcms2=2.12=h2a16943_0
libpng=1.6.37=h1d00b33_2
libpq=12.2=h3235a2c_0
libtiff=4.2.0=h0c97f57_3
lz4-c=1.9.3=h8ffe710_1
mysqlclient=1.4.6=py38_0
olefile=0.46=pyh9f0ad1d_1
openjpeg=2.4.0=hb211442_1
openssl=1.1.1l=h8ffe710_0
pillow=8.2.0=py38h9273828_1
pip=20.2.4=py38_0
psycopg2=2.8.5=py38h7a1dbc1_0
python=3.8.2=he1778fa_13
python_abi=3.8=2_cp38
pytz=2020.1=py_0
reportlab=3.5.68=py38hc3ca807_1
setuptools=50.3.0=py38h9490d1a_1
sqlite=3.33.0=h2a8f88b_0
sqlparse=0.4.1=py_0
tk=8.6.10=he774522_0
typing_extensions=3.7.4.3=py_0
vc=14.1=h0510ff6_4
vs2015_runtime=14.16.27012=hf0eaf9b_3
wheel=0.35.1=py_0
wincertstore=0.2=py38_0
xz=5.2.5=h62dcd97_1
zlib=1.2.11=vc14h1cdd9ab_1
zstd=1.5.0=h6255e5f_0
```

XI. Acknowledgements

I would like to give my thanks to all the people that provided me with the strength, support, encouragement and joy that I have had the honor to be a part of throughout my life and my time in the university.

Firstly, I would like to thank Ma'am Avegail D. Carpio for being my adviser for her calmness, her support and her expertise in the development of my thesis. She served as my supervisor and supporter throughout the project's lifespan. Her knowledge and experience proved useful and essential in making the project robust and readily usable for the project's client.

Next, I would like to thank my family for their everlasting love and support through the years of my life. Their patience and perseverance throughout the events of my life has continuously inspired me to do the same and to always be better in all aspects of my life. My mother, father, and my sister continuously help me to be a better person and to be prepared for life in the university and for the eventual life ahead, through their own experiences and beliefs. Additionally, my extended family has provided me with their own experiences, advice, and opinions that helped shaped me into the person I am today. Their collective presence in my life has also helped me through the trying times in recent years, in helping me to come to terms with the uncertainties and fears that I constantly enter my mind.

Lastly, I would like to thank my block mates for being there for me through my stay in the university. They are my second family as we have shared many precious and significant memories together, and together we forge through to the end of our stay. We may not have all graduated together, but I still consider ourselves to be one group, as we continue to support and to provide help to one another.