School of Electronic Engineering and Computer Science
Queen Mary University of London

Final REPORT
2020-21

Programme of Study:
Computer Science BSc(Hons)

Project Title:
allPOS – Electronic Point of Sale System

Supervisor:
Paulo Oliva

Student Name:
Anton Petrov Nyagolov

Final Year
Undergraduate Project 2020/21

Abstract

Over the last five years in the UK, I have worked in the hospitality industry, more specifically in coffee shops. Managing such a business successfully does not require only hard-working staff but also using software that automates time-consuming tasks such as stock managing, employee scheduling or order taking.

There is plenty of choices on the market when it comes to management software automation. Some applications are designed to take orders from customers, other's main functionality is to provide a stock management solution, and some of them offer employee scheduling functionality. However, there is no single application that incorporates the solution to the three problems mentioned above.

Moreover, to my knowledge, all of them are device-dependent, requiring businesses to be tied to specific hardware.

This project would close the gap by providing a single web application that would combine an electronic point of sale, employee scheduling system and stock managing functionality.

Additionally, the software would be hardware independent and its implementation would allow it to be run on most electronic devices.

Those type of managing systems usually employ peripherals such as card readers, bar code scanners, printers and many more. This project would aim to combine the three software solutions into a single hardware-independent application and therefore, implementation for such devices would be left for future work.

Table of Contents

List of Figures

List of Tables

1. Introduction

During the past five years of residing within the UK, I have been in full-time employment, essentially working as a Head of Coffee in several companies around London. Head of Coffee is a management role in speciality coffee shops, including many administrative tasks such as the management of everyday operations, staff rota creation, stock management, and others.
Furthermore, managers usually have a larger number of responsibilities than employees. They not merely should work and operate the same service as other employees, but managers likewise have to:

- Keep track of stock raw goods
- Order raw goods when low on stock
- Calculating raw material inventory
- Inventory optimisation
- Report generation
- Employee scheduling
- Cash reports
- Dealing with Staff Holiday Request
- Analyse financial data and conclude financial reports

Those are a couple of essential management duties. During the time I worked in those commercial environments, I noticed that few organisations were managing with those matters differently. Three primary types of software are utilised across the sector to automate a portion of the tasks referenced above:

- EPOS also know as Electronic Point of Sale System
- Employee Scheduling Software
- Stock Management Software

Every one of those three frameworks is sold independently as a solitary product. Premium bundles exist, which implements a combination of them. Notwithstanding, as the name of the product suggests (premium), it is highly-priced and unaffordable for most single store retailers.
There are several reasons behind the absence of full system implementation (3 systems used by one retailer) across retail and hospitality, but the primary ones are cost and cross-platform adaptability.
My idea is to build up a device-independent application that would unravel all those issues collectively, and besides, it will provide some extra functionalities. This project will combine an EPOS, ESS and SMS system into one simple to use web-based Django application. It will include all the main features that a native software supports. Additionally, the Stock Management Software would be able to gain data from past sale patterns and automate the ordering process of crude products with insignificant administration commitment.

Currently, I am taking a module in Web Programming learning the Django Framework. My software experience is mostly school-related, implying that I have not developed any complex application previously. The main challenge for this project will be the intricacy to consolidate the three sorts of frameworks with every one of their functionalities into one web application and execute them so that they exchange information.

1.1. Objectives

The project summarises four high-level objectives that the app will consist of:

- Implemented and fully functioning Electronic Point of Sale
- Implemented and fully functioning Employee Scheduling
- Implemented and fully functioning Stock Management
- Integrated automate raw goods ordering

1.2. Scope

There is an extensive list of companies who develop software and hardware such as EPOS, ESS, and SMS systems for the hospitality and retail industry. Heretofore, I was unable to distinguish an organisation that has built up every one of the three strategies together.
The software industry is a highly dynamic and growing sector. New software is released every day, tackling various issues in innovative ways. Therefore, it ought to be considered that this project is undertaken under a tight time frame and resources. As a result, 5 of the most rated EPOS, ESS, and SMS available on the market were researched, studied and analysed.

Some of the EPOS companies provide custom hardware, designed specifically for their software. For instance, EPOS software is extendable to work with barcode scanners, cash drawers, receipt printers, card readers, voice control stations and several more.
Nonetheless, this research only takes into account the product execution of the systems discussed in the introduction section (page 5).

1.3. Overview

Chapter one, *Introduction*, introduces the project: "allPOS - cloud-based Electronic Point of Sale" and provide the reasoning behind the idea of undertaking this specific project.

Chapter two, *Research*, illustrates some basic concepts obtained from the background research to familiarise the user with the used technology and terminology and how this project fits in.

Chapter three, *Requirements Specification,* provides the functional and non-functional requirements of the project. Furthermore, use-case examples will be provided.

Chapter four, *Design and Implementation*, describes the different design approaches taken in the implementation process, high and low-level design, application walk-through and GUI.

Chapter five, *Testing*, walks the reader through all the testing techniques and results. Furthermore, it lists any bugs and flows found and they resolution.

Final Chapter, *Conclusion*, reports study limitations, outlines the scope for further studies and conclude the project.

2. Research

Research on EPOS systems and their architectural design has been conducted to identify the primary potential issues, acquire knowledge of the EPOS system market, research into existing software companies that provide similar solution and identify their strengths and weaknesses. Comprehensive software architecture research has been conducted to gain a comprehension of how complex systems are build and the diverse design patterns and principles they implement.

2.1. Django Web Framework

Before moving toward the aspects of what an EPOS system is and how it operates, we will proceed with a detailed introduction on the Django Web Framework. The introduction in this subsection would portray the technical aspects of the framework and familiarise the readers with the fundamental functionalities.

2.1.1. What is a Web Framework

For building applications that interfere with a server database and the web, developers are using web application frameworks. Web frameworks aim to encapsulate the low-level implementation of web services and provides tools and libraries to simplify the development process.

Nowadays, web applications are becoming more sophisticated than ever. A modern web application is capable of producing a dynamic based content, it communicates with many micro-services simultaneously, maintains a significant amount of data and many more. One of the most common frameworks is the Model-View-Controller design pattern which intends to divide the code for each application component into independent modules.

2.1.2. Django

A programming language named Python was initially created in the 1990s by Guido van Rossum as a member of the National Research Institute of Mathematics and Computer Science. Python is an interpreted, high-level programming language and was first released in 1991 [1].
'Django, on the other side, is a high-level Python web framework that enables rapid development of secure and maintainable websites' [5]. Django is free and open-sourced, with an incredible, thriving community, comprehensive documentation and free support.

2.1.3. Django architecture

Django utilises the Model-View-Controller (a.k.a MVC) architecture which distributes code responsibilities into different modules:

- The model outlines a relational database abstraction, and it is the logical low-level data structure. Models abstract the creation of the relational database and implement a simple to use interface for database queries. The interface acts as a mechanism between the high-level part of the framework and the database. Each model maps to a single database, and it is a definitive source of information about the data [2].

- The view also know as a Template, is the component of the Django application that represents the Graphical User Interface. The GUI is composed of HTML, CSS and JavaScript files to provide a dynamic and modern User Interface. Most Django applications use the view to render the model database[2].

- The controller is the most vital part of the application. In Django terms, the controller is known as a view. Views are python files that are responsible for managing web request and responses, manipulating data coming from the models, connecting the model's data with the user interface (view - the component above controller), user authentication, data sanitisation, managing cookies and user sessions, and many more [2].

Figure 2-1 illustrates how the Model-View-Controller generates a response based on some user request.
For instance, if we want to register on a website "A" when the user inserts its personal data into an HTML form, the data is sent with a "POST" request to the controller, the controller has the responsibility to sanitise the data, and substantiate that the data is valid. After validation, data is sent to the model. Once the data is received and processed, the model has the responsibility to send back to the controller an acknowledgement. The controller then gets the acknowledgement and transfers it to the view. The view renders the graphical representation of the model's response. For instance, in our case, a message "Registration Successful" could be printed onto the user's screen.

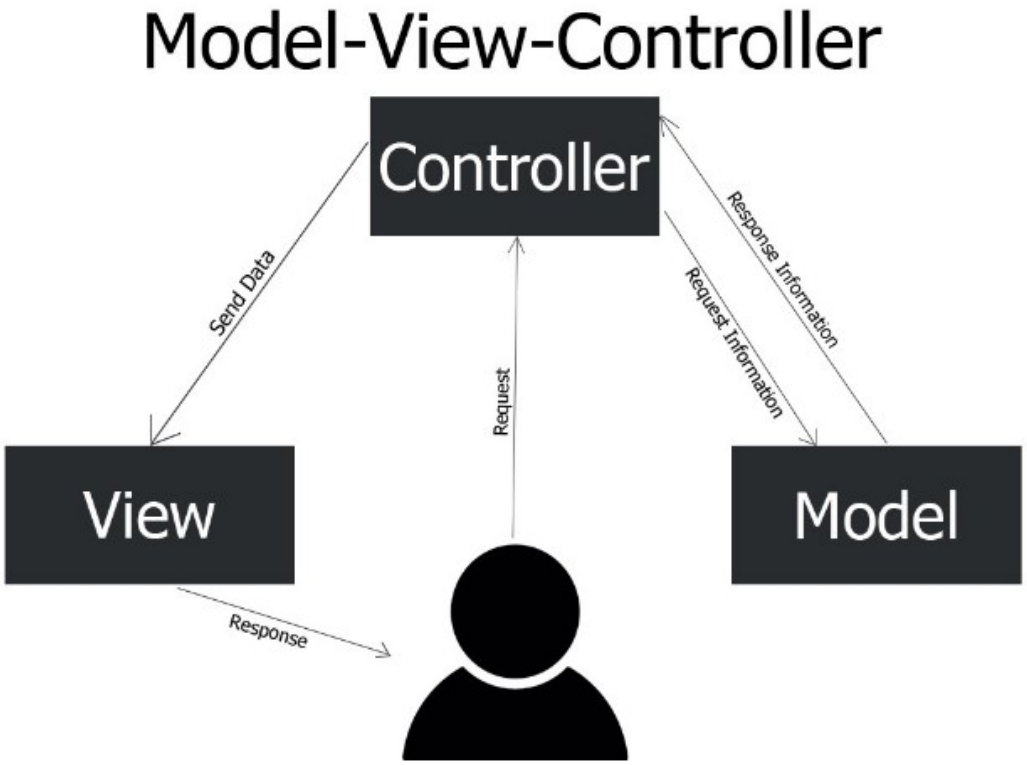


Figure 2-1: Model-View-Controller design pattern [3]

2.2. Electronic Point of Sale

An EPOS, also known as electronic point of sale system represents hardware and software that work collectively in an attempt to process transactions more efficiently. The hardware usually represents a central computer. Upon business prerequisites, companies can order additional devices identified as "peripherals" when connected to the computer, they incorporate software usability.

The system accommodates several software customisations so it can fit into the business requirements and help with specific analytics and data aggregation explicit to the company.

It is installed either directly to the computer, or connected to the cloud, allowing monitoring and analysation of sales across several locations. Business performance examination shifts into a more affordable and easily obtainable assignment, as a result of the electronic tracking of sales and reports. At a glance, it can show what the store sells, where it sells, whom it sells to, and which of the salespeople sells it best. Moreover, whether a business is in the retail or hospitality sector, EPOS systems can modernise the processes and revolutionise the marketing divisions to suit the buying behaviour of the most loyal customers.

Figure 2-2: EPOS system User Interface

Another critical aspect of a successful EPOS system is the Graphical User Interface. 'Some time ago POS interfaces have replaced traditional cash registers in offline stores, and now they do a lot more than just process payments. Accordingly, POS system designers must embrace the modern context and offer complex solutions that benefit the customer and simplify things for a business.' [4]
Figure 2-2 demonstrates the sophistication behind the graphical user interface of an EPOS system. According to [4], those are the most common EPOS design principles:

- Many unnecessary UI elements could cause a cognitive load, therefore, simplicity is one of the main approaches during the EPOS design planning and implementation. 'In a nutshell, follow the KISS principle (Keep it Simple, Stupid) to keep everything right' [4]

- Usability, as a result of a user who utilises a specific product to accomplish a particular goal, the most critical elements to examine is the effectiveness, the efficiency and satisfaction from the results [6].

- Another aspect of an exemplary GUI is design consistency [4]. Implementation of multiple new elements on every single page does not account for good graphical design, and it often confuses users. Moreover, the reusability of a set of UI elements would significantly increase design optimisation.

- Page/view navigation should be straightforward to achieve. Furthermore, all tasks flow need to be logically designed. In this manner, it would promote the 'KISS' principle and incorporate Usability, providing an obvious and straightforward flow.

2.3. Employee Schedule

Retail and food alike businesses use an employee scheduling system to support the process of employee scheduling, enhancing punctuality of their employees while providing them with the flexibility of creating their working patterns. Furthermore, employee scheduling systems provide transparency by globally promoting every new schedule consisting of all the employees that are part of a particular department. As a result, it enhances communication between them.
In contrast, spreadsheets scheduling is very time-demanding and complex. As the business grows, the number of employees grow accordingly, making spreadsheet scheduling hard to manage and change on request.
Moreover, employee scheduling software also reduces labour expenses. According to a Hospitality Technology study (cited in [7]), an employee scheduling system accomplished an average annual saving of 2.2% labour cost.
Besides, by utilising shift booking frameworks, supervisors can diminish planning time, ease the stress, and forestall communication breakdowns among managers and staff. Particularly in the hospitality sector, where workers support is pivotal, zeroing in on time adaptability and representative prosperity will, at last, bring expanded consumer loyalty and better performance.

Hourly employees, for instance, students or workers with kids who cannot work all day, need adaptability in their work routines to study or take care of their youngsters. In any case, in the food-service business, it is customary for employees not to have control over their work routines. According to a study

from Wonk and Ko (cited in [7]), it is a common issue for workers who experience the ill effects of burnout on account of the unpredictability of the time-sensitive clash and need to work through clashing requests in their personal lives.

By changing to a work planning framework, organisations can smooth out their labour force plans and improve the correspondence between hourly workers and their managers. Employees can get their timetables straightforwardly on their phones and can easily demand downtime with one click. Moreover, online employee booking has made greater adaptability to the hourly workers work routines since they have more control over their work hours and have more impact over their timetables. This, thus, could cause an individual to feel that work encounters improve the nature of their lives and that there is more prominent harmony among work and home balance [7].

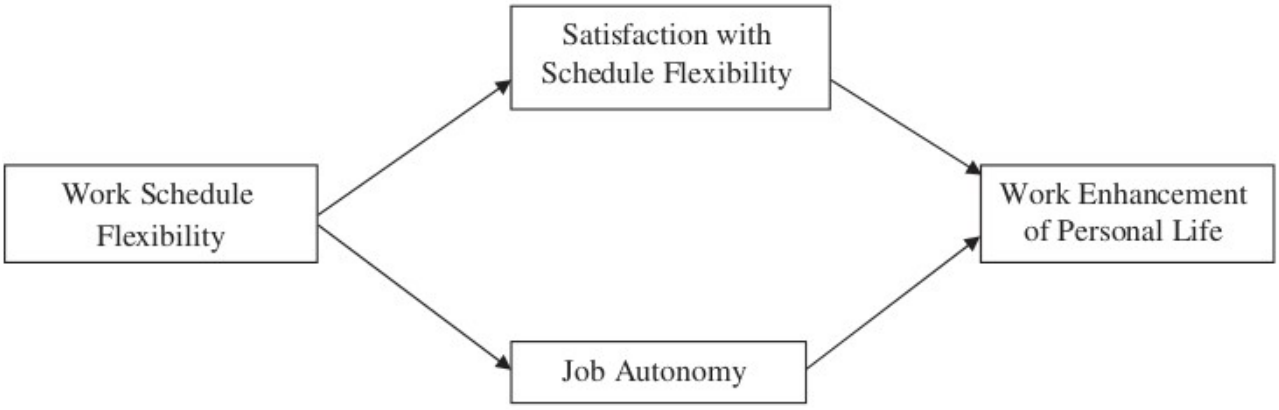


Figure 2-3: Mechanisms for improving employees quality of life

Figure 2-3 illustrates the two essential components to improve employees personal satisfaction. It enhances schedule flexibility and provides job autonomy which is beneficial to employees personal life.

2.4. Stock Control

With regards to stock, the hospitality sector is one of the most disregarded industries yet as should be evident from the amazingly tough food-service market, what should matter most to any food business is the viable administration of orders, following menu items and in general stock management.

The term inventory is highly utilised within the accounting department. It refers to merchandise that is in different phases of being made available for sale, including:

- Completed products (that are accessible to be sold)
- Work-in-progress (products that are not yet completed)
- Crude materials (those are utilised to produce more competed goods)

Businesses ensure that all bookkeeping records are up-to-date and accurate by taking a stock count towards the end of each bookkeeping period, which is commonly quarterly or over year. Furthermore, companies that do a day by day stock count are considered to take interminable stock because their count is consistently current.
A strategy called "just-in-time" inventory is utilised to reduce the number of items that are in stock. By using this method, companies have stock just in time to meet customer needs. 'As a result, there is less inventory sitting around, waiting to be produced or sold.' [8].

According to [9], there are four types of inventory in the hospitality and retail business:

- The current inventory that a business has on a stock is called a "sitting inventory" Sitting inventory could be measured either by total weight, a quantity or as a monetary amount. [9]
- The amount of products a venue uses on a given time-frame is called "depletion", and its measurement is logged in the system likewise the sitting inventory. [9]
- When ordering goods, a business should know how long the sitting ordered inventory will last for. In this instance, the sitting inventory is divided by the average depletion time. Consequently, it abstains from running out of items, over-ordering and food wastage. [9]
- A difference within what a company ordered as a raw good and what the company sold as a single product is called a "variance". Variance is crucial, especially in the hospitality business. It is essential to understand how much of raw goods is wasted, as it helps get to the bottom of the wastage problems and reduce them. [9]

A stock management system implements an intuitive way to register all raw products that are received from a supplier. It allows for tracking their quantities and analysing how different items are used in a given time frame while reducing wastage.

The stock management would be implementing cron-job that represents a background process executed at a given time period (based on user settings). The cron-job would have the functionality of checking for ingredients that are low on stock. Once those ingredients are collected, a background service would generate emails for the needed ingredients and send those emails to their corresponding supplier. Based on user's requirements, they can enable and disable automated ordering, keep track of previously ordered ingredients and get notified once an email has been sent to a raw goods supplier.

2.5. Critical Review of existing applications

In this section, I will be evaluating similar applications that are available on the market and will be comparing their functionalities and features. However, there are limitations to this practice:

1. A functionality description or review does not cover the specific feature in-depth.

2. It only describes what the software does, but not how it does it.

3. GUI flows cannot be analysed.

2.5.1. Comparison to other EPOS systems

After extensive Google research of the top five EPOS systems, the following companies appear to provide a significantly large amount of functionalities. Besides, those companies are one of the most rated in the market:

- TouchBistro
- vendPOS
- Lightspeed Restaurant
- Square Point of Sale
- Epos Now

2.5.2. Non-functional EPOS comparison

Application name	allPOS (my project)	Touch Bistro	Vend POS	Lighspeed Retail	Square Point of Sale	Epos Now
1. Built for	Hospitality and Retail	Hospitality	Retail	Hospitality and Retail	Hospitality and Retail	Hospitality and Retail
2. Pricing	Initially free	£49+ per iPad/mo	£49+ per till/mo	£69+ per till/mo	Free	£25+ per till/mo
3. Compatible devices	Any device that has internet browser and internet connection	Ipad	Ipad, Mac, Windows	Ipad, Mac, Windows	Ipad, iPhone, Android tablets and smartphones	Ipad, Android tablets, Mac, PC touchscreen

Table 2-I: Non-functional EPOS comparison

2.5.3. Inventory Management comparison

Inventory Management	allPOS (my project)	Touch Bistro	Vend POS	Lighspeed Retail	Square Point of Sale	Epos Now
1.Categories and variants	Yes	Yes	Yes	Yes	Yes	Yes
2.Stock levels	Yes	Yes	Yes	Yes	Yes	Yes
3.Auto-supply	Yes	--	Yes	--	--	Yes
4.Ingredient Tracking	Yes	Yes	--	Yes	--	Yes
5.Automated decision-ordering	Yes	--	--	--	--	--

Table 2-II: Inventory Management comparison

2.5.4. Reports and Analytics comparison

Reports and analytics	allPOS (my project)	Touch Bistro	Vend POS	Lighspeed Retail	Square Point of Sale	Epos Now
1.Daily Reports	Yes	Yes	Yes	Yes	Yes	Yes
2.Close/end-of-day report	Yes	Yes	Yes	Yes	--	Yes
3.Sales analytics	Yes	Yes	Yes	Yes	Yes	Yes

Table 2-III: Reports and analytics comparison

2.5.5. Employee and Store management comparison

Employee and store management	allPOS (my project)	Touch Bistro	Vend POS	Lighspeed Retail	Square Point of Sale	Epos Now
1.User logins	Yes	Yes	Yes	Yes	Yes	Yes
2.Different permission levels	Yes	Yes	Yes	Yes	Yes	Yes
3.Staff analytics	Yes	Yes	Yes	Yes	Yes	Yes
4.Shift-scheduling	Yes	--	--	--	--	--

Table 2-IV: Employee and store management comparison

2.6. Research conclusion

The tables above illustrate the different features of the reviewed applications. It can be seen that there is a gap between their functionalities, platform compatibility, and cost.
Three of the five companies provide "auto-supply" functionality. This feature allows managers to generate a list of of products with fixed quantity for each, that will be ordered on a given period. However, it needs to be noted that this feature is distinct from my idea of "auto-ordering". While the "auto-supply" feature is capable of ordering automatically on a given time frame, it is ineffective in determining how many of the specific items need to order.
On the other side, I am planning to implement an intuitive system that will automatically re-supply items based on some decisions.
Furthermore, three of the five systems are highly expensive, eliminating small and medium retailers from their customer targets.
From the reviewed applications, although all five systems are device-dependent, only one company provides multi-device comparability, however, it lacks one of the most critical features.
Moreover, none of the reviewed applications implements an employee scheduling system and automated ordering feature natively, which could be the gap my project could fill in.

2.7. Summary

This chapter introduces the reader into different technologies, their high-level functionalities and specifications. It describes what a web framework is, Django Web framework, Django architecture, Moved-View-Controller.
Besides, it presents the user with a definition of the three systems that my project will be build of.

The concludes with a comparison research conclusion. It compares the top five rated systems on the market, analysing their features and non-functional requirements.

3. Requirements Specification

In this part, I will introduce the various requirements for this project. Functional requirements would be identified first, consisting of primary and secondary requirements. The majority of the functional requirements are introduced in the tables of chapter 2, subsection 2.4.
It will then continue with non-functional requirements. Non-functional requirements do not affect the functioning of the application but are essential for users.

3.1. Primary Functional Requirements

Primary functional requirements are the critical components of this project. Those are the functionalities that must be implemented to achieve the aim of this project combining the three systems features into one whole application.

- **Main menu navigation**: When a user logs in, the "main menu navigation" screen is shown in the left hand side and the EPOS system would takes most of the application screen.
  Users should be able to swap between different parts of the application using the main menu navigation on left. The main functionalities in the left-hand side menu include links to the “EPOS” system screen, “Employee Schedule” screen, admin “Stock Management” screen, Settings screen and superuser Admin panel. Additionally, each of those, mentioned earlier, would have subsections in their main view so users can quickly navigate to the options of the three main applications.

- **POS:** The point of sale screen would be split into two equal-size parts. The left-hand side would represents the order that is currently taken including each product, its quantity, cost per unit and total cost of the product.
  Additionally, an employee can edit the product quantity, enter the order number, the type of order (have in or take away), sending the order to the kitchen/bar, cancel the order or putting the order on hold.
  The right-hand side has to render all the categories in a horizontal scrollable tab with buttons. On selecting a specific category, the front-end displays all its relative products in the form of buttons.

- **Payment:** The payment screen would consists of an order number, discounts applied (if any), service fee, subtotal of the order, tax, the total amount of the order and balance due. Furthermore, the following options would be added: add tip amount, generate a gift receipt, email the receipt of the order, reprint receipt and refund the order.

- **Employee Schedule:** The main screen of this application would consist of a table showing:

    - the employees working today
    - date
    - employee start time
    - employee end-time
    - the time an employee clocked in for the day
    - the time an employee clocked out for the day

    Furthermore, this view would implement clickable links to another three subsections of the scheduling functionality. Users would be able to view the weekly rota calendar for all the employees including their working times and days off. Additionally, managers should generate weekly, monthly or daily rota from the "Update Rota" subsection of the application. Another feature of the scheduling module would be the review of the timestamp history. This feature will render all the clock-ins and clock-outs for each user so shift activity can be monitored on request.

- **Stock Management**: Stock management screen would allow managers to see current ingredients, products and their related inventory ingredients. Moreover, it will contains all the ingredients transactions, allowing managers to track the lifespan of a certain ingredient quantity over a specific time frame.
  In addition, adding, deleting and editing of products, ingredients and inventory ingredients would be implemented providing an easy and intuitive small admin panel.

3.2. Secondary Functional Requirements

Secondary requirements are crucial extra features of the primary requirements, nevertheless, secondary requirements have to be implemented once the primary requirements are satisfied.

- **Employees Profiles:** the profile is used for several features:

    - General employee information

    - Employee scheduling
    - Employee log in/out of the system, clock in/out
    - Order details – every order will consist of the employee who made the order and took the payment.

- **Log In/Out:** A crustal part of the system is to follow the activities of each employee. For instance, each order would include the employees name. When the application is not in use it automatically logs out the last user who used the application and a login screen is shown.

- **Clock In/Out:** When employee starts/finishes it’s shift it needs to use this clock in/out feature in order to record it’s starting/finishing shift time. Users who have not used the “Clock In/out” feature would not be able to log in to the system in order to place an order.

- **Analytics:** The analytics view will consist of a chart representing the ten most sold items for the day. This chart dynamically changes based on placed orders.
  The second analytics data required by this view is a bar chart comparing the revenue for each day for the last two weeks. It will include a comparison of a "today's revenue" versus "last week” same day revenue.

  Even though those are the only planned analytics, they will play a big role in providing managers with data that is crucial for the business. It will allow them to take decisions such as whether or not to keep certain products on sale, or which days of the week are the busies and which are the quietest.

  In addition, the view will incorporate a detailed listing of all the orders placed so far providing the user with isolated data about a particular order.

- **Ingredient tracking:** allPOS would be able to build items out of ingredients. In this manner, hospitality businesses would be able to track their menu and the specific ingredient that they need. The stock system stores items, which could be either build of ingredients or “solo items” meaning that they are already a product.

- **Permissions:** Three permission levels would be available:

    - Employee: employee permissions allow users to take orders, take payments, view only the 10 most sold items and view only their own

schedule. In addition, they are able to see only the low on stock items from the stock management system.

- Supervisor: supervisors have access to the “employee” permissions. Moreover, they are able to view sales analytics, generate refunds, view all employees schedule and clock in/out timestamp history. Supervisors can also create employee and supplier records, but are not allowed to edit existing data.

- Manager: manager have the likewise admin permission. It gives them user access to “employee” and “supervisor” permissions. Furthermore, manager permission allow the user to schedule employees rota, change employees pay-rate, add/remove/delete items from the stock, initialise end of the day reports and create users of all permission levels. Manager are able to enable/disable automation ordering and change its relating settings.

3.3. Non-Functional Requirements

- **Compatibility**: allPOS should be available to any device that has a browser and an internet connection. Furthermore, the web app should be responsive for devices with smaller resolution. Despite the device used, it would have to behave in the same way across several devices. A PC, iPhone, Android phone, Android tablet will be used to test the compatibility and responsiveness of the app.

- **Reusability**: Django applications consist of several apps. Django app is a small library that represents an independent part from a larger project. Moreover, apps could be reused in another project.

- **Security**: All user data should be stored in an encrypted format. In addition, form inputs should be salted and sanitised to avoid injection attacks. Cookies should be inaccessible to JavaScript, HTTP only. Furthermore, a Cross-Site Request Forgery protection across all the forms in the website should be applied. Avoid session hijacking by storing user’s IP address in session variables. Implement an HTTPS to prevent wiretapping and man-in-the-middle attacks.

- **Efficiency**: The system should take no more than 5 seconds to render different pages, update, add or remove any data from the database.
- **Availability**: The system must be available at list 99% of the time. If there is any maintenance that needs to be conducted, all customers must

be informed. In addition, maintenance should be led in the early morning of the first day of the week.

3.4. Use-Case Examples

This section will introduce the key use-cases of the application shown in Figure 3-1, 3-2, 3-3 and 3-4.

- **allPOS:** The use-case from figure 3-1 represents one of the most critical component for users to use allPOS system – they need to be registered first. It can be seen that a Manager can register all the employees that a supervisor can register. Additionally, only a manager can register another manager.

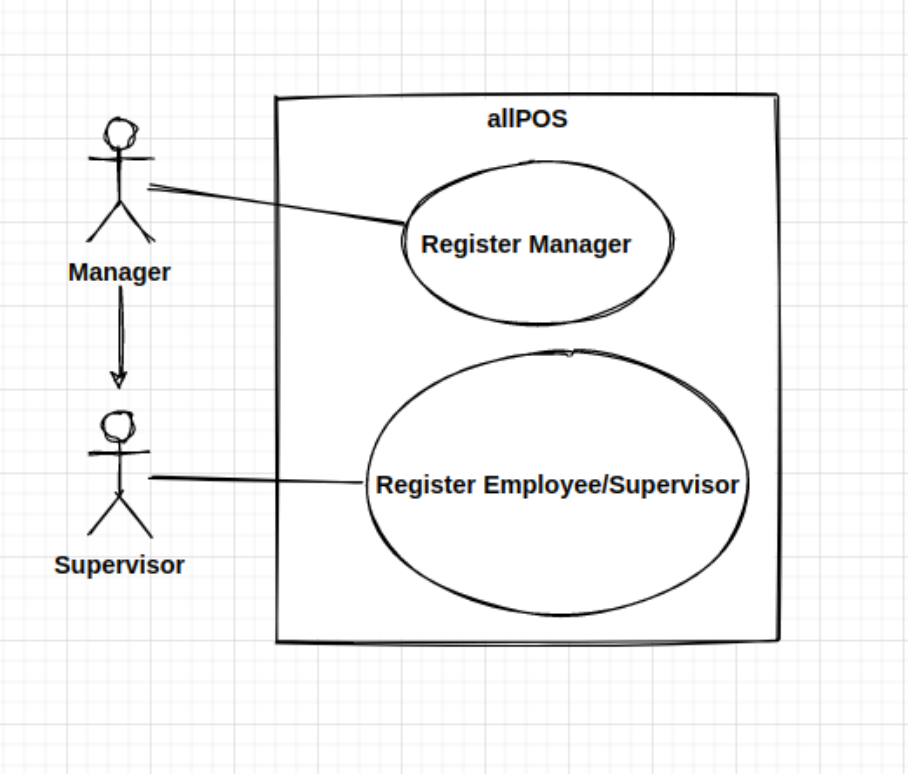
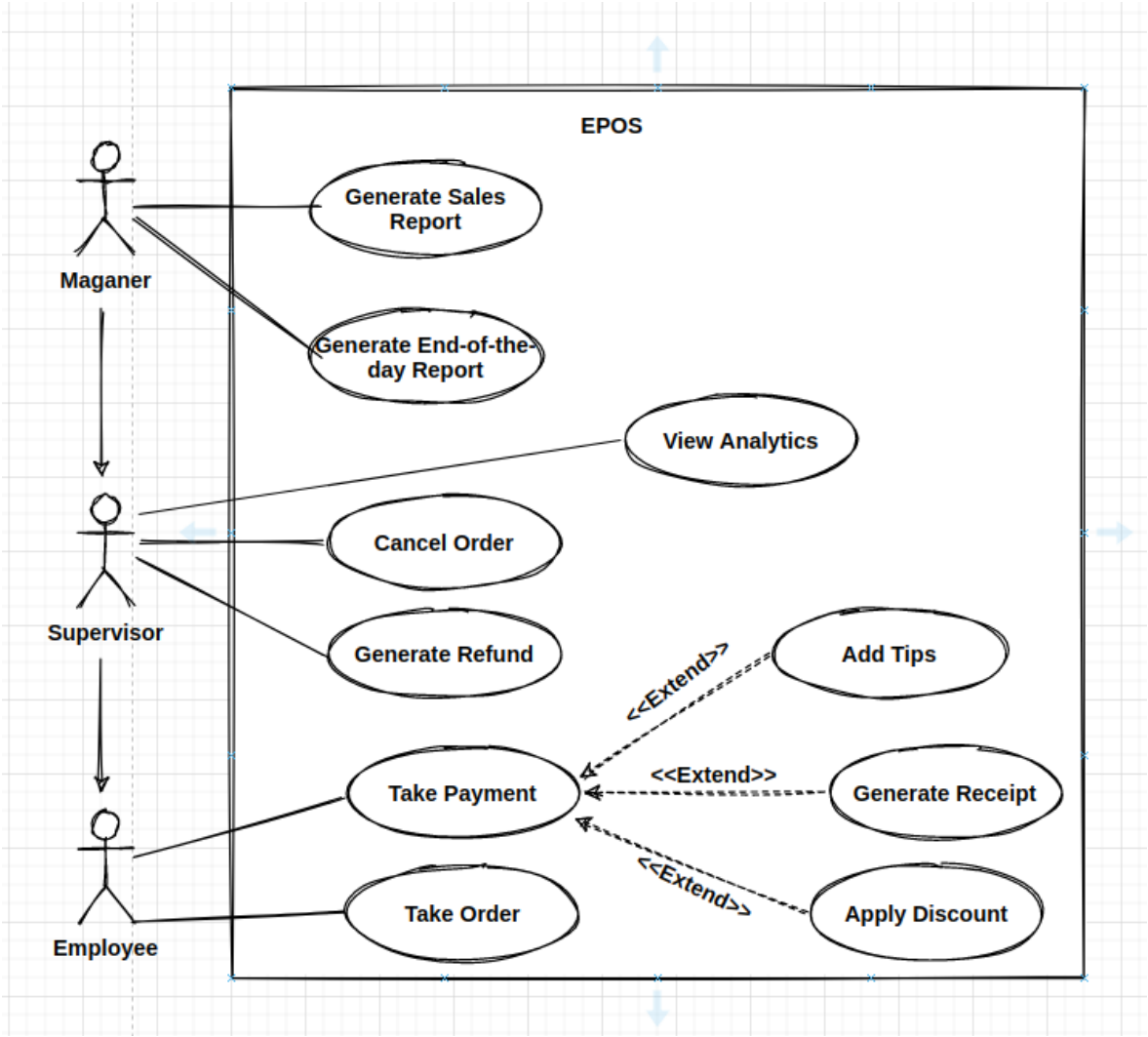


Figure 3-1: allPOS Use-Case

- **EPOS:** Figure 3-2 illustrates the most used component of allPOS. This is where employees take customer orders, add items, generate bills and take payments. Employees can take orders and take payments. Additionally, they have the extended functionalities of adding tips, apply discounts and to generate receipts. Supervisors can do all the task an employee can do. Additionally, they can refund customers from the payment screen, cancel an order and view weekly and daily sales analytics. Sales analytics would be placed on the main screen menu, but it would show up depending on the access level. Managers can perform

all tasks a supervisor and an employee can. Furthermore, they can generate end-of-the-day sales report.



- **Employee Schedule:** Figure 3-3 represent the employee schedule use-case. It can be seen that an employee will only be able to see its own schedule for the day. The daily schedule will include time-started, how many hours an employee has worked so far, and finishing time. Supervisors inherit all the tasks employees can perform. Furthermore, supervisors would be able to view all the employees that are working today, including, starting-time, finishing-time, hours generated so far. Managers, on the other hand, would be able to create a schedule. Managers would be able to edit schedules, remove schedules, input holiday and day off requests. Moreover, they will be able to change employees pay rate and generate a report for an individual, or all employees worked on a particular day. The report will include the cost of having this employee and the net profit the employee-generated over the requested time frame.

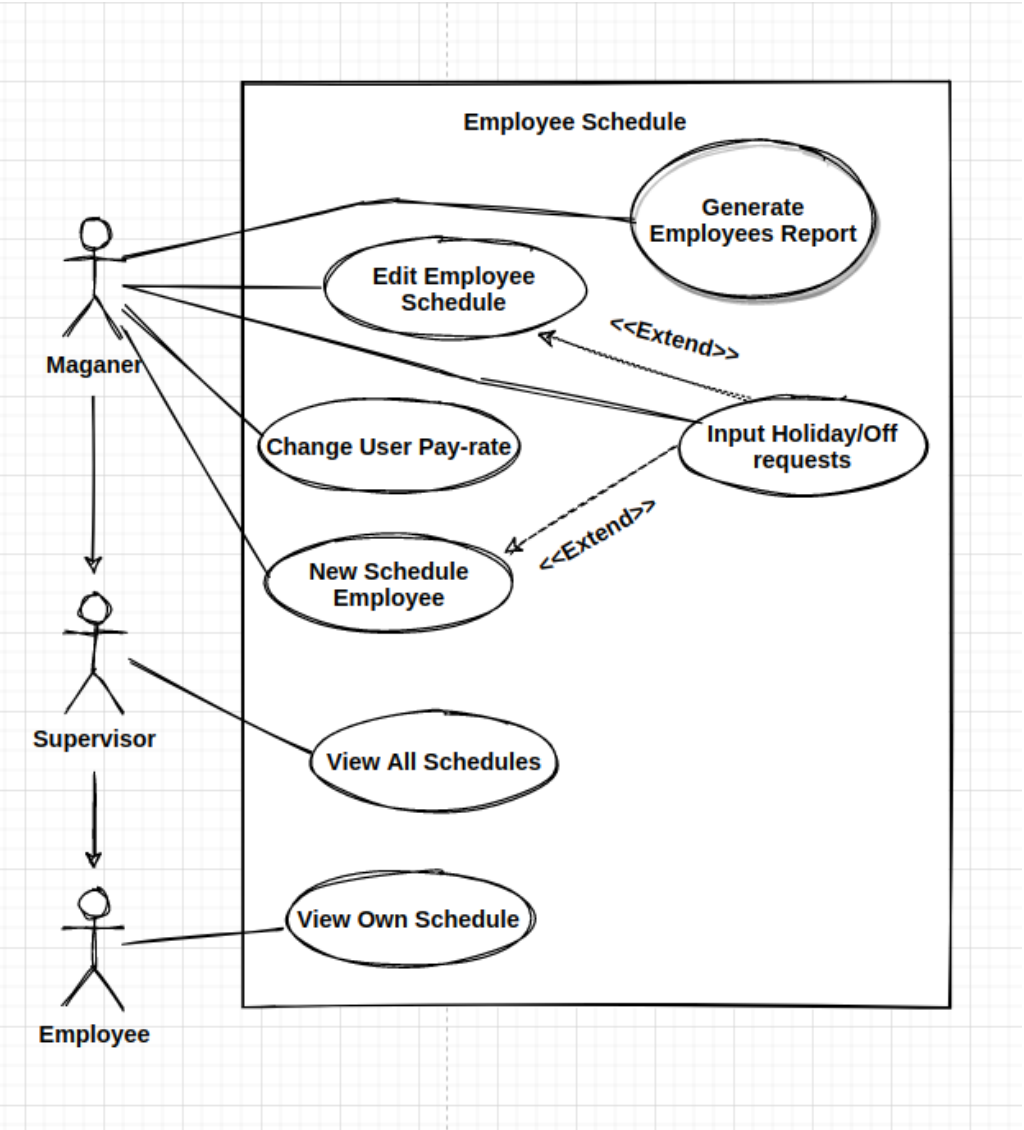


Figure 3-3: Employee Schedule Use-Case

- **Stock Control:** Figure 3-4 illustrate the Stock Control Use-Case. Employees can only see the items that are low on stock. In this way, they would be able to advise a supervisor or a manager. Supervisors, however, would be able only to view all stock items. Managers will have to add a supplier's account and items in stock. Once the automation order is activated, the module will regularly require information from the supplier's database accounts and items on stock. Once the module detects items low on stock will generate a list of items that need to be ordered. Once the list is generated, it will generate emails and send then to the suppliers and notify managers (if notification settings is on).

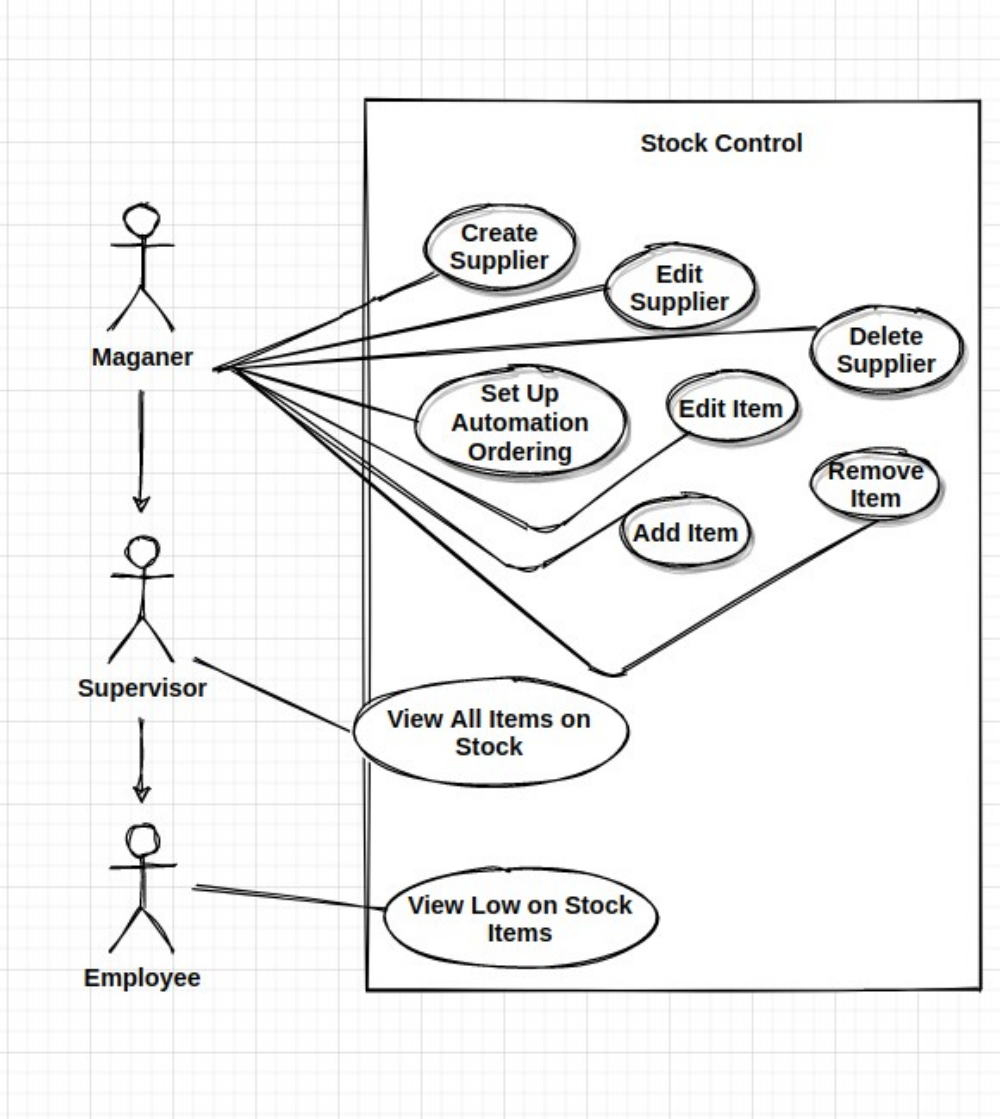


Figure 3-4: Stock Control Use-Case

3.5. Summary

The functional and non-functional requirements were described in this chapter. Functional remeasurements were divided in primarily and secondary requirements. The application functionality is not dependent on the non-functional requirements but those are required for the user experience.
The chapter concludes with the main use cases of the apps.

4. Design and Implementation

The fourth chapter contains a detailed description of the implementation of the three main components in allPOS and the software's high-level design. In this way, the reader will become acquainted with the project's critical elements and will be able to comprehend how each aspect of the software contributes to the overall application functionality. Moreover, the chapter would include an overview of the graphical user interface, and an application working flow would be provided for the most crucial features of the system.

4.1. Implementation Environment

Django implementation environment is simple to set up and get started. Once the framework is installed on the local machine, it is generally used for software developing and testing. 'The main tools that Django itself provides are a set of Python scripts for creating and working with Django projects, along with a simple development web-server that we can use to test local (i.e. on your computer, not on an external web server) Django web applications on your computer's web browser.' [10]
The IDE of my choice was a Visual Studio Code in addition to the following extensions:

- Django Template
- Django - beautiful syntax
- Djaneiro - Django Snippets
- Pylance
- Kite AutoComplete

4.2. High-level overview of MVC implementation

Section 2.1.3. briefly introduced the Model View Controller architecture of Django. However, this section will provide detailed information about how the MVC pattern fits into this project's technical requirements.
Considering all the entities and computations involved in this project (accounting, stock management, employee scheduling, record keeping, etc.), a great way to separate each of those concerns is indeed the MVC pattern. Using the MVC, I was able to work on different "levels" of the applications without worrying about the other "levels". For instance, we can write code for the back-end with the MVC pattern, which will be independent and testable without the front-end code. This property is due to the "loosely-coupled" architecture of the MVC. The fact that the different elements of a Django application are loosely coupled means that there is no strict dependency between them.

4.2.1. Model

Models represent physical or logical entities of the real world. They are an abstraction of a database tables and each model consist of different attributes.

In addition, some model attributes represent a relation connecting it to other models. Models are modified by the controller, which can create, edit or delete model instances. Models such as, "Order", "Product", "Employee" are an example of real world physical entities, whereas, "InventoryIngredientTransaction" represents a logical entity. Figure 4-1 illustrates the high level class diagram of the application models. For the full data model diagram, please refer to page 55, Figure D: Database Design Diagram

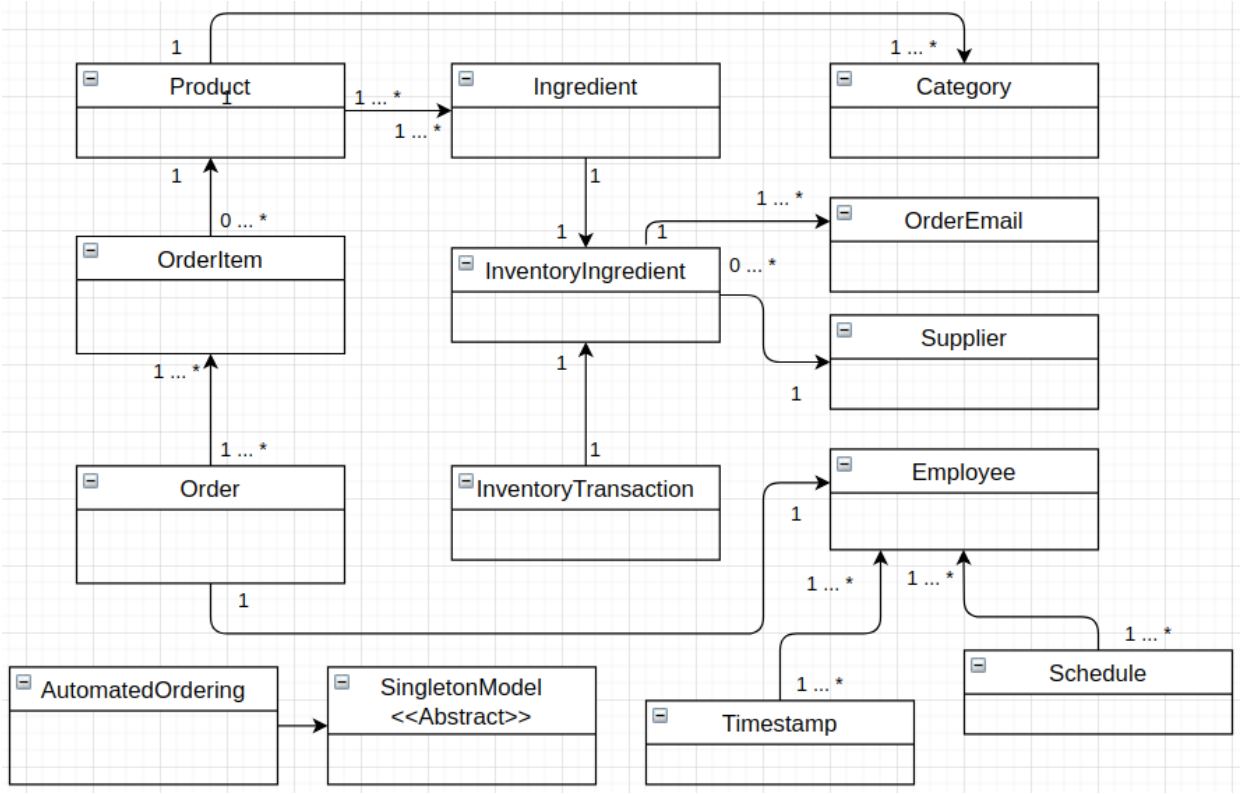


Figure 4-1: High-level class diagram

4.2.2. View

Views are a vital component of this project. Django represents HTML documents as templates that enable to separates logic and code from design. Templates render the front end of the application, allowing the user to enter, change, edit, delete or view instances of models. Moreover, regardless of the user command, the template will obtain some inputs: data or a model modification request and passes them on to the controller. Templates in Django also have a special syntax. The controller sends some database models to the templates, and using this special syntax, the template can render database data dynamically which can be seen in figure 4-2.
Furthermore, figure 4-2 represents another property of templates. Templates have a hierarchy enabling components reusability among different web pages. For instance, a “base.html” document implements all the standard components, scripts and styling that would be repeated among all the web

pages. Examples of which are background colour, navigation bar, footer, JavaScript code. Base.html would inherit all other pages of the website, allowing them to reuse the components that are in the file. Figure 4-3 shows the template hierarchy aspect of the Django views.

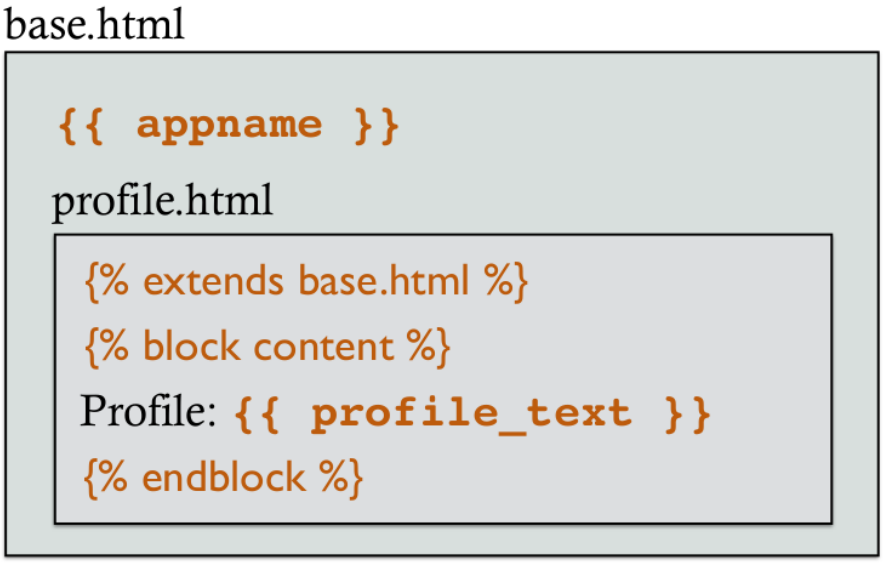


Figure 4-2: Template syntax and template hierarchy [13]

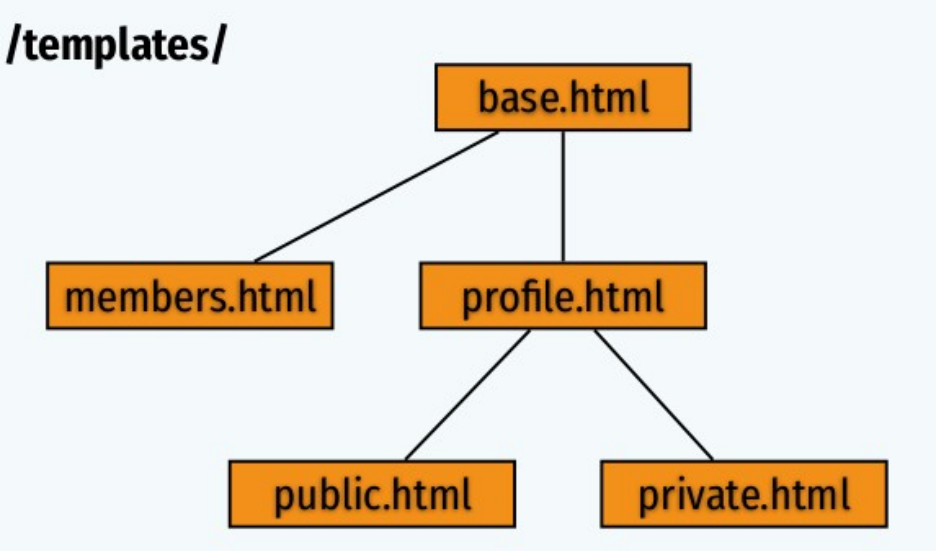


Figure 4-3: Template hierarchy [13]

4.2.3. Controller

The controllers are the "engine" of the application and represent all the application's business logic. The project's general three features (epos, employee scheduling, stock management) are implemented in three different controllers. For instance, the epos system controller is implemented in the epos app under the name views.py. It consists of methods that form the back-end

side of the epos application. It includes order creation, orders listing, stock take and many more.
By implementing different controllers, I managed to separate components for loose coupling and allow reusability of a specific feature. For instance, an epos system could be implemented on its own, independently from the stock management and employee scheduling components. Figure 4-4 illustrates the separation of the project's three main functionalities, which would also provide reusability and independency of each of those parts.

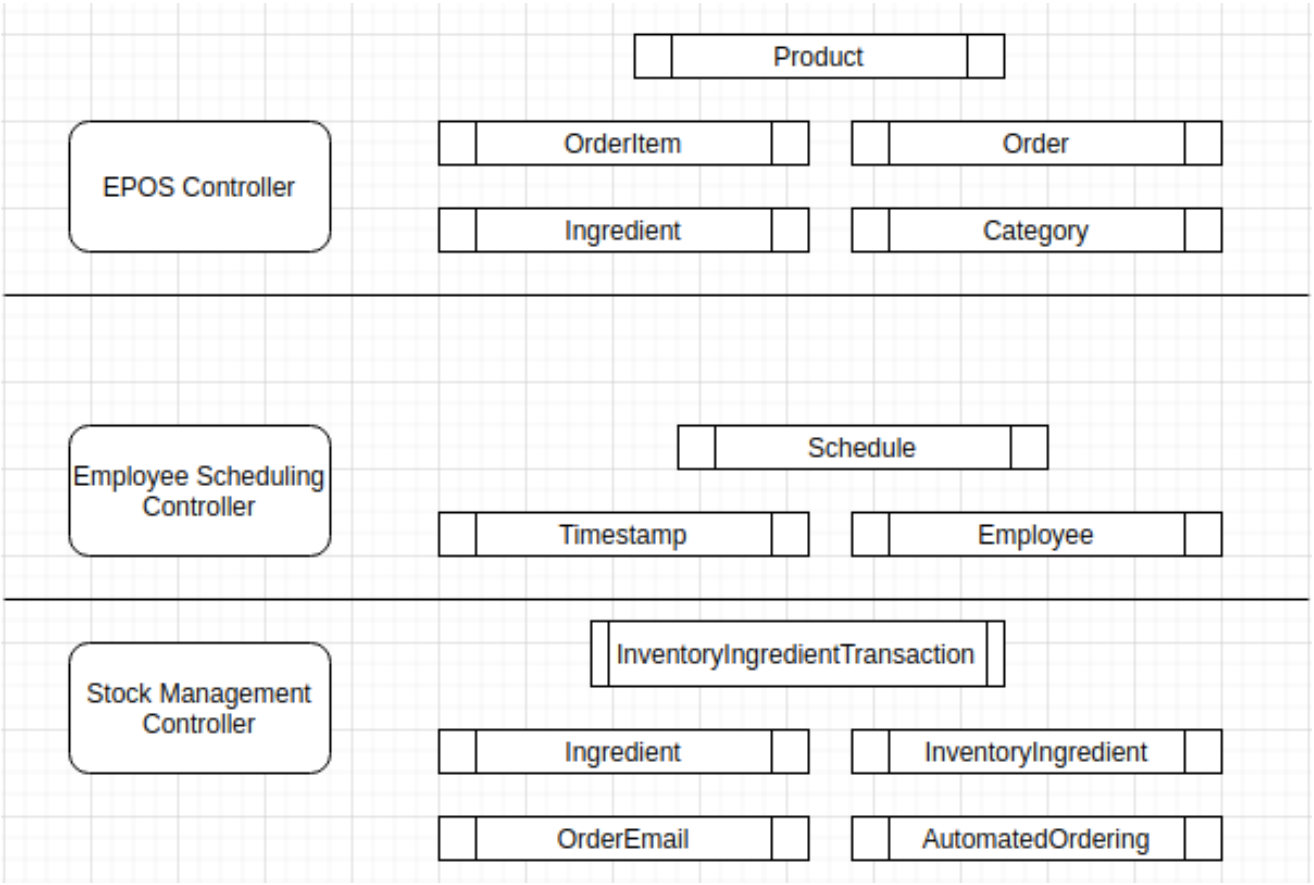


Figure 4-4: Separation of Functionalities

4.3. Low level design

This subsection would be introducing my code implementation decisions. I will go through the different parts of the application (database models, back-end, front-end) and explain specific functionalities.

4.3.1. Model

Models in Django define a piece of single specific information about the data that maps to a table in the database. This data is implemented as a Python class that has attributes and typical behaviour.

- **Employee:** This model represents an employee of the company. The class contains employee personal details and some company-related information. The model also separates employees into different position categories - manager, supervisors and employees. Based on the employee's category, the user has different access to the application features.

- **Timestamp:** When an **Employee** start or end their shift, they must "clock in/out" with their pin to record the starting/ending time and the date. **Timestamp** class represents the "clock in/out" of the **Employee**. Wage calculations, reports, and employee analytics functionalities are using **Timestamp** data for their generation.

- **Schedule:** This is a logical entity representing the shift of an **Employee**. Schedules contain **Employee**, time and date. On creating the weekly staff rota, the manager produces instances of the **Schedule** model in the database for each Employee representing their working days. The view for rendering the weekly rota uses **Schedule**s instances, listing each Employee and their working days and days off.

- **InventoryTransaction:** This class records data relative to the action of taking or adding a specific amount from/to an **InventoryIngredient**. The transaction holds details for the **InventoryIngredient** taken/stored, its quantity, date, time and reason for the transaction. For instance, reasons such as: used for an order, wastage, added to stock, removed for the stock. In this way, managers would be able to monitor ingredient levels and **InventoryIngredient** usage**.**

- **InventoryIngredient:** This model stores the total amount in stock for each ingredient. When the company sells a product, for instance, still water, and the stock contains 20kg of still water, considering that a water bottle is 0.050gr, the **InventoryIgredient** model's total amount of the still water would decrease by 0.050.

- **Product:** This class represents a product that the store sells. For instance, a product could be coffee. Besides, a single product instance of the model contains one to many **OrderItem**. One benefit of this design is that the product's cost price is the sum of each used OrderItem times its quantity. For instance, if 1kg of coffee beans is £15, the cost for the coffee needed in a double espresso, which contains 0.018 grams of coffee, is 15*0.018. This kind of data is used for analytics, transaction logs and automated ordering.

- **Ingredient:** This represents a specific amount of the **InventoryIngredient** model and it has one-to-many relationship. With this database design, the system can use the same ingredient for different products with a different amount. Moreover, the managers would keep track of a specific ingredient's current stock, monitor all of the ingredient transactions, and have a system that eventually could place an order for ingredients automatically to a supplier based on some previous information.

- **OrderItem:** This is an abstraction of a **Product** model. It counts the number of the same **Product**s added to an order.

- **Order:** This model represents an order which holds information such as **OrderItem's**, type of the order, payment method, date, time, the **Employee** who took the order, the total amount of the order and the payment status.

- **Category:** A good way of separating products is by sorting the products into categories. In this way, the EPOS GUI can list only categories needed for the order and hide the unnecessary ones. For instance, the "hot drinks" category contains coffee, tea, hot chocolate and any other hot drink. If the customer pops in the store for tea, the Employee can easily find all the hot drinks by pressing the hot drinks categories, making the whole process much more intuitive and agile.

- **Supplier:** By storing supplier information as a class, the system can easily make raw goods ordering. Besides, the stock management tool would have an automation raw goods ordering functionality that will be using this class to order an ingredient that is low on stock.

- **AutomatedOrdering**: this model inherits from a **SingletonModel** model. This design decision has been taken in order to model the settings functionality of the project in a single model that would have only one instance and would not allow the creation of a second instance. It holds the following settings for the automation-ordering process:

    - Enable/Disable automation-ordering

    - Enable/Disable email confirmation – if enabled, once an order has been sent if could notify all the managers, for the placed orders via email

    - Enable/Disable logs of the placed orders – if enabled, it save an instance of a **OrderEmail** model in the database keeping details such as: date and time of the sent order, sending recipient, subject of the email and the email body text. In this way managers are able to check all the emails that where sent to suppliers for a certain period of time

- Email Subject – users can create a subject template that would be used by the email sending cron job to generate the subject of the order emails

- Email Greetings – users can create a greetings template that would be used by the email sending cron job to generate the text before the list of ingredients that need to be ordered

- Email Footer - users can create a footer template that would be used by the email sending cron job to generate the text after the list of ingredients that need to be ordered

4.3.2. View

In Django, views are the HTML documents or also called "Templates". Views are the front-end part of the application. Views implement several functionalities that help the user interact with the application's data and do some calculations. Views contain images, text, inputs fields, buttons and any other UI component that is either statically or dynamically rendered. Those are the main views implemented in allPOS:

- **base** - as discussed in section 4.2.2, Django features a Template Hierarchy. Base.html represent the "super" document of this hierarchy, meaning that all other documents (HTML pages) of allPOS inherit from "base.html". The document contains JavaScript files, CSS files, and the implementation of a navigation bar which are automatically inherited by all web pages that are a child of "base.html". This powerful feature of Django allows us to reuse code and avoid repetitions.

- **Index** - when an employee (user) logs in to the application, this page will be displayed. It implements the electronic point of sale functionality. It is composed of product buttons, product categories buttons, the name of the logged-in employee, date, time, order type (have-in or take-away), order number, products added to the order and their quantity, total order amount, discount, tax, and buttons for holding the order, cancelling the order and sending the order. This webpage is an essential part of the application.

- **analytics** - this webpage provides sales analytics. It contains two main charts, the ability to generate the end of the day report and data related to all orders made so far, where orders are listed based on the date. The most recent orders are listed first. User can also check the order details of each order. The first chart is a pie chart. This pie chart only illustrates the ten most sold items for today's date. The second chart is a bar chart. The bar chart compares the current week revenue with the previous week revenue in addition to today's revenue and "last weeks same day revenue."

- **modals** - ‘The Modal Element can be used to display additional content via a pop-up window on a page. Modal boxes are hidden by default and can only be triggered by a menu item, a button element or a modal text / HTML link element, which can be either text or an image. ‘[12]. allPOS consists of several modals for different implementation and organisational purposes. This file holds all the modal components used in the web app.

- **rota** - this view renders a weekly calendar with all the employees and their rota. A manager can use a rota view to check if an employee works on a particular day or it is off. User can also navigate to four different views:
    - **today's schedule** - lists all the employees working today, date, starting-finishing time, clock in/out time.

    - **update rota** - a view that enables managers to edit, delete or create schedules.

    - **timestamp history** - list all timestamp history

    - **employee reports** - generate employee reports

- **settings** - this is the web app settings menu that allows users to edit VAT percentage, list, edit, delete, create employees and suppliers. The automation ordering settings menu is also included in this view. It provides the functionality of enabling/disabling the automation ordering, enabling/disabling notifications of the automation ordering, enabling/disabling the automation ordering logs creation, and edit templates for the generated emails.

- **inventory_ingredients –** represents the stock management system. It allows users to monitor current inventory ingredients status as well as editing, creating and deleting inventory ingredients. Moreover, it has an additional sub-menus for creating, editing, deleting ingredients and products. Besides, users can also see all the inventory ingredients transactions.

4.3.3. Controller

The controller consists of three main parts. Each of those parts is liable for one of the three principle functionalities - EPOS controller, employee scheduling controller and stock administration controller. We can distinguish two types of functions implemented within the controllers - a view function and an API function.

The view functions take some request and return a response, including an HTML document with database data. When a view function is called, the program refreshes its page.

On the other hand, Ajax utilizes the API functions used to asynchronously get, alter, erase, or create new back-end information. When an Ajax request is sent to the server, the browser doesn't refresh the current webpage, and it gives a feeling of a native application. The server sends back a response consisting only of some message or data, but not with an HTML document as in the view functions.

- **EPOS** - The EPOS controller is the business logic related to the electronic point of sale system. Those are the most important functions behind the epos controller:
    - **home_view**: it queries all the products, categories, and the user currently logged in and passes the data to the index view.
    - **analytics_view**: this method is responsible for getting the data for both charts - "10 most sold items" pie chart and "14 days revenue" bar charts and send it to the analytics view.
    - **get_product_API**: This function is the code responsible for listing products based on categories. Every time a user presses a category button, it requests this function and the function then passes the products to the front-end.
    - **create_order_API:** This function gets all the products from the order, the total amount, the employee currently signed-in, type of order, order number, time and data, payment type, and creates an order model instance in the database. Moreover, it deduces the amount of ingredient from the stock database used for each product in the order by using an **IngrientOrderTransaction** with the reason - "order: Order number".
    - **get_order_API**: An Ajax call that queries a specific order based on the primary key passed as an argument to the function. It is then passed to the front-end to render all the details of the queried order.
    - **get_orders_list_API**: It is used so a user can review all the existing orders in the database.
- **Employee Schedule** - Controller responsible for all the rota related operations. The essential functions of the controller are:
    - **todays_schedule_view -** This method queries all working employees' shifts "today" and passes the data to today's schedule view.

- **get_rota_view** - Query weekly rota with all employees and their working time/days off and pass it to the rota view.

- **update_rota_view** - The methods allow the manager to modify the rota of each employee for a given week. The allowed modifications are edit, create new, delete. It calls the following Ajax methods: **create_schedule_API, delete_schedule_API, update_schedule_API.**

- **get_schedule_for_rota_API** - When a manager wants to access different weekly rota for some modification, the front-end calls this function, and the function returns the weekly rota for the requested week.

- **Stock Management –** controller is where all the inventory ingredients are listed or altered. Some of its most important view and API functions:

    - **stock_view –** This method queries all inventory ingredients and passes the data to the inventory_ingredients view

    - **ingredient_view** - This method queries all ingredients and passes the data to the inventory_ingredients view

    - **stock_create_API, stock_delete_API, stock_update_API –** those API functions are responsible for the alteration of the inventory_ingredient model.

    - **ingredient_create_API, ingredient_delete_API, ingredient_update_API –** those API functions are responsible for the alteration of the ingredient model.

    - **product_create_API, product_delete_API, product_update_API –** functions are used for altering the products model.

    - **inventory_transactions_view –** returns a list of all inventory transactions

    - **get_automated_ordering_API –** is a method that returns asynchronously all the automated ordering settings currently used.

    - **update_automated_ordering_settings_API –** used for changing the automated ordering settings.

4.3.4. Custom Back-End Authentication

Django implements a user authentication, which is sufficient for most use cases. Nonetheless, allPOS executes a custom user authentication with an extended back-end that permits clients (employees) to authenticate only with a pin code. At the point when a client is registered within the framework, they get a unique PIN code which is utilized in two use cases:

- The client can Clock-In/Out creating a timestamp in the system
- The client can rapidly sign into the framework when a customer order should be taken, assuming they have already clocked-in.

The custom back end abrogates the default validation, permitting clients to authenticate only with a PIN code instead to username and password.
The purpose of the design is to empower speedy admittance to all the framework functionalities in a second. Then again, authenticating a user with a username and password would defer the store's working flow each time a worker needs speedy admittance to the application.
In terms of security, allPOS is designed to be used only internally within an organization. Hence, pin authentication does not compromise the application's security. Moreover, an HTTP request would live only within the organization's network, whether the network has an internet connection or not. Consequently, attacks like "Packet sniffing" and "Brute-force" are not applicable.

4.3.4. Django Crone-tab

A library called "Django-crontab" was used in this project. This library represents powered job scheduling for Django, also known as cron jobs.

What crons allow us to do is to run commands or scripts periodically at times.

We can specify how often we want to run specific jobs in the Django settings.py file. Moreover, this allows us to run different jobs at different times independent of each other.
For instance, if we want to periodically send emails to employees reminding them of their upcoming shifts, a crons.py file would be implemented in the employee scheduling app containing the code for the "shift reminding" functionality.

The following implementation has been followed to accomplish automated ordering of stock raw good:

- Every day at 4 pm a job called "my_scheduled_job" is executed. The job calls the service "automated_ordering_service" which is in charger of the automation ordering

- The service first checks if the automated ordering is enabled, if it is disabled, the execution stops and the same job is executed on the following day

- Assuming the ordering is enabled, the service collects all the ingredients that have "current stock" less than the "minimum amount needed" attribute of the inventory ingredient model.

- Then, the list with ingredients is sent to a service that groups all the ingredients based on their supplier.

- After the grouping has finished, the service generates n number of emails where n is the number of suppliers for the needed ingredients.

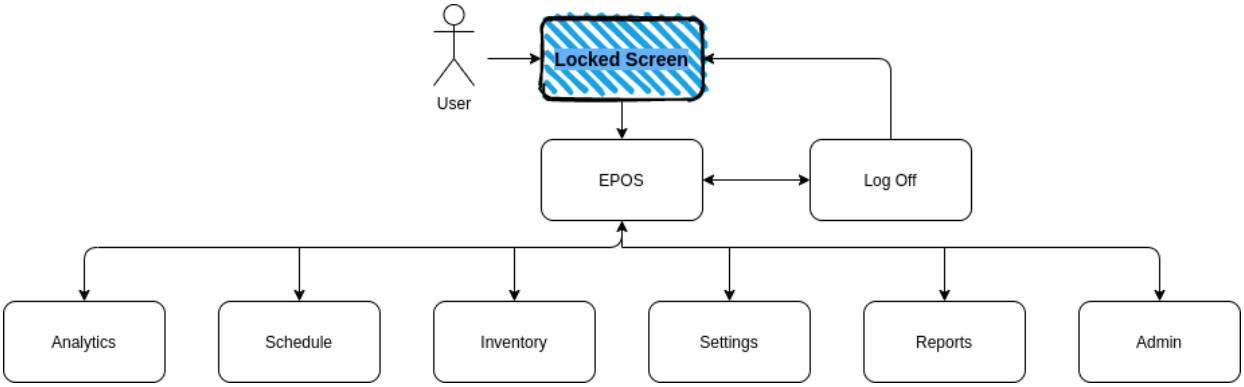
- Then the sending starts and orders for stock ingredients are placed

- The second last job of the service is to check whether the automation ordering settings have "manager notification" enabled. If it has, it sends a notification email to all managers notifying them that orders for the missing ingredients have been placed, including a copy of the sent emails.

- If the automation ordering settings have "orders logs" enabled then it finishes its job, otherwise deletes all the generated email objects

4.4. High-Level Application Flow

The initial screen illustrated in figure 4-5 is the locked screen. This screen provides the employee with two functionalities. A user can log in only if it has been clocked in for the day. Otherwise, an error message is shown indicating that the user must clock in first. At the end of a shift, users must clock out. Once employees are clocked out for the day, they cannot log in. When the app is not in use for a minute, it automatically signs out the logged user, and it redirects to the locked screen.



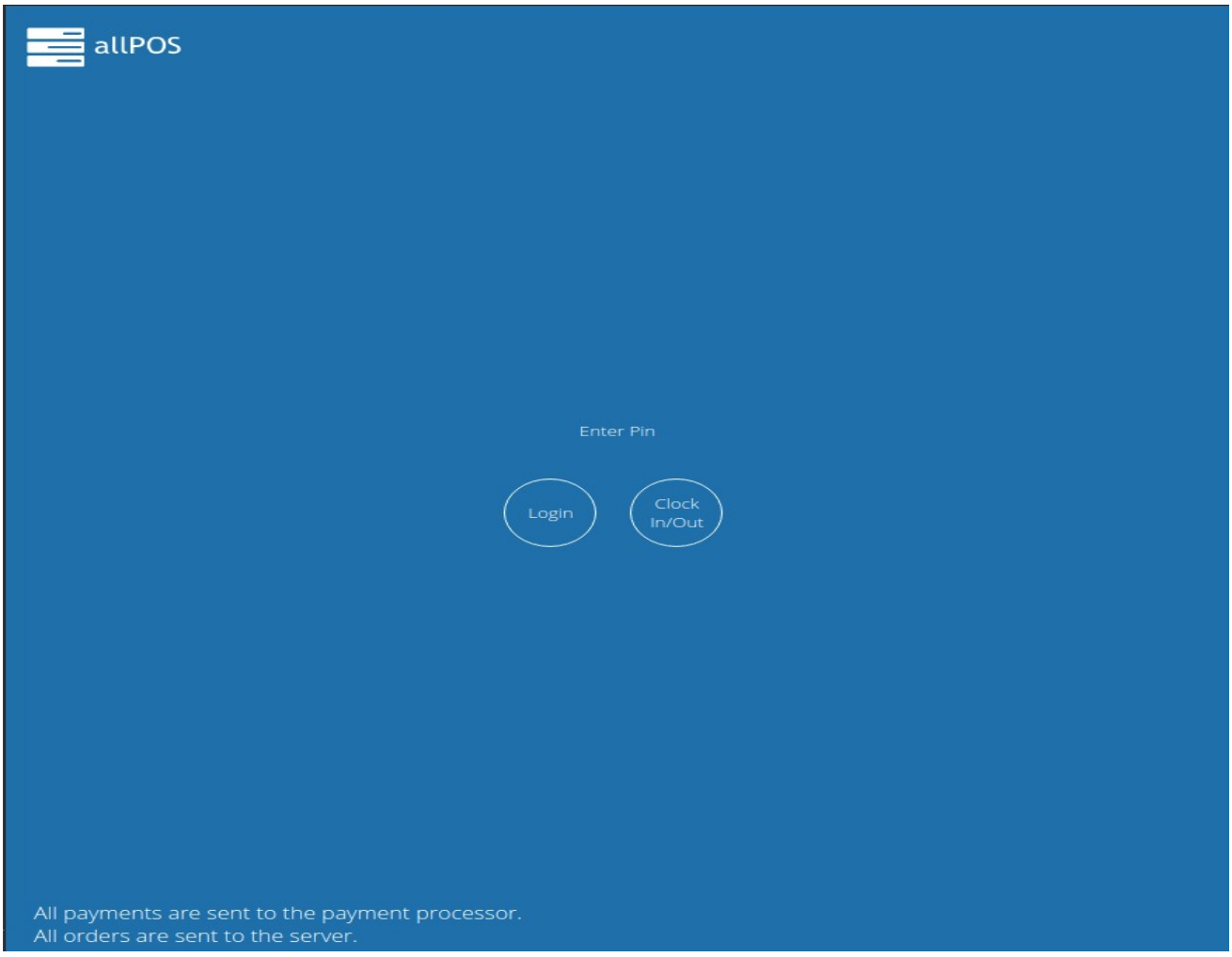


Figure 4-5: Locked screen

On user authentication (when an employee enters its pin), the user is diverted to the application's main page. The electronic point of sale (EPOS) system provides the core functionality of the project. It interconnects many models and supports specific data alteration for analytics and additions system decisions. This screen represents all the categories and products the venue offers. A simple and intuitive interface has been executed so the user can rapidly take orders and make changes. Categories are horizontally scrollable, and products are dynamically changed on a changed category. This view is accessible by all kind of employees – managers, supervisors and a basic employee

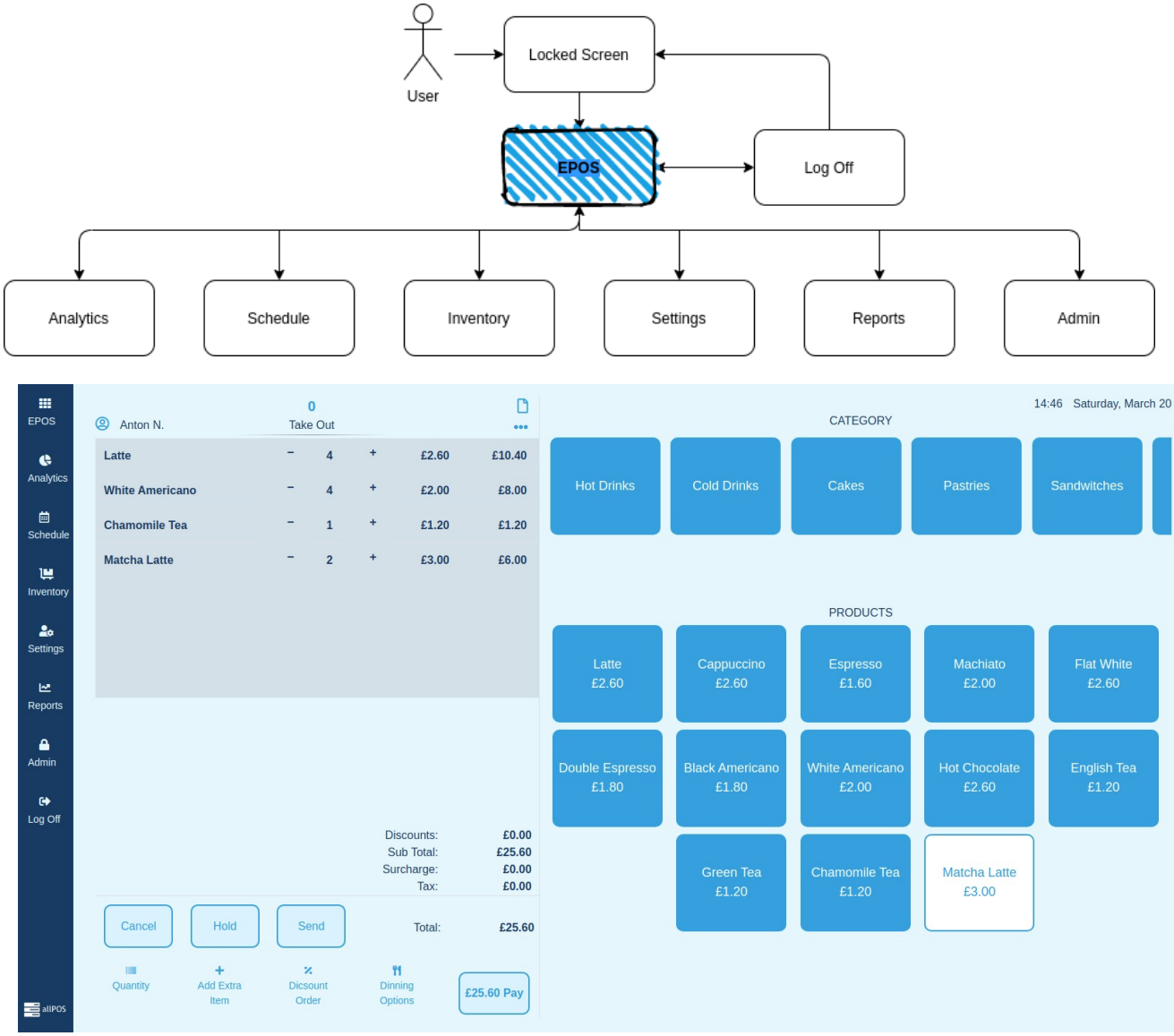
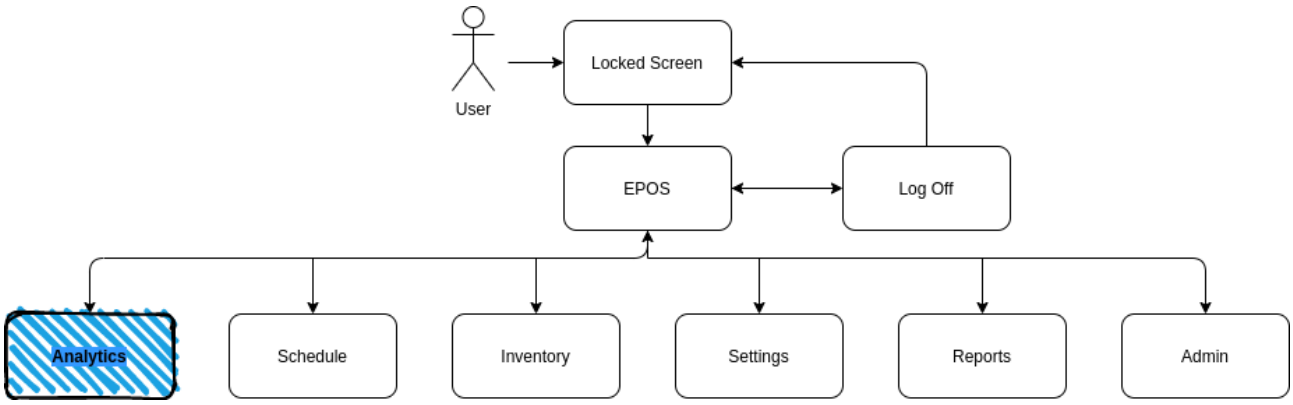


Figure 4-6: EPOS screen

Assuming the logged user is a manager (managers have full access to the application), they can look at different revenue and sold items analytics. Figure 4-7 represents the analytics view with some other standard functionalities - view all orders, new order, catering, delivery management, appointments, invoices and shipping. In addition, user can initialise the end of the day process, see tips and on hold payments. The basic employee account permission has only access to the top 10 sold products. A supervisor does not have access to catering, delivery management, appointments, invoices and shipping.



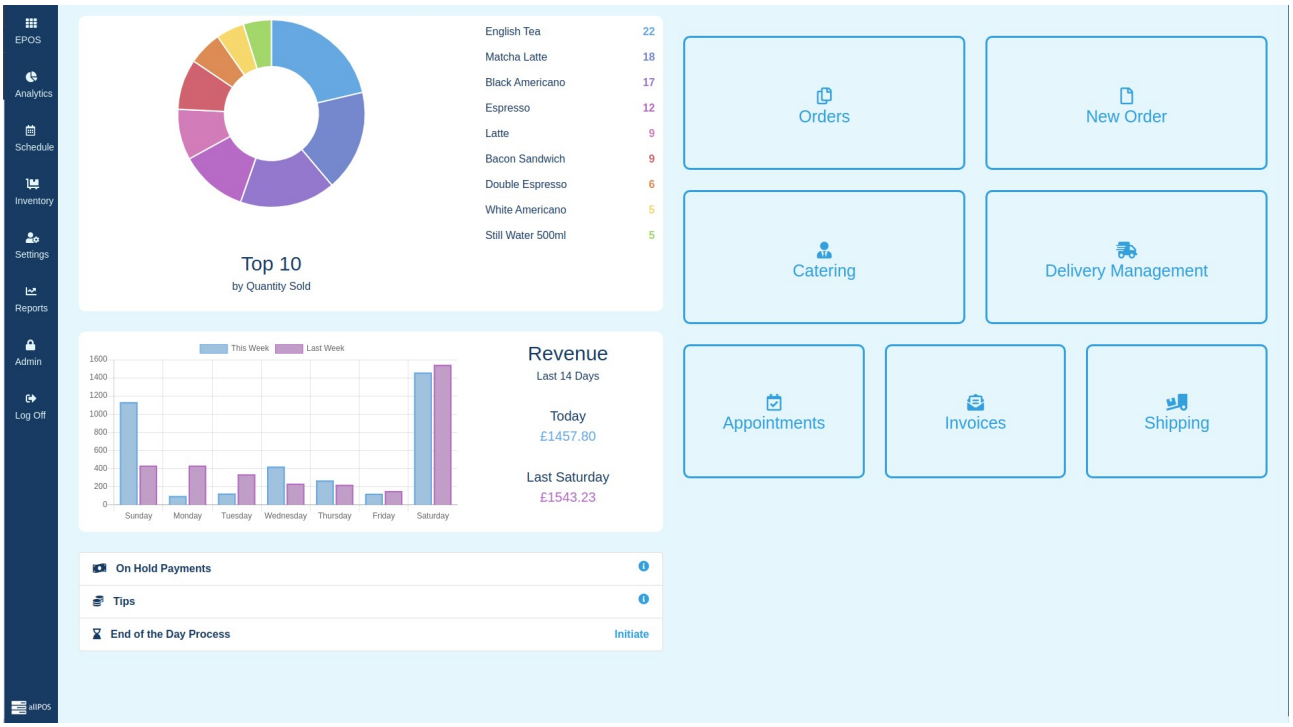
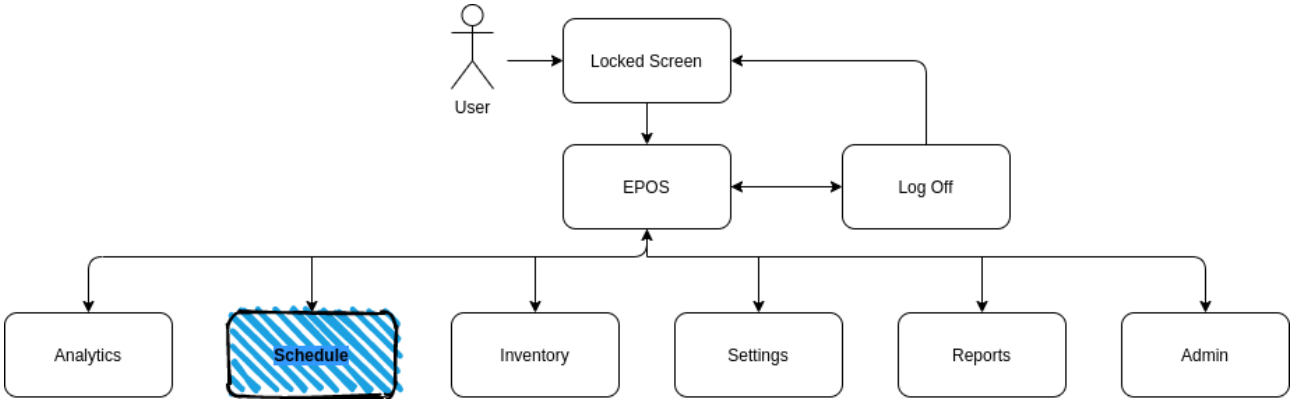


Figure 4-7: Analytics screen

The next figure list all the employees that are on shift today. In this manner, managers can check at what time a particular employee has started or finished. Moreover, it provides their actual starting and ending hours in addition to the functionalities of viewing the weekly rota, updating employees schedules, employee timestamp history and generating employee reports. Supervisors have a limiting access to the Employee Schedule functionalities. They can see only weekly rota calendar, timestamp history of employees and the “today’s schedule”. Basic employee are only able to see the weekly rota and their own “today’s schedule”.



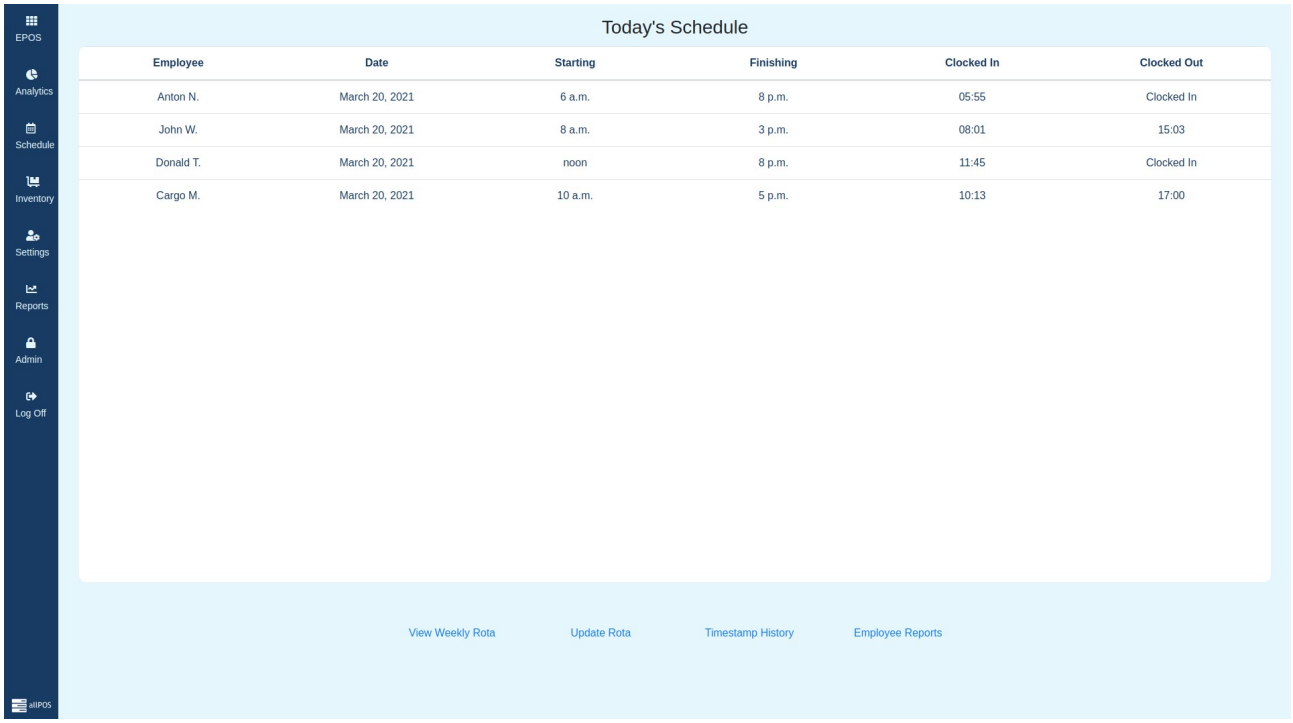
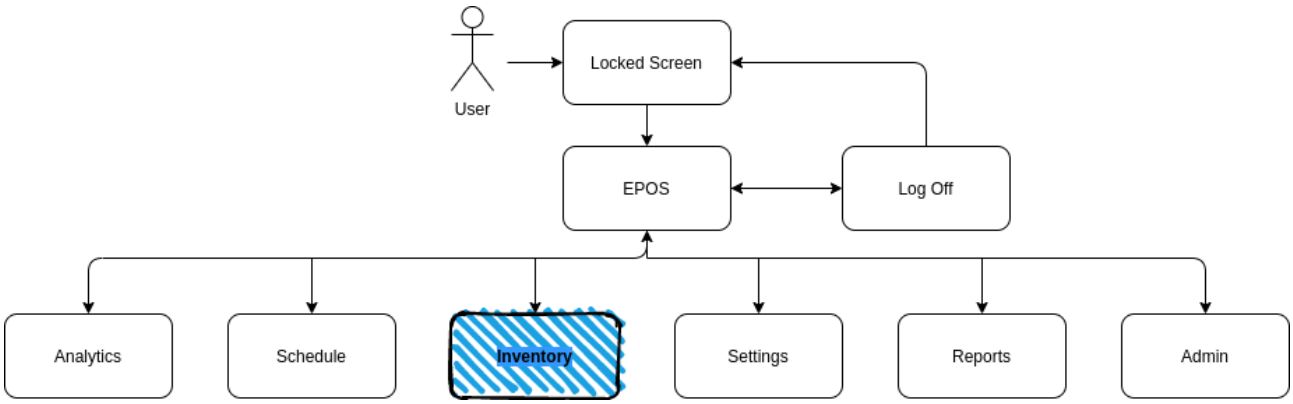
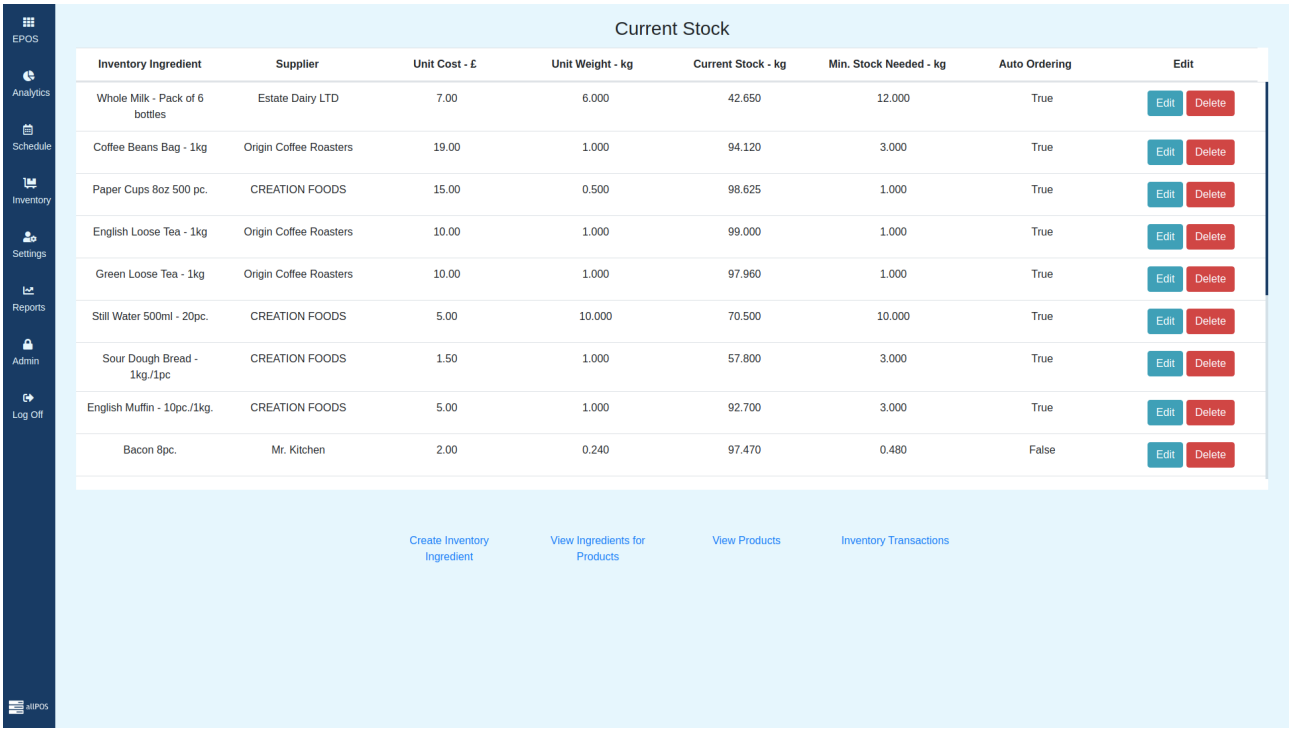


Figure 4-8: Schedule screen

The stock management functionality allows managers to monitor what is happening with the ingredients in the stock. It logs all the ingredients transactions, meaning that a user is able to see how a specific ingredient quantity has changed over a time period including its usage and current stock supply.
In addition, managers can create, delete, edit ingredients, products and inventory ingredients through this application. Basic employees are only able to see ingredients that are low on stock so they can notify their manager or supervisee but do not see any buttons or sub-menus. Supervisors on the other hand can see all the stock ingredients.

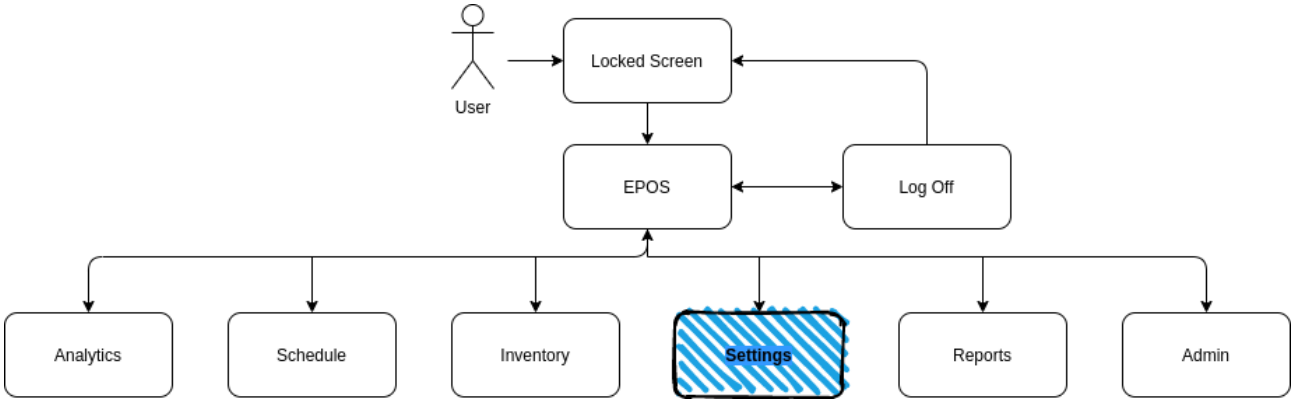




Inventory Ingredient	Supplier	Unit Cost - £	Unit Weight - kg	Current Stock - kg	Min. Stock Needed - kg	Auto Ordering	Edit
Whole Milk - Pack of 6 bottles	Estate Dairy LTD	7.00	6.000	42.650	12.000	True	Edit Delete
Coffee Beans Bag - 1kg	Origin Coffee Roasters	19.00	1.000	94.120	3.000	True	Edit Delete
Paper Cups 8oz 500 pc.	CREATION FOODS	15.00	0.500	98.625	1.000	True	Edit Delete
English Loose Tea - 1kg	Origin Coffee Roasters	10.00	1.000	99.000	1.000	True	Edit Delete
Green Loose Tea - 1kg	Origin Coffee Roasters	10.00	1.000	97.960	1.000	True	Edit Delete
Still Water 500ml - 20pc.	CREATION FOODS	5.00	10.000	70.500	10.000	True	Edit Delete
Sour Dough Bread - 1kg./1pc	CREATION FOODS	1.50	1.000	57.800	3.000	True	Edit Delete
English Muffin - 10pc./1kg.	CREATION FOODS	5.00	1.000	92.700	3.000	True	Edit Delete
Bacon 8pc.	Mr. Kitchen	2.00	0.240	97.470	0.480	False	Edit Delete

Figure 4-9: Inventory screen

The figure below represents the settings menu of allPOS. The menu allows users to create new employees and suppliers, update VAT charges for client bills, list all employees and suppliers, and enable/disable automatic raw good ordering. In addition, it includes a settings menu for the automation ordering process which allows you to customise the automatic ordering job. Accounts with basic employee permissions do not see this menu. Supervisors on the other hand are able to only create an employee or suppliers account.



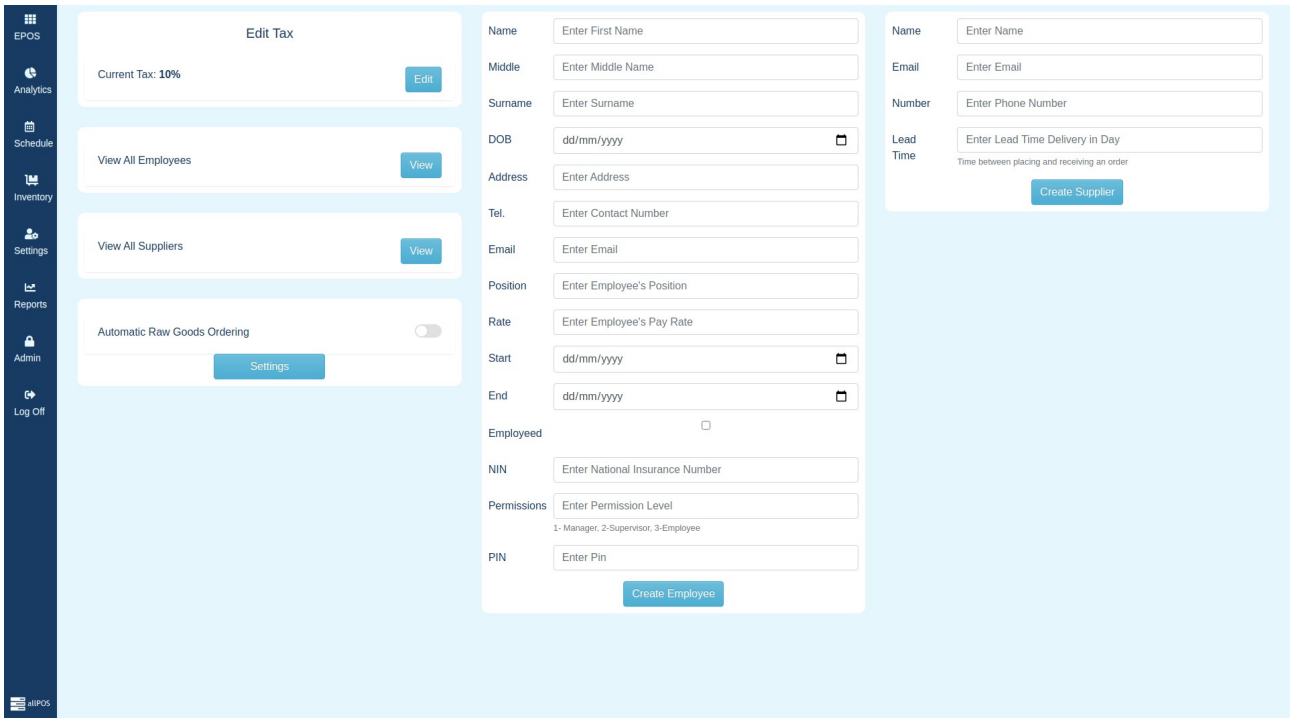


Figure 4-10: Settings screen

Perhaps, one of the most impressive functionality of Django is the programmed administrator interface. It peruses metadata from the Django models to give a speedy, model-driven interface where clients can manage the website's content. The access to the Admin is restricted, and only owners/managers have access to it. Figure 4-11 represents the Django admin interface to which only managers have access.
It provides all the database tables, and the user can view, edit, create or delete table rows.

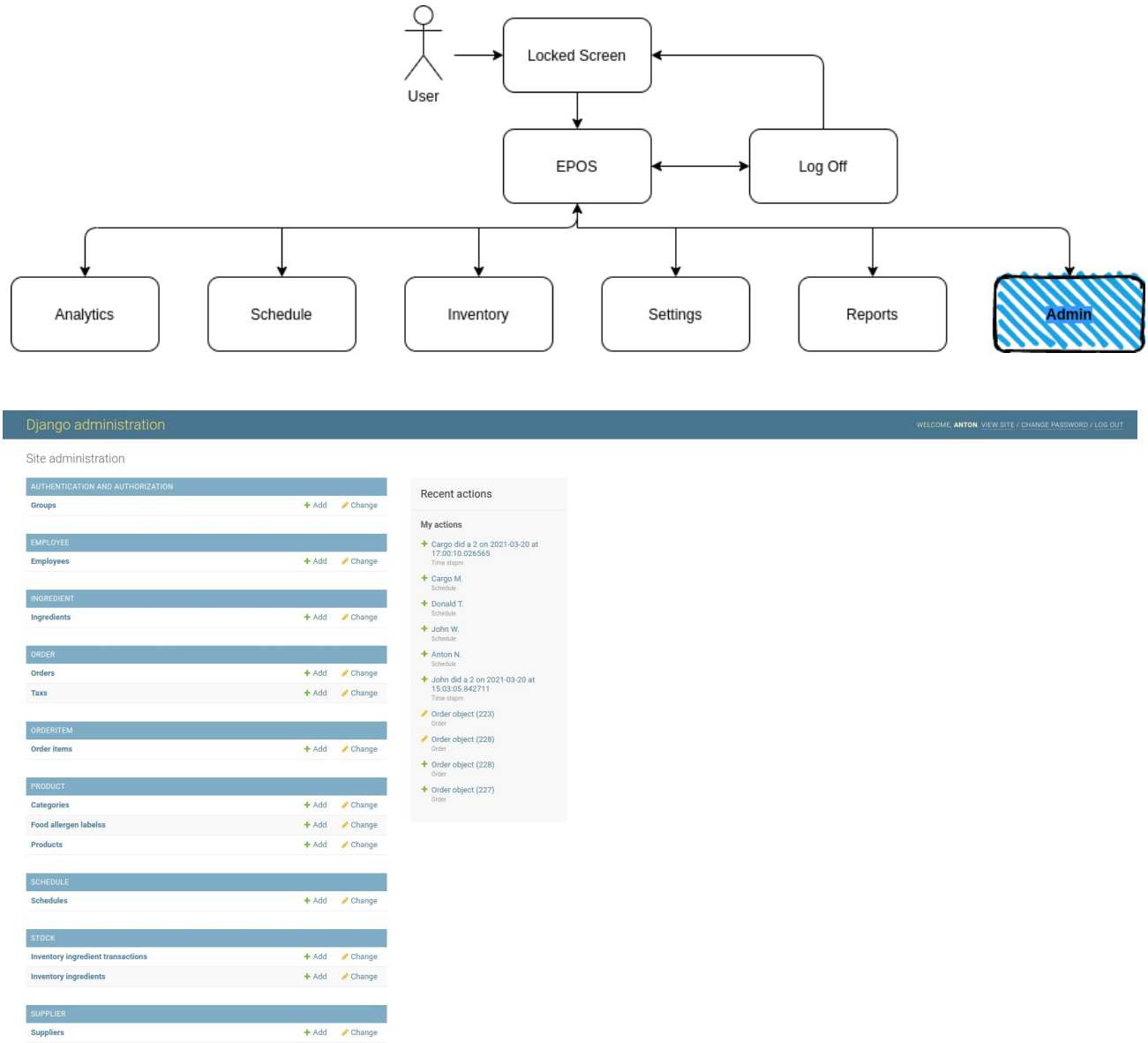


Figure 4-11: Admin screen

4.6. User Interface design

One of the project's main requirements requires allPOS to be compatible with any electronic device that is connected to the internal organisation network. I have considered several different screen size and devices through the app development process and employed the UI design to be compatible with all of them.

The colour scheme is different shades of blue and white. Buttons are easily accessible due to their size and borders and can be clicked by both - mouse pointer or finger (touchscreen).

A side nav-bar was implemented so the user can access each primary functionality regardless of their current position in the web page hierarchy.

4.7. Summary

This chapter illustrates how allPOS is developed, employing an MVC design pattern. It shows the high-level design of the MVC pattern and the relation with the different application components. Moreover, it defines the low-level implementation of the software and describes the development process's decisions. Furthermore, a high-level application flow is presented alongside a user interface design description.

5. Testing

A test-driven development is applied through the coding process of allPOS, allowing the isolation testing on each function and checking for their validation and correctness. On passing a test, the function is considered valid, and the implementation process is moved on to the next application functionality. However, after failing a test, the code is examined, refactored, a new code is developed and tested.
This manner provides a secure way of extending the developed functionalities and ensuring that the entire app works as expected.

Moreover, after the end of the development process an end-to-end approach was adopted for further functionalities acceptance. The testing covers most of the user web interactions monitoring the state of the database, based on the users input.

5.1. Compatibility

Compatibility is listed as the first non-functional requirement of allPOS and has been fully achieved. Each view has been developed through the development process to fit different screen sizes, and the application is entirely responsive and functional on each of those devices.
Moreover, allPOS can be deployed on a mobile device that successfully uses most of the functionalities. The UI development's main focus was on a computer/tablet/laptop screen due to the project's time limitations. The primary screen sizes that allPOS is developed for and tested on are:

- Laptop/Monitor size of minimum 1440 x 900 pixels
- Nexus 7 - landscape mode 960 x 600 pixels
- iPhone 4 - vertical mode 320 x 480 pixels
- Galaxy S5 - vertical mode 360 x 640 pixels
- Pixel 2 - vertical mode 411 x 721 pixels
- iPhone 5, 6, 7,8, X, 11, 12 - vertical mode 320 x 568 to 375 x 812
- iPad, iPad Pro 1024 x 768 to 1366 x1024 pixels
- Surface Duo - vertical and horizontal mode 720 x 1114 to 1114 x 720

5.2. Efficiency

Efficiency is a vital requirement for allPOS. Moreover, a response time less or equals 5 seconds allows users to perform their day-to-day job efficiently. Furthermore, it provides a more satisfying customer experience due to the rapid order process that the EPOS provides. This project adopts a response time of 5 second or less. Anything exceeding 5 seconds is considered a "bad" response time. Tests were conducted on all devices mentioned in section 5.1. All devices were running the application within a reasonable response time.

5.3. End-To-End Testing

If unit testing aims to test a specific unit of the code and integration testing tests how those units work together, end-to-end testing tests the entirety of the application. It tests the front-end and the back-end, the rendering of every single page/functionality that we are testing in a browser environment. End-to-end testing is important because it makes sure that the application is working correctly as a whole.

The developed end-to-end tests include the testing off the most important features of the application and simulate how a user might interact with the webpages. In the initialisation process of each individual test a database is created an filled with data passed as dump data json files. Regardless of the database alteration during tests, on test termination the database is restored to its initial state so the next test can use it. In this way we encapsulate tests from deployment database allowing us to alter objects in the database without affecting the main data of the application.

The following test have been conducted:

- **Employee**: clocking-in, clocking-out, login with correct credentials, login with wrong credentials
- **EPOS**: testing different functionalities of the electronic point of sale terminal, including, order placing, amount of added products changing, changing type and number of order
- **Analytics**: changing in the analytics view based on placed orders, it simulates order placing and changing in statistics
- **Employee Schedule**: it simulates the creation and edition of employee shifts and their appearance in the weekly shift calendar
- **Inventory**: the stock management system is designed to handle changes in the inventory and monitoring of low stock. Those tests cover the creation and edition of ingredients, products and inventory ingredients
- **email sending service (used by cronjob)**: this test aims to provide several ingredients that are low on stock. Then it fires a request to the automated_ordering_service and verifies if emails have been generated and sent to the ingredient’s suppliers.

5.4. Summary

Chapter 5 provided information about how testing was conducted and the procedures used. It includes general testing of the system while in development, non-functional requirements testing and final system testing representing complete end-to-end testing over the whole application.

6. Conclusion

Chapter six would summarise the learning experience from developing the final year project. Furthermore, it would describe the skills acquired during the development process, list the faced challenges, and describe further development.

6.1. Leaning outcomes and achievements

The final year project presented me with an exceptional opportunity to further develop my analytical skills and acquire in-depth knowledge of software development processes, documentation, and testing.
Compared to my academic and personal projects, this is the largest software project developed so far, and it gave me an in-depth knowledge of web development, design patterns and UI. Besides, I obtained an offer for a full-time job after graduation working as a Full Stack developer with two of the technologies used in this project - Django and JavaScript.

Overall, I am incredibly pleased with the outcome of the application, the knowledge that I gained and the experience of developing medium to large scale software.

The fundamental modules that notably contributed to this project's development were Web Development (Year 3, Semester A) and Software Engineering (Year 2, Semester A).
Web Development introduced the fundamentals of web frameworks and underlying concepts involved in a web application's development processes. Besides, it provided an in-depth technical run-through of the Django framework and the MVC pattern that Django employs. I found the module incredibly useful because I was able to apply successfully all the knowledge gained from this module.
The Software Engineering module laid the foundations of how large scale software is built. I gained an understanding of documentation, requirements, stakeholders, different types of UML diagrams and testing processes. Furthermore, I learned about the different design principles behind most software patterns and how they fit into different use-cases.

6.2. Faced challenges

Due to my experience in the retail and hospitality industry, I assumed that I have all the necessary accounting knowledge to successfully implement this project. However, I realised that that was not the case. All the micro-transactions behind the ordering processes, scheduling, analytics and calculations required me to further investigate the field of business accounting and basic management calculations. This experience provided me with important knowledge that would serve me well in the future, regardless of the job I take.

Another faced challenge was the database design planning and implementation. This project requires multiple micro-transactions per one user action. The high-level design of the project hides the complexity of the database and the back-end. For instance, when placing an order the user only sees the creation of an order. The back-end, however, needs to communicate with several different models in order to generate the required order.
In addition to the generated new order objects, another 4 objects are created for different purposes such as analytics and monitoring.

This example showcases the complexity of the database and its underlying back-end functions. However, the solution to the problems mentioned above required me to further investigate not only how to develop the database to achieve the mentioned functionalities, but also to implement it in a way that would be sustainable for a large amount of data.
In addition, the database design when through several transformations during the implementation period and thought me different concepts when it comes to business database implementation.

The successful implementation of allPOS clearly showcases the skills that I acquired in order to overcome those challenges and learn from them.

6.3. Further development

To fully implement a system of this scope, a team of several people is needed. Even though allPOS covers its primary and secondary requirements and achieves the identified gaps in the market, there are still plenty of functionalities yet to be implemented in order to establish the application on the market. For this reason, I have added some of them as non-functional buttons, but the implementation of the actual features was not achieved due to time and resource limitations.

Furthermore, reports generation functionality was dropped due to time constrains. This does not affect the primarily goal of the projects and could be left for future development as well as other features as peripheral device integration.

6.4. Conclusion

This chapter concludes the objectives and implementation of the project. It provides the reader with an overview of my experience, faced challenges and achieved goals. In addition it describes the further work that needs to be done in order to have a fully functional and market competitive product.

Reference List:

[1] John Wolfe (2018). A Brief History of Python.
Available from: https://medium.com/@johnwolfe820/a-brief-history-of-python-ca2fa1f2e99e [Accessed 10th November 2020]

[2] Timothy Ko (2017). A Quick Glance of Django.
Available from: https://medium.com/@timmykko/a-quick-glance-of-django-for-beginners-688bc6630fab#:~:text=robust%20and%20scalable.-,Django%20Architecture,such%20as%20MySql%2C%20Postgres). [Accessed 10th November 2020]

[3] Joseph Spinelli (2018). MVC Overview
Available from: https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5 [Accessed 11th November 2020]

[4] Sveta Yurkevich (2020). POS Interface Design Principles.
Available at: https://agentestudio.com/blog/design-principles-pos-interface
[Accessed 10th November 2020]

[5] Hamish Willee (2020). Django Introduction
Available from:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction
[Accessed 10th November 2020]

[6] Xaview Ferre, Natalia Juristo (2001). Google Scholar: Usability basics of software developers.
Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=903160 [Accessed 10th November 2020]

[7] Shi Xu (2017) The effect of online scheduling on employees' quality of life
Available from:
https://www.tandfonline.com/doi/full/10.1080/15378020.2017.1364592?scroll=top&needAccess=true [Accessed 13th of November 2020]

[8] Shopify (unknown) Inventory definition.
Available from: https://www.shopify.co.uk/encyclopedia/inventory [Accessed 13th November 2020]

[9] Renae Smith (2019) Inventory management for hospitality businesses
Available from: https://www.myob.com/au/blog/inventory-management-software-hospitality/ [Accessed 13th November 2020]

[10] Peter Bengtsson (2021) Setting up Django development environment
Available from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/development_environment [Accessed 16th March2021]

[11] Smitha Mundasad (2014) Food allergy laws enforced in restaurants and takeaways
Available from: https://www.bbc.com/news/health-30395142 [Accessed 18th March2021]

[12] Theme Fusion (2021) Modal Element
Available from:
https://theme-fusion.com/documentation/avada/elements/modal-element/
[Accessed 18th March 2021]

[13] Paulo Oliva (2020) Web frameworks – ECS639: Web Programming
Available from:
https://qmplus.qmul.ac.uk/pluginfile.php/2278293/mod_resource/content/19/lecture02-django.pdf
[Accessed 18th March 2021]

Appendix

The final year project is a great way to demonstrate the skills that we have acquired at university for the last four years. Besides, it allows us to progress in a very niche direction of our choice, by working on a project that can influence our future employment and it's important to us.

To successfully implement my final year project and define a specific sector in which I wanted to dive in, I started to establish the scope of the project by reading different articles related to restaurant and retail systems. After the research stage, I had identified the gap in the hospitality system sector, and the scope of my project was established.

The second stage of the final year project was the literature review and literature research. It was carried for a couple of weeks by reading different research papers on the topic and processing the information that is relevant to my project. Additionally, software research was done alongside the literature review, examining different scenarios for the project implementation and design architecture. Implementation articles, libraries and templates were gathered to support the developing stage.

To maintain the monitoring of the project progress, I have decided to use a work management tool called Jira. Currently, I am defining the backlog with independent development milestones that are divided into tasks, also known as stories in the Jira ecosystem. Moreover, a project skeleton is currently under development and research of different libraries is in progress.

The project plan was clearly defined, and implementation goals were established in the following manner.
The development would be performed into three stages, as follows:

- **Stage One**: Point of Sale system
    - Start Date: 20/11/2020
    - End Date: 30/12/2020

- **Stage Two**: Employee Scheduling system
    - Start Date: 01/01/2020
    - End Date: 28/02/2020

- **Stage Three:** Stock Management system
    - Start Date: 01/03/2020
    - End Date: 30/04/2020

The project would be conducted in behaviour-driven development using a top-down approach. It defines the work in rapid, small iterations manner to increase the feedback of each module by continuous testing.
The starting point would be a user story implementation, testing and refactoring. TDD would be driven into weekly sprints, leaving the last sprint of each stage for final acceptance testing. Furthermore, a sprint would be starting with a user story, progressing into testing. If bugs are found it would continue with implementation and further testing.

Currently, the following requirements are organised into sprints for the first stage (Point of Sale system) of the implementation:

- **Sprint 1** (20/11 - 27/11):
    - Set up GitHub
    - Set up Jira
    - Set up OpenShift - deployment
    - Define all necessary libraries and front-end templates

- **Sprint 2** (27/11 - 04/12):
    - Define Relational Models
    - Implement Relational Model
    - Implement skeleton home page
    - Test models and home page

- **Sprint 3** (04/12 - 11/12):
    - Implement views
    - Implement front-end
    - Test views

        - Test front end

- **Sprint 4** (11/12 - 18/12):
    - Implement views
    - Implement front-end
    - Test views
    - Test front end

- **Sprint 5** (18/12 - 31/12)
    - Unity testing of each functionality
    - Smoke testing of the whole POS application
    - Integration Testing
    - System Testing

Employee Scheduling and Stock Management systems would be planned in the same manner. Consisting of individual sprints.

The first figure, bellow represents a Gantt chart for the overall project. It includes tasks that are done, tasks in progress and future task planning. In addition, due to the size of the Gantt chart, further parts of the graph would be provided bellow the overall chart of the project to illustrate details on each stage of the implementation.

From the charts, it could be seen that different stages of the project are planned to be developed in a different manner. For instance, stage one has less number of sprints. That is due to the fact that it’s implementation is less complicated than Employee scheduling and Stock management systems.

Furthermore, Employee Scheduling system has less Back-End sprints than the Stock Management system. This is a component independent decision. The scheduling part of allPOS has more front-end functionalities, whereas the stock control module provides the business logic of allPOS, therefore, needing more back-end development.

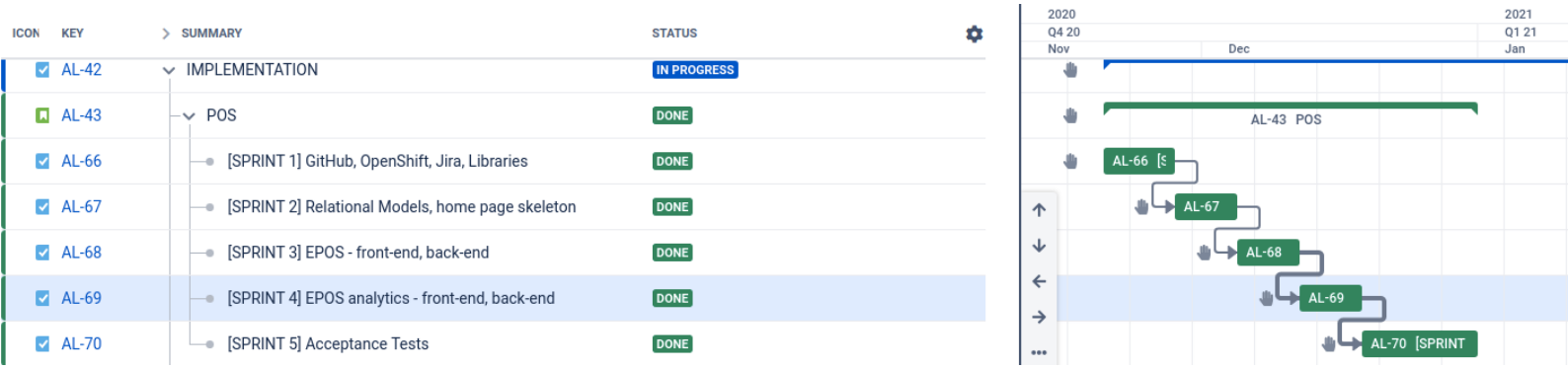


Figure A: Point of Sale

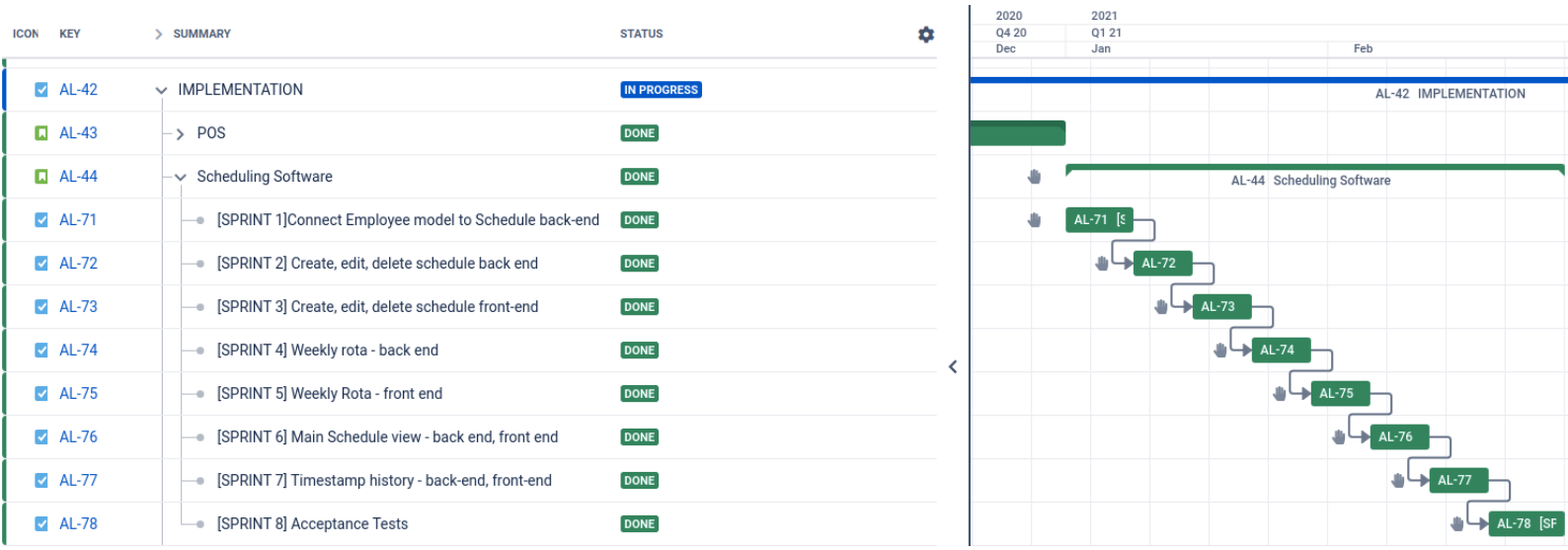
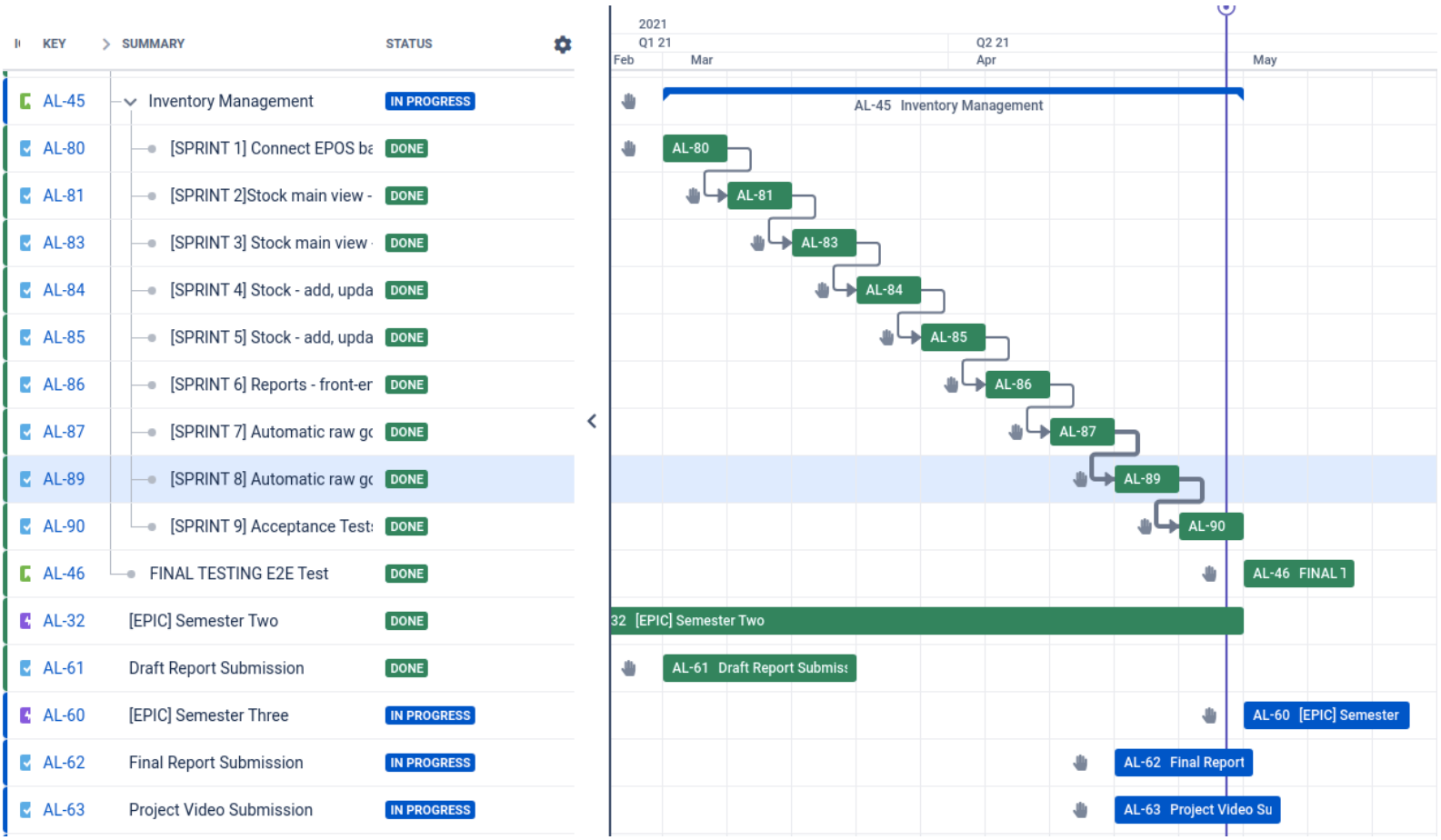


Figure B: Employee Schedule


KEY
SUMMARY
STATUS
AL-45 Inventory Management IN PROGRESS
AL-80 [SPRINT 1] Connect EPOS ba DONE
AL-81 [SPRINT 2]Stock main view - DONE
AL-83 [SPRINT 3] Stock main view DONE
AL-84 [SPRINT 4] Stock - add, upda DONE
AL-85 [SPRINT 5] Stock - add, upda DONE
AL-86 [SPRINT 6] Reports - front-er DONE
AL-87 [SPRINT 7] Automatic raw gc DONE
AL-89 [SPRINT 8] Automatic raw gc DONE
AL-90 [SPRINT 9] Acceptance Test: DONE
AL-46 FINAL TESTING E2E Test DONE
AL-32 [EPIC] Semester Two DONE
AL-61 Draft Report Submission DONE
AL-60 [EPIC] Semester Three IN PROGRESS
AL-62 Final Report Submission IN PROGRESS
AL-63 Project Video Submission IN PROGRESS
2021
Q1 21
Q2 21
Feb
Mar
Apr
May
AL-45 Inventory Management
AL-80
AL-81
AL-83
AL-84
AL-85
AL-86
AL-87
AL-89
AL-90
AL-46 FINAL T
32 [EPIC] Semester Two
AL-61 Draft Report Submis
AL-60 [EPIC] Semester
AL-62 Final Report
AL-63 Project Video Su

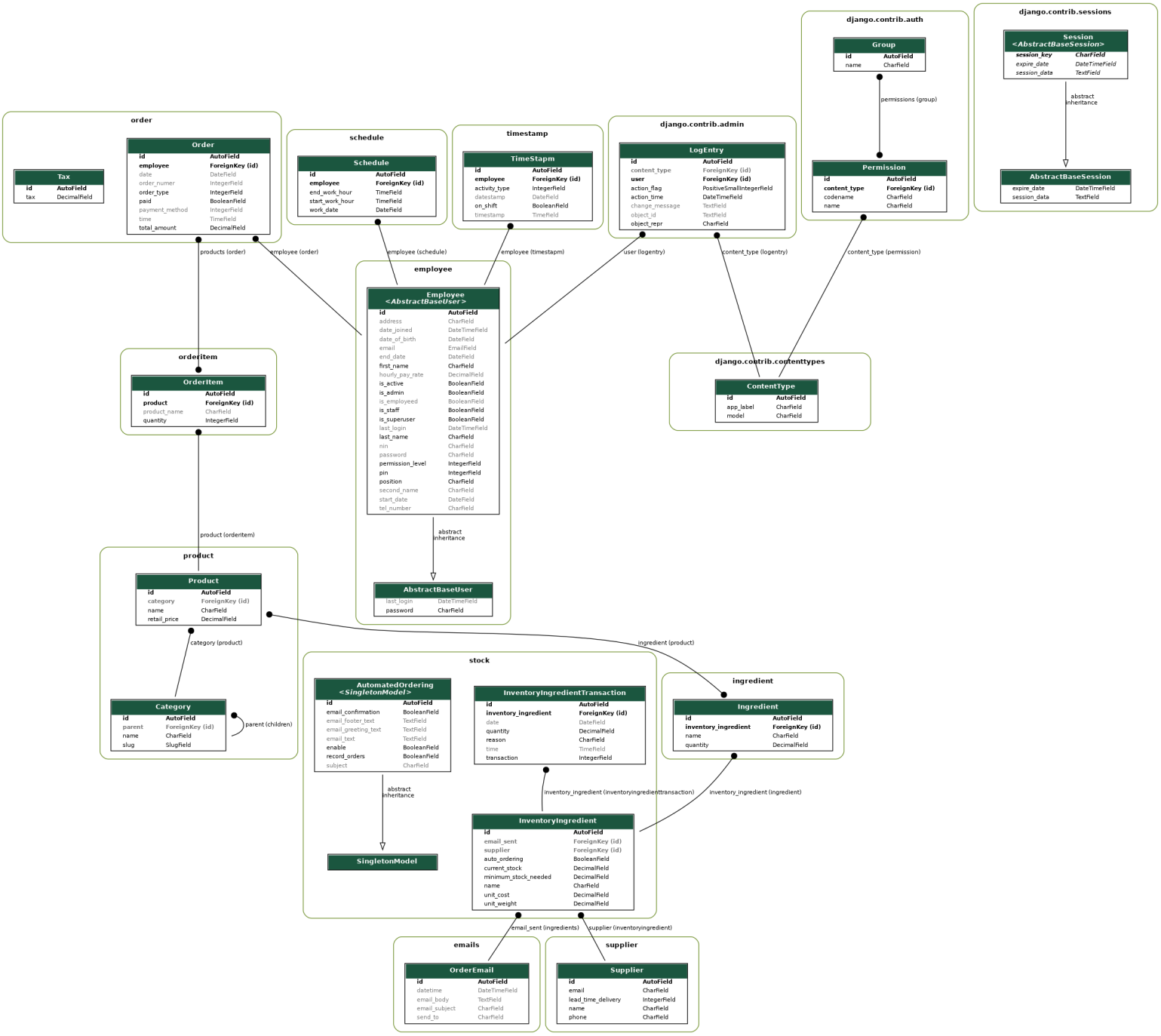


Figure D: Database Design Diagram