RETAIL PHARMACY INVENTORY MANAGEMENT SYSTEM

GERALYN TING YIK HUEY

SESSION 2022/2023

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
JULY 2023

RETAIL PHARMACY INVENTORY MANAGEMENT SYSTEM

BY

GERALYN TING YIK HUEY

SESSION 2022/2023

THE PROJECT REPORT IS PREPARED FOR

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
IN PARTIAL FULFILLMENT
FOR

BACHELOR OF COMPUTER SCIENCE
B.CS (HONS) SOFTWARE ENGINEERING

FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

JULY 2023

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

_____

Name: Geralyn Ting Yik Huey
Student ID: 1181102223
Faculty of Computing and Informatics
Multimedia University
Date: 14 July 2023

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor Dr. Yeoh Eng Thiam for guiding me throughout this Final Year Project and providing feedback whenever I needed it. Your patience is very much appreciated and has encouraged me to work out my problems independently and with greater depth.

ABSTRACT

This project is developed to tackle the complications of inventory management, specifically in the retail pharmacy domain. Many retail pharmacies still use traditional book keeping and manual systems. Perhaps this is due to the lack of affordable software solutions, awareness of current solutions on the market, or the existing systems are too lackluster and incomplete for what is demanded of retail pharmacy inventory management systems.

Chapter 1 details the plans, timeline and main objectives, which act as a guideline for the direction of the project. Chapter 2 details the research and background study covered to determine existing systems, points of view from literature survey, problems to be solved, and technological background study to find which existing technology will be most useful in the proposed solution.

From the background study, it is gathered that in addition to the changing demands of this climate, one aspect that grows in importance is analytics and specifically for the pharmaceutical industry, reducing deadstock as much as possible since the deadstock is largely medication that is not safe to use anymore. Not only does this incur losses from a business perspective, but more importantly, has implications towards environmental damage since it is complicated to dispose of medication safely.

Based on the information gathered in Chapter 2, requirements are constructed in Chapter 3 and visualised with a System Overview, Use Case Diagrams, Use Case Description Tables and an Entity Relationship Diagram to detail out the data structure to be implemented. Chapter 4 takes the requirements from Chapter 3 and further develops them into design components, namely Software Architecture, Sequence Diagrams, Screen Designs and Data Dictionaries which act as a floorplan for the prototype.

Finally, Chapter 5 goes over the implementation plan for Phase 2 of the project as well as some reflections on which aspects can be improved on. Future work will focus on further developing the ordering and reporting functionalities and the inclusion of analytics and recommendation functions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 Project Overview:

Pharmacies and medical institutions involved in the medical supply chain face a common problem that if not handled properly, can incur massive revenue loss and environmental damage. The aforementioned problem is excessive medicinal waste. This is due to overstocking on perishable medication that eventually ends up being thrown away when they are unsold and too risky to keep in stock.

This Retail Pharmacy Inventory Management System has an emphasis on reducing the amount of wasted medicinal inventory through analytic tracking, reporting, forecasting and recommendations. The system includes the management of other activities such as order and appointment scheduling, patient records, and stock ordering within a retail pharmacy to keep data organised and easily accessible for staff within their respective pharmacy branches. In order to calculate and generate accurate reports for optimal inventory management recommendations, the system also includes a Point of Sale (POS) interface that directly links to the inventory.

This project will be developed as a Python Web application and completed within 2 trimesters with Django framework and MySQL for database management. For prediction and forecasting purposes, Sci-kitlearn and Keras libraries will be compared for accuracy and optimization.

1.2 Project Objectives:

The main objectives of this project are as follows:-

- To study the requirements of the inventory management system.
- To design and develop a prototype system based on the requirements
- To improve the efficiency in managing the products in the pharmacies of the organisation
- To provide an effective way to maximise inventory and reduce medication wastage
- To identify the most suitable Artificial Intelligence library for predictive modelling to give accurate recommendation results.

1.3 Goals of project:

The goals of this project is as follows:-

- To create a working inventory system prototype with a functioning database
- To create working and accurate predictive models using machine learning and deep learning methods and compare the two.
- To implement sales, inventory, stock and supplier reporting.
- To study the requirements of retail pharmacies and what functionalities are prioritized.

1.3 Project Plan

1.3.1 Milestones

Table 1.1 Milestones for Trimester 1

Week	Milestones	Due date
2	Project Plan	6.10.2022
4	Background Study	20.10.2022
7	Requirements Use Case Diagram Entity Relationship Diagram	11.11.2022
9	First Draft of Report	15.12.2022
9	Design Software Architecture Sequence Diagram Screen Design Data Dictionary	15.12.2022
14	Prototype	19.01.2023
15	Report Submission	26.01.2023
16-17	Presentation	2.02.2023 - 9.02.2023

Table 1.2 Milestones for Trimester 2

Week	**Milestones**	**Due date**
2	FYP 2 Plan Update	6.04.2023
6	Database Schema Update	27.04.2023
8	Admin Module Completion	11.05.2023
10	Branch Pharmacist Module Completion	25.05.2023
12	Assistant Module Completion	8.06.2023
9	First Draft of Report	18.05.2023
11	Produce Test Plan	1.06.2023
12	Produce Test Data	8.06.2023
12	R-Squared, MAE, RMSE tests for predictive models	11.06.2023
13	GUI & Functional Testing	15.06.2023
13	Evaluation	15.06.2023
13	System Finalisation	15.06.2023
14	Commercialisation Proposal	22.06.2023
15	Final Report & Poster	29.06.2023
16-17	Presentation	3.07.2023 - 14.07.2023

1.3.2 Gantt Chart

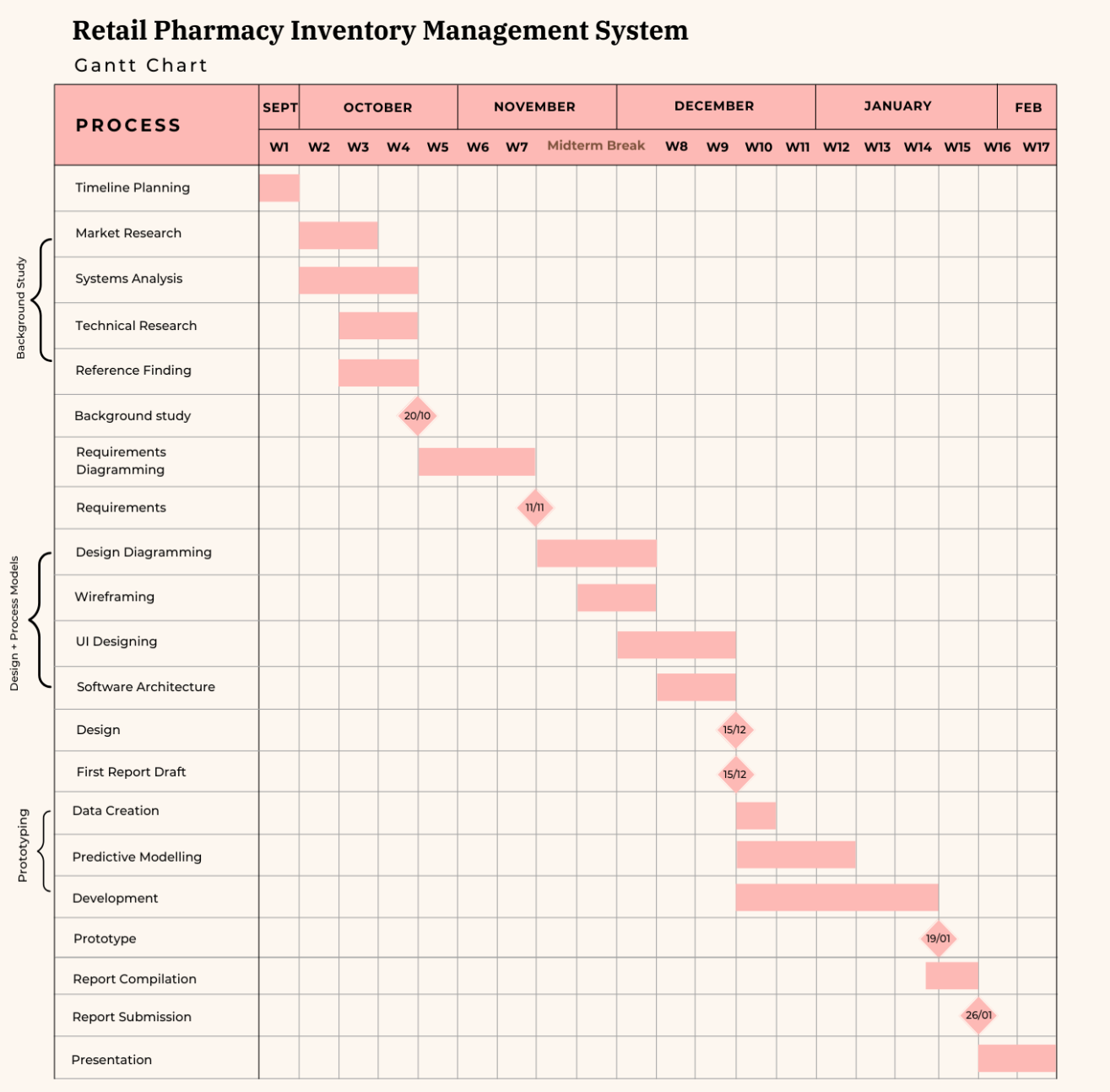

Figure 1.1 Gantt chart for Trimester 1

The Gantt Chart above shows the timeline for Final Year Project 1 which pertains to the main activities to be done. Week 1 focuses on setting objectives and goals of the project and using the Gantt Chart to visualise the timeline. Week 2 to Week 4 concerns the background

study which will include market research to find out what the pharmacy retail industry demands are and what existing solutions are available and most used. 3 weeks will then be taken to analyze the systems of these solutions and any additional system requirements that seem feasible as well as valuable will be considered for the project. Next, technical research will be done to ascertain what tools would best fit the project. At the same time, reference finding will be conducted. Week 5 to 7 consists of requirements diagramming like those of use case diagrams and class diagrams. The midterm break to week 9 concerns all the designing aspects such as diagramming, wireframing and finally User Interface designing. At this point, the first draft of the report will be ready. Next, Week 9 to Week 14 will cover the implementation of the project, starting with data creation and the predictive modelling function and finally the implementation of the other parts of the project to create the first prototype. To wrap it up, the remaining weeks 14 to 15 is reserved for the report compilation and submission followed by the presentation to the moderator somewhere between week 16 and 17.

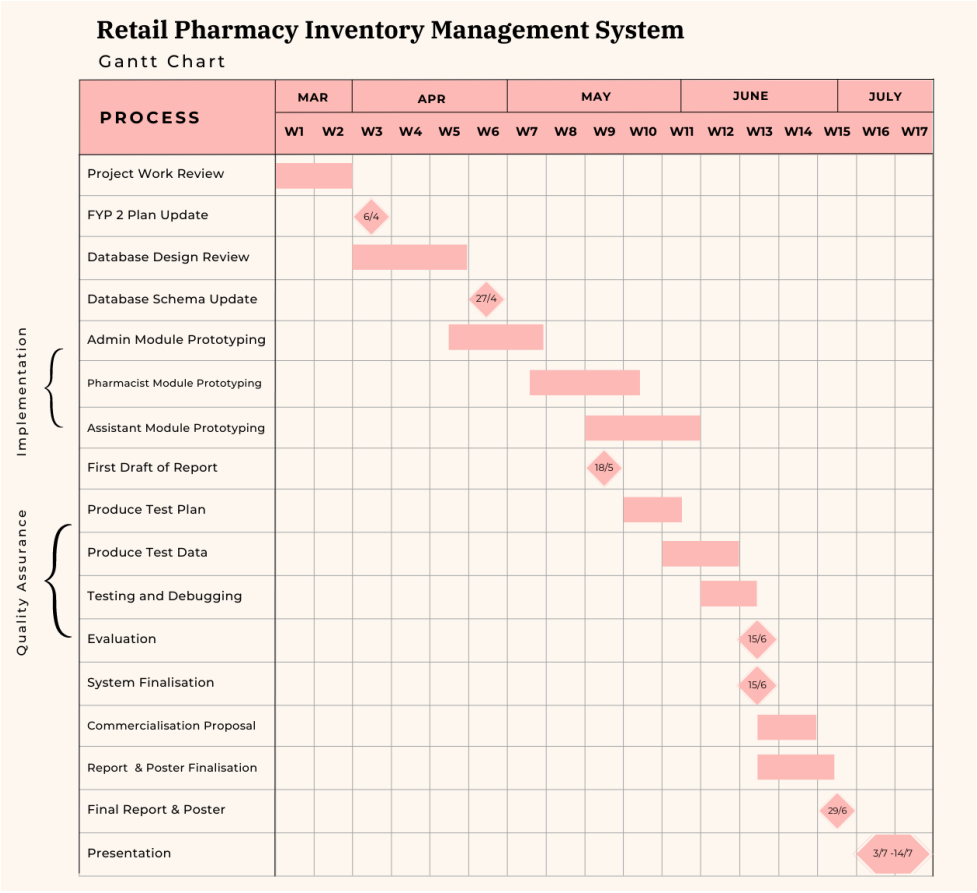

Figure 1.2 Gantt chart for Trimester 2

This Gantt Chart depicts the flow for Trimester 2. Project Work Review should be done from Week 1-2. A Plan Update will be worked on and done by Week 3. Then, a Database Design Update will be done to make the database design better. Week 5 all the way to Week 12, the implementation of the application will be done on an iteration basis to ensure continuous improvement. First draft of the Phase 2 report is to be done by 18/5/2023. Quality assurance elements such as test planning, test data producing and the actual testing will be carried out to fix any problems within the system. By Week 13, or 15/8/2023, the system should be finalised and evaluated which leaves 2 more weeks to finalise the report, poster and commercialisation proposal by 29/6/2023. Finally, the presentation will be done by Week 16.

Chapter 2: BACKGROUND STUDY

2.1 Existing Systems

Well-known existing systems with high user ratings were analyzed. These systems are PioneerRX, WinPharm Pharmacy Management and PrimeRx. A common feature they share is that their market segmentation is largely targeted on small-businesses and mid-market. One thing to note is that the term “RX” is commonly used in regards to these pharmaceutical systems, which denotes doctor prescriptions.

2.1.1 PioneerRX

Amongst all the systems available on the market, PioneerRX has the most extensive list of features to cover pharmacy workflow, Point-of-Sales and financial management from insurance claims to drug loans. While having a long list of capabilities means that most user scenarios are covered, it also means that the design of the system is far from minimalist. This is evident with how the UI is designed to be packed with information and borders on clutter, which makes the experience feel overwhelming to first time users.

Some notable features that stand out from the rest of the software systems include automated processes such as Automatic Inventory Reordering, Automated Billing and Automatic Prescription Refilling. Additionally, there is a Competitive Pricing Analysis function which takes pricing statistics from pharmacies within the country and shows them to the user so that the pharmacy can consider pricing the products at a competitive rate and avoid blind pricing to attract more clientele.

PioneerRX uses End-to-End Encryption (E2EE) to protect their patrons' credit card information. It is unstated as to whether E2EE or any other security feature is used to

protect their database of patient medical records, however any medical practitioner (including pharmacies, clinics and hospitals) is required by law to maintain the confidentiality of patient medical records. Finally, this software system has Central Office capabilities which allows a pharmacy with more than one location to manage each branch from one terminal.

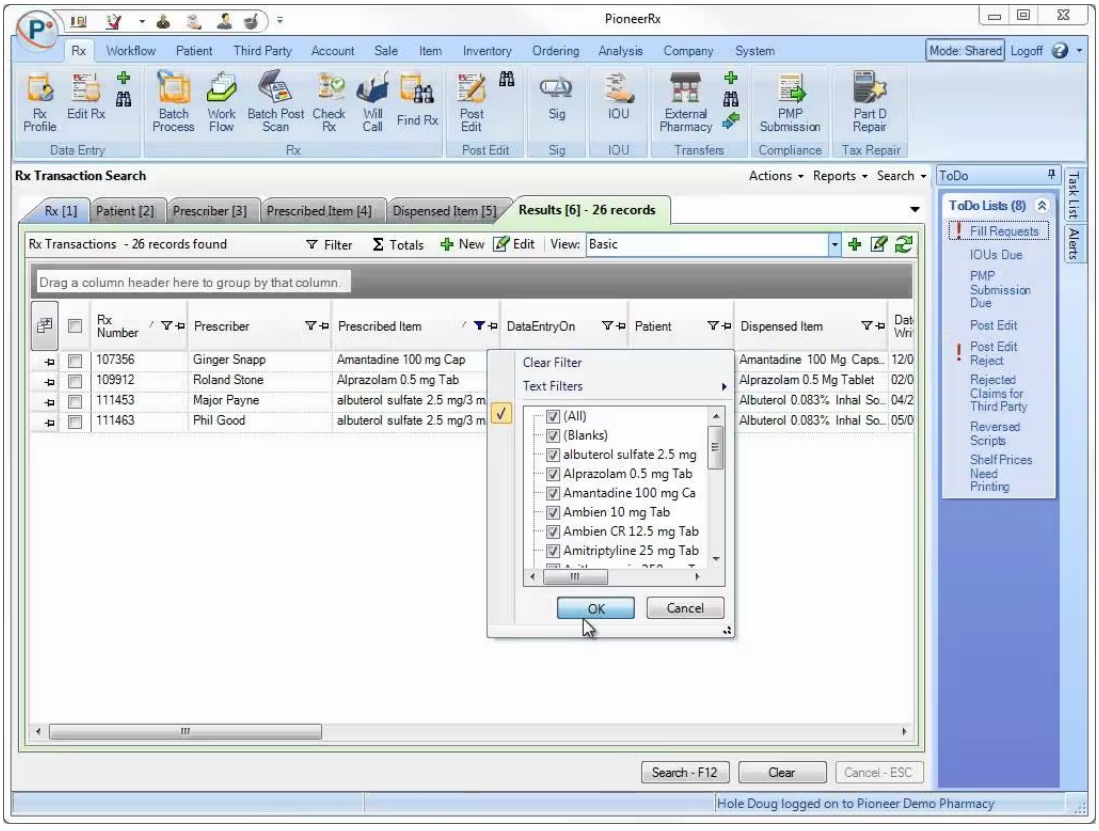

Figure 2.1 The transaction search screen of PioneerRX

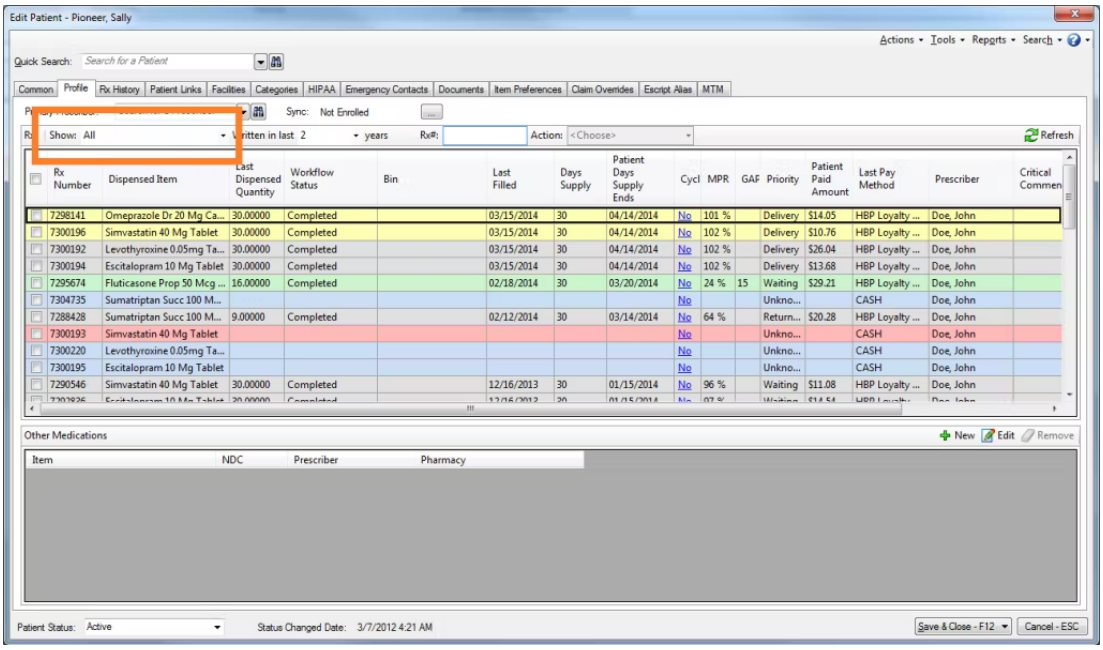

Figure 2.2 The edit patient screen of PioneerRX

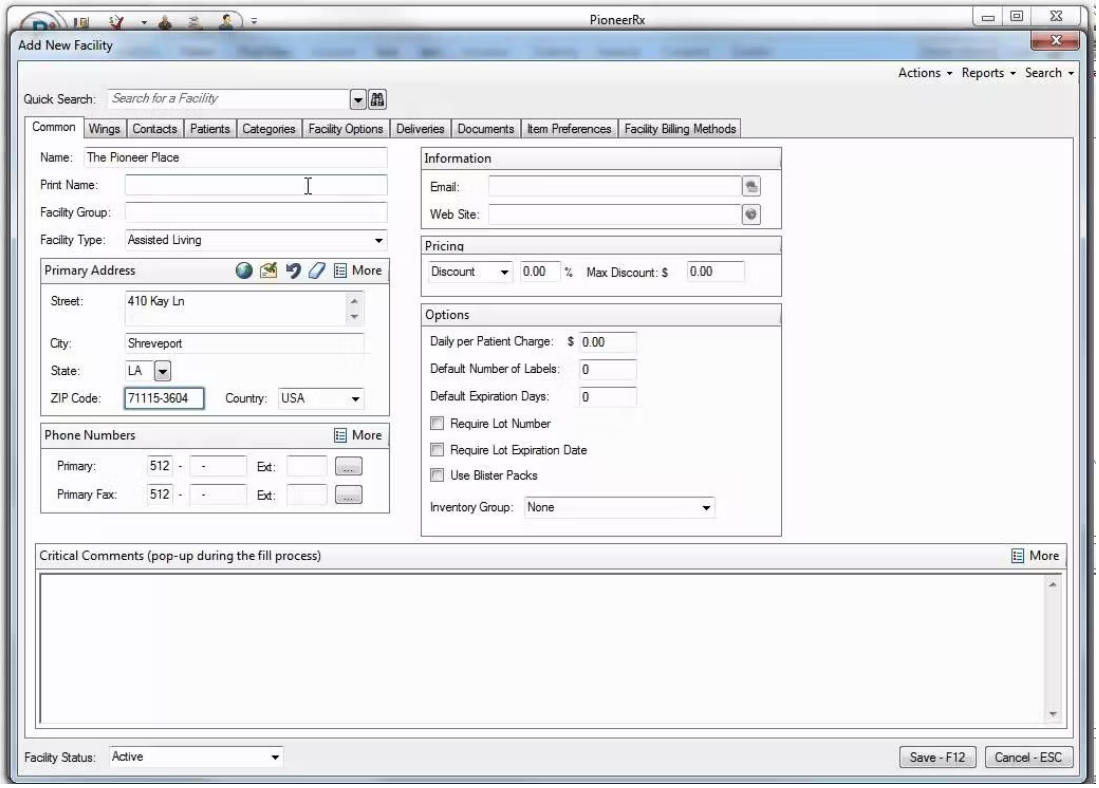

Figure 2.3 The add new facility screen of PioneerRX

2.1.2 WinPharm Pharmacy Management

WinPharm is a pharmacy management system that works alongside Datascan POS and Datascan Electronic Delivery System as part of the Datascan Pharmacy Software Suite. Since this system works separately from the rest of the suite, there is no Point-of-Sales (POS) and it focuses primarily on the pharmacy inventory, workflow and patient management aspects.

The UI design is dated and fairly simple, as are the functions. The pros are that the system is easy to navigate, has the essential functionalities for workflow and inventory management, as well as has a to-do list in plain view on the home page . The con is that while from a business perspective it is smart to license the POS separately for more revenue, it is inconvenient for the pharmacy to have to either license from Datascan or try to use a third-party POS which might not be completely compatible with WinPharm.

Figure 2.4 The home screen of Winpharm Pharmacy Management Software

2.1.3 PrimeRX

PrimeRX is a system that is targeted at independent pharmacy owners and does not have the functionality for the user to manage several pharmacy branches at once. It has inventory management, patient and customer management, and secure audit trail functions. However the main drawback of this system is the cluttered and dated look of the UI, the inability to search by prescription (RX) number, and the fact that there is no integrated POS system.

The highlights are the ability to view cost comparisons between wholesalers function and there is a dashboard which displays all the necessary stocking information so that users know the current status of the inventory. This dashboard can be customized to display metrics such as scheduled stock refills and expired refills

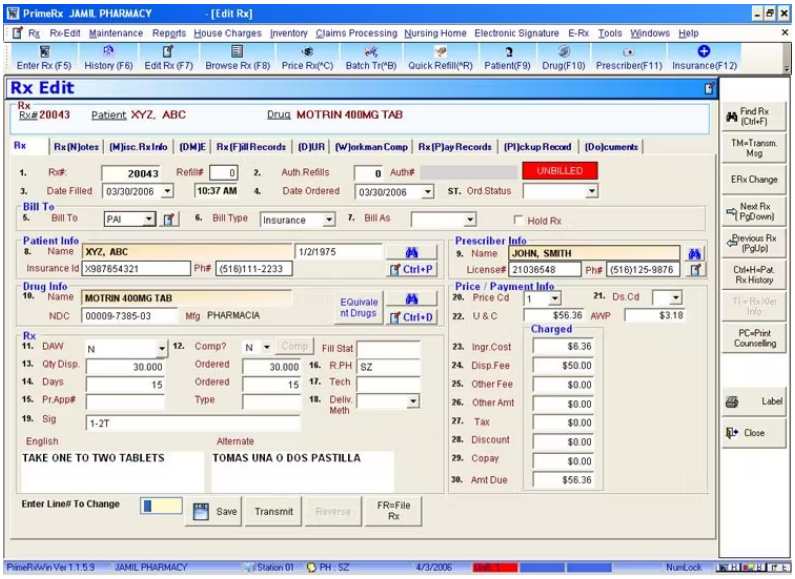

Figure 2.5 The main screen of PrimeRx

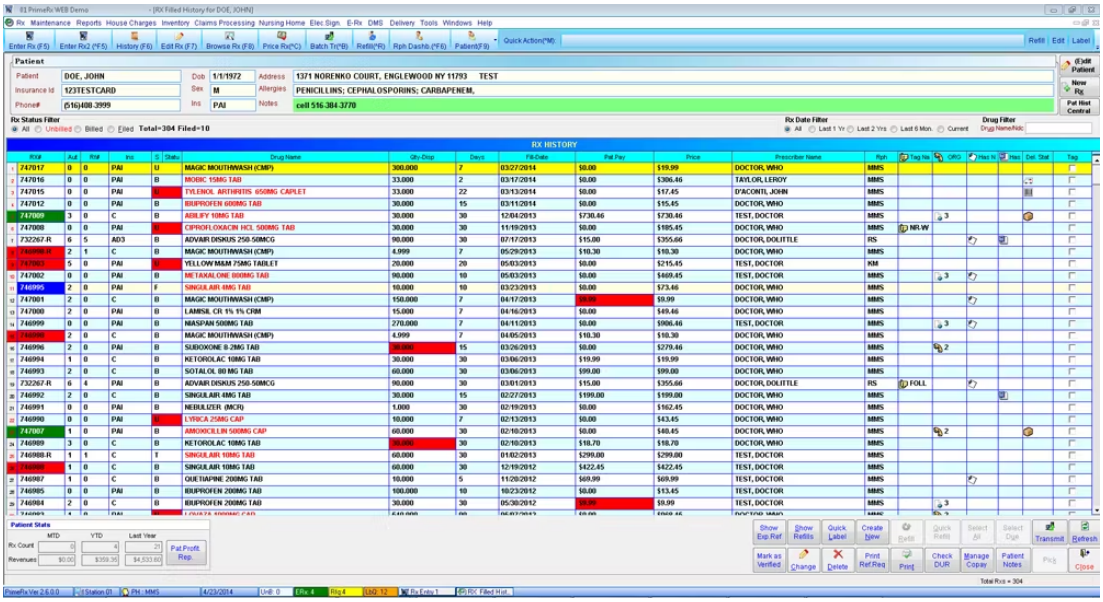

Figure 2.6 The patient history screen of PrimeRx

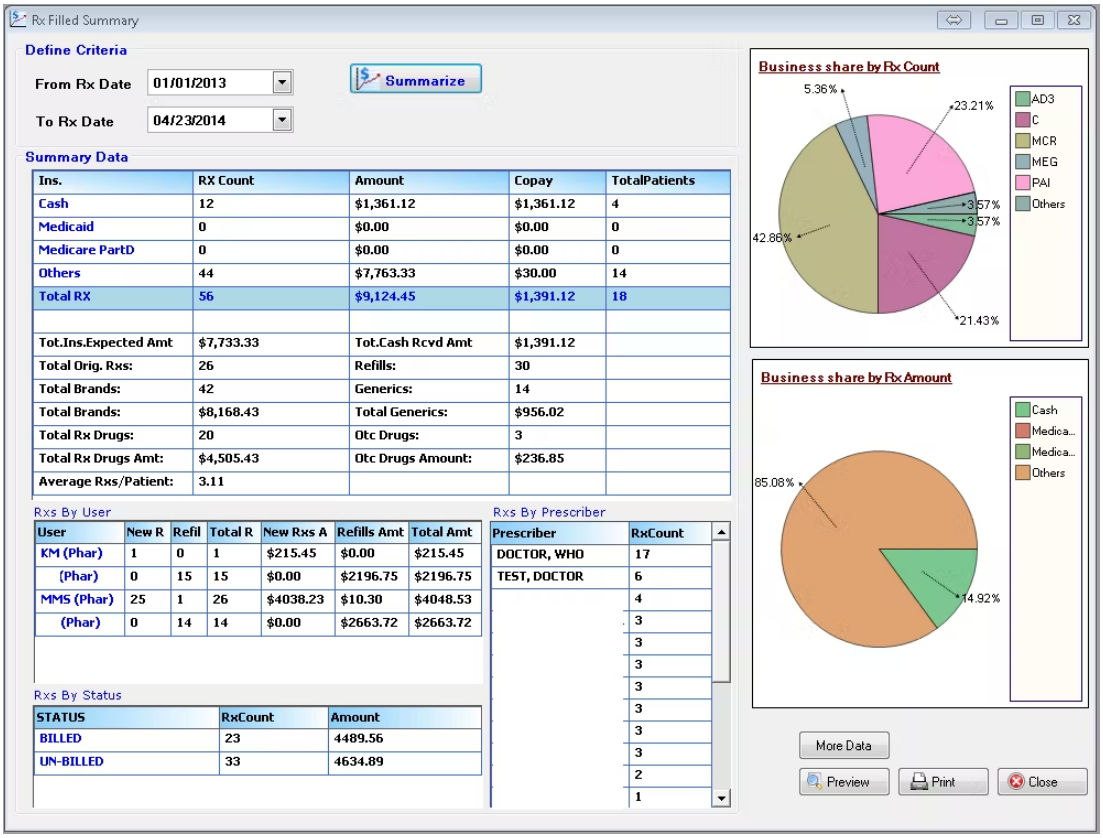

Figure 2.7 The pharmacy summary screen of PrimeRx

2.2 Literature Survey

The main problem which prompted the project topic is medication wastage, which is identified as any medication that is unused or expires at any time within the pharmaceutical supply chain. Lack of proper management will interfere with the long term sustainability of healthcare systems since the cost of the waste itself presents as an economic burden (West et al., 2014) (Alnahas et al., 2020). A single pharmacy can accrue a loss of around RM175,000 just from expired medication returned by patients (Saripin, 2019). Expired medication is particularly finicky as patient safety and ecology is put at risk and it all boils down to overproduction and overprescription in the pharmaceutical world. (Alnahas et al., 2020). In order for pharmacies to be able to take more control over the lifecycle of their products, the tools that they use to manage inventory and prescription needs to be optimized (Mahyadin et al., 2015).

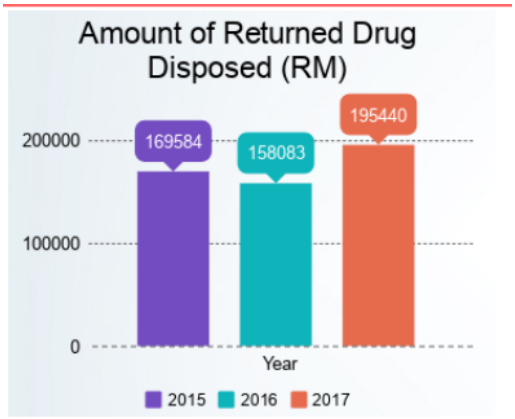

Figure 2.8 Medication Wastage Statistics 2018, Outpatient Pharmacy, HCTM UKM

There are several approaches that can be considered in order to optimise inventory management practices in pharmacies and they cater towards resource allocation, scheduling and demand forecasting. Accurately predicting the required supply of inventory depends on the type of clientele and their illnesses, which are likely to vary greatly based on the location of the pharmacy. To elaborate on this, some life-saving medication needs to always be in stock in case of an emergency (Saha & Ray, 2019) even if there is no current demand for it and some commonly used medication such as Paracetamol is always expected to be available.

In order to accommodate this complexity, one can consider building a model on top of ABC Analysis, which leverages the Pareto principle aiming to highlight the 20% of the inventory that generates 80% of profits (Kheybari et al., 2019). One of these methods is to implement a 3D selective inventory control model through segmenting data by ABC, XYZ and VED to gauge the usage value, clinical importance and demand pattern of the inventory and then recommend stocking accordingly (Kumar & Shukla, 2022). In the case studies using this model it was found that the inventories were better synchronized to demand, the cost to hold the stock was also reduced since there was less unnecessary stockholding, excess inventories were less since there was less dead stock and the turnover rate was also increased (Bialas et al., 2019, 5) (Kumar & Shukla, 2022).

Table 2.1 Recommendation Matrix based on 3D Selective Inventory Control

ABC-FSN matrix element	SUB-CATEGORY	% by volume	CONTROL	REVIEW	STOCKING REMARKS	PLACE IN STORE
AF	I	7.89	Strict	Superior	Maximize	Counter
AS	I	18.94	Strict	Superior	Maintain	Rack
AN	I	1.57	Strict	Superior	Elimination needed	Back store room
BF	I	3.15	Strict	Superior	Maintain	Rack
CF	I	0.52	Strict	Superior	Keep safety stock only	Counter
BS	II	23.68	Moderate	Periodic	moderate	Rack
BN	II	3.68	Moderate	Periodic	Keep safety stock only	Back store room
CS	II	25.78	Moderate	Periodic	Minimize up to safety stock	Rack
CN	III	14.73	Less	Not needed	Minimize	Back store room

For demand forecasting, quantitative models involve taking historical data and using a mathematical model to project future trends and there are several methods that can be considered. These quantitative models are generally categorised into either trend projections via time series models or causal models and can be carried out using machine learning techniques or deep learning techniques. With the use of Artificial Neural Networks (ANN), the time series forecasting and causal models perform very similarly (Said et al., 2013) but can be differentiated based on cost.

2.3 Problems to be solved

The main problem, as stated in the literature survey portion, is that medication waste at excess is detrimental to the healthcare system and consumers. Thus the main focus for the proposed system is to optimise the management efforts within a retail pharmacy through demand analysis and forecasting that generates action recommendations for the pharmacy so that deadstock will be reduced while still maintaining safe consumption through batch expiration monitoring. Additional effects would be a higher Return of Investment (ROI) and a lower cost of inventory retention since the pharmacy would not have to pay housing deadstock.

From the functionality point of view, upon going through the existing systems, it appears that most of them are targeted for independent pharmacies and do not support a network of pharmaceutical branches like PioneerRX's Central Office capability. Having this functionality is important for any pharmacy that wishes to expand and be still able to monitor and manage the business.

From the user interface perspective, an easy to navigate interface which is also aesthetically pleasing is preferable so that users will not feel overwhelmed trying to learn how to use the system and so that the effective functionality of the system will be supported.

2.4 Technological Background

The system will be built as a Web application so one has to consider using the right tools. Ideally, the development language will be Python for the ease of implementing python libraries for machine learning forecasting functionalities and to manage databases using SQLite3. The system will be designed to include categorizing drugs by unit doses and forms for cataloging purposes.

2.4.1 Type of Applications

In order to create a working prototype for the system, three variations of applications are considered, namely mobile, Web, and desktop applications. Picking the right one will be vital to the experience of the users so there are a few criteria to compare them to.

Table 2.2 Differences between Mobile, Web and Desktop Apps

Criteria	Mobile App	Web App	Desktop App
Installation	App Directories (Google Playstore, Apple Store, Windows Store)	No download required - accessible via Web browser	App Directories or via company site
Personal Device Dependency	Yes	No	Yes
Screen Size	Flexible: Small-Medium	Flexible: Medium-Large	Large
Internet dependency	No	Yes	No
Speed	Fast	Slowest	Fastest
Updates and	Updates must be packaged	Centralised updates	Updates must

Maintenance	then downloaded & installed by the user	and common codebase from developer, no actions on user side	be packaged then downloaded & installed by the user
Development Time	Longer	Shorter	Longer

Both mobile and desktop applications can operate without the internet natively but due to recent developments, they are not the only ones. Progressive Web Apps (PWA) can utilise a feature called the Web-worker which enables these applications to operate even when not connected to the internet. This is because they can be cached on a user's device instead of being completely dependent on being connected to a server on the internet. The drawbacks to this is that the PWA would have limited functionality while not connected to the internet and not all the browsers support it.

2.4.2 Framework:

In general, there are two categories for frameworks namely Full-Stack and Non Full-Stack frameworks. Full-Stack frameworks provide the tools to enable the entire development of the application from the frontend to backend. On the other hand, Non Full-Stack frameworks usually only include the basic functionalities to develop the application's backend and can be either microframeworks or asynchronous frameworks. One advantage of Full-stack over the others is that all the necessary components are in one place, but that also means that the developer is limited to only those specific components and cannot make changes to the modules.

For application development using Python, the few framework options being considered are Django, Flask, and CherryPy. The following table compares the capabilities of each of these frameworks. The programming paradigm criteria essentially determines whether or not the frameworks utilise OOP principles which are important to ensuring the system's code is readable, reusable and flexible. To deploy Django, it is essential to implement an interface like Web Server Gateway Interface (WSGI) or Asynchronous Server Gateway Interface(ASGI) with Apache since most Web servers do not natively speak Python. Popularity would be important since it reflects on how big the community is and how available the respective online resources are for the ease of development.

Table 2.3 Differences between Django, Flask and CherryPy Frameworks

Criteria	**Django**	**Flask**	**CherryPy**
Type	Full-Stack	Microframework	Microframework
Programming Paradigm	Object-oriented	Not Object-oriented	Object-oriented
Templating Engine	Built-in	Built-in (Jinja2)	No built-in templating engine
Popularity	Popular (67.1k stars on Github)	Popular (61k stars on Github)	Not popular (1.6k stars on Github)
Database Support	Built-in (ORM framework, PostgreSQL, MariaDB, MySQL, Oracle, SQLite)	None - need to get external support through plugins or extensions	None - need to get external support through plugins or extensions
Authentication Features	Built-in authentication, authorization, account management and support for sessions	Provides support for cookie-based sessions only	Built-in implementation of HTTP Basic Access Authentication

Testing	Built-in support using Python's unittest framework	Built-in support using Python's unittest framework	No built-in. Done using WebTest and Nose
Security	Built-in protection against common attacks like CSRF, XSS and SQL injection.	No built-in CSRF protection	Modules are independent and employ data abstraction to reduce complexity

2.4.3 Database Management Systems

The backend of the application will require a database to store all the relevant pharmaceutical and inventory data. In order to access the database, a Database Management System (DBMS) will be used to communicate between the user interface, application and database. The two types of databases are relational and non-relational, which are also known as SQL and NoSQL and they store data in very different ways. Relational databases have a table-based schema which means they organise structured data into tables that relate to one another and thus show clear dependencies between different data points. Non-relational databases deal with unstructured data and can store data in many ways in the form of document, wide-column, key-value, and even graph.

NoSQL databases are scaled horizontally as opposed to SQL databases which are scaled vertically and each variation has their own pros and cons. With horizontal scaling, the number of servers dealing with a request is linear to the number of users in the database but with vertical scaling, everything runs on a single server so in the case that there is a server crash, a SQL database will have a point of failure and there will be downtime. However if a horizontal system has a server that crashes, the next server can pick up the slack so that there will not be downtime or a point of failure. While this makes the system resilient, the

drawback of a horizontal system is that with multiple servers, there is a need to balance the load evenly and since it cannot perform interprocess communication, a horizontal system has to do remote procedure calls which can be slow. Additionally, since the data is spread out in a horizontal system, data might not always be consistent. The following table compares three different but common DBMSs: SQLite3, PostgreSQL, and MongoDB.

Table 2.4 Differences between SQLite3, PostgreSQL, and MongoDB DBMSs.

Criteria	**SQLite3**	**PostgreSQL**	**MongoDB**
Python Support	Yes	Yes	Yes
Scaling	Vertical	Vertical	Horizontal
Database Type	SQL, serverless, file-based, relational	SQL, Object Relational	NoSQL, document-oriented
Implementation language	C	C	C++
SQL Support	Yes	Yes	Read-only SQL queries via the MongoDB Connector for BI
License	Public Domain	Open Source	Open Source
Server-Side Scripting	Yes	User defined functions in proprietary language or in common languages (Python,Perl, etc)	JavaScript
Concurrency for data manipulation	Yes	Yes	Yes
Django Embedding	Yes	No	No
Cloud compatibility	Yes	Yes	Yes
Supported Data Types	Structured, semi-structured	Structured, semi-structured, unstructured	Structured, semi-structured, unstructured

2.4.4 Python Libraries

In order to execute inventory forecasting to increase productivity and minimise wastage, Artificial Intelligence (AI) will be utilised via the use of Python. The reasoning behind this is that Python is open source and has a massive community around it so learning materials are easily accessible, it practices the OOP paradigm, the way it is written is concise and readable, and there are many extensive libraries dedicated to Machine Learning (ML) and AI.

The libraries that will be compared are Scikit-Learn and Keras. Scikit-Learn, or sklearn, is ML centric and is built on NumPy, SciPy and Matplotlib with features including model selection, classification, preprocessing, clustering and regression. Keras is a Deep Learning (DL) centric library built on top of Tensorflow and is more beginner-friendly. In the context of this project, it is yet to be determined whether ML or DL will produce the best results in terms of accuracy and compatibility to the software system.

2.5 Proposed Solution

For this project, a Python Web application with a WSGI interface will be developed with the aid of the Django Full-Stack framework. A relational database will be preferred for its ability to display clear dependencies between data points and for its vertical scaling system which ensures more data consistency and negates the need to do load balancing. Between the few DBMS options compared above, MySQL will be chosen due to personal

familiarity. For deploying the Web application, the Web Server Gateway Interface (WSGI) will be used in conjunction with XAMPP.

To allow users to manage different branches, the system will be designed to allow pharmacy administrators to view and manage all activities of all branches whereas lower tier account holders like branch pharmacist and branch pharmacy assistants will only be able to view, access and manage only their own branches.

In order to determine which method would be best to implement demand forecasting and prediction, an experiment will be run to compare both Machine Learning and Deep Learning techniques using the same data sets to determine which is most suitable in terms of accuracy, efficiency and cost metrics. This will be done using both Scikit-learn library for machine learning and Keras library for deep learning in Python. Python 3.9.17 will be used as it is stable and supports both Scikit-learn and Keras libraries.

Chapter 3: REQUIREMENTS

3.1 System Overview

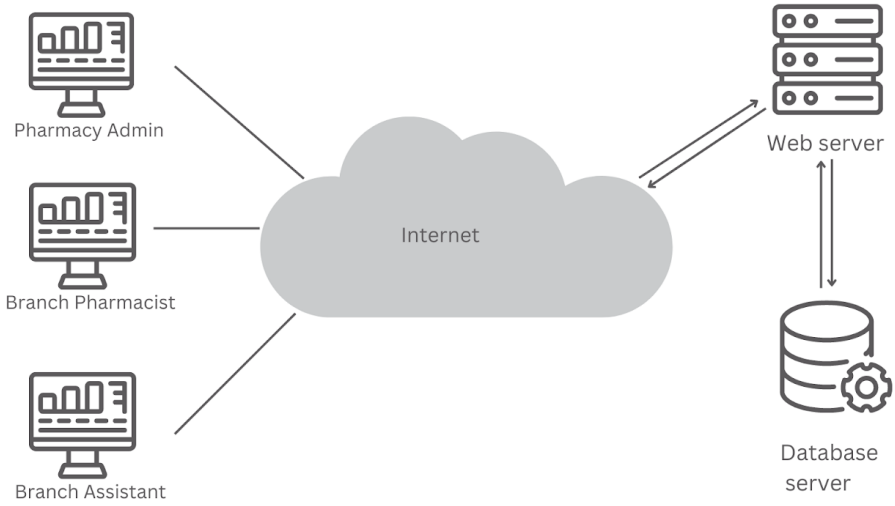

Figure 3.1 System Overview

Fig. 3.1 illustrates the proposed system for the project. The main users of the system are Pharmacy Admin, Branch Pharmacist, and Branch Assistant. All of them will be able to access the system through the Internet. All three roles will be able to log-in and logout of the system via the Internet. In this case, the reports refer to sales, sales analysis and inventory reports. Additionally, each pharmacy will have a centralised calendar and the notifications will contain action recommendations for the inventory.

The Pharmacy Admin will have the ability to view all the processes and access the whole database for the pharmacy regardless of branch location. In addition to that, this user's dashboard will reflect the notifications and current records such as sales and inventory of all pharmacy branches, and the admin can manage and add user roles/accounts. The Branch

Pharmacist can only access content in regards to their assigned branch, create and edit new patient and prescription records, view notifications, and access a calendar for appointments and orders.

The Branch Assistant can only access content in regards to their assigned branch. They can add and delete products in inventory, scan and generate barcodes for labelling, view overall pharmacy inventory, access a calendar for appointments and orders, manage suppliers, and order inventory. Additionally, they have integrated Point of Sales (POS) functionality and can manage checking out selected items within transactions.

Table 3.1 Actors and their respective Use Cases

Actor	**Use Cases**
Pharmacy Admin	● Manage branches ● View inventory ● Manage user roles ● View calendars ● View notifications ● View reports
Branch Pharmacist	● View reports ● Manage patient records ● Manage patient prescription ● View notifications ● Manage calendar
Branch Assistant	● Check out ● Generate product barcode ● Manage inventory ● Manage suppliers ● Order stock ● Update order status ● View notifications ● Manage calendar
User	● Log in ● Log out

3.2 System minimum requirements

3.2.1 Personal Computer

Requirement	**Minimum Versions**
Operating System	Microsoft Windows 10
CPU	1 GHz or faster processor or SoC (System on a Chip)
RAM	1 GB for 32-bit or 2 GB for 64-bit
Hard Disk Space	16 GB for 32-bit OS or 20 GB for 64-bit OS
Graphics Card	DirectX 9 or later with WDDM 1.0 driver
Display	800 x 600 resolution

3.2.2 Standalone server (Django Development Server)

Requirement	**Minimum Versions**
Operating System	Microsoft Windows 10
CPU	1 GHz
RAM	512 MB
Free Disk Space	200 MB
Django	v4.2.1
Python	v3.9.17

3.2.3 Web server (Gunicorn - WSGI)

Requirement	**Minimum Versions**
Operating System	Microsoft Windows 10
CPU	Dual Core
RAM	1GB
Free Disk Space	100 MB
Django	v4.2.1
Python	v3.9.17

3.3 Use Case Diagram

Figure 3.2 Overall Use Case Diagram

There are three main users namely Pharmacy Admin, Branch Pharmacist and Branch Assistant. The User actor represents the common functions shared between all three users, namely logging in and out of their respective accounts.

Table 3.2 Login Use Case Description Table

Use Case Name	Login
Actors	Pharmacy Admin, Branch Pharmacist, Branch Assistant
Description	This use case details how a user logs into the Retail Pharmacy Inventory System.
Pre-Conditions	None
Main Scenario	1. System shows a login form requesting the user to key in a

	username and password. 2. The actor enters a username and password. 3. The username and password credentials are validated with the system backend and the user is logged into the system.
Exceptions	If the actor enters an invalid username or password, the system will display an error message and prompt the actor to try again.

Table 3.3 Log Out Use Case Description Table

Use Case Name	Log Out
Actors	Pharmacy Admin, Branch Pharmacist, Branch Assistant
Description	This use case details how a user logs into the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system.
Main Scenario	1. The actor clicks the Log Out dropdown button on the navigation menu. 2. Actor is logged out of the system and returned to the login page.

3.3.1 Pharmacy Admin Modules

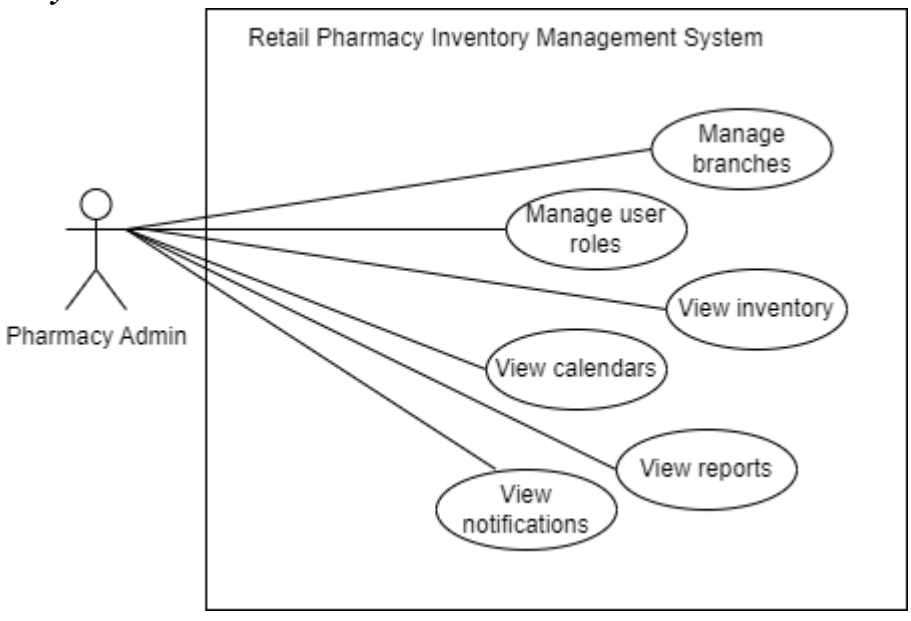

Figure 3.3 Pharmacy Admin Use Case Diagram

Pharmacy Admin denotes a user who is responsible for assigning roles within the system as well as oversee the analytics of the operations of all the given branches.

Table 3.4 Manage Branches Use Case Description Table

Use Case Name	Manage branches
Actors	Pharmacy Admin
Description	This use case details how a user manages the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Branches’ tab 2. The system redirects the actor to the ‘Branches’ page. 3. The system fetches branch data from the database to display. 4. Actor selects a specific branch. 5. The system redirects the actor to a new ‘Branch Information’ page 6. The system fetches data specifically regarding the selected branch and displays it. 7. Actor can choose to edit information and confirm changes. 8. System updates the new information into the database. 9. Actor chooses to delete branch. 10. System sends an alert prompting the actor to confirm or cancel deletion. a. Actor selects yes i. System corresponds with the backend to delete the row from the relevant database table ii. System redirects the actor to the main ‘Branches’ page with the new data updated b. Actor selects no i. The popup closes and the actor stays on the ‘Branch Information’ page. 11. Actor selects ‘Add New Branch’. 12. System redirects the actor to a “Add Branch” page which displays a form for the actor to fill up regarding the new branch details. 13. Actor fills in the form and confirms the new entry. 14. System sends the new data to add a new row into the database and redirects the actor to the main ‘Branches’ page. 15. The system displays the new and updated list of branches.

Table 3.5 Manage User Roles Use Case Description Table

Use Case Name	Manage User Roles
Actors	Pharmacy Admin
Description	This use case details how a user manages the user roles within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Roles’ tab 2. The system redirects the actor to the ‘Roles’ page. 3. The system fetches user data from the database to display. 4. Actor selects a specific user. 5. The system redirects the actor to a new ‘View User’ page 6. The system fetches data specifically regarding the selected user and displays it. 7. Actor chooses to delete user. 8. System sends an alert prompting the actor to confirm or cancel deletion. a. Actor selects yes i. System corresponds with the backend to delete the row from the relevant database table ii. System redirects the actor to the main ‘Roles’ page with the new data updated b. Actor selects no i. The popup closes and the actor stays on the ‘View User’ page. 9. Actor selects ‘Add New User’. 10. System redirects the actor to a “Add User” page which displays a form for the actor to fill up regarding the new user details. 11. Actor fills in the form and confirms the new entry. 12. System sends the new data to add a new row into the database and redirects the actor to the main ‘Roles’ page. 13. The system displays the new and updated list of user roles.

Table 3.6 View Inventories Use Case Description Table

Use Case Name	View Inventories
Actors	Pharmacy Admin
Description	This use case details how a user views the inventories of the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Inventories’ tab 2. The system redirects the actor to the ‘Inventories’ page. 3. Actor selects the branch whose inventory he wishes to view. 4. System fetches the data regarding the relevant branch from the database and displays it.

Table 3.7 View Calendars Use Case Description Table

Use Case Name	View Calendars
Actors	Pharmacy Admin
Description	This use case details how a user views the calendars of the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Calendars’ tab 2. The system redirects the actor to the ‘Calendars’ page. 3. Actor selects the branch whose calendar he wishes to view. 4. System fetches the data regarding the relevant branch from the database and displays it.

Table 3.8 View Reports Use Case Description Table

Use Case Name	View Reports
Actors	Pharmacy Admin
Description	This use case details how a user views the reports of the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Reports’ tab. 2. Actor selects the branch whose reports he wishes to view. 3. Actor selects the type of report he wishes to view. 4. System fetches the report data regarding the relevant branch from the database and displays it.

Table 3.9 View Notifications Use Case Description Table

Use Case Name	View Notifications
Actors	Pharmacy Admin
Description	This use case details how a user views the notifications of the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Notifications’ tab. 2. Actor selects the branch whose notifications he wishes to view. 3. System fetches the notifications from the database and displays them.

3.3.2 Branch Pharmacist Modules

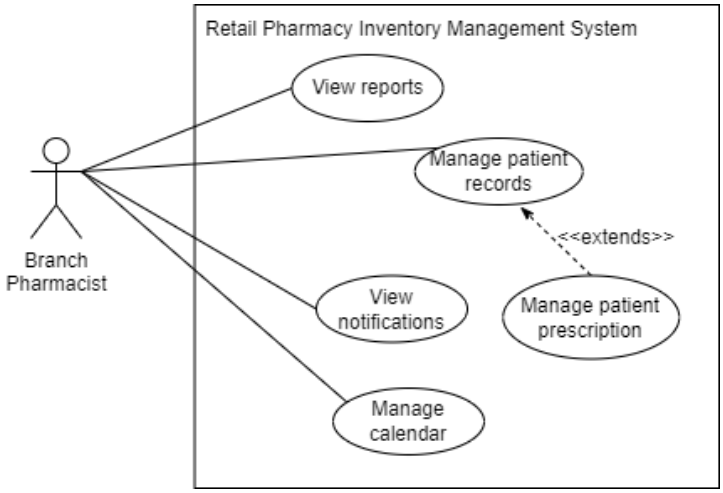

Figure 3.4 Branch Pharmacist Use Case Diagram

Branch Pharmacist denotes a user whose sole responsibility is making sure that patient prescription and medications are filled properly and in a safe manner. This user is qualified to recommend medication to patients which includes prescription medication and over-the-counter medication. As such, this user manages the patient records and their respective prescriptions.

Table 3.10 Pharmacist View Reports Use Case Description Table

Use Case Name	View Reports
Actors	Branch Pharmacist
Description	This use case details how a user views the reports of the branches within the Retail Pharmacy Inventory System. The reports generated by the system are in regards to inventory turnover, medication waste, sales, expenditure and supplier sourcing analysis. The reports will only be related to the pertaining branch.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the 'Reports' tab. 2. Actor selects the type of report he wishes to view. 3. System fetches the report data regarding the relevant branch from the database and displays it.

Table 3.11 Pharmacist View Notifications Use Case Description Table

Use Case Name	View Notifications
Actors	Branch Pharmacist
Description	This use case details how a user views the notifications of the branches within the Retail Pharmacy Inventory System. Receive and view notifications sent by the system regarding low stock, reports, near expiry batches, scheduled orders and appointment reminders. These notifications are only about the pertaining branch.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the 'Notifications' tab. 2. System fetches the notifications from the database and displays them.

Table 3.12 Manage Patient Records Use Case Description Table

Use Case Name	Manage Patient Records
Actors	Branch Pharmacist
Description	This use case details how the user manages the patient records within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the 'Patient Records' tab. 2. The system redirects the actor to the 'Patient Records' page. 3. The system fetches patient data from the database to display. 4. Actor selects 'Add New Patient'. 5. System redirects the actor to a "Add Patient" page which displays a form for the actor to fill up regarding the new patient details. 6. Actor fills in the form and confirms the new entry. 7. System sends the new data to add a new row into the database and redirects the actor to the main 'Patient Records' page. 8. The system displays the new and updated list of user roles. 9. Actor selects a specific patient. 10. The system redirects the actor to a new 'Display Patient' page 11. The system fetches data specifically regarding the selected patient and displays it. 12. Actor selects medical records for a specific patient. 13. The system fetches the medical record data from the backend and displays a list.

Table 3.13 Manage Patient Prescription Use Case Description Table

Use Case Name	Manage Patient Prescription
Actors	Branch Pharmacist
Description	This use case details how a user manages the patient's medical records/prescriptions within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system and navigate to a specific ‘Display Patient’ page from the ‘Patient Records’ page.
Main Scenario	1. Actor selects a specific medical record from the list. 2. The system fetches the specific medical record data from the backend and displays the information in a new page. 3. The actor selects ‘Add New Medical Record’. 4. The system redirects the actor to a ‘Add New Medical Record’ page which displays a form for the actor to fill up regarding the new record details. 5. Actor fills in the form and confirms the new entry 6. System sends the new data to add a new row into the database and redirects the actor to the main ‘Display Patient’ page.

Table 3.14 Manage Calendar Use Case Description Table

Use Case Name	Manage Calendar
Actors	Branch Pharmacist
Description	This use case details how a user manages the calendar of the respective branch within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Calendar’ tab. 2. System fetches the calendar data and displays it to the user. 3. Actor selects ‘Add Appointment’. 4. System navigates the actor to an ‘Add Appointment' page which has a form to fill up details such as time, day and name of the appointment. 5. Actor fills up the form and clicks confirm. 6. System sends the new data to the backend and redirects the user back to the ‘Calendar’ page with the newly updated calendar.

3.3.3 Branch Assistant Modules

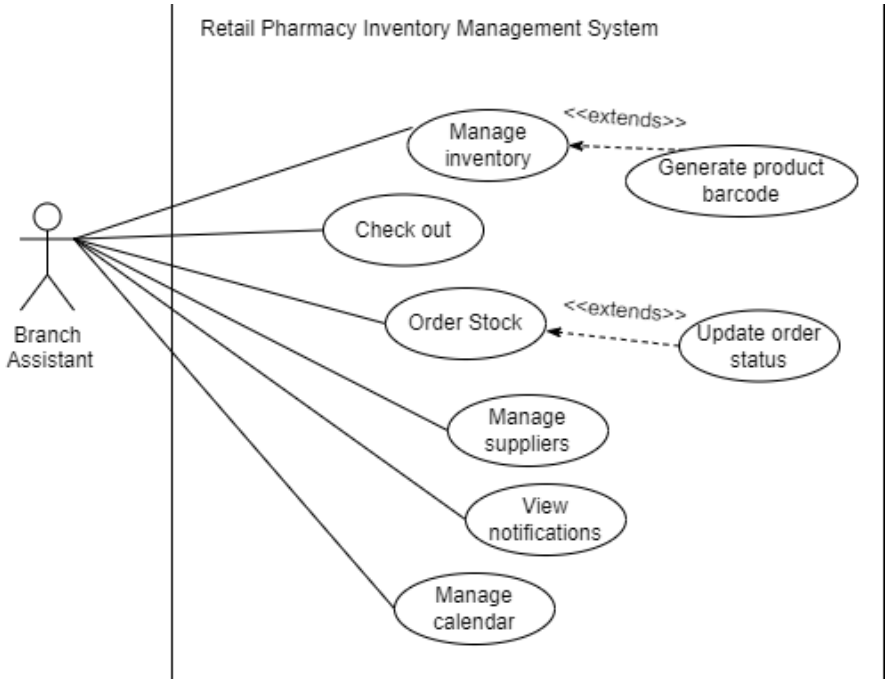

Figure 3.5 Branch Assistant Use Case Diagram

The Branch Assistant denotes a user that mainly assists with clerical work and customer service.

Table 3.15 View Notifications Use Case Description Table

Use Case Name	View Notifications
Actors	Branch Assistant
Description	This use case details how a user views the notifications of the branches within the Retail Pharmacy Inventory System. Receive and view notifications sent by the system regarding low stock, reports, near expiry batches, scheduled orders and appointment reminders. These notifications are only about the pertaining branch.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the 'Notifications' tab. 2. System fetches the notifications from the database and displays them.

Table 3.16 Manage Inventory Use Case Description Table

Use Case Name	Manage Inventory
Actors	Branch Assistant
Description	This use case details how a user manages the inventory of the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. The user navigates to the inventory page via the Inventory tab in the navigation bar. 2. The system fetches the relevant data from the database and displays it for the user. 3. The user selects ‘Add Product’. 4. The system redirects the user to a ‘Add Product’ page containing a form to fill up with new information for product record. 5. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new information will be updated and reflected in the table.

Table 3.17 Generate Barcode Use Case Description Table

Use Case Name	View Notifications
Actors	Branch Assistant
Description	Generate a scannable barcode for the product which translates to the product’s SKU code for easy stock tracking and selling purposes.
Pre-Conditions	Actor must be logged into the system and navigated to the Inventory page.
Main Scenario	1. Actor adds a new product 2. The system generates a unique 13 digit barcode. 3. The system redirects the actor to the Inventory page with a barcode and an option to download the barcode in png format.

Table 3.18 Manage Suppliers Use Case Description Table

Use Case Name	Manage Suppliers
Actors	Branch Assistant
Description	This use case details how a user manages the suppliers of the branch within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. The user selects the Supplier page from the navigation menu. 2. The system will display a table with the existing user roles information. 3. Upon selecting the supplier they wish to view, the system will redirect the user and display the relevant information to the edit and view page for the specific supplier. 4. From there, the user will be able to edit the email, rating, description and phone numbers for the selected supplier. 5. Upon clicking confirm changes, the system will send the information to be updated with the Supplier database. 6. Suppose the user clicks the 'Add New Supplier' button on the Suppliers page, he will be directed to a page containing a form to fill up with new information for the supplier record. 7. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new supplier information will be updated and reflected in the table.

Table 3.19 Order Stock Use Case Description Table

Use Case Name	Order Stock
Actors	Branch Assistant
Description	This use case details how a user lists a stock order of the branches within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. The user will select the Orders page from the navigation menu. 2. The system will display a table with the existing order information. 3. The user will be able to edit the current order status of the respective orders. 4. The updated order status will be sent over to the database and reflected in the table.

	5. Suppose the user clicks the ‘Add New Order’ button on the Roles page, he will be directed to a page containing a form to fill up with new information for the new order. 6. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new order information will be updated and reflected in the table.

Table 3.20 Update Order Status Use Case Description Table

Use Case Name	Update order Status
Actors	Branch Assistant
Description	This use case details how a user updates the order status for a stock order made within the branch of the system.
Pre-Conditions	Actor must be logged into the system and navigated to the Orders page.
Main Scenario	1. Actor clicks on the edit icon related to the order listing. 2. Actor changes the status. 3. System sends the new status update and displays the new listing with updated information.

Table 3.21 Manage Calendar Use Case Description Table

Use Case Name	Manage Calendar
Actors	Branch Assistant
Description	This use case details how a user manages the calendar of the respective branch within the Retail Pharmacy Inventory System.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor selects the ‘Calendar’ tab. 2. System fetches the calendar data and displays it to the user. 3. Actor selects ‘Add Appointment’. 4. System navigates the actor to an ‘Add Appointment' page which has a form to fill up details such as time, day and name of the appointment. 5. Actor fills up the form and clicks confirm. 6. System sends the new data to the backend and redirects the user back to the ‘Calendar’ page with the newly updated calendar.

Table 3.22 Check Out Use Case Description Table

Use Case Name	Checkout
Actors	Branch Assistant
Description	This use case shows how a user can checkout a customer's transaction.
Pre-Conditions	Actor must be logged into the system
Main Scenario	1. Actor navigates to the Checkout page. 2. The system will fetch product inventory data from the product database and display it for the user. 3. From there, the user can add specific items to the cart/customer order summary. 4. When the user is ready to checkout, he can click 'Checkout' and the transaction data will be sent to the sales database and sale detail database for tracking reasons.

3.4 Entity Relationship Diagram

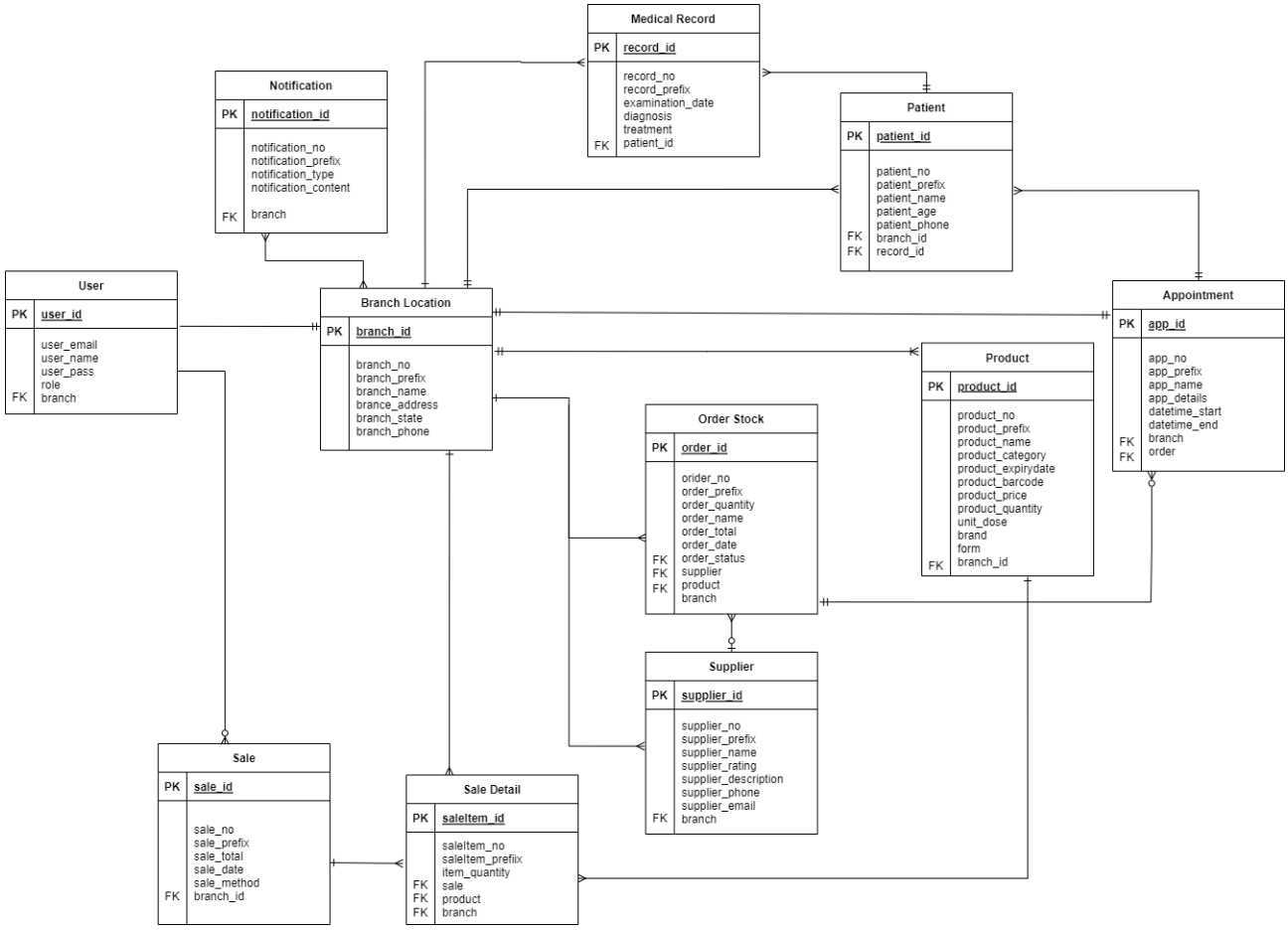

Figure 3.6 Entity Relationship Diagram

Figure 3.6 depicts the relationship between the data elements within the system. Starting from the User entity.

- Branch Location: Many Branch Pharmacists and Branch Assistants can manage exactly 1 Branch Location, which is the one they are assigned to. Pharmacy Admins, however, can manage all the Branch Locations.
- Inventory: Each branch location has exactly one inventory which consists of many products.

- Sale: Branch Assistants manage sales by checking out customers and beginning sale transactions.
- Sale Detail: Based on the sale transaction made and according to the sale_id, Sale Detail contains the data for the quantities of necessary products in the Product table.
- Product: Each product and their corresponding details are retrieved by Sale Detail and Order Stock entities when a sale or order is made.
- Order Stock: When stock is ordered, a listed supplier is called. One supplier can bring many products if ordered. When an order is made, an appointment is scheduled to know when the order is expected to arrive.
- Supplier: One supplier can sell many products.
- Patient: Each branch location has many patients respectively. These patients have prescriptions and can have appointments whether it is to pick up prescriptions or for consultation.
- Medical Records: Each patient has many patient records.
- Appointment: Appointments can be made in the form of patient appointments or stock orders. These are held by a single Calendar within their respective branch location.
- Notification: Different kinds of notifications can be generated for different branch locations.

Chapter 4: DESIGN

4.1 Software Architecture

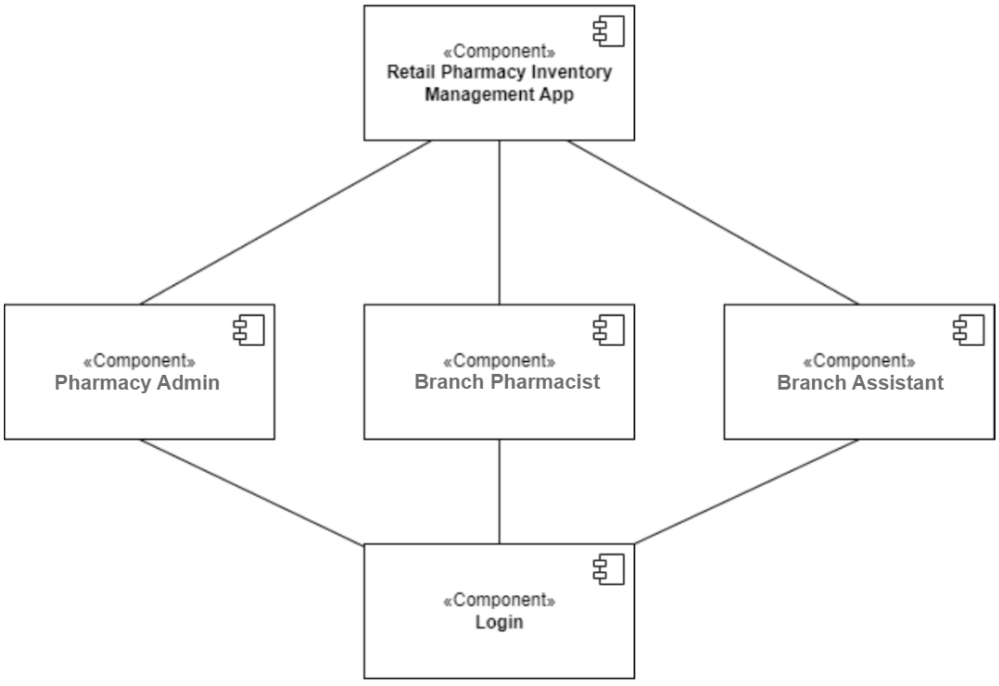

Figure 4.1 Overall Component Diagram

The Retail Pharmacy Inventory Management Application is separated into three main components, namely Pharmacy Admin component, Branch Pharmacist component and Branch Assistant component. These components are connected by a Login component which allows different user roles to access the RPIM application system.

4.1.1 Subsystem 1 (Pharmacy Admin)

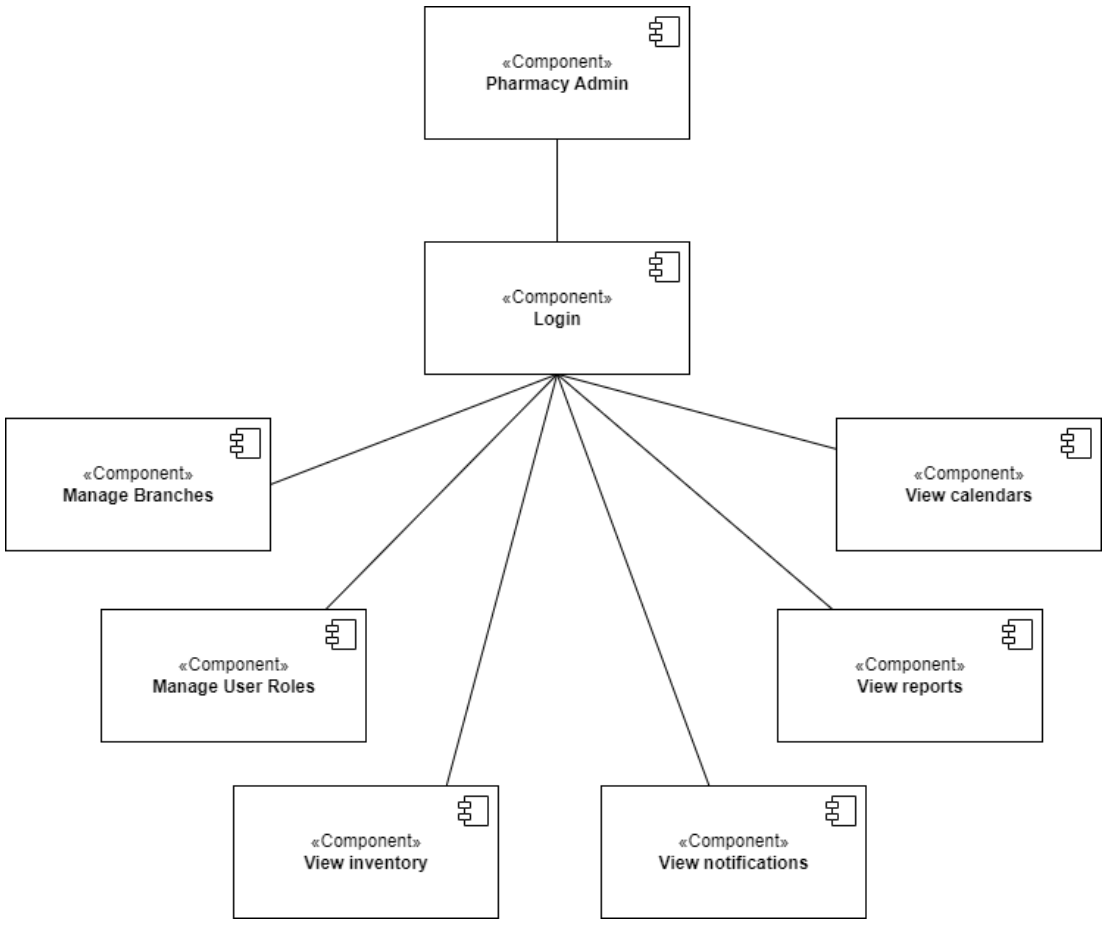

Figure 4.2 Subsystem 1 Component Diagram

Within Subsystem 1, the Pharmacy Admin component can access the rest of the components via the Login component. Beyond Login, the accessible components are Manage Branches, Manage User Roles, View Inventory, View Notifications, View Reports, and View Calendars.

4.1.2 Subsystem 2 (Branch Pharmacist)

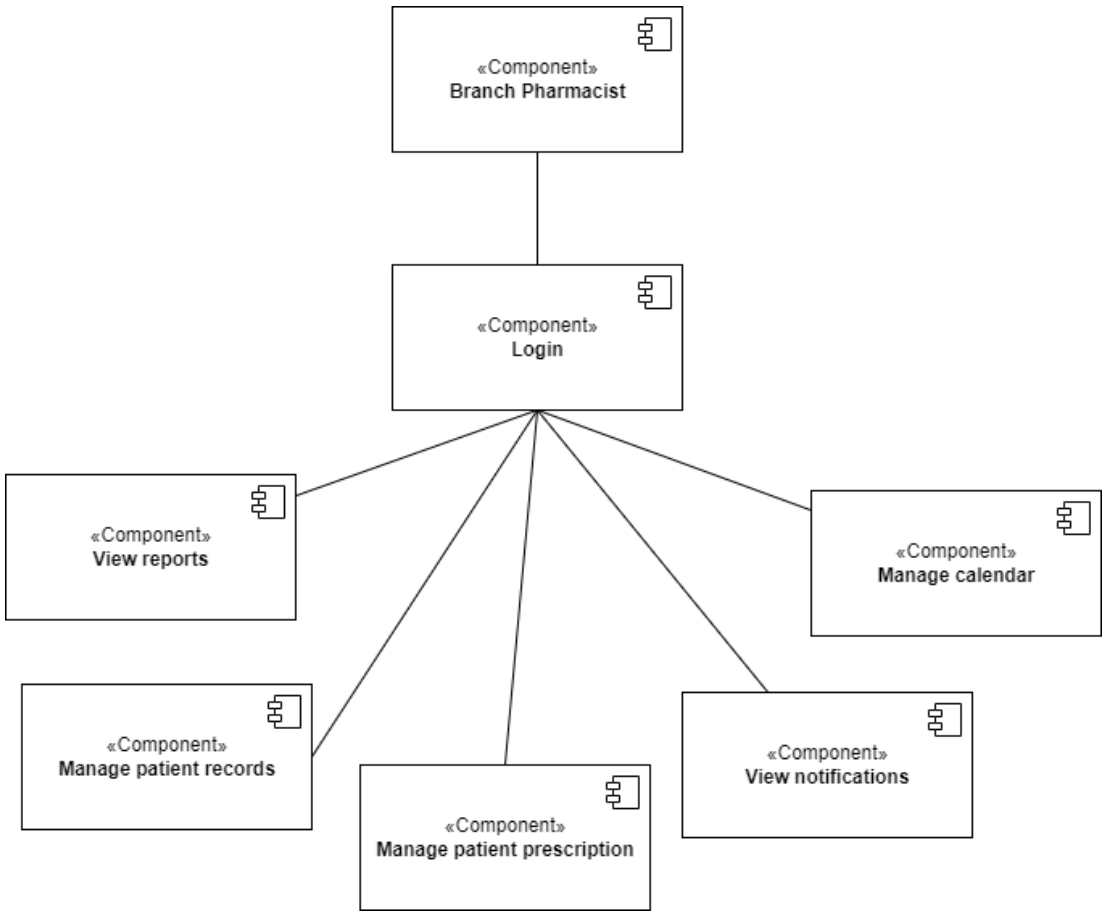

Figure 4.3 Subsystem 2 Component Diagram

Within Subsystem 2, the Branch Pharmacist component can access the rest of the components via the Login component. Beyond Login, the accessible components are View Reports, Manage Patient Records, Manage Patient Prescription, View Notifications, and Manage Calendar.

4.1.3 Subsystem 3 (Branch Assistant)

Figure 4.4 Subsystem 3 Component Diagram

Within Subsystem 3, the Branch Assistant component can access the rest of the components via the Login component. Beyond Login, the accessible components are Manage Inventory, Generate Product Barcode, Check Out, Order Stock, Update Order Status, Manage Suppliers, View Notifications and Manage Calendar.

4.2 Sequence Diagrams

4.2.1 Log In

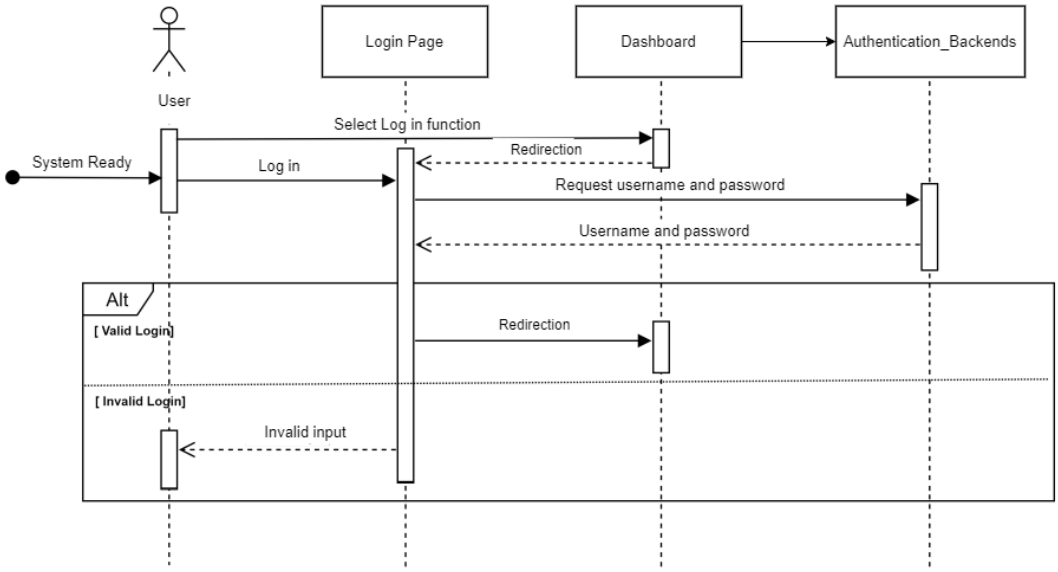

Figure 4.5 User Login Sequence Diagram

1. The User will log into the system at the login page.
2. The system will retrieve data from Authentication_Backends and compare Username and Password.
3. If the login details match the credentials, it is deemed valid and the system will redirect to the dashboard; if the details do not match, it is deemed invalid and the system is brought back to login.

4.2.2 Logout

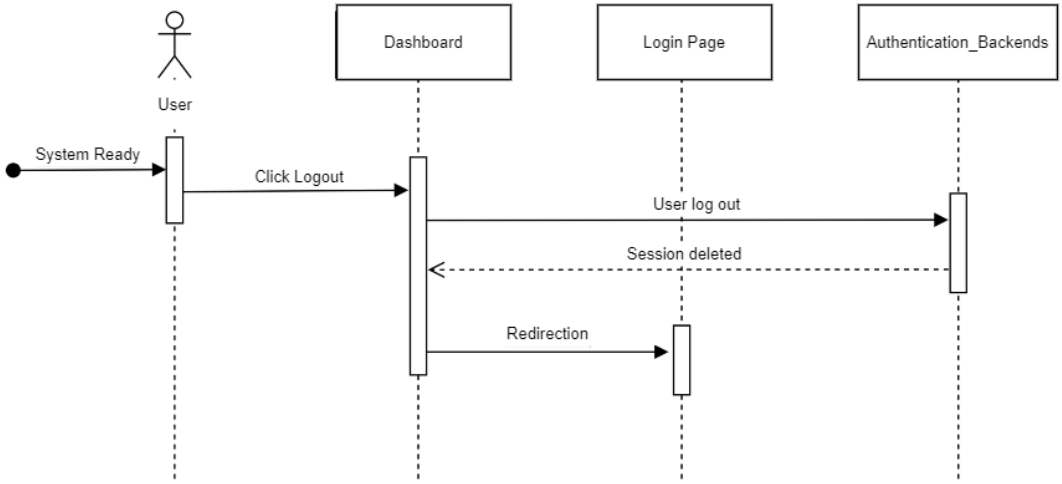

Figure 4.6 User Logout Sequence Diagram

1. The User will click logout at the dashboard page.
2. The system will then log out the User from the Authentication_Backends.
3. The system will delete the session and redirect to the login page.

4.2.3 View Notifications

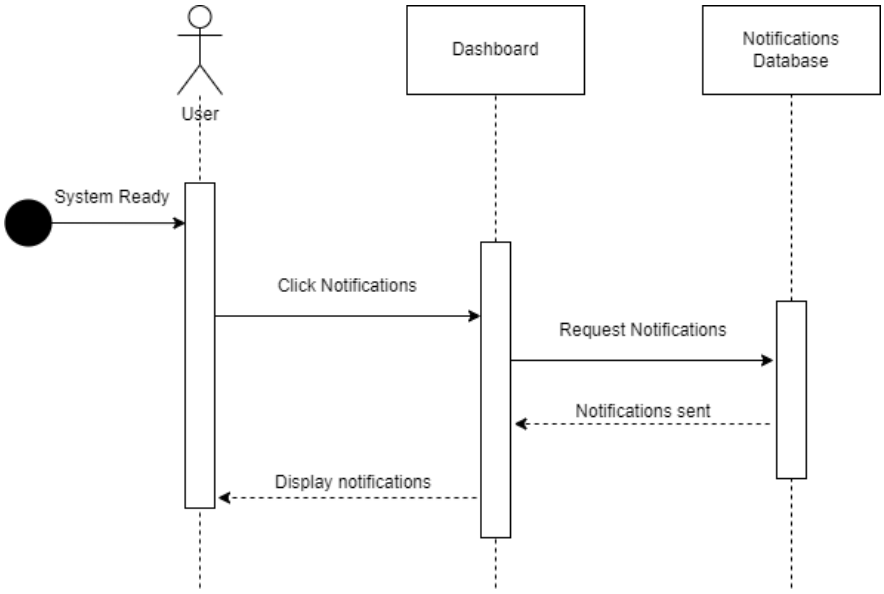

Figure 4.7 User Notifications Sequence Diagram

1. The user will click the Notifications tab on the dashboard.
2. The system will fetch the Notification data from the Notification Database and display the relevant notifications to the user.

4.2.4 View Reports (Pharmacy Admin and Branch Pharmacist)

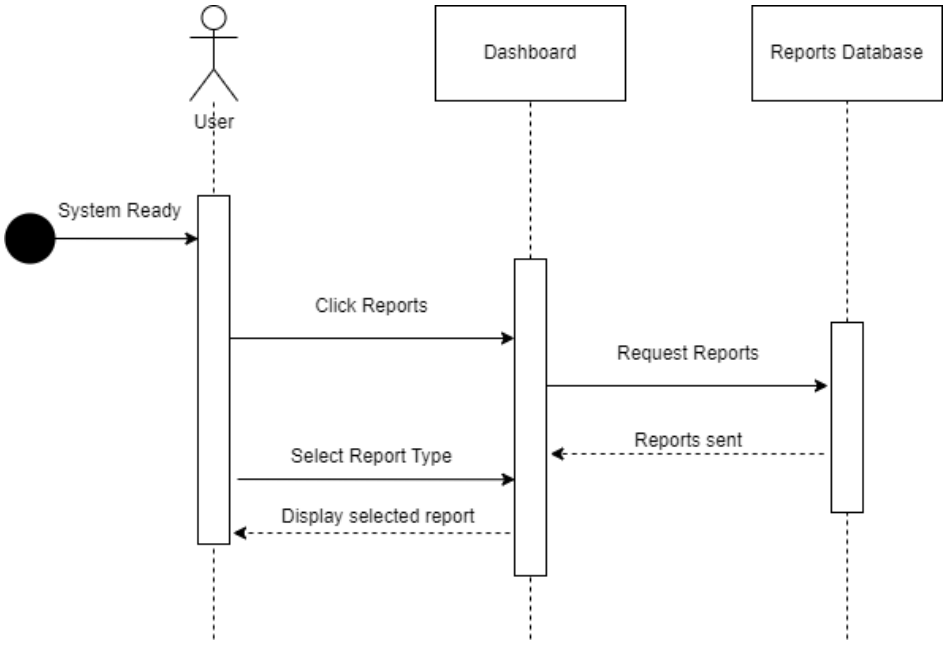

Figure 4.8 User Reports Sequence Diagram

1. The user will click the Report tab on the dashboard.
2. The system will fetch the Report data from the Report Database.
3. The user will select the type of Report he wishes to view and the system will display the relevant notifications to the user.

4.2.4 Manage Branches (Pharmacy Admin)

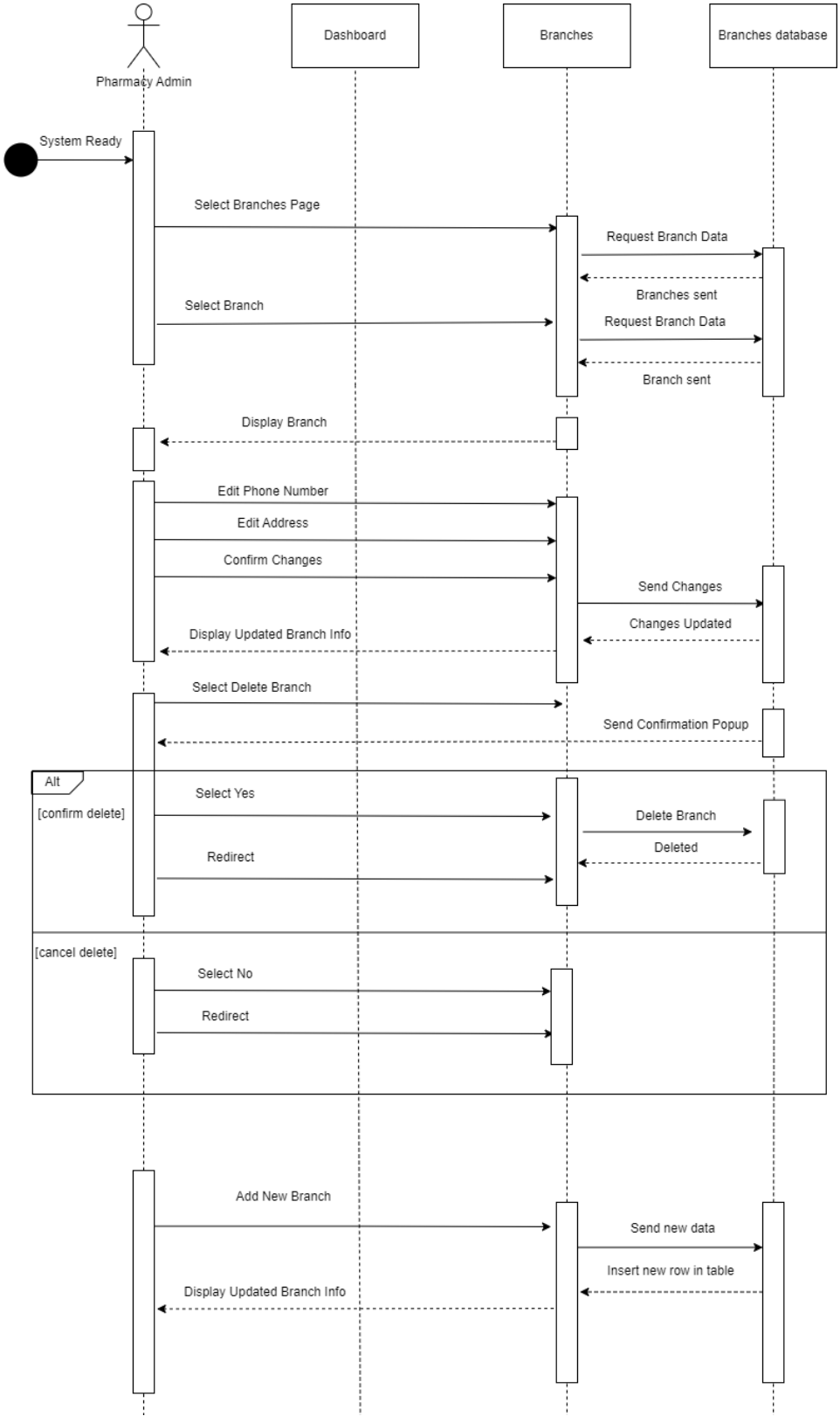

Figure 4.9 Branches Sequence Diagram

1. The user will click the Branches tab on the navigation menu.
2. The system will display a table with the existing branch information.
3. Upon selecting the Branch they wish to view, the system will redirect the user and display the relevant information to the view and edit page for the specific branch.
4. The user can edit information such as phone number and address. After which, they must click on ‘Confirm Changes’.
5. The system will then send the updated information to the database to replace the old data.
6. Suppose the user clicks the ‘Delete Branch’ button, the system will send a confirmation popup alert prompting the user to reconsider his choice to delete the branch in case clicking the button was a mistake.
7. If the user selects ‘Yes’, the system will delete the branch and redirect to the Branches page.
8. If the user selects ‘No’, the system will redirect and simply close the popup alert.
9. Suppose the user clicks the ‘Add Branch’ button on the Branches page, he will be directed to a page containing a form to fill up with new information for the new branch.
10. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new branch information will be updated and reflected in the table.

4.2.7 View Inventories (Pharmacy Admin)

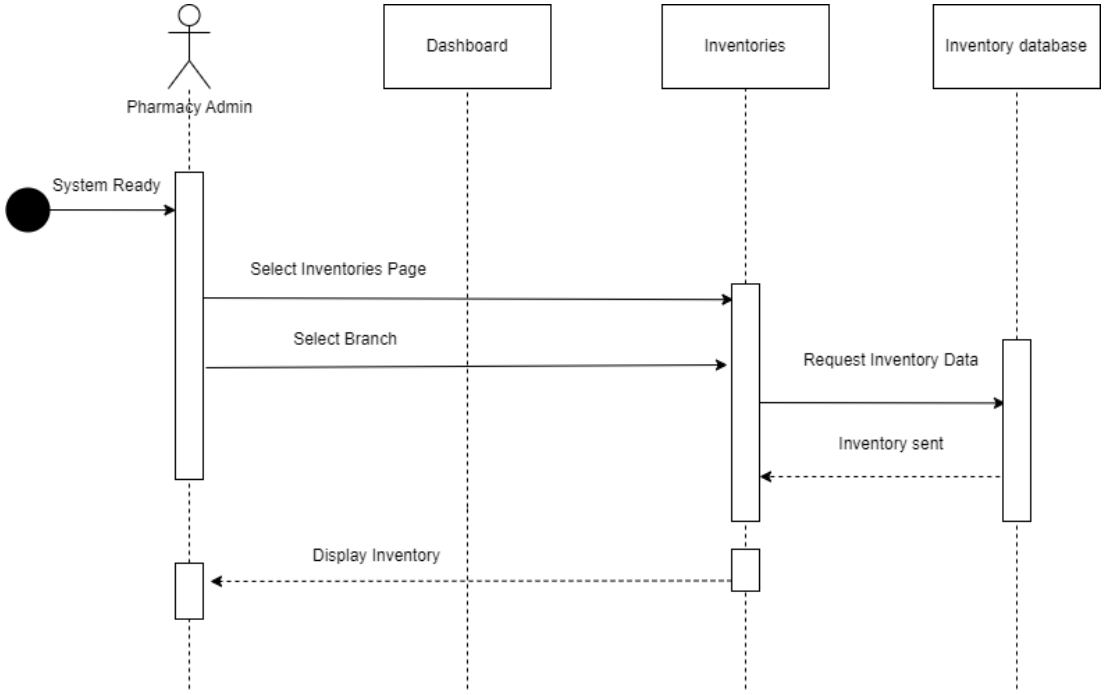

Figure 4.10 View Inventories Sequence Diagram

1. The Pharmacy Admin will select Inventories page from the Dashboard page.
2. Within the Inventories page, the Pharmacy Admin will determine which Branch's inventory he wishes to view.
3. Once the Branch is selected, the system will request inventory data from the Inventory Database and then the relevant information will be displayed.

4.2.8 Manage Roles (Pharmacy Admin)

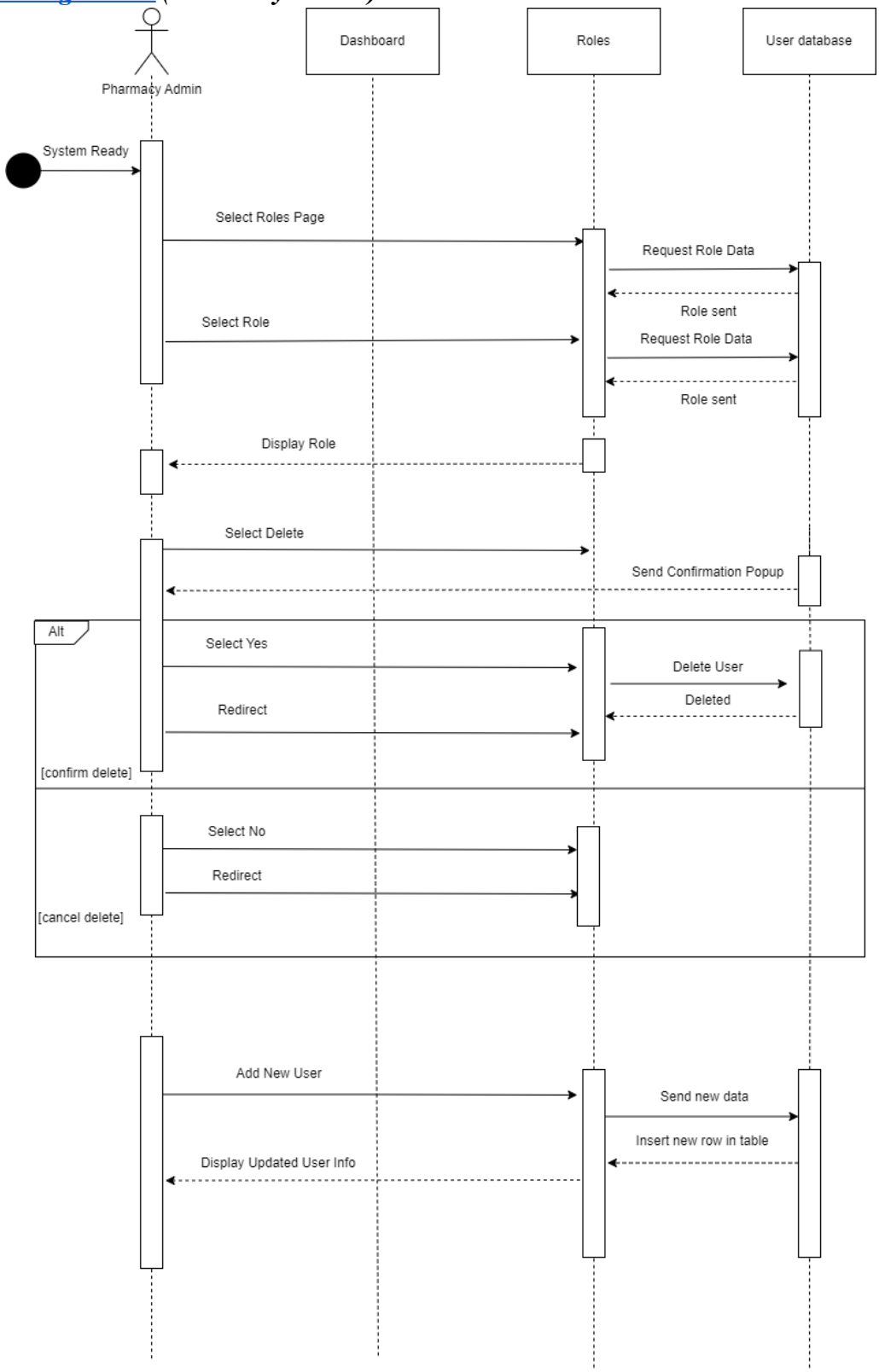

Figure 4.11 Manage Roles Sequence Diagram

1. The user will click the Roles tab on the navigation menu.
2. The system will display a table with the existing user roles information.
3. Upon selecting the user they wish to view, the system will redirect the user and display the relevant information to the view page for the specific user.
4. Suppose the user clicks the ‘Delete’ button, the system will send a confirmation popup alert prompting the user to reconsider his choice to delete the user in case clicking the button was a mistake.
5. If the user selects ‘Yes’, the system will delete the user and redirect to the Roles page.
6. If the user selects ‘No’, the system will redirect and simply close the popup alert.
7. Suppose the user clicks the ‘Add User’ button on the Roles page, he will be directed to a page containing a form to fill up with new information for the new user role.
8. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new user information will be updated and reflected in the table.

4.2.9 View Calendars (Pharmacy Admin)

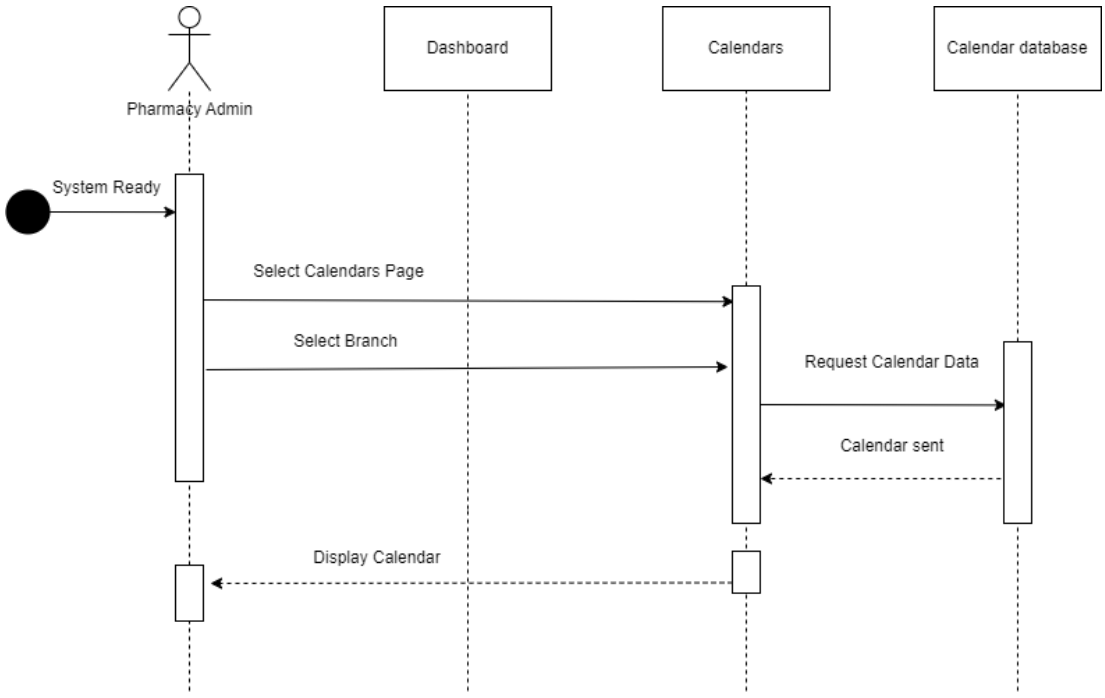

Figure 4.12 View Calendars Sequence Diagram

1. The user will select the Calendars tab in the navigation menu which directs the user to the Calendars page.
2. At the page, the user can select which branch whose calendar he wishes to view.
3. At which point the system requests the calendar data from the database and sends the corresponding information, displaying it to the user.

4.2.10 Manage Suppliers (Branch Assistant)

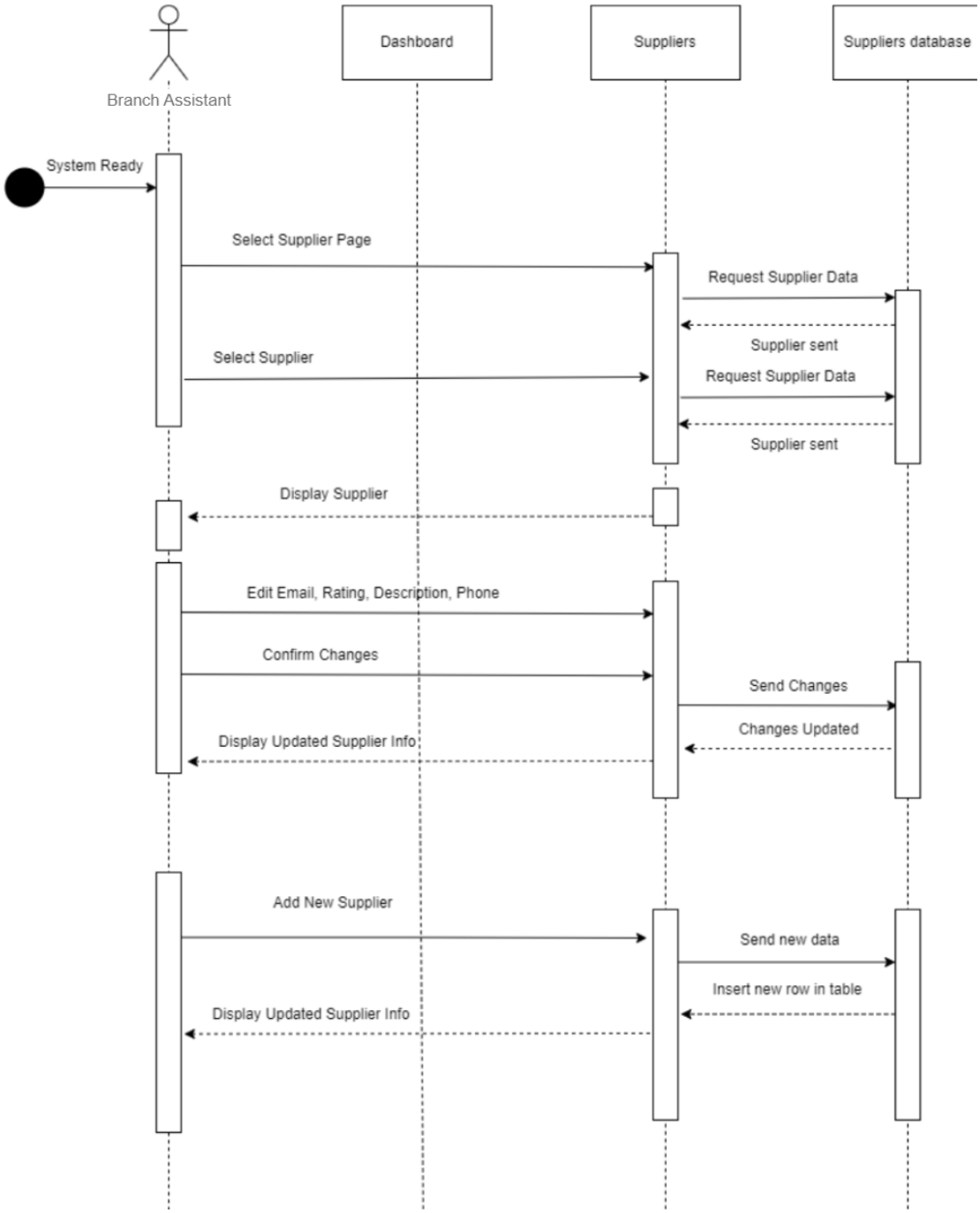

Figure 4.13 Manage Suppliers Sequence Diagram

8. The user selects the Supplier page from the navigation menu.
9. The system will display a table with the existing user roles information.
10. Upon selecting the supplier they wish to view, the system will redirect the user and display the relevant information to the edit and view page for the specific supplier.
11. From there, the user will be able to edit the email, rating, description and phone numbers for the selected supplier.
12. Upon clicking confirm changes, the system will send the information to be updated with the Supplier database.
13. Suppose the user clicks the ‘Add New Supplier’ button on the Suppliers page, he will be directed to a page containing a form to fill up with new information for the supplier record.
14. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new supplier information will be updated and reflected in the table.

4.2.11 Manage Patients (Branch Pharmacist)

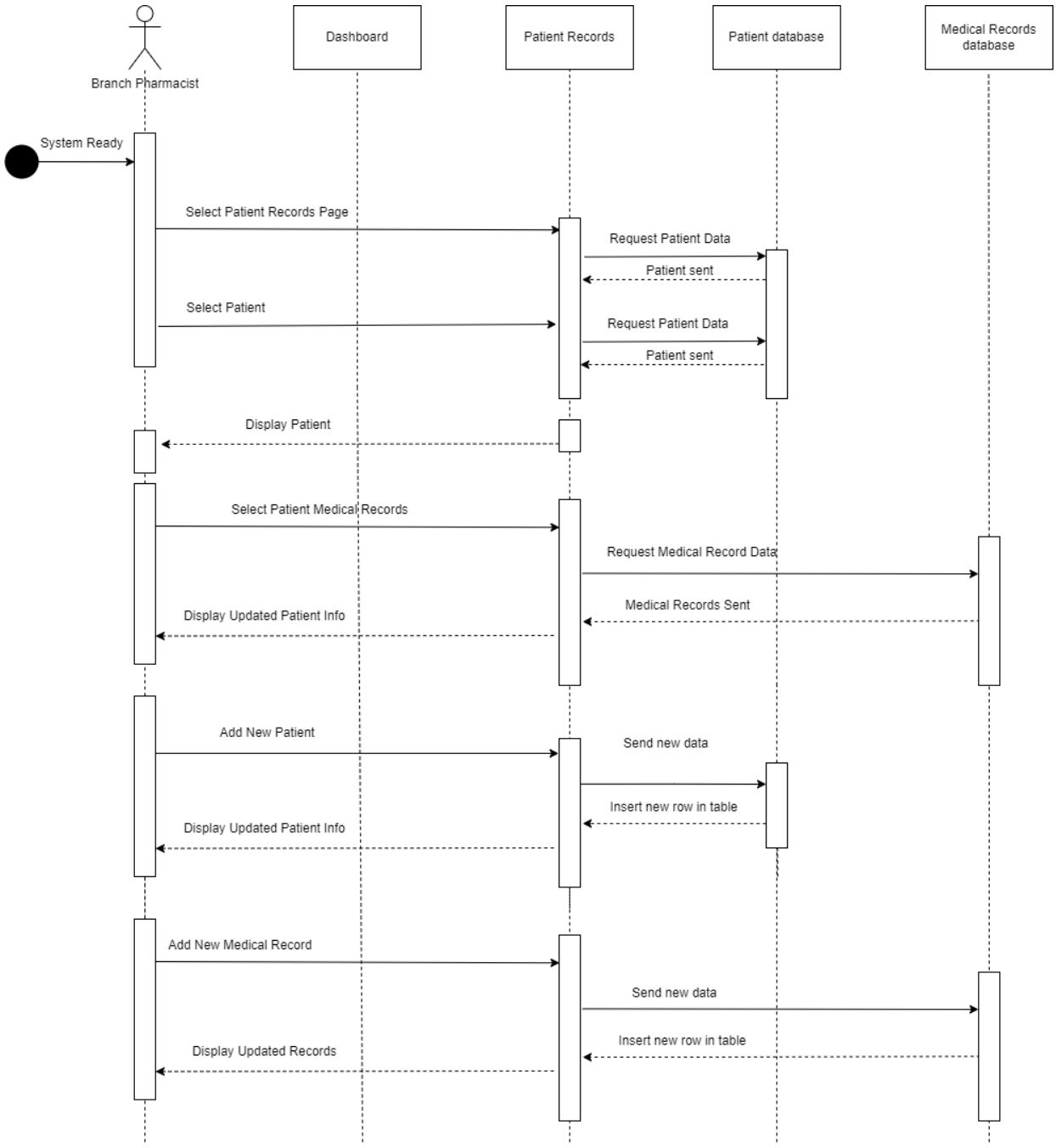

Figure 4.14 Manage Patients Sequence Diagram

1. The user will click on the Patient Records tab on the navigation menu.
2. The system will display a table with the existing patient information.
3. Upon selecting the patient they wish to view, the system will redirect to that specific user's Medical Records page which displays the table for that patient's medical records from the database.
4. The user can add a new medical record, fill in the important details within the form and press submit.
5. The new data will be sent to the database and the user will be redirected to the table where the updated rows will be displayed.
6. Within the Patient Records page, the user can also 'Add New Patient'. He will be directed to a page containing a form to fill up with new information for the new patient.
7. Upon clicking submit, the new data will be sent to the database and a new row will be created.
8. The new patient information will be updated and reflected in the table.

4.2.12 Manage Orders (Branch Assistant)

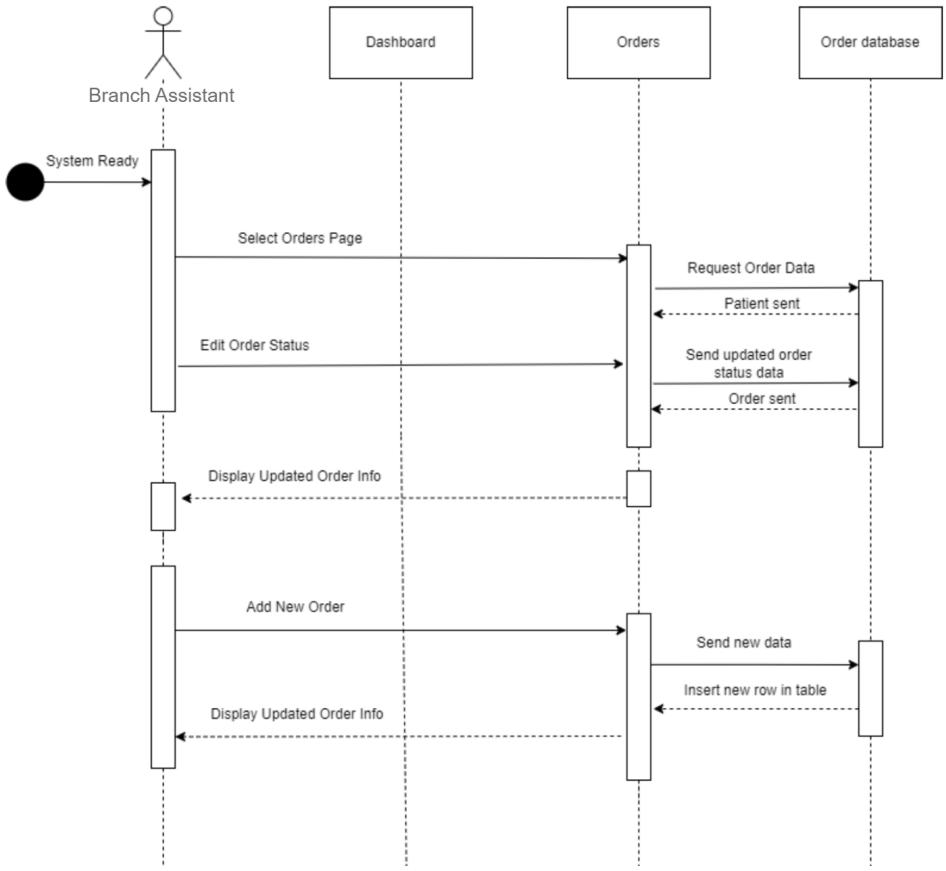

Figure 4.15 Manage Orders Sequence Diagram

7. The user will select the Orders page from the navigation menu.
8. The system will display a table with the existing order information.
9. The user will be able to edit the current order status of the respective orders.
10. The updated order status will be sent over to the database and reflected in the table.
11. Suppose the user clicks the ‘Add New Order’ button on the Roles page, he will be directed to a page containing a form to fill up with new information for the new order.
12. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new order information will be updated and reflected in the table.

4.2.13 Manage Calendar (Branch Pharmacist and Branch Assistant)

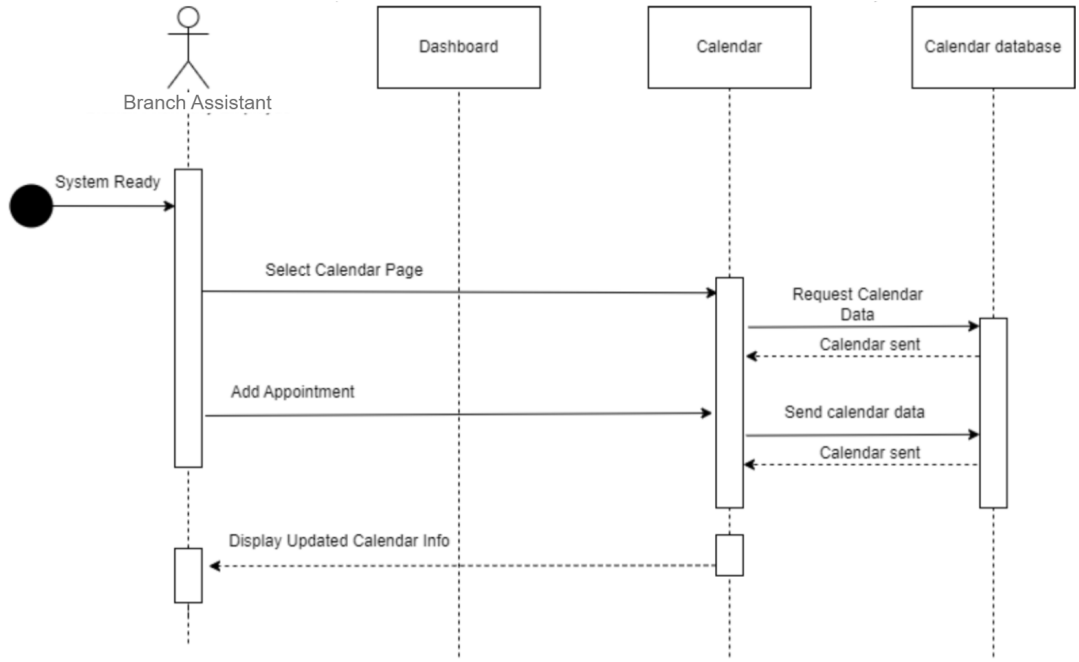

Figure 4.16 Manage Calendar Sequence Diagram

1. The user will select the Calendar tab on the navigation bar.
2. The system fetches the calendar data from the Calendar database and displays the information such as the day, month and appointments for the day.
3. The user can add appointments and the system will send the appointment data to the database then display the updated calendar.

4.2.15 Checkout (Branch Assistant)

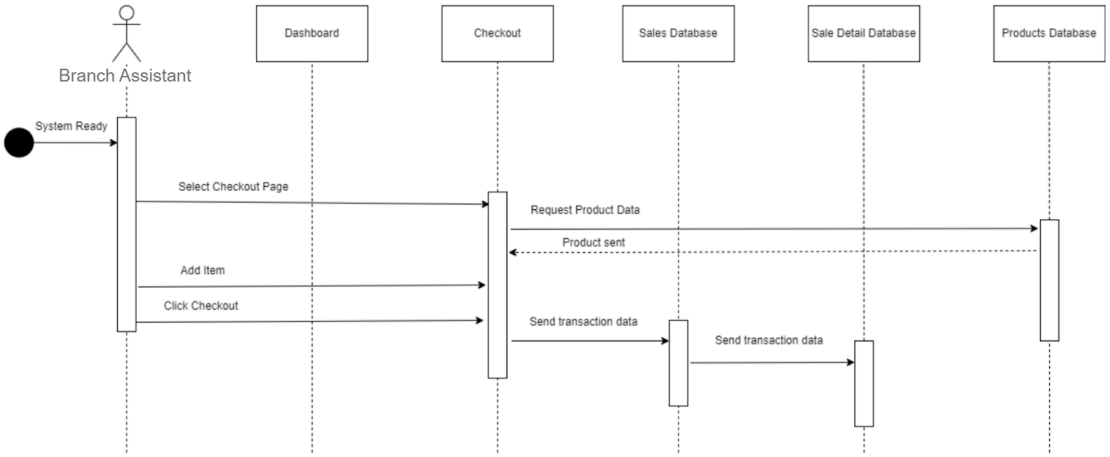

Figure 4.17 Checkout Sequence Diagram

1. The user selects the Checkout page on the navigation bar.
2. The system will fetch product inventory data from the product database and display it for the user.
3. From there, the user can add specific items to the cart/customer order summary.
4. When the user is ready to checkout, he can click ‘Checkout’ and the transaction data will be sent to the sales database and sale detail database for tracking reasons.

4.2.16 Manage Inventory(Branch Assistant)

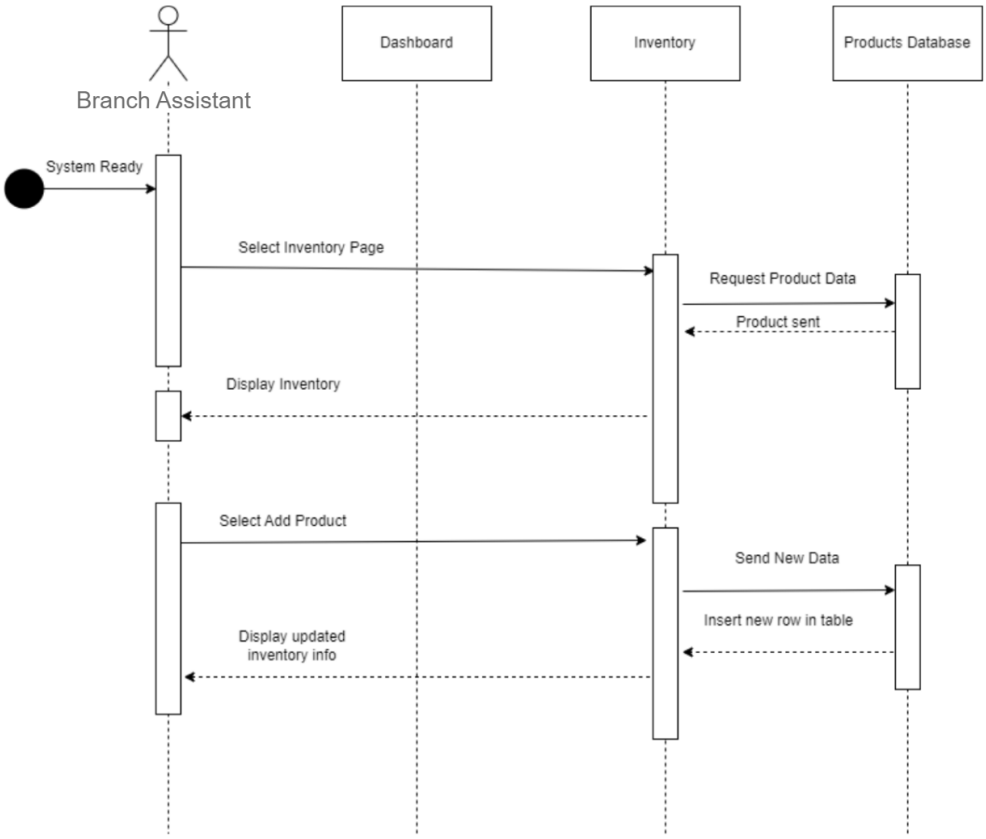

Figure 4.18 Manage Inventory Sequence Diagram

1. The user navigates to the inventory page via the Inventory tab in the navigation bar.
2. The system fetches the relevant data from the database and displays it for the user.
3. The user selects ‘Add Product’.
4. The system redirects the user to a ‘Add Product’ page containing a form to fill up with new information for product record.
5. Upon clicking submit, the new data will be sent to the database and a new row will be created. The new information will be updated and reflected in the table.

4.3 Screen Design

4.3.1 Login Screen

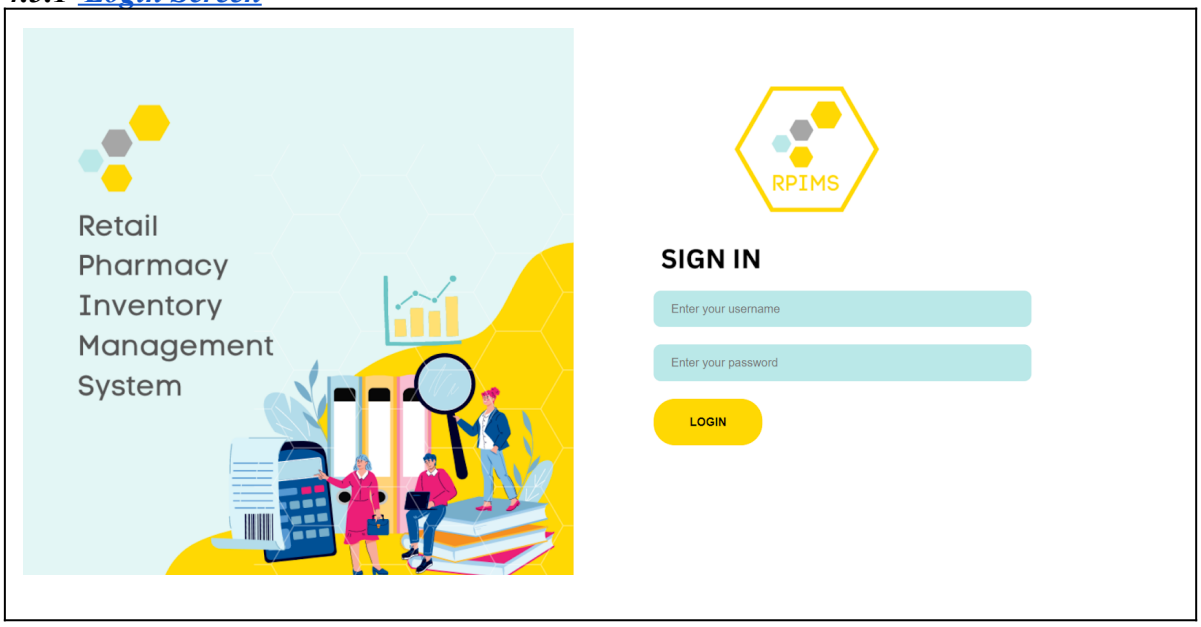

Figure 4.19 Sign In Screen

Figure 4.20 Sign In Exception

The Login Screen is the first thing that all users see before gaining access to the system. In order to login, users simply have to enter their email address and password then click Login.

If the credentials keyed in are invalid and do not match with the authentication backend, an error message will be displayed. If the credentials are valid, they proceed to the next screen, which is the user's assigned role's dashboard.

4.3.2 Dashboard - Notifications(Admin)

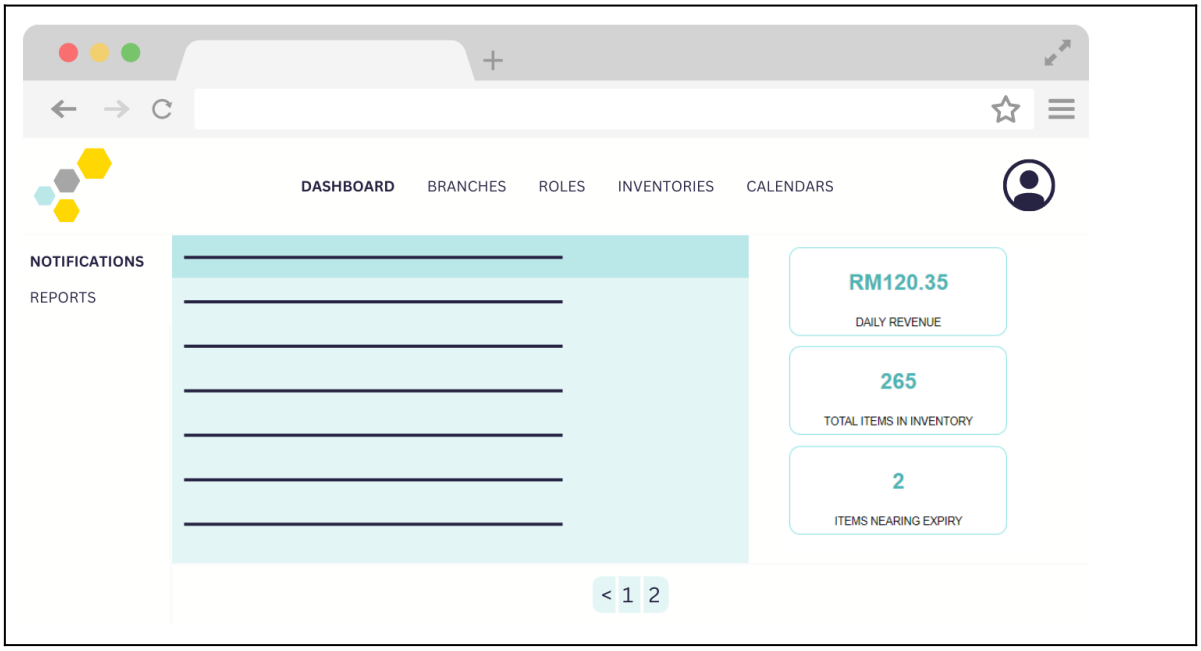

Figure 4.21 Admin Notification Screen

This is the dashboard for the Pharmacy Admin user. On the left, the user can switch between Notifications and Reports. Within the notification section, there are two containers: notifications, and important numerics for the branches.

4.3.3 Dashboard - Reports (Admin)

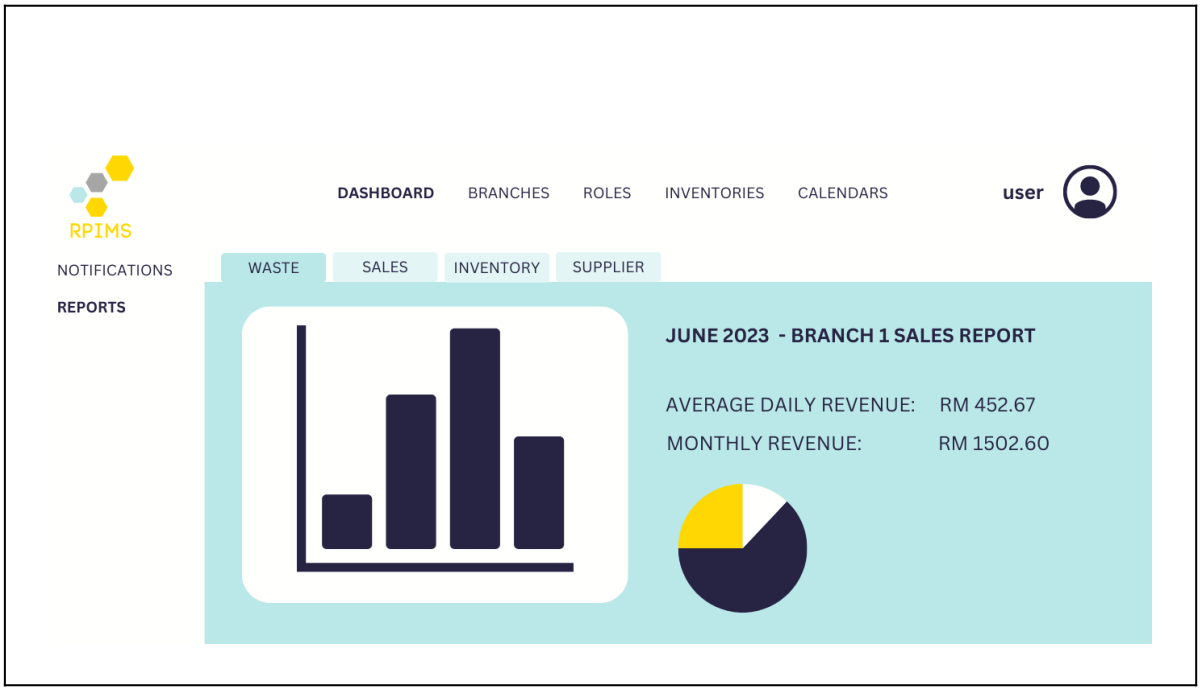

Figure 4.22 Admin Reports Screen

This is the dashboard for the Pharmacy Admin user. On the left, the user can switch between Notifications and Reports. Within the Reports section, there are different tabs such as Sales, Inventory, Waste and Supplier that contain reports for specific metrics. The sales reports include graph representation and sales history.

4.3.4 Branches (Admin)

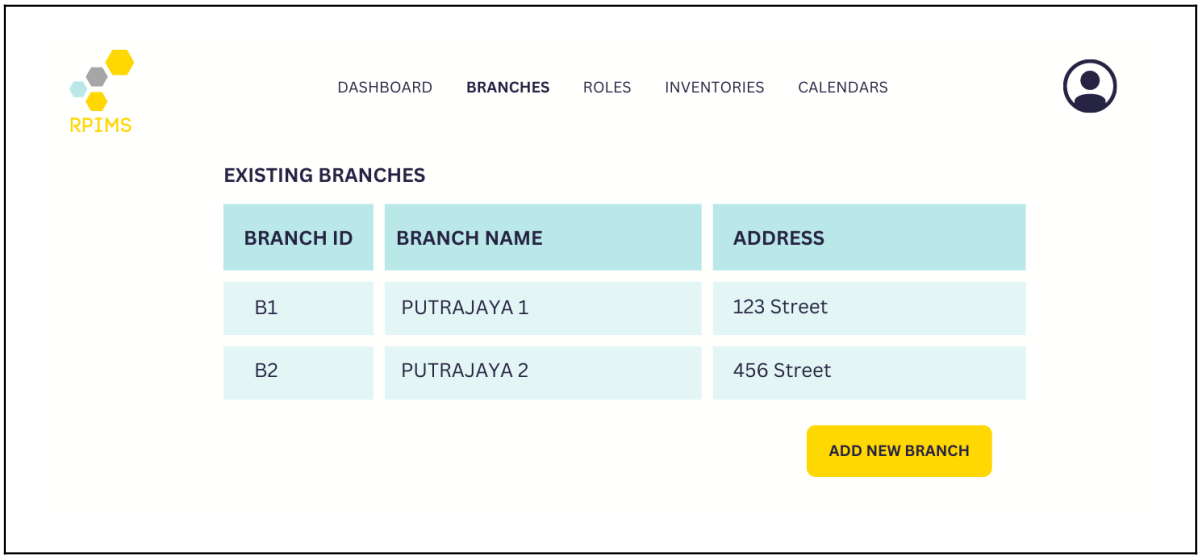

Figure 4.23 Admin Branches Screen

This is the Branches page for the Pharmacy Admin user. The page lists existing branches registered within the system as well as a button to provide the option of adding a new branch. Clicking on the branch will direct the user to an Edit Branch page.

4.3.5 Branches - Edit Branch (Admin)

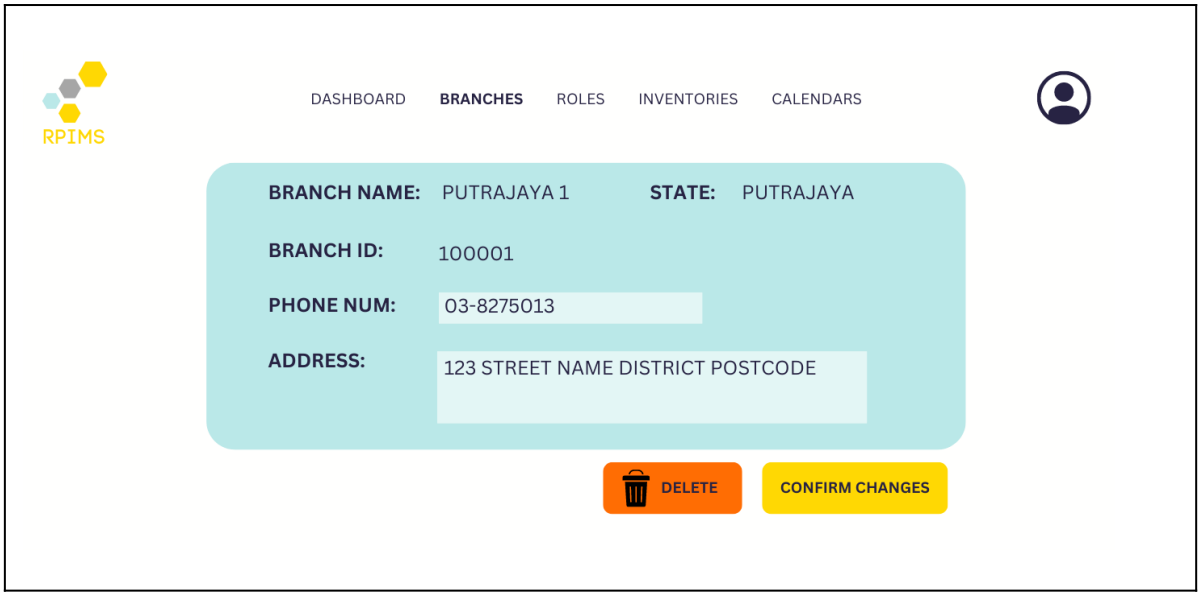

Figure 4.24 Admin Edit Branch Screen

This is the Edit Branch page for the Pharmacy Admin user, accessible through the Branches page. Here, the information for the selected branch is displayed, and information such as the phone number and address can be edited. Additionally, there is an option to delete the branch if ever there is a foreclosure.

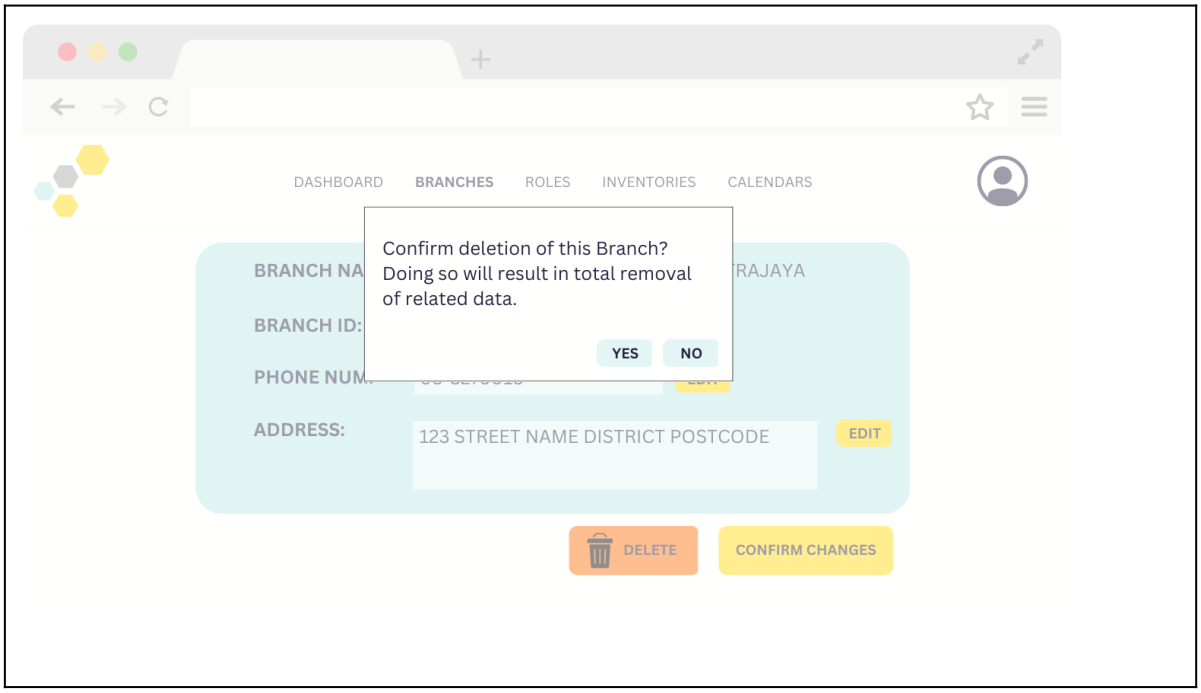

Figure 4.25 Admin Edit Branch Deletion Confirmation

Upon clicking ‘Delete’, a popup alert will be displayed, prompting the user to double confirm if they wish to proceed with the deletion of the branch record. If the user selects ‘yes’, the data is removed from the database and the user is redirected to the Branches page.

4.3.6 Branches - Add New Branch (Admin)

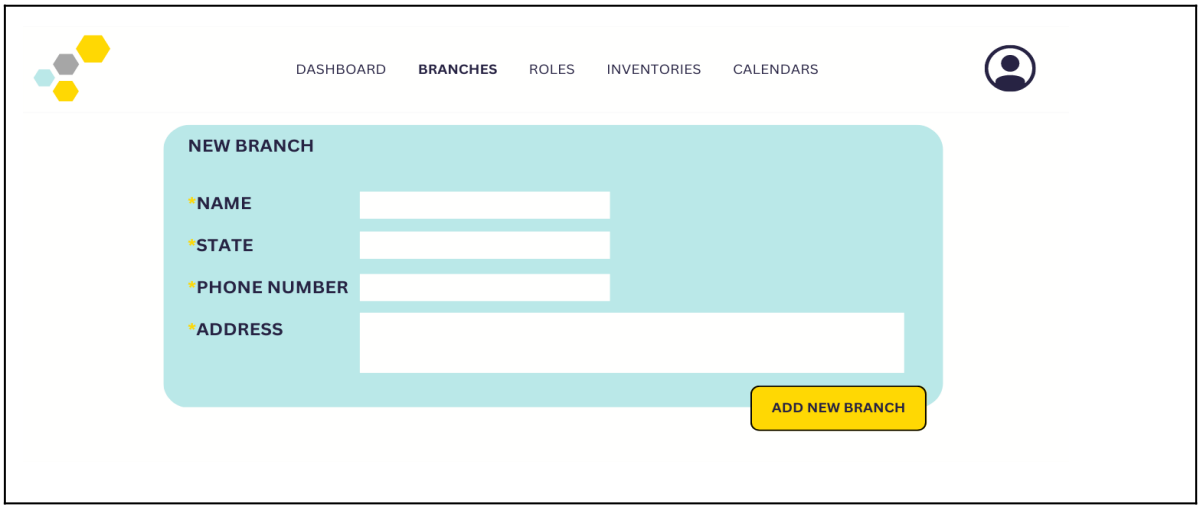

Figure 4.26 Admin Add New Branch Screen

This is the Add New Branch page. The admin user has to fill in all the details for the new branch, which are required fields before they can proceed. Upon clicking 'Add New Branch', the new information will be sent to the database and the user will be redirected to the Branches page where the list will be updated and display the new entry.

4.3.7 Roles (Admin)

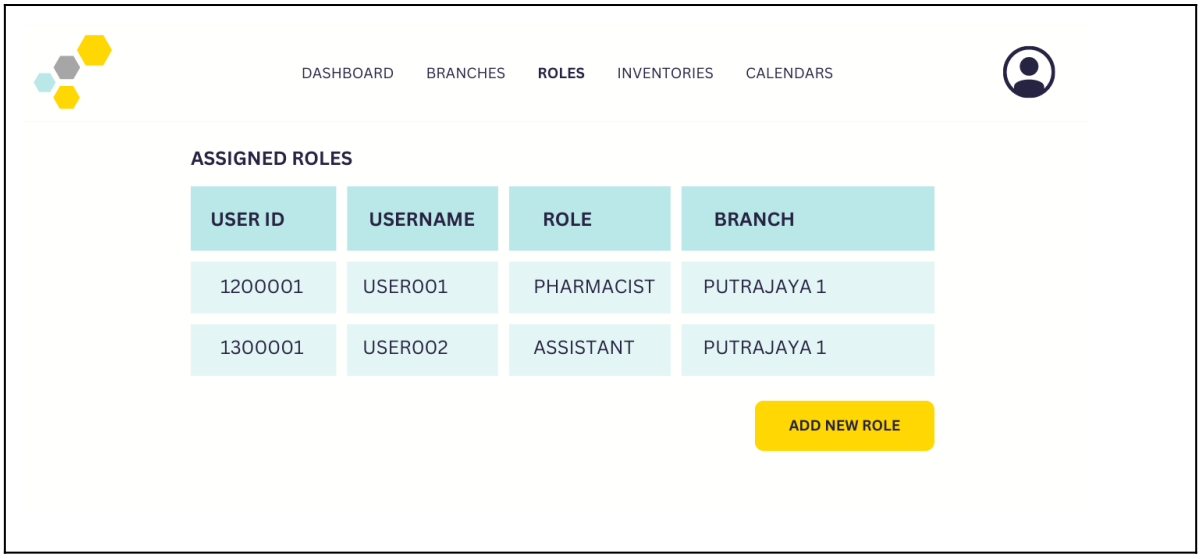

Figure 4.27 Admin Roles Screen

The Roles page displays the current Pharmacist and Assistant users that are registered within the system. Similarly to the Branches page, there is an ‘Add New Role’ button which takes the admin user to the ‘Add New User’ page. Additionally, upon clicking on the user ID, the admin user will be redirected to a ‘View User’ page containing the information on the pertaining user.

4.3.8 Roles - Add New User (Admin)

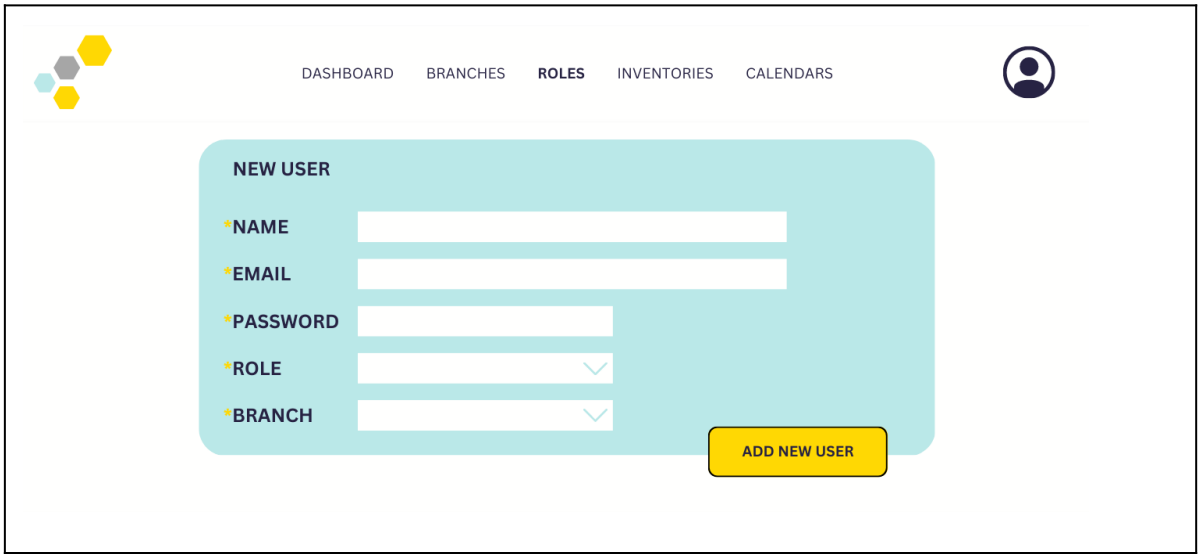

Figure 4.28 Admin Add New User Screen

This is the ‘Add New User’ page. The admin user has to fill in all the details for the new user, which are required fields before they can proceed. Upon clicking ‘Add New User’, the new information will be sent to the database and the user will be redirected to the Roles page where the list will be updated and display the new entry.

4.3.9 Roles - View User (Admin)

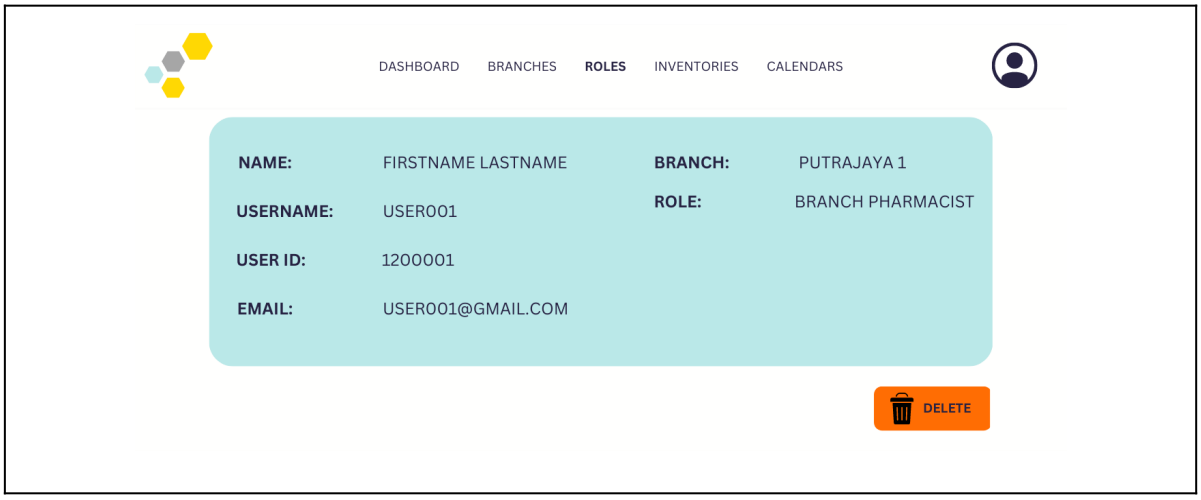

Figure 4.29 Admin View User Screen

This is the View User page that displays all the relevant information pertaining to the selected user. If the admin user clicks on delete, a popup alert will appear prompting the admin user to reconsider deletion of the user data. Upon confirmation, the user admin will be redirected to the Roles page which would be updated with the relevant information.

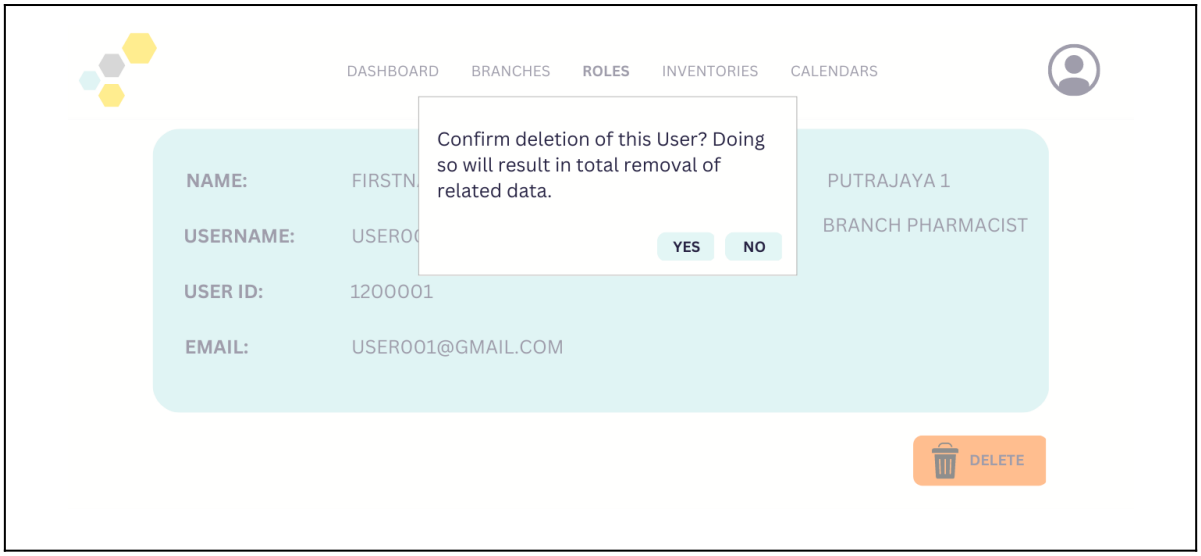

Figure 4.30 Admin Role Deletion Confirmation Screen

4.3.10 Inventories (Admin)

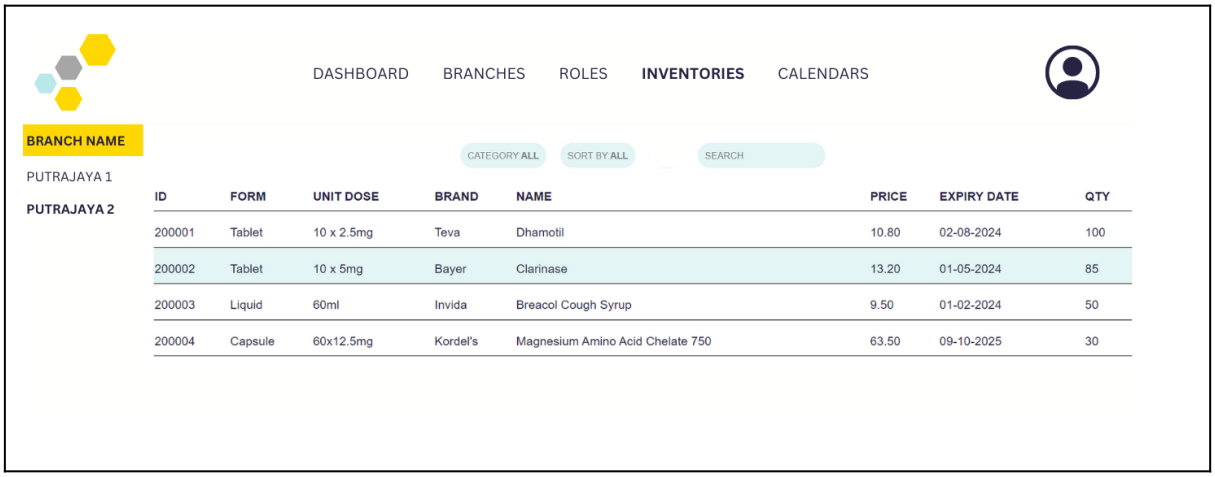

Figure 4.31 Admin Inventories Screen

The Inventories page for the admin user shows a left panel that lists the branches so the admin user can choose which inventory from which branch he wishes to view. Above the inventory table listing, there are two options to sort by category or by ID and expiration date. Additionally, there is a search bar for the user to search specific entries by keywords.

4.3.11 Calendars (Admin)

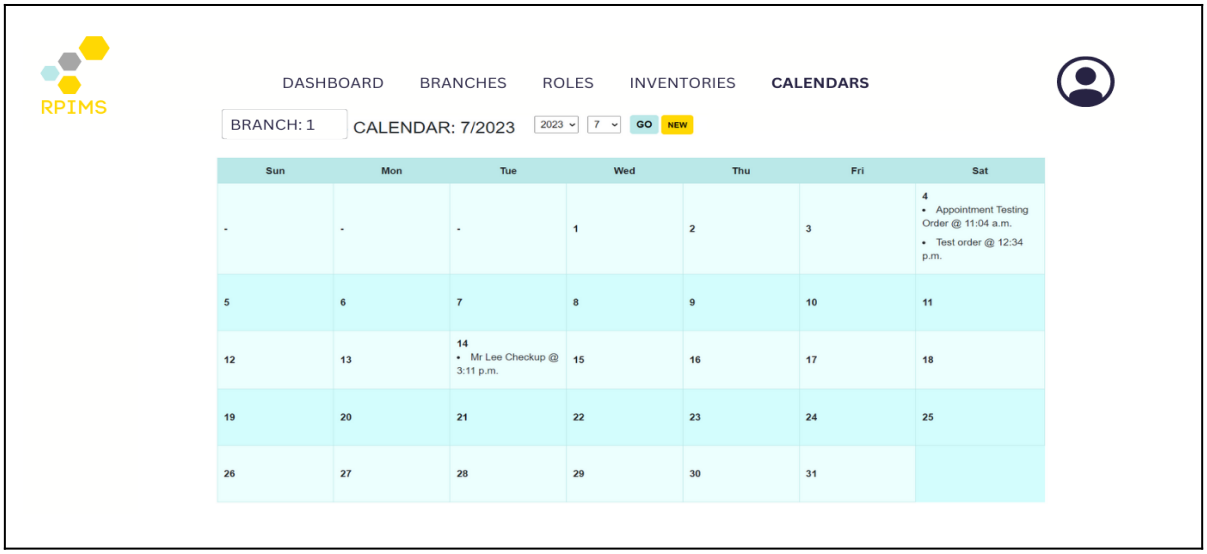

Figure 4.32 Admin Calendars Screen

The Calendars page for the admin user shows a left panel that lists the branches so the admin user can choose which calendar from which branch he wishes to view. The calendar will then display the current day and its set appointments for the day.

4.3.12 Dashboard - Notification (Branch Pharmacist)

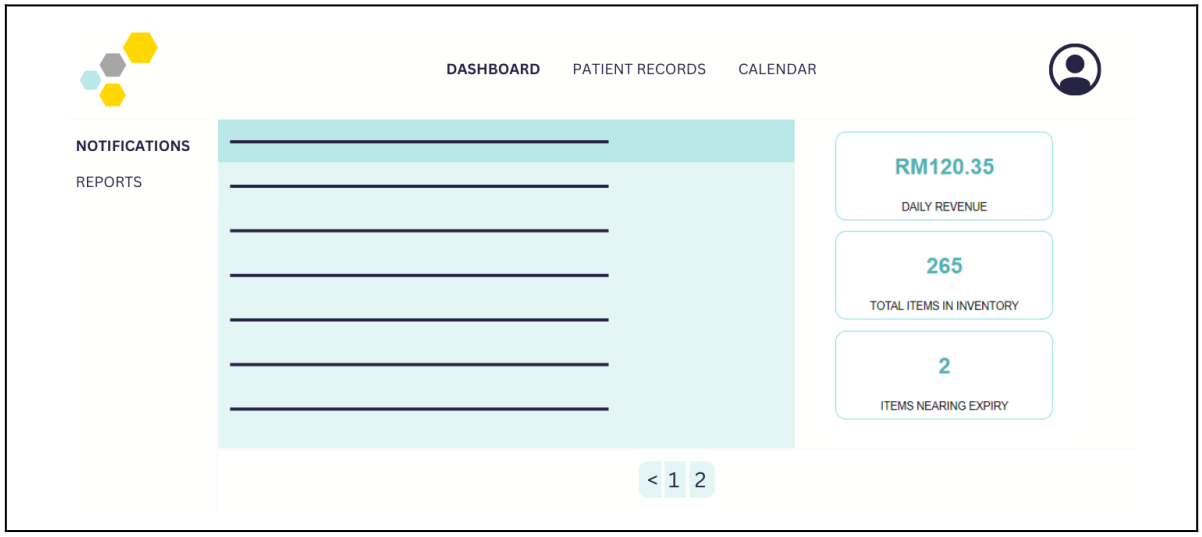

Figure 4.33 Pharmacist Notification Screen

This is the Dashboard for the Branch Pharmacist. The user has 3 main pages, as displayed in the navigation bar. On the left, the user can switch between Notifications and Reports. Within the notification section, there are two containers: notifications, and important numerics for the branches.

4.3.13 Dashboard - Reports (Branch Pharmacist)

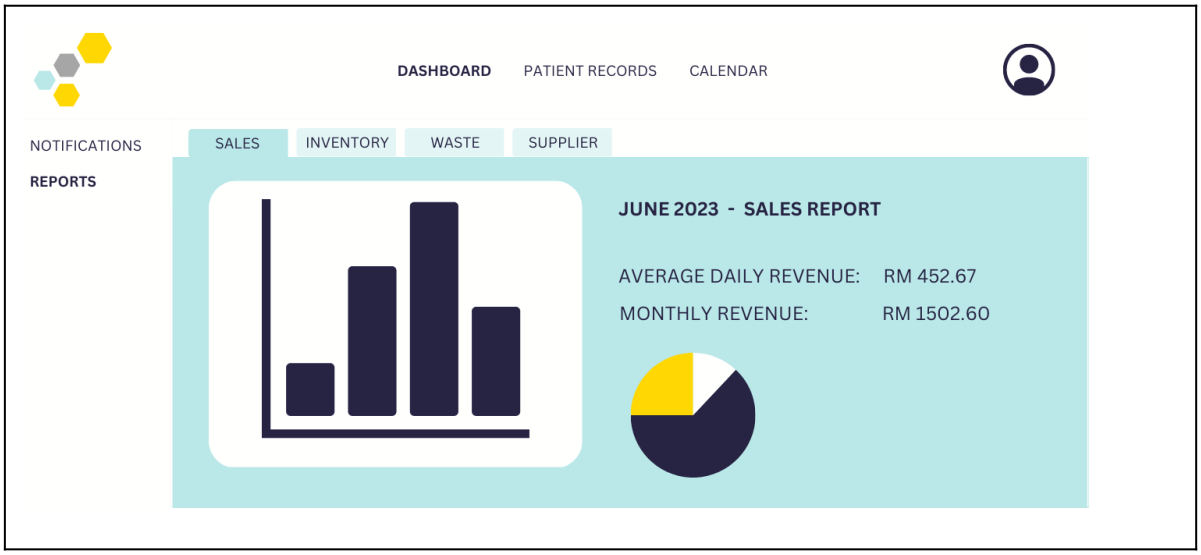

Figure 4.34 Pharmacist Reports Screen

This is the dashboard for the Branch Pharmacist user. On the left, the user can switch between Notifications and Reports. Within the Reports section, there are different tabs such as Sales, Inventory, Waste and Supplier that contain reports for specific metrics. The sales reports include graph representation and sales history.

4.3.14 Patient Records (Branch Pharmacist)

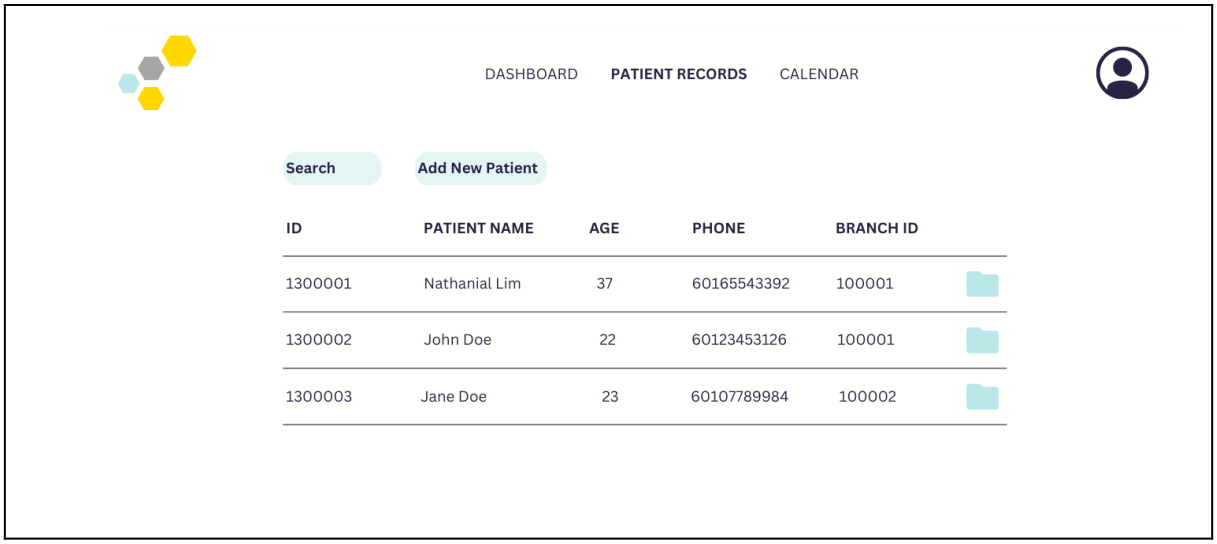

Figure 4.35 Pharmacist Patient Records Screen

This is the Patient Records page which displays the available patient data as per the relevant branch. There is a search bar to search patients with specific keywords, and an 'Add New Patient' which will redirect the user to an Add New Patient Page. Beside each listing, there is a folder icon that allows the pharmacist user to access the medical records of the pertaining patient.

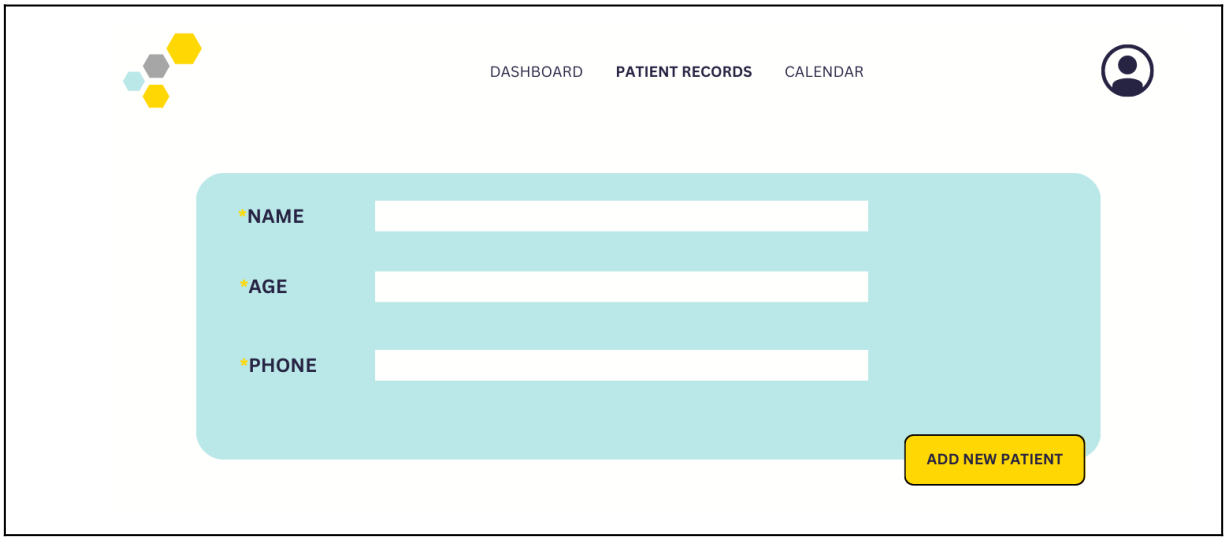

Figure 4.36 Pharmacist Add New Patient Screen

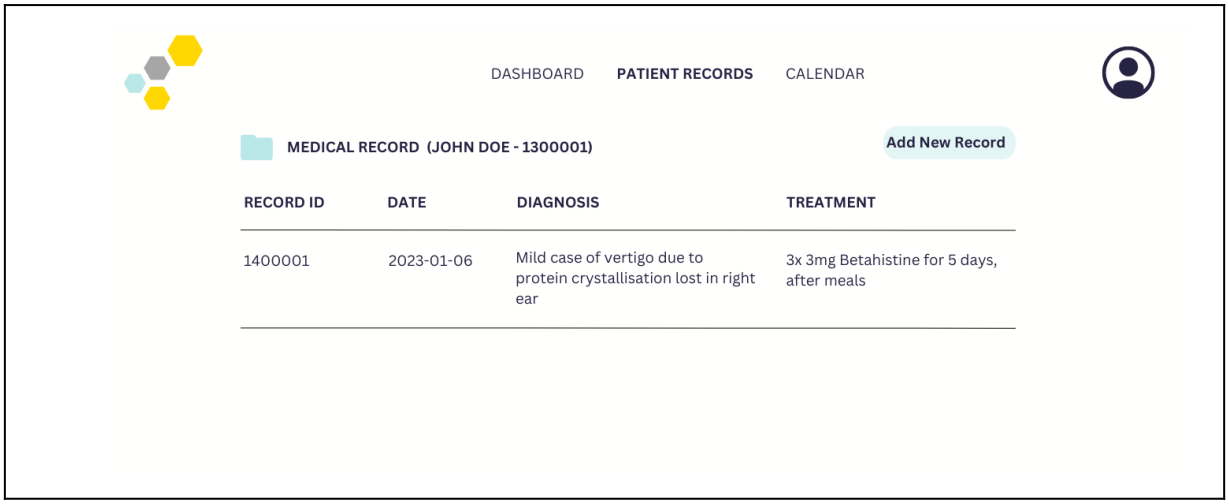

Figure 4.37 Pharmacist Medical Record Screen

Within the medical records page, there is a history list of available records as per the patient involved. The user can also click an ‘Add New Record’ button which redirects to another page to fill in details.

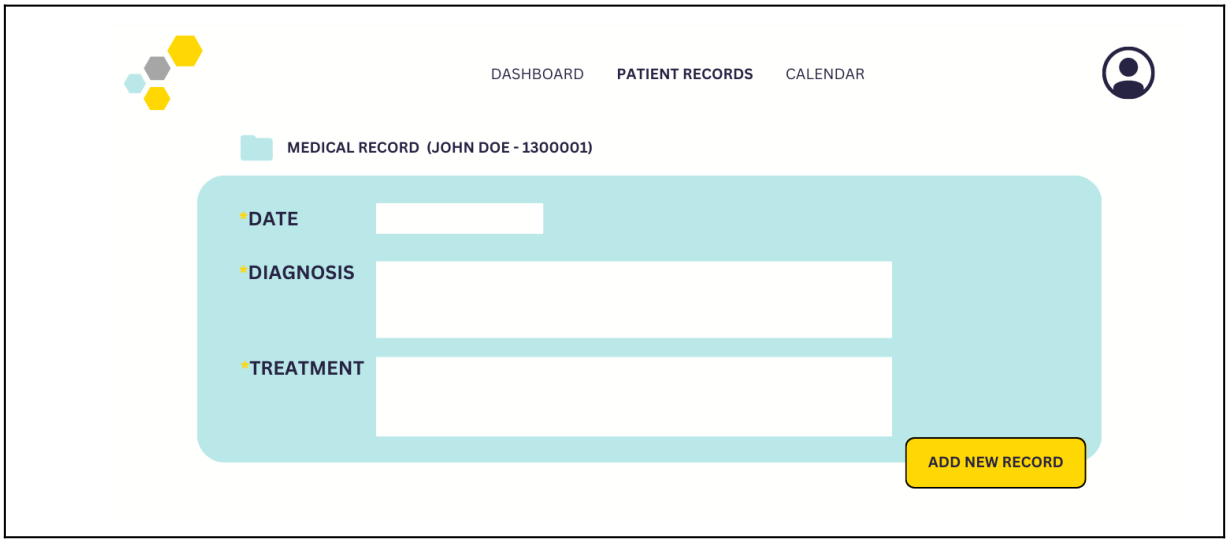

Figure 4.38 Pharmacist Add New Medical Record Screen

Upon filling up all the fields, which are required, the user can click the ‘Add New Record’ button. This will redirect the user to the medical records screen, which will also display the new record.

4.3.15 Calendar (Branch Pharmacist)

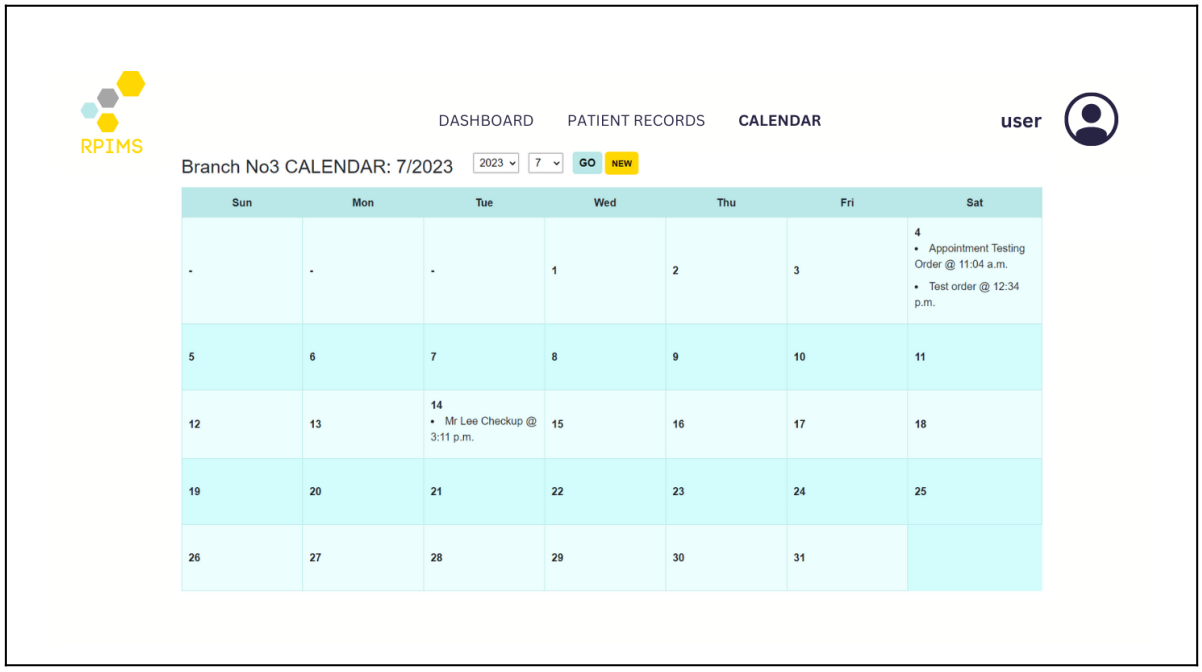

Figure 4.39 Pharmacist Calendar Screen

This is the Calendar page for the Branch Pharmacist. This page only displays the information relevant to the user's specific branch. The calendar will then display the current day and its set appointments for the day.

4.3.16 Dashboard (Branch Assistant)

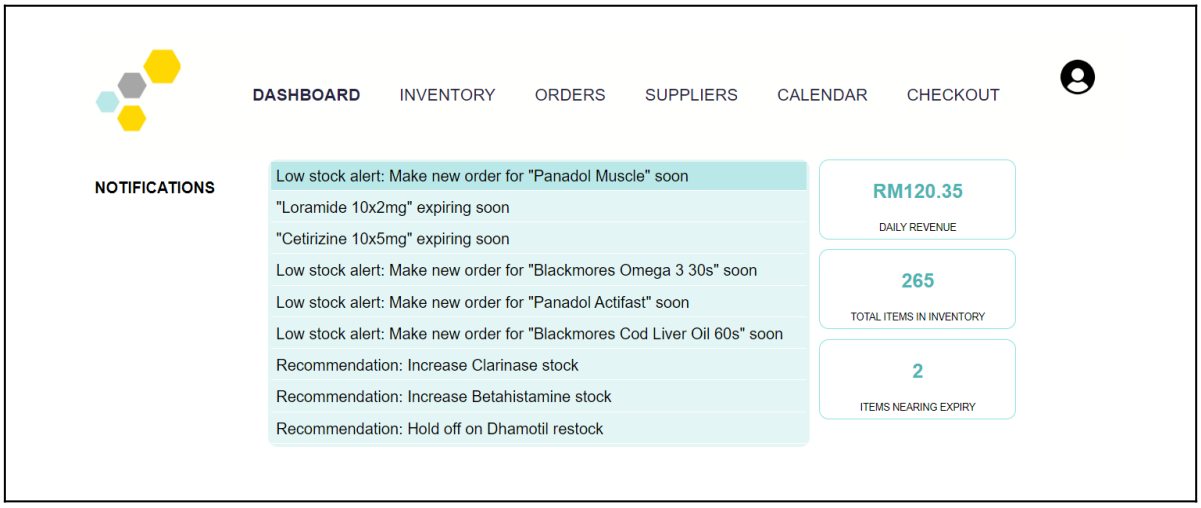

Figure 4.40 Assistant Dashboard Screen

This is the Dashboard for the Branch Assistant. The user has 6 main pages, as displayed in the navigation bar. Within the notification section, there are two containers: notifications, and important numerics for the branches.

4.3.17 Inventory (Branch Assistant)

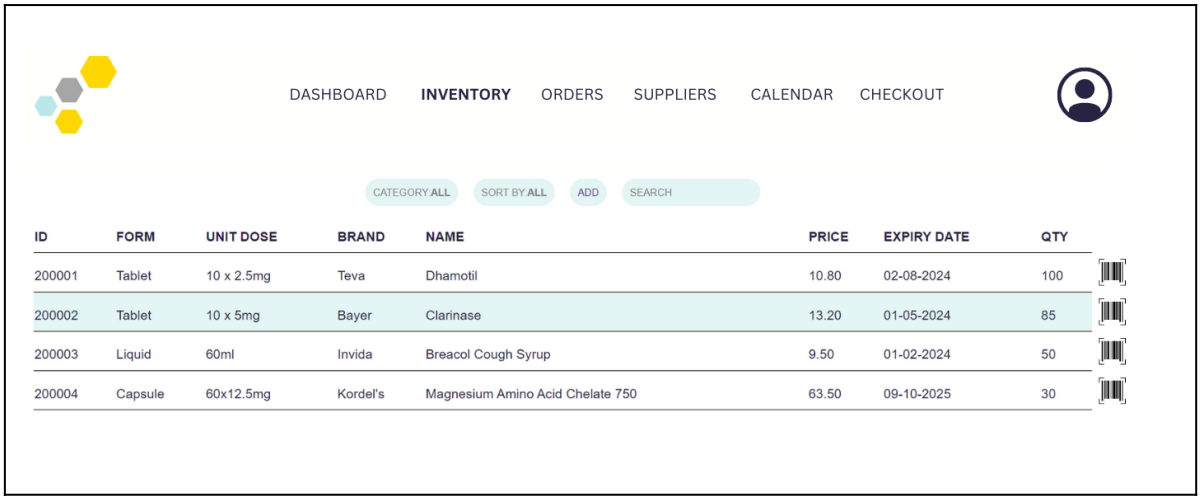

Figure 4.41 Assistant Inventory Screen

This is the Inventory page from the Branch Assistant perspective. Above the inventory table listing, there are two options to sort by category or by ID and expiration date. Additionally,

there is a search bar for the user to search specific entries by keywords and an ‘Add’ button which allows the user to be directed to a new page to add a listing.

Next to each listing, there is a barcode generation icon. Upon clicking the icon, the barcode image will be generated and downloaded to the user’s machine.

4.3.18 Calendar (Branch Assistant)

Figure 4.42 Assistant Calendar Screen

This is the Calendar page for the Branch Assistant. This page only displays the information relevant to the user’s specific branch. The calendar will then display the current day and its set appointments for the day.

4.3.19 Suppliers (Branch Assistant)

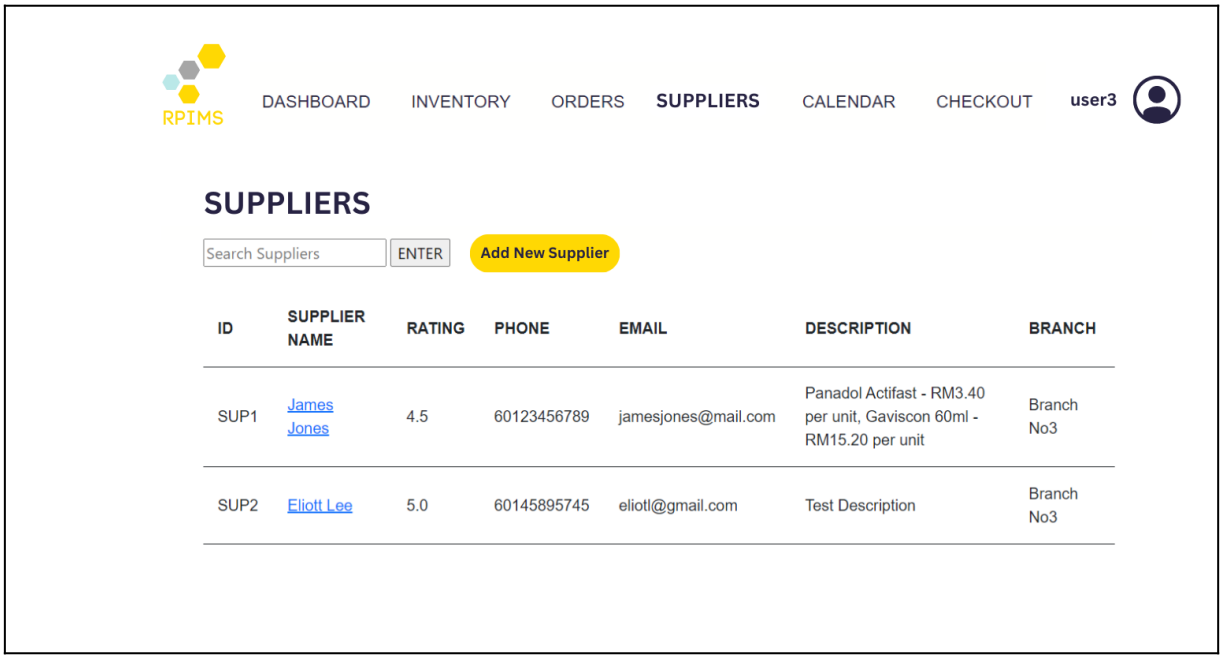

Figure 4.43 Assistant Suppliers Screen

This is the Suppliers page for the Branch Assistant, displaying the table for supplier information. Above the table, there is a search bar for the user to search by specific keyword as well as an 'Add New Supplier' button which redirects the user to a new page. Clicking on the supplier ID will redirect the user to an editing page.

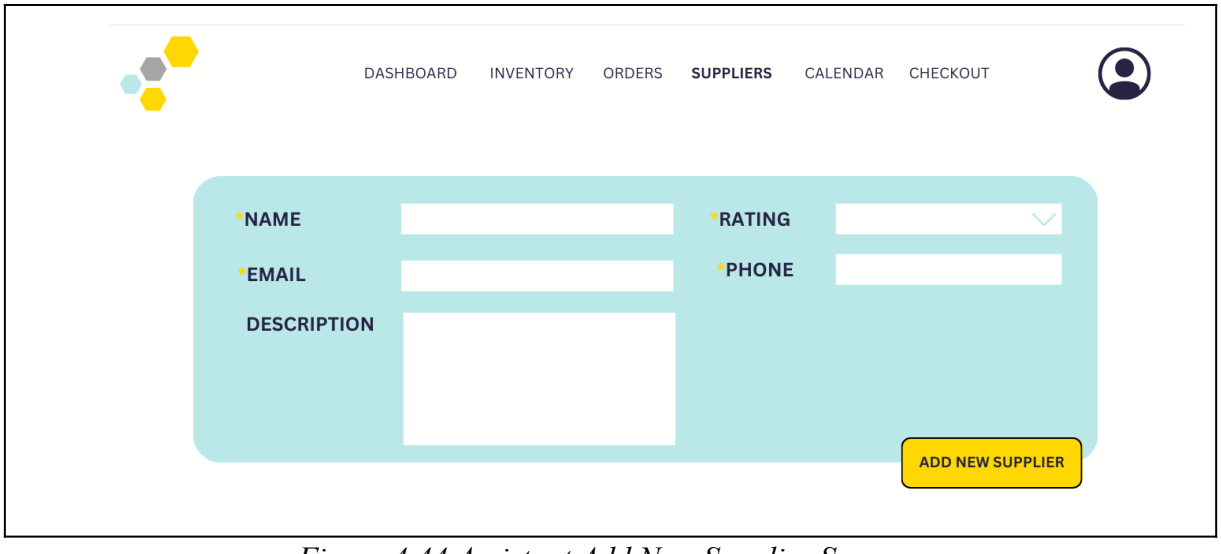

Figure 4.44 Assistant Add New Supplier Screen

Within the Add New Supplier page, there is a form with name, rating, email and phone as the required fields and description as an optional field. Upon filling up all the fields, which are required, the user can click the ‘Add New Supplier’ button. This will redirect the user to the Suppliers screen, which will also display the new supplier listing.

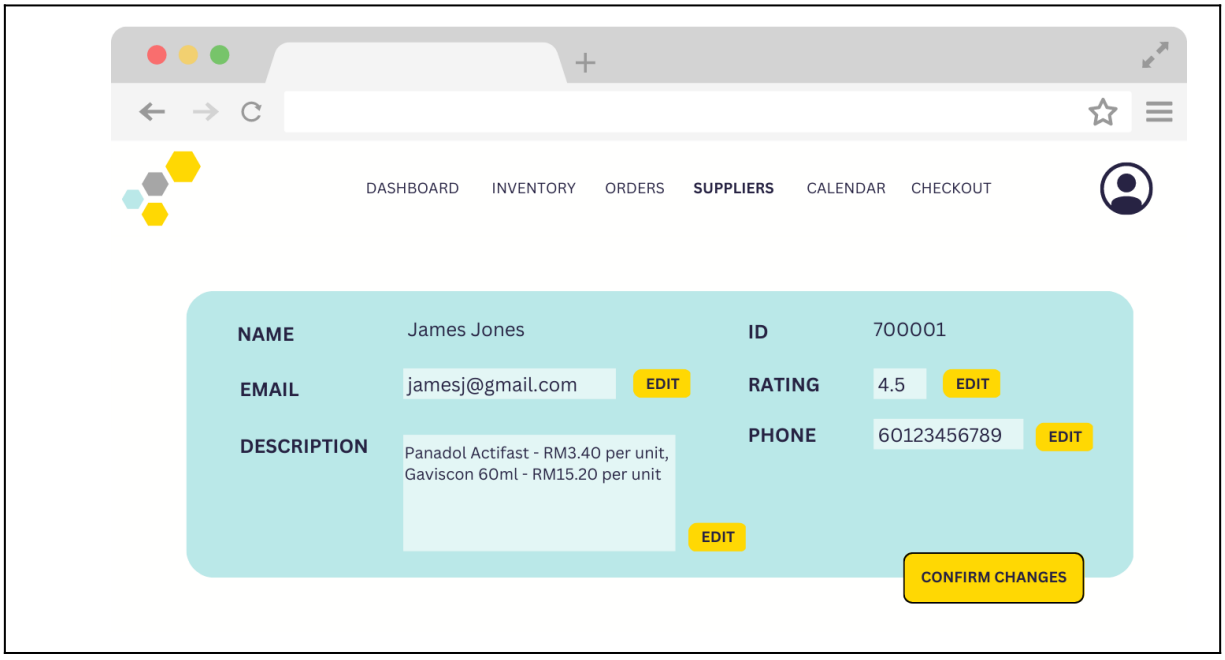

Figure 4.45 Assistant Edit Supplier Screen

This is the Edit Supplier page which displays the details of the existing supplier. Each field can be edited and upon clicking ‘Confirm Changes’, the system will update the database and redirect the user to the Suppliers page.

4.3.20 Checkout (Branch Assistant)

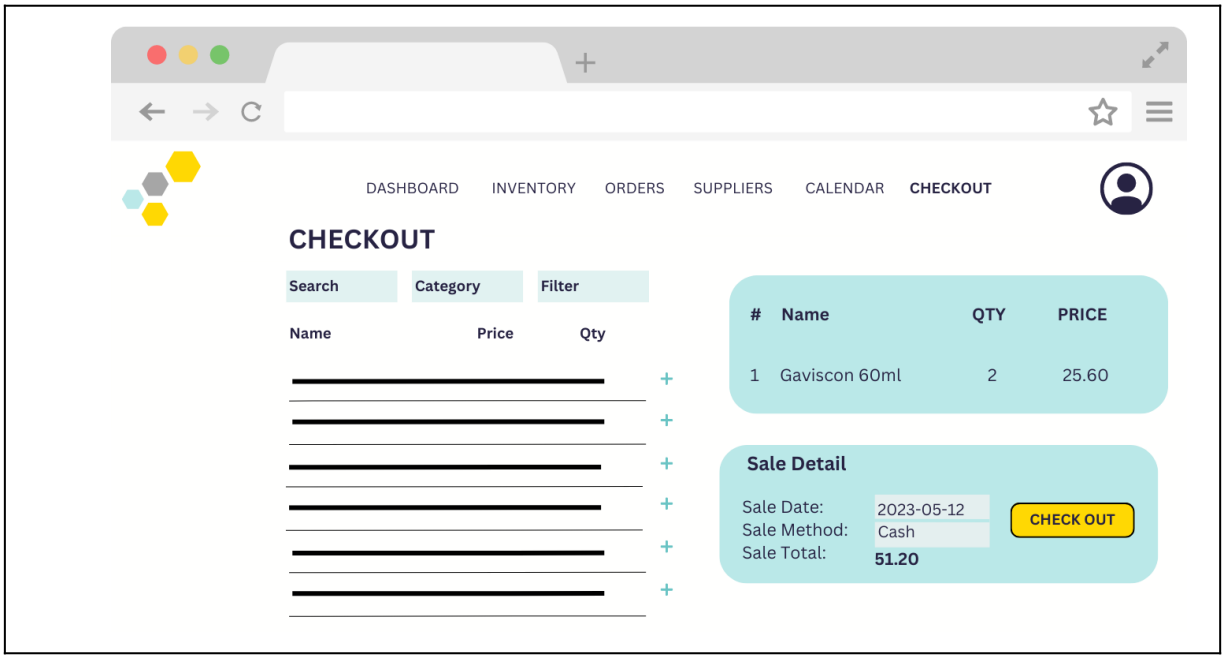

Figure 4.46 Assistant Checkout Screen

This is the Checkout page. On the left vertical navigation bar, there is a list of categories for the user to peruse through. Next to it, there is a search bar for the user to find a specific product that the customer wants, and a table below it which displays the inventory content based on the selected category and a '+' button beside each row so the user can easily add the items to cart. On the right side, there is an order summary container which acts as the customer's cart. Clicking the 'Check Out' button will register the order into the system.

4.3.21 Orders (Branch Assistant)

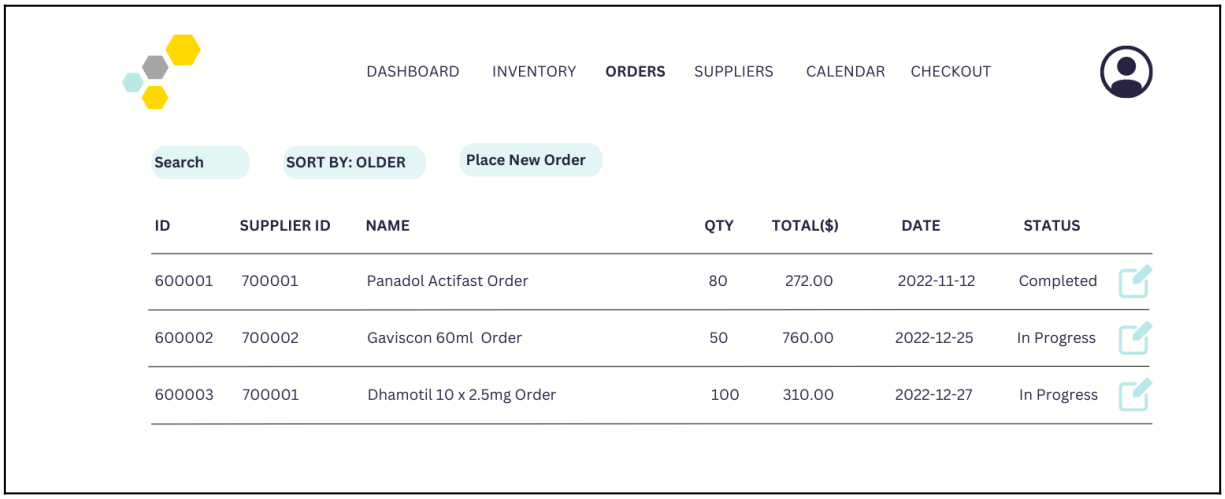

Figure 4.47 Assistant Orders Screen

This is the Orders page, which shows the stock order history table. The top has a search bar, a sorting dropdown button and a button to redirect the user to a 'Place New Order' page. Next to the status column, there is an edit icon which will allow the user to change the status of the order.

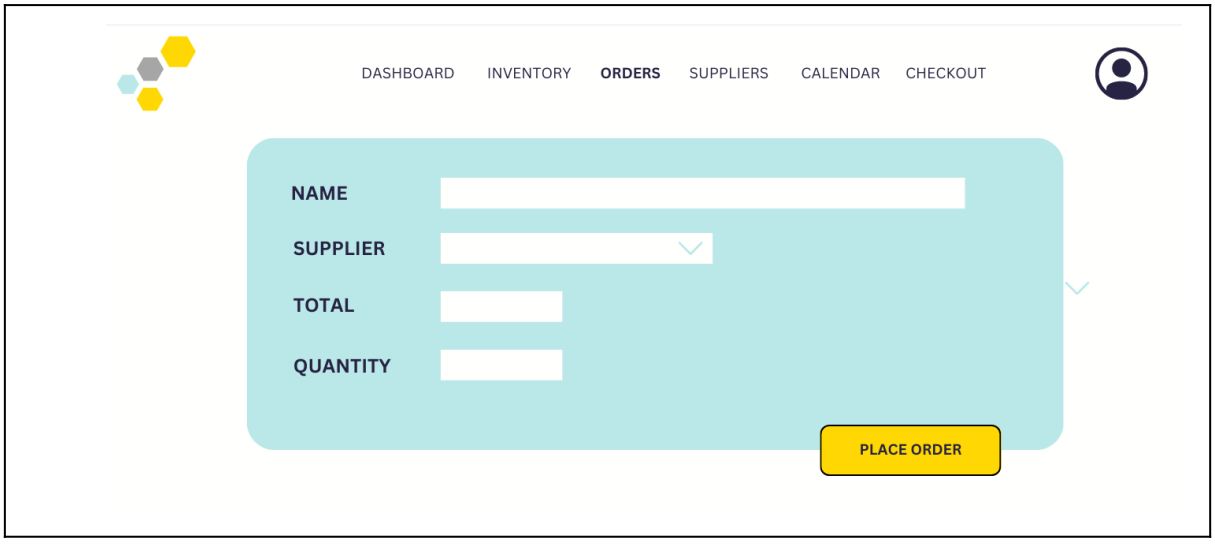

Figure 4.48 Assistant Place New Order Screen

This is the Place New Order page. In this case, the user has to place an order externally first before simply keying in the details in this form for the purposes of keeping track.

4.3.22 Add New Listing (Branch Assistant)

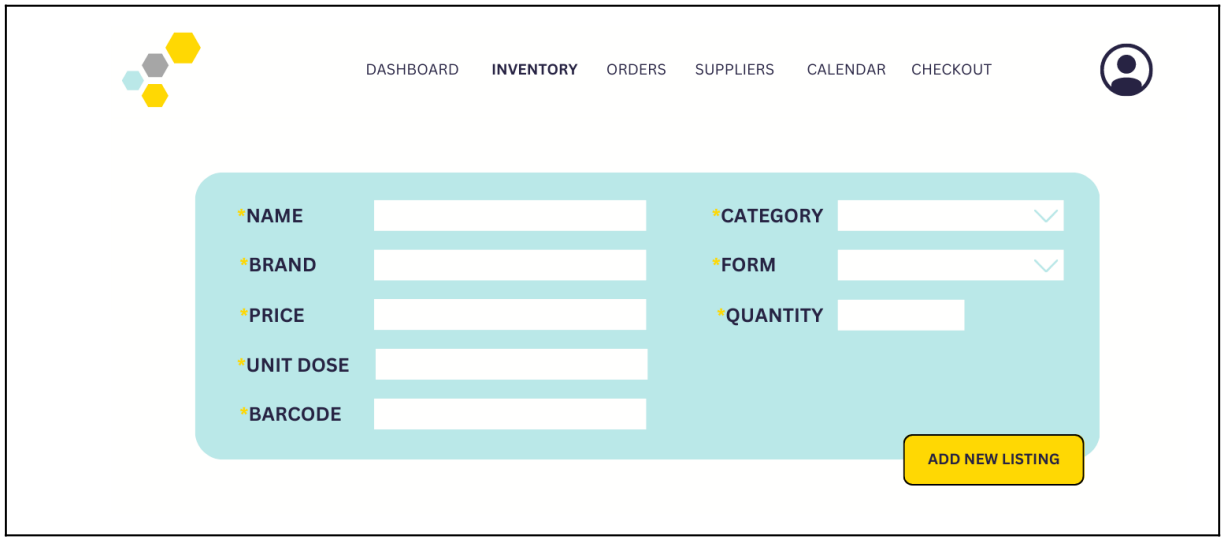

Figure 4.49 Assistant Add New Listing Screen

The Add New Listing page to add new products. Upon filling in the details, the user can click ‘Add New Listing’ to update the database and redirect to Inventory.

4.4 Data Dictionary

The following tables contain the information for the contents of the data tables within the system. Details include the name of the columns, data types, formats, field size, description of each key and an example of how an entry will look like.

Table 4.1 Branch Location Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Branch Location	id	int	N	10	Primary Key of Branch Location Table, auto incremented	1
	Branch_Name	string		50	Name of Branch	Cyberjaya 2 Glomac Branch
	Branch_address	string		255	Branch location address	Lot 123 Glomac Cyberjaya 2, 63000
	Branch_state	string		20	Branch location state	Selangor
	Branch_phonenumber	int		6000000000	Branch Location Office Phone Number, using max value validation	0312345678

Table 4.2 User Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
User	id	int	N	10	Primary Key of User Table, auto incremented	1
	username	varchar		50	Username for login purposes	johndoe1
	password	varchar		20	Login password	password1234
	email	email			Email associated with user	johndoe1@gmail.com
	role	varchar		20	User's role	Pharmacist
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.3 Product Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Product	id	int	N	10	Primary Key of Product Table, auto incremented	1
	Product_Category	string		50	Product Category Name	Supplements
	Product_Name	string		100		
	Product_Expirydate	date	YYYY-MM-DD	10	Expiration Date	2022-04-12
	Product_Barcode	text		13	Barcoding number	012345678912
	Product_Price	decimal		(1000, 2)	Price	50.95
	Product_Quantity	int			Stock quantity	40
	Unit_Dose	text		50	How each unit is sold in terms of packaging units and dosage	10 x 2.5mg
	Brand	string		50	Brand name of product	Kordel's
	Form	string		50	Physical form of product	Tablet
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.4 Sale Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Sale	id	int	N	10	Primary Key of Sale Table, auto incremented	1
	sale_total	decimal		(1000, 2)	Totaled price of all the items sold	126.40
	sale_date	date	YYYY-MM-DD	255	Date of transaction	2022-04-12
	sale_method	string		20	Customer's selected payment method	e-wallet
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.5 Sale Detail Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Sale Detail	id	int	N	10	Primary Key of Sale Detail Table, auto incremented	1
	item_quantity	int		255	Quantity of items in transaction	4
	sale	string	NNNNNN	6	Foreign Key	S00001
	product	string	NNNNNN	6	Foreign Key	P00001
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.6 Order Stock Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Order Stock	id	int	N	10	Primary Key of Order Table, auto incremented	1
	order_quantity	int		50	Quantity of ordered items	150
	order_name	String		50	Name of order	Panadol Reorder 3
	order_total	decimal		(1000, 2)	Total cost of order	1234.50
	order_date	date	YYYY-MM-DD	10	Date order is placed	2022-04-12
	order_time	time	hh:mm:ss	255	time of the appointment	12:15:02
	order_status	string		50	Current status of order	Completed
	supplier	string		100	Foreign Key - logs the supplier in charge of the order, referring to supplier name	SUP00001
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.7 Supplier Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Supplier	id	int	N	10	Primary Key of Supplier Table, auto incremented	1
	Supplier_Name	String		100	Supplier's Name	John Doe
	Supplier_Rating	Decimal	N.N	(2,1)	Rating of supplier out of 5	4.5
	Supplier_Description	text		255	Description and notes about supplier	Panadol Actifast - RM3.40 per unit, Gaviscon 60ml - RM15.20 per unit
	Supplier_Phone	int		6000 0000 00	Supplier's phone number, using max value validation	60123456789
	Supplier_Email	email		255	Supplier's email	johndoe@gmail.com
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.8 Appointment Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Appointment	id	int	N	10	Primary Key of Appointment Table, auto incremented	1
	app_detail	string		255	Details of Appointment	Consultation with Mr Lim
	date	date	YYYY-MM-DD	255	date of the appointment	2022-04-12
	time_start	time	hh:mm:ss	255	time of the appointment	12:15:02
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.9 Patient Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Patient	id	int	N	10	Primary Key of Patient Table, auto incremented	1
	Patient_Name	string		100	Name of patient	John Doe
	Age	int		150	Patient's age, using max value validation	32
	Phone	int		600000000	Patient's phone number, using max value validation	60123456789
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.10 Medical Record Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Medical Record	id	int	N	10	Primary Key of Medical Record Table, auto incremented	1
	Examination_Date	date	YYYY-MM-DD	10	Date of consultation or examination	2022-04-12
	Diagnosis	text		255	Diagnosis of patient problem	Mild case of vertigo due to protein crystallisation lost in right ear
	Treatment	text		255	Treatment plan for patient	3x 3mg Betahistine for 5 days, after meals
	patient	string		100	Foreign Key referring to patient name	John Doe
	branch	string		50	Foreign Key referring to branch name	Branch No1

Table 4.11 Notification Table

Table	Field Name	Data Type	Data Format	Field Size	Description	Example
Notification	id	int	N	10	Primary Key of Notification Table, auto incremented	1
	notification_type	string		25	classification	Low Stock
	notification_content	string		255	notification generated	Make new order for "Panadol Actifast" soon
	branch	string		50	Foreign Key referring to branch name	Branch No1

Chapter 5: IMPLEMENTATION

5.1 Deployment

For simplicity and to take advantage of the Github student developer pack, this project is deployed on Heroku, a cloud-based platform that simplifies web application deployment and management such as handling the infrastructure and scalability. Though, generally any Platform as a Service (PaaS) can be used in this case to host and manage the application. The deployment will be similar overall though depending on the platform, there may be small differences in setting up as well as different levels of cost based on the choice and scale of the application.

Additionally, it is important to note that the choice of PaaS should be chosen based on business objectives as they each have varying methods of application management and monitoring. However, for this project, Heroku is useful as it supports continuous deployment, allowing one to automate the deployment process through Continuous Integration/Continuous Deployment (CI/CD) pipelines with tools like GitHub Actions, which will be extremely beneficial in the progression of this project.

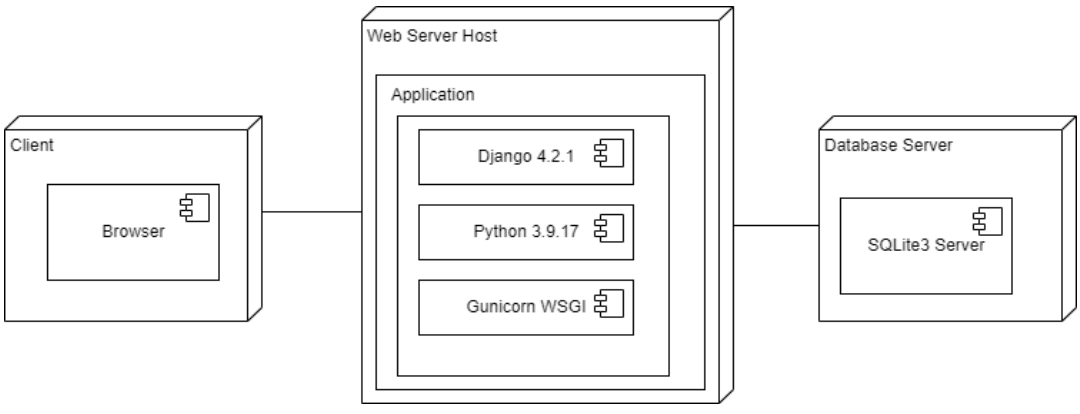

Figure 5.1 General Deployment Diagram

5.1.1 System Configuration

This documentation provides an overview of the system configuration for deploying a Django application on Heroku with GitHub integration. The configuration includes the interaction between different components involved in the deployment process.

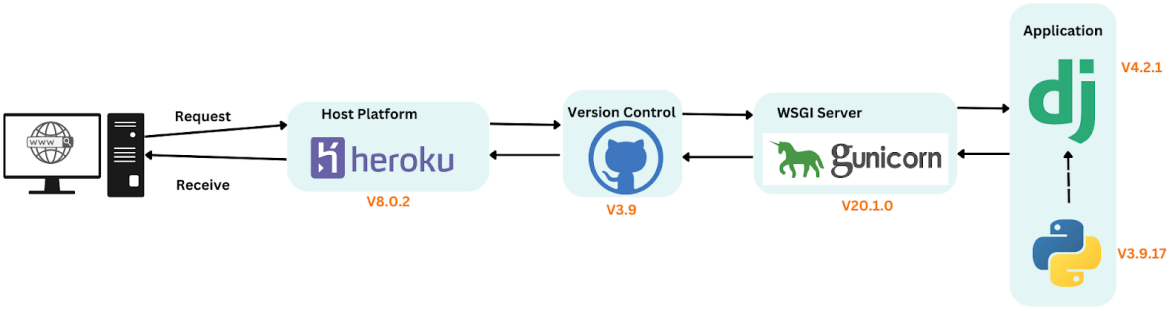

Figure 5.2 System Configuration Diagram

The system configuration follows the following flow:

1. End User: The system begins with an end user accessing the application through a web browser.
2. Heroku Platform: The request from the end user is directed to the Heroku platform, which manages the deployment and hosting of the application.
3. GitHub (Version Control): The Heroku platform communicates with GitHub, a version control system, to retrieve the application's source code. The source code is maintained and managed on GitHub.

4. WSGI (Gunicorn): Once the source code is obtained, the Heroku platform initializes the WSGI server, Gunicorn, which acts as the intermediary between the web browser and the Django application.
5. Django Application: The request is passed from Gunicorn to the Django application, where it is processed using Python code. The Django application contains the business logic and handles the request to generate an appropriate response.
6. Response Flow: After processing the request, the response flows back through the Django application and Gunicorn.
7. GitHub (Version Control): The response is then sent to GitHub for version control purposes, ensuring the application's codebase is up to date.
8. Heroku Platform: Finally, the response is sent back to the Heroku platform, which delivers the response to the end user's web browser.

5.1.2 Database

Using SQLite3 as the database, Django allows the management of data tables through implementation within the “model.py” files within every application folder within the project, as depicted in Figure 5.2. After entering the code, the user has to apply the changes through telling the terminal to make migrations and migrate in order for the table to be recognised.

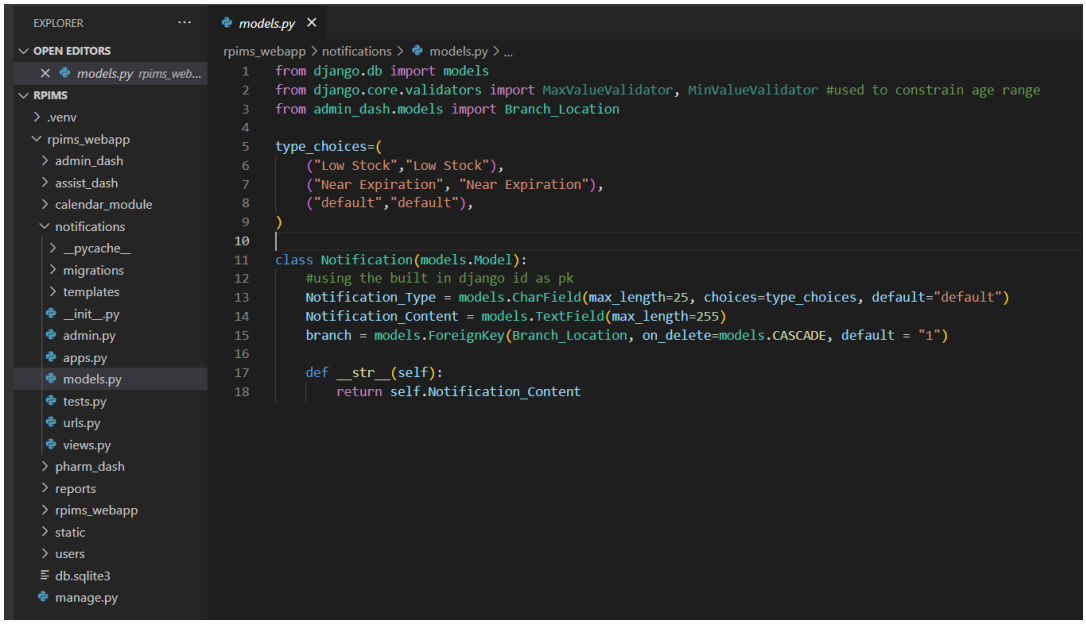

Figure 5.3 Django Models

At this point, the model can be further utilised by appending management to the admin panel, which is a feature that all Django projects have to easily view and manage the database. In order to do this, one has to go to the “admin.py” of the selected application and register the model to the admin site, as indicated in Figure 5.3. The result will be as in Figure 5.4, which is the final result of the RPIMS database implementation.

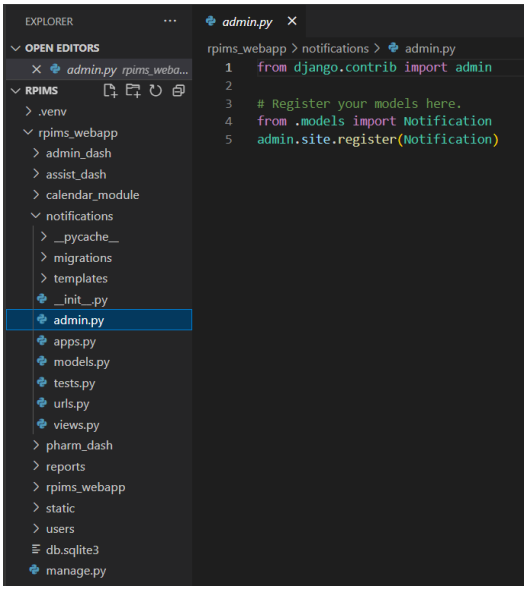

Figure 5.4 Registering a Model

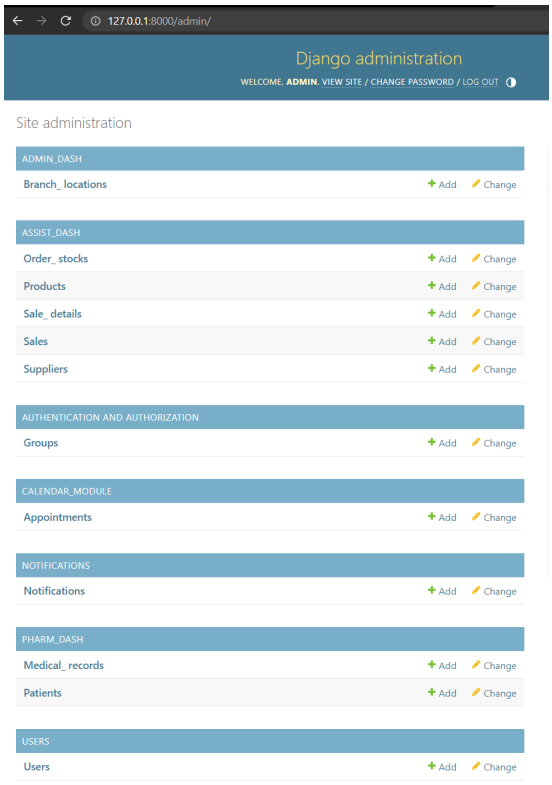

Figure 5.5 Django Admin Site

5.1.2.1 User Data

Since Django has a built-in user authentication interface and user model, some changes were made manually in a user "model.py" to include the user's role (Admin, Pharmacist, Assistant) and which branch the user will be assigned to. Figure 5.5 shows the pertaining "model.py" file and how it is set up.

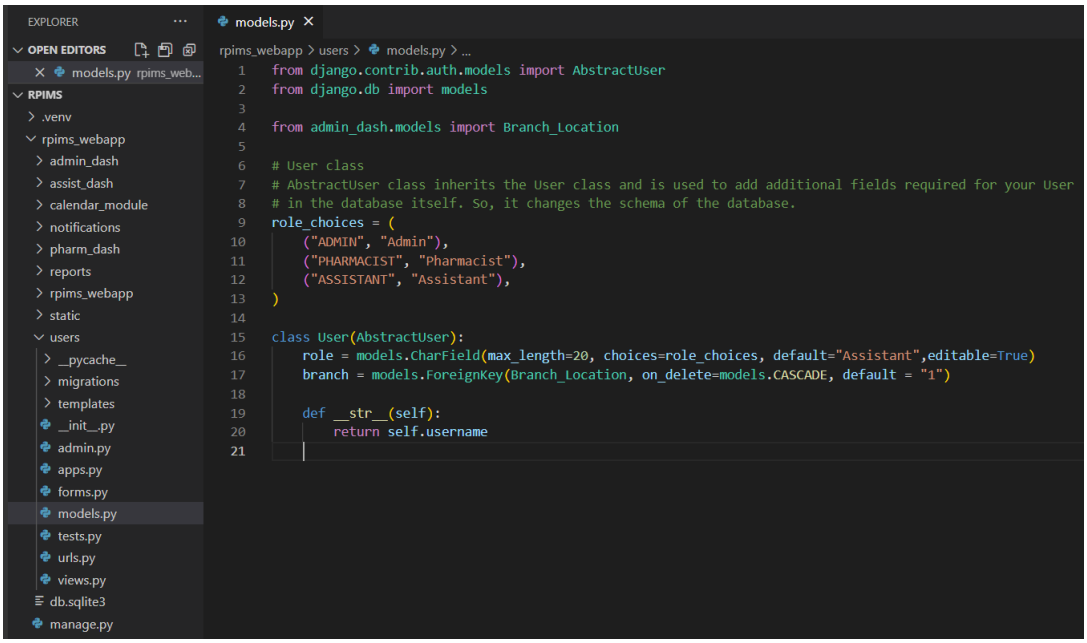

Figure 5.6 User Model Customisation

To further customise the user model, additional changes were made to the user "admin.py" before assigning the new model to the admin panel like in Figure 5.6.

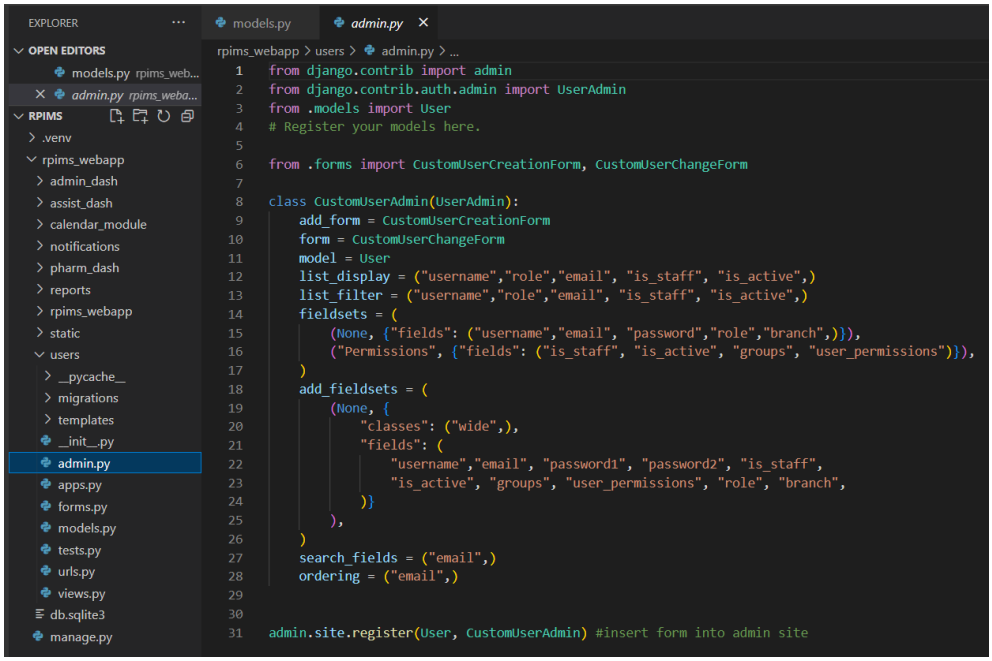

Figure 5.7 User Model to Admin Panel

The first change is to add more fields to the "Add User" form so that it looks like Figure 5.8.

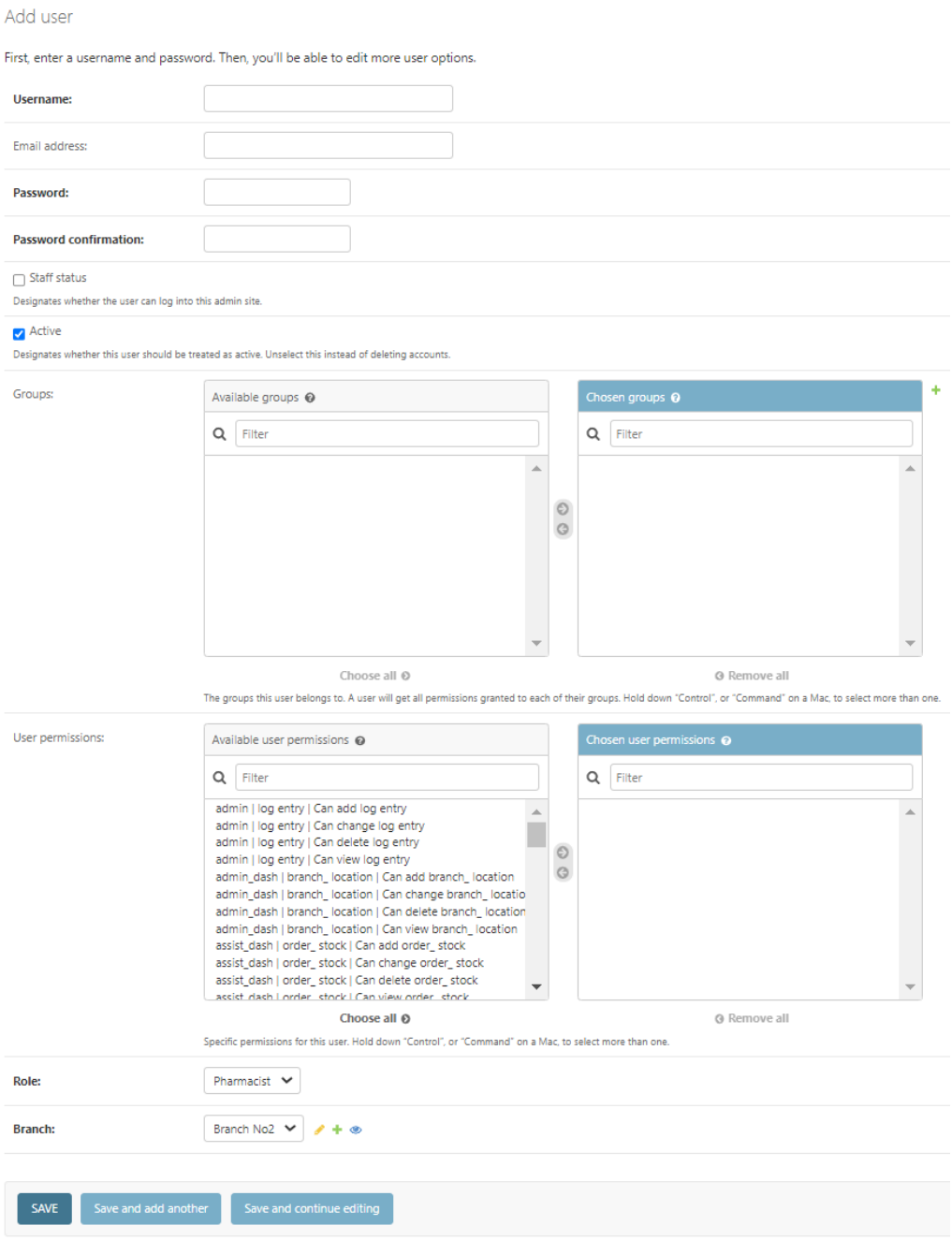

Figure 5.8 Admin Site Add User

The second change is to display the existing users with easily viewable information at a glance, and add filters to sort specific user groups like in Figure 5.8. This is intended to make it easy to manage users in the case that the program scales to have more than 30 users.

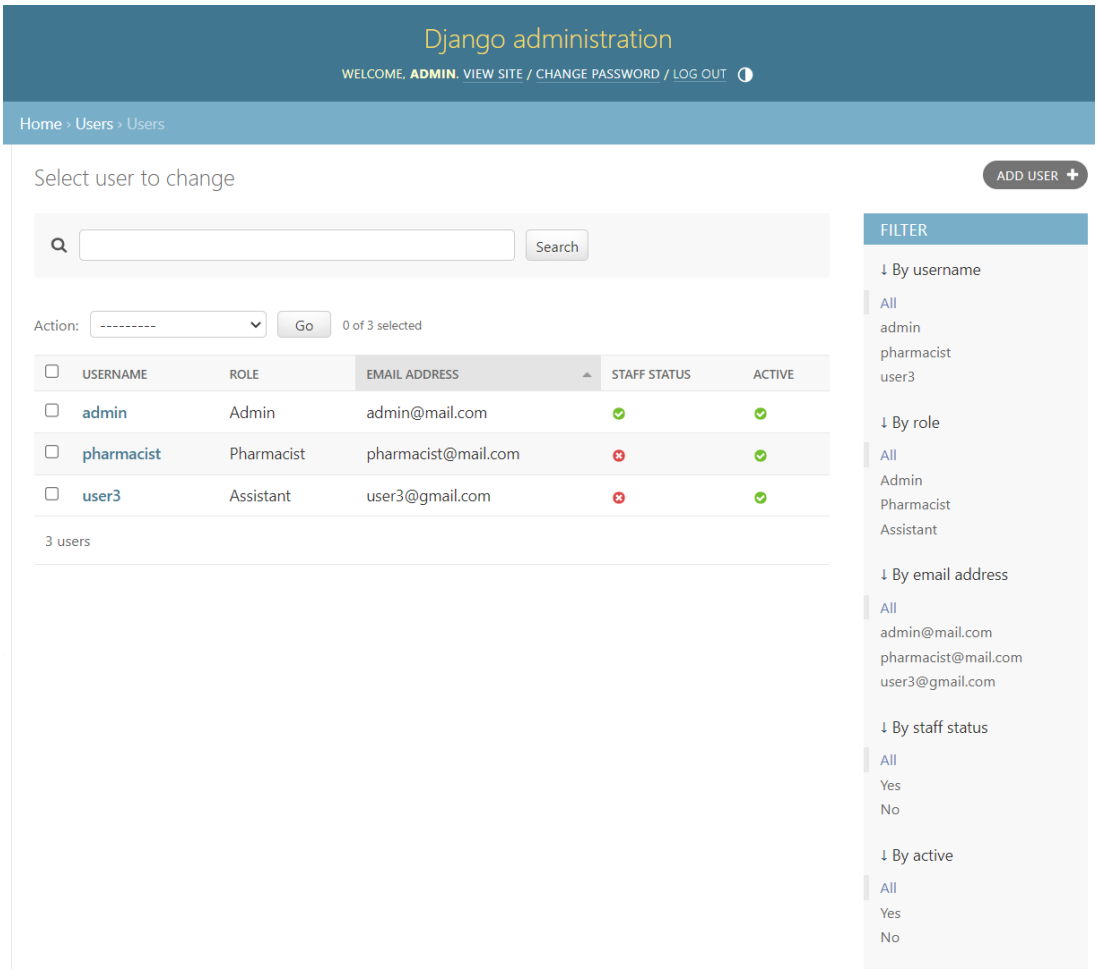

Figure 5.9 Admin Site Users

5.2 Development Environment

The main development environment for this project is Django Framework and edited through Visual Studio Code for convenience.

5.2.1 Development IDE

Visual Studio Code is the choice of Integrated Development Environment (IDE) for this project. The main reason is that there is an extensive set of extensions and integrations to aid in development such as Intellisense, an integrated terminal, Git version control support, as well as it has a very user friendly and intuitive interface.

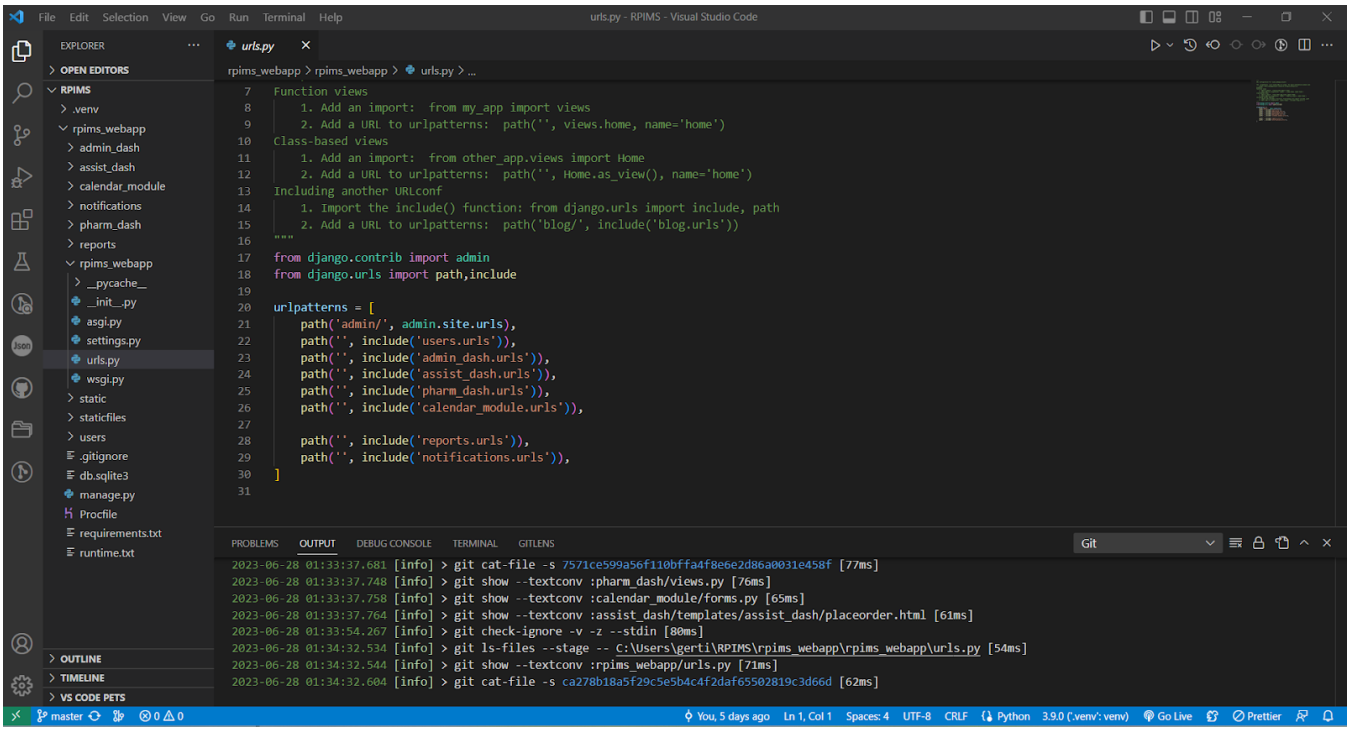

Figure 5.10 Sample Screen of the Project in Visual Studio Code

Throughout development, a Python virtual environment is utilised to prevent packages from being unnecessarily installed globally as well as to isolate the project and manage

dependencies more efficiently. At the time of this project, the current version being utilised is the latest version: Version 1.75.

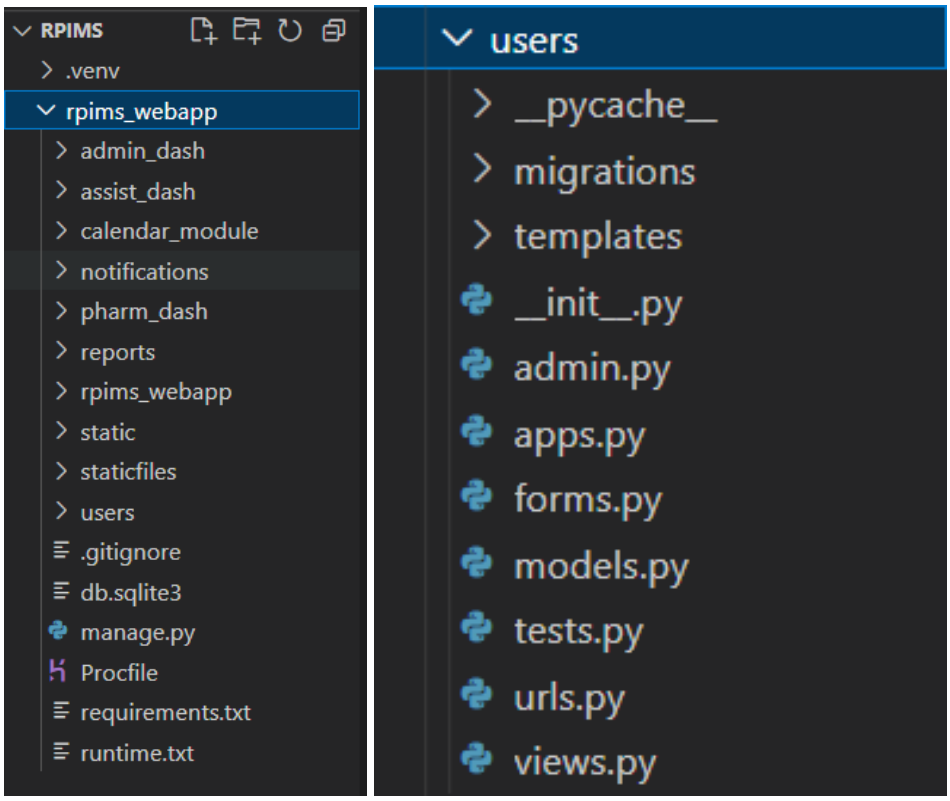

Figure 5.11 Folder Structure

Figure 5.4 depicts how the directory is generally structured based on the Django project format:

- Project: *rpims_webapp*
- Apps: *admin_dash, assist_dash, calendar_module, notifications, pharm_dash, reports, rpims_webapp, users*
- App content: *_pycache_, migrations, templates, _init_.py, admin.py, apps.py forms.py, models.py, tests.py, urls.py, views.py*

Each application contains similar files which serve different purposes in deciding the program behaviour.

1. **_pycache_**: This directory contains Python bytecode files that are generated and used by Python's interpreter for faster execution. It improves the performance of the application but is not necessary for version control, so it can be safely ignored.
2. **migrations**: This directory stores database migration files. Django's migration framework allows you to define and manage changes to your database schema over time. Each migration file represents a set of instructions for creating, modifying, or deleting database tables, fields, or relationships.
3. **templates**: This directory contains HTML templates used to render the views and generate dynamic web pages. Django follows the Model-View-Template (MVT) architectural pattern, and templates define how the data from models is presented to the user.
4. **__init__.py**: This empty file signifies that the directory it is placed in is a Python package. It is generally left empty unless you need to perform some initialization tasks for the app.
5. **admin.py**: This file is used to configure the app's administrative interface. It allows you to define how models should be displayed and manipulated in the Django administration site.
6. **apps.py**: This file defines the app's configuration. It contains metadata and settings specific to the app, such as the app's name, label, and any app-level configurations.
7. **forms.py**: This file contains Django forms that define how data is collected and validated from users. Forms handle user input, perform validation, and prepare data for processing.
8. **models.py**: This file defines the app's data models using Django's Object-Relational Mapping (ORM) system. Models define the structure and behavior of the

application's data, including database tables, fields, relationships, and methods for data manipulation.

9. **tests.py**: This file contains unit tests for the app. Django encourages writing tests to ensure the reliability and correctness of the application. Tests help validate that each component of the app functions as intended.
10. **urls.py**: This file defines the app's URL patterns or routes. It maps URLs to views, specifying which view function or class should handle a particular URL pattern and the corresponding action to be taken.
11. **views.py**: This file contains the views or view classes that define the logic for handling HTTP requests and returning HTTP responses. Views receive requests, interact with models and forms, and render templates to generate dynamic content.

5.2.2 Development Tools

The chosen development tools and their purpose within this project are as follows:

- Bootstrap: CSS framework for frontend development
- Visual Studio Code: Code editor with easily accessible terminal and extensions (eg Intellisense) for development aid.
- Keras: Data computation and machine learning modelling for reporting and recommendation modules.

5.3 Software Modules

5.3.1 CRUD Algorithms

For the purposes of describing the Create, Read, Update and Delete (CRUD) algorithms, the Branch Location data table will be referred to as demonstration.

CREATE:

```
def addbranch(request):
    submitted = False
    if request.method == "POST":
        form = addbranchform(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect('/addbranch?submitted=True')
    else:
        form = addbranchform
        if 'submitted' in request.GET:
            submitted = True

    form = addbranchform
    return render(request,'admin_dash/addbranch.html', {'form':form, 'submitted':submitted})
```

Figure 5.12 Create Branch Data Sample Code

The creation of table data is done by letting the user enter and submit new data through a form in the html templates. Within the respective view for the page, the user posts the form

data through submission and the backend checks if the data submitted is valid. If the data is valid, then the system will let the user know that the form was successfully submitted and that the table is now updated.

READ:

```
def branches(request):
    mybranches = Branch_Location.objects.all().values()
    template = loader.get_template('admin_dash/branches.html')
    context = {
        'mybranches': mybranches
    }
    return HttpResponse(template.render(context,request))
```

Figure 5.13 Read Branch Data Table Sample Code

Reading the data from the database works by calling all the values and saving it to a variable. Then load the html template with the variable and call the data using the format: {{variable_name.field_name}}.

UPDATE:

```
def viewbranch(request,branch_id):
    branch_to_edit = Branch_Location.objects.get(pk=branch_id) #call object from database
    submitted = False
    form3 = editbranchform(request.POST or None, instance=branch_to_edit)

    return render(request,'admin_dash/viewbranch.html',{'branch_to_edit': branch_to_edit,
        'form3' : form3,
        'submitted':submitted,})
```

Figure 5.14 Update Branch Data Table Sample Code

Updating the data combines the use of querying the specific row by getting the user-selected data table primary key, displaying said row's information, and presenting the information as a form within the template.

DELETE:

```
def deletebranch(request,branch_id):
    branch_to_delete = User.objects.get(pk=branch_id) #call object from database
    branch_to_delete.delete()
    return redirect('admin_dash/branches.html')
```

Figure 5.15 Delete Branch Data Sample Code

Deleting the data row begins by querying the specific row by getting the user-selected data table primary key, and connecting a button within the template to call the view function to delete the pertaining row.

5.3.2 Notifications Algorithm

The module has 3 types of notifications, namely Low Stock, Near Expiration, and Expired. The triggers for each notification are if any product has <=9 units in stock, if the expiration date for any product is <= 30 days from current date-time of the system, and if any product has expired respectively.

These notifications do not occur in real time, and instead, whenever a user accesses the Notifications page, the view function calls a “check_notifications” function within a utils.py file. This function goes through the list of products and sees if there are any notifications to be made based on the triggers mentioned. If it finds something, it checks if there already exists the same notification in the database. If it doesn’t exist, it saves but one already exists, then it skips creating a duplicate notification. This way, the notification table does not get overcrowded and hard to read if the notification tab keeps getting traffic. Additionally, when called from the database into the template, the notifications are colour-coded based on the types.

```python
def check_notifications():
    # Retrieve all products with stock quantity <= 9
    low_stock_products = Product.objects.filter(Product_Quantity__lte=9)

    # Retrieve all products with expiration date within 30 days from now
    expiration_date_threshold = datetime.utcnow().replace(tzinfo=utc) + timedelta(days=30)
    expiring_products = Product.objects.filter(Product_Expirydate__lte=expiration_date_threshold)
    expired_products = Product.objects.filter(Product_Expirydate__lte=datetime.utcnow().replace(tzinfo=utc))


    # Generate notifications for low stock products
    for product in low_stock_products:

        notification_content = f"The stock quantity for {product.Product_Name} is low ({product.Product_Quantity})."
        # Check if a notification with the same content already exists
        existing_notification = Notification.objects.filter(Notification_Content=notification_content).first()
        if existing_notification:
            continue  # Skip creating duplicate notification
        notification = Notification(
            Notification_Type='Low Stock',
            Notification_Content=notification_content,
            branch=product.branch
        )
        notification.save()

    # Generate notifications for expiring products
    for product in expiring_products:
        notification_content = f"The product {product.Product_Name} is expiring soon on {product.Product_Expirydate}."
        # Check if a notification with the same content already exists
        existing_notification = Notification.objects.filter(Notification_Content=notification_content).first()
        if existing_notification:
            continue  # Skip creating duplicate notification
        notification = Notification(
            Notification_Type='Near Expiration',
            Notification_Content=notification_content,
            branch=product.branch
        )
        notification.save()

    # Generate notifications for expired products
    for product in expired_products:
        notification_content = f"The product {product.Product_Name} has expired on {product.Product_Expirydate}."
        # Check if a notification with the same content already exists
        existing_notification = Notification.objects.filter(Notification_Content=notification_content).first()
        if existing_notification:
            continue  # Skip creating duplicate notification
        notification = Notification(
            Notification_Type='Expired',
            Notification_Content=notification_content,
            branch=product.branch
        )
        notification.save()
```

Figure 5.16 Notifications Utils Code

```html
<table class="notif-content">
    {% for x in notifications reversed %} <!--reversed displays the newest-->
    <tr >
        {% if x.branch == user.branch %}
        <td>
            {% if x.Notification_Type == "Low Stock"%}
            <span style="color: blue;font-weight:bold;">{{x.Notification_Type}}</span>
            {% elif x.Notification_Type == "Expired"%}
            <span style="color: red;font-weight:bold;">{{x.Notification_Type}}</span>
            {% else %}
            <span style="color: orange; font-weight:bold;">{{x.Notification_Type}}</span>
            {% endif %}
            - {{x.Notification_Content}}
        </td>
        {% endif %}
    </tr>
    {% endfor %}

</table>
```

Figure 5.17 Colour Coding Notification

5.3.3 Reports Algorithm

There are four report types that can be generated by the Pharmacist and Admin users, namely Waste, Sales, Inventory and Supplier Reports. The general way they work is that the user gets to query the conditions such as the time range for the data to be reported on, which varies a bit depending on the report type. The following details how each work:

Waste Report:

1. The user can generate a report based on the time range of 'Daily', 'Monthly', and 'Annual' by choosing from a list of drop down selections and clicking "Generate Report" to confirm the form submission.
2. The system will use the time range to select the inventory product listings which have expired, taking the Product Name, Quantity Wasted, Expiry Date and Monetary Value as well as calculating a Total Wasted Inventory Value from there and putting it into the report.
3. A button is available at the bottom of every report to allow the user to download the generated report as a PDF onto his machine.

Sales Report:

1. This report takes time ranges of 1, 3, 6, and 12 months in a similar method to the waste report.
2. Sales data with the selected time range is collated into a dataframe which is then displayed into the report.
3. Using the same sales data, a Long Short Term Memory (LSTM) Recurrent Neural Network (RNN) model references the data and works to make predictions for future

sales based on those. This is done using keras and tensorflow libraries and particularly chosen due to its ability to handle missing/empty entries for when the pharmacy does not make any sales, as well as its high compatibility with temporal data. In time, when there is more data available, the model should be able to make more accurate predictions and recommendations.

Inventory Report:

1. The user can generate a report based on the time range of 'Daily', 'Monthly', and 'Annual' by choosing from a list of drop down selections and clicking "Generate Report" to confirm the form submission.
2. The system uses the time range to select the relevant data from the database and outputs some insights such as:
    a. High mobility products
    b. Stagnant products
    c. Products nearing expiry

Supplier Report:

1. The user can generate a report based on the time range of 'Daily', 'Monthly', and 'Annual' by choosing from a list of drop down selections and clicking "Generate Report" to confirm the form submission.
2. The system uses the time range to select the relevant data from the database and outputs some insights such as Supplier Rating Summaries and Order Summaries.

5.3.5 Checkout Algorithm

1. **Input:** Gather the necessary input from the user, which typically includes the items being purchased, their quantities, and any other relevant details like customer information and payment method.
2. **Check Inventory:** Verify the availability of the purchased items in the inventory to ensure they can be fulfilled. Compare the requested quantities with the available stock. If any items are out of stock or insufficient, either inform the user or adjust the quantities accordingly.
3. **Calculate Total:** Calculate the total cost of the items being purchased based on their quantities and prices. Apply any discounts, taxes, or additional charges as required.
4. **Update Inventory:** Deduct the quantities of the purchased items from the inventory to reflect the sale. Update the inventory records accordingly to ensure accurate stock management. Consider implementing appropriate measures for concurrency control if multiple users can make simultaneous purchases.
5. **Store Sales Data:** Create a new entry in the sales table or database to store the details of the transaction. This typically includes information such as the sale date and time, customer details, purchased items, quantities, prices, and any other relevant information needed for reporting or analysis.
6. **End:** Complete the checkout process and ensure all relevant data is securely stored and synchronized.

5.3.6 Calendar Algorithm

```
def placeorder(request):
        submitted = False
        if request.method == "POST":
            form3 = placeorderform(user=request.user, data=request.POST)
            if form3.is_valid():
                form3.save() #save the Order_Stock object

                #getting data from the form entry
                appdetaildata = form3.cleaned_data.get('Order_Name')
                appdatedata = form3.cleaned_data.get('Order_Date')
                apptimedata = form3.cleaned_data.get('Order_Time')

                #creating a new object from form data in Appointment
                Appointment.objects.create(app_detail = appdetaildata, date = appdatedata, time_start = apptimedata, branch = request.user.branch )
                return HttpResponseRedirect('/placeorder?submitted=True')
        else:
            form3 = placeorderform(user=request.user)
            if 'submitted' in request.GET:
                submitted = True
        return render(request,'assist_dash/placeorder.html', {'form':form3, 'submitted':submitted})
```

Figure 5.18 Calendar Sample Code

The calendar appointment module gets its objects from two places: stock ordering and appointment registration directly on the calendar page. Within the stock ordering view function, whenever an order or appointment is made through a form, specific form data is taken and used to create another appointment object in the database. After creating appointment objects, it can be called and displayed onto the respective branch employee calendar pages.

5.3.7 Barcoding Algorithm

Barcoding is automatically implemented via the backend using the python-barcode library whenever a new product form is submitted. The barcode format used is EAN-13 (European Article Number 13) which is not only the main barcode used in Malaysia but also the most used type worldwide. The upsides are that it can be read right-side-up or even upside-down in the case that there are high volume purchases or high store traffic at the pharmacy. Further implementation if this project is to be expanded on will require the usage of omni-directional scanners. Figure 5.19 demonstrates how the barcode number should be formatted but for the

purpose of simply demonstrating the capability of the system, the barcode number is just set to be randomly generated within 13 digits and unique.

Figure 5.19 Barcoding Format

The flow of events begins with the user adding a new product through the “Add Product” form. Upon submitting the details, a number of things happen. First, the view function validates the form data to make sure it meets the defined constraints. If the form data is valid, the next thing to tackle is the barcode. The form function calls a generate unique barcode function within itself which returns a 13 digit number that is unique and doesn’t match the existing barcodes within the system. Then the form uses that number with another function in ‘barcode_utils.py’ which uses the barcode library and converts the 13 digits into a barcode image to save into a designated folder.

```
def generate_unique_barcode(self):
    while True:
        Product_Barcode= str(randint(1000000000000, 9999999999999))
        if not Product.objects.filter(Product_Barcode=Product_Barcode).exists():
            return Product_Barcode
def save(self, commit=True):
    product = super().save(commit=False)
    product.Product_Barcode = self.generate_unique_barcode()

    if commit:
        product.save()

    # Generate and save the barcode image
    output_path = f"static/barcodes/{product.Product_Barcode}"
    generate_ean13_barcode(product.Product_Barcode, output_path)
    product.save()
    return product
```

Figure 5.20 Barcoding form function code sample

```
rpims_webapp > assist_dash > barcode_utils.py > ...
1  from barcode import EAN13
2  from barcode.writer import ImageWriter
3
4  def generate_ean13_barcode(number, output_path):
5      barcode = EAN13(str(number), writer=ImageWriter())
6      barcode.save(output_path)
7
```

Figure 5.21 barcode_utils.py code sample

5.4 Sample Outputs

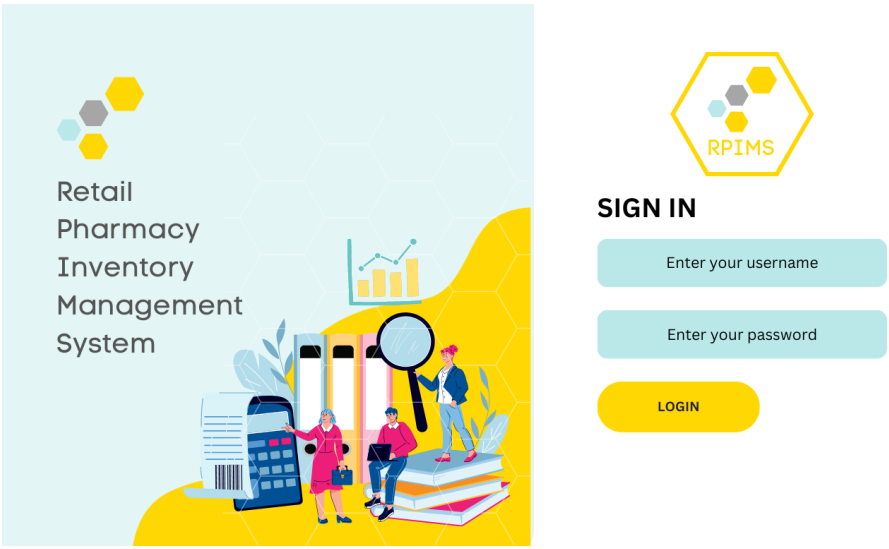

Figure 5.22 Sign In Page

The Sign In Page is the central point of access for all three of the users. If the credentials do not match the ones in the database and cannot be authenticated correctly, the user will encounter an error message, prompting them to try again. Additionally, this is the screen that the users are redirected to after they've logged out. Upon which there will be a message next to the login button saying "You've been logged out" as a confirmation.

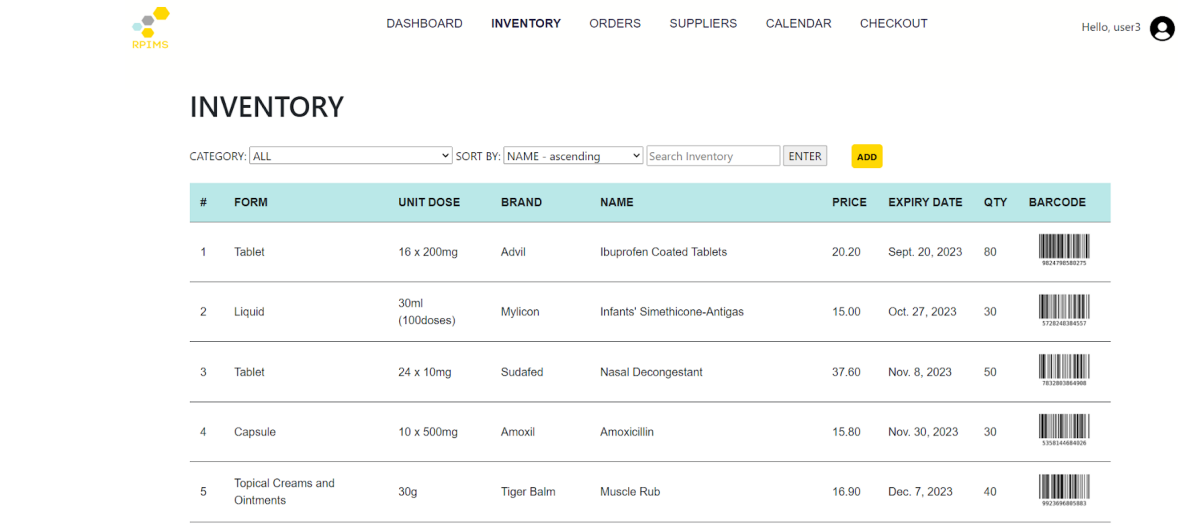

Figure 5.23 Inventory Page

The Inventory Page displays all the Product listings belonging to the logged-in user's branch. There is a filter and search bar at the top to easily find listings and an "Add" button next to it to allow the user to add a new product listing into the system. Each listing also has its own generated EAN-13 barcode. When the user clicks on the barcode, it can be downloaded onto the user's machine.

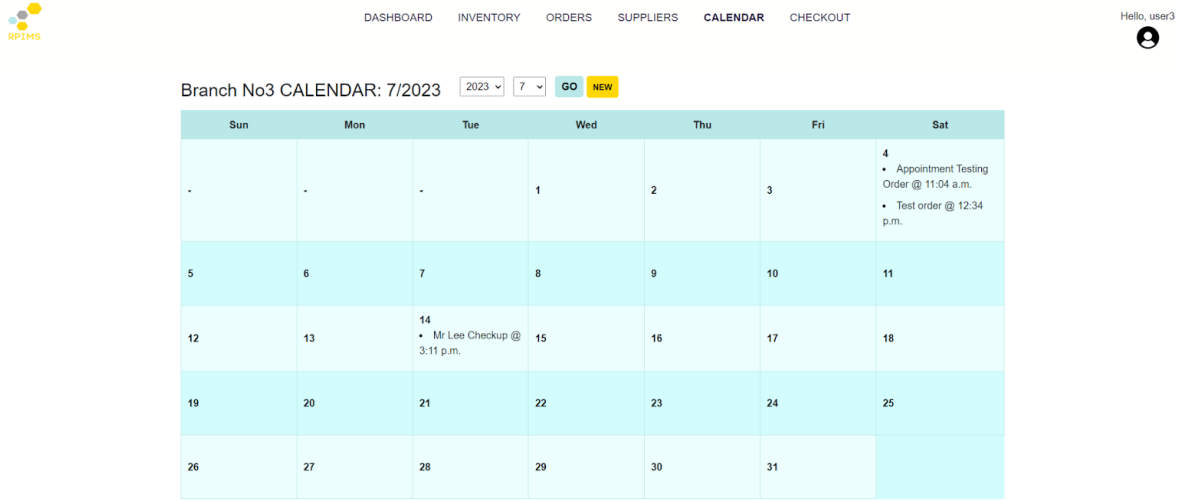

Figure 5.24 Calendar Page

The Calendar Page displays all the appointments (patient and order) pertaining to the user and his branch. Different months and years can be chosen to see an instant view of what is going on during that month.

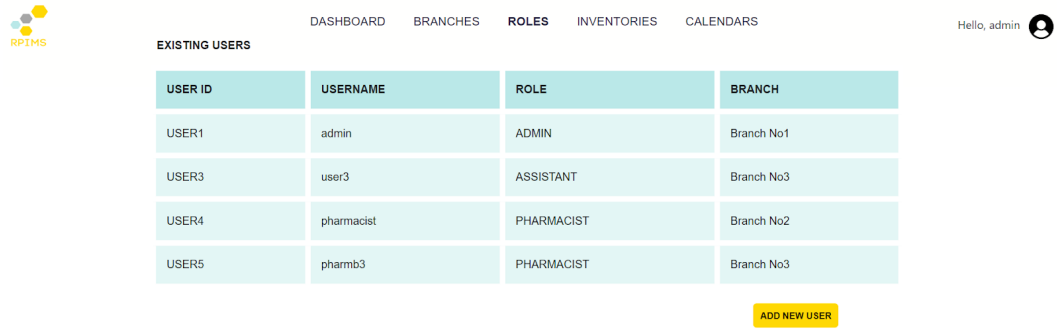

Figure 5.25 Roles Page

This page shows an overall view of the current users registered within the system. This also acts as an easy access for the admin to add any new users or select some to edit their details.

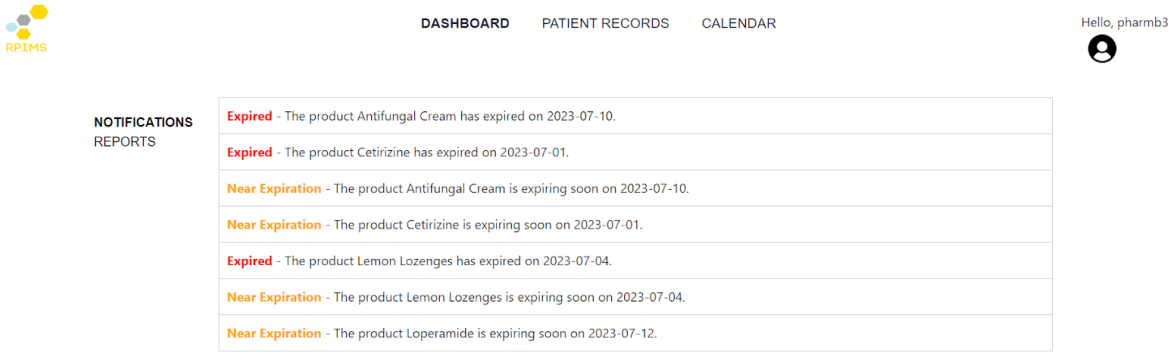

Figure 5.26 Notification Page

This page holds notifications regarding the pertaining user branch's inventory. Notification types are colour-coded based on urgency so that the user can prioritise and make an efficient plan on how to deal with the things mentioned in the alerts.

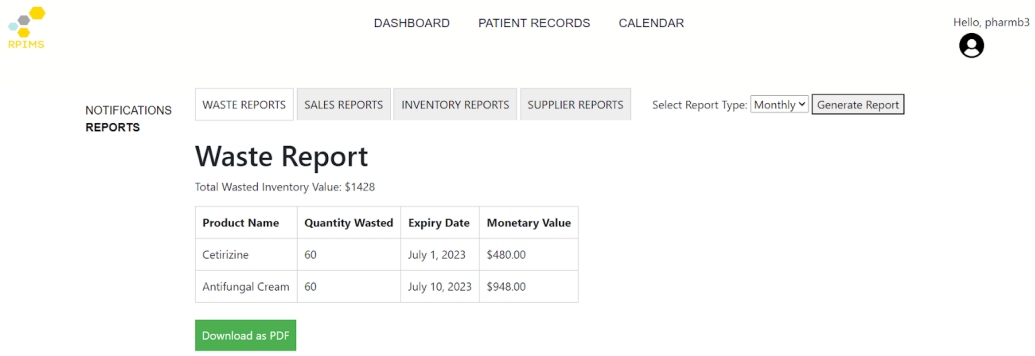

Figure 5.27 Reports Page

This is the view for the pharmacist reports. A bar of tabs sits on the top for easy navigation to different varieties of reports, and next to it there is an option to choose different time ranges. Below is a Download button so the current generated report can be downloaded into the user's machine.

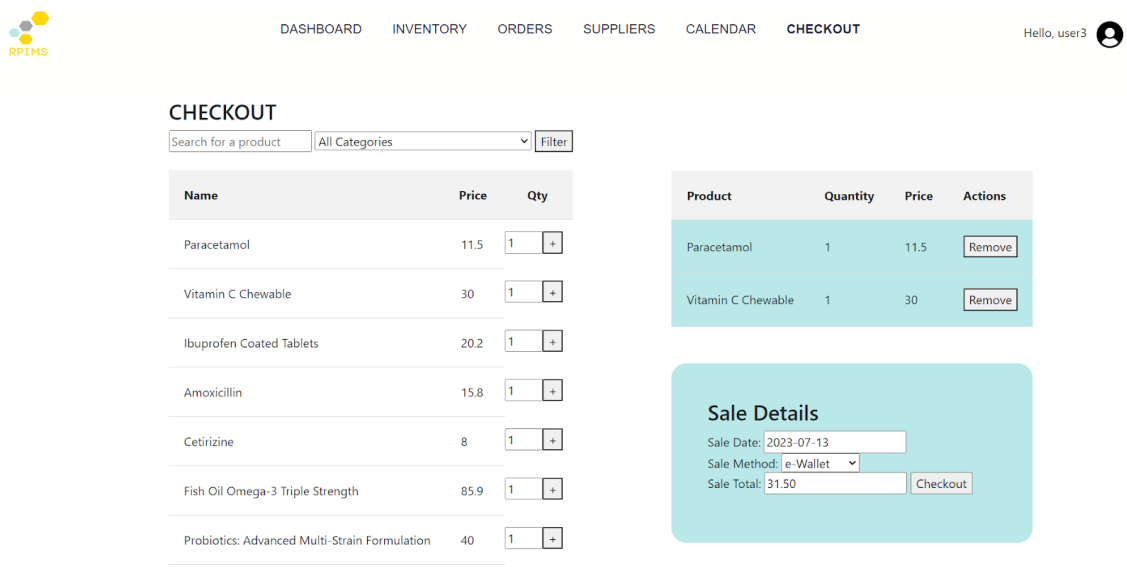

Figure 5.28 Checkout Page

The Checkout Page contains a filter and search bar for the user to select which items to add to the cart on the right, which updates dynamically depending on if the user removes or adds items. Sale details indicate the date, method and total of the transaction.

Chapter 6: TESTING

6.1 Test Plan

There will be three test plans for each of the modules in Admin, Assistant and Pharmacist, and an additional test plan for some shared features between the users.

Table 6.1 Test plan for Admin Module

Test Plan				
Module	No	Test ID	Function	Test Date
Admin	1	T01	Manage branches	4.06.2023
	2	T02	Manage user roles	4.06.2023
	3	T03	View Calendars (all branches)	8.06.2023
	4	T04	View Inventory (all branches)	8.06.2023
	5	T05	View Notifications (all branches)	29.06.2023
	6	T06	View Reports (all branches)	30.06.2023

Table 6.2 Test plan for Assistant Module

Test Plan				
Module	No	Test ID	Function	Test Date
Assistant	1	T07	Manage Suppliers	10.06.2023
	2	T08	Order Stock & update order status	10.06.2023
	3	T09	Manage Inventory	15.06.2023
	4	T10	Generate product barcode	15.06.2023
	5	T11	Check Out	30.06.2023

Table 6.3 Test plan for Pharmacist Module

Test Plan				
Module	No	Test ID	Function	Test Date
Pharmacist	1	T12	Manage Patient Records	28.06.2023
	2	T13	Manage Patient Prescription	28.06.2023
	3	T14	View Reports	30.06.2023

Table 6.4 Test plan for Shared Functions

Test Plan				
Module	No	Test ID	Function	Test Date
Assistant and Pharmacist	1	T15	View Notifications (own branch)	30.06.2023
	2	T16	Manage Calendar (own branch)	30.06.2023
all	3	T17	Log In	15.06.2023

6.1.1 Test Data

The gist of the test data is distributed amongst the three user modules, and the respective test data will reflect in **Appendix E**:

Table 6.5 Test Data Summary

Module	Test Case		Relevant Test Data
Admin	T01	Manage branches	Name, Address, State, Phone Number
	T02	Manage user roles	Username, Password, Email, Role, Branch
	T03	View Calendars (all branches)	Appointment detail, date, time, branch
	T04	View Inventory (all branches)	Category, Name, Expiry Date, Barcode, Price,

			Quantity, Unit Dose, Brand, Form, Branch
	T05	View Notifications (all branches)	Notification Type, Content, Branch
	T06	View Reports (all branches)	Branch, Expiration Date, Supplier Ratings, Sales, Sale Date
Assistant	T07	Manage Suppliers	Name, Rating, Description, Phone, Email
	T08	Order Stock & update order status	Supplier, Order Quantity, Name, Total, Date, Status
	T09	Manage Inventory	Category, Name, Expiry Date, Barcode, Price, Quantity, Unit Dose, Brand, Form
	T10	Generate product barcode	Product Barcode
	T11	Check Out	Product quantity, sales, sale total
Pharmacist	T12	Manage Patient Records	Patient Name, Age, Phone
	T13	Manage Patient Prescription	Examination Date, Diagnosis, Treatment, Patient
	T14	View Reports	Expiration Date, Supplier Ratings, Sales, Sale Date
Assistant and Pharmacist	T15	View Notifications (own branch)	Notification Type, Content
	T16	Manage Calendar (own branch)	Appointment detail, date, time
All	T17	Log In	Username, Password,Role

6.2 Test Results

Table 6.6 Test Case 1

Test Case	Test Case ID	T17
	Description	Check user login to validate existing users in accordance to user roles
	Precondition	A valid user account is available in the system
	Post Conditions	The pharmacist is successfully logged into the application and has access to the pharmacist-specific features.
Test Script	Test Steps	1. Launch the application. 2. Enter the valid username for a pharmacist. 3. Username: **pharmacist** 4. Enter the valid password for the pharmacist. 5. Password: **ppass1234** 6. Click on the "Login" button. 7. Verify that the user is successfully logged in. 8. Expected Result: The user is redirected to the pharmacist dashboard. 9. Verify that the appropriate pharmacist-related features are available on the dashboard. 10. Expected Result: The dashboard should display functionalities such as reports, patient records and calendar.
	Expected Result	User should be able to log in to the respective user dashboard
	Actual Results	Successful login to admin, pharmacist, and assistant views

Table 6.7 Test Case 2

Test Case	Test Case ID	T01
	Description	Check adding, editing and deleting branches.
	Precondition	A valid Admin user is logged in.
	Post Conditions	New branches can be added, edited and deleted.
Test Script	Test Steps	1. Navigate to the “Branches” tab via the navigation bar. 2. Click on “Add Branches” 3. Fill in the form data (Branch Name, State, Phone Number, Address) 4. Select “Add New Branch” 1. Click on the listed branch in “Branches” to edit. 2. System redirects to view branch page 3. Edit the address 4. Select “Confirm Changes” 1. Click on the listed branch in “Branches” to edit. 2. System redirects to view branch page 3. Select “Delete”
	Expected Result	● Page displays “New Branch Added Successfully” and the new branch is viewable in “Branches”. ● Updated branch details are viewable in “Branches” ● Deleted branch is no longer in the list.
	Actual Results	Successful adding, editing and deletion of branch

Table 6.8 Test Case 3

Test Case	Test Case ID	T02
	Description	Check adding, editing and deleting users.
	Precondition	A valid Admin user is logged in.
	Post Conditions	New users can be added and deleted.
Test Script	Test Steps	1. Navigate to the “Roles” tab via the navigation bar. 2. Click on “Add New User” 3. Fill in the form data (Username,Email address, Role, Branch, Password, Password Confirmation) 4. Select “Add New User” 1. Click on the listed user in “Roles” to edit. 2. System redirects to view user page 3. Select “Delete”
	Expected Result	● Page displays “New User Added Successfully” and the new user is viewable in “Roles”. ● Deleted user is no longer in the list.
	Actual Results	Successful adding and deletion of user

Table 6.9 Test Case 4

Test Case	Test Case ID	T03
	Description	View the calendars for all of the branches
	Precondition	A valid Admin user is logged in.
	Post Conditions	Calendars for the branches can be viewed
Test Script	Test Steps	1. Navigate to the “Calendars” page 2. Select branch from the dropdown list 3. Select the month to view and click “Go”
	Expected Result	● Calendar displays for each of the branch selected
	Actual Results	Successful viewing of calendars

Table 6.10 Test Case 5

Test Case	Test Case ID	T04
	Description	View the Inventory for all of the branches
	Precondition	A valid Admin user is logged in.
	Post Conditions	Inventories of the branches can be viewed
Test Script	Test Steps	1. Navigate to the “Inventories” page. 2. Select branch from the dropdown list. 3. Check the filter and search functions.
	Expected Result	● Inventory displays for each of the branch selected
	Actual Results	Successful viewing of inventories with working filters and search.

Table 6.11 Test Case 6

Test Case	Test Case ID	T05
	Description	View the Notifications for all of the branches
	Precondition	A valid Admin user is logged in.
	Post Conditions	Notifications for the branches can be viewed
Test Script	Test Steps	1. Navigate to the “Notifications” page 2. Select branch from the dropdown list
	Expected Result	● Notification set displays for each of the branch selected
	Actual Results	Successful viewing of notifications

Table 6.12 Test Case 7

Test Case	Test Case ID	T06
	Description	View the Reports for all of the branches
	Precondition	A valid Admin user is logged in.
	Post Conditions	Reports for the branches can be viewed
Test Script	Test Steps	1. Navigate to the “Reports” page 2. Select the type of report to be viewed. 3. Select branch from the dropdown list
	Expected Result	● Reports display for each of the branch selected
	Actual Results	Successful viewing of reports

Table 6.13 Test Case 8

Test Case	Test Case ID	T07
	Description	Add, edit and delete Suppliers for the branch.
	Precondition	A valid Assistant user is logged in.
	Post Conditions	Suppliers can be viewed, added, edited and deleted.
Test Script	Test Steps	1. Navigate to the “Suppliers” page. 2. Select the “Add New Supplier” button. 3. Enter form data (Supplier Name, Rating, Phone Number, Email, Description) 1. Click on the supplier listed. 2. Change any detail such as supplier rating, description, phone and email. 3. Click “Confirm Changes”. 1. Click on the supplier listed. 2. Click “Delete”
	Expected Result	● Page displays “New Supplier Added Successfully” and the new supplier is viewable in “Suppliers”.

		● Updated supplier details are viewable in “Suppliers” ● Deleted supplier is no longer in the list.
	Actual Results	Successful viewing of notifications

Table 6.14 Test Case 9

Test Case	Test Case ID	T08
	Description	Order stock and update order status
	Precondition	A valid Assistant user is logged in.
	Post Conditions	Orders can be created and their status (in progress, completed, cancelled) can be updated.
Test Script	Test Steps	1. Navigate to the “Orders” page. 2. Select the “Place New Order” button. 3. Enter form data (Order Quantity, Order Name, Order Total, Date, Time, Supplier and Branch which is auto set to the current user branch) 4. Select “Place New Order” 1. Click the listed supplier 2. Select the status to change to from the dropdown. 3. Select “Confirmed Changes”
	Expected Result	● Page displays “New Order Added Successfully” and the new order is viewable in “Orders”. ● Updated order details are viewable in “Orders”
	Actual Results	Successful viewing of notifications

Table 6.15 Test Case 10

Test Case	Test Case ID	T09
	Description	Add, filter, sort and delete products
	Precondition	A valid Assistant user is logged in.
	Post Conditions	Products can be added, filtered, sorted and deleted.
Test Script	Test Steps	1. Navigate to the “Inventory” page. 2. Select the “Add” button. 3. Enter form data (Name, Brand, Price, Unit Dose, Category, Form, Quantity, Expiration Date, Branch which is auto set to the user’s branch) 4. Select “Add New Product” 1. Select a category and select “Enter” 2. Select a sort-by choice and select “Enter” 3. Type any form, brand or name into search bar and select “Enter” 4. Select a category, sort-by choice and select “Enter”. 5. Select category and sort-by choice and click enter. 6. Select a sort-by choice and type any form, brand or name into the search bar and select “Enter”. 7. Select a category and type any form, brand or name into the search bar and select “Enter”. 1. Click on the product listing. 2. Select “Delete”
	Expected Result	● Page displays “New Order Product Successfully” and the new product is viewable in “Inventory”. ● Page displays all the relevant data for any of the variations of filters. ● Deleted Product is no longer in the inventory
	Actual Results	Successful adding, filtering, sorting, searching and deletion.

Table 6.16 Test Case 11

Test Case	Test Case ID	T10
	Description	Make sure the barcode for each item is unique, viewable and saveable into the user machine.
	Precondition	A valid Assistant user is logged in and there are products listed.
	Post Conditions	The barcode for each item is unique, viewable and saveable into the user machine.
Test Script	Test Steps	1. Navigate to “Inventory” 2. Make sure barcode is visible for any and every product listed 3. Click on the barcode to download the image.
	Expected Result	● The barcode for each item is unique, viewable and saveable into the user machine.
	Actual Results	The barcode for each item is unique, viewable and saveable into the user machine.

Table 6.17 Test Case 12

Test Case	Test Case ID	T11
	Description	Checkout Function
	Precondition	A valid Assistant user is logged in and there are products listed.
	Post Conditions	● Sale record is created in the database with the selected product and quantity ● Sale total is calculated based on the product price and quantity ● The products can be searched and filtered by category
Test Script	Test Steps	1. Navigate to the “Checkout” page. 2. Select some products and enter the quantity 3. Remove a product from the cart 4. Search for an item 5. Filter the list 6. Fill in the sale details (sale date, sale method) 7. Select “Checkout” button

	Expected Result	• The user is redirected to a confirmation page
	Actual Results	• The user is redirected to a confirmation page • Sale record is created in the database with the selected product and quantity • Sale total is calculated based on the product price and quantity • The products can be searched and filtered by category

Table 6.18 Test Case 13

Test Case	Test Case ID	T12
	Description	Make sure the patients can be added, listed with medical records that can be added and viewed.
	Precondition	A valid Pharmacist user is logged in.
	Post Conditions	Patients can be added, listed with medical records that can be added and viewed.
Test Script	Test Steps	1. Navigate to “Patient “Records” 2. Select “Add New Patient” 3. Fill in form data (Patient Name, Age, Phone) 4. Select “Add New Patient” button 1. Enter the name or phone number of the patient you want to look up into the search bar and click “Enter”.
	Expected Result	• User is redirected to “Patient Records” page with updated list of patients including recent addition • Patients can be searched
	Actual Results	Patients can be added, listed with medical records that can be added and viewed and within the patient records, patients can be searched.

Table 6.19 Test Case 14

Test Case	Test Case ID	T13
	Description	Manage Patient Prescription by adding medical records for patients
	Precondition	A valid Pharmacist user is logged in.
	Post Conditions	Patients' medical records can be added and viewed.
Test Script	Test Steps	1. Navigate to “Patient “Records” 2. Click on the folder icon of a patient. 3. Select “Add New Record” 4. Fill in form data (Date, Diagnosis and Treatment) 5. Select “Add New Record” button
	Expected Result	● The new medical record is listed on the patient’s page.
	Actual Results	The new medical record is listed on the patient’s page successfully.

Table 6.20 Test Case 15

Test Case	Test Case ID	T14
	Description	View waste, sales, inventory and supplier reports.
	Precondition	A valid Pharmacist user is logged in and there is available data for waste (expired products), inventory, and suppliers.
	Post Conditions	Reports can be viewed, generated and downloaded
Test Script	Test Steps	1. Navigate to the Reports tab under the Dashboard. 2. Select the type of report to view. 3. Select the time range for the report and click generate. 4. Click the “Download Report” button.
	Expected Result	The reports can be generated and downloaded.
	Actual Results	The reports can be generated and downloaded.

Table 6.21 Test Case 16

Test Case	Test Case ID	T15
	Description	View generated notifications for the user's own branch.
	Precondition	A valid Pharmacist or Assistant user is logged in and there are products that suit the condition of near expiry (<=30 days), Low in Stock (<=9 units) and Expired.
	Post Conditions	Colour-coded notifications for the specific branch are viewable.
Test Script	Test Steps	1. Create a product with a close expiry date. 2. Create an expired product. 3. Create a low stock product. 4. Navigate to the Notifications tab under the Dashboard.
	Expected Result	Colour-coded notifications for the specific branch are viewable.
	Actual Results	Colour-coded notifications for the specific branch are viewable.

Table 6.22 Test Case 17

Test Case	Test Case ID	T16
	Description	View and add appointments via a centralised branch calendar
	Precondition	A valid Pharmacist or Assistant user is logged in and there is a stock order scheduled.
	Post Conditions	The appointments and scheduled stock are listed and viewable by filtered months.
Test Script	Test Steps	1. Create a stock order and set a date. 2. Navigate to the Calendar page. 3. Click on “New” to create a new appointment. 4. Fill in the form data (Appointment Name, Date, Time, Branch which is auto set to the current branch) 5. Click “Add New Appointment” 6. Go back to the Calendar Page 7. Select the year and month to view and click “Go”.
	Expected Result	Scheduled appointments should show up in the correct month.
	Actual Results	The scheduled appointments show up in the correct month.

Chapter 7: CONCLUSION

In conclusion, Phase 1 of the project concerns conceptualisation through background studies, literature reviews to compare suitable technologies, the requirements for the system from a user and developer perspective as well as the design of the application in terms of user interface, experience and the data design. The main problem faced here is time management and understanding of different components and how they interface with each other. Now that a deeper understanding has been gained, it will be a little bit easier to move on to Phase 2.

Phase 2 of the project focuses on putting the pieces together from project Phase 1 and comparing which python libraries are best suited for prediction. There are still aspects of Phase 1 that required fine tuning to be optimised especially in the database and user interface perspective. Some system and data design were changed to prevent redundancy and to further optimize the system.

Still there are some limitations for this project which have a lot of room for improvement. For instance, more flexible inventory handling, external barcode scanning integration, and more valuable reporting and insights. Overall this was a very invaluable experience and learning opportunity.

REFERENCES

Alnahas, F., Yeboah, P., Fliedel, L., Abdin, A. Y., & Alhareth, K. (2020). Expired Medication: Societal, Regulatory and Ethical Aspects of a Wasted Opportunity. *International Journal of Environmental Research and Public Health*, *17*(3), 17. 10.3390/ijerph17030787

Bialas, C., Revanoglou, A., & Manthou, V. (2019). Improving hospital pharmacy inventory management using data segmentation. *American Journal of Health-System Pharmacy*, *77*(5), 7. 10.1093/ajhp/zxz264

Kheybari, S., Naji, S. A., Rezaie, F. M., & Salehpour, R. (2019, March 04). ABC classification according to Pareto's principle: a hybrid methodology. *OPSEARCH*, *56*(539–562 (2019)), 24. 10.1007/s12597-019-00365-4

Kumar, A., & Shukla, A. C. (2022, May). Selective Inventory Control Using ABC And FSN Analysis in Retail Sector: A Case Study. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, *10*(5), 12. 10.22214/ijraset.2022.43648

Mahyadin, F. A., Saad, R., Mohd Asaad, M. N., & Zien, R. (2015, September 22). *The Influence of Inventory Management Practices Towards Inventory Management Performance in Malaysian Public Hospitals*. IAR Journal. Retrieved October 25, 2022, from http://www.iarjournal.com/wp-content/uploads/IBTC2015-p142-148.pdf

Saha, E., & Ray, P. K. (2019). Modelling and analysis of inventory management systems in healthcare: A review and reflections. *Elsevier Computers & Industrial Engineering*, *137*(106051), 16. 10.1016/j.cie.2019.106051

Said, B., Benhra, J., & Hassani, H. E. (2013, August). Causal Method and Time Series

Forecasting model based on Artificial Neural Network. *International Journal of Computer Applications*, *75*(7), 7. 10.5120/13126-0482

Saripin, N. H. (2019, 7 2). *Medication Wastage by Patients: The Insights*. HCTM UKM. Retrieved October 25, 2022, from https://hctm.ukm.my/farmasi/wp-content/uploads/2020/09/11.2018.-Medication-wastage-by-patients.pdf

West, L. M., Diack, L., Cordina, M., & Stewart, D. (2014). A systematic review of the literature on 'medication wastage': an exploration of causative factors and effect of interventions. *International Journal of Clinical Pharmacy*, *36*(5), 9. 10.1007/s11096-014-9981-2

The different dosage forms manufactured in the industry. (n.d.). Retrieved January 29, 2023, from https://www.lfatabletpresses.com/articles/dosage-forms-manufactured

MySQL 8.0 Reference Manual. (n.d.). Retrieved January 29, 2023, from https://dev.mysql.com/doc/refman/8.0/en/

Barcode. (2015, February 17). Retrieved from GTIN INFO Global Trade Item Number Website: https://www.gtin.info/barcode-101/