Acknowledgement

Before all, we would like to thank almighty God for blessing us to finish this project. We feel extreme gratitude to our parents, families and friends who helped us throughout the whole process of learning and doing the project. Without your help, none of this would be possible. Then we would like to thank our advisor Tesfaye Eyasu for his insightful and consistent feedback. The completion of this project would have been nearly impossible without your positive guidance. Lastly, we would like to express our appreciation for all the pharmacies that helped us in the process of the project; Ato Mubarak from Ahyan Pharmacy and W/o Lemlem from Khfaziel Pharmacy, your help is greatly appreciated.

TABLE OF CONTENTS

List Of Tables

List Of Images

Abstract

This project is intended as Partial Fulfilment of the Requirements for the B.Sc. Degree in Computer Science. The goal of this project is to create a pharmacy management system with artificial intelligence decision support that will enable any pharmacy to manage its daily operations and make decisions using the decision support it offers.

The evident deficiencies in the current pharmacy management system are addressed in the paper below. Using Object-Oriented Software Engineering (OOSE) techniques, it addresses the various shortcomings of the current system. Our team used different data collections like interview and document review to collect data, understand the requirements and design the necessary application. We also utilised ASP.NET Core 6 and React to develop a web application that has an Inventory Management System, Cashless Payment System, and a Decision Support System in order to achieve the desired objective of the project.

In conclusion, this project does exactly what it intended to do and solves the problem facing the pharmacies facing Ethiopia today. During the course of the project, our team learned about the need in our country to catch up with the rest of the world in terms of efficiently run businesses and health sector services.

1. Introduction

1.1. Background

In any community, health care providing facilities are one of the most important institutions. These health care institutions can be hospitals, clinics, drug stores, pharmacies and pharmaceutical manufacturers. In Ethiopia, the availability and the management of these essential institutions is very poor. Among the above mentioned facilities, the pharmaceutical industry holds a major place. The number of retail pharmacies in the country fall far short of the overall demand. As of 2019, the total retail pharmacy market in Ethiopia is 3,327 pharmacies and 4, 476 drug stores [1]. According to Proclamation 1112/2019, every retail pharmacy should install a quality control mechanism for their stores [2]. These quality controls have to do with proper purchase and storage of medicines and regular check of the expiry dates of those medicines. Therefore, while there is a lot of work to be done to increase the number of these institutions, good management of the existing ones is essential to improve one major aspect of the healthcare service in the country.

To achieve the needed quality assurance controls, an efficient and effective management system for every pharmacy and drug store is necessary. Pharmacies should keep track of where their medicines come from, when it is purchased, when it will expire and for how long it has been sitting on their shelves. One of the most important things in the management process is keeping track of the expiry dates of the medicines that are being sold in the store. Expired medicine is very dangerous and if it finds a way into the community, the results are severe illness and often death. So, there needs to be a way to reduce and eventually eliminate the occurrence of this event. This and other problems facing the currently deployed management systems is discussed in the next section. The proposed system will address those issues and create a new, efficient management system for any pharmacy retailers to deploy and easily meet the necessary quality control requirements.

The reason for choosing this topic is because it will create an easily deployable system that will ensure that pharmacies will manage their stores to the highest quality while increasing their profits as a business as well. This inturn will help every person in the community to have access

to good, well regulated and quality medicines. The importance of well regulated medicines available in need cannot be stressed enough. Even though we currently are short of meeting that availability in numbers in our country, we can ensure better distribution and quality assurance by having a better management system for the existing pharmacies and drug stores.

1.2. Statement of the Problem

Pharmacies need to have an efficient management system in order to stay in business and better provide a vital service to the community. However, most pharmacies in our country rely on a traditional paper-based system for the management of their day to day business. The use of this outdated system has left most pharmacies in dire need of their entire system.

With the exception of chain pharmacies owned by huge corporations, almost all pharmacies use a paper based inventory management system. They use a filebook to record their wholesale purchase of medicine and their daily sales. This includes writing every medicine that is brought in; recording every medicine that is sold every time a sale is made, and balancing the inventory and the sales record manually to know what they sold and what's left in their shelves. This process comes with lots of shortcomings. Human error is the most common problem encountered during the process. People may forget what they sold when there are multiple customers that visit the pharmacy at once. The durability of the filebooks is also another issue that is faced by the pharmacies.

The payment system currently in place in most pharmacies also presents an additional challenge. Most pharmacies use cash registers during sales and are left to manage huge amounts of money in cash everyday. In addition, the tracking of the income and expenses is time consuming and error prone.

In the business world, making informed decisions is a very important factor that predicts success. Pharmacy owners need to know which products are being sold faster, which products spend the longest time on the shelf, which ones expire without being sold etc. With this information, the owners can make better decisions when it comes to which medicines to

purchase and provide to the community they are serving. Currently, coming to these decisions is a long, energy consuming and expensive process.

The importance of well run pharmacies can not be overemphasised. Not only will it benefit the pharmacies themselves, the community that utilises their service is also left in a better state as a result. The proposed system will eliminate all of the problems mentioned above and make the management of pharmacies a much easier task by automating their inventory management, decision support and payment systems. Upon the completion of this project, we are positive that all pharmacies that implement the system will have smooth running, profitable businesses that will benefit everyone in the community.

1.3. General Objective

The general purpose of this project is to achieve easy and efficient management of pharmacies by automating their inventory management, providing cashless payment service, and implementing a fully automated decision support system. To achieve the general objective stated above, the following specific objectives should be met.

1.4. Specific Objectives

- Discover and collect requirements from randomly selected pharmacies for the requirement elicitation process
- Analyse the requirements and describe the system using different models
- Design the system by dividing it into subsystems
- Integrate each subsystem to make the whole functional system
- Test and verify that the specified documentation is implemented consistently

1.5. Scope of the Project

The system being developed is not specifically designed to suit the needs of a single pharmacy. Instead, it is designed to be implemented for any pharmacy with a simple customization. Any pharmacy that has not implemented a computerised management system for their store can use this system and will solve the difficulties that are mentioned in the problem statement.

The system will provide the following major functionalities:

- Inventory Management
    - ➔ The system will provide a form to register incoming medicines purchased from wholesalers
    - ➔ The system will keep track of medicines that are sold
    - ➔ The system will keep track of the expiry dates of every medicine currently on the shelf
    - ➔ The system will keep track of the providers of each medicine
- Cashless Payment System
    - ➔ They system will provide an online payment system
- The system will provide a Decision Support System (DSS)

1.6. Limitations

The collection of prescriptions is not included in the proposed system as they need to be collected and provided for the Health Bureau as per the regulation. Therefore, all the prescriptions provided by the customers are collected and stored physically.

1.7. Methodology

For the development of the proposed system, a combination of Agile and Waterfall Methodology is implemented. Scrum development framework is specifically chosen in the development process. Scrum is a project management framework that focuses on teamwork and having a deliverable product as fast as possible. It prioritises customer interaction, feedback and adjustments rather than focusing on documentation and prediction. Scrum has numerous advantages in the development process. Some of the advantages that made it a better choice are:

- Quick and efficient delivery of project deliverables
- The ability to breakdown the system into small and easily manageable sprints
- Allows for easier testing process
- Easy adaptation to changing requirements during the development process

- Daily scrum meetings will highlight each member's unique contribution in the development

1.7.1. Data Collection and Sampling

The proposed system is being designed to meet the needs of any pharmacy that wants to implement a computerised management system. Therefore, the data collection is conducted on multiple pharmacies in order to have enough data for the requirement identification process. The pharmacies chosen in the data collection range in location, size and ownership, making them an ideal source to make a generic pharmacy management system.

To collect the data from the selected pharmacies, Observation, Interview and Document Review methods are used. Every pharmacy's operations will be observed by the team members and then the owners and workers are interviewed informally. A document review will then be conducted by each team member. Through this process the main requirements and specific details of most features are extracted.

1.7.2. Analysis and Design

Object-Oriented Analysis and Design methods focus on capturing the existing system, to be able to get hold of the entire perspective of what the whole system is trying to achieve [3]. Discussing requirements is one of the most important stages of software development. Requirements Elicitation is a complex process of gathering, organising, and structuring requirements [4]. In the Requirements Elicitation phase, we will specifically identify use cases, objects and describe the relationships using conceptual models.

Scenarios and other object models namely sequence diagrams, activity diagrams, class diagrams, and state-chart diagrams will be used to model the system. All the diagrams used in the modelling process adhere to the Unified Modelling Language standards.

Tools used in this phase are:

Google Docs - to prepare the necessary documentation

LucidChart Diagrams - to prepare the UML diagrams in the documentation

1.7.3. Implementation and Testing

In this phase, all the deliverables of the previous phases i.e., documents and models are put into action [4]. An executable source code is produced that implements the expected functionalities into a single system. In order to achieve this the following tools are used.

- Programming Languages
  - ➔ C# Version 10
  - ➔ JavaScript ES2015
- Database
  - ➔ MSSQL
  - ➔ Microsoft Identity Authentication
- Frameworks
  - ➔ React Version 18.2.0
  - ➔ ASP.NET Core 6, REST APIs with .NET and C#
- IDEs
  - ➔ Visual Studio 2022 Community
  - ➔ Visual Studio Code

When this phase is completed, all subsystems developed in the process are expected to integrate and interact with each other to achieve the overall goal. Once that is achieved, an integration testing will be carried out to verify if the product does what it is supposed to do. Integration testing is a type of testing where software modules are integrated logically and tested as a group[10]. The goal of the testing process is to expose, and as a result fix, defects that may arise when the subsystems are integrated together to make the whole system.

1.8. Beneficiaries and Users

Pharmacies

Any pharmacy that chooses to implement the proposed system will greatly benefit from the smooth running of their business. A better inventory management and DSS will allow for efficient and profitable operation.

Customers

The system will allow customers that are prescribed long term medicines they have to purchase on a regular basis (for example people with diabetes or people undergoing cancer treatment) to have their medicine delivered right to their doorsteps on time. The system will also notify them about when their next medicine purchase is, helping them to buy the medicines in advance and never skip any dose.

1.9. Task Breakdown

Automated Management System for Pharmacies

Project Proposal

Requirements Analysis and Specification

1.1.1. Requirement Identification

1.1.1.1. Observation

1.1.1.2. Interview

1.1.1.3. Document Review

1.1.2. Requirement Elicitation

1.1.2.1. Requirement Specification

1.1.2.2. Functional Requirement

1.1.2.3. Non-functional Requirement

1.1.2.4. Identifying Actors

1.1.2.5. Identifying Scenarios

1.1.2.6. Identifying Use Cases

1.1.3. Requirement Analysis

1.1.3.1. Functional Model

1.1.3.2. Dynamic Model

1.1.3.3. Static Model

Outcome: Requirement Specification Document

System Design

1.1.4. System Decomposition

1.1.5. Addressing Design Goals

1.1.6. User Interface Design

1.1.7. Database Design

1.1.8. Architectural Design

Outcome: System Design Document

Object Design

1.1.9. Subsystem Decomposition

1.1.10. Class Interface

1.1.11. Package Diagram

Outcome: Object Design Document

System Implementation

1.1.12. Sprint 1: Inventory Management System

1.1.12.1. Front End Design and Development

1.1.12.2. Back End Development

1.1.12.3. Testing

1.1.12.4. User Manual

1.1.13. Sprint 2: Online Payment API Integration

1.1.13.1. Back End Development

1.1.13.2. Testing

1.1.14. Sprint 3: Decision Support System

1.1.14.1. Front End Design and Development

1.1.14.2. Back End Development

1.1.14.3. Testing

1.1.14.4. User Manual

1.1.15. Sprint 4: Delivery System with Mobile Application

1.1.15.1. Front End Development

1.1.15.2. Back End Development

1.1.15.3. Testing

1.1.15.4. User Manual

Outcome: Software Products and Implementation Report

1.10. Feasibility Analysis

The purpose of this feasibility analysis is to estimate the potential success of the project by considering important components of the project. The important components this analysis takes into consideration are its technical and operational feasibility.

Technical Feasibility

The proposed system can be sufficiently developed, run and managed within the currently available technologies. The hardware requirements of the system is the availability of computers that can run a Windows 10 operating system. With regard to the software requirements, the proposed system is easily affordable and is developed using freely available technologies that are subject to upgrades.

Operational Feasibility

The proposed system is used to eliminate the outdated and problematic paper based management system of pharmacies. The automation process can be a little tedious at the beginning but after the completion of the data migration, everything is expected to go smoothly. Therefore, the process of automation is deemed useful.

1.11. Project Schedule

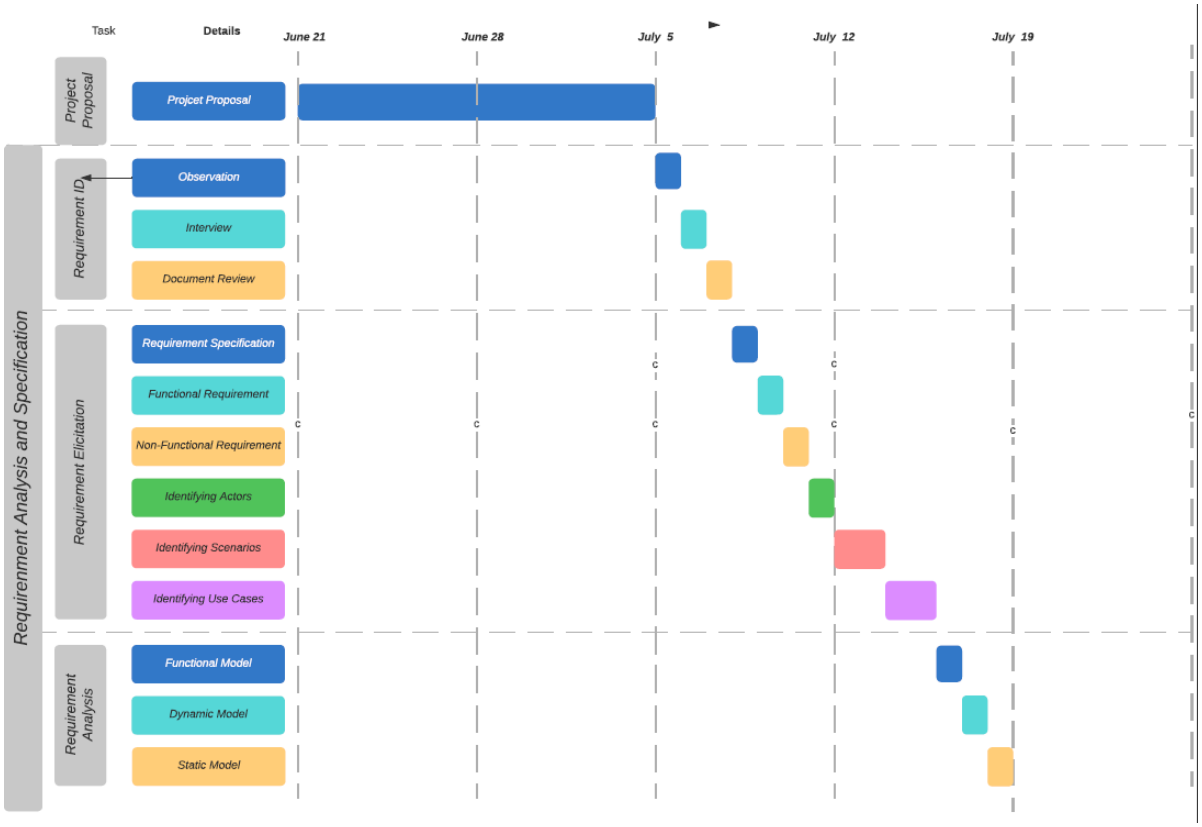

Figure 1: Project Schedule (Part 1)

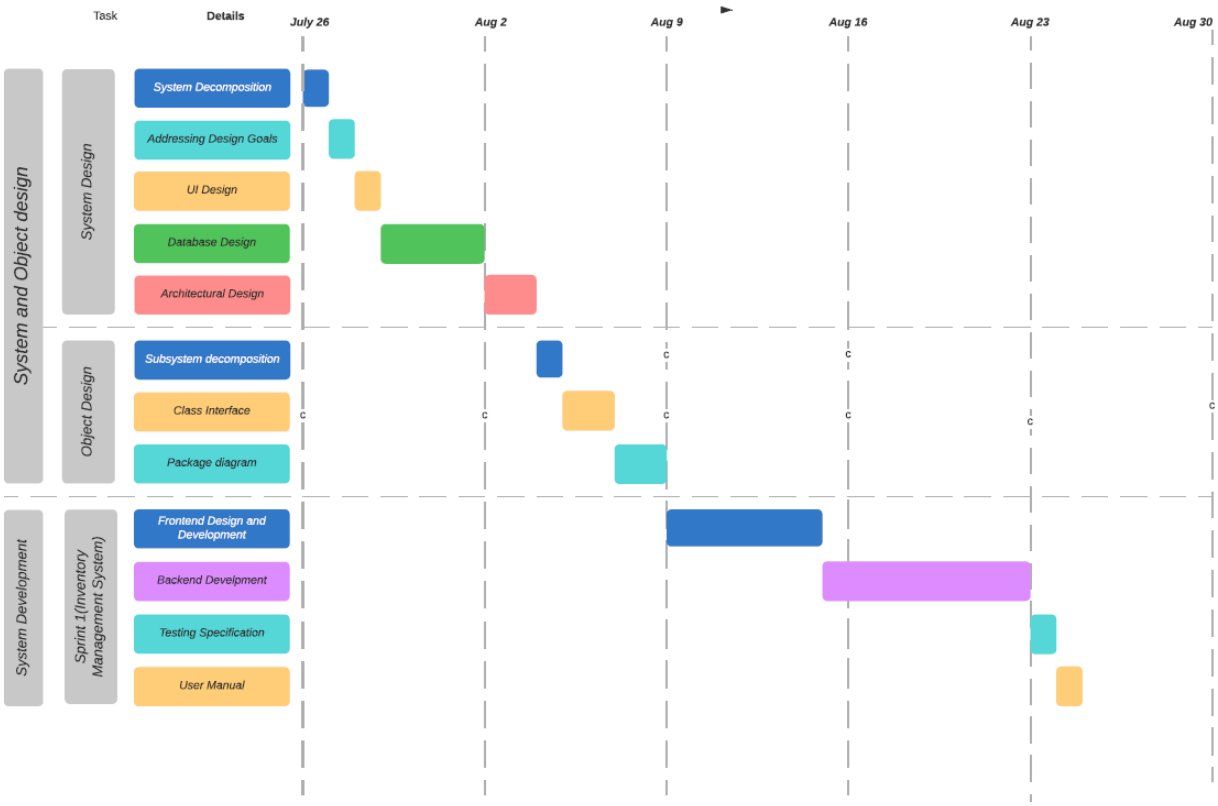

Figure 2. Project Schedule (Part 2)

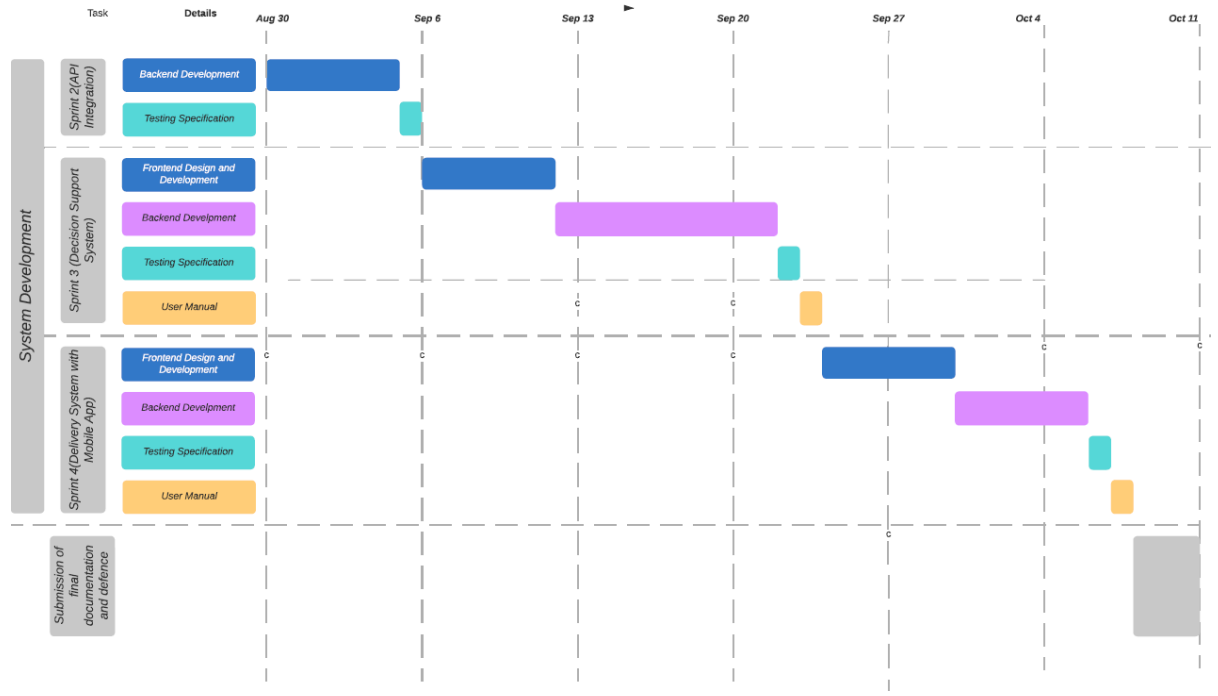

Figure 3: Project Schedule (Part 3)

2. System Requirements Specification Document

2.1. Introduction

Overview

The importance of efficient and quality healthcare providence facilities in a country is something that cannot be overlooked. Pharmacies are one of the integral parts of a healthcare system. According to Proclamation 1112/2019, every retail pharmacy, government owned or private, should install a quality control mechanism [3]. The most important thing for these quality control mechanisms is an efficient management system for the stores. The management system that is deployed at the moment is lacking in various aspects and makes managing and controlling medicine throughout the country unnecessarily difficult. This project aims to provide an applicable solution to the problems facing the current system.

Purpose

The purpose of the proposed system is to create a computer based management system that will replace the currently used paper based system. The system will provide a reliable and efficient inventory management system, an online payment system and an AI based decision support system. In the end, the system aims to aid pharmaceutical retailers to better control their businesses and provide a quality healthcare service to the country. In the meantime, the data driven decision support system will provide an insight into how to make the businesses profitable and everlasting.

Scope

The proposed system being developed is not specifically designed to suit the needs of a specific pharmacy. Instead it is designed to be implemented to any pharmacy, providing services that will solve the problems and difficulties of most pharmacies.

In accordance with the problems identified, the system will have an inventory management system that will take care of registering, selling and tracking the expiry dates of medicines. The system will also provide a cashless payment system and a decision support system that analyses

data created during transactions and generates reports that make decision making well informed.

2.2. Existing System

Currently, apart from chain pharmacy retailers owned by large corporations, most of the pharmacies in Ethiopia run their business using a paper based management system. The few pharmacies that use a computerised management system mostly use a system developed by CNET Software Technologies. The system provided by the software company, however, provides mainly an inventory management system and a cash payment system only. As of now, a decision support system or a cashless online payment system is not something that has been deployed in their software updates.

In the paper-based systems, on the other hand, retailers use physical forms and file books to record and keep track of everything they buy and sell. And finding any errors or balancing the stock is done manually by checking each form and file and balancing the ins and outs. Below is a thorough discussion of the process that has to be followed by most pharmacies and drug stores.

When obtaining medicine and other products from suppliers, every pharmacy keeps the official receipt and attachment that is issued for them during payment. Then they will conduct an initial inspection of each batch of medicine to ensure they are not damaged and fit the quality requirements needed to distribute among the public. After this, every medicine should be transported to and stored in a designated store room. Every item that is procured should first be stored in the warehouse for further inspection. After a thorough inspection and approval of a qualified pharmacist, the item is then deemed fit for distribution and can be transferred to the shelf for sale.

During this process, the pharmacist records every medicine that is approved for sale in a stock card. A stock card is a document that is used to track stock based on issuing and receiving orders [6, p. 36]. After the items are inspected and approved, they can then be distributed to the public according to the distribution guidelines provided by the MoH. During sales of a medicine, every transaction is recorded on a document called a bin card. A bin card is similar to

a stock card but it is prepared for each product. Every transaction that involves any medicine, like issued, received, loss/adjustment, is recorded in the bin card that is kept beside it at all times [6, p.36].

Throughout the process described above, whenever a transaction is recorded in the bin card, the stock balance must be updated. The bin card and stock card are then checked, along with a physical count, to determine if there is any product loss. It is critical for pharmacies to periodically balance their stock in order to know how much they have distributed, or lost due to various reasons. Doing that in the currently wide-used system requires going through their bin card and stock cards and balancing for loss/adjustment manually, which is tedious and error prone. In addition, decision making about profitable business choices also becomes difficult and complicated using the current system.

As it is discussed above, the currently deployed management system of pharmacies, one of the vital hospital services, is very inadequate in terms of efficiency, accuracy and swiftness. The capacity for human error is high; it makes decision making at the managerial position complicated, and takes a lot of time and effort to provide the high standard service that is necessary. The proposed system will solve all these problems by computerising the pharmacy management system while upholding the high standards that are required by the MoH.

2.3. Proposed System

2.3.1. Functional Requirements

Functional requirements are requirements with what the system should do. They describe the interactions between the system and its environment independent of its implementation [3, p.125]. These requirements can be written in a variety of ways, ranging from general terms that cover what the system should do to very specific requirements that reflect local working practices [5, p. 105]. Below are the functional requirements for the proposed system.

FR-1: The system shall allow users to log in as manager or store keeper using username and password.

FR-2: The system shall the manager add customers to a notification service using their names and phone number.

FR-3: The system shall notify customers subscribed to a notification service using SMS about their next purchase date and times of taking medicine.

FR-4: The system shall let the manager register incoming medicines and other products into the store room.

FR-5: The system shall let the manager or qualified personnel approve or decline a medicine to be sold.

FR-6: The system shall let the manager add medicines to the dispensary.

FR-7: The system shall generate a unique identification number for each medicine or product.

FR-8: The system shall notify the manager about any medicine or other products that will expire within the coming 6 months or less.

FR-9: The system should display a push notification urging to remove medicine that has one month or less expiration date from the dispensary.

FR-10: The system shall let the manager remove expired or damaged medicine and other products from the dispensary.

FR-11: The system shall let the manager or store keeper to sell medicine that is on the shelf.

FR-12: The system shall accept and process cashless payment for the sale of any medicine.

FR-13: The system shall let the manager or store keeper return medicine according to the store's return policy.

FR-14: The system shall let the manager or store keeper see the availability of a product in the dispensary or store room.

FR-15: The system shall let the manager or store keeper have full information about the medicine in the shelf or warehouse.

FR-16: The system shall keep a log of every transaction that is done in the database.

FR-17: The system shall produce a stock card within the required time frame on demand.

FR-18: The system shall generate a bin card for any item in the dispensary on demand.

FR-19: The system shall keep track of the shelf life of every medicine and product.

FR-20: The system shall keep track of the purchasing price and selling price of every item on the shelf.

FR-21: The system shall generate a profit/loss report.

FR-22: The system shall utilise all of the transaction information and other information in the database to create a decision support system.

FR-23: The system shall analyse data from past transactions and present a report of the best possible options for business in the future.

FR-24: The system shall generate a product recommendation report.

FR-25: The system shall generate a sales forecasting report.

2.3.2. Non-functional Requirements

Non-functional requirements are requirements that are not directly related to any of the specific services or functionalities provided by the system. They describe different attributes of the system like reliability, performance, usability etc. Non-functional requirements usually specify or constrain characteristics of the system as a whole [5, p. 107]. In Agile development, an adaptive, incremental approach of defining and implementing is necessary due to the dynamic nature of the development method. Below are the non-functional requirements that the proposed system should meet.

Usability: the system shall be easy to learn and operate. A user should be able to learn all the operations of the system after an hour of training. It should have a simple and consistent user interface that enables users to learn to operate, prepare inputs for and interpret outputs of the system with ease. It should also have a well-written documentation and a thorough user manual as well.

Dependability: the system should be able to reliably perform its required functions under normal circumstances for more than 12 hours without any problems. It should be able to detect and handle any specified errors that may arise from invalid input or stressful environment conditions, and function correctly despite encountering those errors.

Performance: The system shall have a quick response time for any of the users' requests. All of users' actions like data entry, modification and queries should be completed within or under 5 seconds.

Security: The system should ensure the confidentiality, availability and integrity of all the data that it generates or is put into it.

Safety: The system shall keep a backup of the data it uses and generates on a regular basis (according to the bulk of data and policy of the pharmacy) in case of an unprecedented incident caused by factors that are out of control like natural disasters or complete hardware failure. The backup should be stored in a remote server and be accessible at any time it is needed.

2.3.3. Business Rules

Business rules are common practices that are followed in the day to day running of businesses. These rules might be government policiesthat must be followed or practices that are widely followed in the industry. Business rules can either be policies or constraints. Below are some of the business rules that are followed in the pharmacy retail business.

BR1: A pharmacist should ask for and carefully check prescriptions before giving out medicines for any customer.

Type: Policy

Static/Dynamic: Static

Source: Document Review

BR2: Every medicine that is brought into a pharmacy or drugstore should be inspected and deemed fit for sale by an expert personnel before it makes its way to the dispensary and the public.

Type: Policy

Static/Dynamic: Static

Source: Interview

BR3: Any medicine that has one month or less for its expiration date should be removed from the dispensary and be reported to the concerned government body for proper disposal.

Type: Constraint

Static/Dynamic: Dynamic

Source: Document Review

BR4: The profit margin for any medicine shall not exceed 25% of the purchasing price.

Type: Constraint

Static/Dynamic: Static

Source: Interview

BR5: A medicine can not be returned after 24 hours it is purchased.

Type: Policy

Static/Dynamic: Dynamic

Source: Interview

BR6: Narcotic and Psychotropic drugs should be kept in a separate shelf with locks and only be dispensed by a licensed pharmacist only.

Type: Policy

Static/Dynamic: Static

Source: Interview

2.4. Proposed System Models

The system models in terms of scenarios, use case descriptions, use case diagrams, class diagrams, sequence diagrams, activity, and user interface will be covered in this part.

2.4.1. Identified Actors

Manager: A person who is responsible for the day to day running of the pharmacy. A manager can be the owner of the pharmacy or a person appointed to the position by the owners or management of the business.

Pharmacist: A person that sells medicine with a licence to dispense medicine to the public.

Customer: A person that purchases medicine for use.

2.4.2. Scenarios

Scenario Number	SC-01
Scenario Name	Login as Manager
Participating Actors	Initiated By: Manager
Flow of Events	1. The manager opens the app into the login page. 2. The manager enters their username and password into their respective fields. 3. The manager clicks on the login button. 4. The manager is redirected to their home page.

Table 1. Scenario for Login as Manager

Scenario Number	SC-02
Scenario Name	Create a Pharmacist Account
Participating Actors	Initiated By: Manager Communicated With: Pharmacist
Flow of Events	1. The manager opens their home page. 2. The manager clicks on add pharmacist option on the menu. 3. A registration form is displayed. 4. The manager provides the necessary information and clicks on the register button. 5. The confirmation alert is displayed. 6. The manager provides the username and password for the pharmacist.

Table 2. Scenario for Create a Pharmacist Account

Scenario Number	SC-03
Scenario Name	Log in as Pharmacist
Participating Actors	Initiated By: Pharmacist
Flow of Events	1. The pharmacist opens the app into the login page. 2. The pharmacist enters their username and password into the respective fields. 3. The pharmacist clicks on the login button. 4. The pharmacist is redirected to their homepage. 5. A notification is displayed prompting them to change the password for the account.

Table 3. Scenario for Login as Pharmacist

Scenario Number	SC-04
Scenario Name	Change Password
Participating Actors	Initiated By: Pharmacist/ Manager (User)
Flow of Events	1. The user logs into their account. 2. The user opens the profile page; clicks on the change password button. 3. A change password form is displayed. 4. The user provides all the necessary information into their respective fields. 5. The user clicks on the Change Password button. 6. The user is logged out of their account and redirected to the login page.

Table 4. Scenario for Change Password

Scenario Number	SC-05
Scenario Name	Register User for Notification Service
Participating Actors	Initiated By: Pharmacist, Manager (User) Communicated With: Customer
Flow of Events	1. The user opens the Services page. 2. The user clicks on the Notification Services button. 3. A Notification Services page is displayed. 4. The user clicks on the New Customer button. 5. A customer registration form is displayed. 6. The user fills in the form with the necessary information and clicks on the Register button. 7. A confirmation SMS text is sent to the customer. 8. The user is redirected to the Notification Services page.

Table 5. Scenario for Register User for Notification Service

Scenario Number	SC-06
Scenario Name	Register Medicine
Participating Actors	Initiated By: Manager
Flow of Events	1. The manager opens the Register Medicine page. 2. A register form is displayed. 3. The manager enters the necessary information into the appropriate fields. 4. The manager clicks on the Add Item button. 5. A notification of confirmation is displayed. 6. The manager is redirected to the Register Medicine page.

Table 6. Scenario for Register Medicine

Scenario Number	SC-07
Scenario Name	Search for an Item
Participating Actors	Initiated By: Pharmacist, Manager (User)
Flow of Events	1. The user enters the necessary information into the search bar on the homepage. 2. The user clicks on the search button. 3. All the relevant information is displayed.

Table 7. Scenario for Search for an Item

Scenario Number	SC-08
Scenario Name	Sell an Item
Participating Actors	Initiated By: Pharmacist
Flow of Events	1. The pharmacist searches for an item. 2. The pharmacist adjusts the quantity and clicks on add to cart. 3. After all the items are added to cart, the pharmacist clicks on checkout. 4. A payment page is displayed. 5. The customer pays for the items online. 6. The system confirms the payment. 7. The system prompts the pharmacist to print a physical receipt and attachment. 8. If yes, the system prints out receipt and attachment. 9. The pharmacist is redirected to their homepage.

Table 8. Scenario for Sell an Item

Scenario Number	SC-09
Scenario Name	Return an Item
Participating Actors	Initiated By: Pharmacist, Customer Communicated With: Manager
Flow of Events	1. The pharmacist searches for an item

Table 9. Scenario for Return an Item

Scenario Number	SC-10
Scenario Name	Update Item Information
Participating Actors	Initiated By: Manager
Flow of Events	1. The manager searches for an item. 2. The manager clicks on the item they want. 3. A page displaying the information is displayed. 4. The manager clicks on the edit button. 5. A page with editable fields is displayed. 6. The manager edits the information. 7. The manager clicks on the save changes button. 8. The manager is redirected to the page displaying information about the item.

Table 10. Scenario for Update Item Information

Scenario Number	SC-12
Scenario Name	Remove Expired Items
Participating Actors	Initiated By: Pharmacist, Manager (User) Communicated With: Manager
Flow of Events	1. The system sends notification about an expired item to the home page of the user. 2. The user clicks on the notification. 3. A search list of every expired item is displayed. 4. The user collects all the listed items from the dispensary/ store room and stores them separately. 5. The user clicks on the remove all button. 6. A notification about the success of the operation is displayed. 7. The user is redirected to their home page.

Table 11. Scenario for Remove Expired Items

Scenario Number	SC-13
Scenario Name	Generate a Bin Card
Participating Actors	Initiated By: Manager
Flow of Events	1. The manager clicks on the reports page. 2. The manager clicks on the bin card button. 3. A bin card page with a search bar is displayed. 4. The manager searches for a medicine and clicks the search button. 5. All relevant results are displayed. 6. The manager clicks on the desired item. 7. A bin card is displayed with all the necessary information filled out automatically. 8. The manager clicks on the print button. 9. The system prints the bin card. 10. The manager is redirected to the reports page.

Table 12. Scenario for Generate a Bin Card

Scenario Number	SC-14
Scenario Name	Generate a Stock Card
Participating Actors	Initiated By: Manager
Flow of Events	1. The manager clicks on the reports page. 2. The manager clicks on the stock card button. 3. A stock card page with fields is displayed. 4. The manager enters the time interval needed. 5. The stock report card is displayed with all the necessary information automatically filled. 6. The manager clicks on the print button. 7. The system prints the card. 8. The manager is redirected to the reports page.

Table 13. Scenario for Generate a Stock Card

Scenario Number	SC-15
Scenario Name	Generate a Returned/ Unusable Commodities Report
Participating Actors	Initiated By: Manager
Flow of Events	1. The manager clicks on the reports page. 2. The manager clicks on the Returned/ Unusable Commodities button. 3. A RUC page is displayed. 4. The manager enters the time interval they want. 5. The Returned/ Unusable Commodities is displayed with all the necessary information automatically filled. 6. The manager clicks on the print button. 7. The system prints the report. 8. The manager is redirected to the reports page.

Table 14. Scenario for Generate Returned/ Unusable Commodities Report

Scenario Number	SC-16
Scenario Name	Generate Profit/Loss Report
Participating Actors	Initiated By: Manager Communicated With: Owners/ Administration
Flow of Events	1. The manager opens the reports page. 2. The manager clicks on the finances button. 3. A profit/loss page is displayed. 4. The manager enters the time interval they want. 5. The manager clicks the next button. 6. A report of net and gross profit/loss is displayed. 7. The manager clicks on the print button. 8. The system prints the report. 9. The manager is redirected to the reports page. 10. The manager forwards the report to the owners/ administration.

Table 15. Scenario for Generate Profit/Loss Report

Scenario Number	SC-17
Scenario Name	Generate Product Recommendation
Participating Actors	Initiated By: Manager Communicated with: Owners/ Administration
Flow of Events	1. The manager opens the DSS page. 2. The manager clicks on the Product recommendation button. 3. A report on which products would sell the most is displayed. 4. The manager clicks on the print button. 5. The report is printed. 6. The manager is redirected to the DSS page. 7. The manager forwards the report to the owners/ administration.

Table 16. Scenario for Generate Product recommendation

Scenario Number	SC-18
Scenario Name	Generate Sales Forecasting
Participating Actors	Initiated By: Manager Communicated With: Owners/ Administration
Flow of Events	1. The manager opens the DSS page. 2. The manager clicks on the Sales forecasting button. 3. A report is displayed. 4. The manager clicks on the print button. 5. The report is printed. 6. The manager is redirected to the DSS page. 7. The manager forwards the report to the owners/administration.

Table 17. Scenario for Generate Sales Forecasting

2.4.3. Use Case Diagram

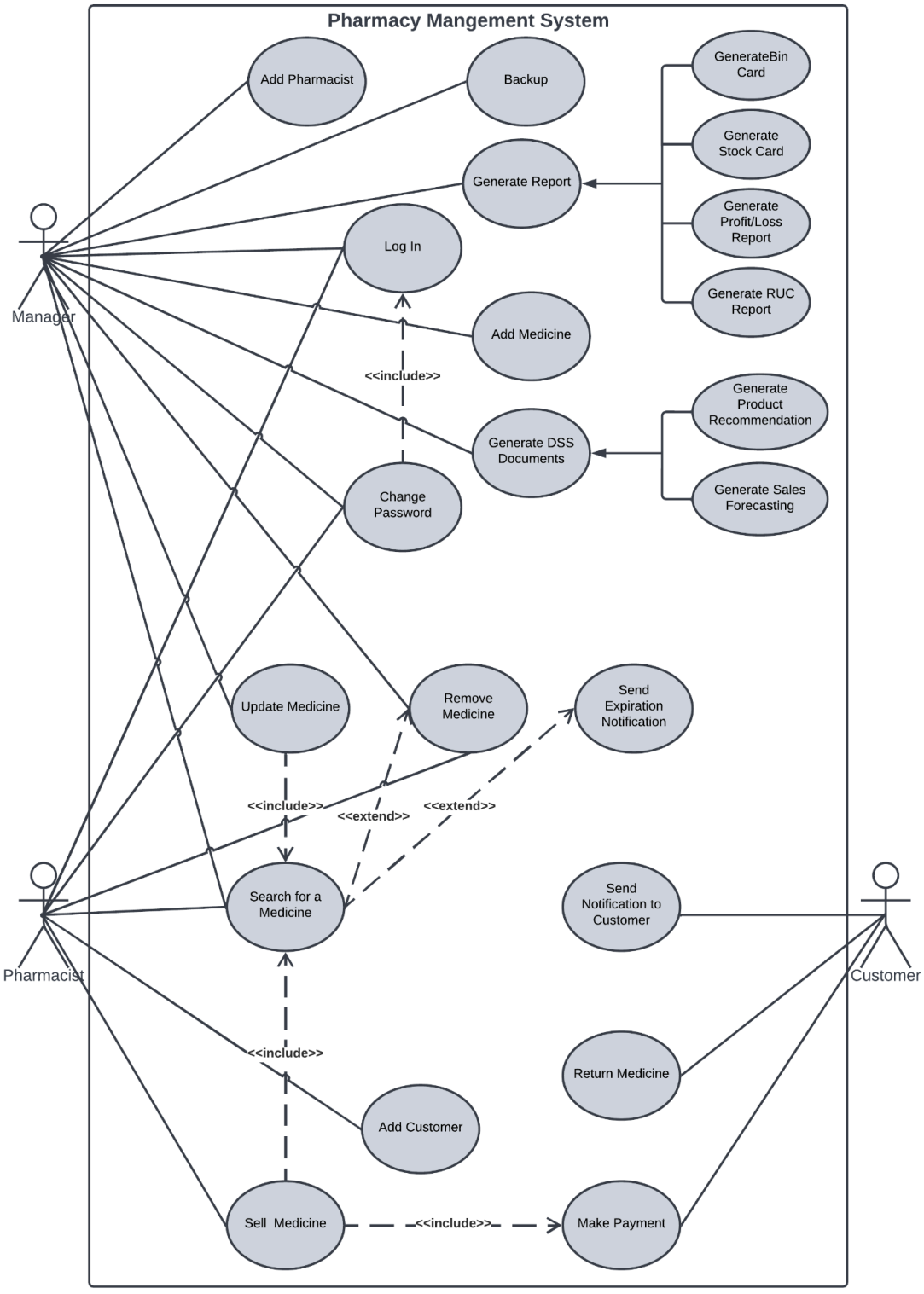

Figure 4. Use Case Diagram

2.4.4. Use Case Descriptions

Use Case ID	UC01
Use Case Name	Log In
Actors	Manager, Pharmacist (User)
Description	Enables staff to provide username and passwords to log into their account and use the application.
Entry Condition	The user must have a username and password.
Flow of Events	1. The application presents the “Login” page. 2. The user provides the username and password in their respective fields and clicks on the “Login” button. 3. The application fetches the credentials and verifies if they are correct. 4. The user is logged into their account.
Exit Condition	5. The application presents the home page.
Alternate Flow	4a. The application shows an error message and prompts the user to provide the credentials again.

Table 18. Use Case Description for Login

Use Case ID	UC02
Use Case Name	Add Pharmacist
Actors	Manager, Pharmacist (User)
Description	Enables the manager to create an account for a new pharmacist.
Entry Condition	The manager must be logged into their account.
Flow of Events	1. The manager clicks on the “New Employee'' button. 2. The application presents a page with a registration form. 3. The manager provides the necessary information into their respective fields and clicks on the “Save'' button. 4. The application checks if all the information is provided correctly. 5. The application creates the account and sends a notification to the new pharmacist to change their password.

Exit Condition	6. The application redirects the manager to their home page.
Alternate Flow	5a. The application shows an error message and prompts the manager to enter the information again.

Table 19. Use Case Description for Add Pharmacist

Use Case ID	UC03
Use Case Name	Change Password
Actors	Manager, Pharmacist (User)
Description	Enables staff to change the password for their account.
Entry Condition	The user must be logged into their account.
Flow of Events	1. The user opens the “Accounts'' page and opens the “Change Password” page. 2. The application presents the “Change Password” page. 3. The user provides the old and new passwords in their respective fields and clicks on the “Change Password” button. 4. The application checks the old password and saves the changes.
Exit Condition	5. The application logs the user out of their account and displays the login page, includes ***Log in***
Alternate Flow	4a. The application shows an error message and prompts the user to enter the old password correctly again.

Table 20. Use Case Description for Change Password

Use Case ID	UC04
Use Case Name	Add Customer
Actors	Pharmacist (User)
Description	Enables staff to register customers for notification service.
Entry Condition	The user must be logged into their account.
Flow of Events	1. The user opens the “Services” page and clicks on the “New Customer” button. 2. The application presents a page with a customer registration form. 3. The user provides the necessary information into the

	respective fields and clicks on the “Save” button. 4. The application checks if all the information is provided correctly. 5. The application saves the information and sends a SMS notification to the customer.
Exit Condition	6. The application redirects the user to the “Services” page.
Alternate Flow	5a. The application shows an error message and prompts the user to enter the information again.

Table 21. Use Case Description for Add Customer

Use Case ID	UC05
Use Case Name	Send Notification to Customer
Actors	Customer
Description	Enables the system to send a SMS notification about the estimated next purchase date of a medicine to a customer.
Entry Condition	The customer must be registered for a notification service.
Flow of Events	1. The application checks if the date for a customer's next purchase is within two days. 2. The application sends SMS to the customer’s phone. 3. The application updates the next purchase date.
Exit Condition	4. The application receives a delivery report from the ISP.
Alternate Flow	1a. The customer’s next purchase date is not within two days; the application halts the process. 4a. The application doesn’t receive a delivery report, waits for 15 minutes and sends the SMS again.

Table 22. Use Case Description for Send Notification to Customer

Use Case ID	UC06
Use Case Name	Add Medicine

Actors	Manager
Description	Enables the manager to add newly obtained items into the store.
Entry Condition	The manager must be logged into their account.
Flow of Events	1. The manager opens the “Medicine” page and clicks on the “Add Medicine” button. 2. The application presents a registration form with fields to enter all the information needed. 3. The manager provides the necessary information in their respective fields and clicks on the “Add” button. 4. The application checks if all the information is provided correctly. 5. The application saves the provided information on the database and displays a confirmation notification.
Exit Condition	6. The application redirects the manager to the “Register Item” page.
Alternate Flow	5a. The application shows an error message and prompts the manager to enter the information again.

Table 23. Use Case Description for Add Medicine

Use Case ID	UC07
Use Case Name	Search for a Medicine
Actors	Manager, Pharmacist (User)
Description	Enables staff to search for and see information about items in the store.
Entry Condition	The user must be logged into their account.
Flow of Events	1. The application presents the home page with a search bar. 2. The user enters a query in the search bar and clicks on the “Search” button. 3. The application fetches the query and looks up for it in the database.
Exit Condition	4. The application displays all the relevant information to the user.
Alternate Flow	4a. The application displays a message informing the user nothing matched with their query.

Table 24. USe Case Description for Search for a Medicine

Use Case ID	UC08
Use Case Name	Update Medicine
Actors	Manager
Description	Enables the manager to change information about items in the store.
Entry Condition	● The manager must be logged into their account. ● The item should be registered beforehand.
Flow of Events	1. The manager searches for an item and selects the item they want. includes ***Search for an Item*** 2. The application presents a page with all the information about the item. 3. The manager clicks on the “Edit” button, edits the information and clicks on the “Save Changes” button. 4. The application checks if all the information is provided correctly. 5. The application saves the information and displays a confirmation notification to the manager.
Exit Condition	6. The application redirects the manager to the information page.
Alternate Flow	5a. The application shows an error message and prompts the manager to enter the information again.

Table 25. Use Case Description for Update Medicine

Use Case ID	UC09
Use Case Name	Sell Medicine
Actors	Pharmacist, Customer
Description	Enables the pharmacist to dispense items to customers.
Entry Condition	● The pharmacist must be logged into their account. ● The item should be registered beforehand.
Flow of Events	1. The pharmacist searches for an item. include ***Search for an Item*** 2. The pharmacist selects the item, adjusts the quantity and clicks on the “Add to Cart” button. 3. The application adds all the selected materials to the cart. 4. The pharmacist clicks on the “Checkout” button. 5. The application presents the “Checkout” page. 6. The customer pays for the item. include ***Make Payment*** 7. The application presents a page with a receipt and prompts the pharmacist to print the receipt and attachment. 8. The pharmacist clicks on the “Print” button. 9. The application prints out the receipt and attachment.
Exit Condition	10. The application redirects the pharmacist to their home page.
Alternate Flow	8a. The pharmacist clicks on the “Save as PDF” button. 9a. The application saves the document to the computer.

Table 26. Use Case Description for Sell Medicine

Use Case ID	UC10
Use Case Name	Return Medicine
Actors	Pharmacist, Customer
Description	Enables the customer to return an item back to the pharmacy.
Entry Condition	• The pharmacist must be logged into their account. • The item should be purchased from the pharmacy within the last 24 hours.
Flow of Events	1.
Exit Condition	The application redirects the pharmacist to their home page.
Alternate Flow	

Table 27. Use Case Description for Return Medicine

Use Case ID	UC11
Use Case Name	Remove Medicine
Actors	Manager, Pharmacist (User)
Description	Enables the user to remove items from the dispensary.
Entry Condition	• The user must be logged into their account. • The item should be registered beforehand.
Flow of Events	1. The user searches for an item and selects the item they want. extend ***Search for an Item*** 2. The user clicks on the “Delete” button. 3. The application presents a notification asking if the user is sure to carry on with the action. 4. The user clicks on the “Yes” option. 5. The application makes a copy of the item and removes it. 6. The application saves the copy of the item in RUC. 7. The system displays a notification about the success of the operation.
Exit Condition	8. The application redirects the user to their home page.
Alternate Flow	1.a The user clicks on an expiry notification. **extend *Send Expiration Notification***

Table 28. Use Case Description for Remove Medicine

Use Case ID	UC12
Use Case Name	Send Expiration Notification
Actors	Pharmacist, Manager (User)
Description	Enables the system to notify about expired or expiring items in the store.
Entry Condition	● The user must be logged into their account. ● The item should have a month or less left for expiration.
Flow of Events	1. The application checks the expiration dates of each item in the dispensary. 2. If there are items with 30 days or less of expiration date, the application selects them. 3. The application sends a notification to the home page of the user. 4. The user clicks on the notification.
Exit Condition	5. The application displays all relevant information to the user.
Alternate Flow	2.a There are no items with 30 days or less of expiration date, the system halts the operation. 4.a The user doesn't click on the notification, the application waits an hour and sends the notification again.

Table 29. Use Case Description Send Expiration Notification

Use Case ID	UC13
Use Case Name	Generate Bin Card
Actors	Manager
Description	Enables the manager to create a document consisting of the sales history of a certain item.
Entry Condition	● The manager must be logged into their account. ● There should be a transaction involving the selected item prior to the creation of the document.
Flow of Events	1. The manager opens the "Reports" page and clicks on the "Bin Card" button. 2. The application presents a page with a search bar. 3. The manager searches for and selects the item. includes ***Search for an Item***

	4. The application presents a page with the bin card in it. 5. The manager clicks on the “Print” button. 6. The application prints the bin card.
Exit Condition	7. The application redirects the manager to the Reports page.
Alternate Flow	5a. The manager clicks on the “Save as PDF” button. 6a. The application saves the document on the computer.

Table 30. Use Case Description for Generate Bin Card

Use Case ID	UC14
Use Case Name	Generate Stock Card
Actors	Manager
Description	Enables the manager to create a document consisting of every item in the store.
Entry Condition	● The manager must be logged into their account.
Flow of Events	1. The manager opens the “Reports” page and clicks on the “Stock Card” button. 2. The application presents a page with a date field. 3. The manager enters the date and clicks on the “Next” button. 4. The application presents a page with the stock card in it. 5. The manager clicks on the “Print” button. 6. The application prints the bin card.
Exit Condition	7. The application redirects the manager to the “Reports” page.
Alternate Flow	5a. The manager clicks on the “Save as PDF” button. 6a. The application saves the document on the computer.

Table 31. Use Case Description for Generate Stock Card

Use Case ID	UC15
Use Case Name	Generate Profit/Loss Report
Actors	Manager
Description	Enables the manager to create a document consisting of the gross and net calculations of profit or loss in a specified time frame.
Entry Condition	● The manager must be logged into their account.
Flow of Events	1. The manager opens the “Reports” page and clicks on the

	"Finances" button. 2. The application presents a page with a "Start Date" and "End Date" fields. 3. The manager enters the date and clicks on the "Calculate" button. 4. The application presents the gross and net profit/loss for that time frame. 5. The manager clicks on the "Print" button. 6. The application prints the document.
Exit Condition	7. The application redirects the manager to the "Reports" page.
Alternate Flow	4a. The application shows an error message and prompts the manager to enter dates correctly again. 5a. The manager clicks on the "Save as PDF" button. 6a. The application saves the document into the computer.

Table 32. Use Case Description Generate Profit/Loss Report

Use Case ID	UC16
Use Case Name	Generate Returned and Unused Commodities (RUC) Report
Actors	Manager
Description	Enables the manager to create a document consisting of all the returned and unused items in the store.
Entry Condition	● The manager must be logged into their account.
Flow of Events	1. The manager opens the "Reports" page and clicks on the "RUC" button. 2. The application presents a page with "Start Date" and "End Date" fields. 3. The manager enters the dates and clicks on the "Next" button. 4. The application presents a page with all the returned and unused commodities for that time frame. 5. The manager clicks on the "Print" button. 6. The application prints the document.
Exit Condition	7. The application redirects the manager to the Reports page.
Alternate Flow	4a. The application shows an error message and prompts the manager to enter dates correctly again. 5a. The manager clicks on the "Save as PDF" button. 5b. The application saves the document into the computer.

Table 33. Use Case Description for Generate RUC Report

Use Case ID	UC17
Use Case Name	Generate Product Recommendation Report
Actors	Manager
Description	Enables the manager to create a document that consists of suggestions of which products to procure in the future based on data from past transactions.
Entry Condition	● The manager must be logged into their account.
Flow of Events	1. The manager opens the “DSS” page and clicks on the “Product Recommendation” button. 2. The application presents a page with the fetched data. 3. The manager clicks on the “Print” button. 4. The application prints the document.
Exit Condition	5. The application redirects the manager to the “DSS” page.
Alternate Flow	3a. The manager clicks on the “Save as PDF” button. 4a. The application saves the document into the computer.

Table 34. Use Case Description for Generate Product Recommendation Report

Use Case ID	UC18
Use Case Name	Generate Sales Forecasting Report
Actors	Manager
Description	Enables the manager to create a document consisting of sales predictions for the near future based on past trends and sales.
Entry Condition	● The manager must be logged into their account.
Flow of Events	1. The manager opens the “DSS” page and clicks on the “Sales Forecasting” button. 2. The application presents a page with the fetched data. 3. The manager clicks on the “Print” button. 4. The application prints the document.
Exit Condition	6. The application redirects the manager to the DSS page.
Alternate Flow	3a. The manager clicks on the “Save as PDF” button. 4b. The application saves the document into the computer.

Table 35. Use Case Description for Generate Sales Forecasting Report

Use Case ID	UC19
Use Case Name	Backup
Actors	Manager
Description	Enables the application to create a copy of all of its data to a cloud storage to be used in case of failures and disasters.
Entry Condition	● The application should have a stable internet connection. ● The backup should take place outside work time.
Flow of Events	1. The application checks the availability of a stable internet connection. 2. The application starts the backup process.
Exit Condition	3. The application sends a confirmation notification to the manager's account.
Alternate Flow	2a. The application can't detect a stable internet connection. 3a. The application sends a notification informing the manager the backup failed.

Table 36. Use Case Description for Backup Data

2.4.5. Class Diagram

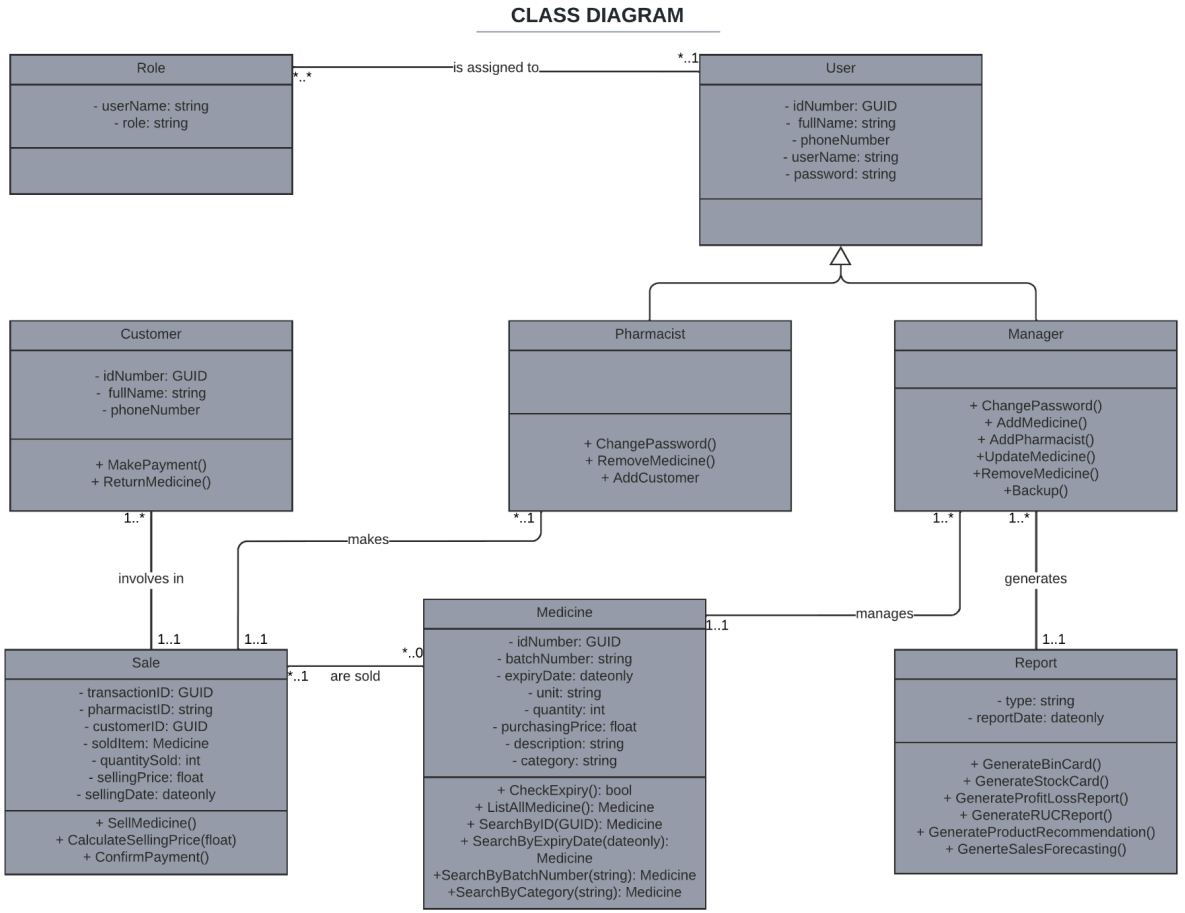

Figure 5. Class Diagram

2.4.6. Sequence Diagram

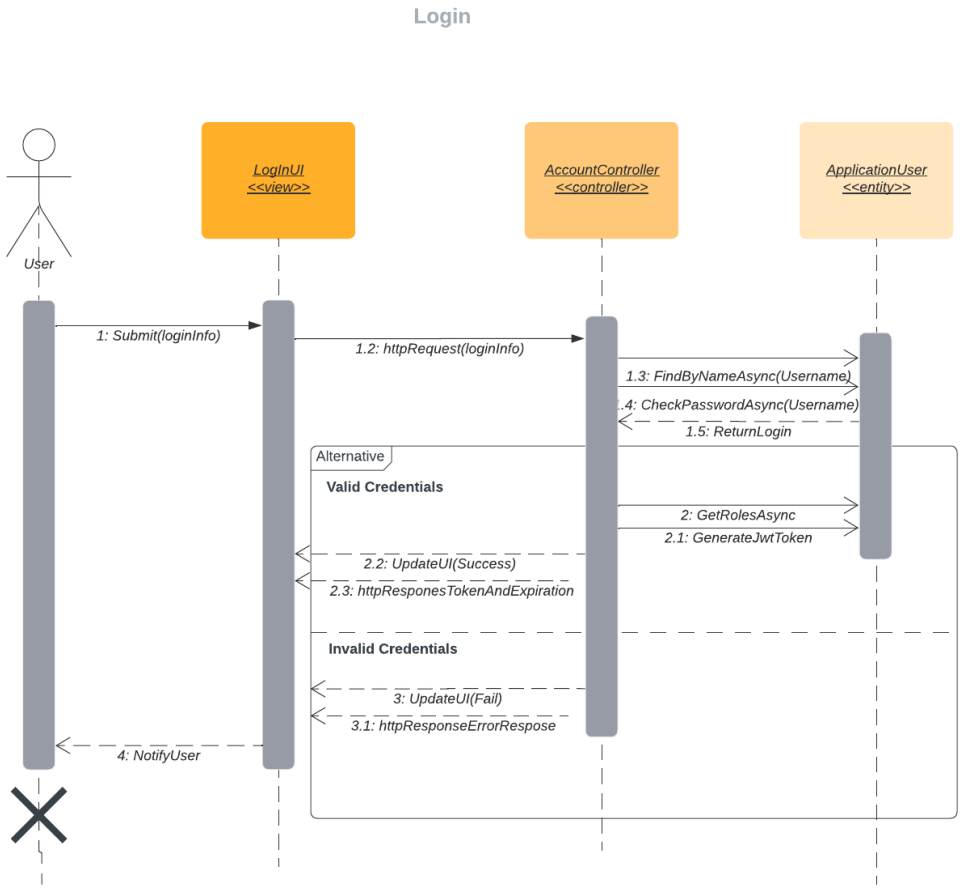

Figure 6. Sequence Diagram for Login

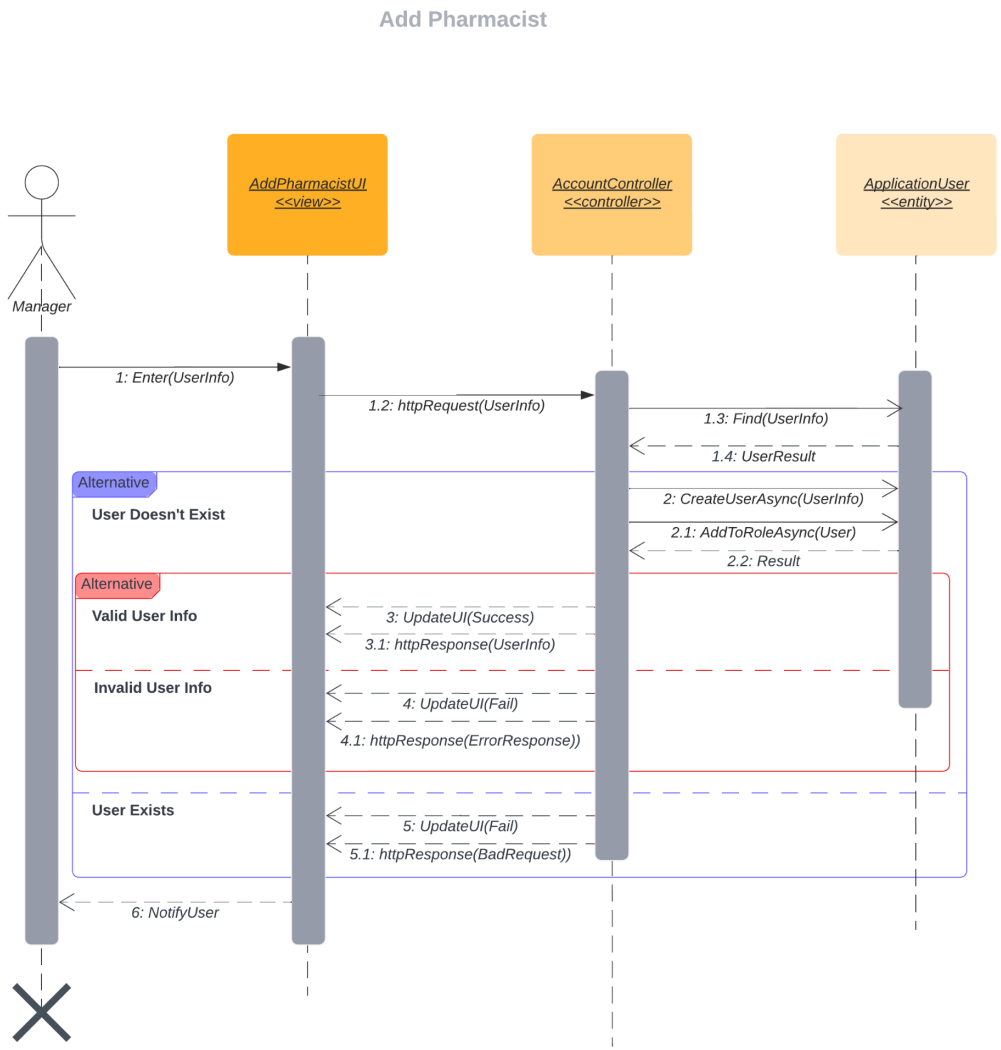

Figure 7. Sequence Diagram for Add Pharmacist

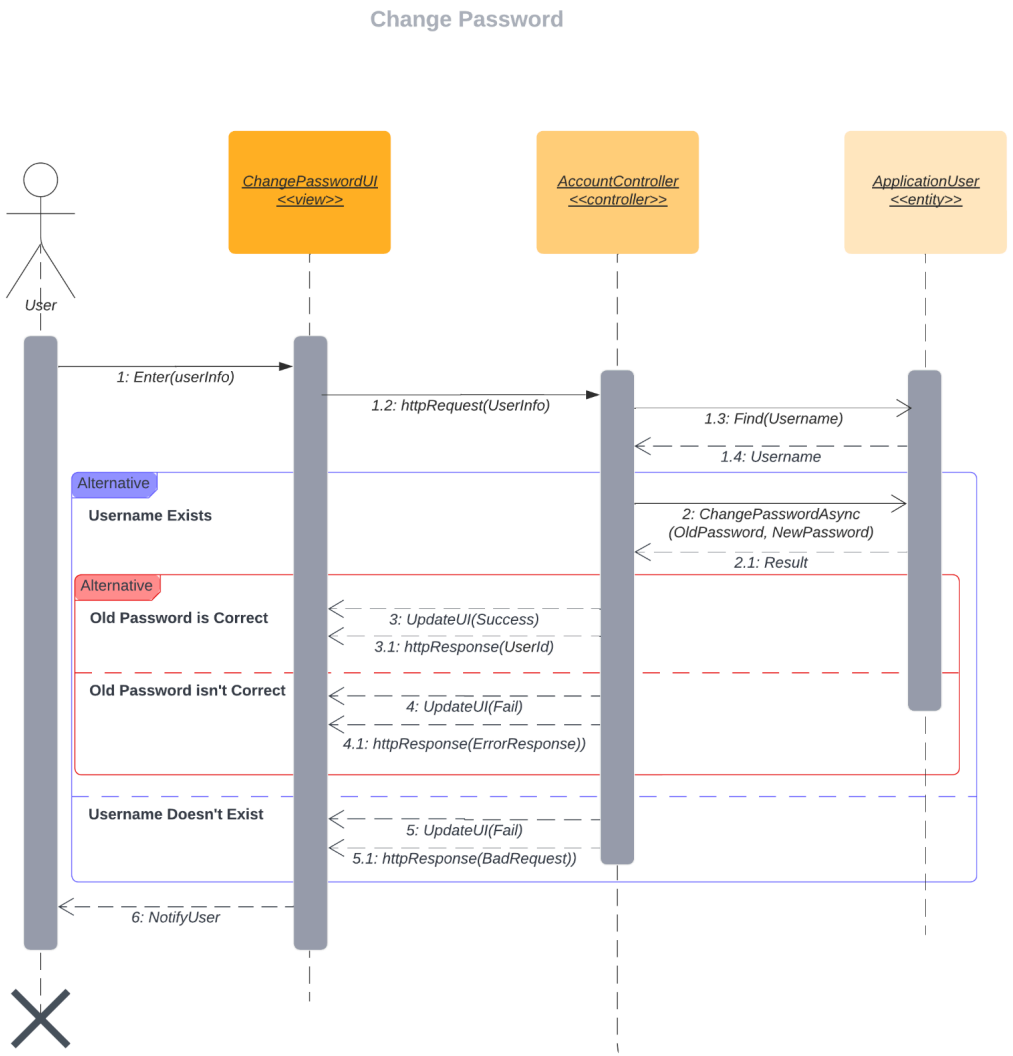

Figure 8. Sequence Diagram for Change Password

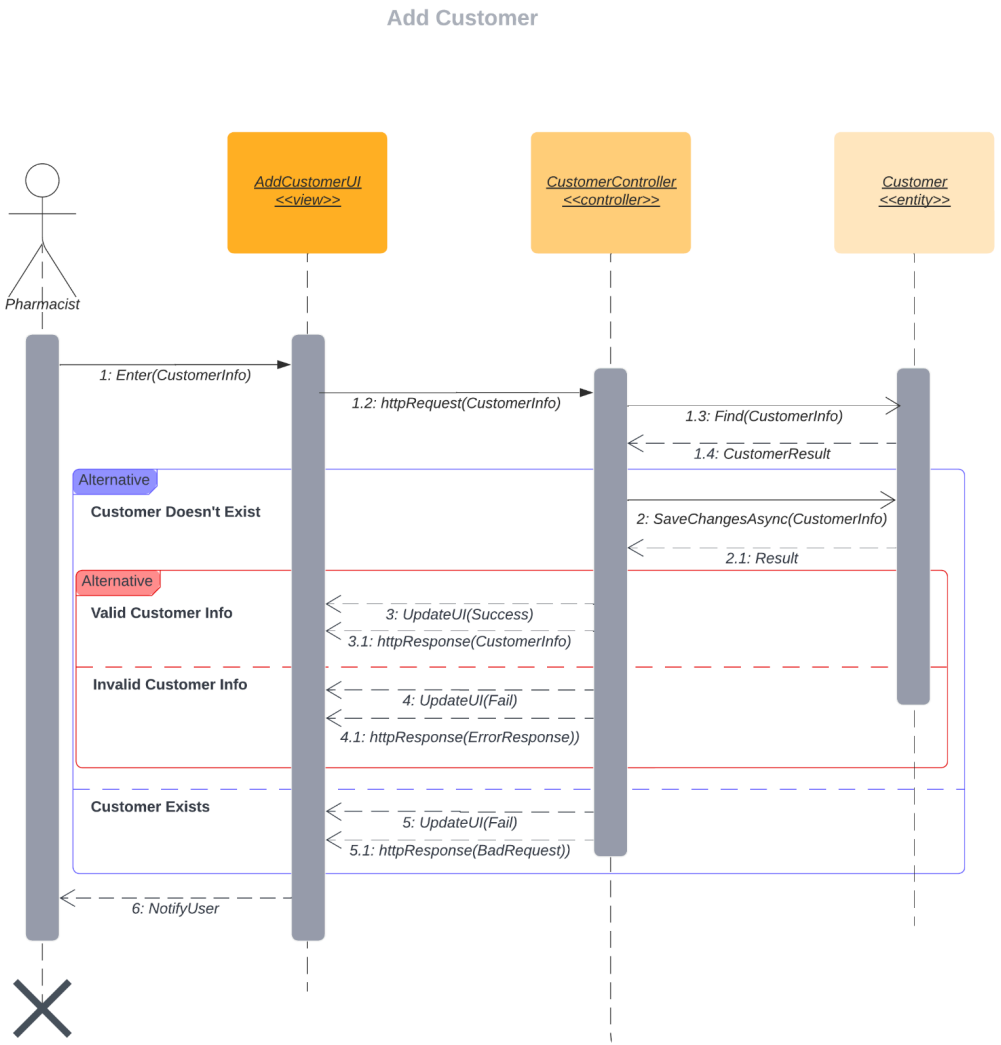

Figure 9. Sequence Diagram for Add Customer

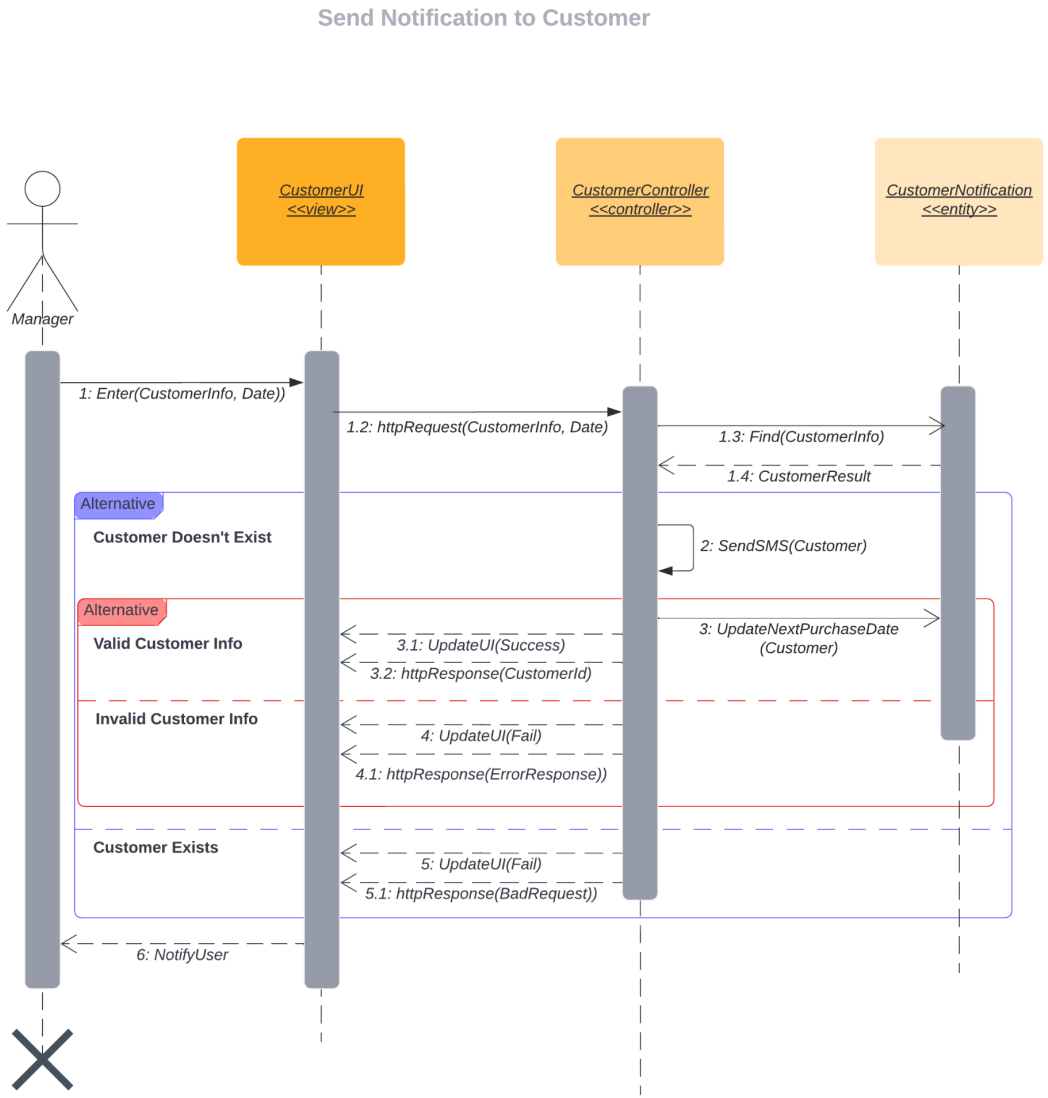

Figure 10. Sequence Diagram for Send Notifications to Customer

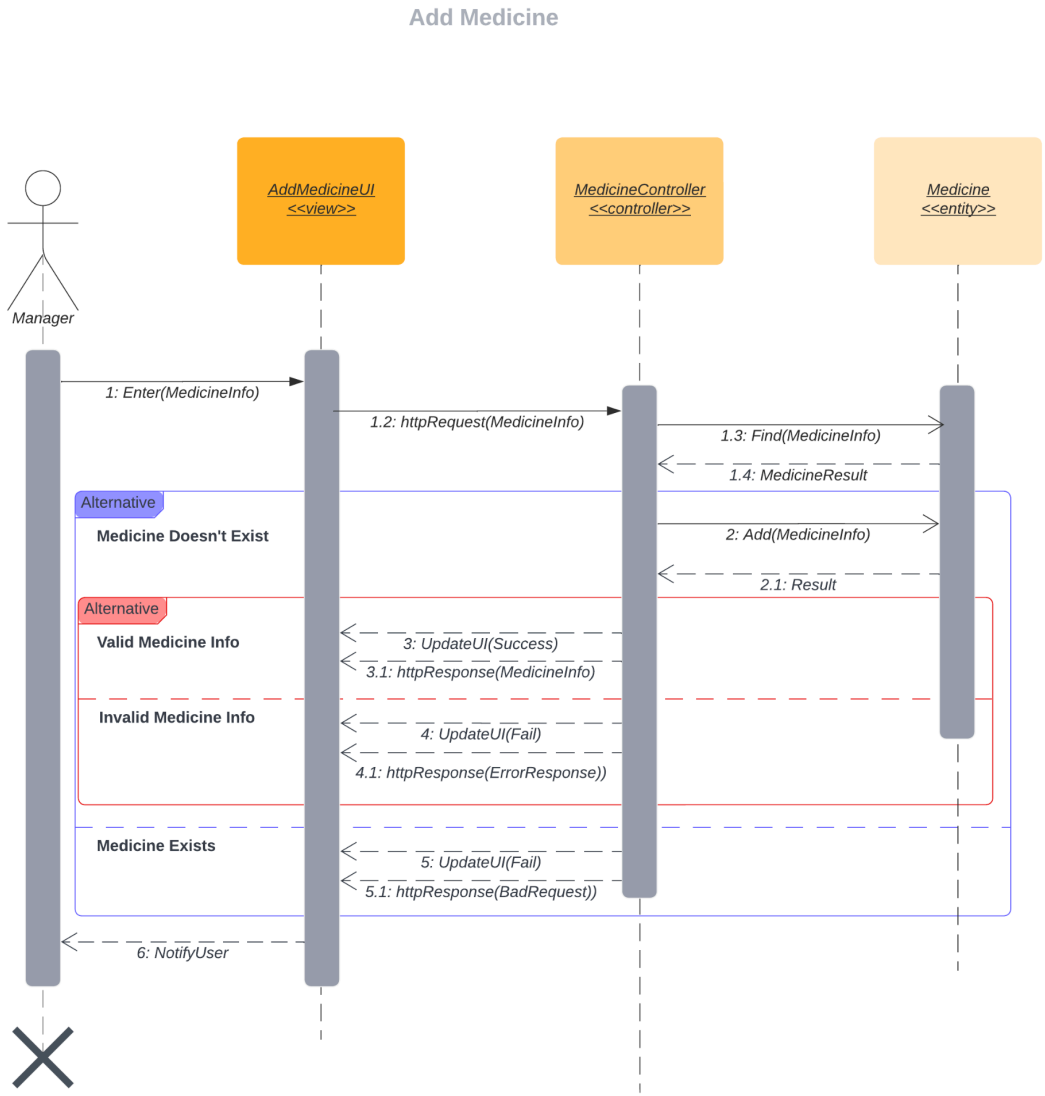

Figure 11. Sequence Diagram for Add Medicine

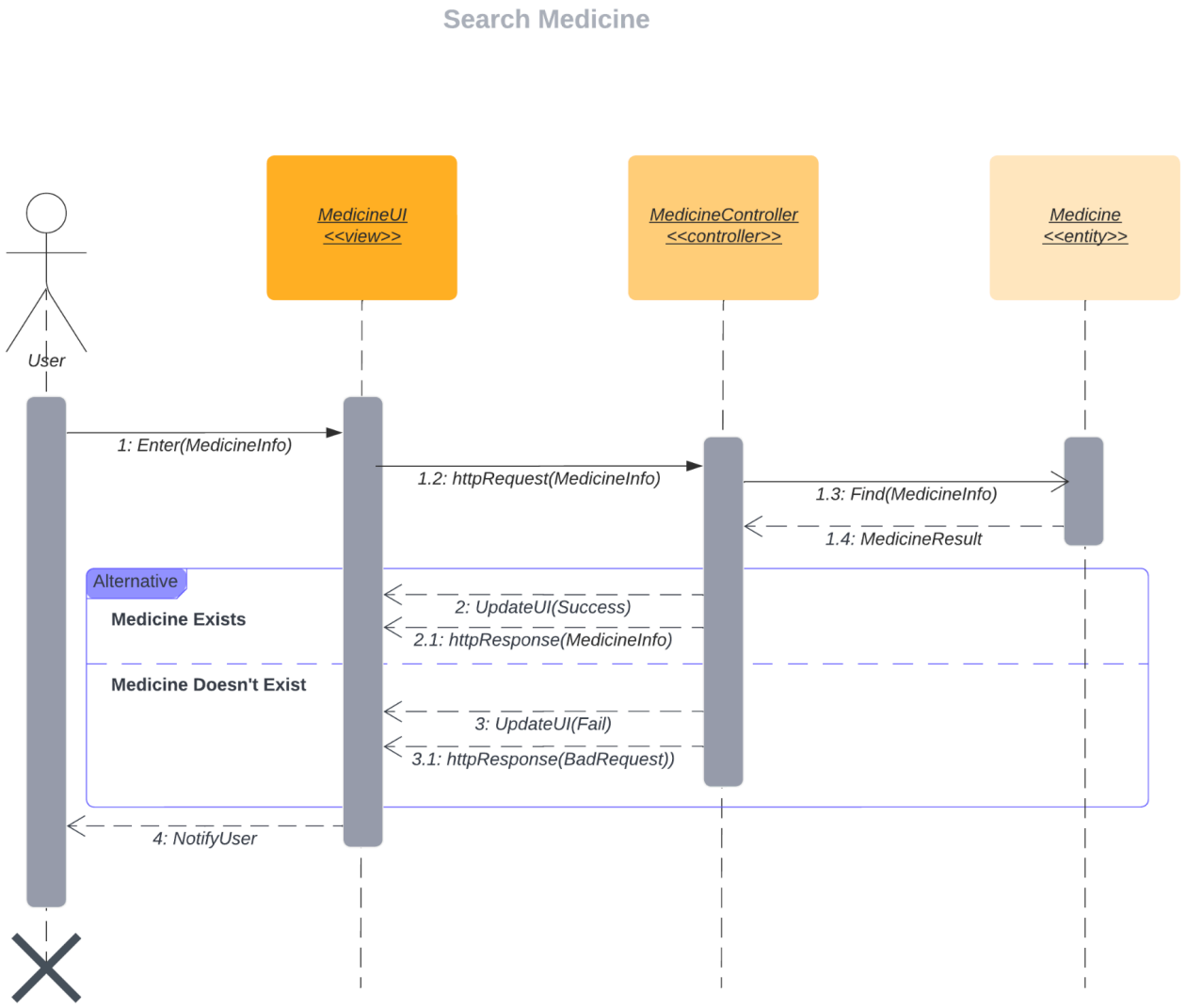

Figure 12. Sequence Diagram for Search for a Medicine

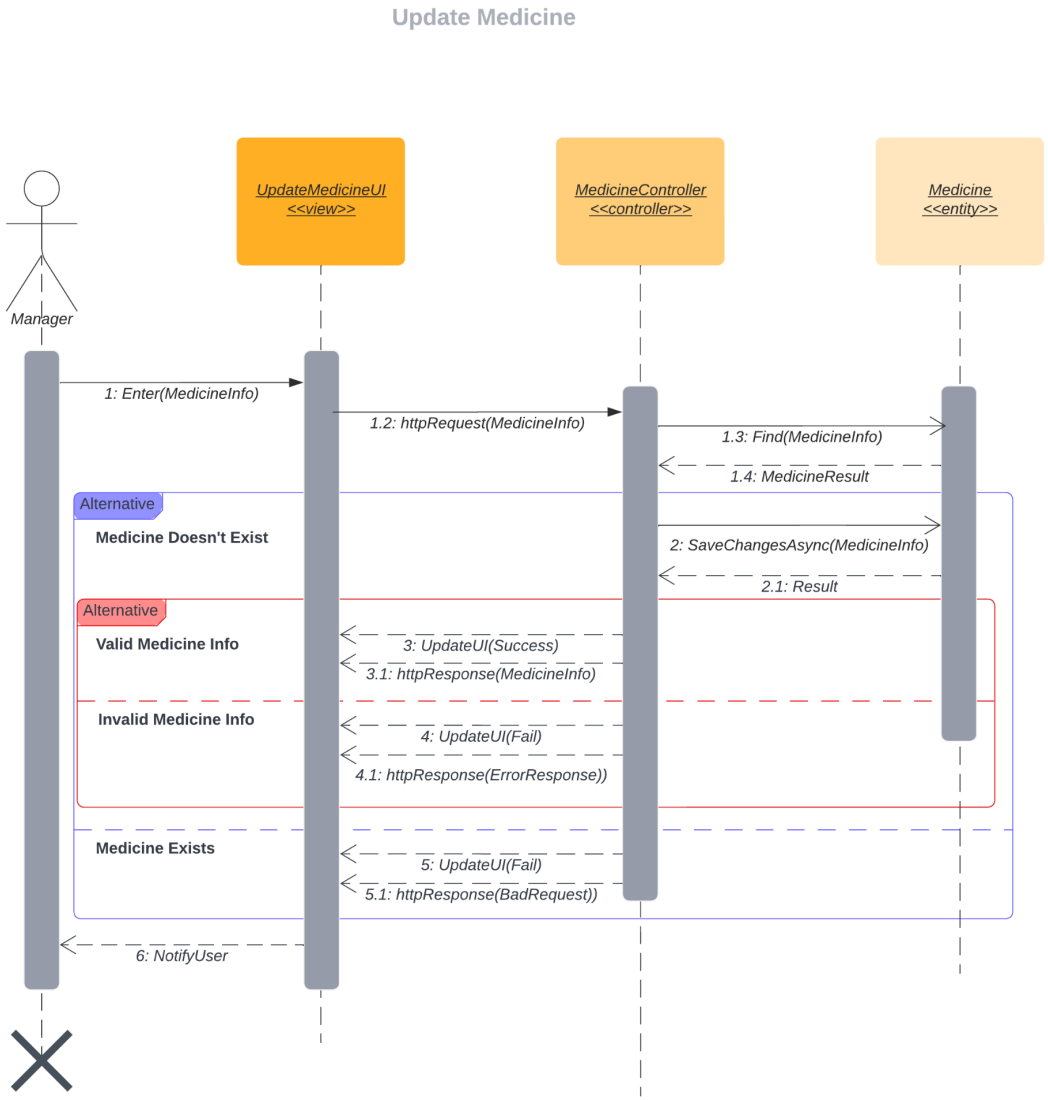

Figure 13. Sequence Diagram for Update Medicine

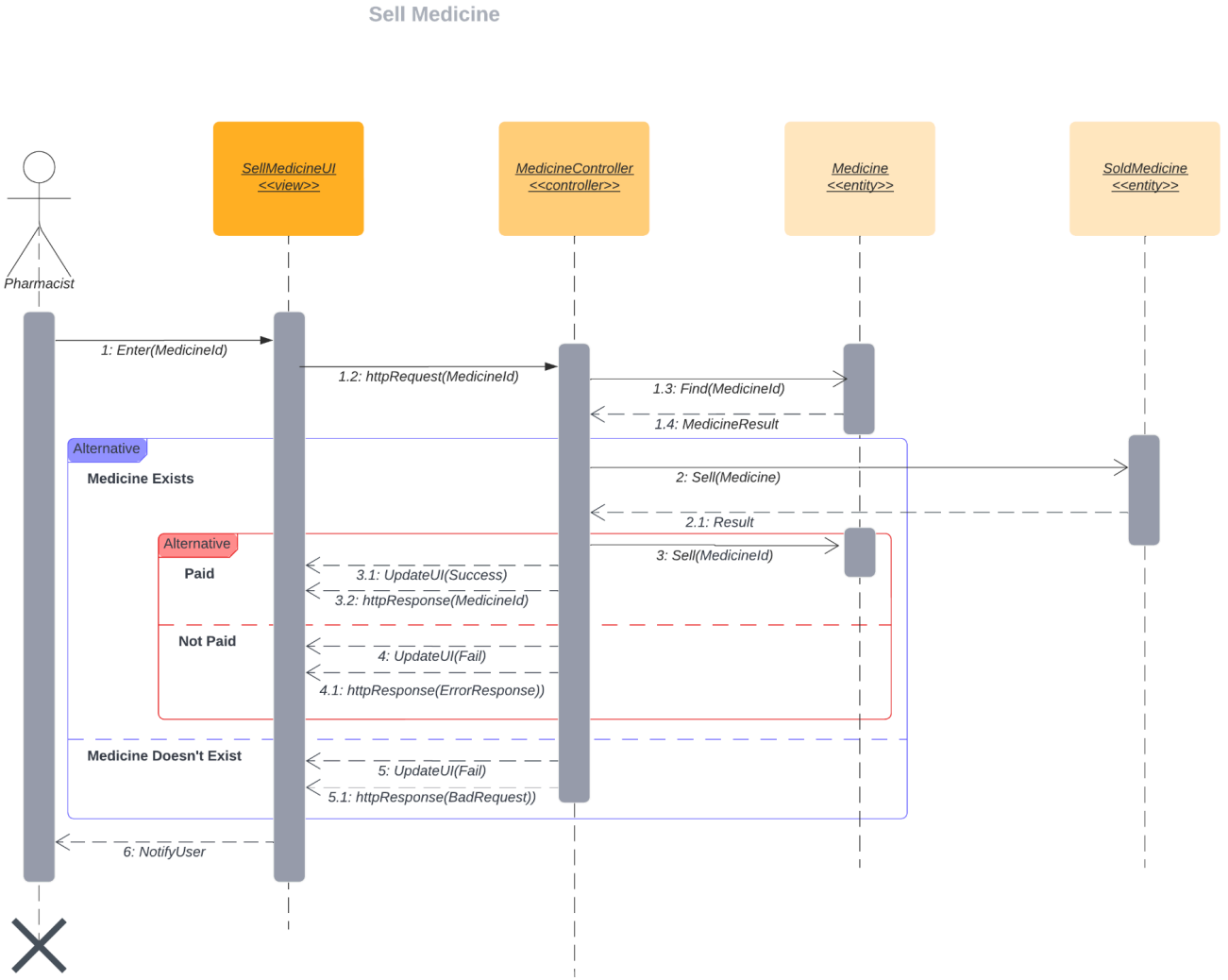

Figure 14. Sequence Diagram for Sell Medicine

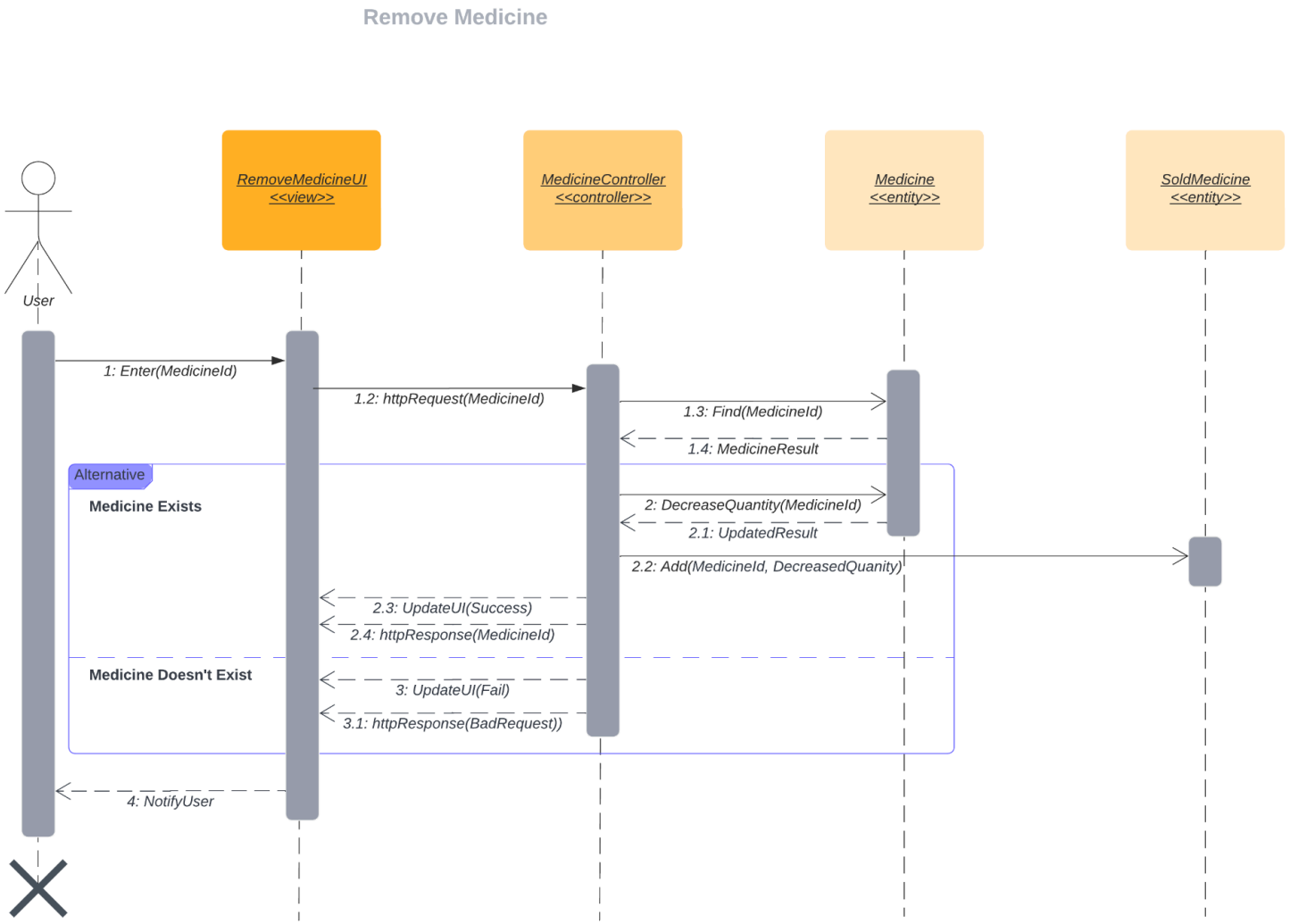

Figure 15. Sequence Diagram for Remove Medicine

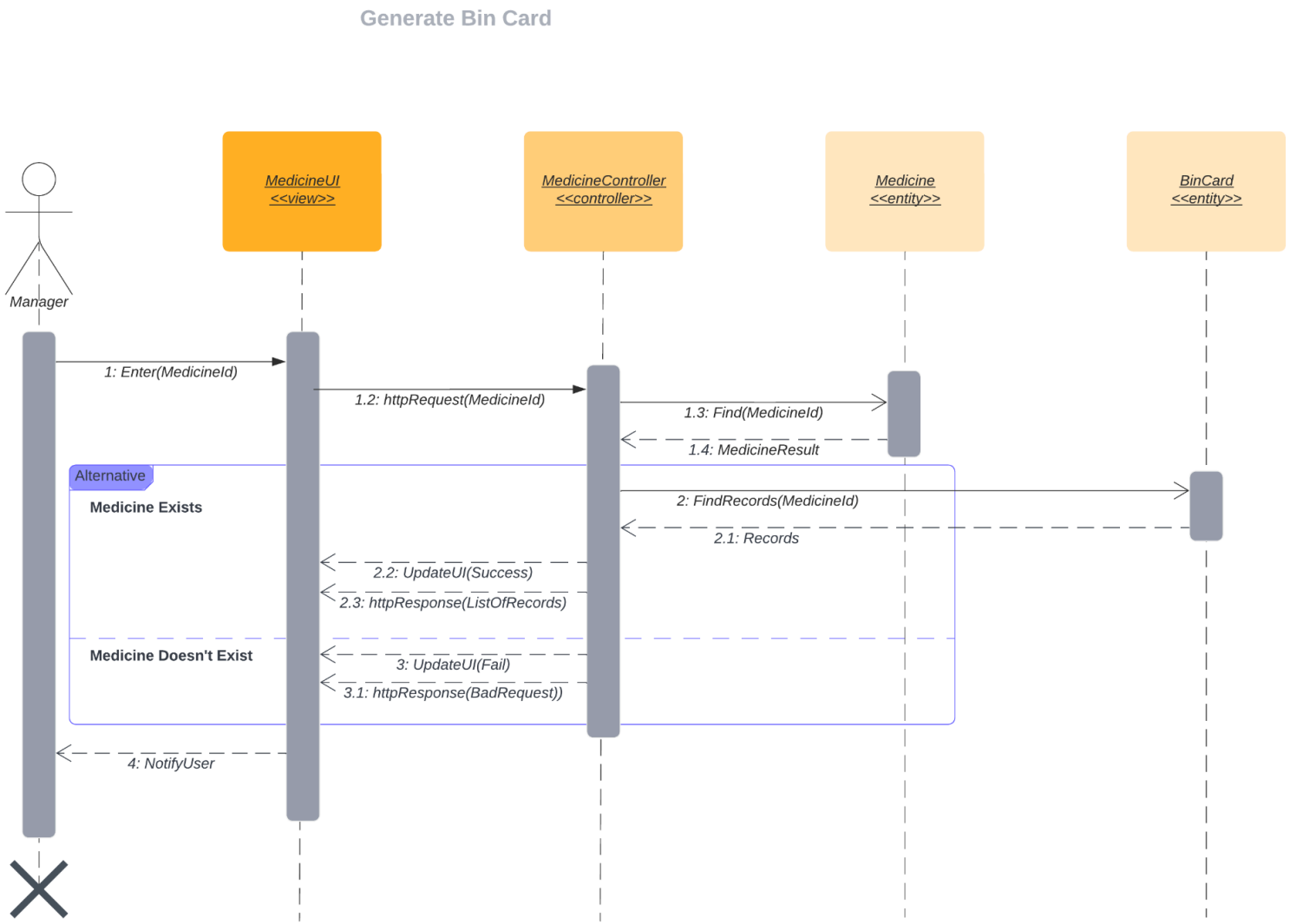

Figure 16. Sequence Diagram for Generate Bin Card

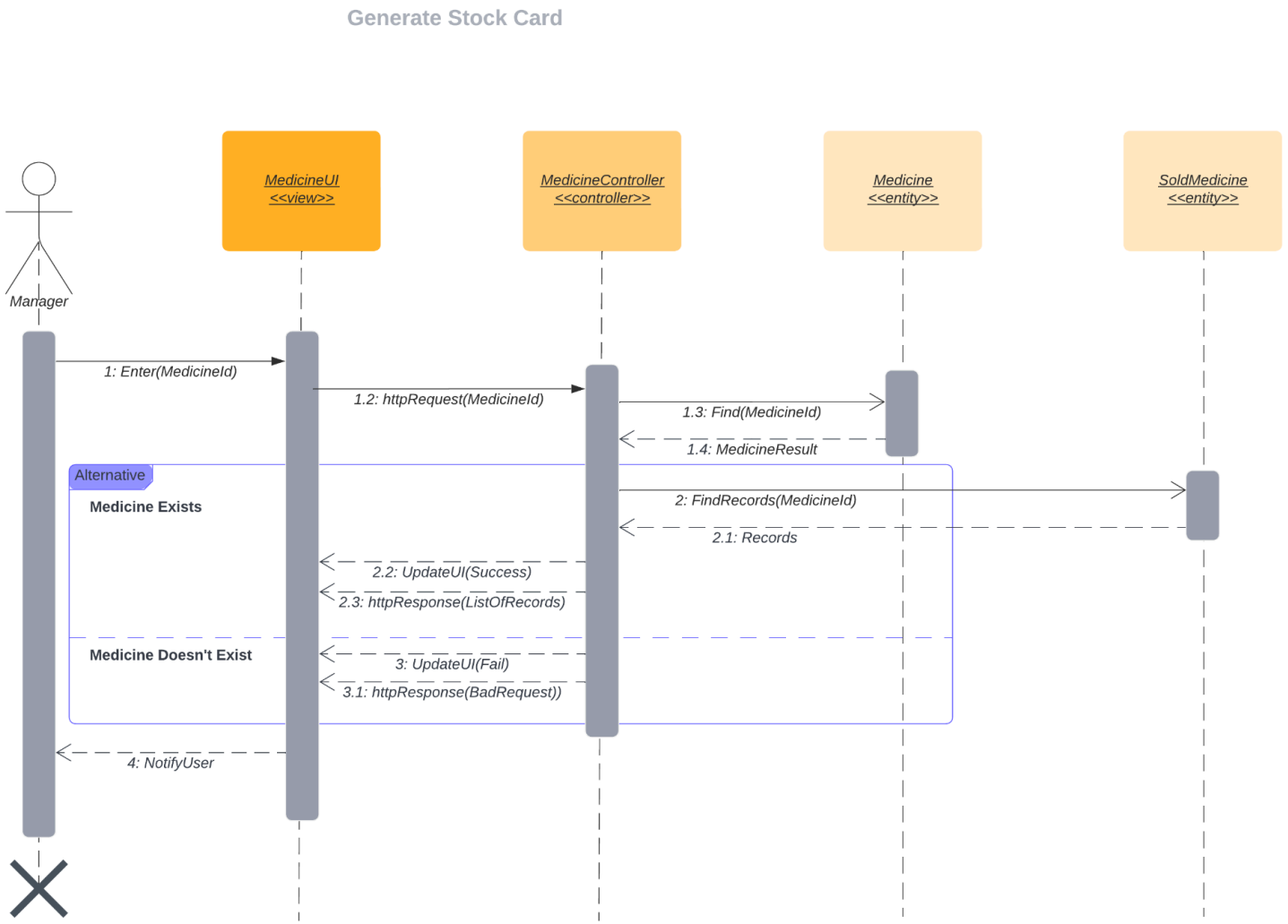

Figure 17. Sequence Diagram for Generate Stock Card

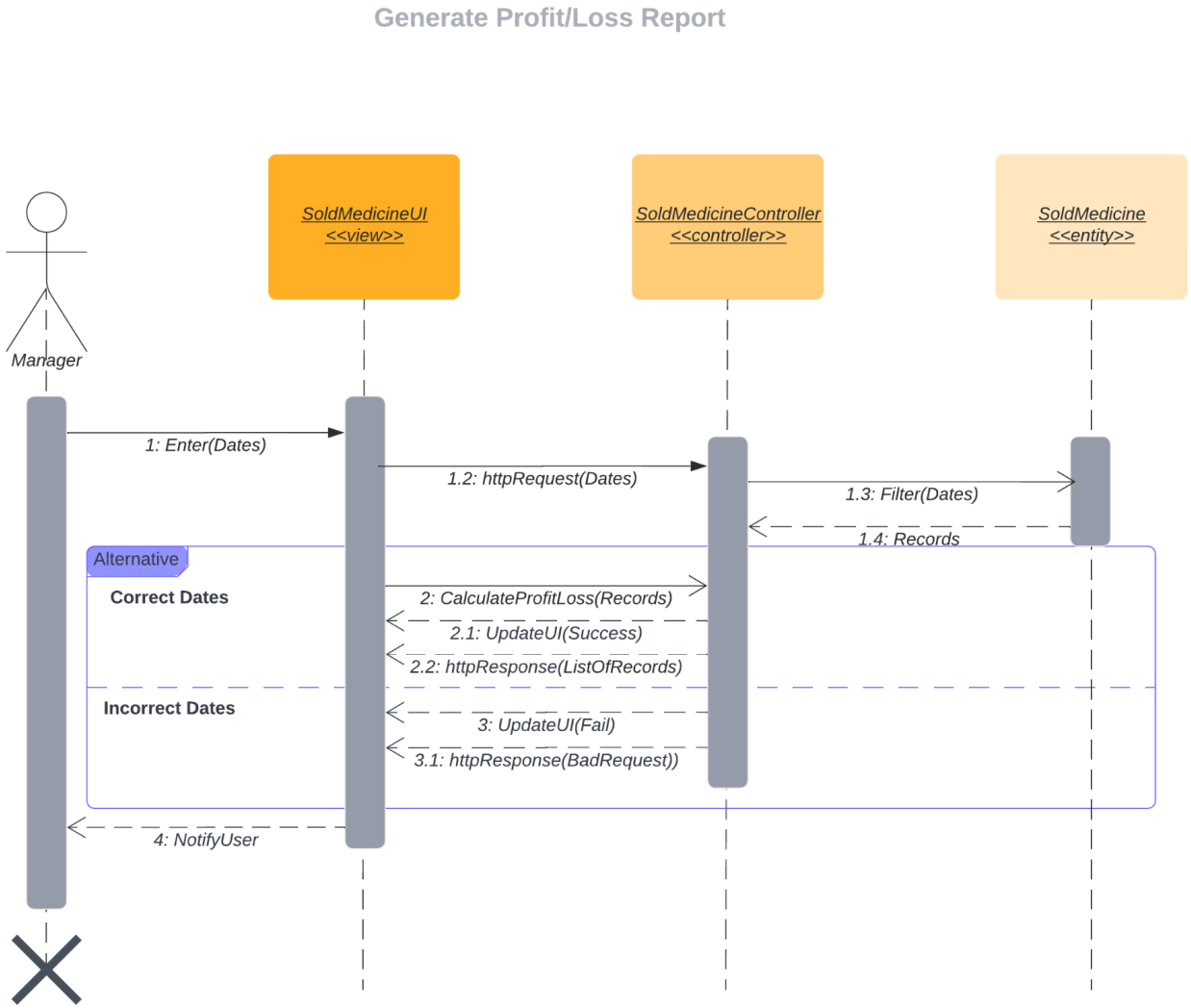

Figure 18. Sequence Diagram for Generate Profit/Loss Report

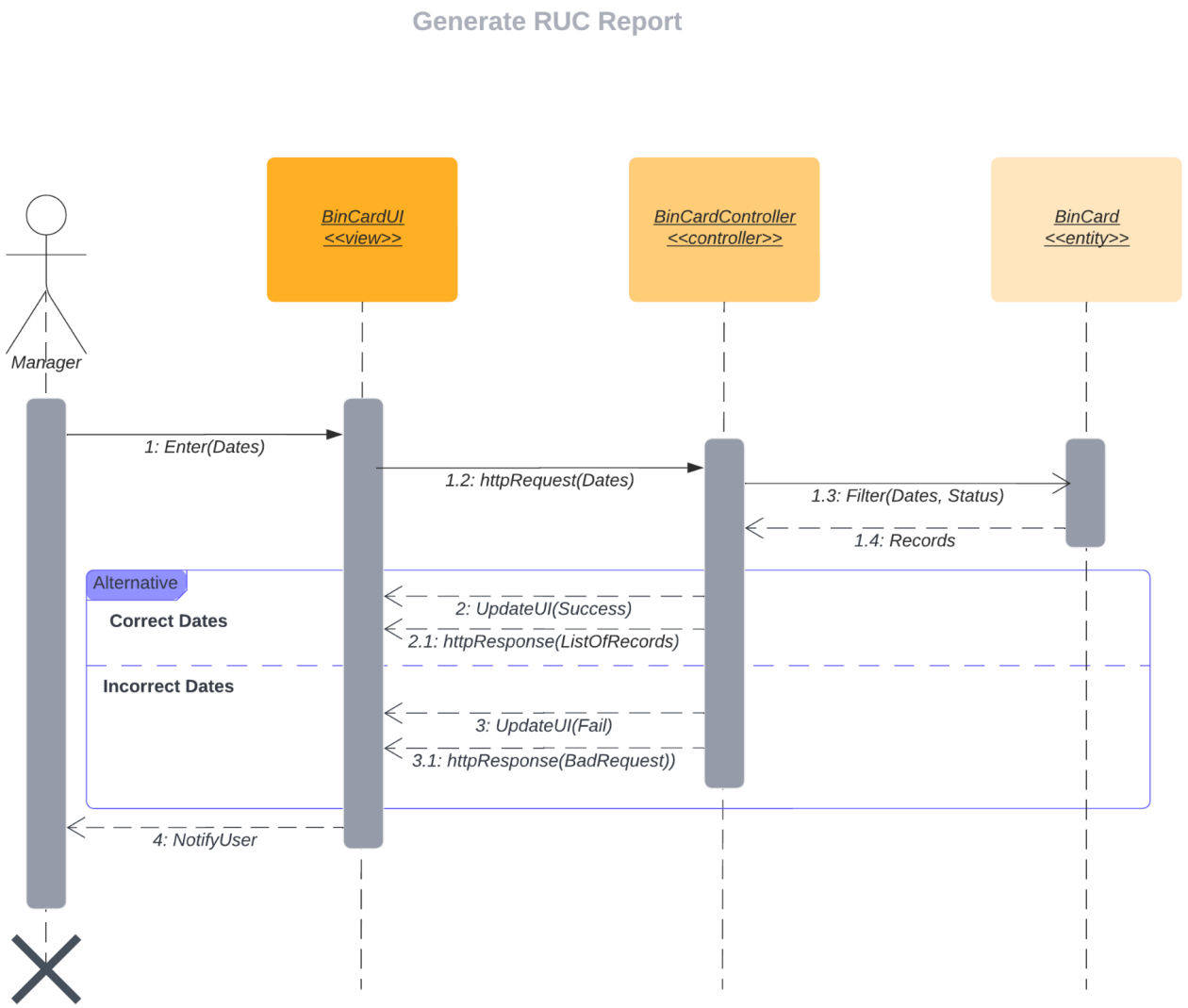

Figure 19. Sequence Diagram for Generate RUC Report

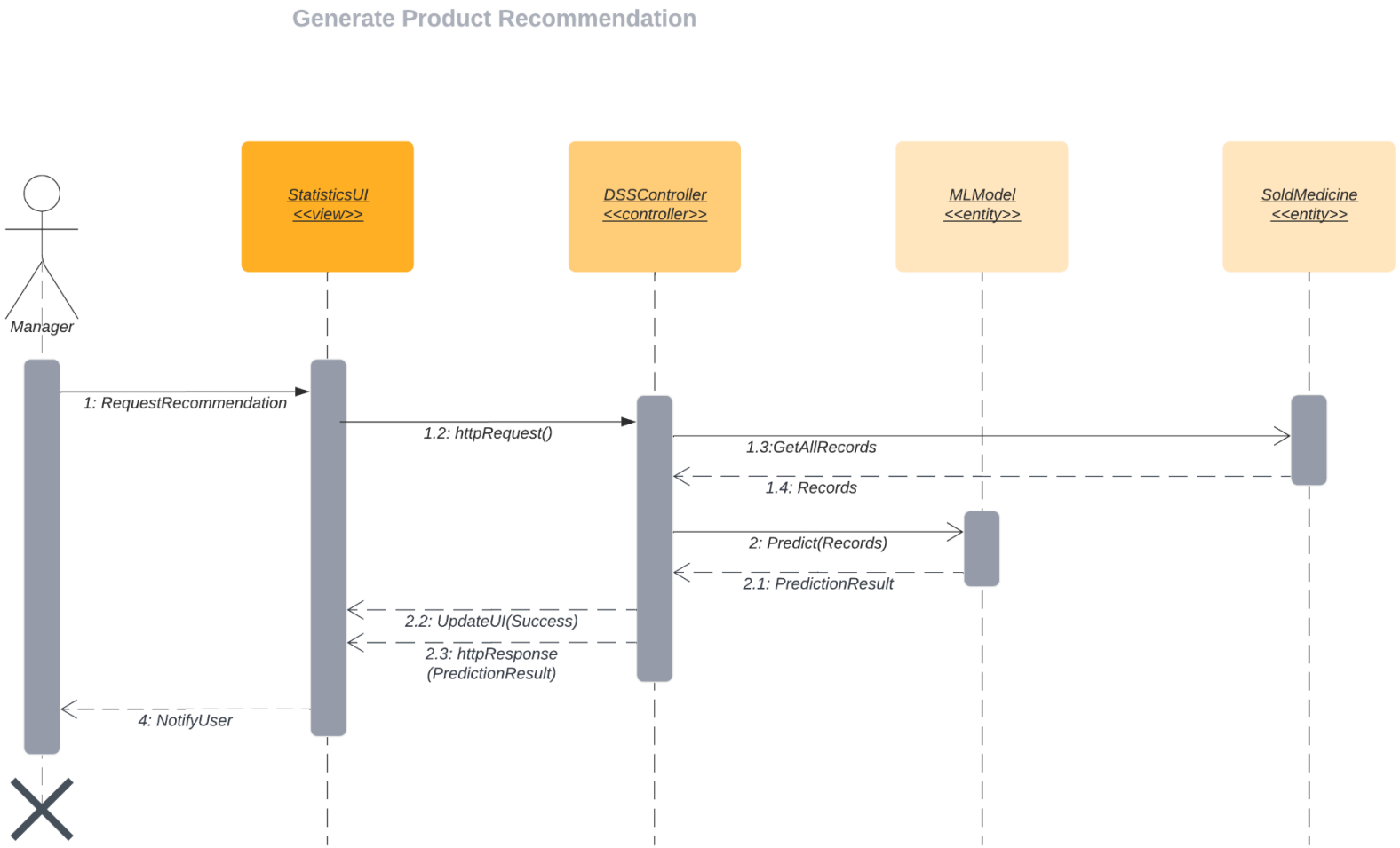

Figure 20. Sequence Diagram for Generate Product Recommendation

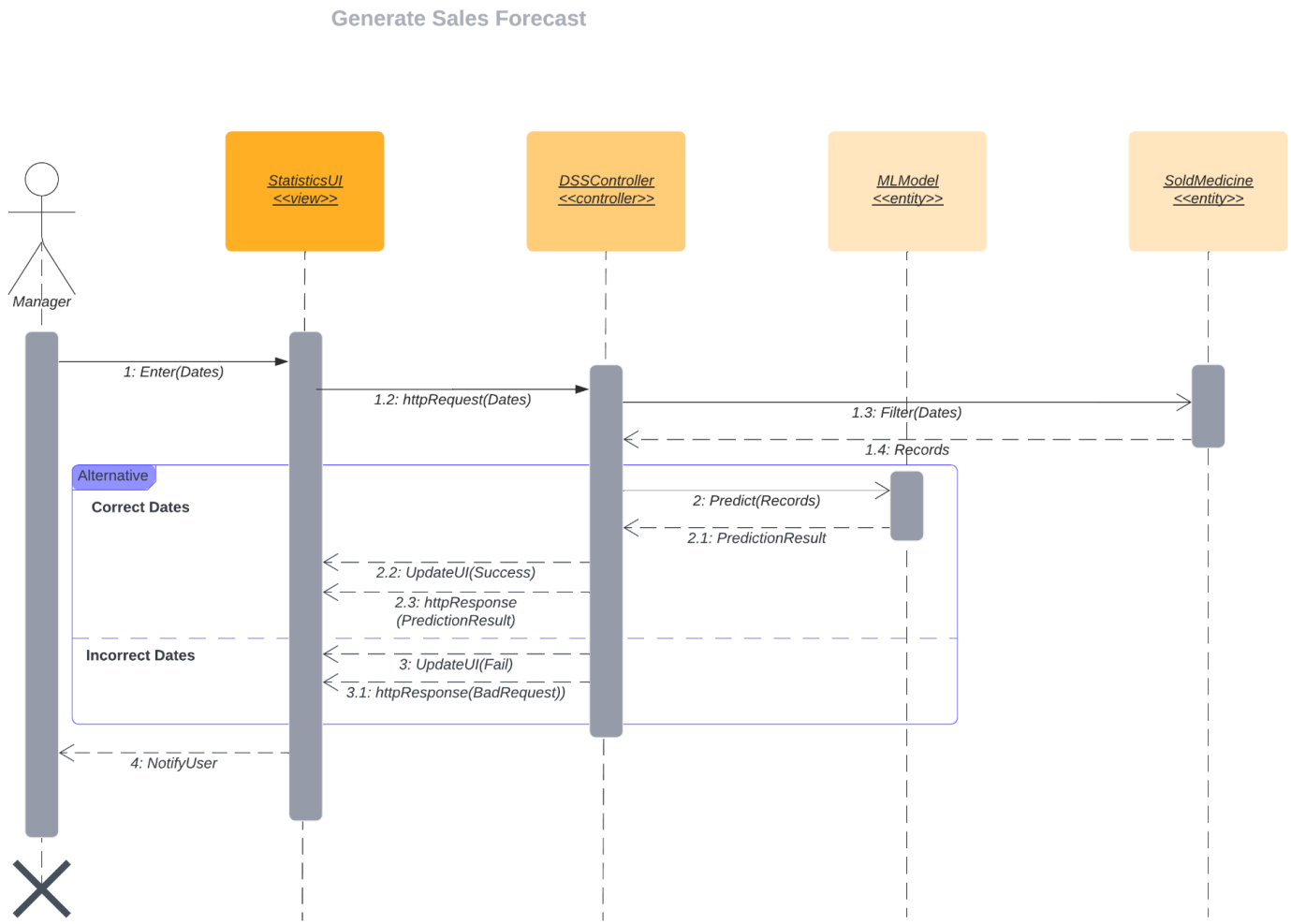

Figure 21. Sequence Diagram for Generate Sales Forecasting

2.4.7. Activity Diagram

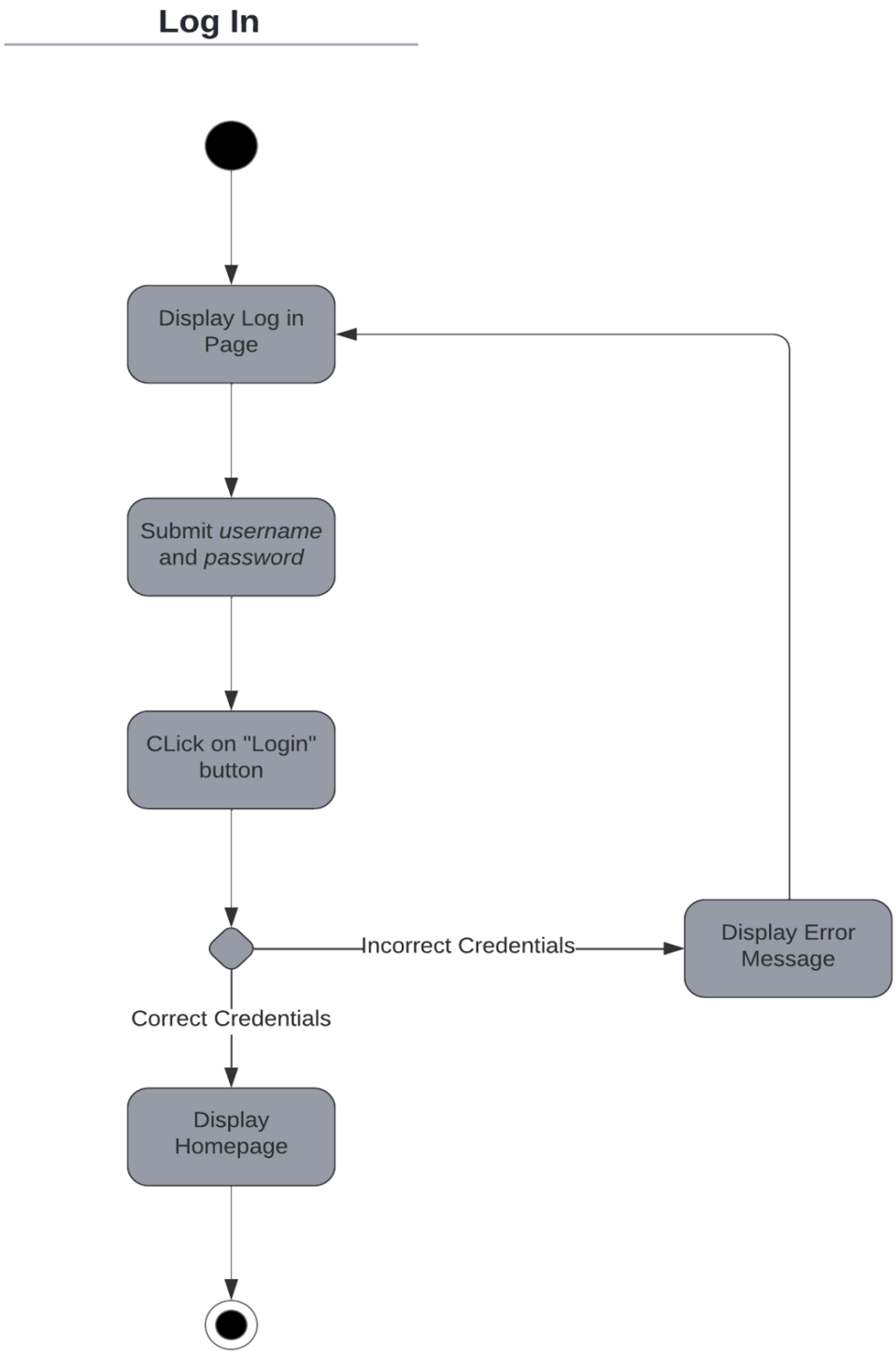

Figure 22. Activity Diagram for Login

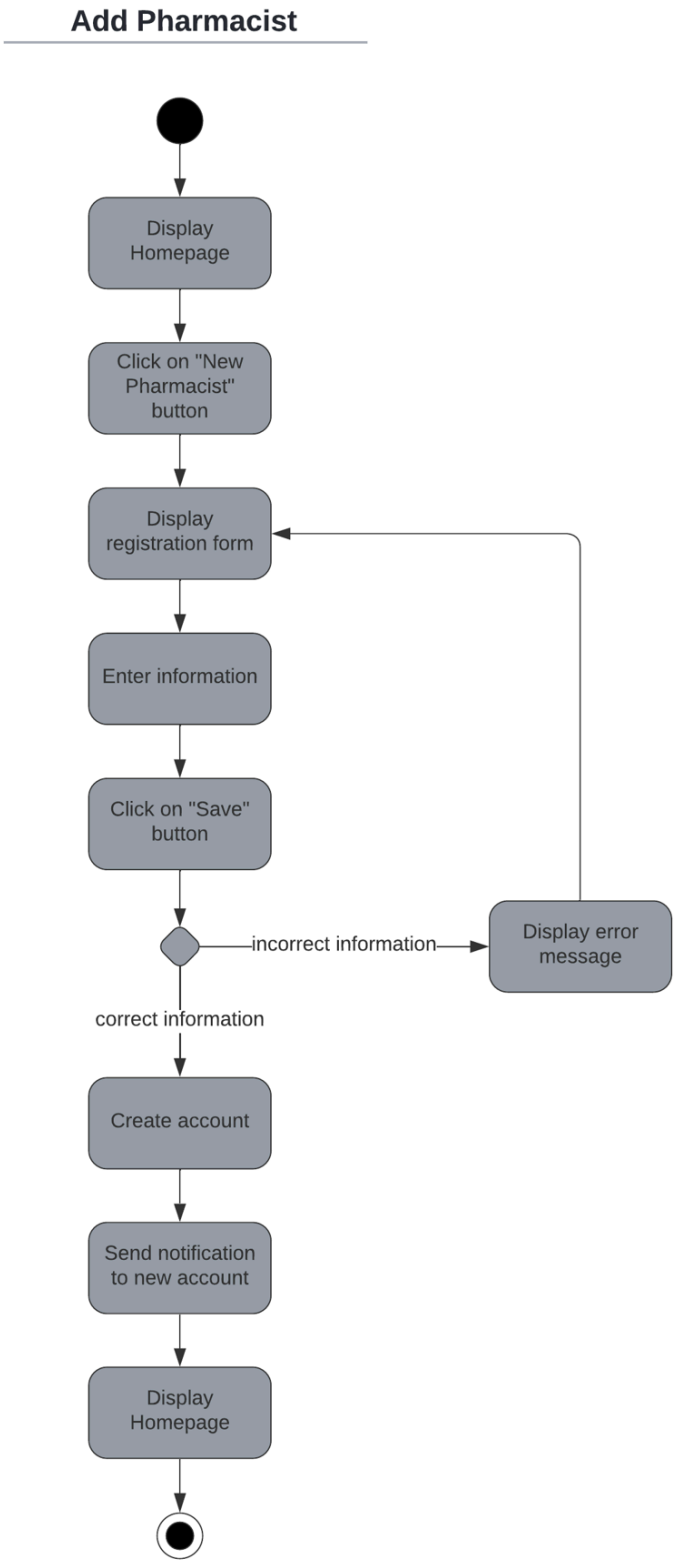

Figure 23. Activity Diagram for Add Pharmacist

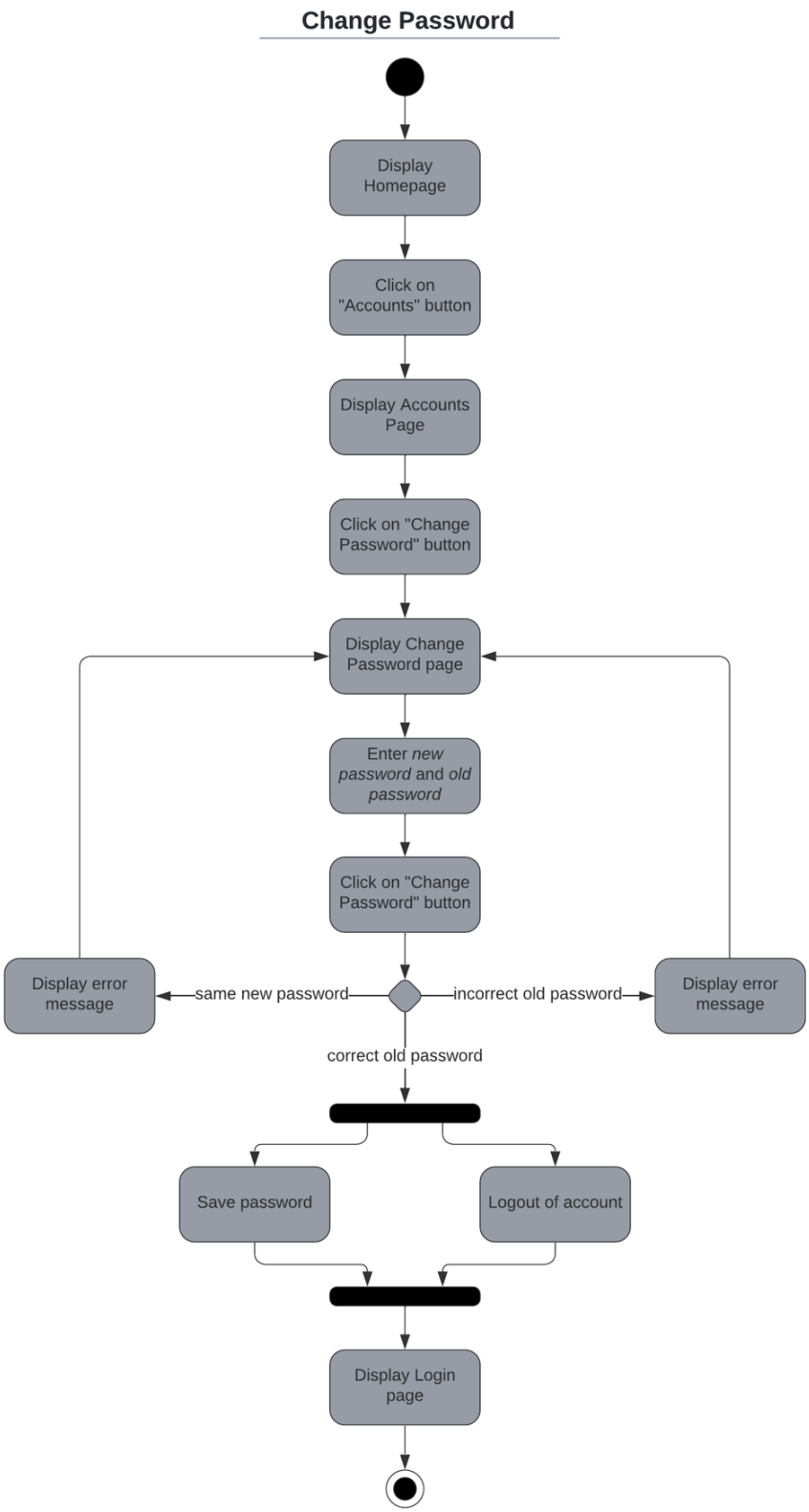

Figure 24. Activity Diagram for Change Password

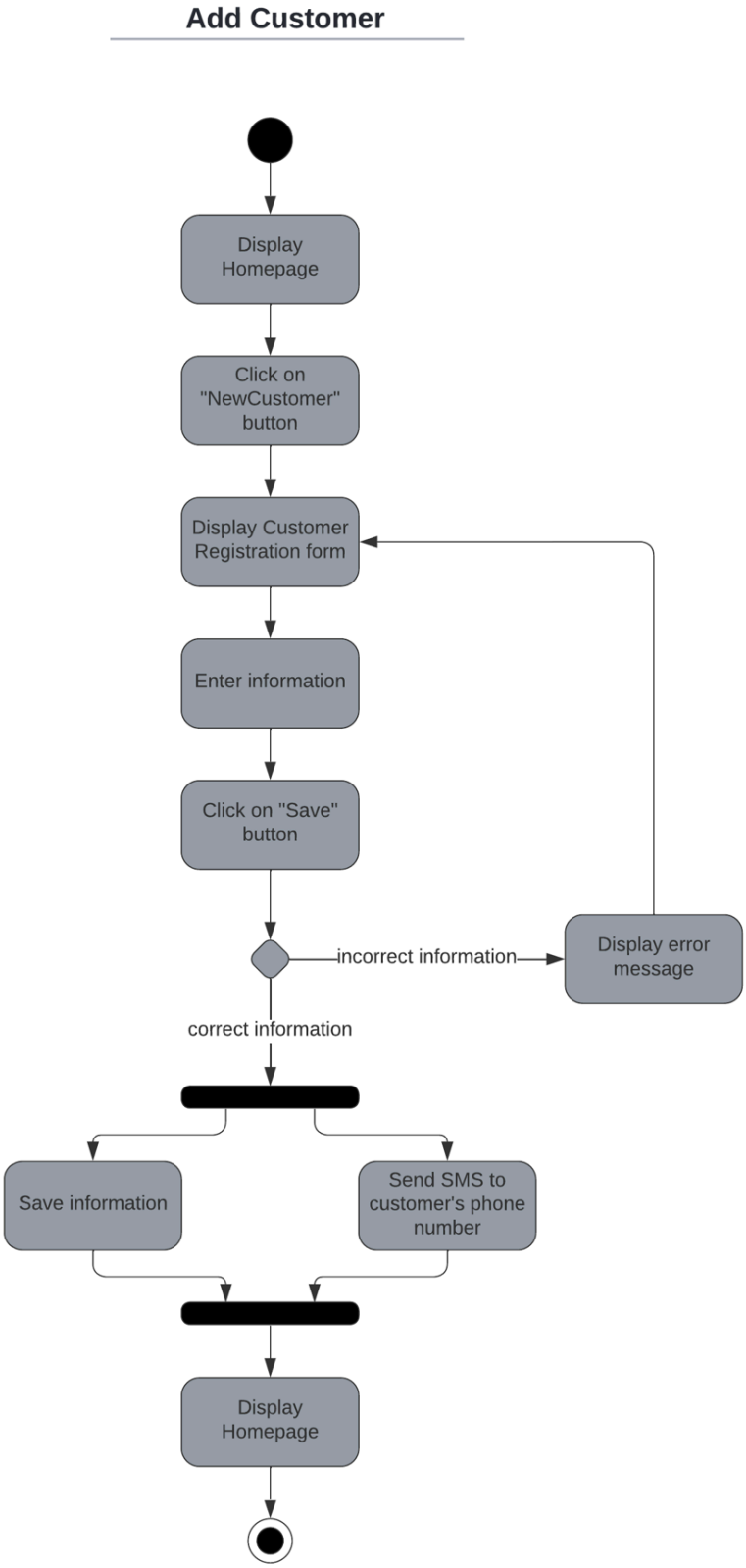

Figure 25. Activity Diagram for Add Customer

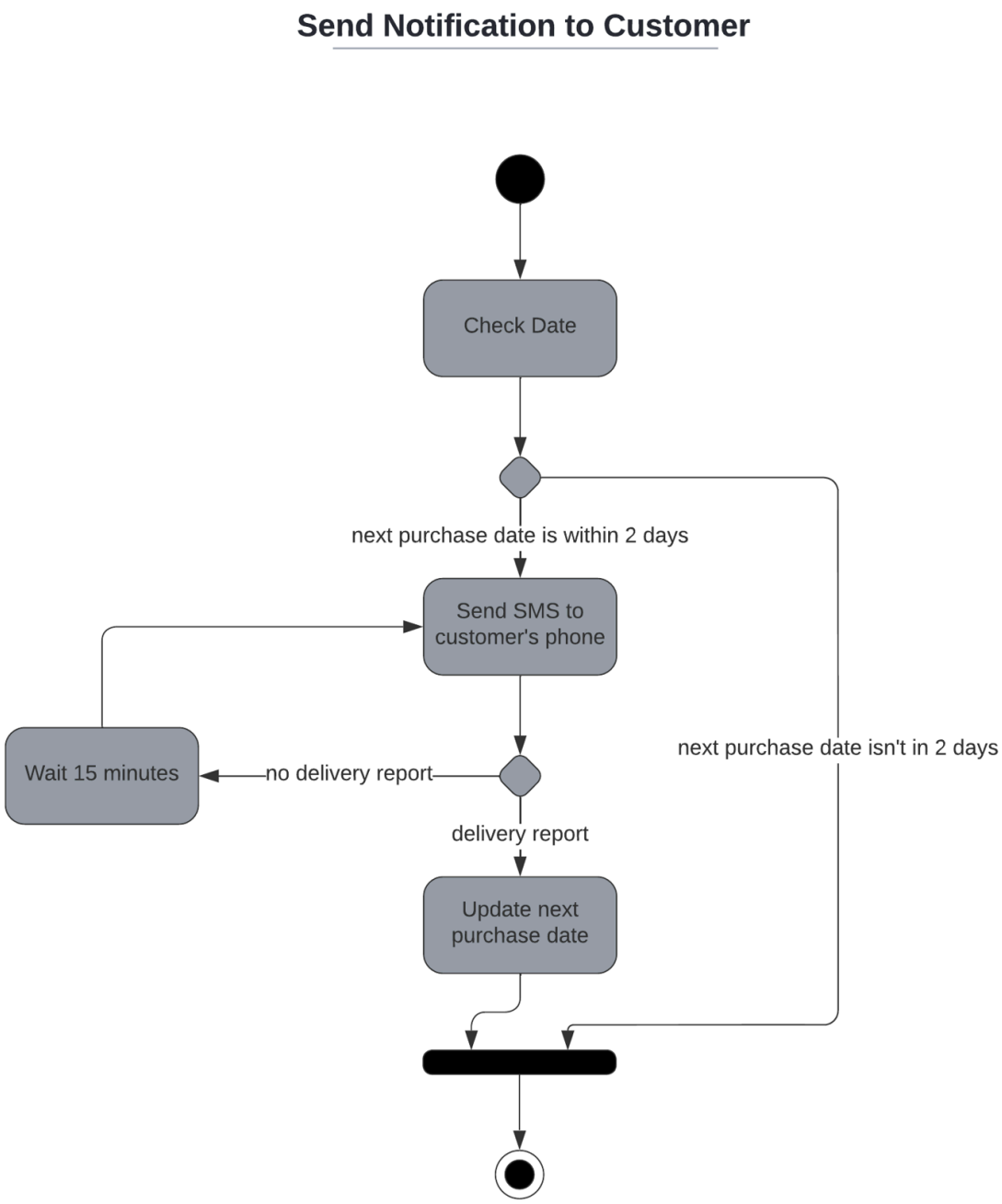

Figure 26. Activity Diagram for Send Notification to Customer

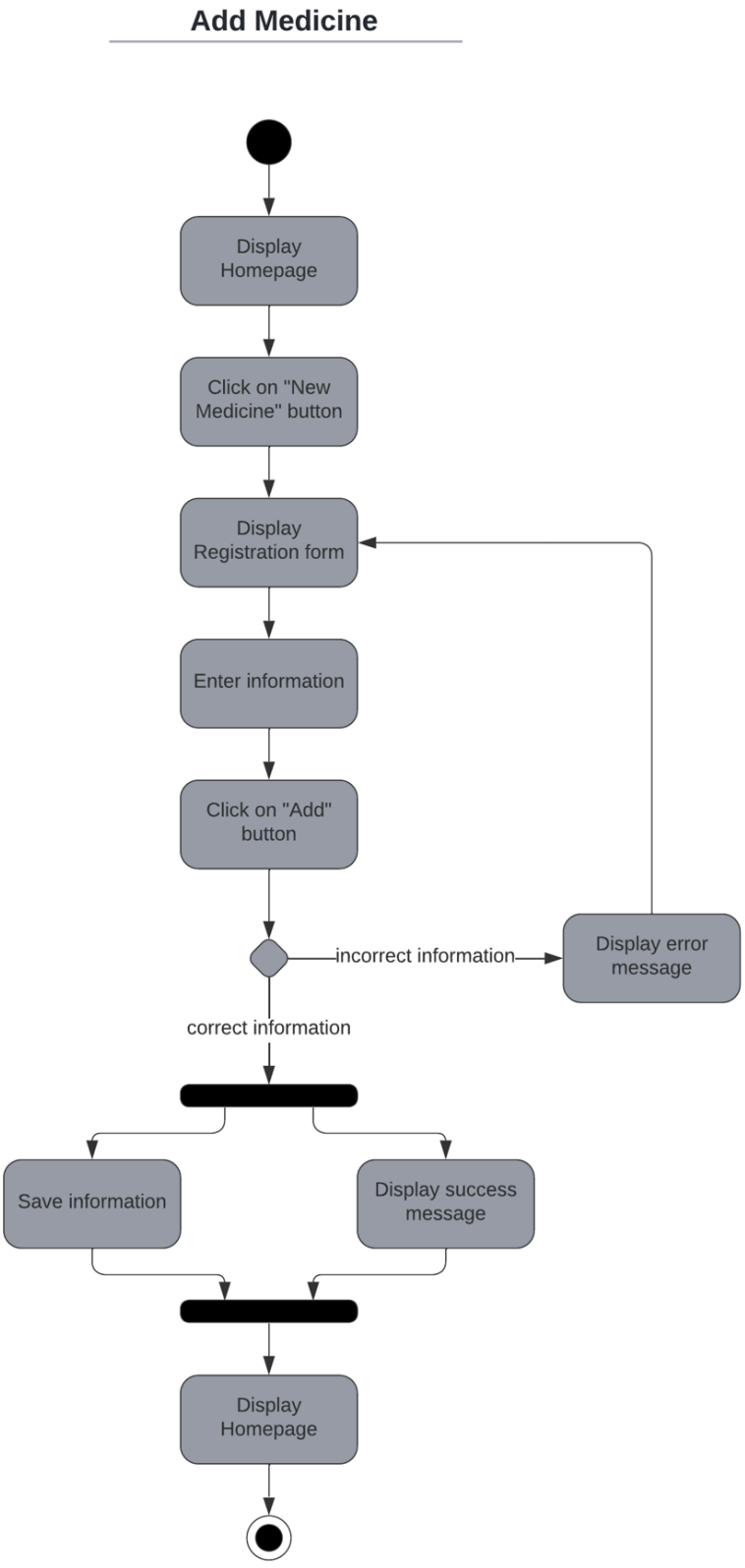

Figure 27. Activity Diagram for Add Medicine

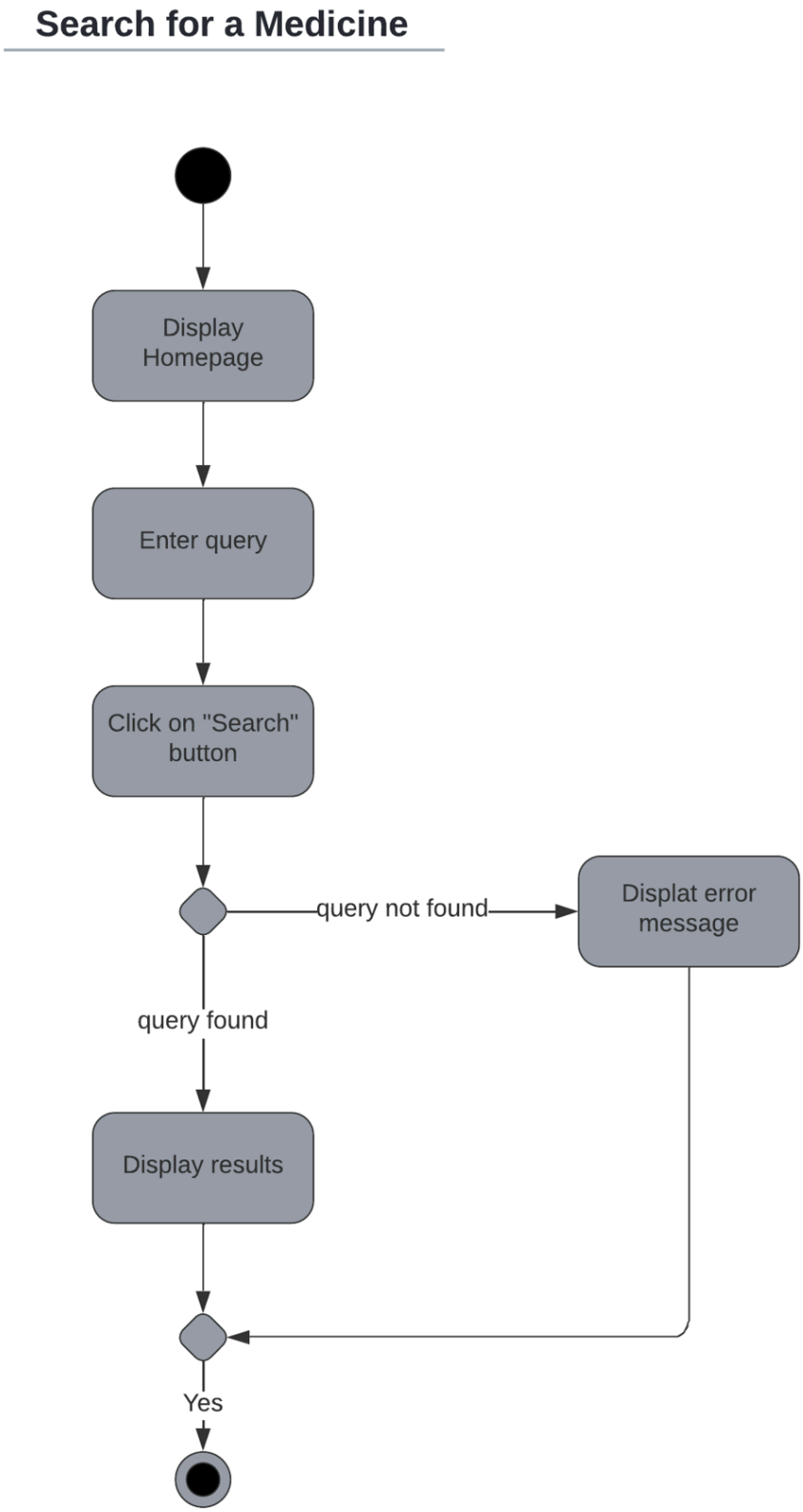

Figure 28. Activity Diagram for Search for an Item

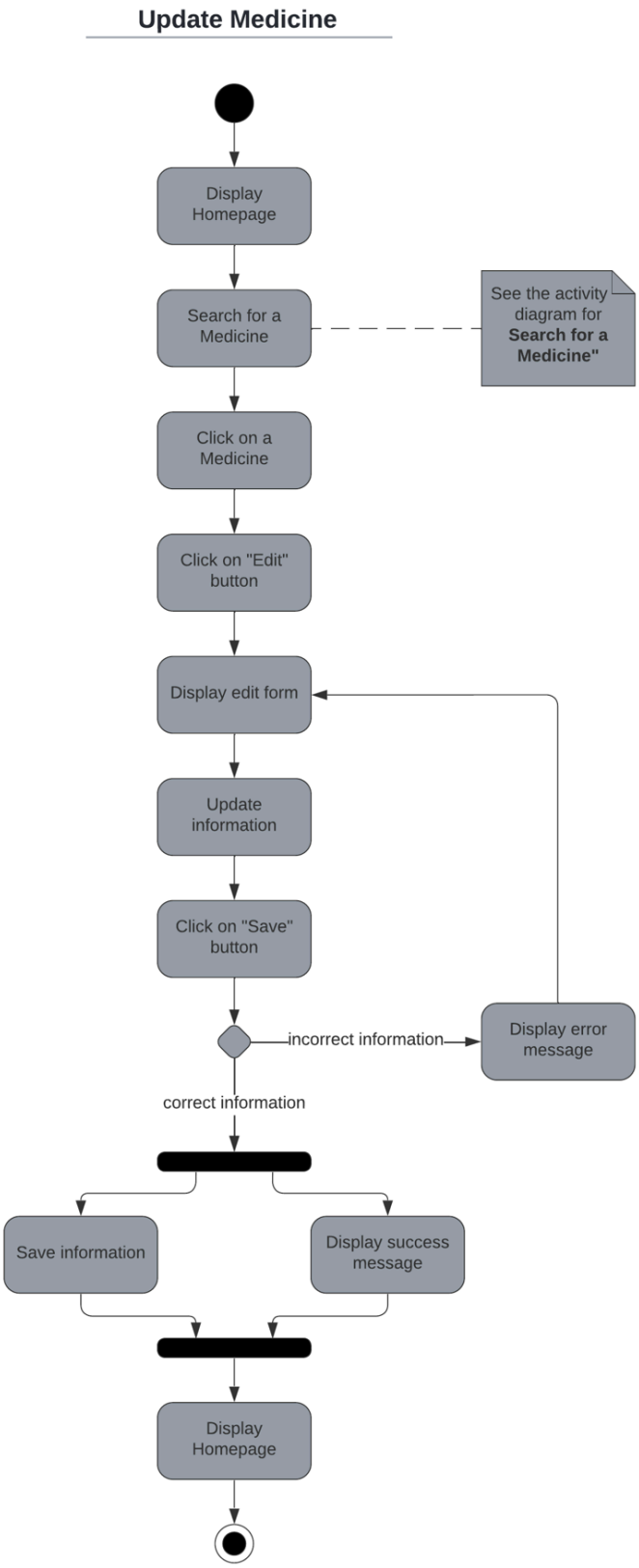

Figure 29. Activity Diagram for Update Item

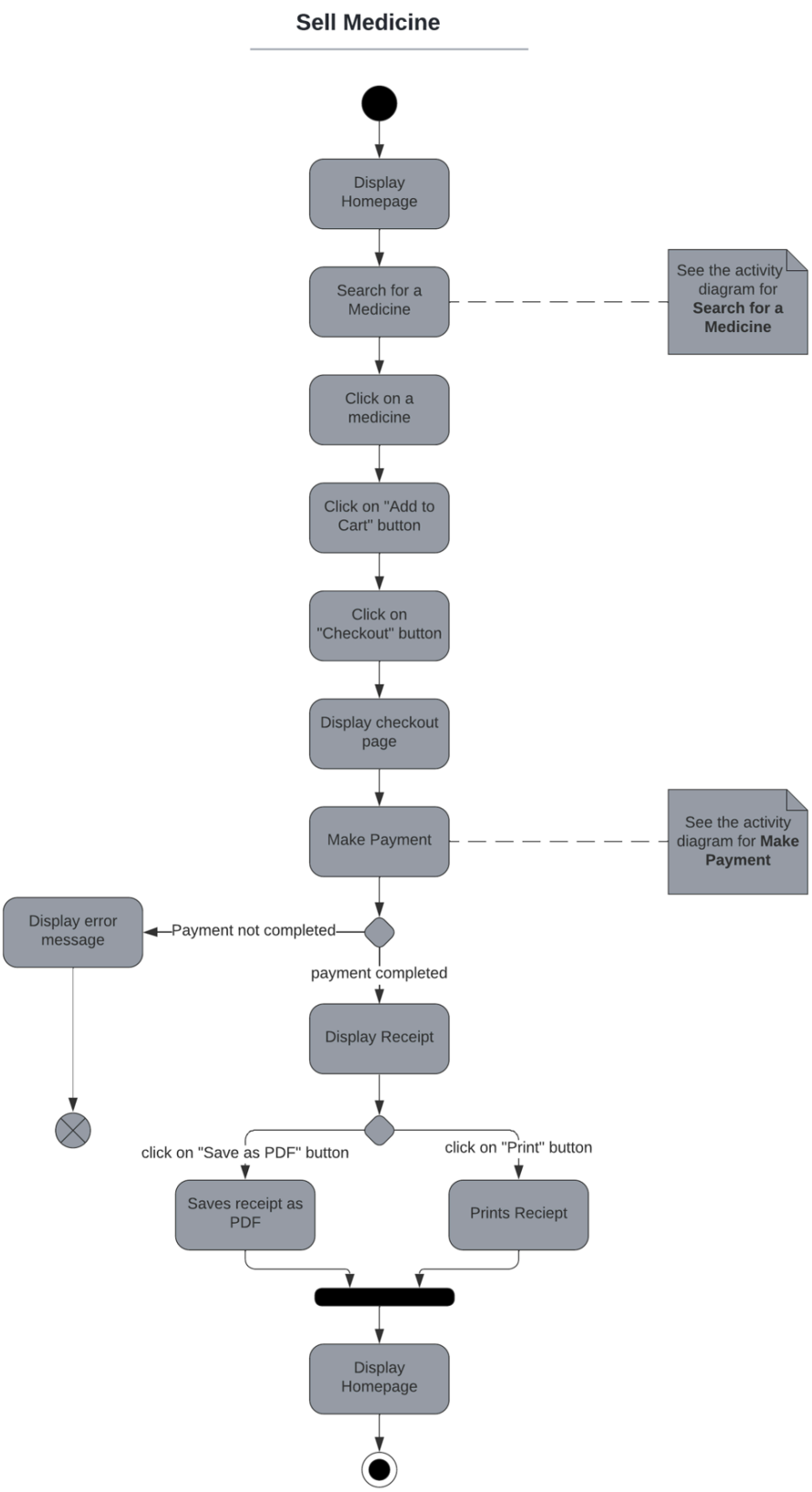

Figure 30. Activity Diagram for Sell Item

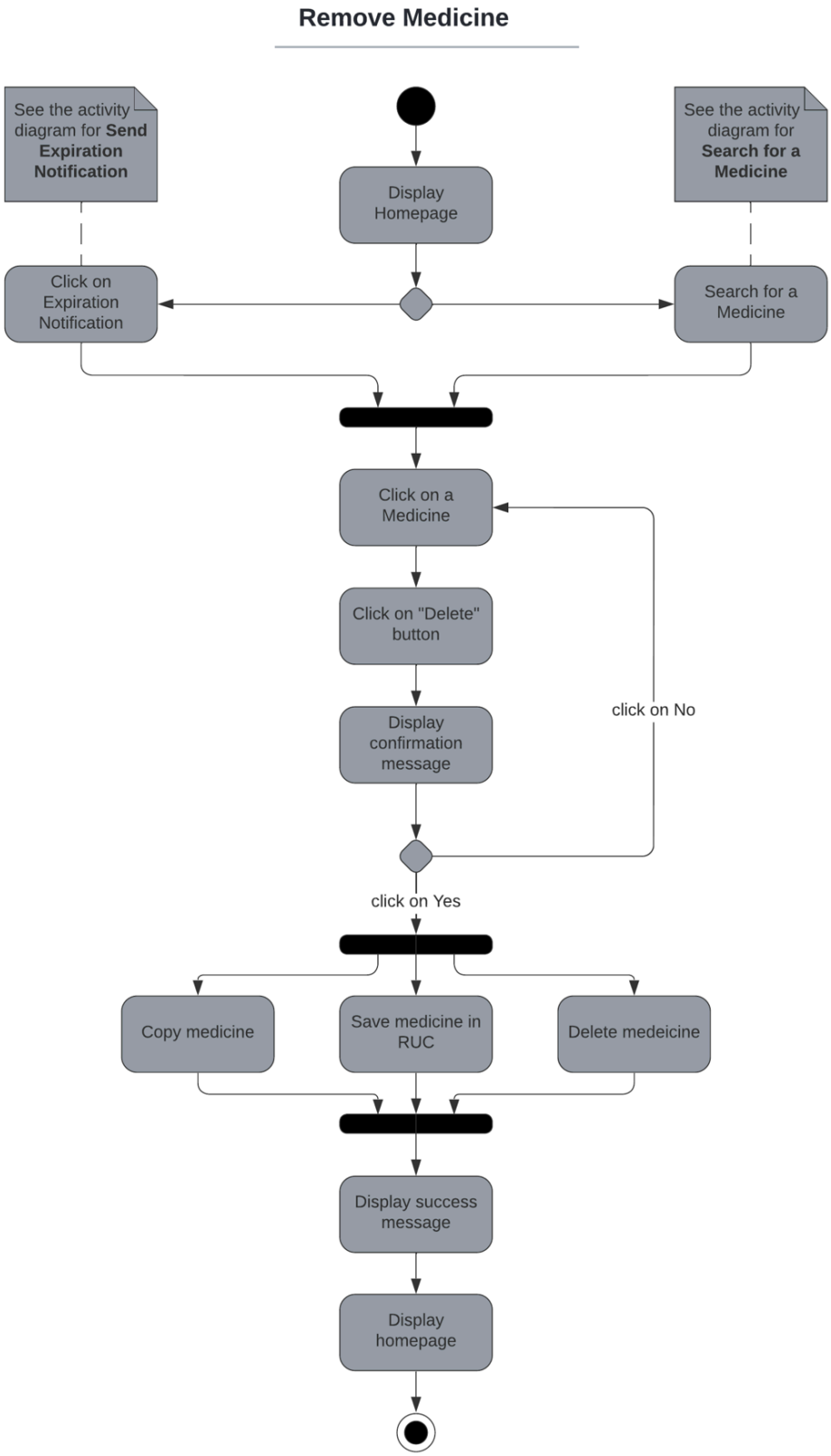

Figure 31. Activity Diagram for Remove Item

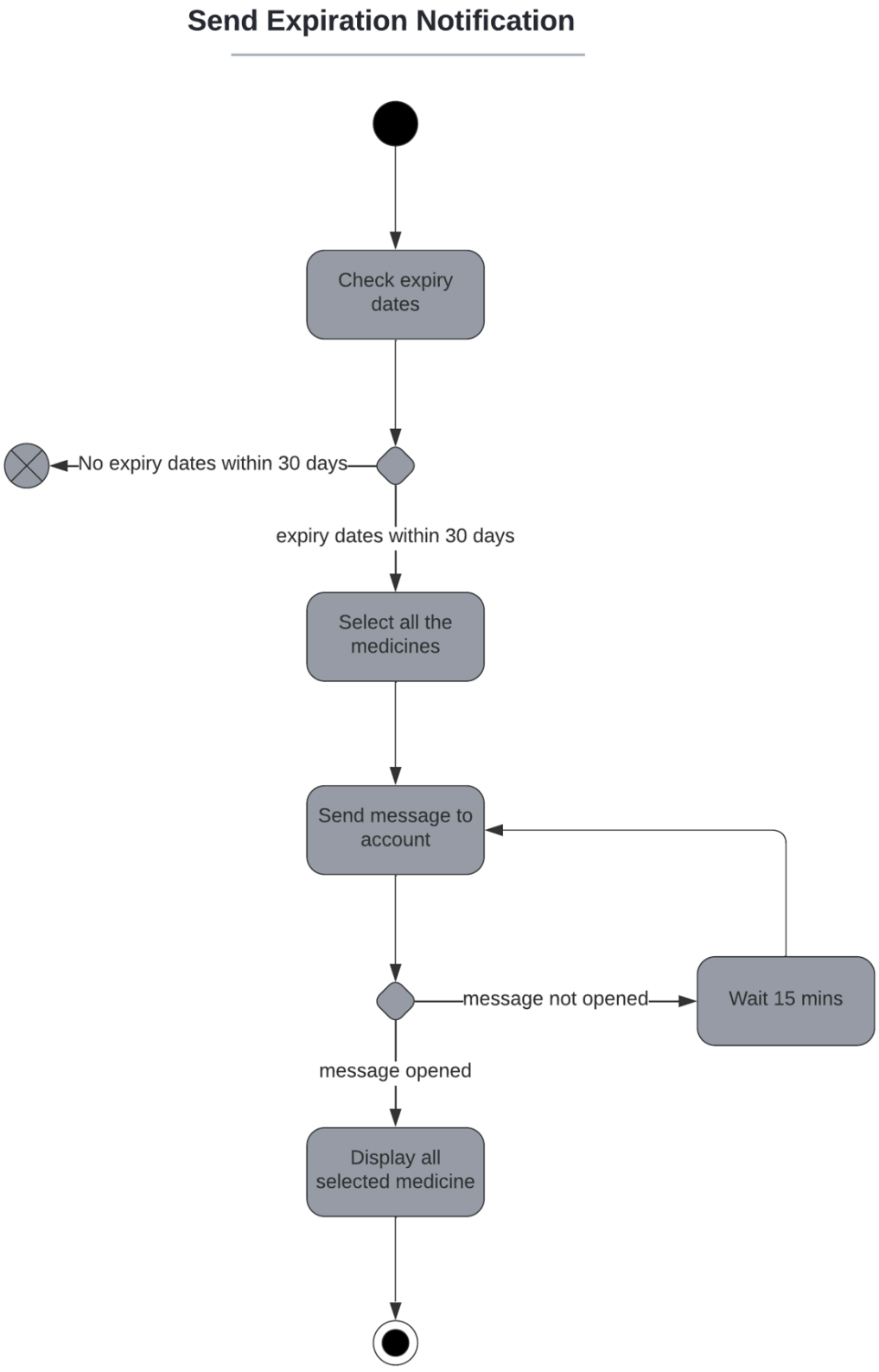

Figure 32. Activity Diagram for Send Expiration Notification

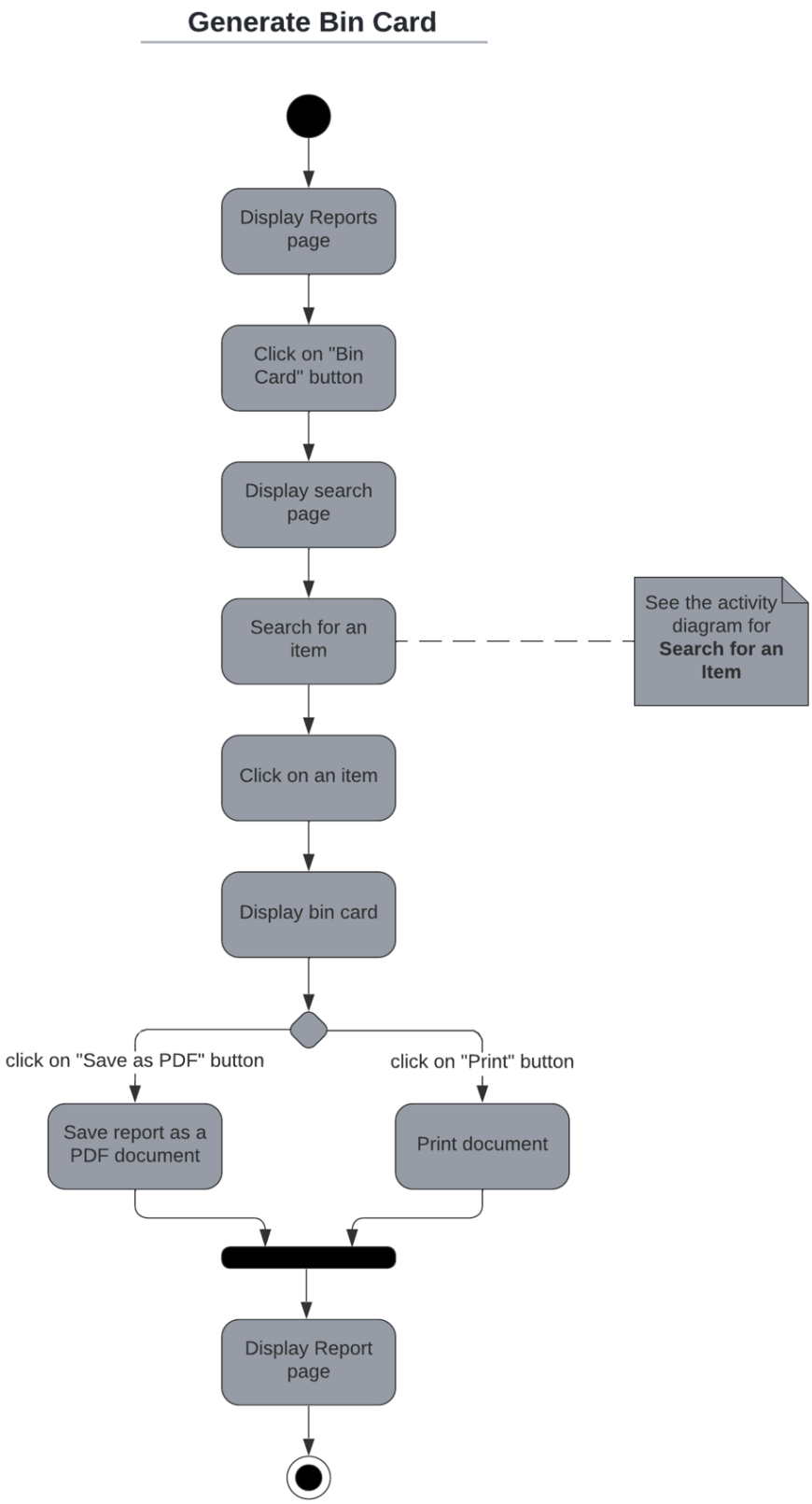

Figure 33. Activity Diagram for Generate Bin Card

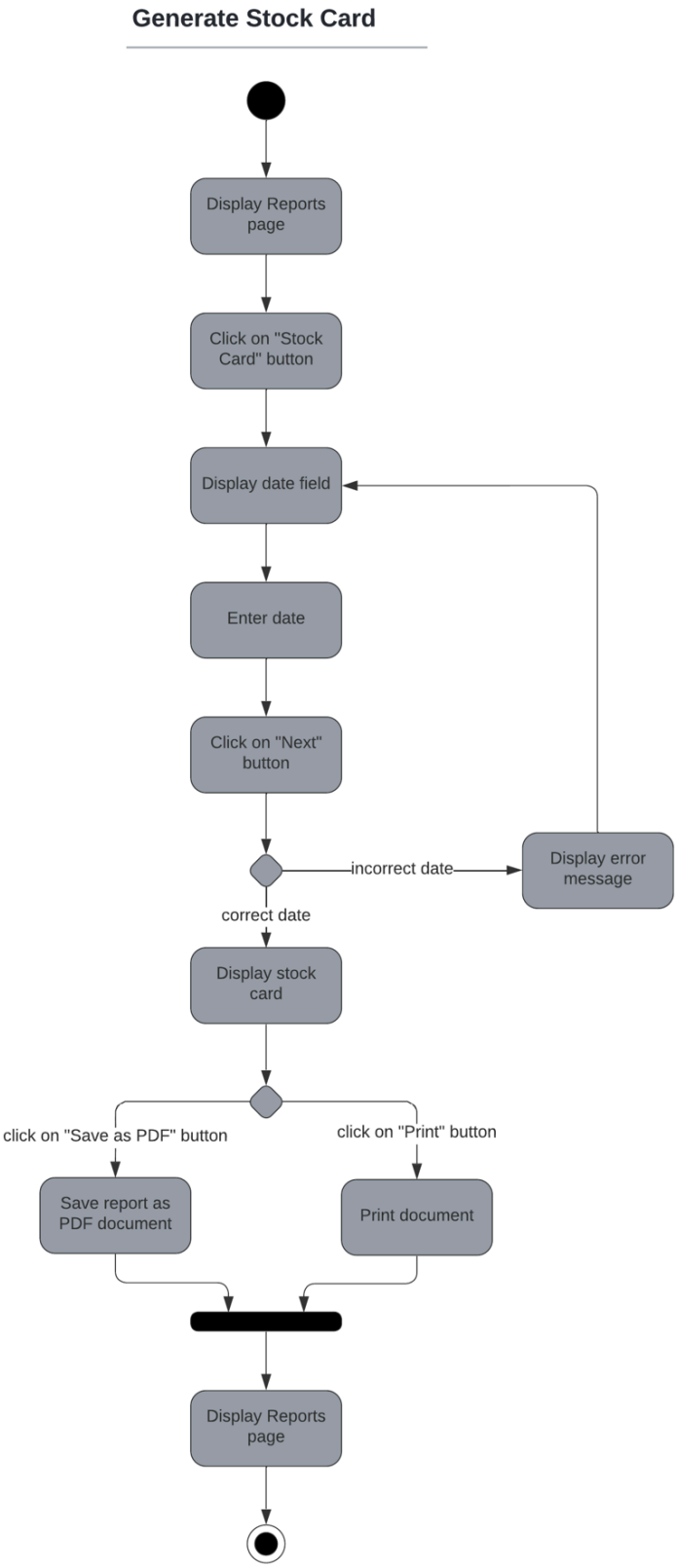

Figure 34. Activity Diagram for Generate Stock Card

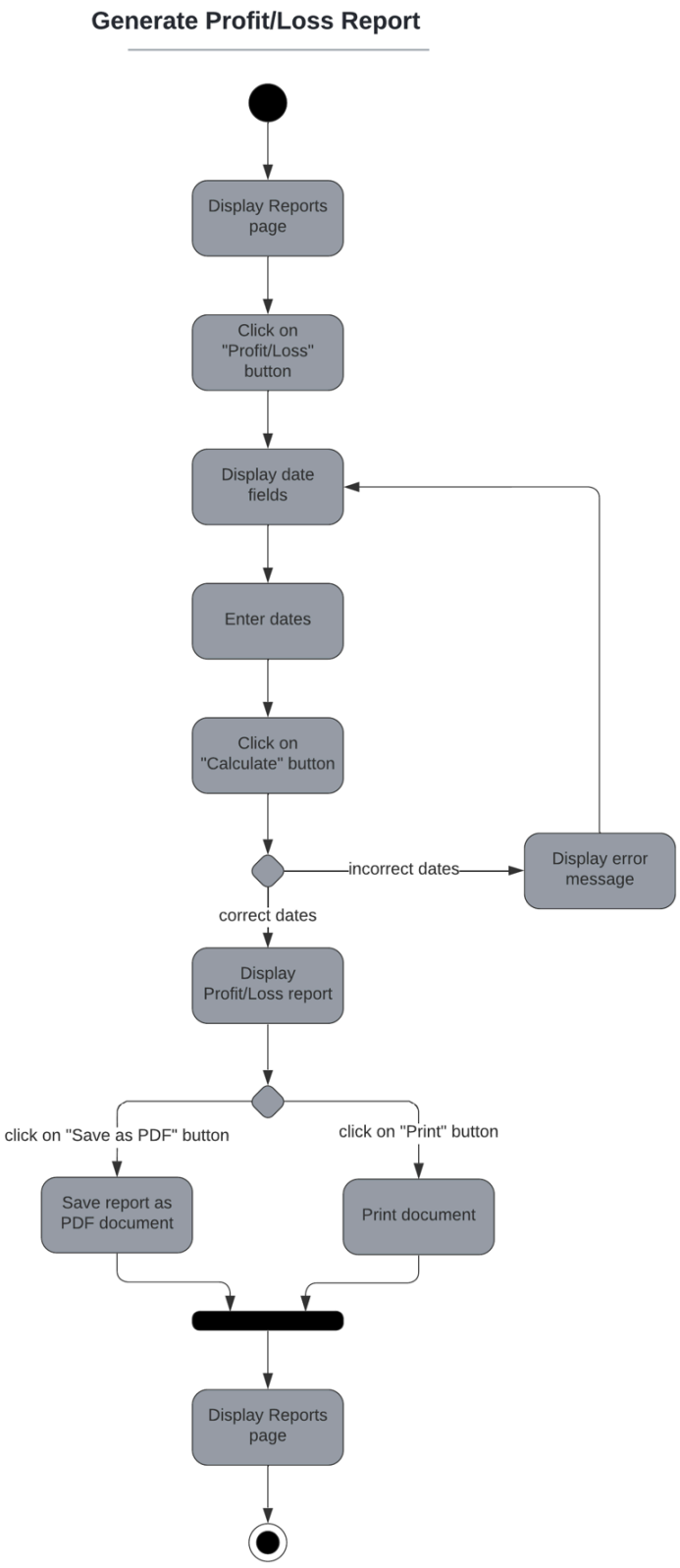

Figure 35. Activity Diagram for Generate Profit/Loss Report

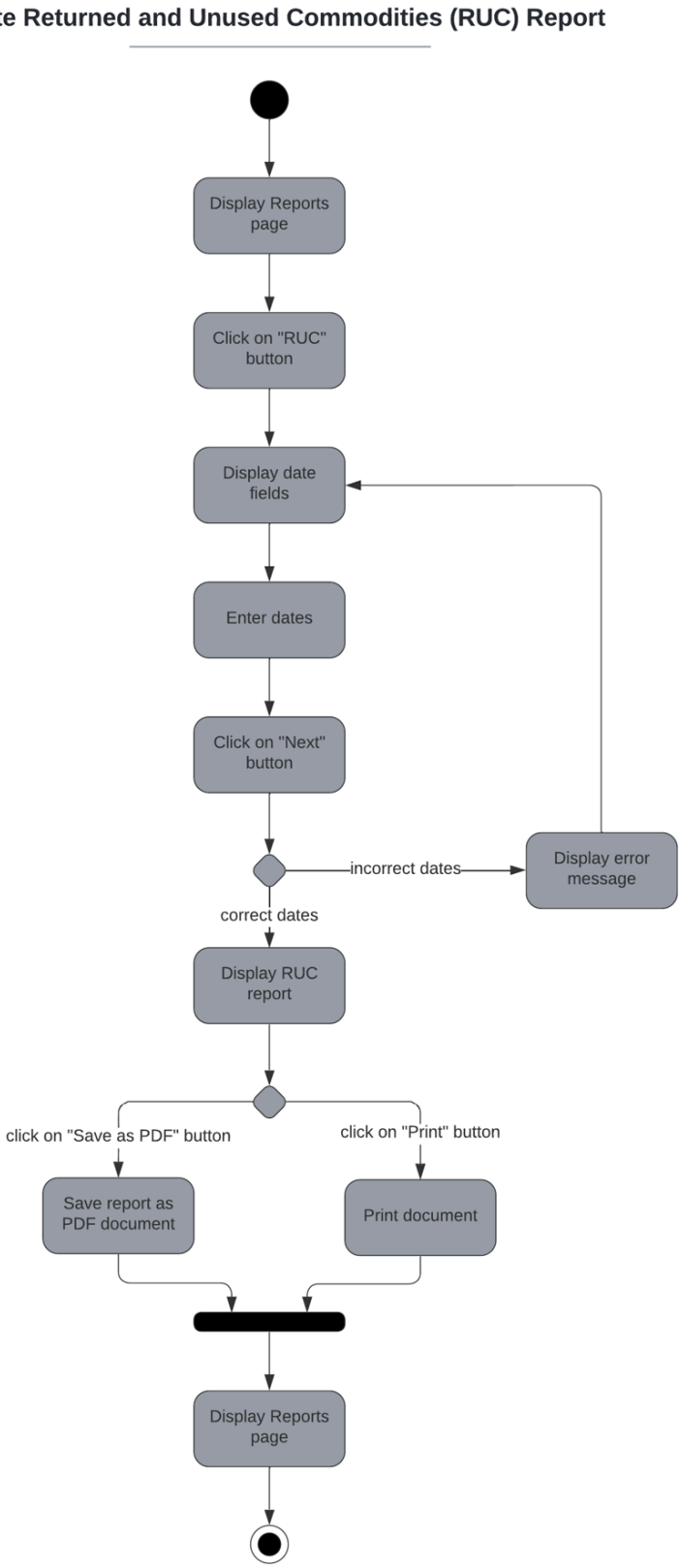

Figure 36. Activity Diagram for Generate Returned and Unused Commodities (RUC) Report

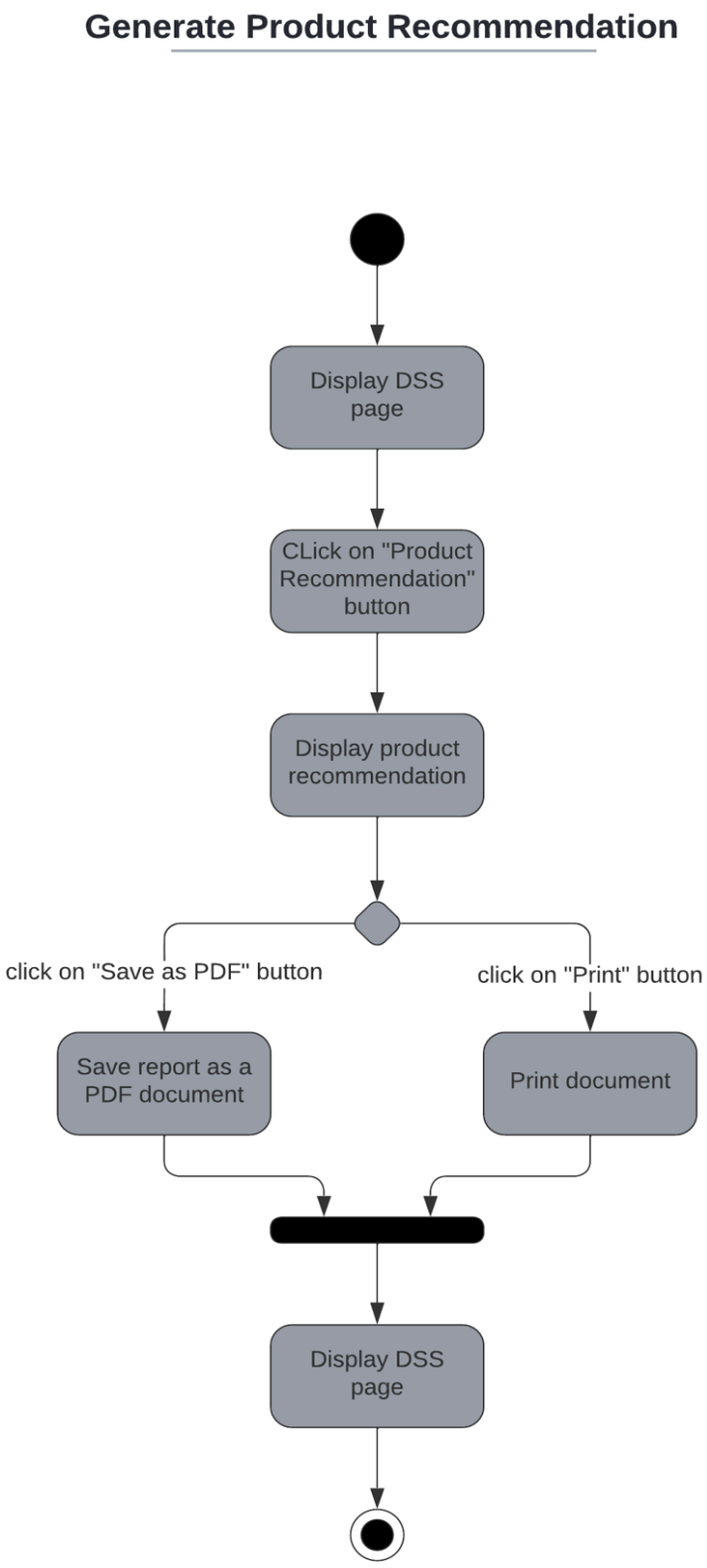

Figure 37. Activity Diagram for Generate Product Recommendation

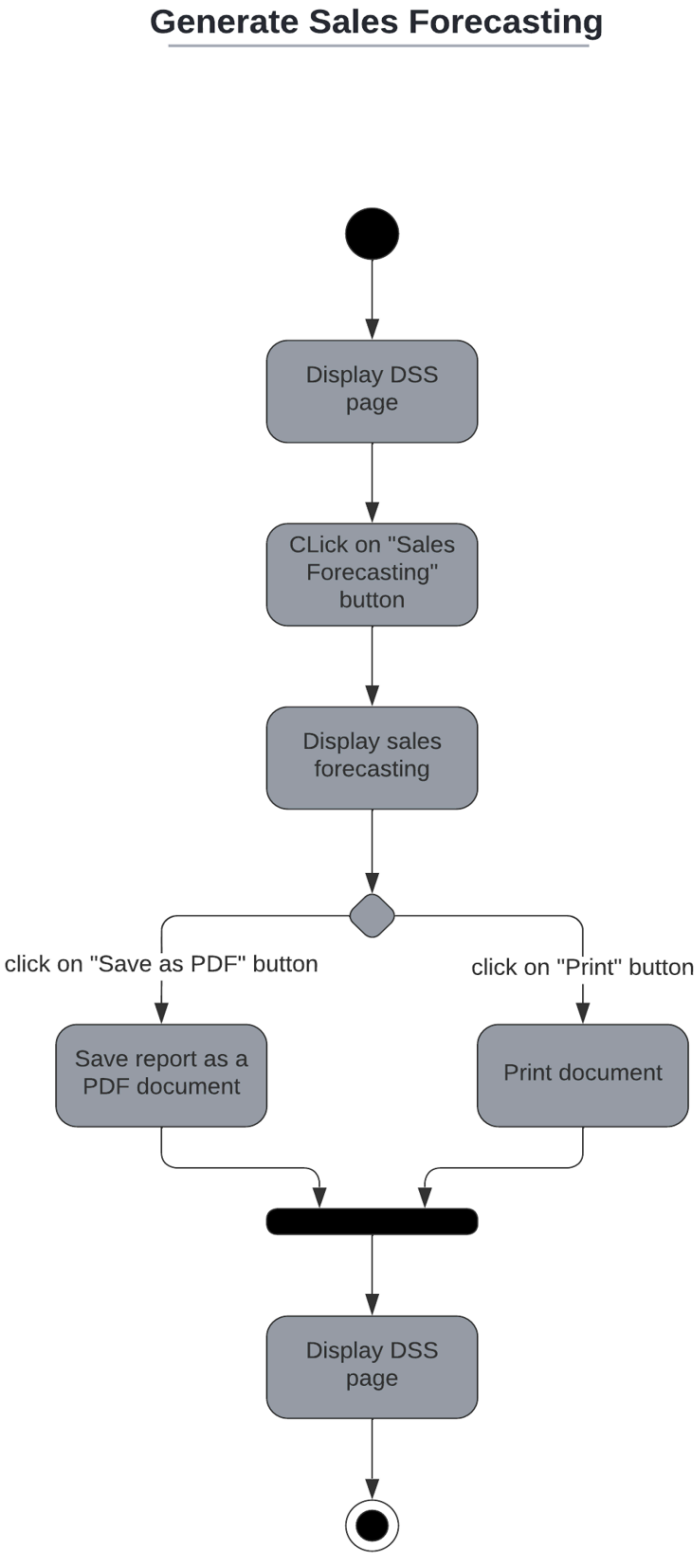

Figure 38. Activity Diagram for Generate Sales Forecasting

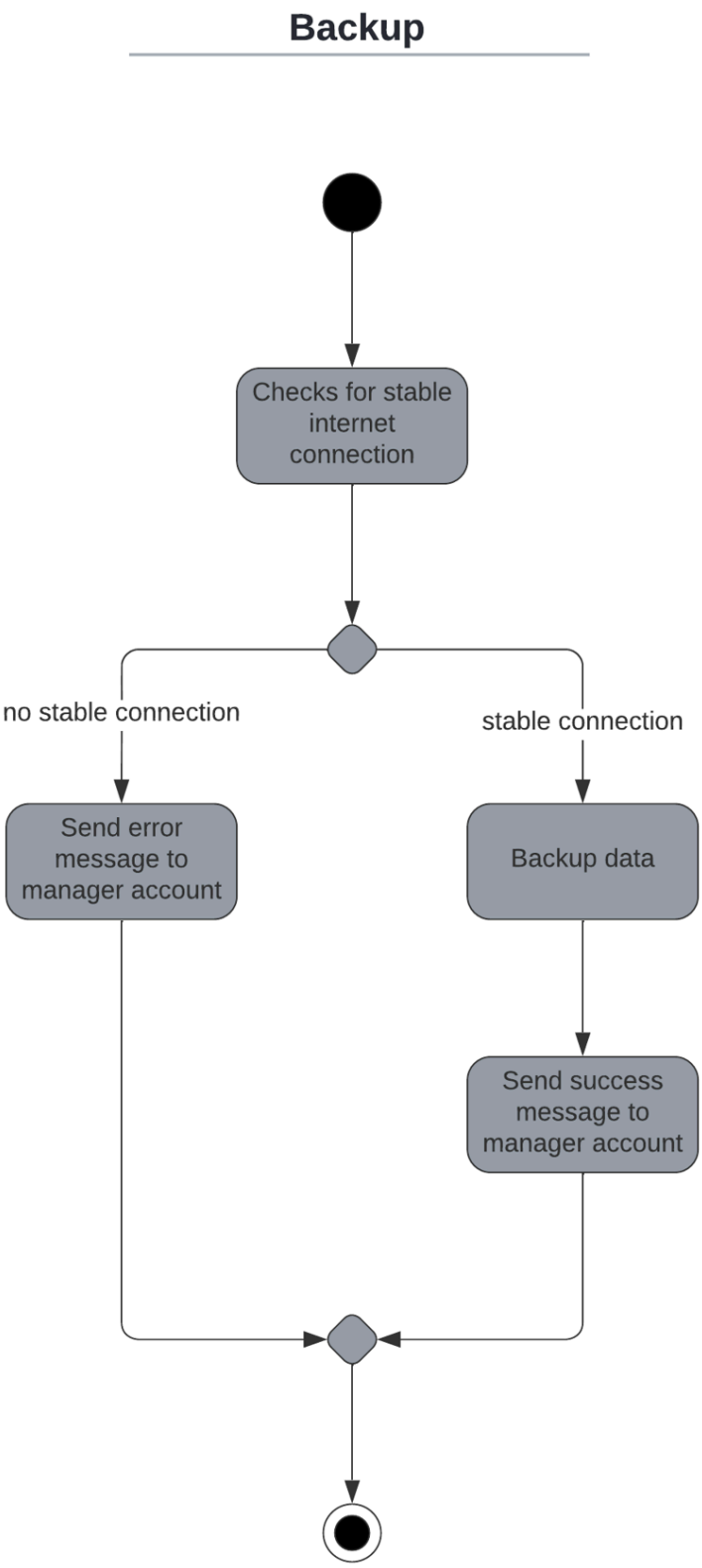

Figure 39. Activity Diagram for Backup

3. System Design Document

3.1. Introduction

The importance of efficient and high-quality healthcare facilities in a country cannot be overstated. Pharmacies are an essential component of every healthcare system. According to Proclamation 1112/2019, every retail pharmacy, whether government or private, should have a quality control mechanism in place [2]. The most crucial aspect of these quality control systems is an effective store management system. The current management system is deficient in several areas, making monitoring and controlling drugs throughout the country excessively complex. This project tries to propose an applicable solution to the current system's difficulties.
The System Design Document (SDD) is a document that allows the conversion of the analysis model into the system design model from which the system is developed. Design goals, software architecture, subsystem decomposition, hardware/software mapping, access control and security, boundary control and subsystem services are thoroughly discussed below.

3.1.1. Purpose of the System

The proposed system's goal is to develop a computer-based management system that will replace the current paper-based system. The system will include an inventory management system that is dependable and efficient, an online payment system, and an AI-based decision support system. Finally, the system intends to assist pharmaceutical retailers in better controlling their companies and providing great healthcare to the country. Meanwhile, the data-driven decision support system will provide insight into how to make organisations lucrative and sustainable.

3.1.2. Design Goals

Design goals are the standards that are met while designing a system. Realising these goals throughout the design will make sure that the system will meet the non-functional requirements and operate to the highest level of ease and satisfaction for the users. Below are the design goals of the proposed system.

Performance

Performance is a design goal that has directly to do with response time of the system. The system should be ready to respond to any requests that the user may have. As stated in the non-functional requirements section of the System Requirements Specification Document, the system should be able to give a response within 5 seconds of any requests made. In order to achieve this, the system implements very efficient querying techniques that have ready made indexes for faster search results. In addition, unnecessary lines of code and comments that may slow down the performance of the system will be removed.

Dependability

Dependability refers to the overall degree of trust the user has on the system. The trust of the user depends on various characteristics of the system.

- **Robustness:** The system should be able to reliably handle the submission of incorrect information by displaying the necessary error message and directing the user to enter the data correctly again. It should be able to do this without crashing or having a major impact on its performance.
- **Reliability:** The system should be able to correctly deliver services as expected by the users. Regardless of whether the user makes an error or not, the system should respond accordingly.
- **Fault Tolerance:** The system should be able to perform under erroneous conditions like internet outage and authentication errors. Any breakdowns that happen should be handled and messages should be passed on to the user via messages displayed.
- **Availability:** The system should be accessible at all times for users, unless there is an uncontrollable catastrophe that is outside the scope of the system like power outage, hardware damage and OS related issues.
- **Security:** The system should be resistant to any intrusions and attempts of damaging or stealing data. The confidentiality, integrity and availability of the information that goes through the system should be ensured. In order to achieve this, the system will enforce

an authentication that requires username and password. In addition, the system's admin account will only be accessible to the manager and its password will be changed every month.

Maintainability

Maintainability is the extent to which a system is capable of being adapted to new requirements. An Object-Oriented approach is used in the development process of the system. Therefore, modifying the system should not pose any problem. The modification of a single or group of functionalities can be done without affecting the system in its entirety. In addition, expanding the system to incorporate new requirements is also possible thanks to the object-oriented approach followed. The addition of subsystems should be very easy as the system is composed of subsystems itself.

Usability

The system shall be easy to learn and operate. A user should be able to learn all the operations of the system after an hour of training. It should have a simple and consistent user interface that enables users to learn to operate, prepare inputs for and interpret outputs of the system with ease. It should also have a well-written documentation and a thorough user manual as well.

3.2. Software Architecture

Software architecture, simply put, is the organisation of a system. This includes all components and subsystems, how they interact with each other and the environment they operate in, and the principles that are followed while designing the software [7]. In the following sections, the current and proposed software architectures are described.

3.2.1. Current Software Architecture

Currently, most of the pharmacies do not use any software to manage their pharmacies. And those who do, use two-tier or three-tier client/server software architectures. Smaller pharmacies that deploy their own small management system usually use the two-software

architecture in their system. In two-tier, the application logic is either buried in the User Interface or in the database on the server side, or on both sides.

In a three-tier, on the other hand, the application logic exists in a middle tier, separated from the User interface and the database[8]. The pharmacies that use the three-tier client/server software architecture are usually owned by large corporations. Apart from the above, most pharmacies are run on a paper based system with stock cards, bin cards and file books.

3.2.2. Proposed Software Architecture

For the system, a combination of REST API Architecture and MVC Architecture is used. REST stands for Representational State Transfer. It's a software architectural style for implementing web services. Web services implemented using the REST architectural style are known as the RESTful web services [9].

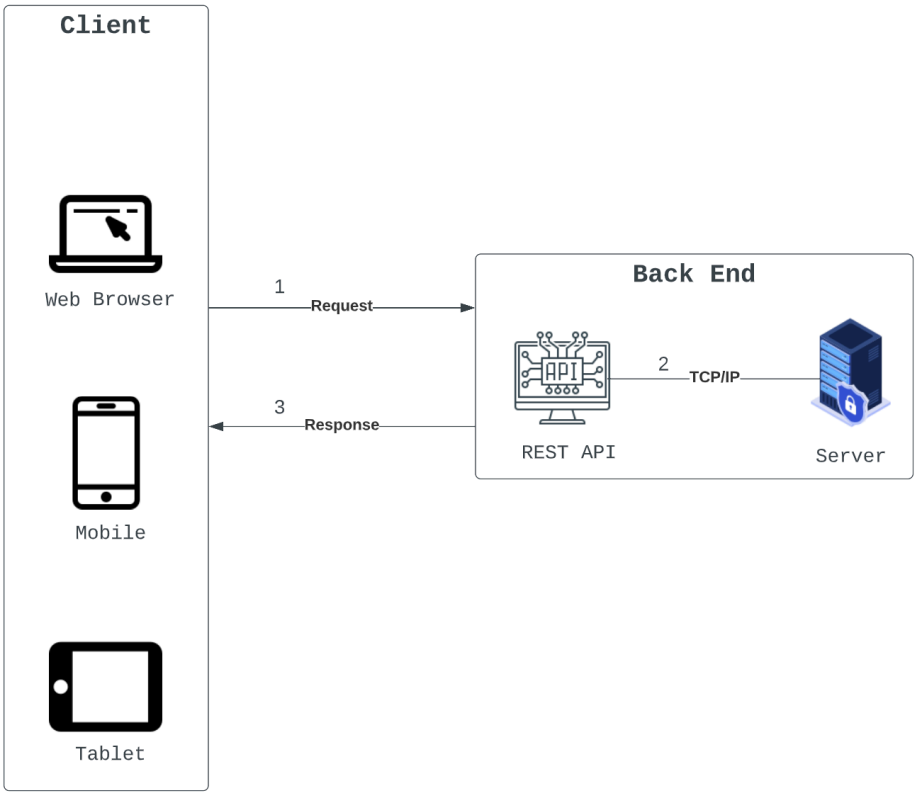

Figure 40. REST API Architecture

3.2.3. Subsystem Decomposition

By breaking down the system into small and manageable sections, subsystem decomposition minimises the complexity of the solution domain. A service defines subsystems and their common purposes. These subsystems are divided to create high cohesion among subsystem objects and loose connectivity with other subsystems to reduce reliance. Loose coupling allows subsystem modifications to have little impact on other subsystems, but strong cohesion only groups subsystems that share common objects. The proposed system has been divided into the following subsystems.

- User Interface Subsystem
- User Management Subsystem
- Inventory Management Subsystem
- Sales Subsystem
- Decision Support Subsystem
- Data Management System

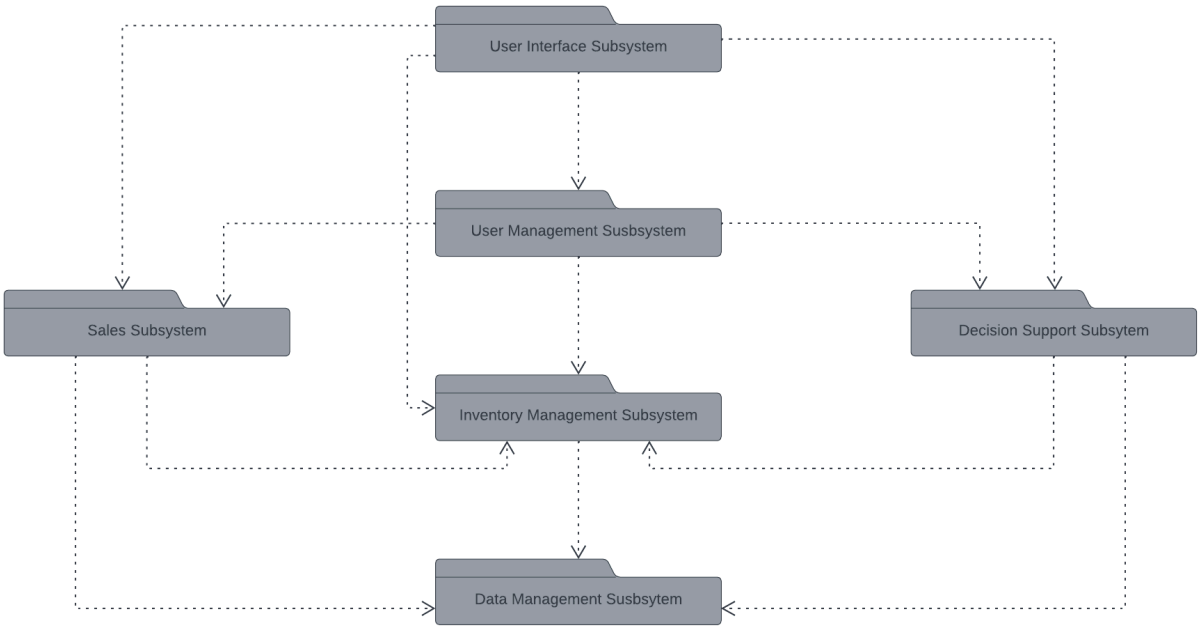

Figure 41. Subsystem Decomposition

User Interface Subsystem

This subsystem provides a different user interface for end users (pharmacist, manager) when they are interacting with the system.

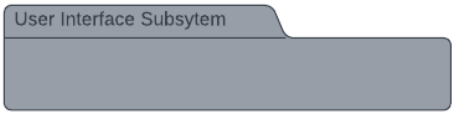

Figure 42. User Interface Subsystem

User Management Subsystem

This subsystem deals with assigning the users roles and enabling them to manage their accounts. Tasks like changing passwords and creating new accounts are handled by this subsystem.

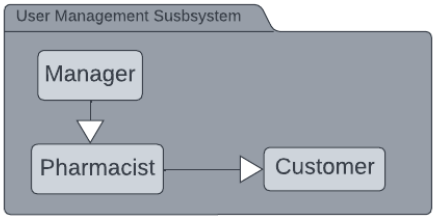

Figure 43. User Management Subsystem

Inventory Management System

This subsystem provides the user with all the items that are in stock and are ready for sale, notification about expiring items and so on.

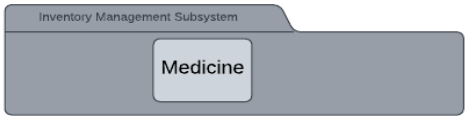

Figure 44. Inventory Management Subsystem

Sales Subsystem

This subsystem handles the execution of every purchase of medicine in the system. It handles payment and records every transaction that is made by the system.

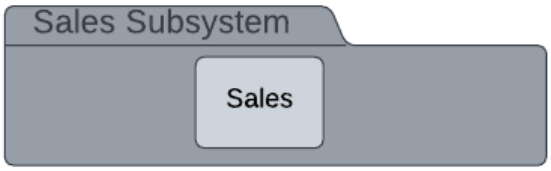

Figure 45. Sales Subsystem

Decision Support Subsystem

This subsystem has to do with the generation of various reports based on data that is generated in the system. It generates bin cards, stock cards, profit/loss reports, RUC reports, product recommendation and sales forecasting reports.

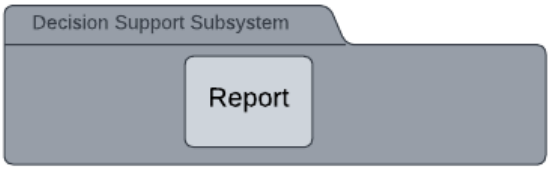

Figure 46. Decision Support Subsystem

Data Management Subsystem

This subsystem has to do with storing the many data that is generated for and by the system. It creates tables, procedures and indexes and stores the relevant information in them for use by the system.

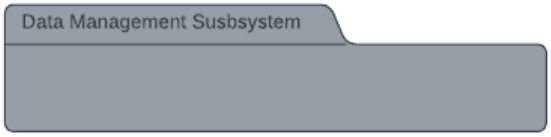

Figure 47. Data Management Subsystem

3.2.4. Hardware/ Software Mapping

In this section a deployment diagram will be used for the depiction of the hardware/software mapping. Hardware/software refers to the process of assigning where the different components or subsystems are deployed on off-the-shelf hardware components. For the proposed system, three nodes are used for deployment. Web browsers on the user's device will be used to access the system. The API is deployed on local Web Server and will contain all the subsystems discussed above except for the Data Management subsystem. The data server is also on a different but local data server and will contain the database. Below is the deployment diagram for the system.

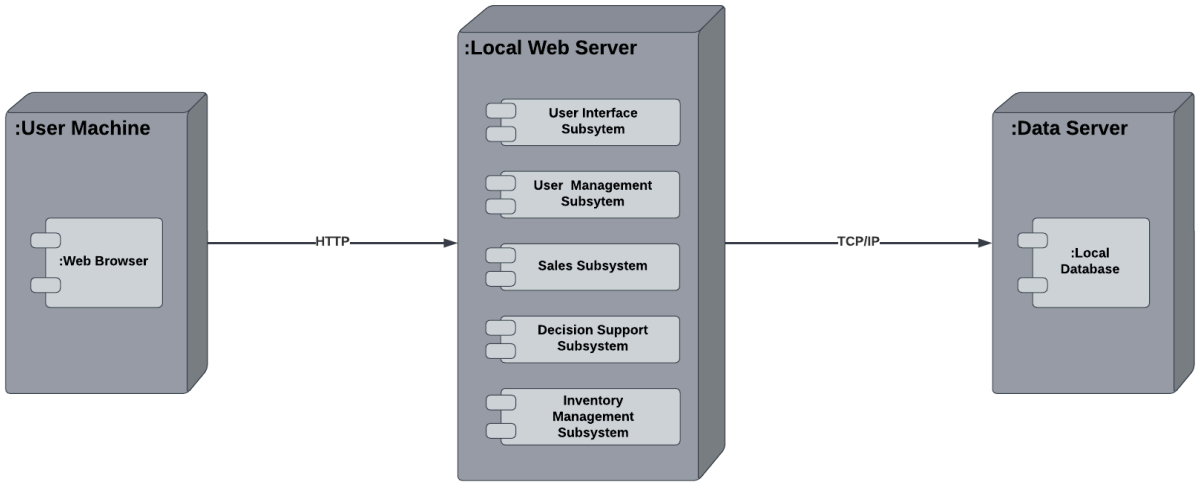

Figure 48. Deployment Diagram

3.2.5. Access Control and Security

Access Control and Security concerns will be addressed in this section. The rights of each user are defined by access control. Access Control rights will be defined using an access matrix. In terms of security, the authentication methods provided and the rules enforced will be discussed.

As mentioned before, the application needs to authenticate a user based on username and password. Based on that, the user's role will be identified and once they are logged in all the privileges that are assigned to that role are given to them. The manager has an administrator role, and is responsible for managing the system. However, certain tasks like selling medicine are deliberately removed from the manager. The pharmacist, in comparison, has less access to the web application. They will be responsible for selling items, registering customers etc. The customer is usually a passive participant in the application. The only time they get involved are when making payment, returning a medicine or receiving a notification.

Use Case	**Operations**	**Actors**		
		Pharmacist	**Manager**	**Customer**
Login	login()	✔	✔	
Add Pharmacist	addPharmacist()		✔	
Change Password	changePassword()	✔	✔	
Add Customer	addCustomer()	✔		
Send Notification to Customer	sendNotificationToCustomer()	✔		
Add Medicine	addMedicine()		✔	

Search for a Medicine	searchByID()	✔	✔	
	searchByExpiryDate()	✔	✔	
	searchByBatchNumber()	✔	✔	
	searchByCategory()	✔	✔	
	listAllMedicine()	✔	✔	
Update Medicine	updateMedicine()	✔	✔	
Sell Medicine	sellMedicine()	✔		
	calculateSellingPrice()	✔	✔	
	confirmPayment()	✔		
Return Medicine	returnMedicine()			✔
Remove Medicine	removeMedicine()	✔	✔	
Send Expiration Notification	checkExpiry()	✔	✔	
	sendExpiryNotification()	✔	✔	
Generate Bin Card	generateBinCard()		✔	
Generate Stock Card	generateStockCard()		✔	
Generate Profit/Loss Report	generateProfitLossReport()		✔	

Generate RUC Report	generateRUCReport()		✔	
Generate Product Recommendation	generateProductRecommendation()		✔	
Generate Sales Forecasting	generateSalesForecating()		✔	
Make Payment	makePayment()			✔
Backup	Backup()		✔	

Table 37. Access Matrix

3.2.6. Boundary Control

In this section, the behaviour of the system in three conditions, namely Startup, Shutdown and Exception, wil be described.

Startup

The startup of the application is upon entering a URL into a web browser on a computer. Then a login page is displayed. If the user provides the correct username and password, they will be logged into their account and can carry out tasks they want.

Shutdown

The web server and the data server are found in the same local area network. Therefore, unless the network is shutdown, the system should be able to function without any interruptions or shutdowns.

Exception

An exception is any unplanned event that may disrupt the functioning of a system, like an invalid input. The application is equipped with validation tools and exception handlings that will ensure the smooth and uninterrupted operation of the application. The validation tools will catch incorrectness in any data submitted and will prompt the user to re-enter the data again. Connection loss and other exceptions are also handled by the system.

3.3. Subsystem Services

Subsystem	Class	Operation	Description
User Management Subsystem	User	login()	Logs the pharmacist into their account.
		changePassword()	Change the password for the account
		addPharmacist()	Creates a new pharmacist account
		sendNotificationToCustomer()	Sends notification to customer's phone
Inventory Management Subsystem	Medicine	addMedicine()	Adds medicine into the stock
		updateMedicine()	Updates medicine information
		removeMedicine()	Deletes medicine from stock
		searchByID()	Searches for medicine using ID as a parameter
		searchByExpiryDate()	Searches for medicine using expiry date as a parameter
		searchByBatchNumb	Searches for

		er()	medicine using batch number as a parameter
		searchByCategory()	Searches for medicine using category as a parameter
		listAllMedicine()	Lists out all medicine available in stock
Sales Subsystem	Sales	sellMedicine()	Sells medicine for customers
		calculateSellingPrice()	Calculates the maximum selling price allowed for that medicine
		makePayment()	Fulfils an online transaction of money for the sales of a medicine
		confirmPayment()	Confirms weather a payment is completed or not
Decision Support Subsystem	Report	generateBinCard()	Generates a bin card for a specific medicine

		generateStockCard()	Generates a stock card
		generateProfitLossReport()	Generates a profit/loss report
		generateRUCReport()	Generates report about returned and unused commodities
		generateProductRecommendation()	Generates a product recommendation report
		generateSalesForecasting()	Generates a sales forecasting report
Data Management Subsystem	Database	Backup()	Backs up the entire system data

Table 38. Subsystem Services

4. Object Design Document

4.1. Introduction

The analytical model was translated into a system design model and explained at a high level in the preceding section. This part discusses the various trade-offs taken, the various rules and conventions followed throughout development, the subsystem's decomposition into packages, and a full explanation of the class interfaces.

4.1.1. Trade-Offs

It is unrealistic to construct a system that is optimally efficient on all fronts while creating it. As a result, various compromises and tradeoffs should be made amongst multiple entities in order to keep the system acceptable to end users.

Performance vs Flexibility

Often, systems are developed to be the most efficient and effective approach to complete a specific task. Although this is true, the majority of solutions now in use in this industry prioritise flexibility over performance. As a result, their performance degrades through time. In this project we have prioritised performance over flexibility because most transactions take place within the pharmacy.

Quality vs Response time

Although quality is vital, the system's response time was prioritised when it was designed. This does not imply that the system does not generate an accurate result; rather, the quality of the output is standard, but the system's response time is optimal. This adds to the fact that performance was the primary focus when creating the system.

Buy vs Build

A recurring theme throughout the system is modularity. The program is created by segmenting it into independent modules that interact with each other to support the entire system.

Although modules require both time and resources, it will be very challenging to design the system without them, and their absence might impair the functionality of the entire program.

4.1.2. Interface Documentation Guidelines

These guidelines are here to improve the understandability of the source code. Moreover, these guidelines will become of use when maintenance is required in the future.

- Variables are written in camelCase. Example: authClaims.
- Methods are written in PascalCase. Example: MedicinesController ().
- Classes are written in PascalCase. Example: MedicinesController.
- Errors that might occur on the input fields will be handled using validators.
- Each input field has a length limit although the length differs according to the input's characteristics.
- Comments have been added to increase the understandability of our implementation.

4.2. Packages

Packages are composed of classes that work with each other to achieve some specific tasks. Any change in a class will affect other classes in the same package. Nevertheless, dependencies exist between classes of different packages. Below are the packages that are in the system.

4.2.1. User Interface Package

Classes of user interface elements that the user will interact with are contained in this package. Since users must communicate with the system, all other classes are related to this package.

4.2.2. User Management Package

All user management related classes are contained here. Functionalities like registering pharmacists, changing passwords and login in are included in this package.

4.2.3. Inventory Management Package

A class that is related to medicine is included in this package. Functionalities like adding, updating, deleting and searching a medicine are included here.

4.2.4. Sales Package

In this package, everything that has to do with sales is included. Selling medicine, making payment, confirming payment and so on are included in this package.

4.2.5. Decision Support Package

This package includes a class that has to do with the generation of various reports that are helpful for the decision making process. Generating profit/loss reports, product recommendation and sales forecasting are included in here.

4.2.6. Data Management Package

This package contains classes that deal with data management. Different requests made by other classes in the other packages for access to manipulation of data are handled in this package.

4.2.7. Package Dependencies

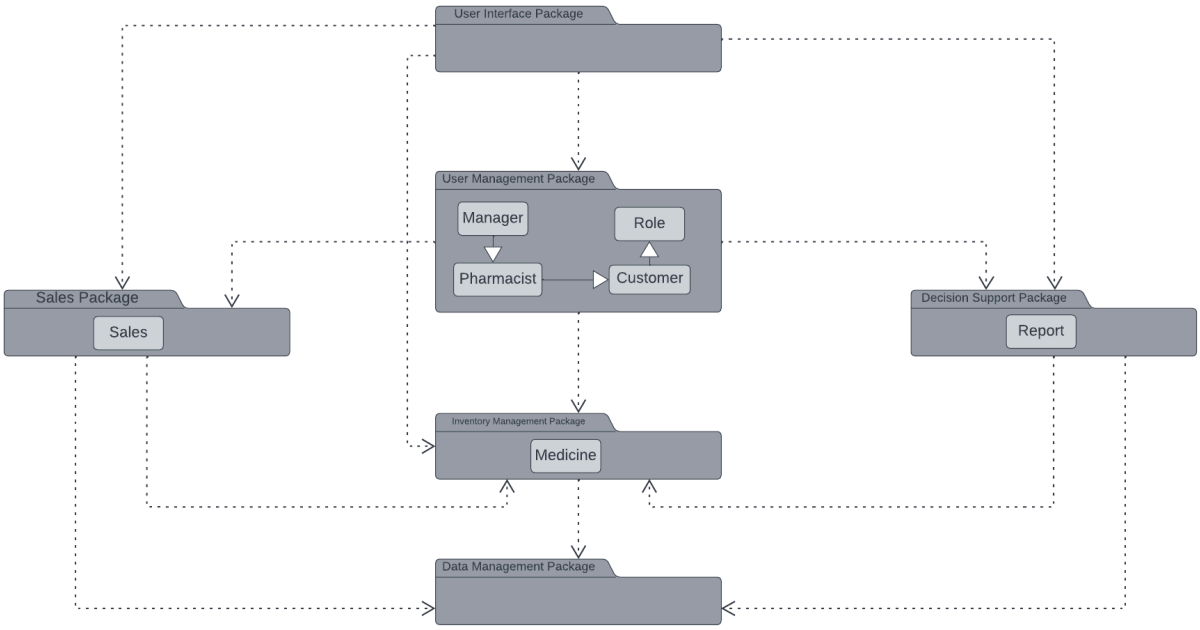

Figure 49. Package Dependencies

4.3. Class Interface

4.3.1. Role Class Interface

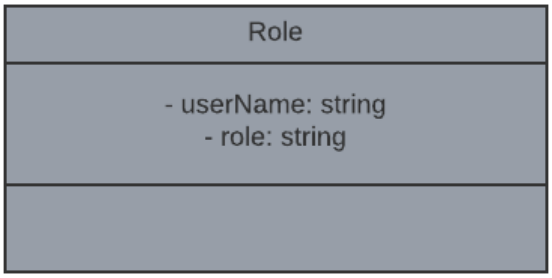

Figure 50. Role Class Interface

Attribute	Data Type	Description
UserName	String	Contains the username of the user(Pharmacist/Manager).
Role	String	Contains the role that the user(Pharmacist/Manager) has.

Table 39. Role Class Attributes

4.3.2. User Class Interface

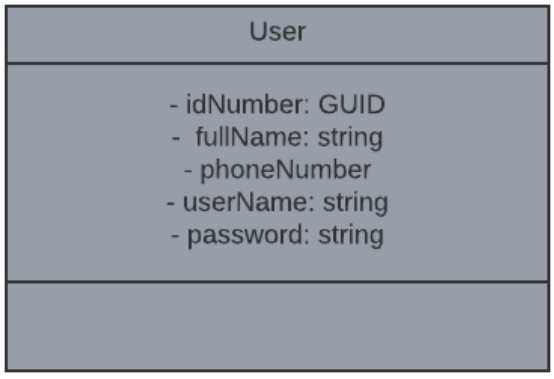

Figure 51. User Class Interface

Attribute Name	Data Type	Description
IDNumber	GUID	Contains the identification number of the user(Pharmacist/Manager).
FullName	String	Contains the full name of the user(Pharmacist/Manager)
PhoneNumber	String	Contains the phone number of the user (Pharmacist/Manager).
UserName	String	Contains the username of the user (Pharmacist/Manager).
Password	String	Contains the password of the user (Pharmacist/Manager).

Table 40. User Class Attributes

4.3.3. Customer Class Interface

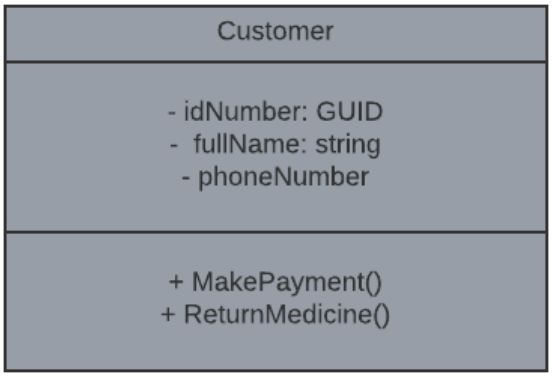

Figure 52. Customer Class Interface

Attribute Name	Data Type	Description
IDNumber	GUID	Contains the identification number of the customer.
FullName	String	Contains the full name of the customer.
PhoneNumber	String	Contains the phone number of the customer

Table 41. Customer Class Attributes

4.3.4. Sale Class Interface

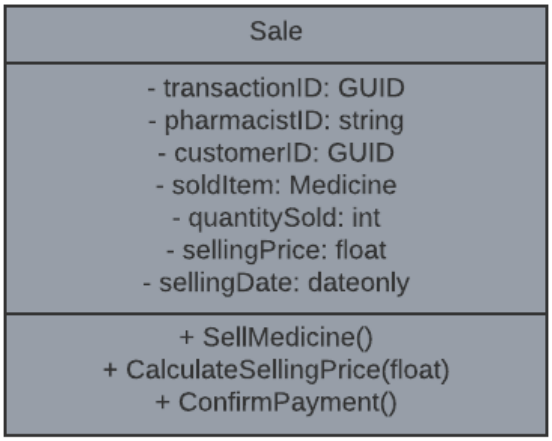

Figure 53. Sale Class Interface

Attribute Name	Data Type	Description
TransactionID	GUID	Contains the identification number of the transaction.
PharmacistID	GUID	Contains the identification number of the pharmacist.
CustomerID	GUID	Contains the identification number of the customer.
SoldItem	Medicine	Contains the username of the user (Pharmacist/Manager).
QuantitySold	Int	Contains the number of items(medicine) sold.
SellingPrice	Float	Contains the selling price of the item(medicine).
SellingDate	dateonly	Contains the selling date of the medicine.

Table 42. Sales Class Attributes

4.3.5. Medicine Class Interface

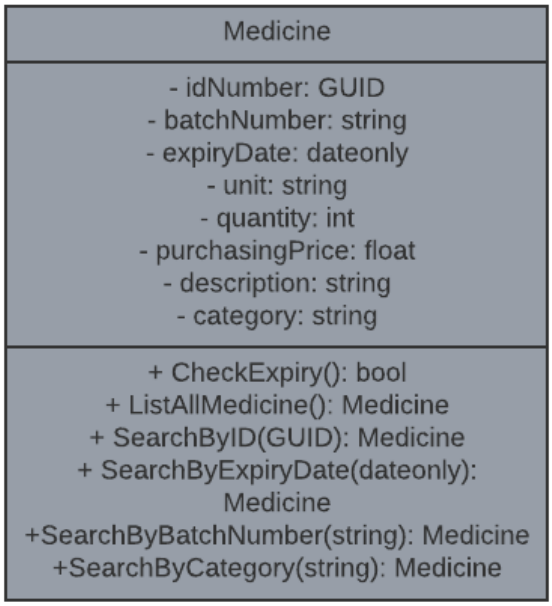

Figure 54. Medicine Class Interface

Attribute Name	Data Type	Description
IDNumber	GUID	Contains the identification number of the medicine.
BatchNumber	String	Contains the batch number of the medicine.
ExpiryDate	dateonly	Contains the expiration date of the medicine.
Unit	String	Contains the type of medicine it is example: syrup, tablet… etc
Quantity	int	Contains the number of medicine that are in store
PurchasingPrice	Float	Contains the price of the medicine
Description	String	Contains a detailed explanation of the medicine.
Catagory	String	Contains the category that medicine is included under.

Table 43. Medicine Class Attributes

4.3.6. Report Class Interface

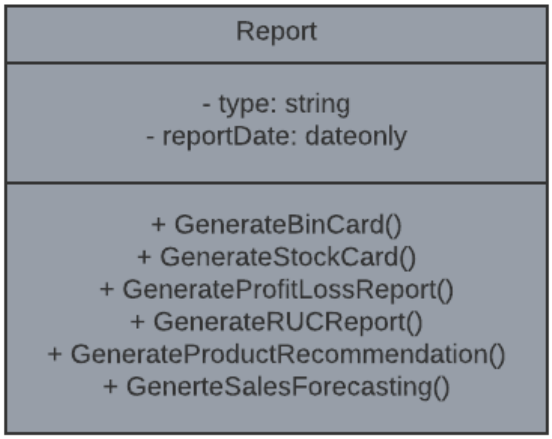

Figure 55. Report Class Interface

Attribute Name	Data Type	Description
Type	String	Contains the type of report it is.
ReportDate	dateonly	Contains the date when the report is generated.

Table 44. Class Interface for Report

5. Implementation Documentation

5.1. Introduction

The main code flows of the entire system are explained in this portion of our document. We have not included every piece of code; instead, we will concentrate on the system's essential features.

When implementing the system, the code is written using various languages, tools and frameworks. The user interface is coded using the React Version 18.2.0 framework. Since it is a web app, HTML and CSS are utilised to make the UI come to life. For the API, REST APIswith .NET and C# is used. Since the system is developed with both REST API architecture and MVC architecture, ASP.NET Core 6 is also used in the coding process. For the database, MSSQL has been utilised to create the database using SQL.

Different IDEs were used to write the codes. Visual Studio 2022 Community is very helpful when implementing the API and the database. Visual Studio Code was used for writing the HTML and CSS using the necessary framework.

- Programming Languages
    - ➔ C# Version 10
    - ➔ JavaScript ES2015
- Database
    - ➔ MSSQL
    - ➔ Microsoft Identity Authentication
- Frameworks
    - ➔ React Version 18.2.0
    - ➔ ASP.NET Core 6, REST APIs with .NET and C#
- IDEs
    - ➔ Visual Studio 2022 Community
    - ➔ Visual Studio Code

5.2. User Interface

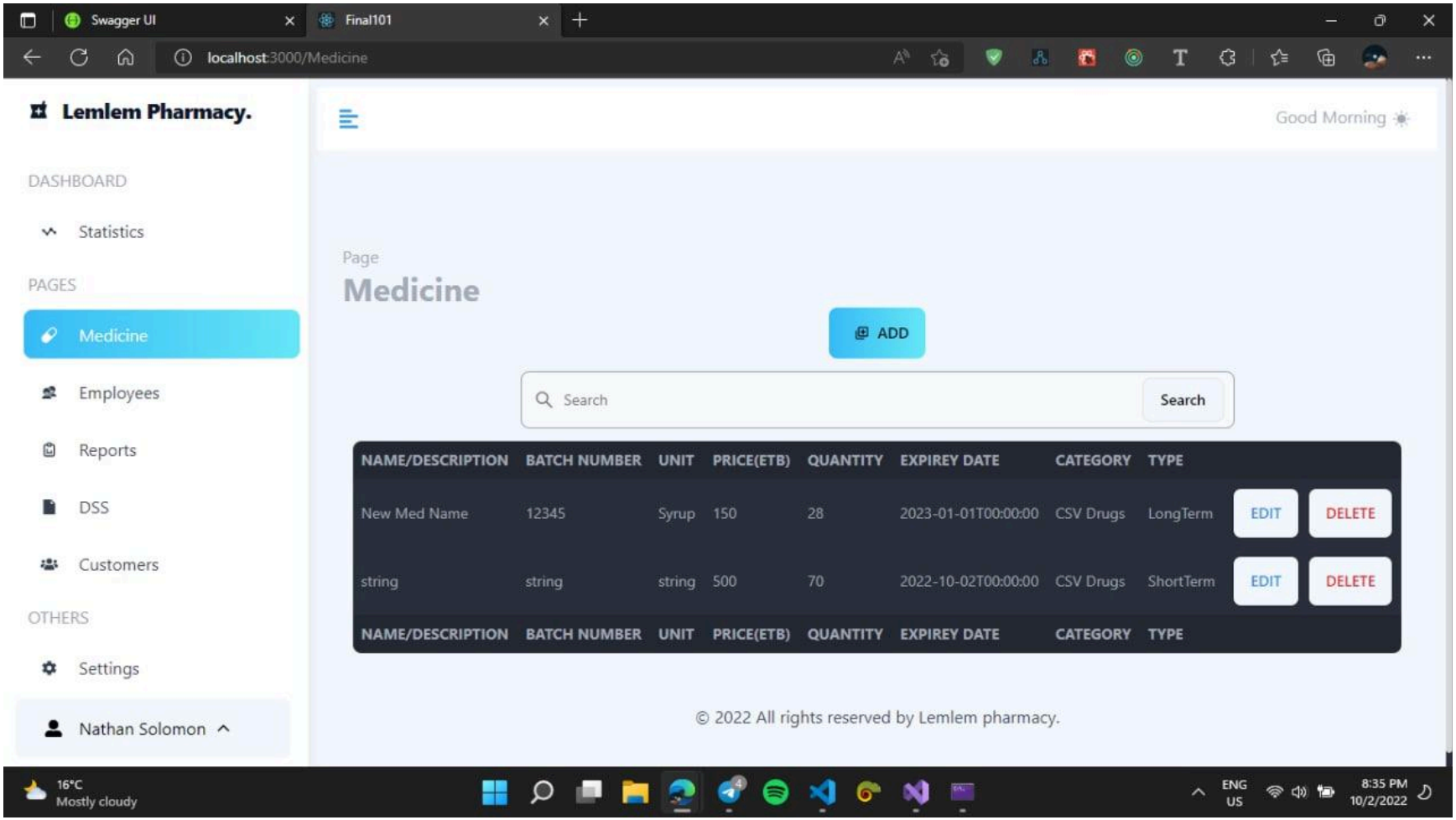

Figure 56. User Interface for Homepage in Light Mode

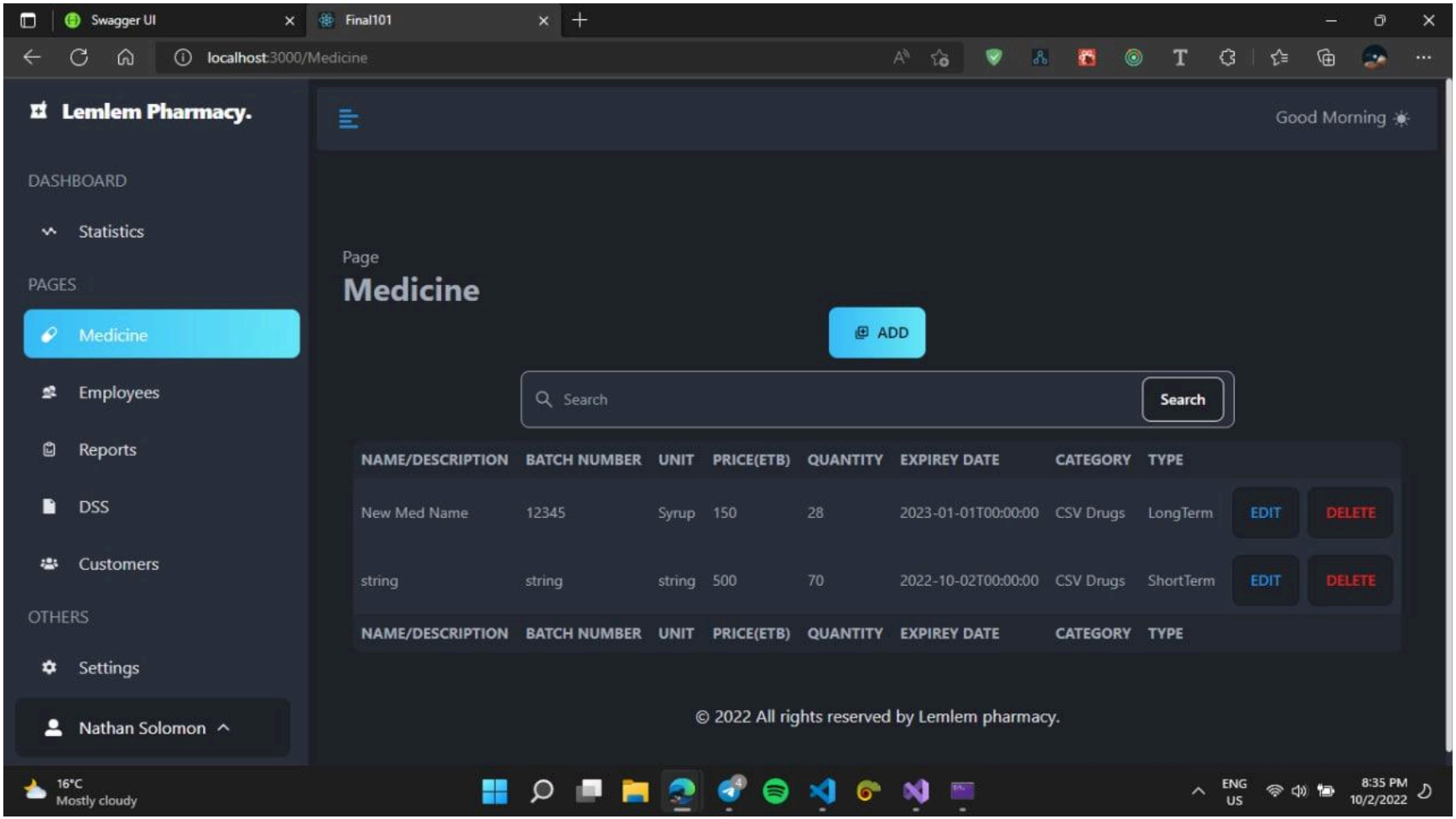

Figure 57. User Interface for Homepage in Dark Mode

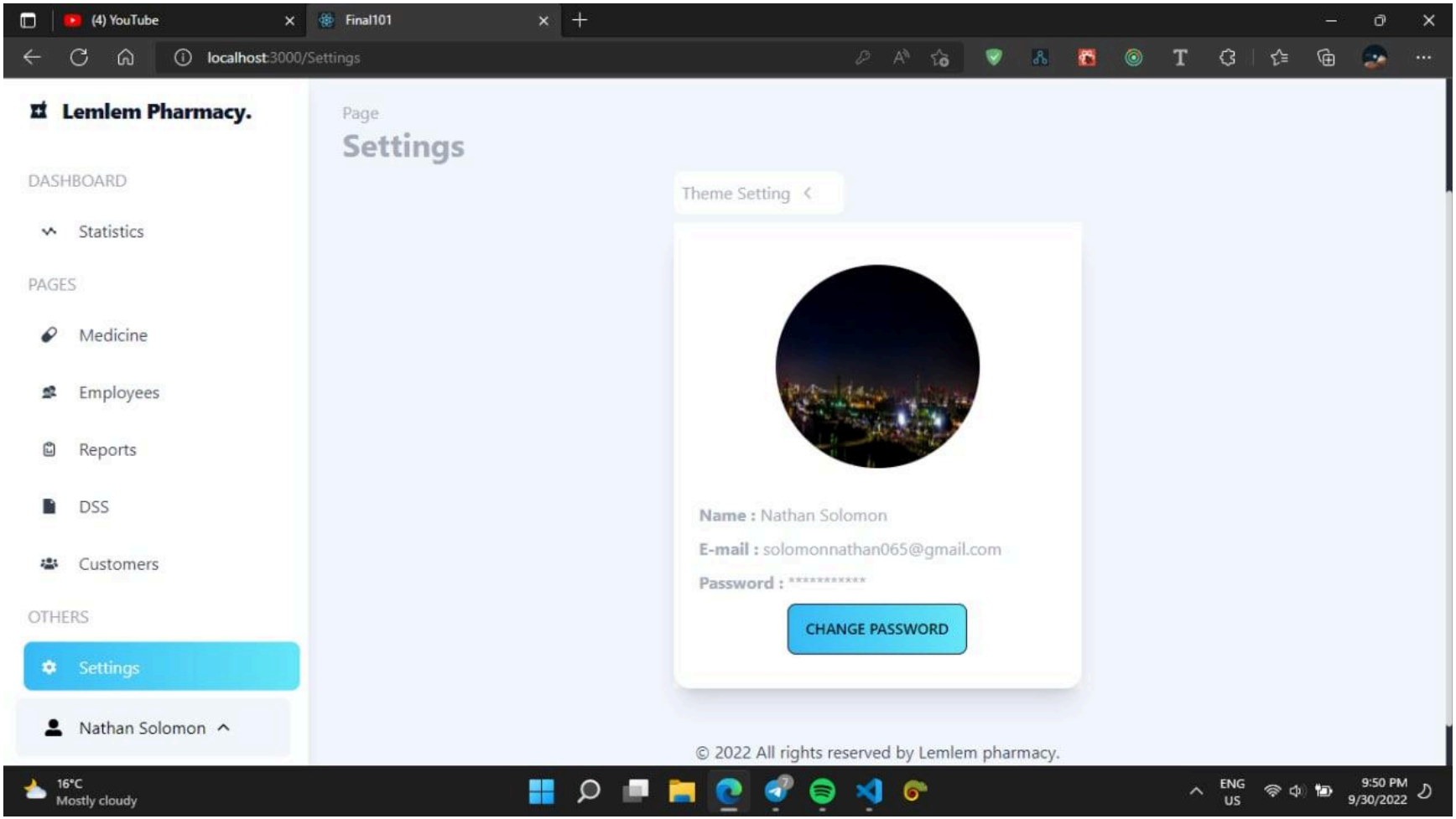

Figure 58. User Interface for Account Settings Page in Light Mode

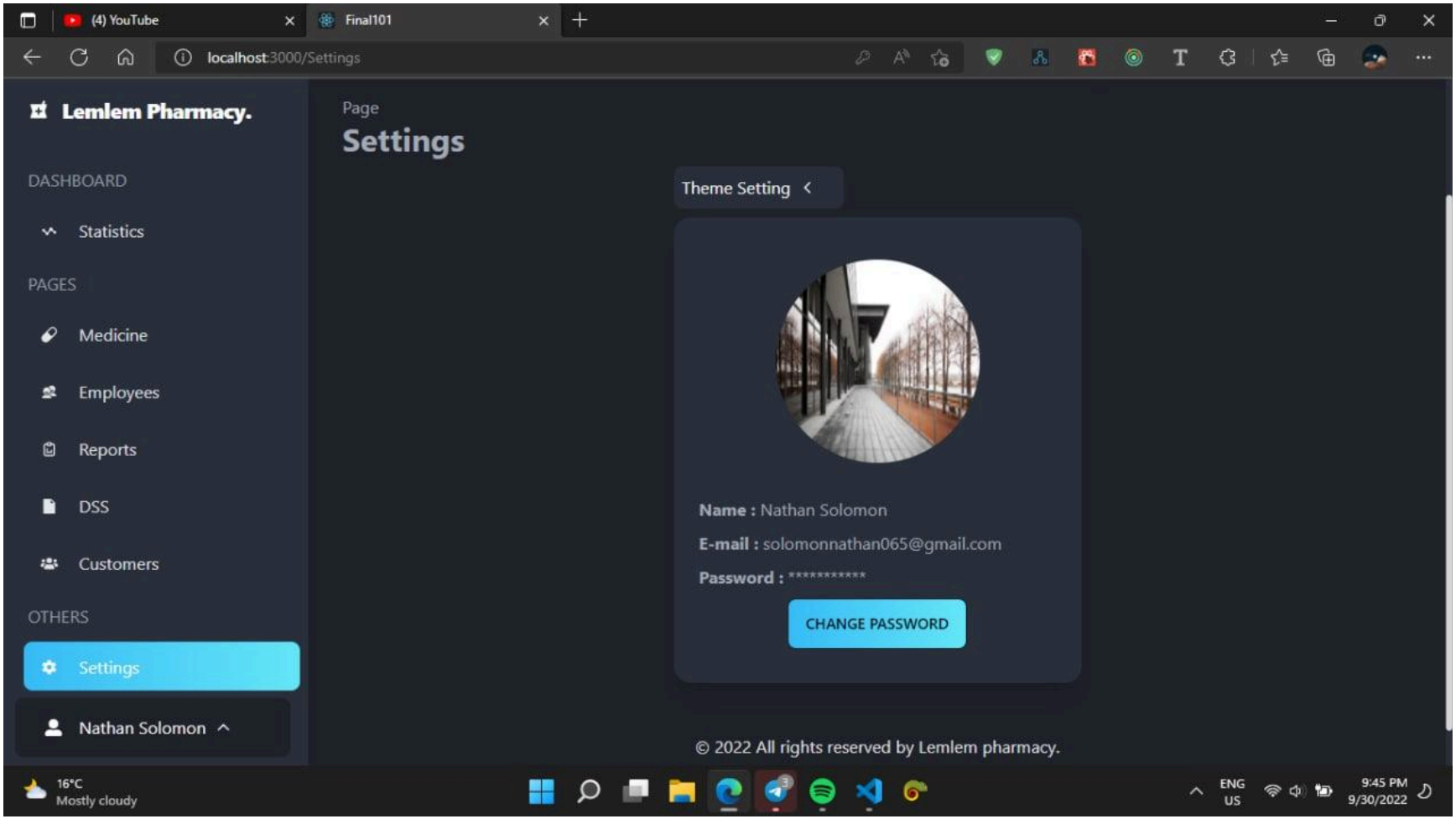

Figure 59. User Interface for Account Settings Page in Dark Mode

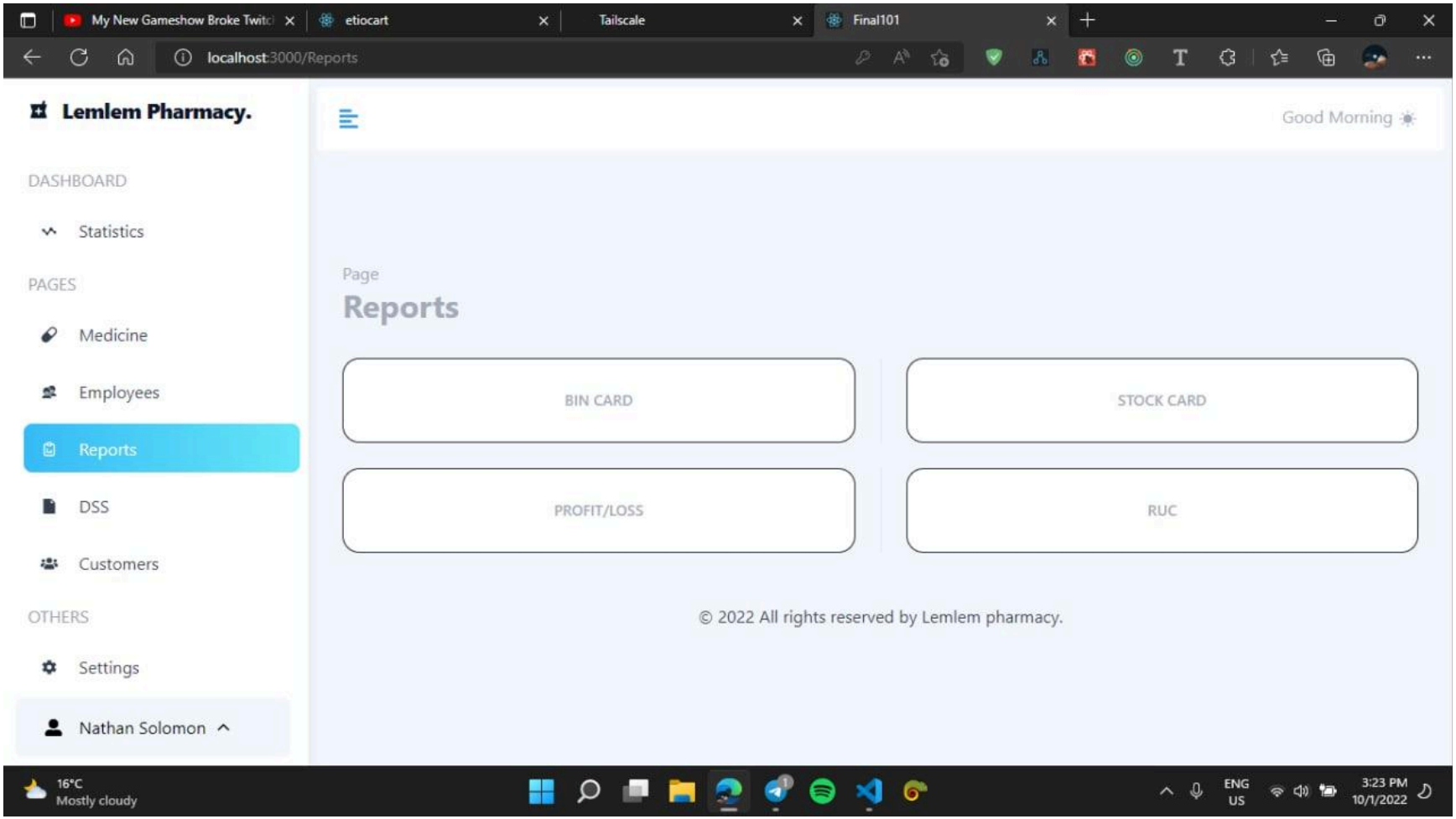

Figure 60. User Interface for Reports Page in Light Mode

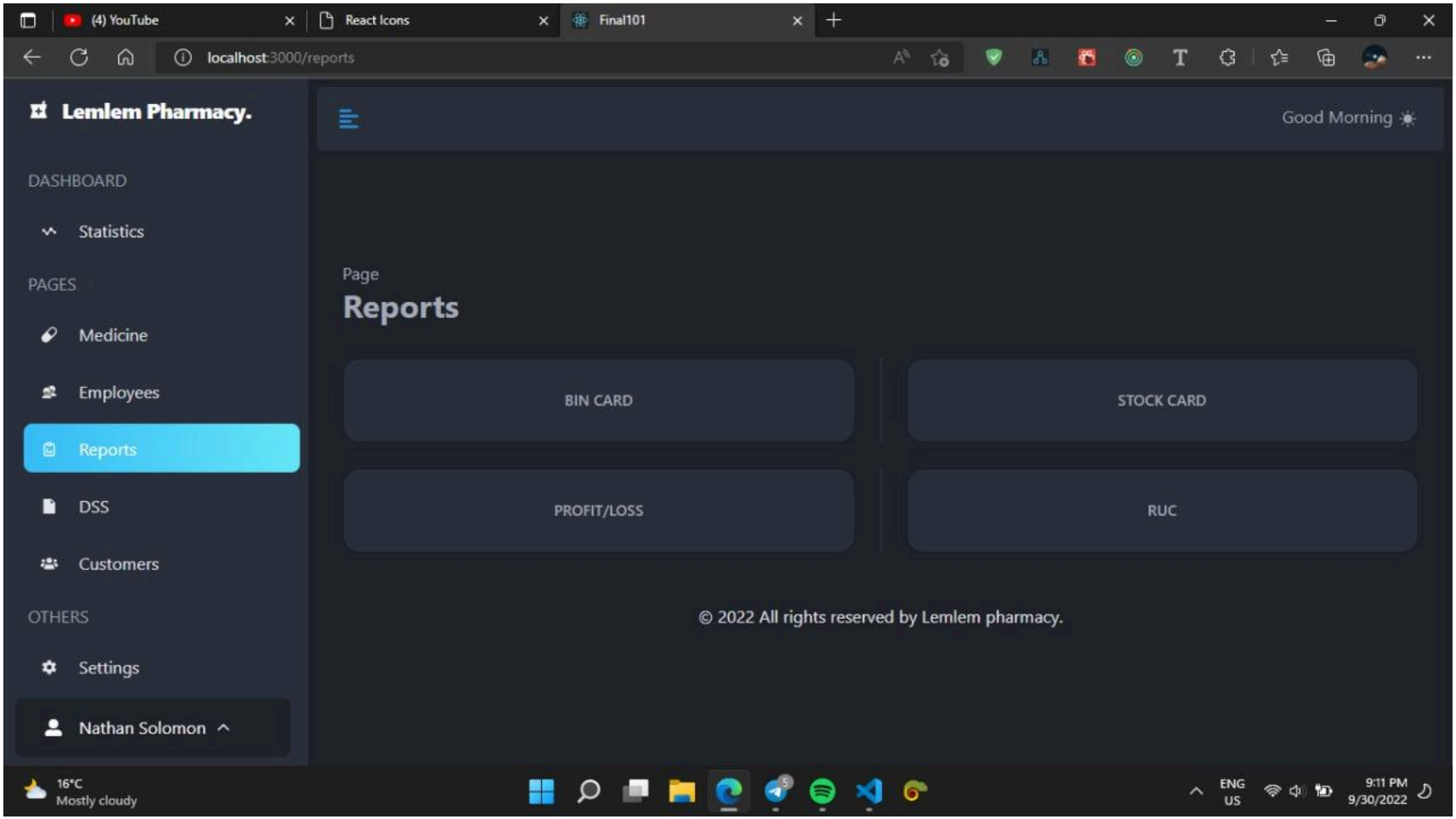

Figure 61. User Interface for Reports Page in Dark Mode

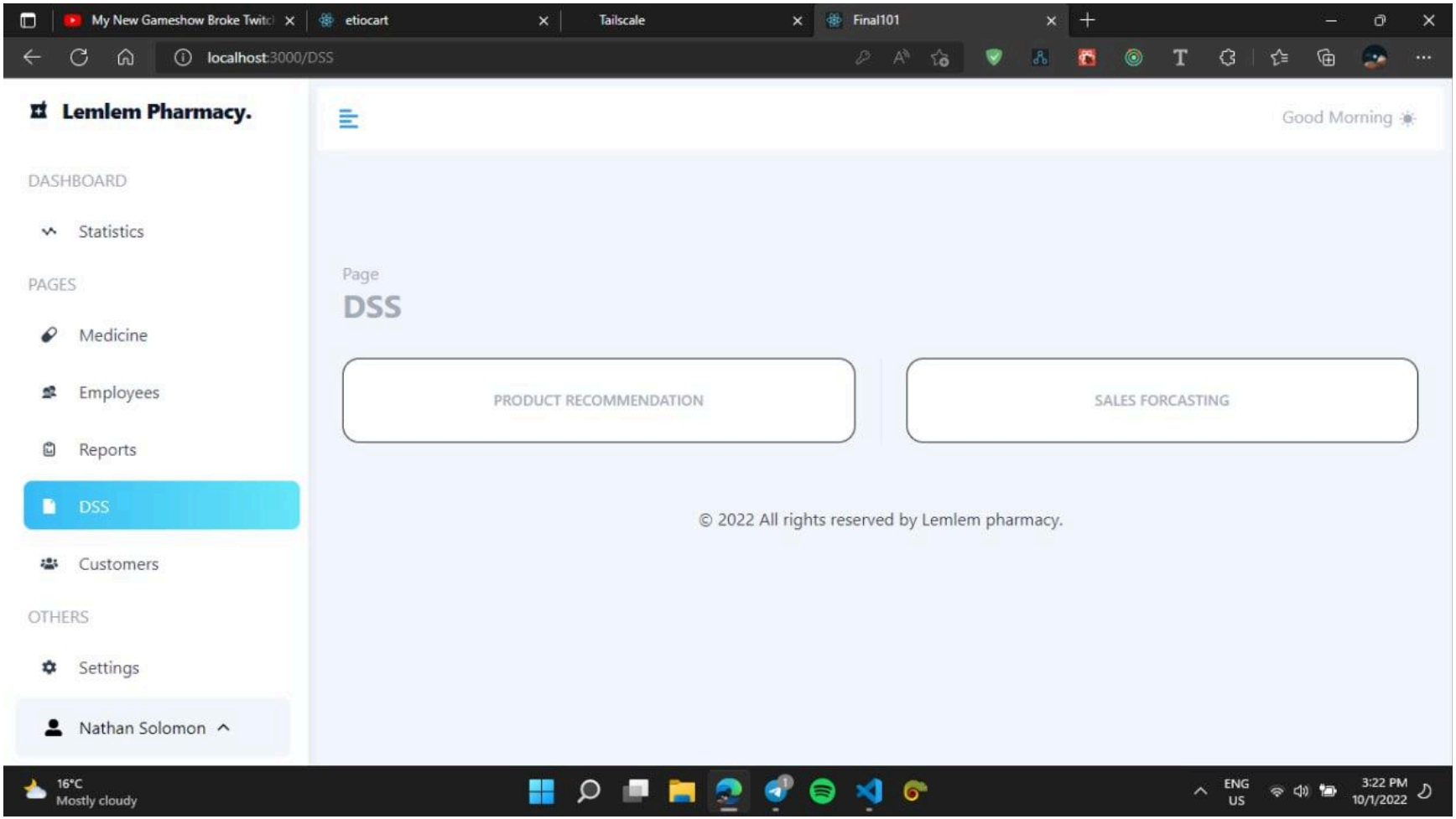

Figure 62. User Interface for DSS Page in Light Mode

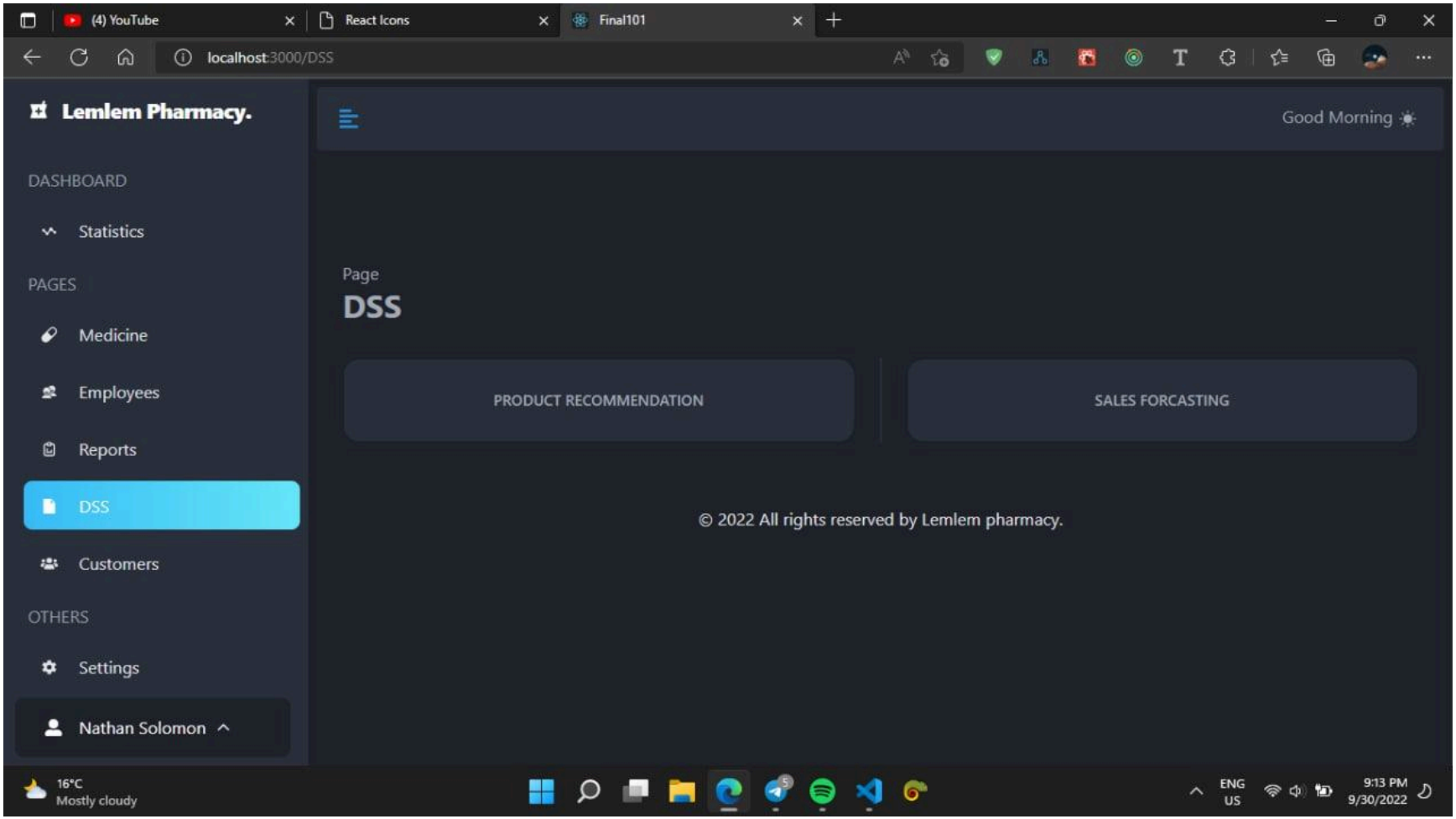

Figure 63. User Interface for DSS Page in Dark Mode

5.3. Core Implementation Source Codes

Medicine Controller

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using LemlemPharmacy.Data;
using LemlemPharmacy.DTOs;
using Microsoft.AspNetCore.Cors;

namespace LemlemPharmacy.Controllers
{
        [Route("api/[controller]")]
        [EnableCors]
  [ApiController]
  public class MedicinesController : ControllerBase
        {
    private readonly LemlemPharmacyContext _context;

    public MedicinesController(LemlemPharmacyContext context)
    {
      _context = context;
    }

                // GET: api/Medicines/5
                [HttpGet("id/{id}")]
                public async Task<ActionResult<MedicineDTO>> GetMedicine(Guid id)
                {
                        try
                        {
                                var medicine = await _context.Medicine.FindAsync(id);

                                if (medicine == null)
                                {
                                return NotFound();
                                }

                                return new MedicineDTO(medicine);
                        }
                        catch(InvalidOperationException e)
                        {
                                return BadRequest("Ere tailscale abra" + e.Message);
                        }
                }

                // GET : api/Medicine/all
                [HttpGet("all")]
    public async Task<ActionResult<IEnumerable<MedicineDTO>>> GetMedicineSP()
```

```
    {
                        var result = await
_context.Medicine.FromSqlRaw($"SpSelectAllMedicine").ToListAsync();

                        if (result == null) return NotFound();

                        var medicineDTOs = new List<MedicineDTO>();
                        foreach (var item in result)
                                medicineDTOs.Add(new MedicineDTO(item));

                        return medicineDTOs;
    }

    // GET : api/Medicine/batchNo/123451235
    [HttpGet("batchNo/{batchNo}")]
    public async Task<ActionResult<IEnumerable<MedicineDTO>>> GetMedicineByBatchNo(string
batchNo)
    {
                        var result = await _context.Medicine.FromSqlRaw($"SpSelectByBatchNo
'{batchNo}'").ToListAsync();

                        if (result == null) return NotFound();

                        var medicine = new List<MedicineDTO>();
                        foreach (var item in result)
                                medicine.Add(new MedicineDTO(item));

                        return medicine;
                }

                // GET : api/Medicine/category/123451235
                [HttpGet("category/{category}")]
                public async Task<ActionResult<IEnumerable<MedicineDTO>>>
GetMedicineByCategory(string category)
                {
                        var result = await _context.Medicine.FromSqlRaw($"SpSelectByCategory
'{category}'").ToListAsync();

                        if (result == null) return NotFound();

        var medicine = new List<MedicineDTO>();
        foreach (var item in result)
           medicine.Add(new MedicineDTO(item));

                        return medicine;
                }

                // PUT: api/Medicines/5
```

```
		// To protect from overposting attacks, we use DTOs
		[HttpPut("updateDetails")]
		public async Task<ActionResult<IEnumerable<MedicineDTO>>>
UpdateMedicineWithoutQuantity([FromBody] UpdateMedicineWithoutQuantityDTO medicine)
		{
			string StoredProc = "EXEC SpUpdateMedicineWithoutQuantity " +
				"@id = '" + medicine.Id + "'," +
				"@batchNo = '" + medicine.BatchNo + "'," +
				"@expireDate = '" + medicine.ExpireDate + "'," +
				"@unit = '" + medicine.Unit + "'," +
				"@price  = " + medicine.Price + "," +
				"@description  = '" + medicine.Description + "'," +
				"@Category  = '" + medicine.Category + "'," +
				"@Type  = '" + medicine.Type + "'";

			try
			{
				var result = await
_context.Medicine.FromSqlRaw(StoredProc).ToListAsync();
				var medicineDTOs = new List<MedicineDTO>();
				foreach (var item in result)
					medicineDTOs.Add(new MedicineDTO(item));

				return medicineDTOs;
			}
			catch (Microsoft.Data.SqlClient.SqlException e)
			{
				return BadRequest(e.Message);
			}
			catch (Exception)
			{
				return BadRequest();
			}
		}

		// updates the quantity of both medicine and bincard record
		[HttpPut("updateQuantity")]
		public async Task<ActionResult<IEnumerable<MedicineDTO>>>
UpdateMedicineQuantity([FromBody] UpdateMedicineQuantityDTO medicine)
		{
			string StoredProc = "EXEC SpUpdateMedicineQuantity " +
				"@BatchNo = '" + medicine.BatchNo + "'," +
				"@Quantity  = " + medicine.Quantity + "," +
				"@Invoice  = '" + medicine.Invoice + "'," +
				"@DateReceived  = '" + medicine.DateReceived + "'";

			try
			{
```

```
                                if (medicine.DateReceived == null) medicine.DateReceived =
DateTime.Now;
                                var result = await
_context.Medicine.FromSqlRaw(StoredProc).ToListAsync();
                                var medicineDTOs = new List<MedicineDTO>();
                                foreach (var item in result)
                                        medicineDTOs.Add(new MedicineDTO(item));

                                return medicineDTOs;
                        }
                        catch (Microsoft.Data.SqlClient.SqlException e)
                        {
                                return BadRequest(e.Message);
                        }
                        catch (Exception)
                        {
                                return BadRequest();
                        }
                }

                [HttpPut("addQuantity")]
                public async Task<ActionResult<IEnumerable<MedicineDTO>>>
AddMedicineQuantity([FromBody] UpdateMedicineQuantityDTO medicine)
                {
                        string StoredProc = "EXEC SpAddMedicineQuantity " +
                                "@BatchNo = '" + medicine.BatchNo + "'," +
                                "@Quantity  = " + medicine.Quantity + "," +
                                "@Invoice  = '" + medicine.Invoice + "'," +
                                "@DateReceived  = '" + medicine.DateReceived + "'";

                        try
                        {
                                if (medicine.DateReceived == null) medicine.DateReceived =
DateTime.Now;
                                var result = await
_context.Medicine.FromSqlRaw(StoredProc).ToListAsync();
                                var medicineDTOs = new List<MedicineDTO>();
                                foreach (var item in result)
                                        medicineDTOs.Add(new MedicineDTO(item));

                                return medicineDTOs;
                        }
                        catch (Microsoft.Data.SqlClient.SqlException e)
                        {
                                return BadRequest(e.Message);
                        }
                        catch (Exception)
                        {
```

```
                    return BadRequest();
                }
            }

            [HttpPut("remove")]
            public async Task<ActionResult<IEnumerable<MedicineDTO>>>
RemoveMedicine([FromBody] RemoveMedicineDTO medicine)
            {
                string StoredProc = "EXEC SpRemoveMedicine " +
                    "@Id = '" + medicine.Id + "'," +
                    "@Quantity = " + medicine.Quantity + "," +
                    "@DateReceived  = '" + medicine.DateReceived + "'";

                try
                {
                    if (medicine.DateReceived == null) medicine.DateReceived =
DateTime.Now;
                    var result = await
_context.Medicine.FromSqlRaw(StoredProc).ToListAsync();
                    var medicineDTOs = new List<MedicineDTO>();
                    foreach (var item in result)
                        medicineDTOs.Add(new MedicineDTO(item));

                    return medicineDTOs;
                }
                catch (Microsoft.Data.SqlClient.SqlException e)
                {
                    return BadRequest(e.Message);
                }
                catch (Exception)
                {
                    return BadRequest();
                }
            }

            // POST: api/Medicines
            // To protect from overposting attacks, we use DTOs
            [HttpPost("add")]
    public async Task<ActionResult<IEnumerable<MedicineDTO>>>
PostMedicine([FromBody]AddMedicineDTO medicine)
            {
        string StoredProc = "EXEC SpAddMedicine " +
            "@BatchNo = '" + medicine.BatchNo + "'," +
            "@ExpireDate = '" + medicine.ExpireDate + "'," +
          "@Unit = '" + medicine.Unit + "'," +
            "@Quantity = " + medicine.Quantity + "," +
            "@Price  = " + medicine.Price + "," +
            "@Description  = '" + medicine.Description + "'," +
```

```
                "@Category  = '" + medicine.Category + "'," +
                "@Type  = '" + medicine.Type + "'," +
                "@Invoice  = '" + medicine.Invoice + "'," +
                "@DateReceived  = '" + medicine.DateReceived + "'";

                        try
                        {
                                if (medicine.DateReceived == null) medicine.DateReceived =
DateTime.Now;
          var result = await _context.Medicine.FromSqlRaw(StoredProc).ToListAsync();
          var medicineDTOs = new List<MedicineDTO>();
          foreach (var item in result)
                medicineDTOs.Add(new MedicineDTO(item));

          return medicineDTOs;
                        }
                        catch (Microsoft.Data.SqlClient.SqlException e)
                        {
                                return BadRequest(e.Message);
                        }
        catch (Exception)
        {
          return BadRequest();
        }
      }

                private bool MedicineExists(Guid id)
      {
        return _context.Medicine.Any(e => e.Id == id);
      }
         }
}
```

Medicine Model

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.Models
{
   public class Medicine
          {
      [Key]
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public Guid Id { get; set; }

      [Required]
      [StringLength(100, ErrorMessage = "BatchNo character length cannot exceed 100!")]
      public string BatchNo { get; set; } = string.Empty;
```

```
        [Required]
        [DataType(DataType.Date)]
        public DateTime ExpireDate { get; set; }

        [Required]
        [StringLength(64, ErrorMessage = "Unit character length cannot exceed 64!")]
        public string Unit { get; set; } = string.Empty;

        [Required]
        public int Quantity { get; set; }

        [Required]
        public float Price { get; set; }

        [Required]
        [StringLength(1024, ErrorMessage = "Description character length cannot exceed 1024!")]
        public string Description { get; set; } = string.Empty;

        [Required]
        [StringLength(64, ErrorMessage = "Category character length cannot exceed 64!")]
        public string Category { get; set; } = string.Empty;

        [Required]
        [StringLength(64, ErrorMessage = "Type character length cannot exceed 64!")]
        public string Type { get; set; } = string.Empty;

                public ICollection<SoldMedicine>? SoldMedicines { get; set; }
        public ICollection<BinCard>? BinCards { get; set; }

    }
}
```

Medicine Data Transfer Object

```
using LemlemPharmacy.Models;
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.DTOs
{
    public class MedicineDTO
    {
        public Guid Id { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "BatchNo character length cannot exceed 100!")]
        public string BatchNo { get; set; } = string.Empty;

        [Required]
        [DataType(DataType.Date)]
        public DateTime ExpireDate { get; set; }
```

```
        [Required]
        [StringLength(64, ErrorMessage = "Unit character length cannot exceed 64!")]
        public string Unit { get; set; } = string.Empty;

        [Required]
        public int Quantity { get; set; }

        [Required]
        public float Price { get; set; }

        [Required]
        [StringLength(1024, ErrorMessage = "Description character length cannot exceed
1024!")]
        public string Description { get; set; } = string.Empty;

        [Required]
        [StringLength(64, ErrorMessage = "Category character length cannot exceed 64!")]
        public string Category { get; set; } = string.Empty;

        [Required]
        [StringLength(64, ErrorMessage = "Type character length cannot exceed 64!")]
        public string Type { get; set; } = string.Empty;


        public MedicineDTO()
        {

        }

        public MedicineDTO(Guid id, string batchNo, DateTime expireDate, string unit, int
quantity, float price, string description, string category, string type)
        {
                Id = id;
                BatchNo = batchNo;
                ExpireDate = expireDate;
                Unit = unit;
                Quantity = quantity;
                Price = price;
                Description = description;
                Category = category;
                Type = type;
        }

        public MedicineDTO(Medicine medicine)
        {
                Id = medicine.Id;
                BatchNo = medicine.BatchNo;
```

```
                    ExpireDate = medicine.ExpireDate;
                    Unit = medicine.Unit;
                    Quantity = medicine.Quantity;
                    Price = medicine.Price;
                    Description = medicine.Description;
                    Category = medicine.Category;
                    Type = medicine.Type;
                }
        }
}
```

Add Medicine Data Transfer Object

```
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.DTOs
{
        public class AddMedicineDTO
        {
                [Required]
                [StringLength(100, ErrorMessage = "BatchNo character length cannot exceed 100!")]
                public string BatchNo { get; set; } = string.Empty;

                [Required]
                [DataType(DataType.Date)]
                public DateTime ExpireDate { get; set; }

                [Required]
                [StringLength(64, ErrorMessage = "Unit character length cannot exceed 64!")]
                public string Unit { get; set; } = string.Empty;

                [Required]
                public int Quantity { get; set; }

                [Required]
                public float Price { get; set; }

                [Required]
                [StringLength(1024, ErrorMessage = "Description character length cannot exceed
1024!")]
                public string Description { get; set; } = string.Empty;

                [Required]
                [StringLength(64, ErrorMessage = "Category character length cannot exceed 64!")]
                public string Category { get; set; } = string.Empty;

                [Required]
                [StringLength(64, ErrorMessage = "Type character length cannot exceed 64!")]
                public string Type { get; set; } = string.Empty;
```

```
                public string Invoice { get; set; } = string.Empty;

                [Required]
                [DataType(DataType.Date)]
                public DateTime? DateReceived { get; set; }
        }
}
```

Sell Medicine Data Transfer Object

```
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.Interfaces
{
   public class SellMedicineDTO
        {
      [Required]
      public string PharmacistId { get; set; } = string.Empty;

      public string CustomerPhone { get; set; } = string.Empty;

      [Required]
      public Guid MedicineId { get; set; }

      [Required]
      public int Quantity { get; set; }

      [Required]
      [DataType(DataType.Date)]
      public DateTime? SellingDate { get; set; }
        }
}
```

Update Medicine Quantity Data Transfer Object

```
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.DTOs
{
        public class UpdateMedicineQuantityDTO
        {
                [Required]
                [StringLength(100, ErrorMessage = "BatchNo character length cannot exceed 100!")]
                public string BatchNo { get; set; } = string.Empty;

                [Required]
                public int Quantity { get; set; }

                [Required]
                public string Invoice { get; set; } = string.Empty;

                [Required]
```

```
        [DataType(DataType.Date)]
        public DateTime? DateReceived { get; set; }
    }
}
```

Update Medicine without Quantity Data Transfer Object

```
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.DTOs
{
    public class UpdateMedicineQuantityDTO
    {
        [Required]
        [StringLength(100, ErrorMessage = "BatchNo character length cannot exceed 100!")]
        public string BatchNo { get; set; } = string.Empty;

        [Required]
        public int Quantity { get; set; }

        [Required]
        public string Invoice { get; set; } = string.Empty;

        [Required]
        [DataType(DataType.Date)]
        public DateTime? DateReceived { get; set; }
    }
}
```

Bin Card Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using LemlemPharmacy.Data;
using LemlemPharmacy.Models;
using LemlemPharmacy.DTOs;

namespace LemlemPharmacy.Controllers
{
    [Route("api/[controller]")]
  [ApiController]
  public class BinCardsController : ControllerBase
    {
    private readonly LemlemPharmacyContext _context;

    public BinCardsController(LemlemPharmacyContext context)
    {
```

```
        _context = context;
    }

    // GET: api/BinCards
    [HttpGet]
    public async Task<ActionResult<IEnumerable<BinCardDTO>>> GetBinCard()
    {
                    var result = await _context.BinCard.ToListAsync();
                    var binCardDTOs = new List<BinCardDTO>();
                    foreach (var item in result)
                        binCardDTOs.Add(new BinCardDTO(item));

                    return binCardDTOs;
                }

    // GET: api/BinCards/5
    [HttpGet("{id}")]
    public async Task<ActionResult<BinCardDTO>> GetBinCard(Guid id)
    {
        var binCard = await _context.BinCard.FindAsync(id);


        if (binCard == null)
        {
            return NotFound();
        }

        return new BinCardDTO(binCard);
    }

                [HttpGet("batchNo/{batchNo}")]
                public async Task<ActionResult<IEnumerable<BinCardDTO>>>
GetBinCardByBatchNo(string batchNo)
                {
                    var result = await _context.BinCard.FromSqlRaw($"SpSelectBinCardByBatchNo
'{batchNo}'").ToListAsync();

                    if (result == null) return NotFound();

                    var medicine = new List<BinCardDTO>();
                    foreach (var item in result)
                        medicine.Add(new BinCardDTO(item));

                    return medicine;
                }
        }
}
```

Bin Card Model

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.Models
{
   public class BinCard
         {
      [Key]
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public Guid Id { get; set; }

      [Required]
      public string Invoice { get; set; } = string.Empty;

      [Required]
      public string BatchNo { get; set; } = string.Empty;

      [Required]
      [DataType(DataType.Date)]
      public DateTime DateReceived { get; set; }

      [Required]
      public int AmountRecived { get; set; }

      [Required]
      public int Damaged { get; set; }


      public Medicine? Medicine { get; set; }
         }
}
```

Bin Card Data Transfer Object

```
using LemlemPharmacy.Models;
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.DTOs
{
         public class BinCardDTO
         {
                  public Guid Id { get; set; }

                  [Required]
                  public string Invoice { get; set; } = string.Empty;

                  [Required]
                  public string BatchNo { get; set; } = string.Empty;

                  [Required]
```

```
                [DataType(DataType.Date)]
                public DateTime DateReceived { get; set; }

                [Required]
                public int AmountRecived { get; set; }

                [Required]
                public int Damaged { get; set; }

                public BinCardDTO()
                {

                }

                public BinCardDTO(Guid id, string invoice, string batchNo, DateTime dateReceived, int
amountRecived, int damaged)
                {
                        Id = id;
                        Invoice = invoice;
                        BatchNo = batchNo;
                        DateReceived = dateReceived;
                        AmountRecived = amountRecived;
                        Damaged = damaged;
                }

                public BinCardDTO(BinCard binCard)
                {
                        Id = binCard.Id;
                        Invoice = binCard.Invoice;
                        BatchNo = binCard.BatchNo;
                        DateReceived = binCard.DateReceived;
                        AmountRecived = binCard.AmountRecived;
                        Damaged = binCard.Damaged;
                }
        }
}
```

Customer Controller

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using LemlemPharmacy.Data;
using LemlemPharmacy.DTOs;
using System.Text.RegularExpressions;

namespace LemlemPharmacy.Controllers
{
        [Route("api/[controller]")]
        [ApiController]
        public class CustomersController : ControllerBase
```

```
	{
		private readonly LemlemPharmacyContext _context;
		private readonly string pattern =
@"(\+\s*2\s*5\s*1\s*9\s*(([0-9]\s*){8}\s*))|(0\s*9\s*(([0-9]\s*){8}))";

		public CustomersController(LemlemPharmacyContext context)
		{
			_context = context;
		}

		// GET: api/Customers
		[HttpGet]
		public async Task<ActionResult<IEnumerable<CustomerDTO>>> GetCustomer()
		{
			var result = await _context.Customer.ToListAsync();
			var customerDTOs = new List<CustomerDTO>();
			foreach (var item in result)
				customerDTOs.Add(new CustomerDTO(item));

			return customerDTOs;
		}

		// GET: api/Customers/5
		[HttpGet("{id}")]
		public async Task<ActionResult<CustomerDTO>> GetCustomer(Guid id)
		{
			var customer = await _context.Customer.FindAsync(id);

			if (customer == null)
			{
				return NotFound();
			}

			return new CustomerDTO(customer);
		}

		// PUT: api/Customers/
		// To protect from overposting attacks, we use DTOs
		[HttpPut("updateCustomer")]
		public async Task<ActionResult<IEnumerable<CustomerDTO>>>
UpdateMedicineWithoutQuantity([FromBody] CustomerDTO customer)
		{
			string StoredProc = "EXEC SpUpdateCustomer " +
				"@Id = '" + customer.Id + "'," +
				"@Name = '" + customer.Name + "'," +
				"@PhoneNo  = '" + customer.PhoneNo + "'";

			try
```

```
                {
                        if (Regex.IsMatch(customer.PhoneNo, pattern))
                        {
                        var result = await
_context.Customer.FromSqlRaw(StoredProc).ToListAsync();
                        var customerDTOs = new List<CustomerDTO>();
                        foreach (var item in result)
                                customerDTOs.Add(new CustomerDTO(item));

                        return customerDTOs;
                        }
                        else return BadRequest(new Response()
                        {
                        Status = "Error",
                        Message = "Phone number not in the right format. Example: +251 91
234 5678 +251912345678"
                        });
                }
                catch (Microsoft.Data.SqlClient.SqlException e)
                {
                        return BadRequest(e.Message);
                }
                catch (Exception e)
                {
                        return BadRequest(e.Message);
                }
        }

        // POST: api/Customers
        // To protect from overposting attacks, we use DTOs
        [HttpPost("add")]
        public async Task<ActionResult<IEnumerable<CustomerDTO>>>
AddCustomer([FromBody] AddCustomerDTO customer)
        {
                string StoredProc = "EXEC SpAddCustomer " +
                        "@Name = '" + customer.Name + "'," +
                        "@PhoneNo  = '" + customer.PhoneNo + "'";

                try
                {
                        if (Regex.IsMatch(customer.PhoneNo, pattern))
                        {
                        var result = await
_context.Customer.FromSqlRaw(StoredProc).ToListAsync();
                        var customerDTOs = new List<CustomerDTO>();
                        foreach (var item in result)
                                customerDTOs.Add(new CustomerDTO(item));
```

```
                    return customerDTOs;
                    }
                    else return BadRequest(new Response()
                    {
                    Status = "Error",
                    Message = "Phone number not in the right format. Example: +251 91
234 5678 +251912345678"
                    });
                }
                catch (Microsoft.Data.SqlClient.SqlException e)
                {
                    return BadRequest(e.Message);
                }
                catch (Exception e)
                {
                    return BadRequest(e.Message);
                }
            }

            // DELETE: api/Customers/5
            [HttpDelete("{id}")]
            public async Task<IActionResult> DeleteCustomer(Guid id)
            {
                var customer = await _context.Customer.FindAsync(id);
                if (customer == null)
                {
                    return NotFound(new Response()
                    {
                    Status = "Error",
                    Message = "Customer not found."
                    });
                }

                try
                {
                    _context.Customer.Remove(customer);
                    await _context.SaveChangesAsync();
                    return NoContent();
                }
                catch (Exception e)
                {
                    return BadRequest(new Response()
                    {
                    Status = "Error",
                    Message = e.Message
                    });
                }
            }
```

```
        }
}
```

Customer Model

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using System.Configuration;

namespace LemlemPharmacy.Models
{
   public class Customer
        {
      [Key]
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public Guid Id { get; set; }

      [Required]
      public string Name { get; set; } = string.Empty;

      [Required]
      [RegexStringValidator(@"^\\+(?:[0-9]●?){6,14}[0-9]$")]
      public string PhoneNo { get; set; } = string.Empty;


      public ICollection<SoldMedicine>? SoldMedicines { get; set; }
        }
}
```

Medicine Track Record

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace LemlemPharmacy.Models
{
        public class MedicineTrack
        {
                [Key]
                [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
                public Guid Id { get; set; }

                [Required]
                public string BatchNo { get; set; } = string.Empty;

                [Required]
                public string Invoice { get; set; } = string.Empty;
        }
}
```

Sold Medicine Controller

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using LemlemPharmacy.Data;
using LemlemPharmacy.Models;
using System.Reflection.Metadata.Ecma335;
using LemlemPharmacy.Interfaces;
using LemlemPharmacy.DTOs;

namespace LemlemPharmacy.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class SoldMedicinesController : ControllerBase
          {
      private readonly LemlemPharmacyContext _context;

      public SoldMedicinesController(LemlemPharmacyContext context)
      {
         _context = context;
      }

      // POST: api/SoldMedicines/Sell
      [HttpPost]
      [Route("Sell")]
      public async Task<ActionResult<IEnumerable<SoldMedicineDTO>>> SellMedicine([FromBody]
SellMedicineDTO soldMedicine)
      {
         string StoredProc = "EXEC SpSellMedicine " +
                     "@PharmacistId = '" + soldMedicine.PharmacistId + "'," +
                     "@MedicineId = '" + soldMedicine.MedicineId + "'," +
                     "@Quantity = " + soldMedicine.Quantity + "," +
                     "@SellingDate = '" + soldMedicine.SellingDate + "'," +
                     "@CustomerPhone  = '" + soldMedicine.CustomerPhone + "'";

         try
         {
            if (soldMedicine.SellingDate == null) soldMedicine.SellingDate = DateTime.Now;
                                    var result = await
_context.SoldMedicine.FromSqlRaw(StoredProc).ToListAsync();
                                    var soldMedicineDTOs = new List<SoldMedicineDTO>();
                                    foreach (var item in result)
                                             soldMedicineDTOs.Add(new SoldMedicineDTO(item));

                                    return soldMedicineDTOs;
                           }
```

```
    catch (Microsoft.Data.SqlClient.SqlException e)
    {
        if (e.Message.Contains("FK_SoldMedicine_Customer_CustomerPhone"))
                return BadRequest("Customer doesn't exist");
        if (e.Message.Contains("FK_SoldMedicine_AspNetUsers_PharmacistId"))
                return BadRequest("Pharmacist doesn't exist!");
        return BadRequest(e.Message);
    }
    catch(Exception)
    {
                                    return BadRequest();
                            }
}

// GET: api/SoldMedicines
[HttpGet]
public async Task<ActionResult<IEnumerable<SoldMedicineDTO>>> GetSoldMedicine()
{
    var result = await _context.SoldMedicine.ToListAsync();
    var soldMedicine = new List<SoldMedicineDTO>();
    foreach (var item in result)
        soldMedicine.Add(new SoldMedicineDTO(item));
    return soldMedicine;
}

// GET: api/SoldMedicines/5
[HttpGet("{id}")]
public async Task<ActionResult<SoldMedicineDTO>> GetSoldMedicine(Guid id)
{
    var result = await _context.SoldMedicine.FindAsync(id);

                            if (result == null)
    {
        return NotFound();
    }

    return new SoldMedicineDTO(result);
}

// PUT: api/SoldMedicines/5
// To protect from overposting attacks, we use DTOs
[HttpPut("{id}")]
public async Task<IActionResult> PutSoldMedicine(Guid id, SoldMedicine soldMedicine)
{
    if (id != soldMedicine.TransactionId)
    {
        return BadRequest();
    }
```

```
        _context.Entry(soldMedicine).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!SoldMedicineExists(id))
            {
                    return NotFound();
            }
            else
            {
                    throw;
            }
        }

        return NoContent();
    }

    private bool SoldMedicineExists(Guid id)
    {
        return _context.SoldMedicine.Any(e => e.TransactionId == id);
    }
      }
}
```

Sold Medicine Model

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace LemlemPharmacy.Models
{
        public class SoldMedicine
        {
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public Guid TransactionId { get; set; }

    // From Username
    [Required]
    public string PharmacistId { get; set; } = string.Empty;

    public string CustomerPhone { get; set; } = string.Empty;

    [Required]
    public Guid MedicineId { get; set; }
```

```
        [Required]
        public int Quantity { get; set; }

        [Required]
        public float SellingPrice { get; set; }

        [Required]
        [DataType(DataType.Date)]
        public DateTime? SellingDate { get; set; }


        public Medicine? Medicine { get; set; }
        public Customer? Customer { get; set; }
              public ApplicationUser? ApplicationUser { get; set; }
            }
}
```

Database Context

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using LemlemPharmacy.Models;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace LemlemPharmacy.Data
{
   public class LemlemPharmacyContext : IdentityDbContext<ApplicationUser>
           {
      public LemlemPharmacyContext (DbContextOptions<LemlemPharmacyContext> options)
         : base(options)
      {
      }

      protected override void OnModelCreating(ModelBuilder builder)
      {
         base.OnModelCreating(builder);

         builder.Entity<BinCard>()
            .Property(i => i.Invoice)
                                    .IsRequired(false);

                              builder.Entity<BinCard>()
                                    .HasIndex(i => new { i.BatchNo,i.Id })
                                    .IsUnique();

                              builder.Entity<SoldMedicine>()
```

```
            .Property(s => s.CustomerPhone)
            .IsRequired(false);

        builder.Entity<Medicine>()
            .HasIndex(i => i.BatchNo)
            .IsUnique();

        builder.Entity<Customer>()
            .HasIndex(i => i.PhoneNo)
            .IsUnique();

        builder.Entity<MedicineTrack>()
            .HasIndex(i => new {i.BatchNo, i.Invoice})
            .IsUnique();

        // every customer is not required to be registered on the soldMedicine table
        builder.Entity<SoldMedicine>()
            .Property(s => s.CustomerPhone)
            .IsRequired(false);

        builder.Entity<Medicine>()
            .HasCheckConstraint(
            "CK_Medicine_Category",
            "Category in ('Anti-Fungal', 'Anti-Allergy', 'Anti-Helmentic', 'Hormonal Drugs', 'ENT Drugs',
'NSAI', 'GIT', 'Anti-Respiratory', 'Narcotic and Anti-Psychotropic', 'Anti-Biotic', 'Vitamins and Minerals',
'CSV Drugs')");

        builder.Entity<Medicine>()
            .HasCheckConstraint(
            "CK_Medicine_Type",
            "Type in ('LongTerm', 'ShortTerm')");

        // Medicine-BinCard Relationship (One-to-Many)
        builder.Entity<BinCard>()
            .HasOne(s => s.Medicine)
            .WithMany(c => c.BinCards)
            .HasForeignKey(s => s.BatchNo)
            .HasPrincipalKey(c => c.BatchNo);

        // Customer-SoldMedicine Relationship (One-to-Many)
        builder.Entity<SoldMedicine>()
            .HasOne(s => s.Customer)
            .WithMany(c => c.SoldMedicines)
            .HasForeignKey(s => s.CustomerPhone)
            .HasPrincipalKey(c => c.PhoneNo)
            .OnDelete(DeleteBehavior.SetNull);

        // User-SoldMedicine Relationship (One-to-Many)
```

```
        builder.Entity<SoldMedicine>()
            .HasOne(s => s.ApplicationUser)
            .WithMany(c => c.SoldMedicines)
            .HasForeignKey(s => s.PharmacistId)
            .HasPrincipalKey(c => c.UserName);

                // Medicine-SoldMedicine Relationship (One-to-Many)
        builder.Entity<SoldMedicine>()
            .HasOne(s => s.Medicine)
            .WithMany(c => c.SoldMedicines)
            .HasForeignKey(s => s.MedicineId)
            .HasPrincipalKey(c => c.Id);

    }

    public DbSet<Medicine> Medicine { get; set; } = default!;
    public DbSet<Customer> Customer { get; set; } = default!;
    public DbSet<BinCard> BinCard { get; set; } = default!;
    public DbSet<SoldMedicine> SoldMedicine { get; set; } = default!;
        }
}
```

6. Conclusion

This project is intended as Partial Fulfilment of the Requirements for the B.Sc. Degree in Computer Science. The goal of this project is to create a pharmacy management system with artificial intelligence decision support that will enable any pharmacy to manage its daily operations and make decisions using the decision support it offers.

While preparing and compiling this project, we went through several aspects of the software design and development phases as a team, which were the foundations that enabled us to implement our proposed system. In this senior thesis, we were able to apply the programming, project management, database management, and software testing skills we learned from HiLCoE. We used tools like Lucidchart Diagrams to create the system's scenarios, use cases, sequences, and activities. We used Google Docs to create this document. Visual Studio Community 2022 and Visual Studio Code were used to create our codes. Database management and storage were both done using MSSQL.

We were able to identify numerous issues with the country's present pharmaceutical management system throughout the course of this project. As a result, we think that the approach we've suggested offers a simple fix for every issue the country is currently facing.

7. References

[1] Ethiopia Health Private Sector Assessment, October 2019

[2] Federal Democratic Republic of Ethiopia, Food Medicine and Health care Administration and control Proclamation 1112/2019. *Federal Negarit Gazette.* 2010;16:5157–5191.

[3] Bruegge, B., Dutoit. A. H., *Object-Oriented Software Engineering Using UML, Patterns and Java*, 3rd ed. Prentice Hall, 2010

[4] https://www.apriorit.com/dev-blog/699-requirement-elicitation

[5] Sommerville, I., *Software Engineering*, 10th ed. Harlow: Pearson, 2016

[6] FDRE Ministry of Health, "Ethiopian Hospital Services Transformation Guidelines, Chapter 10: Pharmacy Services", Version 01, August 2017

[7] Schach, Stephen R., *Object-Oriented and Classical Software Engineering,* 8th ed. McGraw Hill, 2011

[8] Difference Between Two-Tier And Three-Tier database architecture - GeeksforGeeks

[9] Richardson, L., Amundsen, *RESTful Web APIs*, 1st ed. O'Rielly, 2013

[10] Kung, David C., *Object-Oriented Software Engineering - An Agile Unified Methodology*, McGraw Hill, 2014

Annex

User Manual

Login

In order to log into your account, follow the process below.

- When you open the application, a page with fields reading “Username” and “Passwords” are displayed respectively.
- Enter your username on the field corresponding to the “Username” description.
- Then, enter your password on the field corresponding to the “Password” description and click on the Login button.
- If both username and password are provided correctly, the system will log you into your account and you will be redirected to your homepage.

NB: The homepage differs from user to user. For example: the manager will have the ability to see a more detailed page compared to the page that the pharmacist will see.

Search for a Medicine

In order to search for any medicine in the system, follow the process below.

- First, make sure you are logged into your account. The search functionality is only available when logged in.
- In the search bar on your homepage, enter the medicine you want to search for.
- Click on the search button.
- If the medicine is present, It will be available on the screen. If not, you will see a nothing found message.

Adding a medicine

In order to add a medicine into the inventory, follow the process below.

- First, make sure you are logged into your account.
- On your homepage, Click the add button.
- Fill in the necessary information (Batch Number, Expire Date, Unit, Quantity, Price, Description, Category, Type, Invoice, Date Received).

- Click add

NB: Note that only a user that has manager level permission.

Updating a medicine

- First, make sure you are logged into your account.
- On the homepage, search for the medicine you want to update.
- After finding the medicine, on the right side you will see an edit button.
- Click on the edit button.
- Do the necessary modification by filling out the form provided.
- Click on the Save button.

Deleting a medicine

- First, make sure you are logged into your account.
- On the homepage, search for the medicine you want to delete.
- After finding the medicine, on the right side there is a delete button.
- Click on the delete button.
- A confirmation message asking you if you are sure will appear.
- Click on Yes.

Adding an employee

- First, make sure you are logged into your account.
- On the navigation bar, click on the Employee button.
- On the Employee page, click on the add employee.
- Fill in the necessary information (UserName, Role, PhoneNo, Email) and click on the add button.

NB: Only users that have manager level permission can complete this task.

Adding a customer

- First, make sure you are logged into your account.
- On the navigation bar, click on the customer button.
- On the customer page, click on the add customer button.
- Fill in the necessary information (Name, Phone number) and click on the add button.

NB: Only users that have pharmacist level permission can complete this task.

Generating a report

- First, make sure you are logged into your account.
- On the navigation bar, click on the Report button. A reports page will be displayed.
- On the report page, there are four types of reports to choose from.
- Click one and follow the prompts.

DSS

- First, log in with an authorised account.
- On the navigation bar, click on the DSS button.
- On the DDS page, there are two options.
- Click on the one you want.
- The system will display a report based on your choice.

Hospital Name: _____

Product Name, Strength and Dosage Form: _____

Unit of Issue: _____ Maximum Stock Level: _____ Emergency Order Point: _____

Date	Doc. No. (Receiving or Issuing)	Received from or Issued t	Quantity				Batch No.	Expiry Date	Remark
			Received	Issued	Loss/Adj.	Balance			

Figure 64. Stock Card

Hospital Name: _____

Product Name, Strength and Dosage Form: _____

Unit of Issue: _____

Date	Doc. No. (Receiving or Issuing)	Received from or Issued t	Quantity				Batch No.	Expiry Date	Remark
			Received	Issued	Loss/Adj.	Balance			

Figure 65. Bin Card

Figure 66. RUC Report

Date: _____

To: _____

(Name of Health Centre or PFSA Hub)

Facility returning commodities:

Region:_____ Zone: _____ Woreda: _____ Kebele: _____

Item Description	Unit	Quantity Returned	Reason for Return/Non-Use

Sending Certification:

Completed by: _____ Signature: _____ Date: _____

Remarks:

Carrier Certification:

Carried by: _____ Signature: _____ Date: _____

Remarks:

Receiving Facility Certification:

Received by: _____ Signature: _____ Date: _____

Remarks: