Table Of Contents

Table Of Figures

Abstract

We discuss medical booking system and its social benefits to fulfill patients' needs by connecting them to the appropriate doctor and we discuss the technical requirements for booking doctors in the easiest way. You can also buy medicines from pharmacy in our mobile application or from website online providing two payment methods which are cash or using credit card; the presentation is not totally completed, but it aims to give an idea of the system-level issues to be considered for real applications. The technology in this area is rapidly developing, and without doubt we will evidence emergence of these applications in the coming years in the market.

Chapter 1: Introduction

1.1. Overview

Our project is a medical information system for hospital which helps patients in their medical needs such as booking doctors, buying medicines online and track orders to know shipping status and providing pneumonia prediction service for doctors.

1.2. Objectives

Project's objective is building a powerful system which provides services to patients to fulfill their requirements digitally such as booking doctors and buying medical supplies online using medical insurance; and provides services to admins such as managing users digitally without using hard copy papers by building ease-use dashboard.

- Now we can say that the most affected by the current system in hospitals are patients who find it difficult to seek medical advice, so they turn to search via the Internet. In a study conducted, it proved that websites and applications for examining symptoms are accurate about 34% of the time, while doctors, when given the same information, diagnosing the condition correctly 72% of the time.
- Doctors also, because their task is made difficult in the current system because there are no ways to facilitate the matter of meeting patients, for example, or knowing their medical history in asking each patient. Those related to the health system in Egypt, so by making it easier for them, they are more attracted to work in Egypt.

1.3. Purpose

- managing patients and their related information.
- Improving patients care by helping them in booking doctors easily and digitally.
- helping radiology doctors in detecting pneumonia using service.
- Helping doctors in managing their appointments.
- Helping admins in accessing users' information.
- Improving efficiency via taken care of processes automatically.
- Increasing data security & retrieve-ability.
- Accounting, laboratory, and pharmacy management.
- Buying medical supplies from pharmacy page online using electronic payment system allowing them to use medical insurance.
- Serving patients from multiple regions using multitenancy (Software as a Service).

1.4. Scope

1.4.1 Introduction:

The scope of the project is to develop a comprehensive Hospital Management System (HMS) that includes a pneumonia prediction service. The system aims to streamline and automate various hospital management tasks while providing an advanced prediction model to assist healthcare professionals in diagnosing and managing pneumonia cases.

1.4.2 Features:

- Patient Management: The system will allow hospital staff to efficiently manage patient records, including registration, medical history, and current symptoms. It will enable easy tracking of patient admissions, discharges, and transfers.
- Appointment Scheduling: A centralized appointment scheduling module will be implemented to facilitate the efficient allocation of resources and minimize waiting times. Patients can book appointments online.
- Pneumonia Prediction Service: The system will integrate a pneumonia prediction model based on machine learning algorithms. It will analyze patient symptoms, vital signs, and relevant medical data to assess the likelihood of pneumonia. The prediction service will provide healthcare professionals with insights to aid in early detection and treatment decisions.

- Electronic Health Records (EHR): The HMS will maintain comprehensive electronic health records for each patient, ensuring easy access to medical history, test results, prescribed medications, and treatment plans. This will enhance data accuracy, reduce paperwork, and improve overall patient care.
- Laboratory and Radiology Integration: The system will integrate with laboratory and radiology departments, enabling seamless communication of test orders, results, and interpretations. This integration will optimize the diagnostic process and allow physicians to make informed decisions based on accurate and timely information.
- Inventory and Pharmacy Management: The HMS will include modules to manage hospital inventory, including medical supplies, drugs, and equipment. It will track stock levels, automate reordering, and manage pharmacy operations efficiently.
- Billing and Insurance: The system will facilitate accurate billing and insurance claims processing, reducing errors and improving financial management. It will generate itemized bills, track payments, and integrate with insurance providers' systems for seamless claims submission.
- Reporting and Analytics: The HMS will provide comprehensive reporting and analytics capabilities, allowing hospital administrators to monitor key performance indicators, resource utilization, patient outcomes, and the effectiveness of the pneumonia prediction service. These insights will support data-driven decision-making and continuous improvement.
- Security and Privacy: The system will adhere to strict security protocols and comply with healthcare data privacy regulations such as HIPAA. Patient data will be encrypted, access controls will be implemented, and regular backups will be performed to ensure data integrity and confidentiality.
- Mobile Access: The HMS will have a mobile-friendly interface or a dedicated mobile app, while also allowing patients to conveniently order prescribed medications through the application. This feature enhances the efficiency and flexibility of healthcare delivery, providing a seamless process for patients to purchase their medicines.

1.5. General constraints

❖ Time constraints:

Our system is only used in the modern era due to our use of modern technologies such as artificial intelligence and due to the presence of computers in all hospitals now so they can use our system.

❖ Spatial constraints:

Since our system is a medical system, this system is used only in hospitals because it serves medical institutions. We also added multitenancy technology to our system, so according to the use of each hospital, the countries in which its branches are located are determined. The system can only be used by the user in these countries specified by the hospital.

❖ Operating requirements:

The user can use our system from three things, namely the mobile app, as it is not required except for the Android or iOS operating system. As for the Dashboard and Web App, the user can run them from any operating system only when it has a good and fast browser and a good internet connectin, then he can run it .

❖ Users Constraints:

Our system can be used by users who want to go to the hospital or buy medicines, as well as doctors, all hospital workers and those involved in the health field.

Chapter 2: Project "Planning and analysis"

2.1. Project planning (Feasibility Study)

In this section we will know everything about the project and study its aspects to understand it very well to start building the system.

A feasibility study is conducted to find out whether the proposed system is possible, affordable, and acceptable for organization. The financial, political, social and time constraints must be considered during this study.

- Possible: to build it with the given technology and resources
- Affordable: given the time and cost constraints of the organization
- Acceptable: for use by the eventual users of the system.

2.1.1. Technical Feasibility

The primary technical requirement includes the availability of a good version of operating system installed in the network. To develop programs, any good Integrated Development Environment is needed, which can be easily acquired after deciding. Reliability, access, power and data security are also available.

- Hardware Requirements:

  ➔ Computer Systems: 3 (Available)

  ➔ Processor: Core i3 Processor (minimum)

  ➔ RAM: Minimum 8 GB. (1 GB extra RAM is required to use Android emulator and Vs code)

  ➔ Disk Space: Using an SSD would be a wise decision, but 256GB SSD can be a good choice.

  ➔ Works on graphic card 4GB to 8GB

- Software Requirements:

  1. Web apps can be developed using a number of different alternative languages and IDEs.

     ➔ Back-End

     A. Xampp local host and Vs code “IDE”

     B. Php V 7.4 “language”

     ➔ Front-End

     A. HTML, CSS “tools”

     B. Local host and Vs code “IDE”

     C. Angular “frame work”

  2. Android or IOS apps can be developed using a number of different alternative languages and IDEs.

     ➔Java Development Kit (JDK) and Android studio “IDE”

➔Git.

➔Dart “language”

➔ Flutter “frame work”

AI feature:

➔Anaconda environment

2.1.2. Economic Feasibility

Whether the MediBooki is cost effective or not? The benefits in the form of reduced cost?

MediBooki is economically Feasible. As the hardware cost on the project is low. Similarly. it’s cost is also under the budget. Moreover, some of the technical requirements are already available and some can be obtained by using a reasonable amount and effort.

2.1.3. Operational Feasibility

MediBooki is operationally feasible. it provides the necessary information to the user as how to enter the information, how to register, selecting the interests, giving permissions to the apps. Some prior knowledge is required for the management to go through the various operations. But for the user basic knowledge of computers is enough.

2.2. Analysis and Limitation of existing system

- At the beginning of our study of the project, we found that the current manual medical system is difficult for the patient these days, so we decided to try to make it easier for patients and doctors as well by making an electronic system that would be an intermediary between them and also between them and the hospital. **We found the following:**
- There are many medical systems, but we did not find one of them that contains all the needs of the three categories patients, doctors and the hospital.
- The patient has to go to the hospital to book his doctor, and he finds it difficult because he sits and waits for a lot of time.
- The patient is forced to go to the hospital to book his doctor, and he finds it difficult because he sits and waits for a lot of time and also book the work of x-rays and medical tests and also receive them.

- We also found that the proportion of patients with pneumonia affects about 15% of children under the age of five around the world, according to the World Health Organization. https://www.who.int/ar/news-room/fact-sheets/detail/pneumonia.
- Under the spread of the Corona virus, the patient, if he suspects that he has the disease, tends to make Lung x-ray, where pneumonia appears.
- We found that these days, the state is working to reduce the circulation of currencies and dealing with them and towards electronic payment.
- We also found it difficult to organize between doctor's and patients' appointments.
- We also found that the doctor does not see the patient's medical history, so the doctor is forced to ask each patient about his medical history and his details, but the medical history is not recorded in order to be preserved if the same patient goes again in follow-up.
- We also found administrative and accounting problems in hospitals.
- We also found that there is a difficulty in dispensing medicines to patients and that they do not reach those who deserve them.

2.3. Need for the new system

To overcome problems found in existing system as mentioned above; then we will build new system that contains the following points: -

- Optimize, manage, and track personal and financial hospital resources.
- **No** chance for **duplicated** patient files and data.
- Manage **lab tests**, and **consultation** of different specialties like cardiology and more.
- Build actionable treatment plans with reminders and targets for patients, staff, and doctors to enhance adherence.
- Manage appointment time slots and timings by lab, clinic, and doctor.
- Access to your portal through our mobile apps.
- You will find support in how to use our website in **Chatbot**.
- Fast **detection** of pneumonia disease.
- Cost effective and easily manageable.
- Easy access to patient data with correct patient history.
- Support Multilingual.

- Support multitenancy.
- Billing and Insurance Management.

2.4. Analysis of the new system

In this section we know who are stakeholders and collect all requirements they want in new system.

2.4.1. Identification of key stakeholders and End-Users

In this chapter we identify all persons who have an interest in the successful implementation of the system either they are inside or outside the organization. Stakeholders are consisting of 3 types: -

a. Internal-Operational:

persons within the organization and regularly interact with the system.

- **Doctor** is the person who examines and diagnoses the patient's condition, determines the optimal treatment, and follows up on his condition and treatment results. He also performs first aid for patients and injured people. He also trains and directs instructions to the nursing staff.
- **Analytics specialist** is the person who receives various samples of blood and other body fluids, marking and sorting and classifying blood samples. he also organizes and stores all chemicals, liquids and compressed gases in accordance with safety instructions. Designs and implements laboratory tests according to standard procedures and takes explanatory notes on the results. He also presents the results of tests and medical examinations to patients and providing specialized doctors with the necessary knowledge for treatment, taking into account the confidentiality of medical laboratory information related to patients.
- **Radiology doctor** is the person who makes sure that the x-ray examination is requested from the treating doctor and determines what part to be photographed and what conditions are required. He also informs patients or department nurses of all the necessary instructions for any examination, such as attending the patient without breakfast or taking a specific tablet. He also adjusts the x-ray tube and determines the x-ray package and the necessary imaging factors. For each patient’s required situation, he chooses the appropriate film size and quality for each examination, puts the patient’s letter and identification number on the

clipboard, and prints the patient's name and examination date on the film, if possible, and his technical number, or write that on the x-ray film. He also develops the films from him or from other radiology assistants according to the order of the work schedule in the radiology department, prepares the chemicals for acidification and daily and periodic cleaning of the acidifying device, and then delivers the x-rays to the patient.

- **Accountant** is the person who Follow up the financial procedures of patients, whether cash payment patients or receivable patients, collecting them, settling the fund, and following up on closing outstanding bills. He also restricts cash payment patients' bills on the system, collects cash from cash payment patients, restricts health insurance patients' bills on the system, follows up on cash payment patients' bills, and follows up pending bills with doctors and administration. Also, organizes the patient's file upon discharge from the hospital and completes financial exit procedures. Also send invoices to public accounting and accounting insurance companies. Also participate in the annual or periodic inventory work in the hospital.

- **Pharmacist** is the person who Dispensing the patient's medication via the doctor's prescription only, as well as dispensing the medication through the health insurance. Educating the patient about the side effects of the drug, writing insurance forms, and communicating with insurance companies to verify that patients get the drugs they want. Also answer patients' inquiries about prescriptions for medicines. Also, organize the pharmacy and its medications in an effective and organized manner, and continuously monitor the expiration dates of the medications. Also, follow-up invoices for purchases and sales of medicines and medical supplies as well. Follow the correct means of storing medicines and try as much as possible to prevent problems of poor storage.

- **Administrator** He is the person responsible for the administrative things in the hospital such as organizing departments, appointing employees, adding departments and things like that.

b. External-Operational

persons outside the organization and regularly interact with the system.

- **Patients** They are the clients who come to the hospital in order to receive the services they request.

- **Pharmaceutical suppliers** They are the people who supply all medicines to the hospital pharmacy as well as all medical supplies.

c. Internal-Executive

persons within the organization and don't directly interact, but use the information or have financial interest.

- **hospital managers** They are the people who have financial interests and who take the highest decisions in the management of the hospital and follow up on all activities in the hospital.

2.4.2. User requirements

Requirements of stakeholders and end-users identified in section 2.4.1.

1. **<u>Administrator:</u>**
   a. **Login:** Admin can login to his personal account.
   b. **View/Edit Profile:** Admin can see and edit his profile.
   c. **Add/View/Delete Doctor:** Admin can Add, view doctor, or delete him from system.
   d. **Add/View/Delete Patient:** Admin can Add, view patient, or delete him from system.
   e. **Add/View/Delete Pharmacist:** Admin can Add, view pharmacist, or delete him from system.
   f. **Add/View/Delete Analytics Specialist:** Admin can Add, view analytics specialist, or delete him from system.
   g. **Add/View/Delete Radiology Doctor:** Admin can Add, view radiology doctor, or delete him from system.
   h. **Add/View/Delete Accountant:** Admin can Add, view accountant, or delete him from system.
   i. **Accept/Reject Join Request:** Admin can accept or reject doctor join request from database.
   j. **Add/View/Update/Delete Specialization:** Admin can Add, view specialization, or update its information, or delete it from system.
   k. **Add/View/Update/Delete Ambulance:** Admin can Add, view ambulance, or update its information, or delete it from system.
   l. **View Appointments:** Admin can see all appointments detail from database.
   m. **View Medicines:** Admin can see all medicines detail from database.
   n. **View Medical Analysis:** Admin can see all medical analysis detail from database.

o. **View X-Rays:** Admin can see all X-Rays detail from database.
p. **View Invoices:** Admin can see all invoices detail from database.
q. **Add/View/Update/Delete Articles:** Admin can Add, view Article, or update its information, or delete it from system.

2. **Hospital Manager:**
   a. **Login:** Hospital Manager can login to his personal account.
   b. **View/Edit Profile:** Hospital Manager can see and edit his profile.
   c. **Add/View/Update/Delete Administrator:** Hospital Manager can add, view administrator, or update his information, or delete him from system.
   d. **Add/View/Update/Delete Role:** Hospital Manager can add, view User Role, or update user's role, or delete this role from user.
   e. **Add/View/Update/Delete Tenant:** Hospital Manager can add new branch of hospital, view, update or delete it.

3. **Doctor:**
   a. **Login:** Doctor can login to his personal account.
   b. **View/Edit Profile:** Doctor can see and edit his profile.
   c. **Request to Join Hospital:** Doctor can Request to join hospital team through 'Join Us' form from UI.
   d. **Diagnose patients:** Doctor can diagnose patients or transfer patient to radiology/analysis department when needed.
   e. **Create Patient Prescription:** Doctor can create patient prescription after diagnosing him.
   f. **Add/View/Update/Delete Service:** Doctor can Add, view services he will do, or update them, or delete them from system.
   g. **View Patient history:** Doctor can view patient history from dashboard.
   h. **Add/view/update/delete Appointments:** Doctor can Add, view appointments dates he will make, or update them, or delete them from system.

4. **Pharmacist:**
   a. **Login:** Pharmacist can login to his personal account.
   b. **View/Edit Profile:** Pharmacist can see and edit his profile.
   c. **Add/View/Update/Delete Medicine:** Pharmacist can Add, view medicine, or update its information, or delete it from system.

5. **Analytics specialist:**
   a. **Login:** Analytics Specialist can login to his personal account.

b. **View/Edit Profile:** Analytics Specialist can see and edit his profile.
c. **Upload medical analysis:** Analytics Specialist can upload medical analysis through dashboard.

6. **Radiology doctor:**
   a. **Login:** Radiology Doctor can login to his personal account.
   b. **View/Edit Profile:** Radiology Doctor can see and edit his profile.
   c. **Upload X-Rays:** Radiology Doctor can upload X-Rays through dashboard.
   d. **Use pneumonia detection service:** Radiology Doctor can upload patient's X-Ray to predict pneumonia.

7. **Accountant:**
   a. **Login:** Accountant can login to his personal account.
   b. **View/Edit Profile:** Accountant can see and edit his profile.
   c. **Add/View/Update Invoice:** Accountant can add new invoice, or view or update its details.
   d. **Add/View/Update/Delete/Reset Due of Insurance:** Admin can Add, view insurance, or update its information, or delete it from system; Admin can also update Due of insurance company.

8. **Patient:**
   a. **Login:** Patient can login to his personal account.
   b. **Register:** Patient can register to the system for the first time.
   c. **View/Edit Profile:** Patient can see and edit his profile.
   d. **Book a doctor:** Patient can make an appointment with doctor based on his services.
   e. **Use Medical Insurance:** Patient can use his medical insurance when dispensing a medicine.
   f. **Talk to Chatbot:** Patient can talk to a chatbot to guide him where to go.
   g. **View Medicines:** Patient can see all medicines detail to buy them online.
   h. **Buy Medicines:** Patient can buy medicines using credit card or cash on delivery.
   i. **View Doctors:** Patient can see all doctors' details to book an appointment.
   j. **Review Doctors:** Patient can review doctor after visiting him in hospital.
   k. **View Specializations:** Patient can see all specializations detail to reach doctors in specific specialization.

2.4.3. System Requirements

System needs Cloud-based to

- Put it in a specific domain to work properly.
- Allows recording of medical data for eventual use.
- Allows data retrieval in real time.
- Allows patients to access and monitor their medical data.
- Does not allow patients to provide their health conditions.
- Supports data sharing only within the same hospital.

2.4.4. Domain Requirements

Based on the medical field, we found that the medical system should contains some sub-systems related to its field like pharmacy management system; So, our system contains many integrated sub-systems such as pharmacy management system, reservation management system and online billing system.

2.4.5. Functional Requirements

They describe what the system/software must do; functionality or services (a function is a useful capability provided by one or more components of a system). Therefore, they specify an action that a system must be able to perform.

- **Login Function: -**
  - **Function:**

    Login
  - **Actors:**

    Administrator, Hospital Manager, Doctor, Pharmacist, Analytics Specialist, Radiology doctor, Accountant, Patient
  - **Priority:**

    High
  - **Description:**
    When the actor login, he can manage the system based on his roles.
  - **Inputs:**

The actor should write the email and password in the right way and correct data for a successful login

- **Outputs:**
  The actor can manage many functions based on roles that assign to him
- **Requirements:**
  Write the right data to can login
- **Pre-condition:**
  The actor already has an account
- **Post-condition:**
  The actor entered his account successfully

➢ View Profile: -

- **Function:**

  View profile
- **Actors:**

  Administrator, Hospital Manager, Doctor, Pharmacist, Analytics Specialist, Radiology doctor, Accountant, Patient
- **Priority:**

  Medium
- **Description:**
  An actor profile is a collection of settings and information associated with a user. It contains critical information that is used to identify an individual, such as their name, age, profile picture, email, and password
- **Inputs:**
  His account must be verified on the site after logging in.
- **Outputs:**
  View personal information successfully.
- **Requirements:**
  Actor information must be added by logging in to the site before viewing his profile.
- **Pre-condition:**
  The actor must be logged in to the system and has permission.
- **Post-condition:**
  The actor can view this profile.

➢ Edit Profile: -

- **Function:**

  Edit profile

- **Actors:**

  Administrator, Hospital Manager, Doctor, Pharmacist, Analytics Specialist, Radiology doctor, Accountant, Patient
- **Priority:**

  High
- **Description:**
  An actor profile is a collection of settings and information associated with a user. It contains critical information that is used to identify an individual, such as their name, age, profile picture, email, and password, and can edit this information based on this role
- **Inputs:**
  His account must be verified on the site after logging in.
- **Outputs:**
  edit personal information successfully.
- **Requirements:**
  Actor information must be added by logging in to the site before viewing his profile.
- **Pre-condition:**
  The actor must be logged in to the system and has permission.
- **Post-condition:**
  The actor can edit this profile.

➢ Add Doctor: -

- **Function:**

  Add Doctor
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the Doctor to the website database and give him an account to make the doctor can do many functions
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully;
- **Outputs:**

The administrator adds doctors to the system and makes accounts for the doctors and gives them some permeations.

- **Requirements:**
  The data about a required doctor will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the doctor give his data to the administrator
- **Post-condition:**
  The doctor is added and has an account.

➢ View Doctor: -

- **Function:**

  View Doctor
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  When the administrator login, he can add the Doctor to the website database and give him an account to make the doctor can do many functions, and he can view doctor details like "see his appointments, personal information, and the number of patients who have examined him.”.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add doctor first; can view the details by clicking “view info”.
- **Outputs:**
  The administrator adds doctors to the system and makes accounts for the doctors and gives them some permeations, views doctors’ details from the system.
- **Requirements:**
  The data about a required doctor will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the doctor give his data to the administrator
- **Post-condition:**
  The administrator views details about all the doctors.

➢ Delete Doctor: -

- **Function:**

  Delete Doctor

- **Actors:**

  Administrator

- **Priority:**

  High

- **Description:**
  When the administrator login, he can add the Doctor to the website database and give him an account to make the doctor can do many functions, and he can delete the doctor.

- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully, and can delete the doctor by clicking “delete doctor”

- **Outputs:**
  The administrator adds doctors to the system and makes accounts for the doctors and gives them some permeations, and can delete doctors’ details from the system

- **Requirements:**
  The data about a required doctor will be added.

- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the doctor give his data to the administrator

- **Post-condition:**
  The administrator deletes all the doctors.

➢ Add Patient: -

- **Function:**

  Add Patient

- **Actors:**

  Administrator

- **Priority:**

- High

- **Description:**

When the administrator login, he can add the patient to the website database and give him an account to make the patient can do many functions.

- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully;
- **Outputs:**
  The administrator adds patients to the system and makes accounts for the patients and gives them some permeations.
- **Requirements:**
  The data about a required patient will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the patient give his data to the administrator
- **Post-condition:**
  The patient is added and has an account.

➢ View Patient: -

- **Function:**

  View Patient
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  When the administrator login, he can add the patient to the website database and give him an account to make the patient can do many functions, and he can view patient details like " Viewing the dates he booked, personal information".
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add patient first; can view the details by clicking "view info".
- **Outputs:**
  The administrator adds patients to the system and makes accounts for the patients and gives them some permeations, views patients' details from the system.
- **Requirements:**

The data about a required patient will be added.

- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the patient give his data to the administrator
- **Post-condition:**
  The administrator views details about all the patients.

➢ Delete Patient: -

- **Function:**

  Delete Patient
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the patient to the website database and give him an account to make the patient can do many functions, and he can delete the patient.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully, and can delete the patient by clicking “delete patient”
- **Outputs:**
  The administrator adds patients to the system and makes accounts for the patients and gives them some permeations, and deletes patients’ details from the system
- **Requirements:**
  The data about a required patient will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the patient give his data to the administrator.
- **Post-condition:**
  The administrator deletes all the patients.

➢ Add Analytics Specialist: -

- **Function:**

  Add Analytics Specialist

- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the analytics specialist to the website database and give him an account to make the analytics specialist can do many functions.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully;
- **Outputs:**
  The administrator adds analytics specialists to the system and makes accounts for the analytics specialists and gives them some permeations.
- **Requirements:**
  The data about a required analytics specialist will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the analytics specialists give his data to the administrator
- **Post-condition:**
  The analytics specialist is added and has an account.

➢ View Analytics Specialists: -

- **Function:**

  View analytics specialists
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  When the administrator login, he can add the analytics specialist to the website database and give him an account to make the analytics specialist can do many functions, and he can view analytics specialist details like " Viewing the personal information".
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add specialist first; can view the details by clicking "view info".

- **Outputs:**
  The administrator adds an analytics specialist to the system and makes accounts for the analytics specialists and gives them some permeations, views analytics specialist details from the system.
- **Requirements:**
  The data about a required analytics specialist will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the analytics specialist give his data to the administrator
- **Post-condition:**
  The administrator views details about all the analytics specialists.

➢ Delete Analytics Specialist: -

- **Function:**

  Delete analytics specialist

- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the analytics specialist to the website database and give him an account to make the analytics specialist can do many functions, and he can delete the analytics specialist.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully, and can delete the analytics specialist by clicking “delete pharmacist”
- **Outputs:**
  The administrator adds analytics specialists to the system and makes accounts for the analytics specialists and gives them some permeations, and deletes the analytics specialist’ details from the system
- **Requirements:**
  The data about a required analytics specialist will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the analytics specialist give his data to the administrator
- **Post-condition:**
  The administrator deletes all the analytics specialists.

➢ Add Pharmacist: -

- **Function:**

  Add Pharmacist

- **Actors:**

  Administrator

- **Priority:**

  High

- **Description:**
  When the administrator login, he can add the Pharmacist to the website database and give him an account to make the Pharmacist can do many functions.

- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully.

- **Outputs:**
  The administrator adds Pharmacists to the system and makes accounts for the Pharmacists and gives them some permeations.

- **Requirements:**
  The data about a required pharmacist will be added.

- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the Pharmacist give his data to the administrator.

- **Post-condition:**
  The Pharmacist is added and has an account.

➢ View Pharmacist: -

- **Function:**

  View Pharmacists

- **Actors:**

  Administrator

- **Priority:**

  Medium

- **Description:**
  When the administrator login, he can add the Pharmacist to the website database and give him an account to make the Pharmacist can do many

functions, and he can view Pharmacist details like " Viewing the personal information".

- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add pharmacist first; can view the details by clicking "view info".
- **Outputs:**
  The administrator adds Pharmacists to the system and makes accounts for the Pharmacists and gives them some permeations, views Pharmacist details from the system.
- **Requirements:**
  The data about a required Pharmacist will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the Pharmacist give his data to the administrator.
- **Post-condition:**
  The administrator views details about all the pharmacists.

➢ Delete Pharmacist: -

- **Function:**

  Delete Pharmacist
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the Pharmacist to the website database and give him an account to make the Pharmacist can do many functions, and he can delete the pharmacist.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully, and can delete the pharmacist by clicking "delete pharmacist"
- **Outputs:**
  The administrator adds pharmacists to the system and makes accounts for the pharmacists and gives them some permeations, and deletes the pharmacist' details from the system
- **Requirements:**

The data about a required Pharmacist will be added.

- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the pharmacist give his data to the administrator
- **Post-condition:**
  The administrator deletes all the pharmacists.

➢ Add Radiology Doctor: -

- **Function:**

  Add Radiology Doctor
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the Radiology Doctor to the website database and give him an account to make the Radiology Doctor can do many functions.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully.
- **Outputs:**
  The administrator adds Radiology Doctors to the system and makes accounts for the Radiology Doctors and gives them some permeations.
- **Requirements:**
  The data about a required Radiology Doctor will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the Radiology Doctor give his data to the administrator.
- **Post-condition:**
  The Radiology Doctor is added and has an account.

➢ View Radiology Doctor: -

- **Function:**

  View Radiology Doctor
- **Actors:**

  Administrator

- **Priority:**

  Medium
- **Description:**
  When the administrator login, he can add the Radiology Doctor to the website database and give him an account to make the Radiology Doctor can do many functions, and he can view Radiology Doctor details like " Viewing the personal information".
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add radiology doctor first; can view the details by clicking "view info".
- **Outputs:**
  The administrator adds Radiology Doctors to the system and makes accounts for the Radiology Doctors and gives them some permeations, views Radiology Doctors' details from the system.
- **Requirements:**
  The data about a required Radiology Doctor will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the Radiology Doctor give his data to the administrator
- **Post-condition:**
  The administrator views details about all the Radiology Doctors.

➢ Delete Radiology Doctor: -

- **Function:**

  Delete Radiology Doctor
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the Radiology Doctor to the website database and give him an account to make the Radiology Doctor can do many functions, and he can delete the Radiology Doctor.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully, and can delete the radiology doctor by clicking "delete radiology doctor"

- **Outputs:**
  The administrator adds radiology doctors to the system and makes accounts for the radiology doctors and gives them some permeations, and deletes the radiology doctor' details from the system
- **Requirements:**
  The data about a required Radiology Doctor will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the radiology doctor give his data to the administrator.
- **Post-condition:**
  The administrator deletes all the radiology doctors.

➢ Add Accountant: -

- **Function:**

  Add Accountant
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the administrator login, he can add the Accountant to the website database and give him an account to make the Accountant can do many functions.
- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully.
- **Outputs:**
  The administrator adds Accountants to the system and makes accounts for the Accountants and gives them some permeations.
- **Requirements:**
  The data about a required Accountant will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the Accountant give his data to the administrator
- **Post-condition:**
  The Accountant is added and has an account.

➢ View Accountant: -

- **Function:**

  View Accountant

- **Actors:**

  Administrator

- **Priority:**

  Medium

- **Description:**
  When the administrator login, he can add the Accountant to the website database and give him an account to make the Accountant can do many functions, and he can view Accountant details like " Viewing the personal information".

- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add accountant first; can view the details by clicking "view info".

- **Outputs:**
  The administrator adds Accountants to the system and makes accounts for the Accountants and gives them some permeations, views the Accountant' details from the system.

- **Requirements:**
  The data about a required Accountant will be added.

- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the Accountant give his data to the administrator

- **Post-condition:**
  The administrator views details about all the Accountants.

➢ Delete Accountant: -

- **Function:**

  Delete Accountant

- **Actors:**

  Administrator

- **Priority:**

  High

- **Description:**

When the administrator login, he can add the Accountant to the website database and give him an account to make the Accountant can do many functions, and he can delete the Accountant.

- **Inputs:**
  Admin should write first name, last name, age, id, username, and password in the right way and correct data to add successfully, and can delete the Accountant by clicking "delete Accountant"
- **Outputs:**
  The administrator adds Accountants to the system and makes accounts for the Accountants and gives them some permeations, and deletes the Accountant' details from the system
- **Requirements:**
  The data about a required Accountant will be added.
- **Pre-condition:**
  The administrator had signed in to his profile (system), has permeation and the radiology doctor give his data to the administrator.
- **Post-condition:**
  The administrator deletes all the Accountants.

➢ Accept Join Requests: -

- **Function:**
  admin accepts Join requests.
- **Actors:**
  Administrator
- **Priority:**
  High
- **Description:**
  The system is available to accept 'doctors requests by the admin, and requests for the doctor to join the site to work on it.
- **Input:**
  Click on the "accept" button on the requested doctor
- **Output:**
  Admin accepts doctor join request.
- **Requirements:**
  Central database to store all doctor join request information.
- **Pre-condition:**

The system is allowed to accept doctor join request.
The admin must be log in system.
The admin has permission to do this.

- **Post-condition:**
  Admin accepts doctor join request.

➢ Reject Join Requests: -

- **Function:**
  admin rejects Join requests.
- **Actors:**
  Administrator
- **Priority:**
  High
- **Description:**
  The system is available to reject 'doctors requests by the admin, and requests for the doctor to join the site to work on it.
- **Input:**
  Click on the "Reject" button on the requested doctor
- **Output:**
  Admin rejects doctor join request
- **Requirements:**
  Central database to store all doctor join request information.
- **Pre-condition:**
  The system is allowed to reject doctor join request.
  The admin must be log in system.
  The admin has permission to do this.
- **Post-condition:**
  Admin rejects doctor join request.

➢ Add Specialization: -

- **Function:**
  Add Specialization
- **Actors:**
  Administrator
- **Priority:**

High

- **Description:**
  When the admin login, he can add a specialization to the website database, and make patient can access to view it
- **Inputs:**
  Admin should write name, id, and some information about the specialization in the right way and correct data to add successfully
- **Outputs:**
  Admin adds specializations to system and patient can access to view it
- **Requirements:**
  The data about a required specialization will be added
- **Pre-condition:**

Admin had signed into his profile (system) and has a permeation

- **Post-condition:**
  specialization is added.

➢ View Specialization

- **Function:**
  View specialization
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  When the admin logins, he can view specialization details like “name and some other info”
- **Inputs:**
  Click on ‘specialization’ button to show all specializations details.
  Outputs:
  Admin views specializations details from the system
- **Requirements:**
  The data about a required specialization that System User wants to view
- **Pre-condition:**
  Admin had signed into his profile (system) and has a permeation
- **Post-condition:**
  Admin view details about all the specializations.

➢ Update Specialization

- **Function:**
  Update specialization
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  Updating Specialization is a behavior done by an Administrator. When the Administrator updates a specialization, the old information of the specialization will be changed to new information.
- **Inputs:**
  The Administrator chooses specialization to make updates on him.
- **Outputs:**
  The specialization will be successfully updated.
- **Requirements:**
  Only the administrator can update the specialization that exists in the system.
- **Pre-condition:**
  The administrator must log in system and verified to update the specialization is existing in the system
- **Post-condition:**
  The specialization will be updated in the database.

➢ Delete Specialization

- **Function:**
  Delete specialization
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  The admin is available to delete a Specialization with his information
- **Inputs:**
  The admin must enter the information about the Specialization system with everything the Specialization contains and with more accurate details.
- **Outputs:**
  The specialization will be deleted successfully.

- **Requirements:**
  Only the administrator can delete this existing specialization from the system.
- **Pre-condition:**
  The administrator must log in system and verify to delete the specialization is existing in the system
- **Post-condition:**
  The specialization will be deleted from the database.

➢ Add Articles

- **Function:**
  Add Articles
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  When the admin login, he can add an article to the website database, and make patient can access to view it.
- **Inputs:**
  Admin should write title, section, and some information about the article in the right way and correct data to add article successfully.
- **Outputs:**
  Admin adds articles to system and patient can access to view it
- **Requirements:**
  The data about the required article will be added.
- **Pre-condition:**
  Admin had signed into his profile (system) and has a permeation.
- **Post-condition:**
  article is added.

➢ View Articles

- **Function:**
  View Article
- **Actors:**

  Administrator
- **Priority:**

Medium

- **Description:**
  When the admin login, he can view article details like “title, section and some other info.”
- **Inputs:**
  Click on ‘articles’ button to show all articles details.
- **Outputs:**
  Admin views articles details from the system
- **Requirements:**
  The data about a required article that admin wants to view.
- **Pre-condition:**
  Admin had signed into his profile (system) and has a permeation.
- **Post-condition:**
  Admin view details about all the articles.

➢ Update Articles

- **Function:**
  Update articles
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  Updating Articles is a behavior done by an Administrator. When the Administrator updates an article, the old information of the article will be changed to new information.
- **Inputs:**
  The Administrator chooses an article to make updates on him.
- **Outputs:**
  The article will be successfully updated.
- **Requirements:**
  Only the administrator can update the article that exists in the system.
- **Pre-condition:**
  The administrator must log in system and verified to update the article is existing in the system.
- **Post-condition:**
  The article will be updated in the database.

➢ Delete Article

- **Function:**
  Delete Article
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  The admin is available to delete an article with his information.
- **Inputs:**
  The admin must enter the information about the article with everything the article contains and with more accurate details.
- **Outputs:**
  The article will be deleted successfully.
- **Requirements:**
  Only the administrator can delete this existing article from the system.
- **Pre-condition:**
  The administrator must log in system and verify to delete the article is existing in the system.
- **Post-condition:**
  The article will be deleted from the database.

➢ Add Ambulance: -

- **Function:**
  Add Ambulance
- **Actors:**

  Administrator
- **Priority:**

  High
- **Description:**
  When the admin login, he can add an Ambulance to the website database, and make patient can access to view it
- **Inputs:**
  Admin should write name, id, and some information about the Ambulance in the right way and correct data to add successfully
- **Outputs:**
  Admin adds Ambulances to the system and patients can access to view it

- **Requirements:**
  The data about a required Ambulance will be added
- **Pre-condition:**

Admin had signed into his profile (system) and has a permeation

- **Post-condition:**
  Ambulance is added.

➢ View Ambulance

- **Function:**
  View Ambulance
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**
  When admin login, he can view Ambulance details
- **Inputs:**
  Click on ‘ambulance’ button to show all ambulance details.
- **Outputs:**
  admin views ambulances detail from the system
- **Requirements:**
  The data about a required Ambulance that System wants to view
- **Pre-condition:**
  Admin had signed into his profile (system) and has a permeation

- **Post-condition:**
  Admin view details about all the Ambulances.

➢ Update Ambulance

- **Function:**
  Update ambulance
- **Actors:**

  Administrator
- **Priority:**

  Medium
- **Description:**

Updating an ambulance is a behavior done by an Administrator. When the Administrator updates an ambulance, the old information about the ambulance will be changed to new information.

- **Inputs:**
  The Administrator chooses an ambulance to make updates on him.
- **Outputs:**
  The specialization will be successfully updated.
- **Requirements:**
  Only the administrator can update the ambulance that exists in the system.
- **Pre-condition:**
  The administrator must log in system and verified to update the ambulance is existing in the system
- **Post-condition:**
  The ambulance will be updated in the database.

➢ Delete Ambulance

- **Function:**
  Delete ambulance
- **Actors:**
  Administrator
- **Priority:**
  Medium
- **Description:**
  The admin is available to delete an ambulance with his information
- **Inputs:**
  The admin must enter the information about the ambulance system with everything the ambulance contains and with more accurate details.
- **Outputs:**
  The ambulance will be deleted successfully.
- **Requirements:**
  Only the administrator can delete this existing ambulance from the system.
- **Pre-condition:**
  The administrator must log in system and verify to delete the ambulance is existing in the system
- **Post-condition:**
  The ambulance will be deleted from the database.

➢ Add Insurance: -

- **Function:**
  Add Insurance
- **Actors:**

  Accountant
- **Priority:**

  High
- **Description:**
  When the accountant login, he can add Insurance to the website database, and make patient can access to view it
- **Inputs:**
  Accountant should write name, id, and some information about the Insurance in the right way and correct data to add successfully
- **Outputs:**
  Accountant adds Insurance to the system and patients can access to view it
- **Requirements:**
  The data about a required Ambulance will be added
- **Pre-condition:**
  Accountant had signed into his profile (system) and has a permeation
- **Post-condition:**
  Insurance is added.

➢ View Insurance

- **Function:**
  View Insurance
- **Actors:**

  Accountant & Admin
- **Priority:**

  Medium
- **Description:**
  When the accountant or admin login, they can view Insurance details
- **Inputs:**
  Click on ‘insurance’ button to show all insurances details.”
- **Outputs:**
  accountant or admin view Insurance detail from the system
- **Requirements:**
  The data about required Insurance that System wants to view

- **Pre-condition:**
  Accountant or Admin had signed into them profile (system) and have a permeation
- **Post-condition:**
  Accountant or Admin view details about all the Insurance.

➢ Update Insurance

- **Function:**
  Update Insurance
- **Actors:**
  Accountant
- **Priority:**
  Medium
- **Description:**
  Updating Insurance is a behavior done by an accountant. When the accountant update Insurance, the old information about the Insurance will be changed to new information.
- **Inputs:**
  The accountant chooses an insurance to make updates on him.
- **Outputs:**
  The Insurance will be successfully updated.
- **Requirements:**
  Only accountant can update the Insurance that exists in the system.
- **Pre-condition:**
  The accountant must log in system and verify to update the Insurance is existing in the system
- **Post-condition:**
  The Insurance will be updated in the database.

➢ Delete Insurance

- **Function:**
  Delete Insurance
- **Actors:**
  Accountant
- **Priority:**
  Medium
- **Description:**

The accountant is available to delete an Insurance with his information

- **Inputs:**
  The accountant must enter the information about the Insurance system with everything the Insurance contains and with more accurate details.
- **Outputs:**
  The Insurance will be deleted successfully.
- **Requirements:**
  Only the accountant can delete this existing Insurance from the system.
- **Pre-condition:**
  The accountant must log in system and verify to delete the Insurance is existing in the system
- **Post-condition:**
  The Insurance will be deleted from the database.

➢ Reset Insurance Due

- **Function:**
  Reset Insurance Due
- **Actors:**
  Accountant
- **Priority:**
  Medium
- **Description:**
  The accountant is available to reset an insurance due.
- **Inputs:**
  Click on 'insurance' button to show all insurance info.
- **Outputs:**
  The insurance due will be reset successfully.
- **Requirements:**
  Only the accountant can reset due of this existing Insurance from the system.
- **Pre-condition:**
  The accountant must log in to the system and verify to reset the insurance due which is existing in the system
- **Post-condition:**
  The insurance due will be reset.

➢ View medical Analysis

- **Function:**

View medical analysis

- **Actors:**
  Administrator
- **Priority:**
  medium
- **Description:**
  When admin login, he can view medical analysis details
- **Inputs:**
  Click on ‘medical analysis’ button to show all analysis details.
- **Outputs:**
  Admin views medical analysis details from the system
- **Requirements:**
  The data about required medical analysis that admin wants to view
- **Pre-condition:**
  Admin had login into his profile (system) and has permission
- **Post-condition:**
  Admin view details about all filtered the medical analysis.

View X-rays

- **Function:**
  View x-rays
- **Actors:**
  Administrator
- **Priority:**
  medium
- **Description:**
  When admin login, he can view the details of the x-ray
- **Inputs:**
  Click on ‘X-Ray’ button to show all rays details.
- **Outputs:**
  Admin views x-ray details from the system
- **Requirements:**
  The data about required x-rays that admin wants to view
- **Pre-condition:**
  Admin had login into his profile (system) and has permission
- **Post-condition:**
  Admin view details about all filtered the x-rays.

➢ Add Medicine: -

- **Function:**
  Add Medicine
- **Actors:**

  Pharmacist.
- **Priority:**

  High
- **Description:**
  Adding Medicine is behavior done by a pharmacist. When the pharmacist adds a medicine, He'll add details for the medicine such as the name and image etc.

- **Input:**
  The pharmacist chooses category he wants to put the medicine in.
- **Output:**
  The medicine will be successfully added.
- **Requirements:**
  Only the pharmacist can add the medicine.
- **Pre-condition:**
  The pharmacist must log in system and verified to add pharmacist.
- **Post-condition:**
  The medicine will be saved to database.

➢ Update Medicine: -

- **Function:**
  Update Medicine
- **Actors:**

  Pharmacist.
- **Priority:**

  medium
- **Description:**
  Updating Medicine is behavior done by a pharmacist. When the pharmacist Update a medicine, the old information of medicine will be changed to new information.

- **Input:**
  The pharmacist chooses medicine to make update on him.
- **Output:**
  The medicine will be successfully updated.

- **Requirements:**
  This medicine exists in system and only the pharmacist can update it.
- **Pre-condition:**
  The pharmacist must log in system and verified to update pharmacist and medicine is exist in system.
- **Post-condition:**
  The medicine will be updated in database.

➢ Delete Medicine: -

- **Function:**
  Delete Medicine
- **Actors:**

  Pharmacist.
- **Priority:**

  high
- **Description:**
  Delete Medicine is behavior done by a pharmacist. When the pharmacist delete a medicine, the medicine will be deleted from the medicine list.

- **Input:**
  The pharmacist chooses medicine to delete.
- **Output:**
  The medicine will be successfully deleted.
- **Requirements:**
  This medicine exists in system and only the pharmacist can delete it.
- **Pre-condition:**
  The pharmacist must log in system and verified to delete medicine and medicine is exist in system.
- **Post-condition:**
  The medicine will be deleted from database.

➢ Upload Medical Analysis: -

- **Function:**
  Upload medical analysis.
- **Actors:**

  Analytics specialist.
- **Priority:**

  medium

- **Description:**
  After Analytics specialist finish medical analysis, he uploads result of medical analysis as file on system.
- **Input:**
  The doctor must send prescription about what type of medical analysis that he does.
- **Output:**
  The medical analysis result will be successfully Added.
- **Requirements:**
  Only the Analytics specialist can Add the result of medical analysis.
- **Pre-condition:**
  The Analytics specialist must log in system and type of medical analysis is exist.
- **Post-condition:**
  The medical analysis result will be saved to database.

➢ Upload X-Rays: -

- **Function:**
  Upload x-ray.
- **Actors:**
  Radiology doctor.
- **Priority:**
  medium
- **Description:**
  After Radiology doctor finish x-ray, he uploads result of x-ray as file on system.
- **Input:**
  The doctor must send prescription about what type of x-ray that he does.
- **Output:**
  The x-ray result will be successfully Added.
- **Requirements:**
  Only the Radiology doctor can Add the result of medical analysis.
- **Pre-condition:**
  The Radiology doctor must log in system and type of x-ray is exist.
- **Post-condition:**
  The x-ray result will be saved to database.

➢ Add User: -

- **Function:**
  Add User.
- **Actor:**
  The Hospital Manager
- **Priority:**
  Critical
- **Description:**
  The Hospital Manager is available to add any user and describe his privileges.
- **Input:**
  Enter User Information
- **Output:**
  The Hospital Manager adds user
- **Requirements:**
  The Hospital Manager **must add users.**
- **Pre-condition:**
  The system is allowed to add users
  The Hospital Manager must be logged in the system
  Users hasn't been created yet.
- **Post-condition:**
  User is added successfully.

➢ View User: -

- **Function:**
  View User.
- **Actor:**
  The Hospital Manager
- **Priority:**
  Critical
- **Description:**
  The Hospital Manager can view the users he created.
- **Input:**
  Click on 'users' button to show all users details..
- **Output:**
  The Hospital Manager views list of users.
- **Requirements:**
  The Hospital Manager **must view users list.**

- **Pre-condition:**
  The system is allowed to view users.
  The Hospital Manager must be logged in the system.
  User has been created.
- **Post-condition:**
  User is viewed successfully.

➢ Update User: -

- **Function:**
  Update User.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**
  The Hospital Manager can update users he created.
- **Input:**
  Click on specific user to update his details.
- **Output:**
  The Hospital Manager updates user.
- **Requirements:**
  The Hospital Manager **can update user description and privileges.**
- **Pre-condition:**
  The system is allowed to update user.
  The Hospital Manager must be logged in the system.
  User has been created.
- **Post-condition:**
  User is updated successfully.

➢ Delete User: -

- **Function:**
  Delete User.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**

The Hospital Manager can delete users he created.

- **Input:**
  Click on specific user to delete him from system.
- **Output:**
  The Hospital Manager deletes users.
- **Requirements:**
  The Hospital Manager **can delete users.**
- **Pre-condition:**
  The system is allowed to delete users.
  The Hospital Manager must be logged in the system.
  User have been created.
- **Post-condition:**
  User is deleted successfully.

➢ Add Role: -

- **Function:**
  Add Role.
- **Actor:**
  The Hospital Manager
- **Priority:**
  Critical
- **Description:**
  The Hospital Manager is available to add roles and describe the privilege for each user. The role is the validities that the user takes, whoever (Admin, Pharmacist, Doctor, Analytics Specialist, Radiology Doctor, Accountant or Patient) to perform certain tasks.
- **Input :**
  The Hospital Manager **create roles and describe privileges for each role**
- **Output :**
  The Hospital Manager adds the roles
- **Requirements :**
  The Hospital Manager **must add roles and describe their privileges**
- **Pre-condition :**
  The system is allowed to add roles
  The Hospital Manager must be logged in the system
  Role hasn't been created yet.
- **Post-condition:**
  Role is created successfully.

➢ View Role: -

- **Function:**
  View Role.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**
  The Hospital Manager can view roles he created.
- **Input:**
  Click on ‘roles’ button to show all roles details.
- **Output:**
  The Hospital Manager view role page.
- **Requirements:**
  The Hospital Manager **can view roles and privileges.**
- **Pre-condition:**
  The system is allowed to view roles.
  The Hospital Manager must be logged in the system.
  Role has been created.
- **Post-condition:**
  Role is viewed successfully.

➢ Update Role: -

- **Function:**
  Update Role.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**
  The Hospital Manager can update roles he created.
- **Input:**
  Role name and Role ID
- **Output:**
  The Hospital Manager updates role
- **Requirements:**
  The Hospital Manager **can update roles and privileges**

- **Pre-condition:**
  The system is allowed to update roles
  The Hospital Manager must be logged in the system
  Role has been created.
- **Post-condition:**
  Role is updated successfully.

➢ Delete Role: -

- **Function:**
  Delete Role.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**
  The Hospital Manager can delete roles he created.
- **Input:**
  Enter Role ID or Role Name
- **Output:**
  The Hospital Manager deletes roles
- **Requirements:**
  The Hospital Manager **can delete roles**
- **Pre-condition:**
  The system is allowed to delete roles
  The Hospital Manager must be logged in the system
  Roles have been created.
- **Post-condition:**
  Roles deleted successfully.

➢ Add Tenant: -

- **Function:**
  Add Tenant.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**

The Hospital Manager is available to add tenant to the system to be available to use the system in many tenant like (UAE, OMAN, etc ,……).

- **Input :**
  The Hospital Manager **create tenants and describe its community and the where is it**
- **Output :**
  The Hospital Manager adds the tenants
- **Requirements :**
  The Hospital Manager **must know tenants and its community or information about it**
- **Pre-condition :**
  The system is allowed to add tenants
  The Hospital Manager must be logged in the system
  Tenant hasn't been created yet.
- **Post-condition:**
  Tenant is created successfully.

➢ View Tenant: -

- **Function:**
  View Tenant.
- **Actor:**
  The Hospital Manager
- **Priority:**
  Critical
- **Description:**
  The Hospital Manager can view tenants he created.
- **Input:**
  Click on 'tenants' button to show all tenants details**.**
- **Output:**
  The Hospital Manager view tenant page and its details.
- **Requirements:**
  The Hospital Manager **can view tenants.**
- **Pre-condition:**
  The system is allowed to view tenants.
  The Hospital Manager must be logged in the system.
  Role has been created.
- **Post-condition:**
  Tenant is viewed successfully.

➢ Update Tenant: -

- **Function:**
  Update Tenant.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**
  The Hospital Manager can update tenants he created.
- **Input:**
  Tenant name and Tenant ID
- **Output:**
  The Hospital Manager updates Tenant
- **Requirements:**
  The Hospital Manager **can update tenants**
- **Pre-condition:**
  The system is allowed to update tenants
  The Hospital Manager must be logged in the system
  Tenant has been created.
- **Post-condition:**
  Tenant is updated successfully.

➢ Delete Tenant: -

- **Function:**
  Delete Tenant.
- **Actor:**

  The Hospital Manager
- **Priority:**

  Critical
- **Description:**
  The Hospital Manager can delete tenants he created.
- **Input:**
  Enter Tenant ID or Tenant Name
- **Output:**
  The Hospital Manager deletes tenants
- **Requirements:**
  The Hospital Manager **can delete** tenants

- **Pre-condition:**
  The system is allowed to delete tenants
  The Hospital Manager must be logged in the system
  Tenant have been created.
- **Post-condition:**
  Tenant deleted successfully.

➢ Add Invoice: -

- **Function:**
  Add Invoice.
- **Actor:**

  Accountant
- **Priority:**

  High
- **Description:**
  Accountant must add invoices and manage the financial part of the hospital.
- **Input:**
  Add Invoice details and for whom this invoice belong to.
- **Output:**
  Accountant creates invoice
- **Requirements:**
  Accountant **should write the** invoice **details**
- **Pre-condition:**
  The system is allowed to create invoices
  The Accountant must be logged in the system
  Patients and Doctors have been created.
- **Post-condition:**
  Invoice is added successfully.

➢ Request to join hospital: -

- **Function:**
  Join doctor.
- **Actor:**

  Doctor
- **Priority:**

  Medium.
- **Description:**

When doctor register by filling his information, he/she will wait until Request will be Accepted then he/she can login into the system.

- **Input:**
  Add doctor`s information.
- **Output:**
  The Doctor`s information saved successfully in database.
- **Requirements:**
  Doctor must write the right data can register.
- **Pre-condition:**
  Doctor don't have an account.
- **Post-condition:**
  Doctor register successfully and can login into the system.

➢ Diagnose Patient: -

- **Function:**
  Diagnose Patient.
- **Actor:**
  Doctor
- **Priority:**
  High
- **Description:**
  Doctor must add patient diagnosis and can send them to the specialist.
- **Input:**
  Add patient details and his/her diagnosis.
- **Output:**
  The Doctor adds patient diagnosis
- **Requirements:**
  Doctor **must diagnose patient and write the diagnosis details , send patient to specialist if needed**
- **Pre-condition:**
  The system is allowed to add diagnosis
  The Doctor must be logged in the system
  Patient, Radiology Doctor, Analytics Specialist and Doctors have been created.
  The Doctor should view patient history.
- **Post-condition:**
  Diagnosis is added successfully.

➢ Create Patient Prescription: -

- **Function:**
  Create prescription
- **Actor:**

  Doctor
- **Priority:**

  High
- **Description:**
  Doctor must create prescription after diagnose patient.
- **Input:**
  Add patient details, date and medicines needed.
- **Output:**
  The Doctor adds prescription.
- **Requirements:**
  Doctor **must diagnose patient and write the diagnosis first and then write the prescription**
- **Pre-condition:**
  The system is allowed to add prescription
  The Doctor must be logged in the system
  Patients and Doctors have been created.
  Patient Diagnosis has been created
- **Post-condition:**
  Prescription is added successfully

➢ Add Service: -

- **Function:**
  Add service
- **Actor:**

  Doctor
- **Priority:**

  Medium
- **Description:**
  Doctor must add service. Service is what the doctor will do when the patient books him.
- **Input:**
  Add service details.
- **Output:**
  The Doctor adds service.

- **Requirements:**
  Doctor **must add service so patient books him**
- **Pre-condition:**
  The system is allowed to add service.
  The Doctor must be logged in the system
  The Doctor has been created.
- **Post-condition:**
  Service is added successfully.

➢ Update Service: -

- **Function:**
  Update service
- **Actor:**
  Doctor
- **Priority:**
  Medium
- **Description:**
  Doctor can update the service which were added.
- **Input:**
  Write service ID. Add service details want to be updated.
- **Output:**
  The Doctor updates service.
- **Requirements:**
  Doctor **must has created service so he could update it.**
- **Pre-condition:**
  The system is allowed to update service.
  The Doctor must be logged in the system
  The Doctor has been created.
  The Service has been created
- **Post-condition:**
  Service is updated successfully.

➢ Delete Services: -

- **Function:**
  Update services
- **Actor:**
  Doctor
- **Priority:**

Medium

- **Description:**
  Doctor can delete the added services.
- **Input:**
  Write service ID. Delete service.
- **Output:**
  The Doctor deletes the service.
- **Requirements:**
  Doctor **must has created service so he could delete it.**
- **Pre-condition:**
  The system is allowed to delete service.
  The Doctor must be logged in the system
  The Doctor has been created.
  The Service has been created.
- **Post-condition:**
  Service deleted successfully.

➢ **View Services: -**

- **Function:**
  View services
- **Actor:**

  Doctor
- **Priority:**

  Medium
- **Description:**
  Doctor can view services which were added.
- **Input:**
  Click on button service list
- **Output:**
  The Doctor's services view.
- **Requirements:**
  Doctor **must has created service so he could view it.**
- **Pre-condition:**
  The system is allowed to view service.
  The Doctor must be logged in the system
  The Doctor has been created.
  The Service has been created.
- **Post-condition:**
  Service is viewed successfully.

➢ Add Appointments:

- **Function:**
  Add appointments
- **Actors:**
  Doctor
- **Priority:**
  Medium
- **Description:**
  The doctor must add appointments. Appointments are the time that the patient books with the doctor for the examination to take place.
- **Inputs:**
  Add appointments details.
- **Outputs:**
  The Doctor adds appointments, user views appointment details from the system
- **Requirements:**
  The doctor must add appointments so the patient books him
- **Pre-condition:**
  The system is allowed to add appointments.
  The Doctor must be logged in to the system
  The Doctor has been created.
- **Post-condition:**
  Appointments are added successfully.

➢ Update Appointments:

- **Function:**
  Update appointments
- **Actors:**
  Doctor
- **Priority:**
  Medium
- **Description:**
  Doctor can update the appointments which were added.
- **Inputs:**
  Write appointments ID. Add appointments details want to be updated.
- **Outputs:**
  The Doctor updates appointments.
- **Requirements:**

Doctor **must has created** appointments **so he could update it.**

- **Pre-condition:**
  The system is allowed to update appointments.
  The Doctor must be logged in the system
  The Doctor has been created.
  The appointments have been created
- **Post-condition:**
  Appointments are updated successfully.

➢ View Appointments:

- **Function:**
  View appointments
- **Actors:**
  Doctor
- **Priority:**
  Medium
- **Description:**
  When Admin login, he can view appointment details like “name, date and some another info”
- **Inputs:**
  Click on ‘appointments’ button to show all appointments details.”
- **Outputs:**
  Admin views appointment details from the system
- **Requirements:**
  The data about required appointment that admin wants to view and has permission
- **Pre-condition:**
  Admin had login into his profile (system) and has permission
- **Post-condition:**
  Admin view details about all filtered the appointment

➢ Delete Appointments:

- **Function:**
  Delete appointments
- **Actors:**
  Doctor
- **Priority:**
  Medium
- **Description:**

Doctor can delete the appointments which were added.

- **Inputs:**
  Write appointments ID. Delete appointment.
- **Outputs:**
  The Doctor deletes appointments.
- **Requirements:**
  Doctor **must has created** appointments **so he could** delete **it.**
- **Pre-condition:**
  The system is allowed to update appointments.
  The Doctor must be logged in the system
  The Doctor has been created.
  The appointments have been created
- **Post-condition:**
  Appointments are deleted successfully.

➢ View Patient History: -

- **Function:**
  View Patient History
- **Actor:**
  Doctor
- **Priority:**
  High
- **Description:**
  Doctor can view patient history to help doctor in diagnoses.
- **Input:**
  Click on specific patient from patient list to get his details and history.
- **Output:**
  The medical history of patient.
- **Requirements:**
  Doctor must view **patient history to help in diagnosis.**
- **Pre-condition:**
  The system is allowed to view patient history.
  The Doctor must be logged in the system
  The Doctor and patient have been created.
- **Post-condition:**
  List of patient's history is viewed successfully.

➢ Pneumonia Detection: -

- **Function:**
  Detect Pneumonia
- **Actor:**
  Radiology Doctor
- **Priority:**
  High
- **Description:**
  Radiology Doctor can upload patient lung`s x-ray to help doctor in pneumonia diagnoses.
- **Input:**
  upload lung`s x-ray.
- **Output:**
  The pneumonia Positive or Negative.
- **Requirements:**
  The Radiology Specialist did the X-ray to the patient so it can be uploaded.
- **Pre-condition:**
  The Radiology Doctor must be logged in the system.
  The X-ray is existed and uploaded.
- **Post-condition:**
  pneumonia diagnoses result and diagnoses x-ray saved in database.

➢ Register function:

- **Function:**
  Register
- **Actors:**
  Patient
- **Priority:**
  High
- **Description:**
  When user register by filling his information, he can login into the system, use its services in system and user`s information is added to the database
- **Inputs:**
  User should write first name, last name, phone number, user name (E-mail) and password in right way and correct data for successfully register
- **Outputs:**
  User can login into system

- **Requirements:**
  Write the right data to can register
- **Pre-condition:**
  User don't have an account
- **Post-condition:**
  User register successfully and can login into the system

➢ Book a doctor:

- **Function:**

  Book a doctor
- **Actors:**
  Patient
- **Priority:**
  High
- **Description:**
  When user login, he can book a doctor and make an appointment based on specialty
- **Inputs:**
  User should choose the doctor who can help him and enter the date and time which suit the user to make the appointment
- **Outputs:**
  User book a doctor and make an appointment
- **Requirements:**
  Write the right data (doctor, date, time)
- **Pre-condition:**
  Users have an account and login into the system
- **Post-condition:**
  Appointment is made

➢ View Doctors

- **Function:**
  View doctors
- **Actors:**
  Patient, Administrator
- **Priority:**
  Low
- **Description:**

When User login, he can filter and view doctor details like “name, specialty and some another info”

- **Inputs:**
  Click on ‘doctors’ button to show all doctors details.
- **Outputs:**
  User views doctor details from the system
- **Requirements:**
  The data about required doctor that user wants to view
- **Pre-condition:**
  User had login into his profile (system) and has permission
- **Post-condition:**
  System User view details about all filtered the doctors

➢ View medicines

- **Function:**
  View medicines
- **Actors:**
  Patient, pharmacist
- **Priority:**
  Medium
- **Description:**
  When User login, he can filter and view medicine details like “name, benefits and some another info” and can buy it online by adding to card
- **Inputs:**
  Click on ‘medicines’ button to show all medicines details.
- **Outputs:**
  User views doctor details from the system
- **Requirements:**
  The data about required medicine that user wants to view
- **Pre-condition:**
  User had login into his profile (system) and has permission
- **Post-condition:**
  User view details about all filtered the medicines

➢ Buy medicines

- **Function:**
  Buy medicines.
- **Actors:**

Patient.

- **Priority:**
  high
- **Description:**
  When Patient login, he can buy medicines using credit card or cash on delivery.
- **Inputs:**
  Patient should choose the medicines to buy.
- **Outputs:**
  The patient paid for the medicine and took it.
- **Requirements:**
  The required medicine and the medicine price are available.
- **Pre-condition:**
  Patient logged in into his system and has permission to buy and view medicines.
- **Post-condition:**
  Patient took the medicine he wanted.

➢ View Specialization:

- **Function:**
  View specialization
- **Actors:**
  Patient, Administrator
- **Priority:**
  medium
- **Description:**
  When user login, he can filter and view specialization details like “name”
- **Inputs:**
  User should write specialization name in right way and correct data to be able to view the details and click “view info”
- **Outputs:**
  User views specialization details from the system
- **Requirements:**
  The data about required specialization that user wants to view
- **Pre-condition:**
  User had login to his profile (system) and has permission
- **Post-condition:**
  User view details about all filtered the specializations

➢ Use medical Insurance:

- **Function:**
  Use medical insurance
- **Actors:**
  Patient
- **Priority:**
  Medium
- **Description:**
  When user login, he can use medical insurance to have a discount on price of medicines
- **Inputs:**
  User should enter your medical insurance in right way
- **Outputs:**
  User dispends the medicine from the system
- **Requirements:**
  The data about required medical insurance that user wants to use
- **Pre-condition:**
  User had login to his profile (system) and has permission
- **Post-condition:**
  User dispends the medicine from the system

➢ Talk to chatbot:

- **Function:**
  Talk of chatbot
- **Actors:**
  Patient
- **Priority:**
  Medium
- **Description:**
  When user login, he can use medical insurance to have a discount on price of medicines
- **Inputs:**
  User should open chatbot to be guided
- **Outputs:**
  User dispends the medicine from the system
- **Requirements:**
  Open the chatbot in right way and has permission
- **Pre-condition:**

User had login to his profile (system) and has permission

- **Post-condition:**
  User talk to chatbot and be guided by it.

➢ Review Doctor:-

- **Function:**
  Review doctor
- **Actors:**
  Patient
- **Priority:**
  Medium
- **Description:**
  When accountant logins, they can review the doctor and give him a rate
- **Inputs:**
  Accountant should write doctor name or the name of specializations in right way and correct data to be able to get the doctor to review him or rate him
- **Outputs:**
  Accountant reviews the doctors or rate them
- **Requirements:**
  The patient should visit doctor in his appointment to review him and the doctor give a confirm if the patient visited him or not
  If not the patient can't review or rate
- **Pre-condition:**
  Patient had login into his profile (system), has permission and already visited the doctor in his appointment
- **Post-condition:**
  Patient reviews the doctors or rate them.

➢ View Invoices:-

- **Function:**
  View invoices
- **Actors:**
  Accountant & Administrator
- **Priority:**
  Medium
- **Description:**
  When accountant or admin login, they can view invoices details
- **Inputs:**

Accountant should write invoices in right way and correct data to be able to view the details and click “view info”

- **Outputs:**
  Accountant or Admin views invoices details from the system
- **Requirements:**
  The data about required invoices that user wants to view
- **Pre-condition:**
  Accountant or Admin had login into his profile (system) and has permission
- **Post-condition:**
  Accountant or Admin view details about all filtered the invoices

2.4.6. Non- Functional Requirements

It specifies system/software properties (such as reliability and safety), and constraints on the services or functions offered by the system (such as timing constraints, response-time), or constraints on the development process.

- **Usability & Humanity.**
  - The product shall be easy to use on the first attempt by a member of the public without training.
  - **Intuitiveness:** the interface is easy to learn and navigate; buttons, headings, and help/error messages are simple to understand

- **Performance.**
  - **Response Time:** The system provides a fast acknowledgment.
  - **User-Interface:** The user interface acknowledges fast as we are using single page application.

- **Maintainability & Support.**
  - Expected changes, and the time allowed to make them.
  - **Back-Up:** The system offers efficiency for data backup.
  - **Errors:** The system must be support error handling and will track every mistake as well as keep a log of it.

- **Security.**
  - **Logon ID: -** Any user who uses the system shall have a Logon ID and Password (Authentication).
  - **Modification: -** Any modification (inert, delete, update) for the Database shall be synchronized and only by the role that user has in the ward (Authorization).

- **Availability.**
  - The system shall be available all the time.

- **Software Quality.**
  - Good quality of the framework= produces robust, bug free software which contains all necessary requirements Customer satisfaction.

- **Reusability.**
  - Is part of the code going to be used elsewhere= produces simple and independent code modules that can be reused.

2.5. Advantages of the new system

- **Validation**: usage of validation and regex when logging into the system and registering for the first time.
- **Verification:** Email verification will be sent to patient when registered.
- **Roles & Permissions:** Each user has his own permission so based on user permission he can does any modification on specified tables in the database (insert, delete, update, etc.).
- **Response Time:** The system provides a fast acknowledgment.
- **User-Interface:** The user interface acknowledges fast as we are using single page application.
- **Back-Up:** The system offers efficiency for data backup.
- **System Tracking:** The system will track every mistake as well as keep a log of it.
- **Availability:** The system is available all the time.
- **Support Multilingual:** The system supports two languages (Arabic and English).

- **Support Multitenancy:** Instead of forcing you to change how you write your code, the system by default bootstraps tenancy automatically, in the background. Database connections are switched, caches are separated, file systems are prefixed.

2.6. Use Case: -

2.7.1. Use Case Diagram:

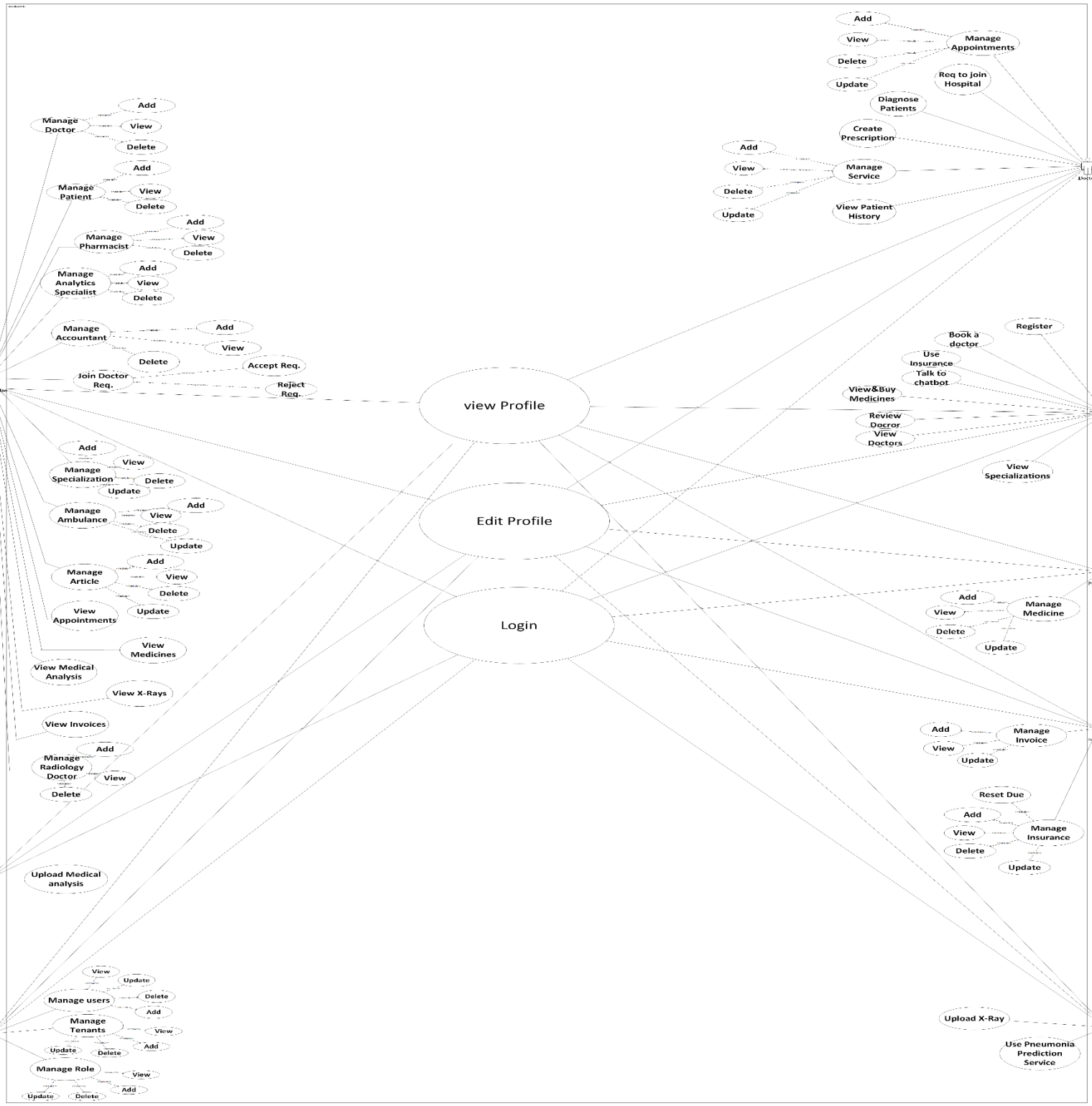
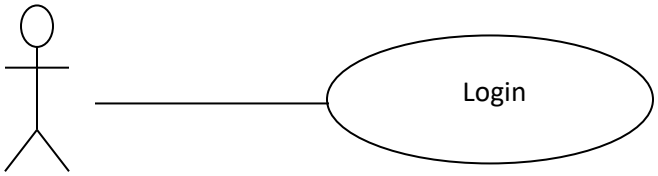


Figure 1: Use Case Diagram

2.7.2. Use Case Scenarios:

Use case: Login
Diagram :



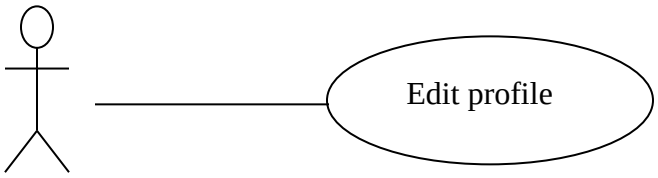
System User

Scenario:

Log in to the system	
Actor who initiates the use case	System User
Pre-condition	The actor already has an account
Basic path	-The system user enters user ID and Password in the custom field. -click on the login button. -the system validation the entered username and Password and logs the user into the system.
Post condition	The actor entered his account successfully
Alternative Paths	If the system user enters invalid ID or Password, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	System User

Use case: Edit profile

Diagram :

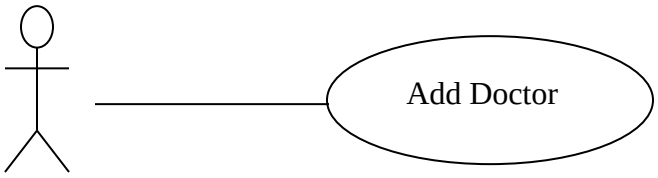


Scenario:

Edit profile	
Actor who initiates the use case	System User
Pre-condition	The actor must be logged in to the system and has permission.
Basic path	By click on edit profile in user profile can edit his data such as their name, age, profile picture, email, and password from the form opened.
Post condition	The actor can edit this profile.
Alternative Paths	If the system user enters invalid or wrong data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	System User

Use case: Add Doctor

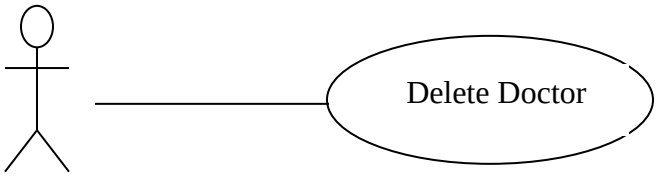
Diagram :



Scenario:

Add Doctor	
Actor who initiates the use case	Administrator
Pre-condition	The administrator had signed in to his profile (system), has permeation and the doctor give his data to the administrator.
Basic path	When the administrator login, he can add the Doctor to the website database and give him an account to make the doctor can do many functions.
Post condition	The doctor is added and has an account.
Alternative Paths	If the Administrator enters invalid doctor data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Administrator

Use case: Delete Doctor
Diagram:



Administrat

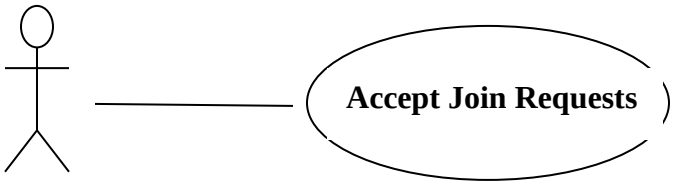
Scenario:

Delete Doctor	
Actor who initiates the use case	Administrator
Pre-condition	The administrator had signed in to his profile (system), has permeation and the doctor give his data to the administrator

Basic path	When the administrator login, he can Delete the Doctor from the website database by click on delete button on the doctor row.
Post condition	The administrator deletes all the doctors
Alternative Paths	If the Administrator in specific doctor page he can also delete doctor by click on delete button on the page.
Actor who benefits from the use case	Administrator

Use case: Accept Join Requests

Diagram:

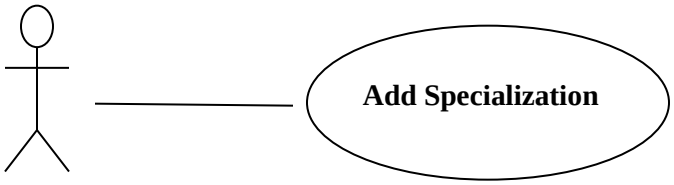


Administrat

Scenario:

Accept Join Requests	
Actor who initiates the use case	Administrator
Pre-condition	-The system is allowed to accept doctor join request -The admin must be log in system -The admin has permission to do this
Basic path	The system is available to accept 'doctors requests by the admin, and requests for the doctor to join the site to work on it
Post condition	Admin accepts doctor join request
Alternative Paths	If the doctor is already existing, the system shall display an appropriate error message.
Actor who benefits from the use case	Administrator

Use case: Add Specialization
Diagram:

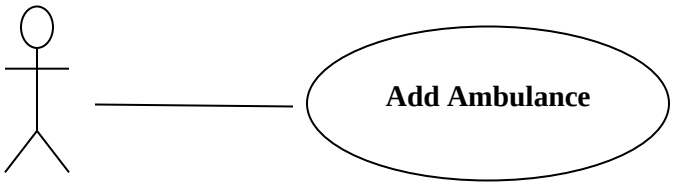


Administrat

Scenario:

Add Specialization	
Actor who initiates the use case	Administrator
Pre-condition	Admin had signed into his profile (system) and has a permeation
Basic path	When the admin login, he can add a specialization to the website database, and make patient can access to view it
Post condition	specialization is added
Alternative Paths	If the Administrator enters invalid Specialization data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Administrator

Use case: Add Ambulance
Diagram:



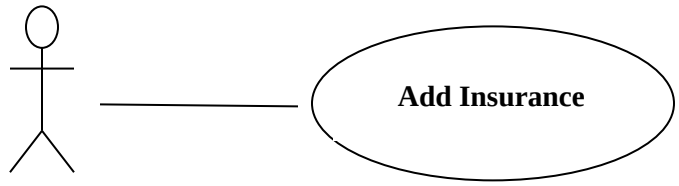
Administrat

Scenario:

Add Ambulance	
Actor who initiates the use case	Administrator
Pre-condition	Admin had signed into his profile (system) and has a permeation
Basic path	When the admin login, he can add an Ambulance to the website database, and make patient can access to view it
Post condition	Ambulance is added
Alternative Paths	If the Administrator enters invalid or wrong Ambulance data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Administrator

Use case: Add Insurance

Diagram:



Administrat

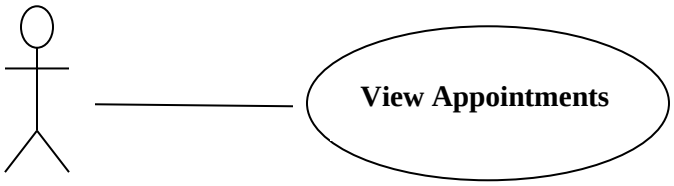
Scenario:

Add Insurance	
Actor who initiates the use case	Accountant
Pre-condition	Accountant had signed into his profile (system) and has a permeation
Basic path	When the accountant login, he can add Insurance to the website database, and make patient can access to view it
Post condition	Insurance is added
Alternative Paths	If the Accountant enters invalid Insurance data, the system shall display an appropriate error

	message and returns them back to the first in the series of operation
Actor who benefits from the use case	Accountant

Use case: View Appointments

Diagram:

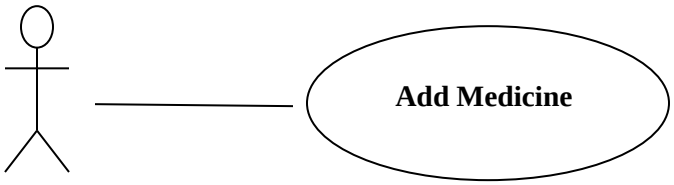


Administrat

Scenario:

View Appointments	
Actor who initiates the use case	Administrator
Pre-condition	The user had login into his profile (system) and has permission
Basic path	When the User login, he can view appointment details like “name, date and some other info
Post condition	User view details about all filtered the appointment
Alternative Paths	If the required Appointments is not found, the system should display not found message
Actor who benefits from the use case	Administrator

Use case: Add Medicine
Diagram:

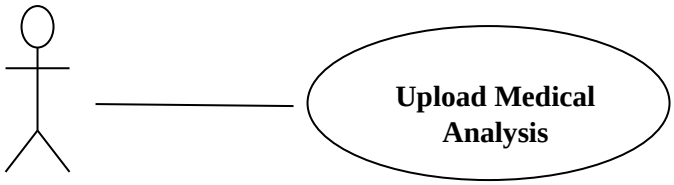


Pharmacist

Scenario:

Add Medicine	
Actor who initiates the use case	Pharmacist
Pre-condition	The pharmacist must log in system and verified to add pharmacist.
Basic path	Adding Medicine is behaviour done by a pharmacist. When the pharmacist adds a medicine, He'll add details for the medicine such as the name and image etc.
Post condition	The medicine will be saved to database.
Alternative Paths	If the Pharmacist enters invalid or wrong Medicine data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Pharmacist

Use case: Upload Medical Analysis
Diagram:



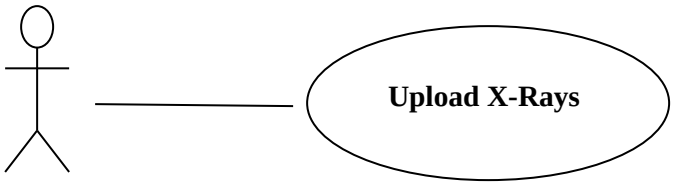
Analytics specialist

Scenario:

Upload Medical Analysis	
Actor who initiates the use case	Analytics specialist.
Pre-condition	The Analytics specialist must log in system and type of medical analysis is exist
Basic path	After Analytics specialist finish medical analysis, he uploads result of medical analysis as file on system
Post condition	The medical analysis result will be saved to database
Alternative Paths	If the Analytics specialist enters empty Medical Analysis, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Analytics specialist.

Use case: Upload X-Rays

Diagram:



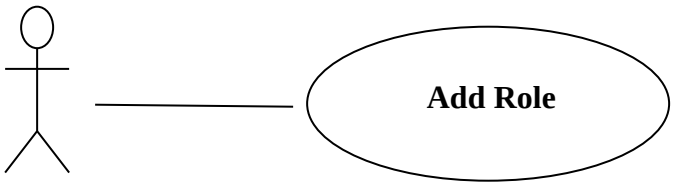
Radiology doctor.

Scenario:

Upload X-Rays	
Actor who initiates the use case	Radiology doctor.
Pre-condition	The Radiology doctor must log in system and type of x-ray is exist
Basic path	After Radiology doctor finish x-ray, he uploads result of x-ray as file on system
Post condition	The x-ray result will be saved to database

Alternative Paths	If the Radiology doctor enters empty X-Rays, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Radiology doctor.

Use case: Add Role
Diagram:

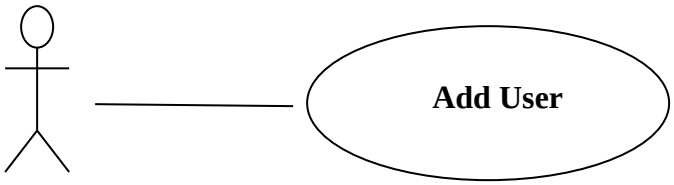


The Hospital Manager

Scenario:

Add Role	
Actor who initiates the use case	The Hospital Manager
Pre-condition	The system is allowed to add roles
Basic path	The Hospital Manager is available to add roles and describe the privilege for each user. The role is the validities that the user takes, whoever (Admin, Pharmacist, Doctor, Analytics Specialist, Radiology Doctor, Accountant or Patient) to perform certain tasks
Post condition	Role is created successfully
Alternative Paths	If the Hospital Manager enters invalid or wrong role data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	The Hospital Manager

Use case: Add User
Diagram:

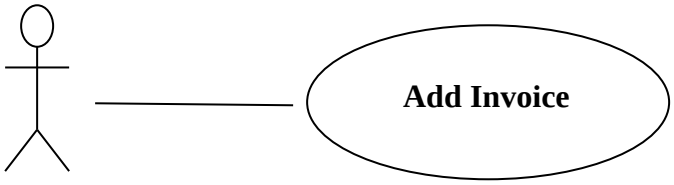


The Hospital Manager

Scenario:

Add User	
Actor who initiates the use case	The Hospital Manager
Pre-condition	-The system is allowed to add users -The Hospital Manager must be logged in the system -User hasn't been created yet.
Basic path	The Hospital Manager is available to add user and describe his privileges
Post condition	User is added successfully
Alternative Paths	If the Hospital Manager enters invalid or wrong User data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	The Hospital Manager

Use case: Add Invoice
Diagram:



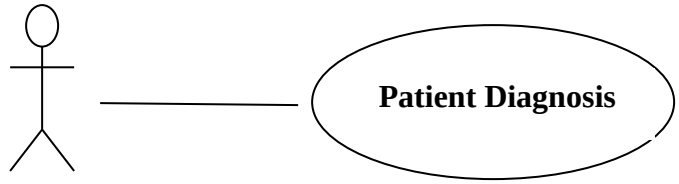
Accountant

Scenario:

Add Invoice	
Actor who initiates the use case	Accountant
Pre-condition	-The system is allowed to create invoices -The accountant must be logged in the system -Patients and Doctors have been created.
Basic path	Accountant must add invoices and manage the financial part of the hospital
Post condition	Invoice is added successfully
Alternative Paths	If the accountant enters invalid or wrong invoice data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Accountant

Use case: Patient Diagnosis

Diagram:



Doctor

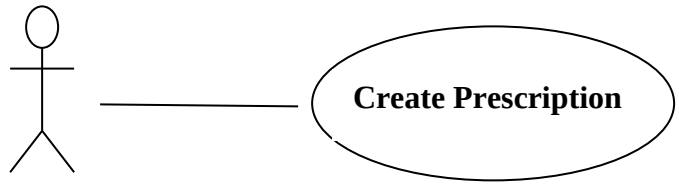
Scenario:

Patient Diagnosis	
Actor who initiates the use case	Doctor
Pre-condition	-The system is allowed to add diagnosis -The Doctor must be logged in the system -Patient , Radiology Doctor ,Analytics Specialist and Doctors have been created. -The Doctor should view patient history.
Basic path	Doctor must add patient diagnosis and can send them to the specialist.

Post condition	Diagnosis is added successfully
Alternative Paths	If the doctor enters invalid or wrong Diagnosis data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Doctor

Use case: Create Prescription

Diagram:

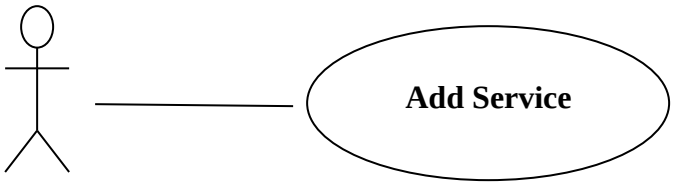


Doctor

Scenario:

Create Prescription	
Actor who initiates the use case	Doctor
Pre-condition	-The system is allowed to add prescription -The Doctor must be logged in the system -Patients and Doctors have been created. -Patient Diagnosis has been created
Basic path	Doctor must create prescription after diagnose patient and add it in patient profile.
Post condition	Prescription is added successfully
Alternative Paths	If the doctor enters invalid or wrong Prescription data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Doctor

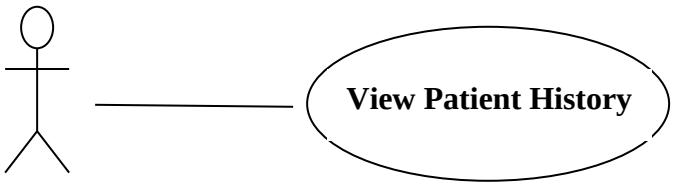
Use case: Add Service
Diagram:



Scenario:

Add Service	
Actor who initiates the use case	Doctor
Pre-condition	-The system is allowed to add service. -The Doctor must be logged in the system -The Doctor has been created
Basic path	By click on add service button will open form the doctor type service details and click submit then the service is available for patient.
Post condition	Service is added successfully
Alternative Paths	If the doctor enters invalid or wrong service data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Doctor

Use case: View Patient History
Diagram:

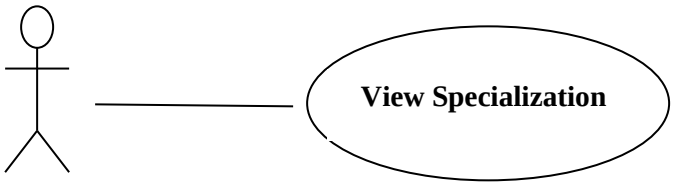


Scenario:

View Patient History	
Actor who initiates the use case	Doctor
Pre-condition	-The system is allowed to view patient history. -The Doctor must be logged in the system -The Doctor and patient have been created
Basic path	After open specific patient profile doctor can view patient history to help doctor in diagnoses
Post condition	List of patient's history is viewed successfully
Alternative Paths	If the required Patient History is not found, the system should display not found message
Actor who benefits from the use case	Doctor

Use case: View Specialization

Diagram:



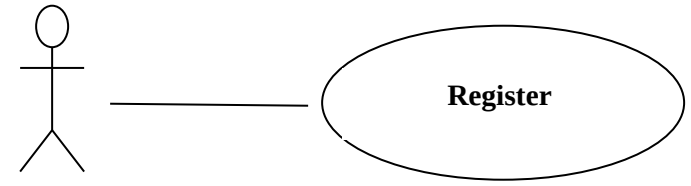
Patient, Administrator, Hospital

Scenario:

View Specialization	
Actor who initiates the use case	Patient, Administrator, Hospital manager
Pre-condition	User had login to his profile (system) and has permission
Basic path	When user login, he can filter and view specialization details like "name"
Post condition	User view details about all filtered the specializations
Alternative Paths	If the required Specialization is not found, admin should add Specialization to view his details

Actor who benefits from the use case	Patient, Administrator, Hospital manager

Use case: Register
Diagram:

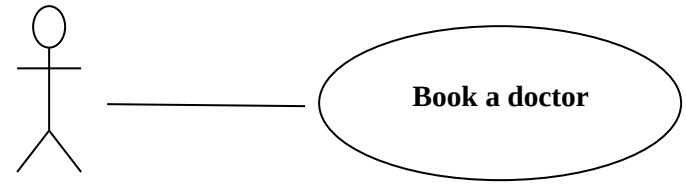


Patient

Scenario:

Register	
Actor who initiates the use case	Patient
Pre-condition	Patient don't have an account
Basic path	By click on register Button on navbar will open the registration form the user type his data in every field and by click on submit button the account is created and user can login now.
Post condition	User register successfully and can login into the system
Alternative Paths	If the system user enters invalid ID or not strong Password or invalid user data, the system shall display an appropriate error message and returns them back to the first in the series of operation
Actor who benefits from the use case	Patient

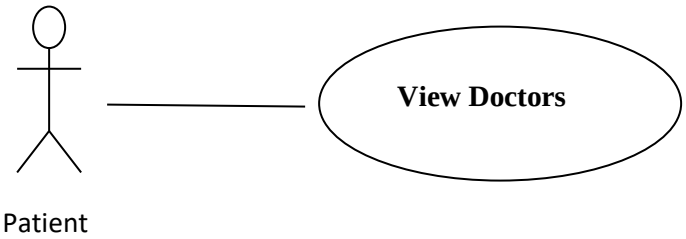
Use case: Book a doctor
Diagram:



Scenario:

Book a doctor	
Actor who initiates the use case	Patient
Pre-condition	Patient have an account and login into the system
Basic path	When user login, he can book a doctor and make an appointment based on specialty
Post condition	Appointment is made
Alternative Paths	If the doctor isn't available the system should display error message
Actor who benefits from the use case	Patient

Use case: View Doctors
Diagram:



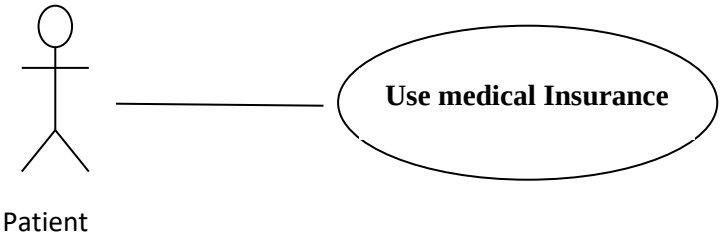
Scenario:

View Doctors	
Actor who initiates the use case	Administrator, Hospital manager

Pre-condition	User had login into his profile (system) and has permission
Basic path	When User login, he can filter and view doctor details like “name, specialty and some another info
Post condition	System User view details about all filtered the doctors
Alternative Paths	If the required doctor is not found, the system should display not found message
Actor who benefits from the use case	Administrator, Hospital manager

Use case: Use medical Insurance

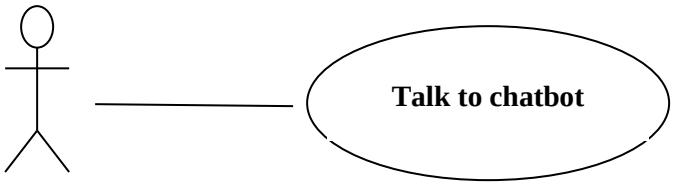
Diagram:



Scenario:

Use medical Insurance	
Actor who initiates the use case	Patient
Pre-condition	User logged in to his profile (system) and has permission
Basic path	When user login, he can use medical insurance to have a discount on price of medicines
Post condition	User dispends the medicine from the system
Alternative Paths	If patient did not add insurance the system should ask user to add insurance
Actor who benefits from the use case	Patient

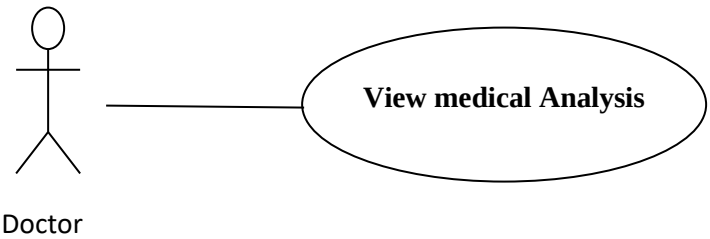
Use case: Talk to chatbot
Diagram:



Scenario:

Talk to chatbot	
Actor who initiates the use case	Patient
Pre-condition	User logged in to his profile (system) and has permission
Basic path	By click on chatbot icon chat will open and patient can type any question and bot will respond with the answer
Post condition	User talk to chatbot and be guided by it
Alternative Paths	If the user typed a wrong question, the system should ask the user to type another one
Actor who benefits from the use case	Patient

Use case: View medical Analysis
Diagram:

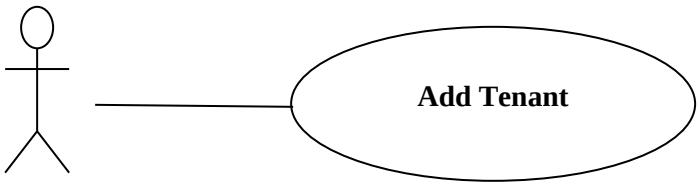


Scenario:

View medical Analysis	
Actor who initiates the use case	Doctor
Pre-condition	User logged in to his profile (system) and has permission
Basic path	When User login, he can view medical analysis details
Post condition	User view details about all filtered the medical analysis
Alternative Paths	If the required medical Analysis is not found, the system should display not found message
Actor who benefits from the use case	Doctor

Use case: Add Tenant

Diagram:



The Hospital Manager

Scenario:

Add Tenant	
Actor who initiates the use case	The Hospital Manager
Pre-condition	The system is allowed to add Tenant
Basic path	The Hospital Manager is available to add Tenant and describe the Tenant data for each Tenant like its domain, DB name and branch or tenant name.
Post condition	Tenant is created successfully
Alternative Paths	If the Hospital Manager enters invalid or wrong Tenant data, the system shall display an appropriate error message.
Actor who benefits from the use case	The Hospital Manager

2.7. Activity Diagrams

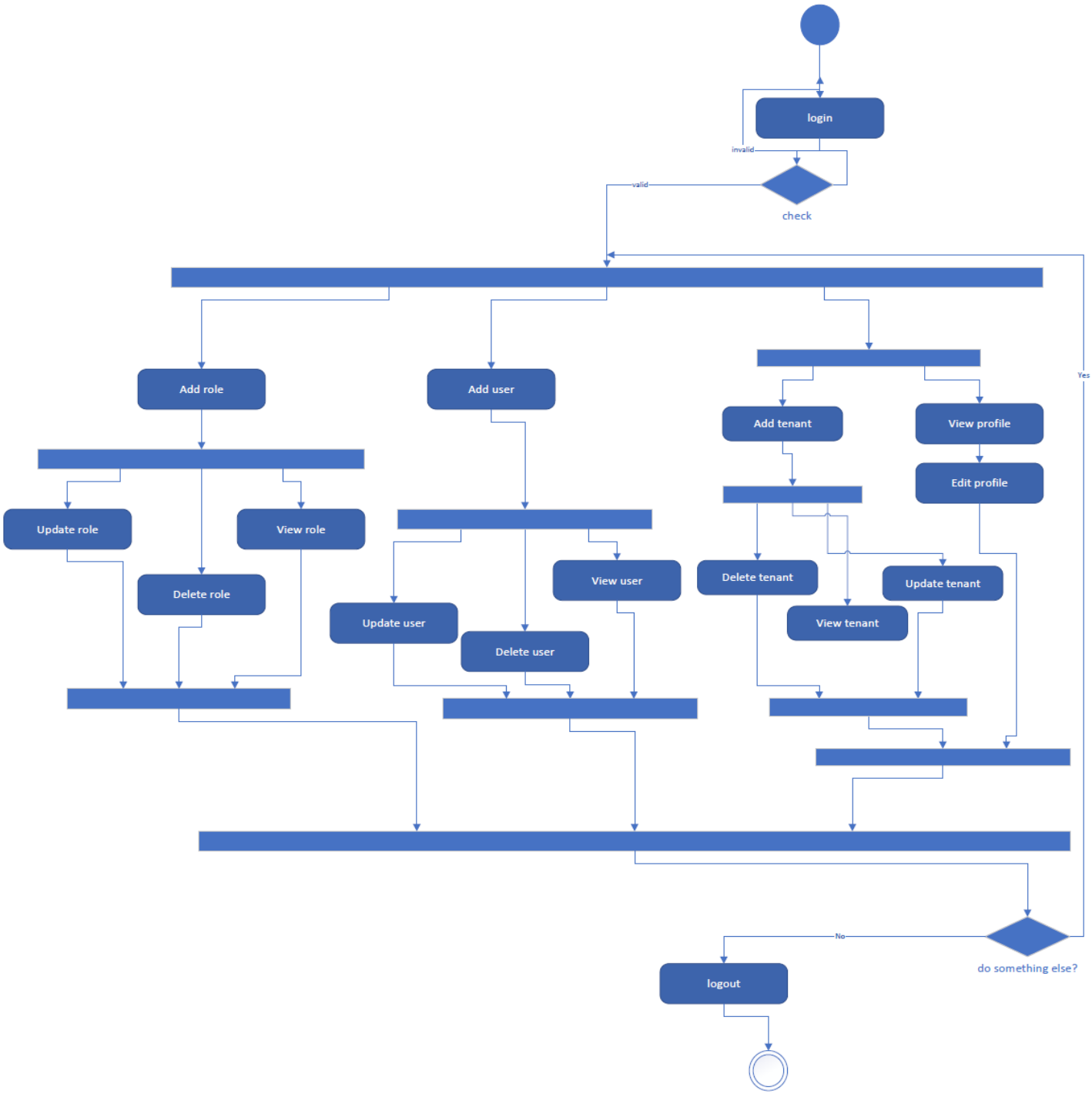


Figure 2: Hospital Manager Activity Diagram

Doctor

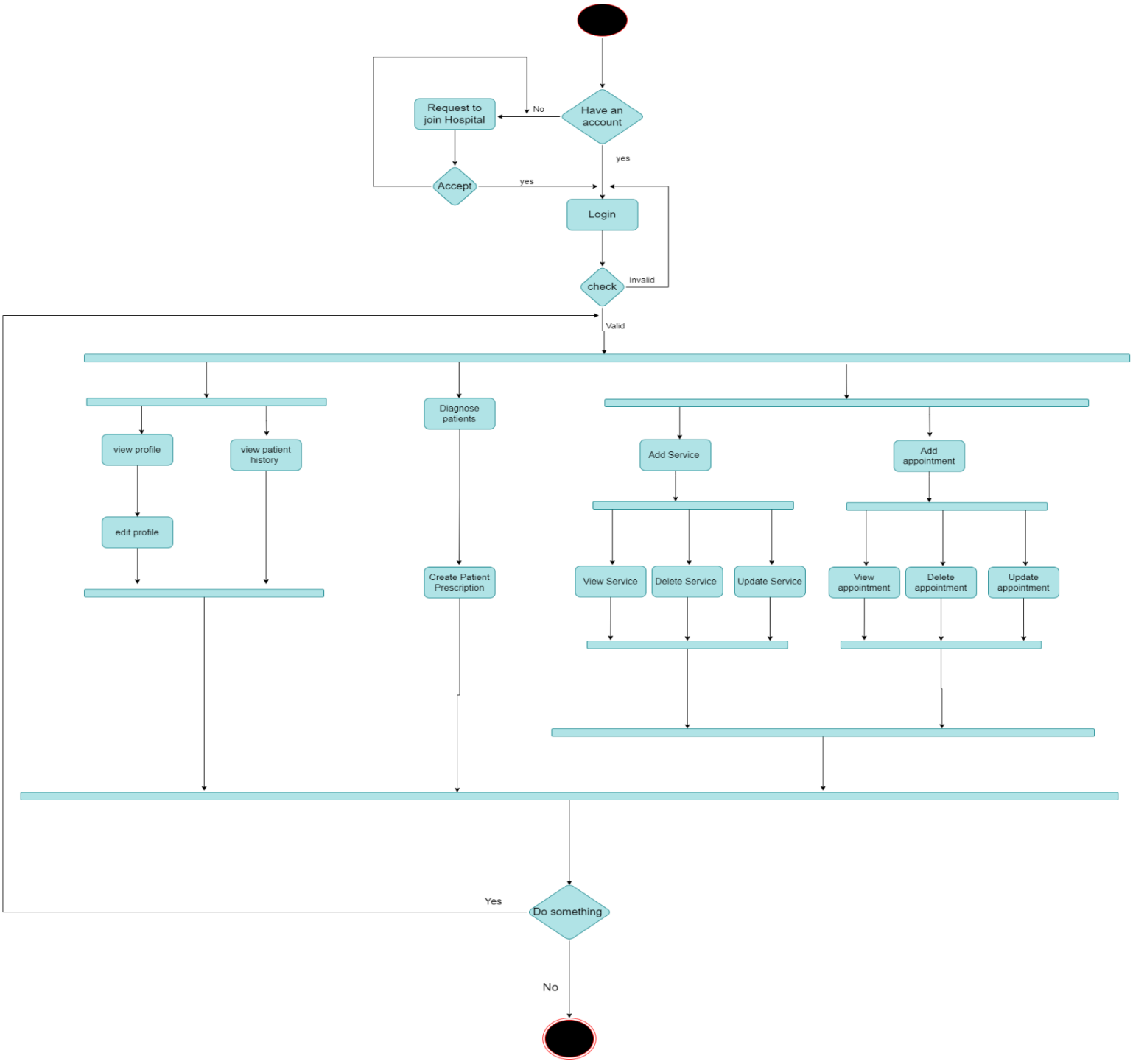


Figure 4: Doctor Activity Diagram

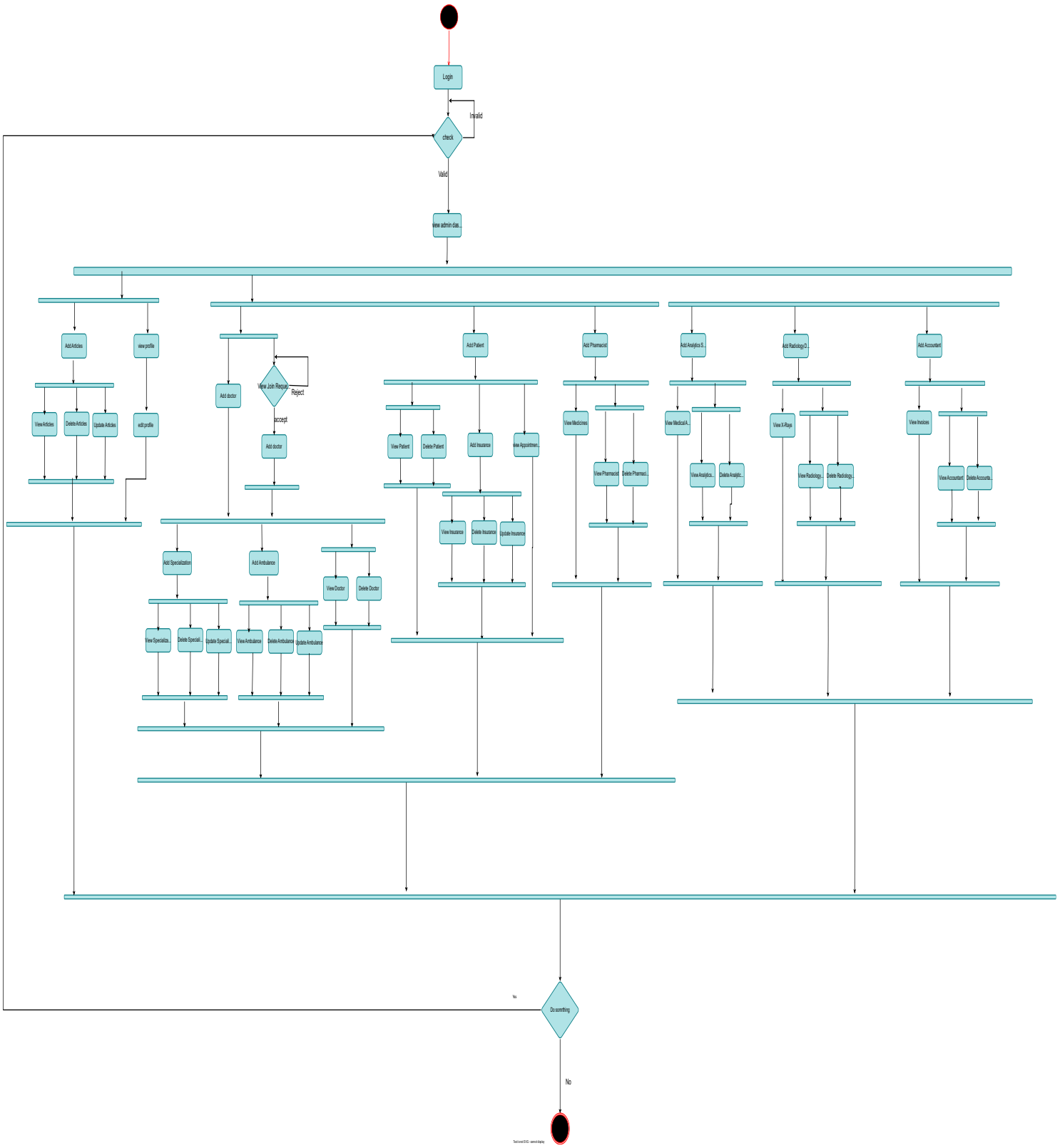


Figure 5: Admin Activity Diagram

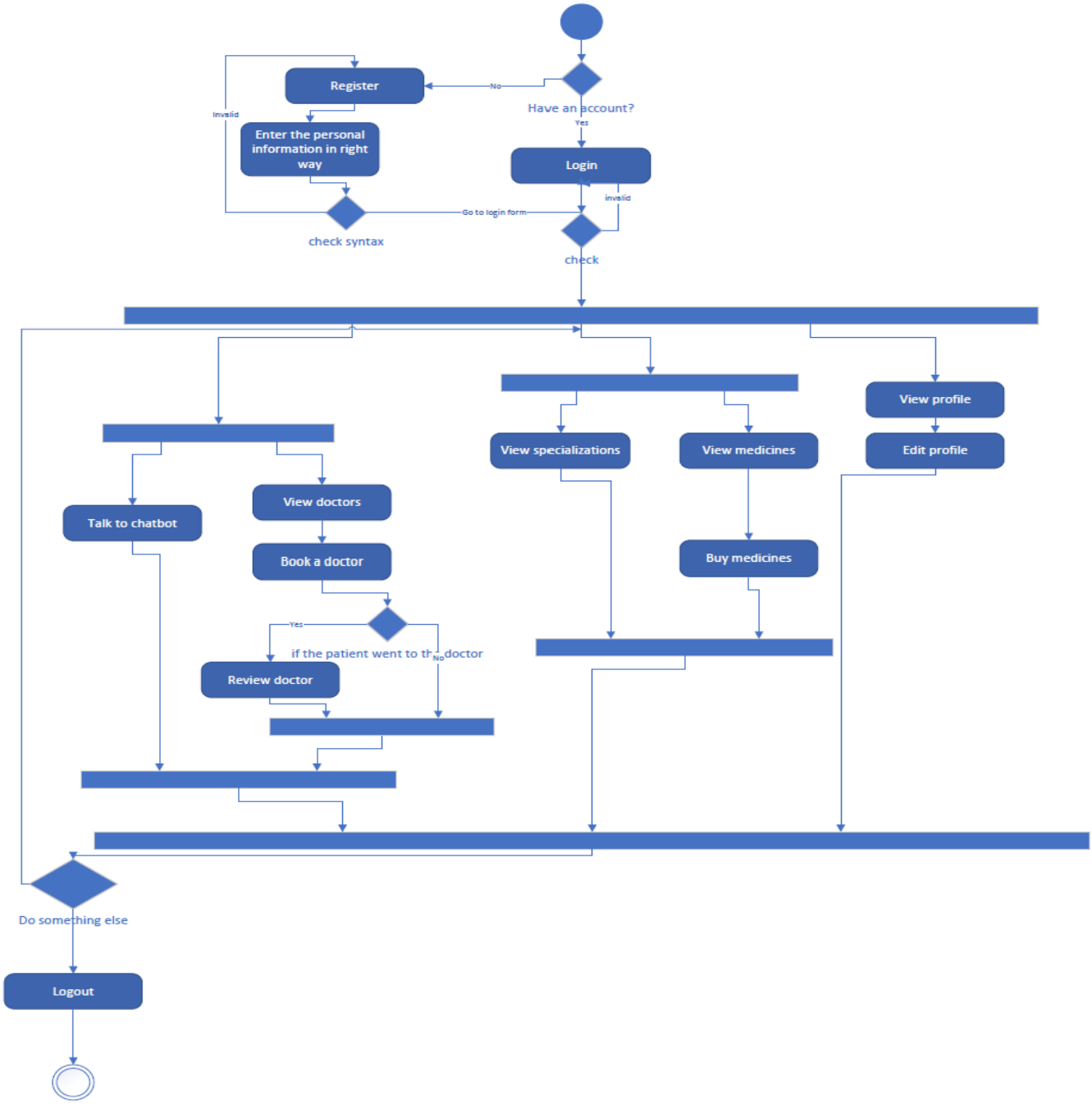


Figure 6: Patient Activity Diagram

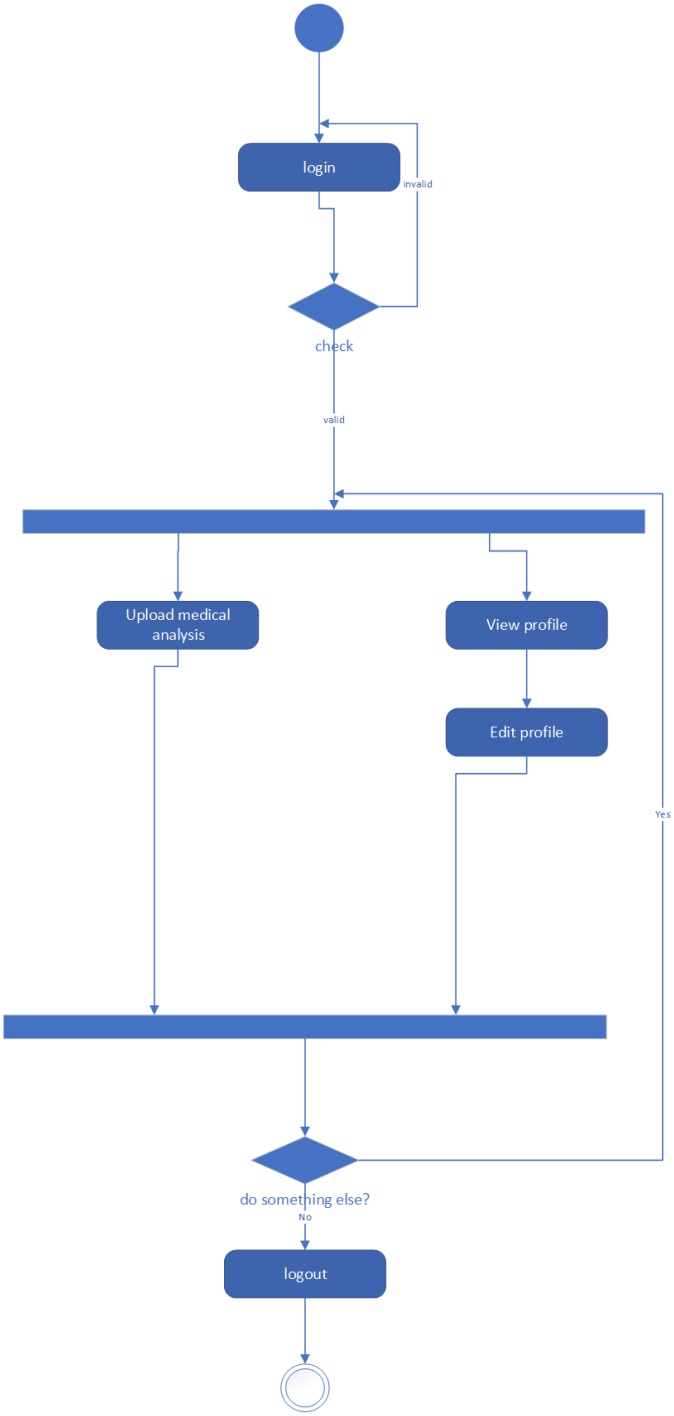


Figure 7: Analysis Specialist Activity Diagram

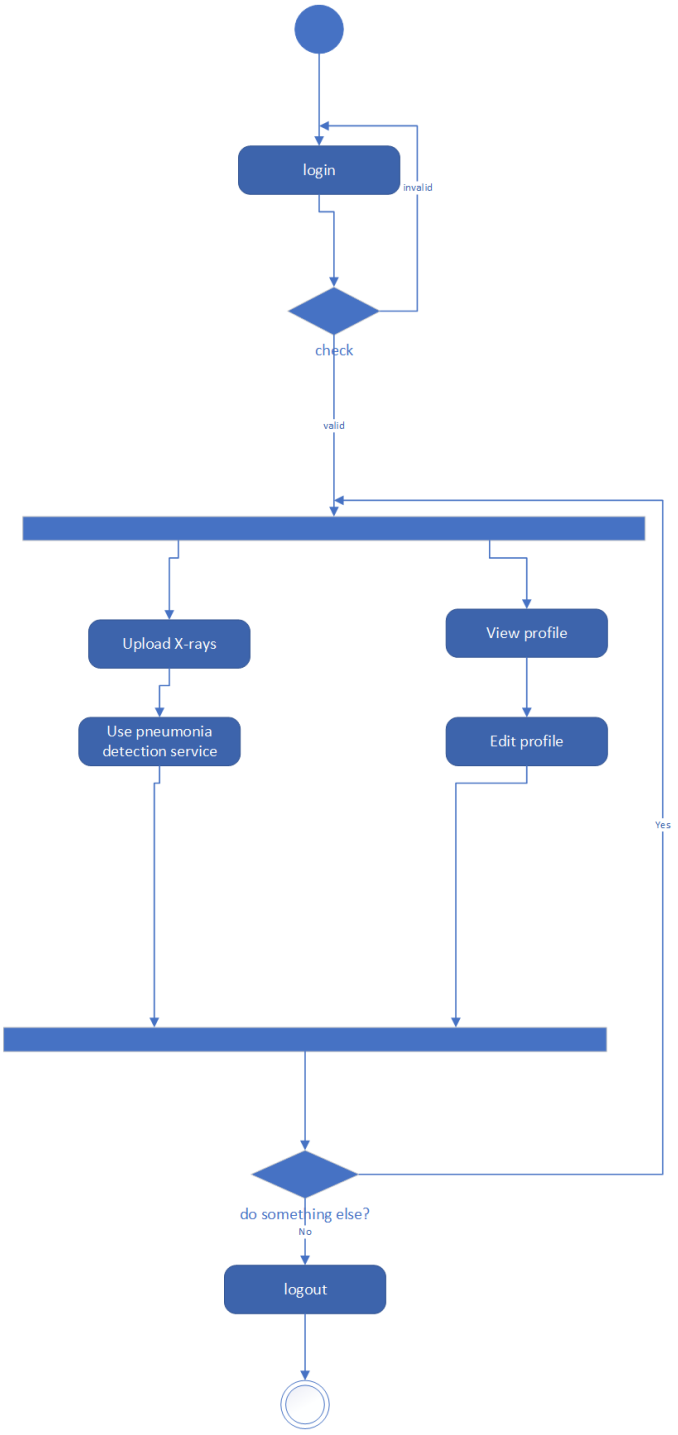


Figure 8: Radiology Doctor Activity Diagram

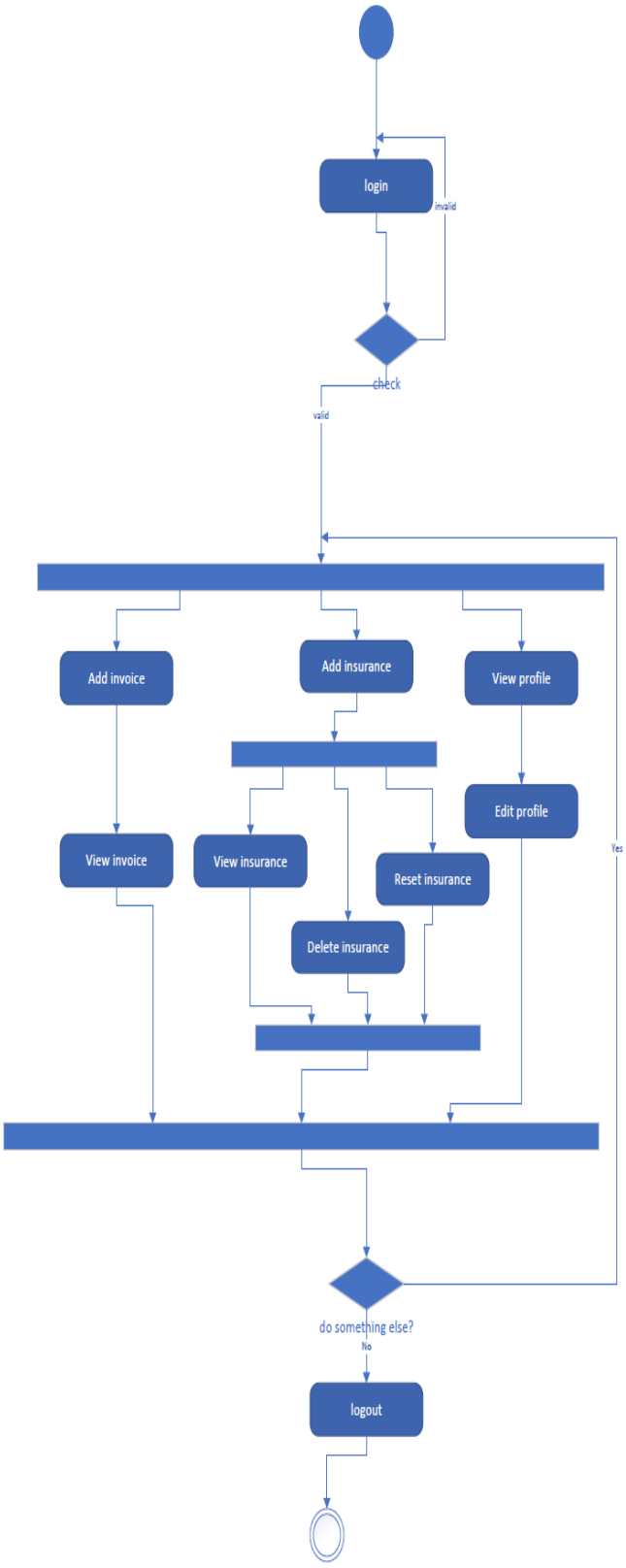


Figure 10: Accountant Activity Diagram

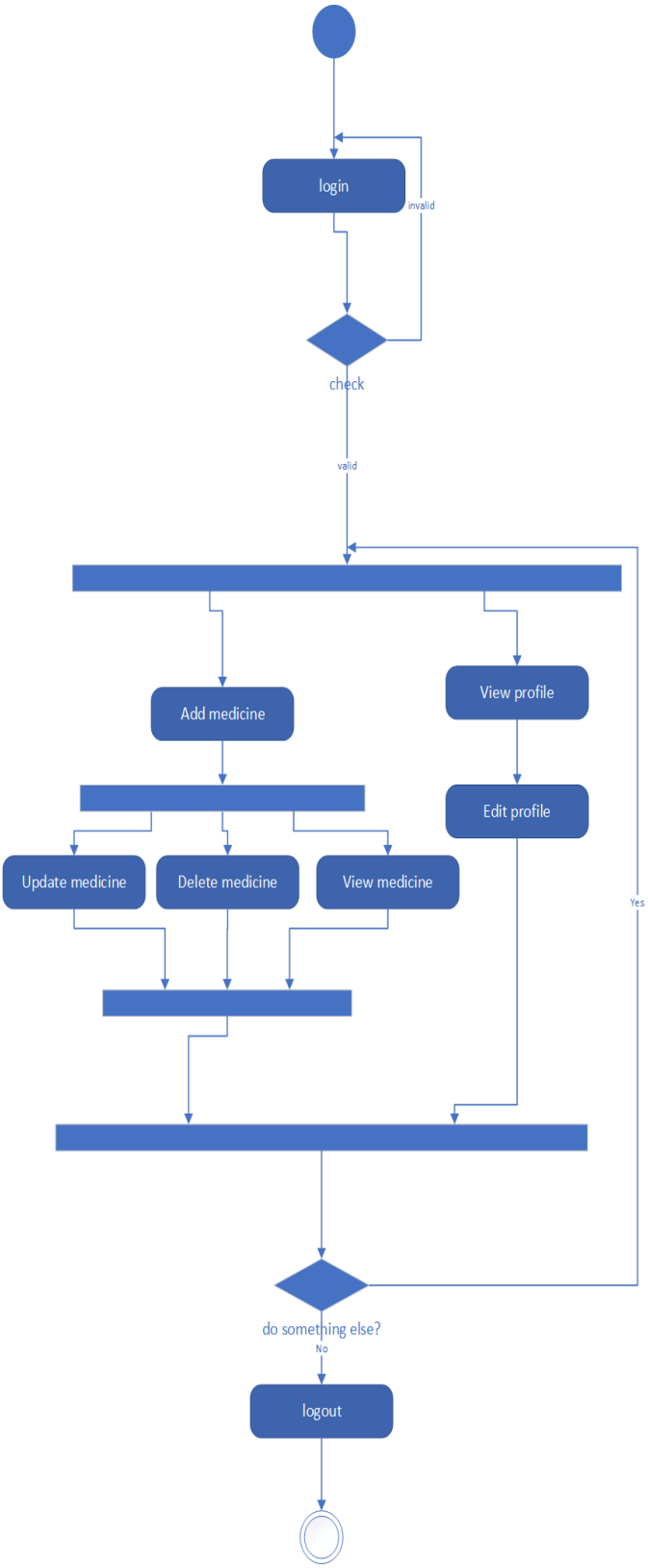


Figure 9: Pharmacy Activity Diagram

Chapter 3: Software Design

3.1. Design of database (Class Diagram)

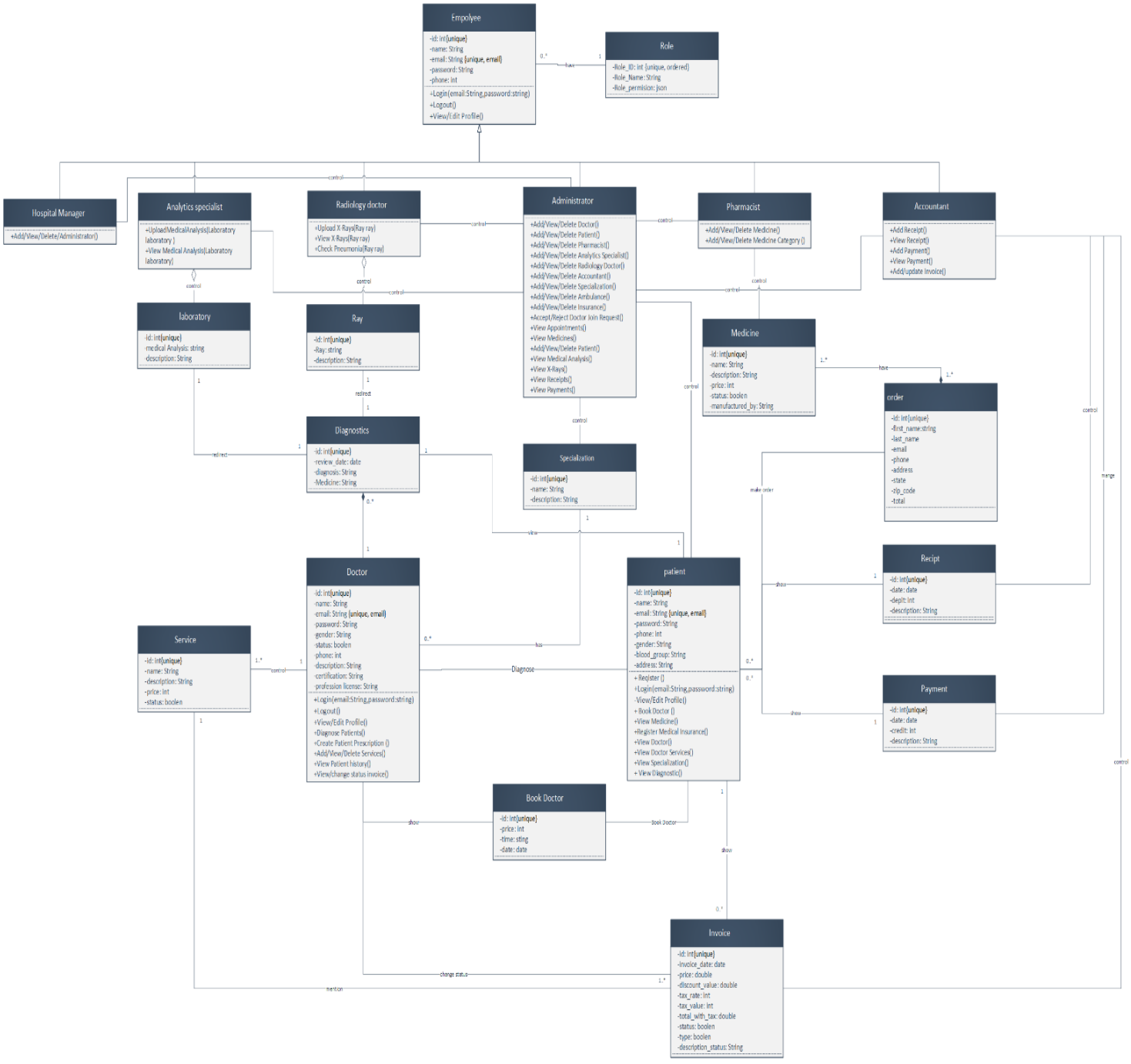


Figure 11: Class Diagram

3.2. Sequence Diagram

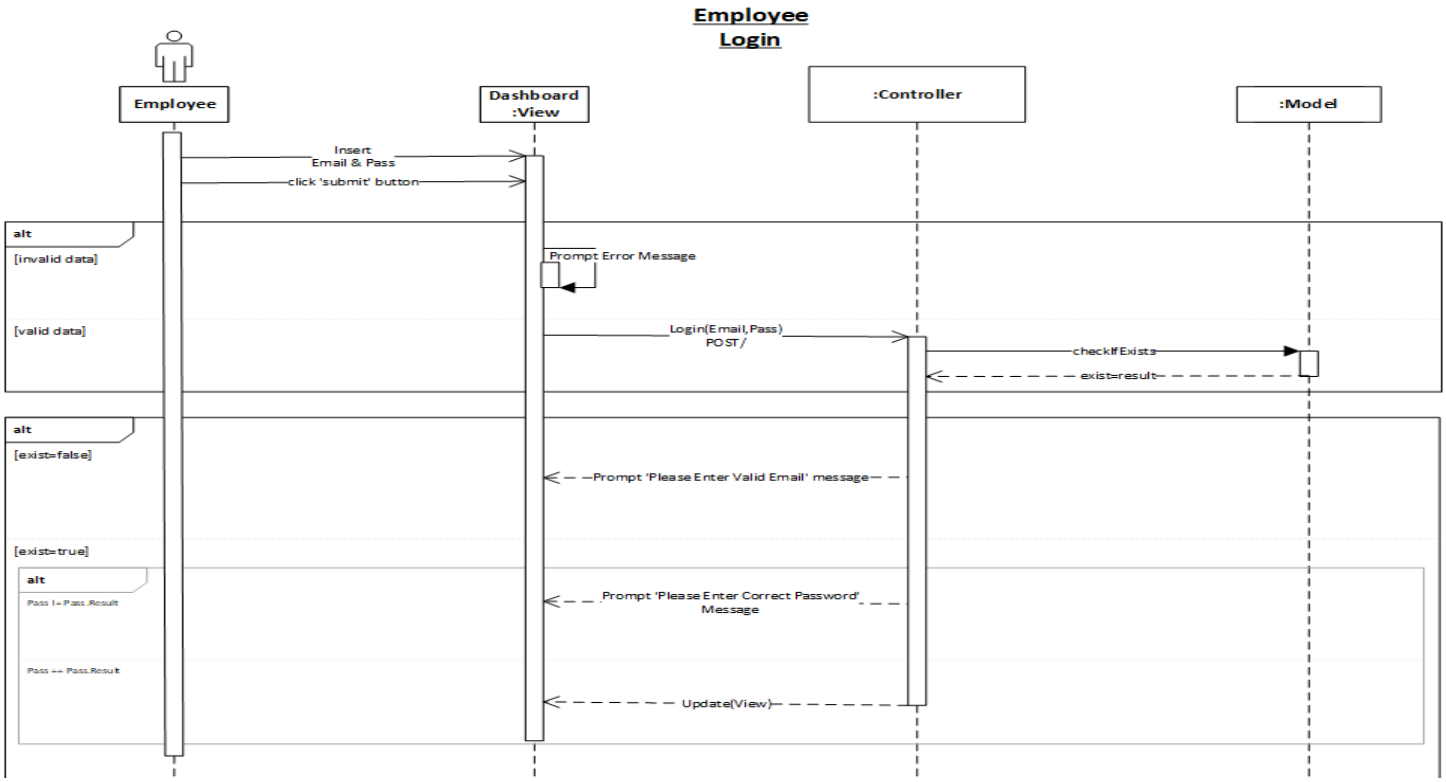


Figure 12: Employee Login Sequence Diagram

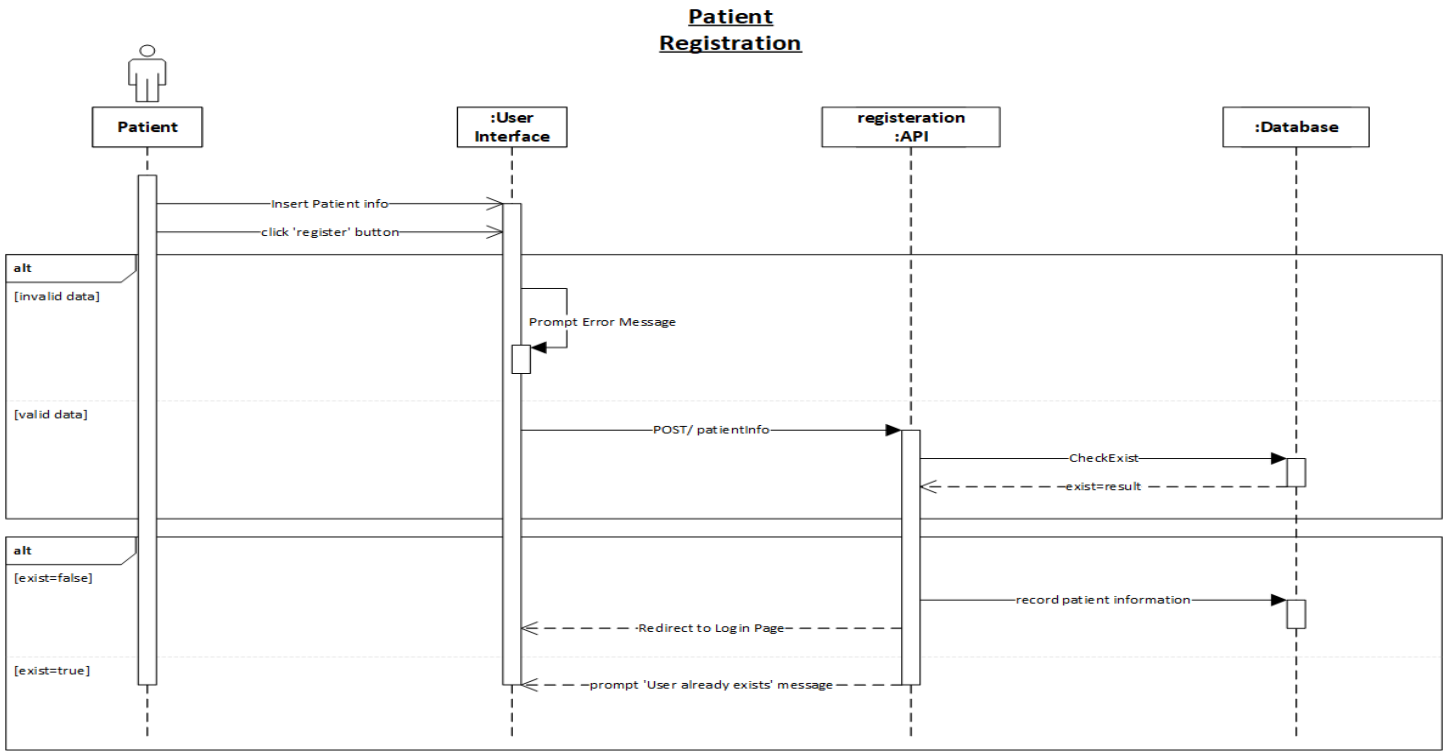


Figure 13: Patient Registration Sequence Diagram

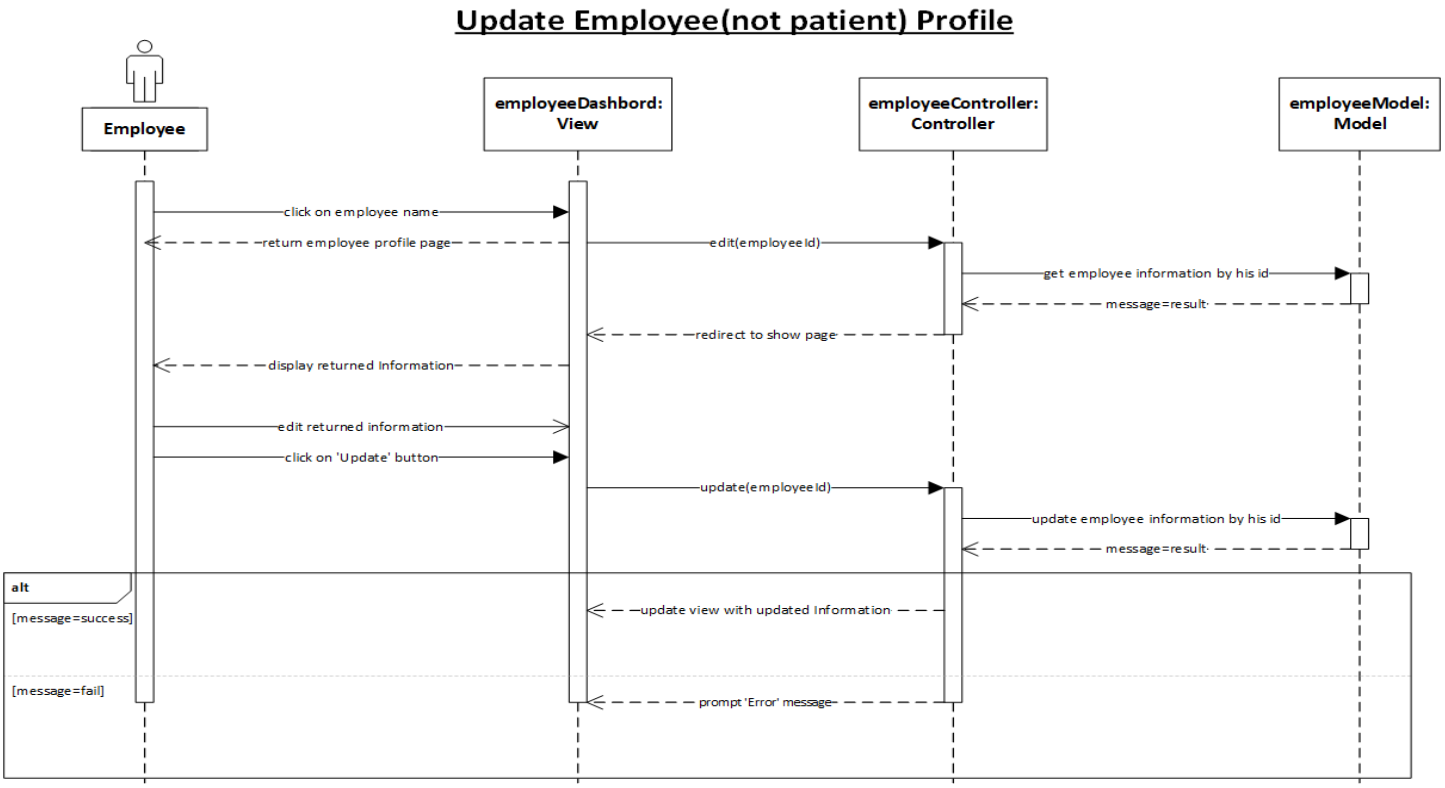


Figure 14: Update Employee Profile Sequence Diagram

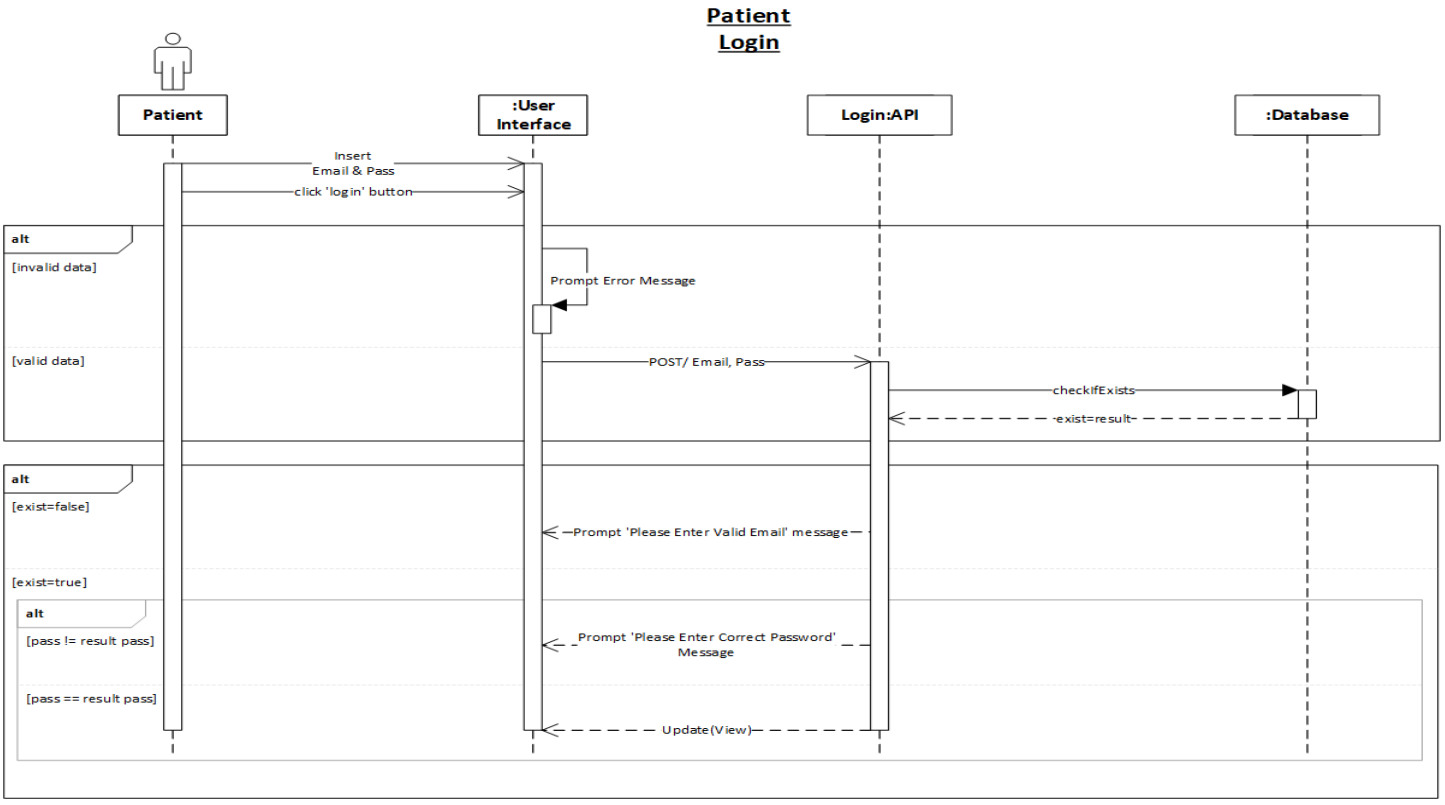


Figure 15: Patient Login Sequence Diagram

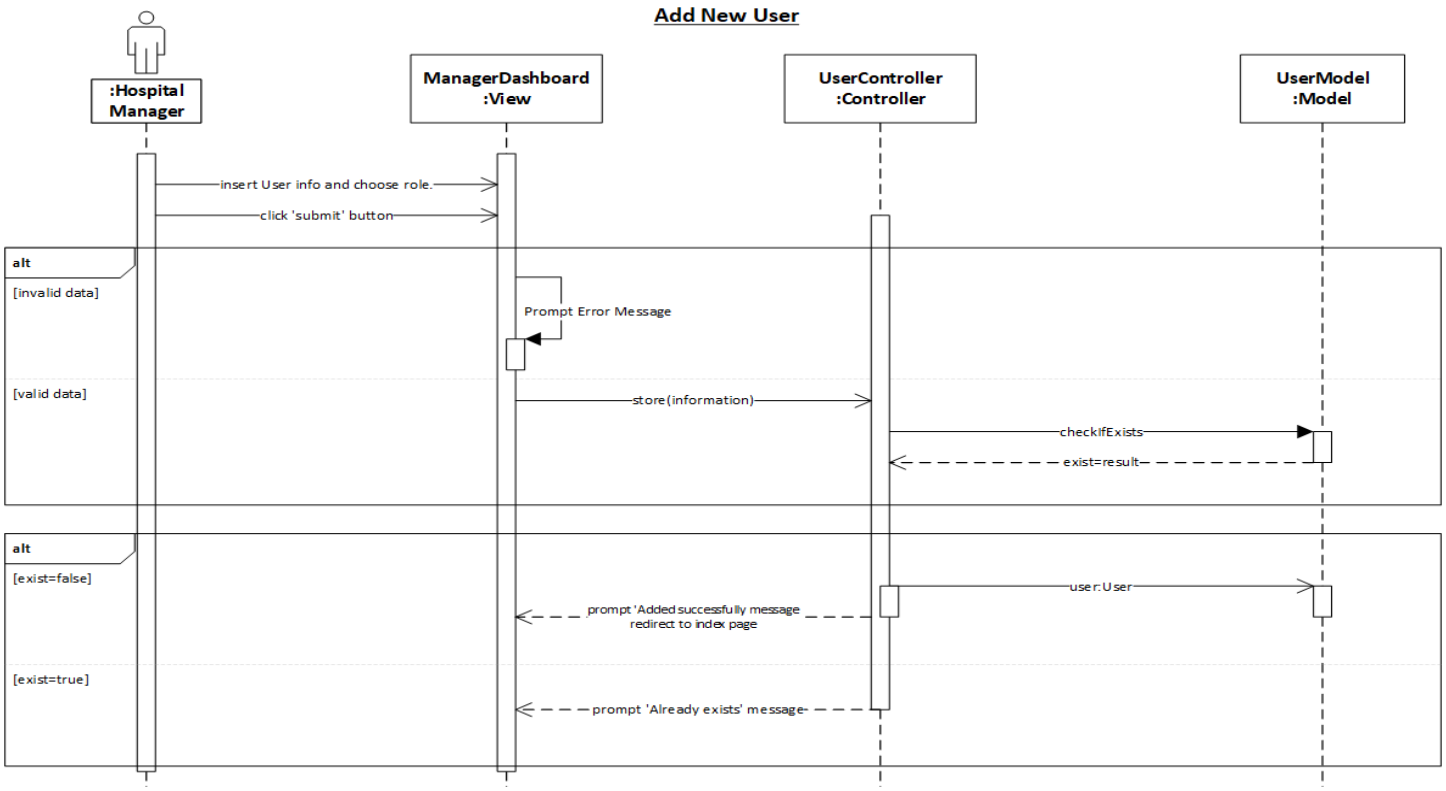


Figure 17: Hospital Manager Add User Sequence Diagram

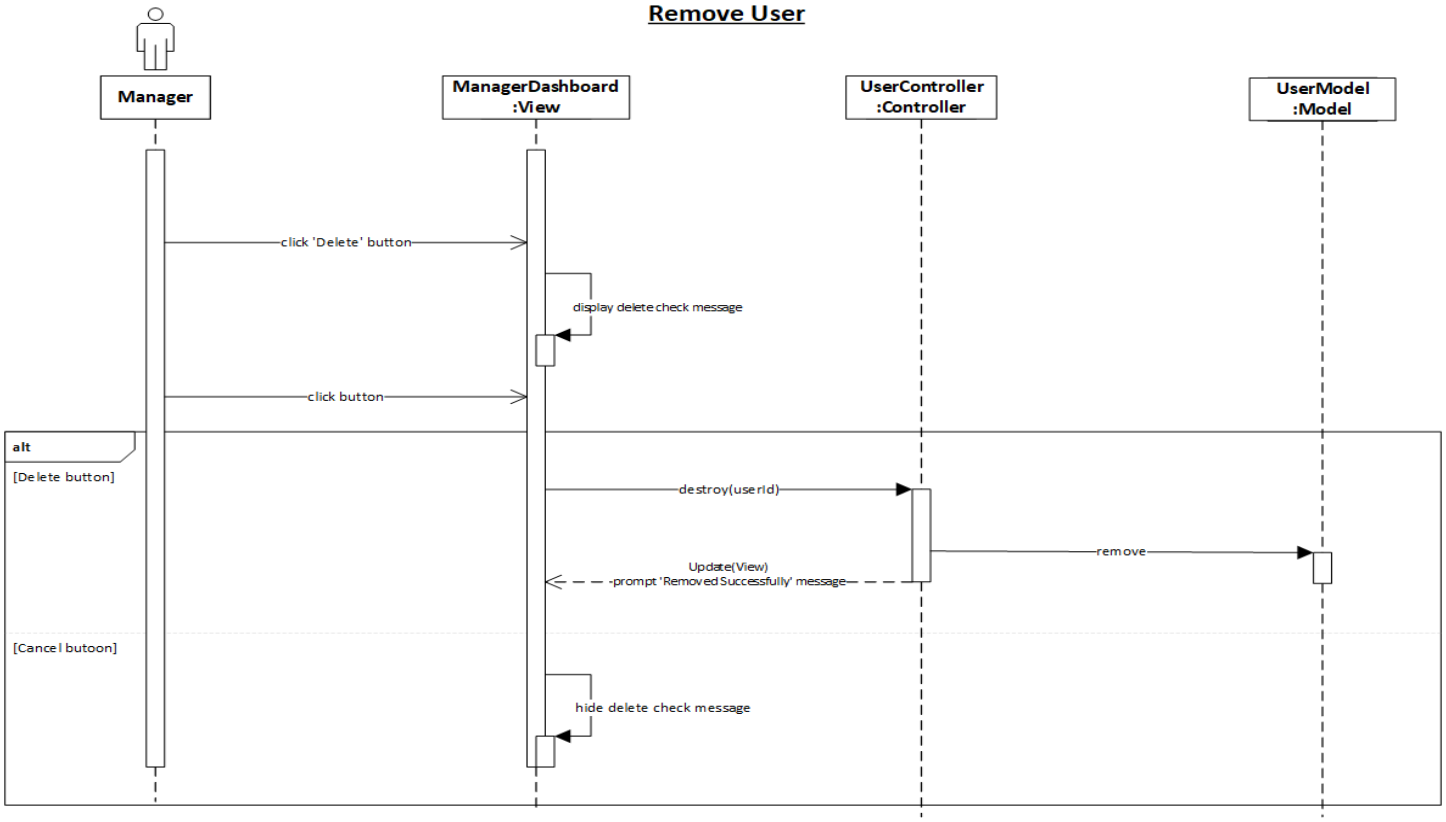


Figure 16: Hospital Manager Remove User Sequence Diagram

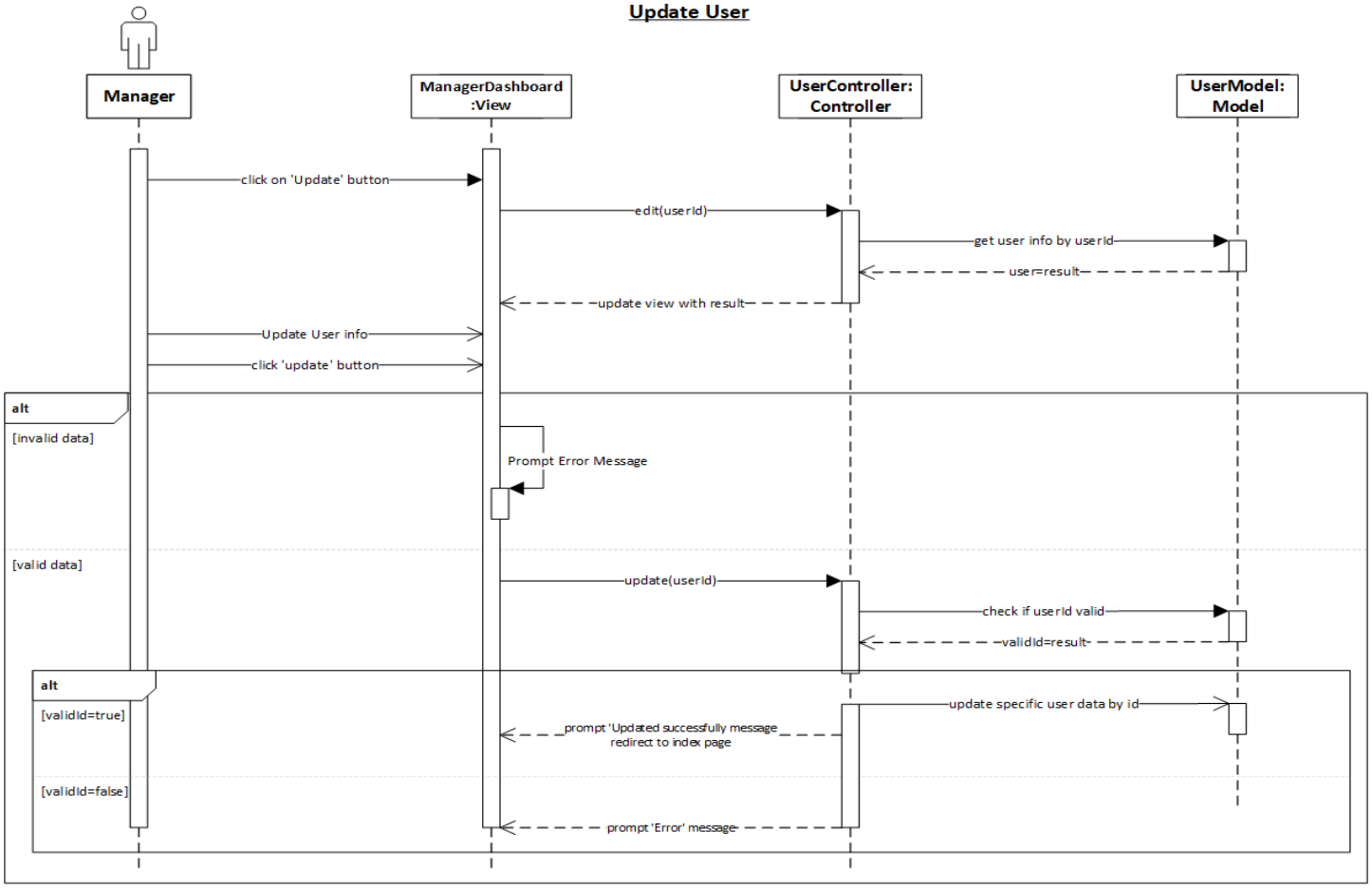


Figure 19: Hospital Manager Update User Sequence Diagram

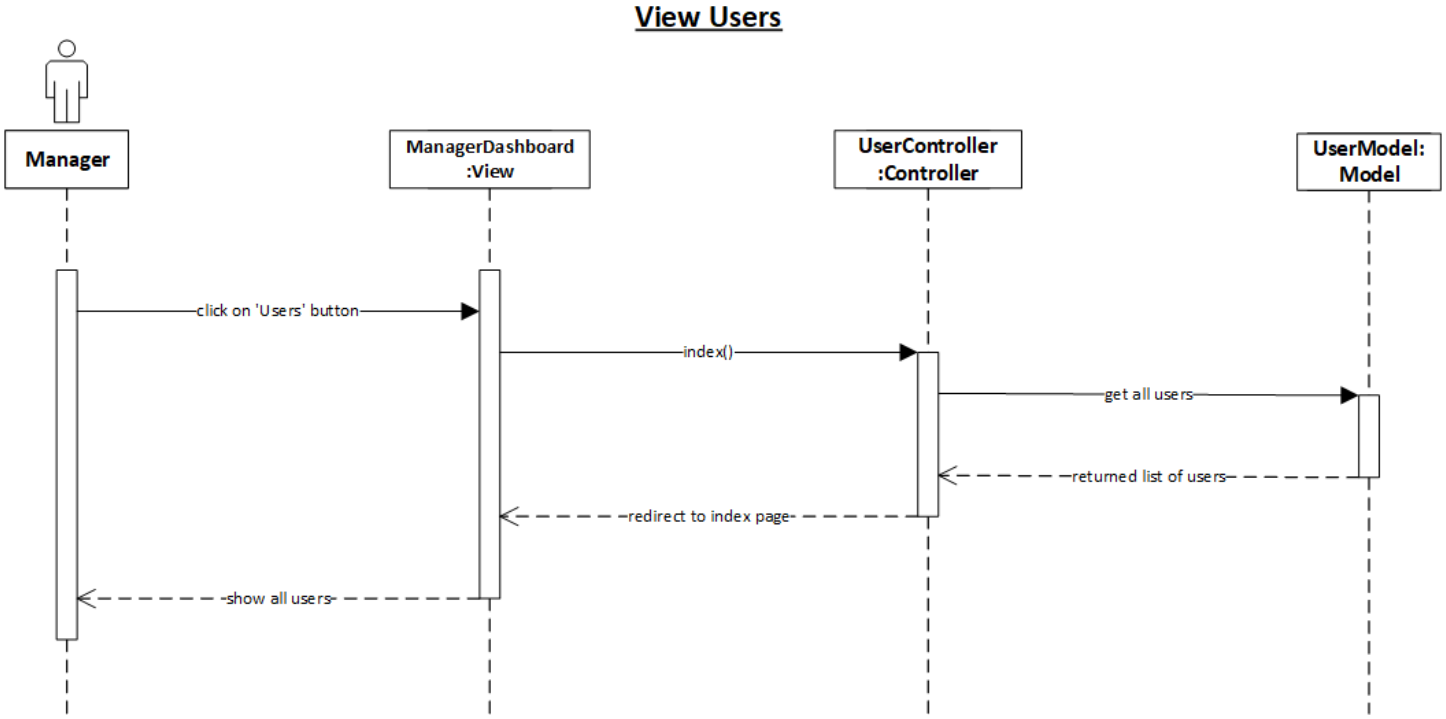


Figure 18:Hospital Manager View Users Sequence Diagram

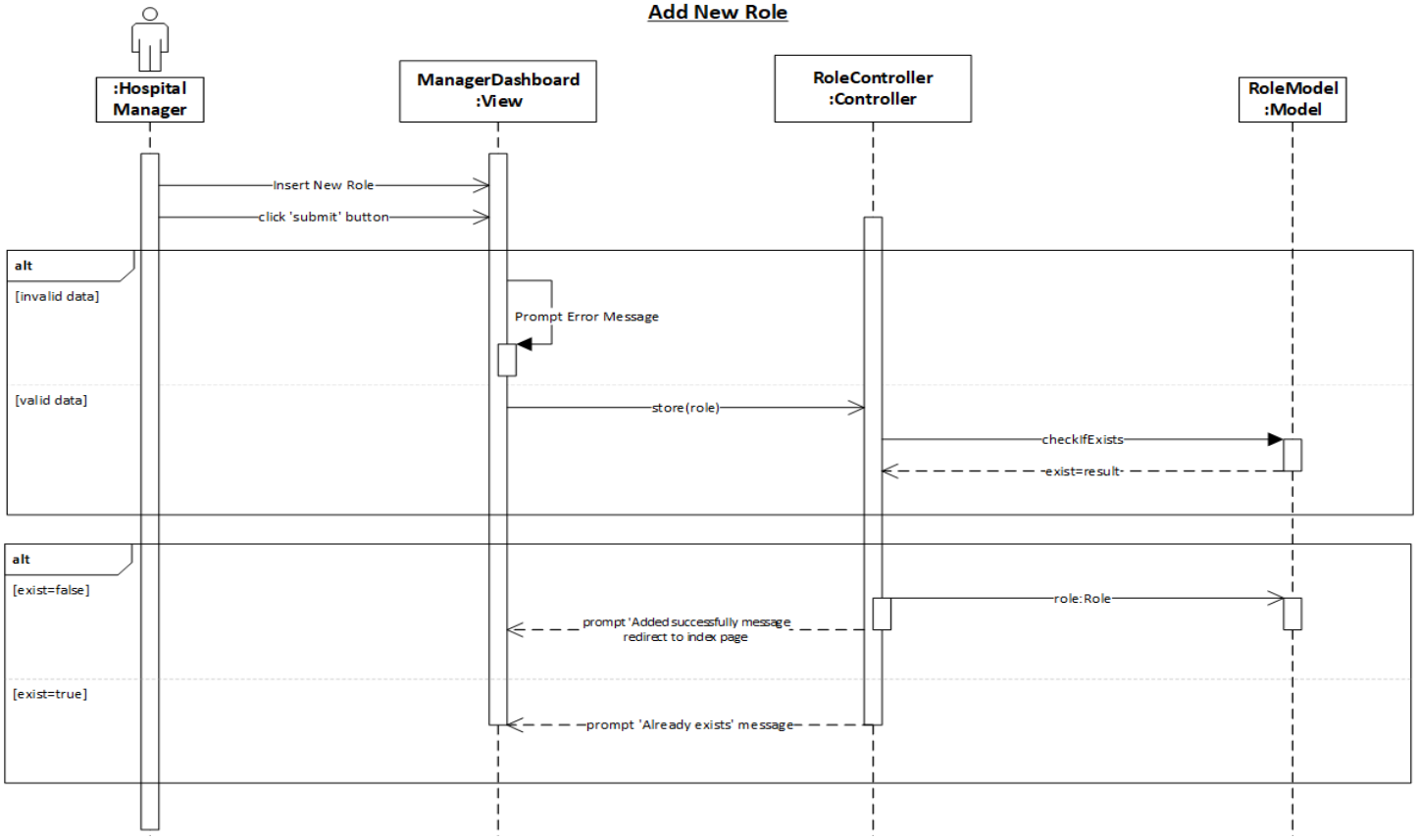


Figure 21: Hospital Manager Add Role Sequence Diagram

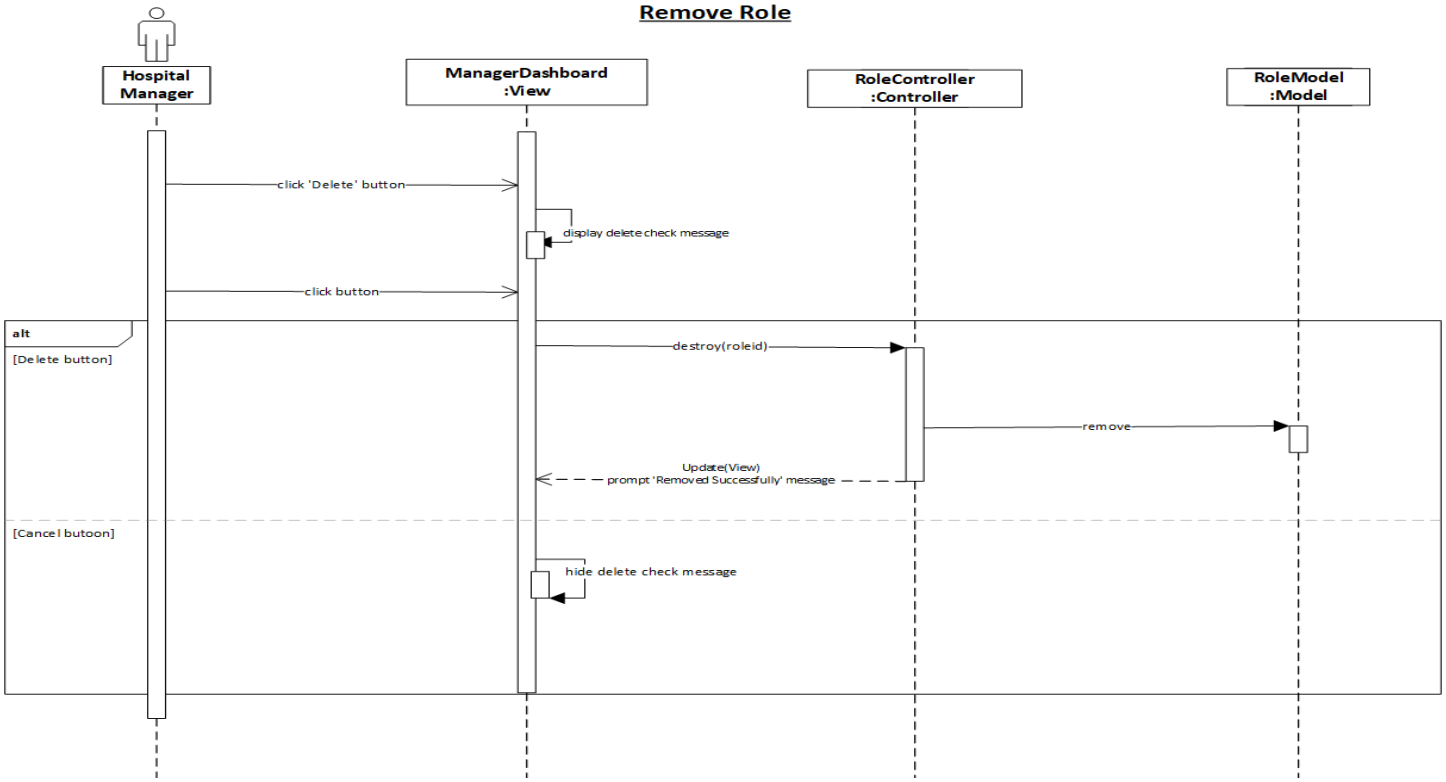


Figure 20: Hospital Manager Remove Role Sequence Diagram

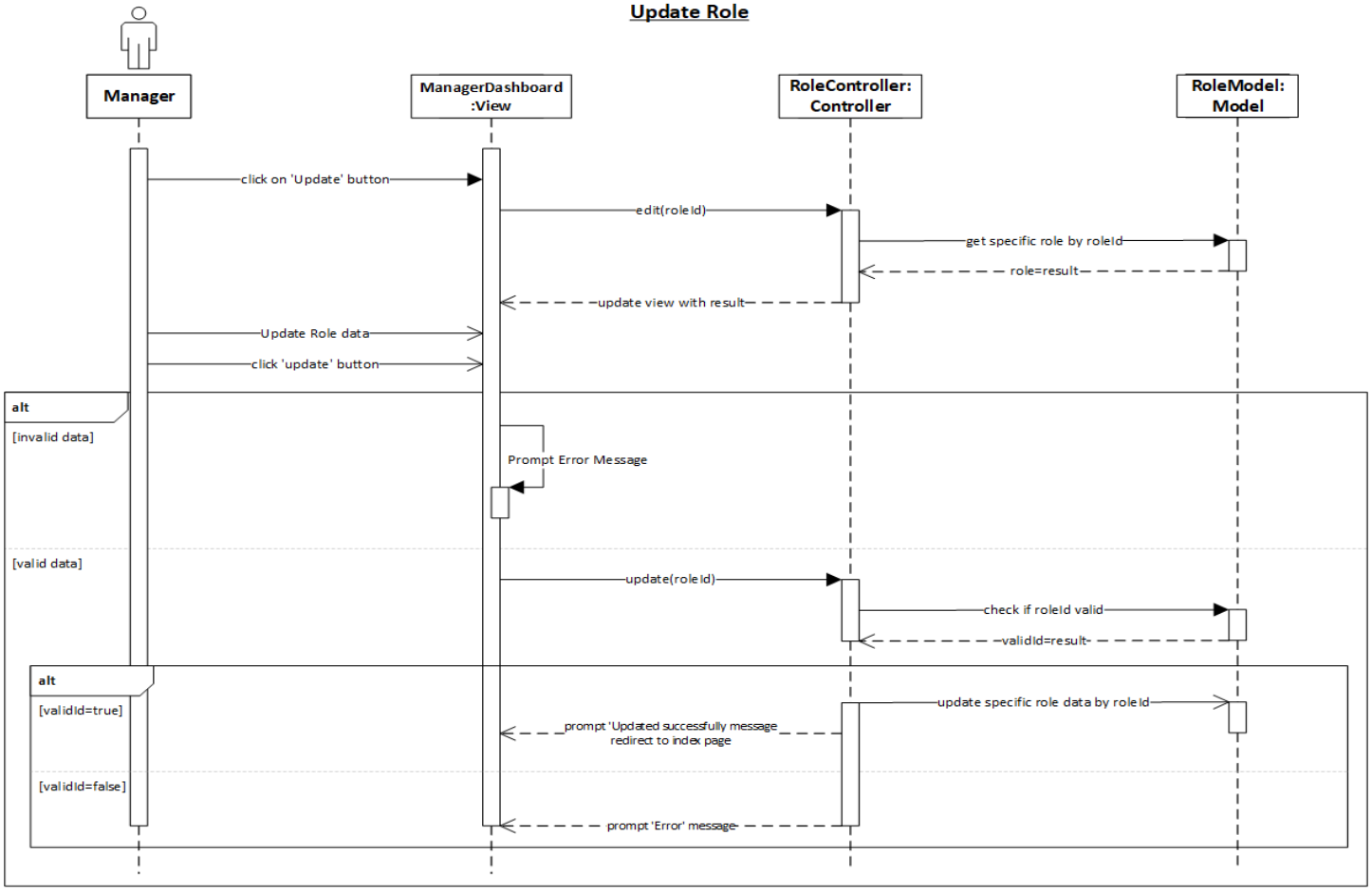


Figure 23: Hospital Manager Update Role Sequence Diagram

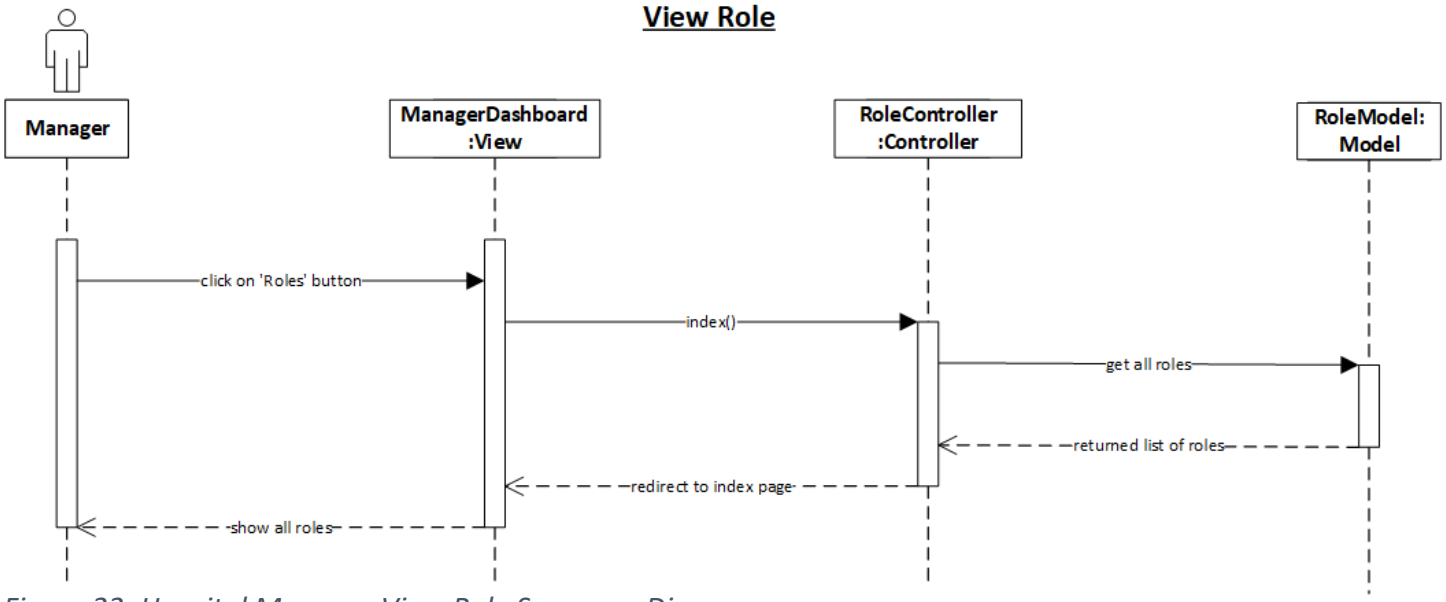


Figure 22: Hospital Manager View Role Sequence Diagram

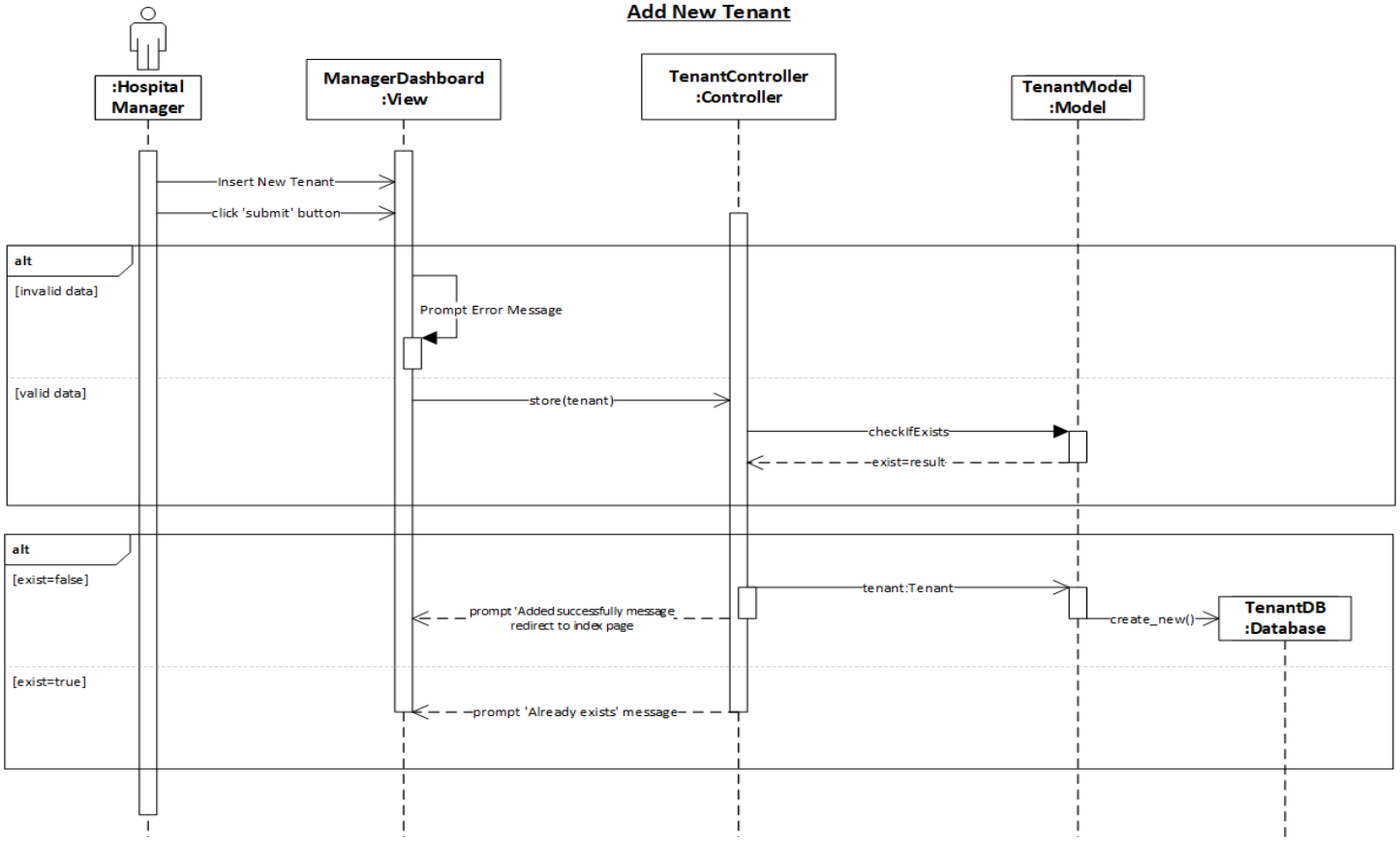


Figure 25: Hospital Manager Add Tenant Sequence Diagram

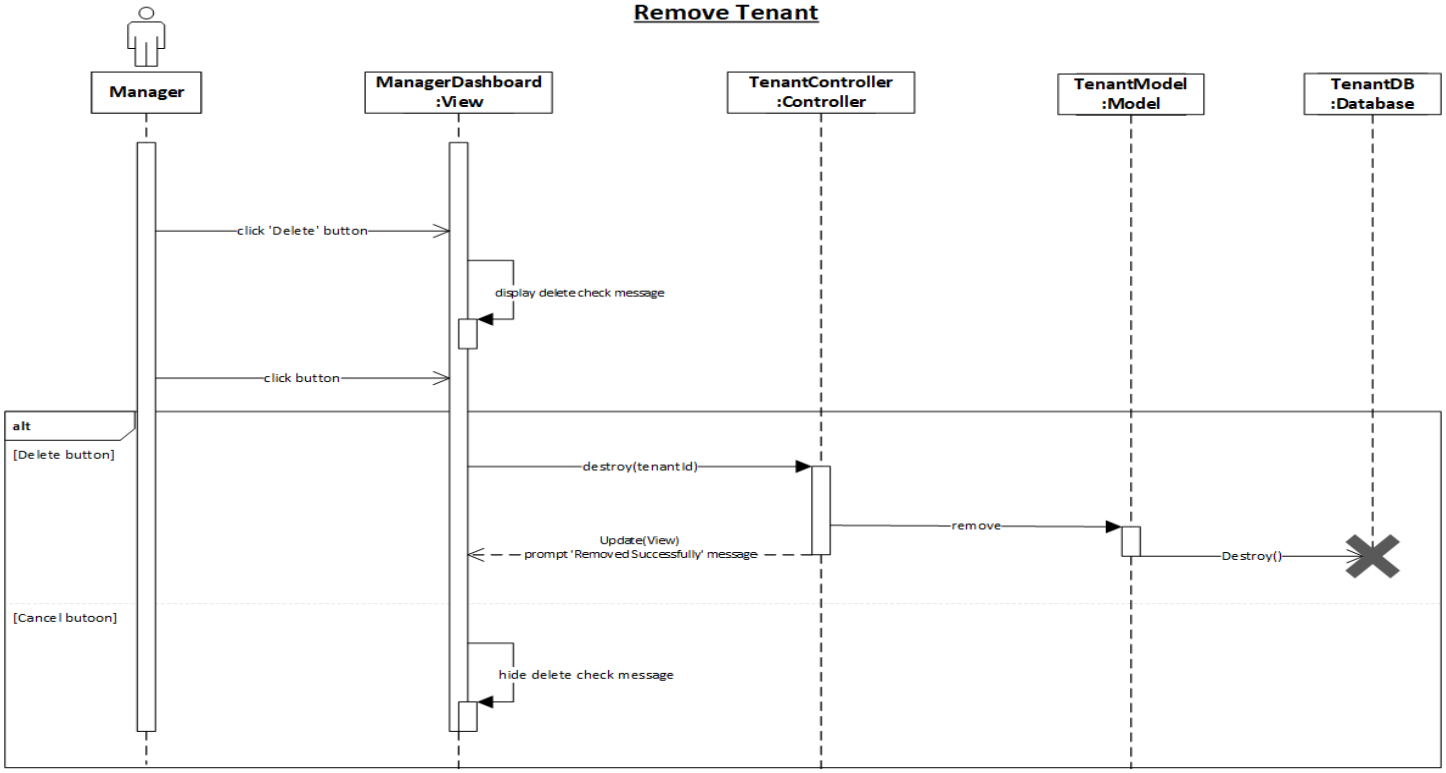


Figure 24: Hospital Manager Remove Tenant Sequence Diagram

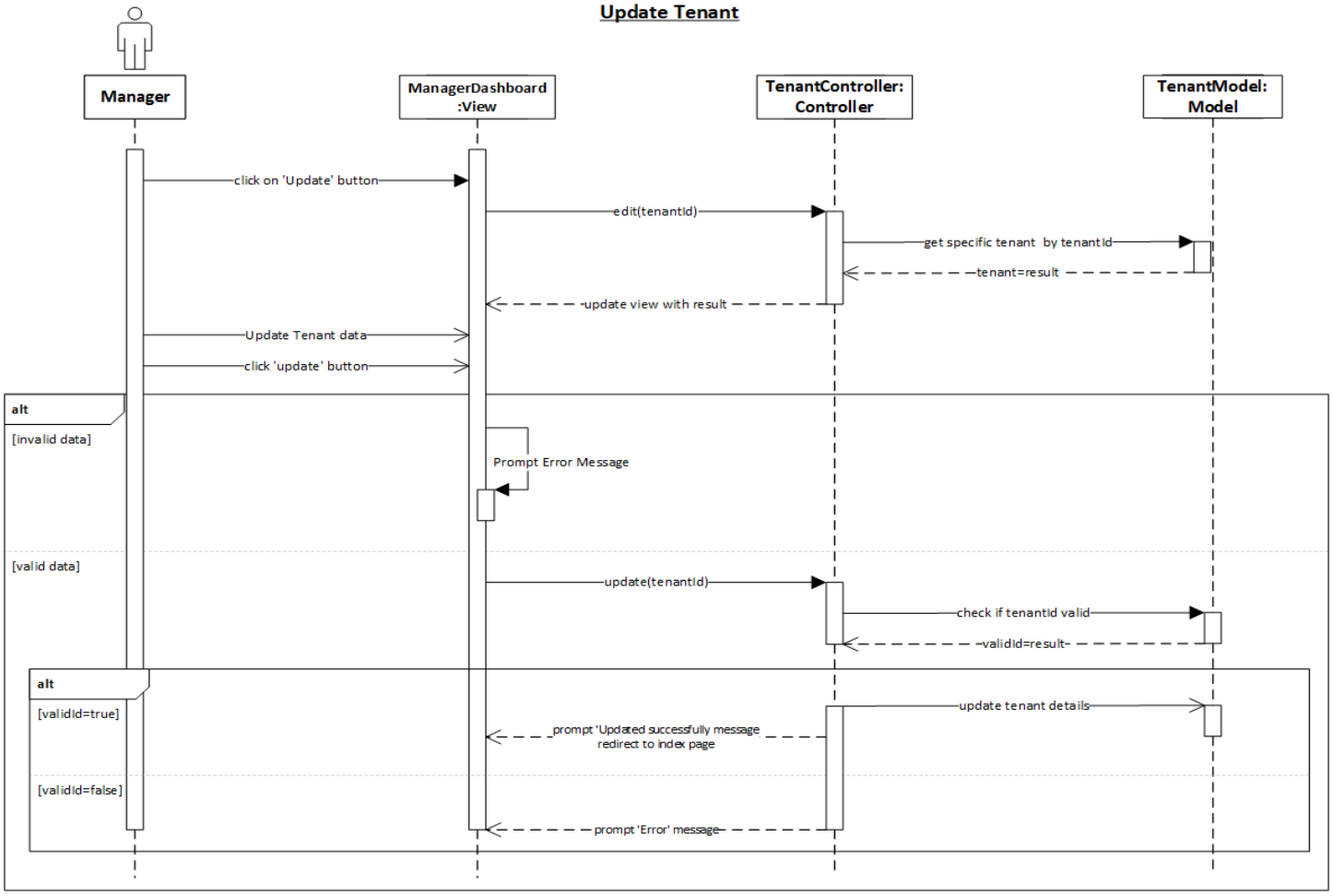


Figure 27: Hospital Manager Update Tenant Sequence Diagram

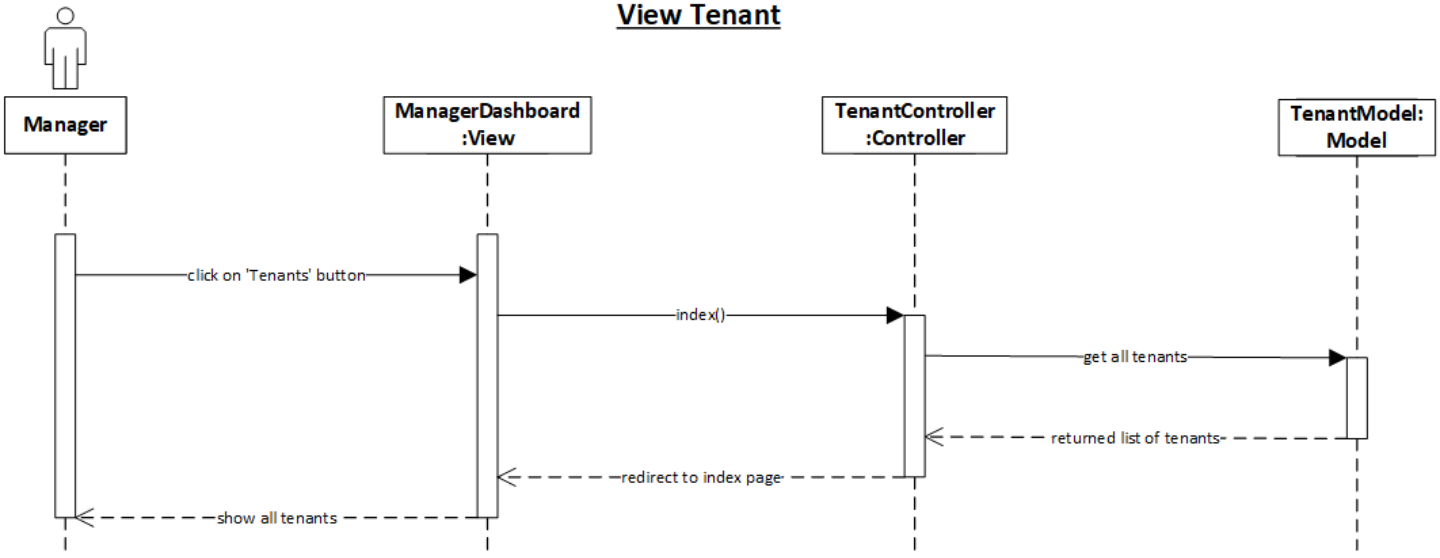


Figure 26: Hospital Manager View Tenant Sequence Diagram

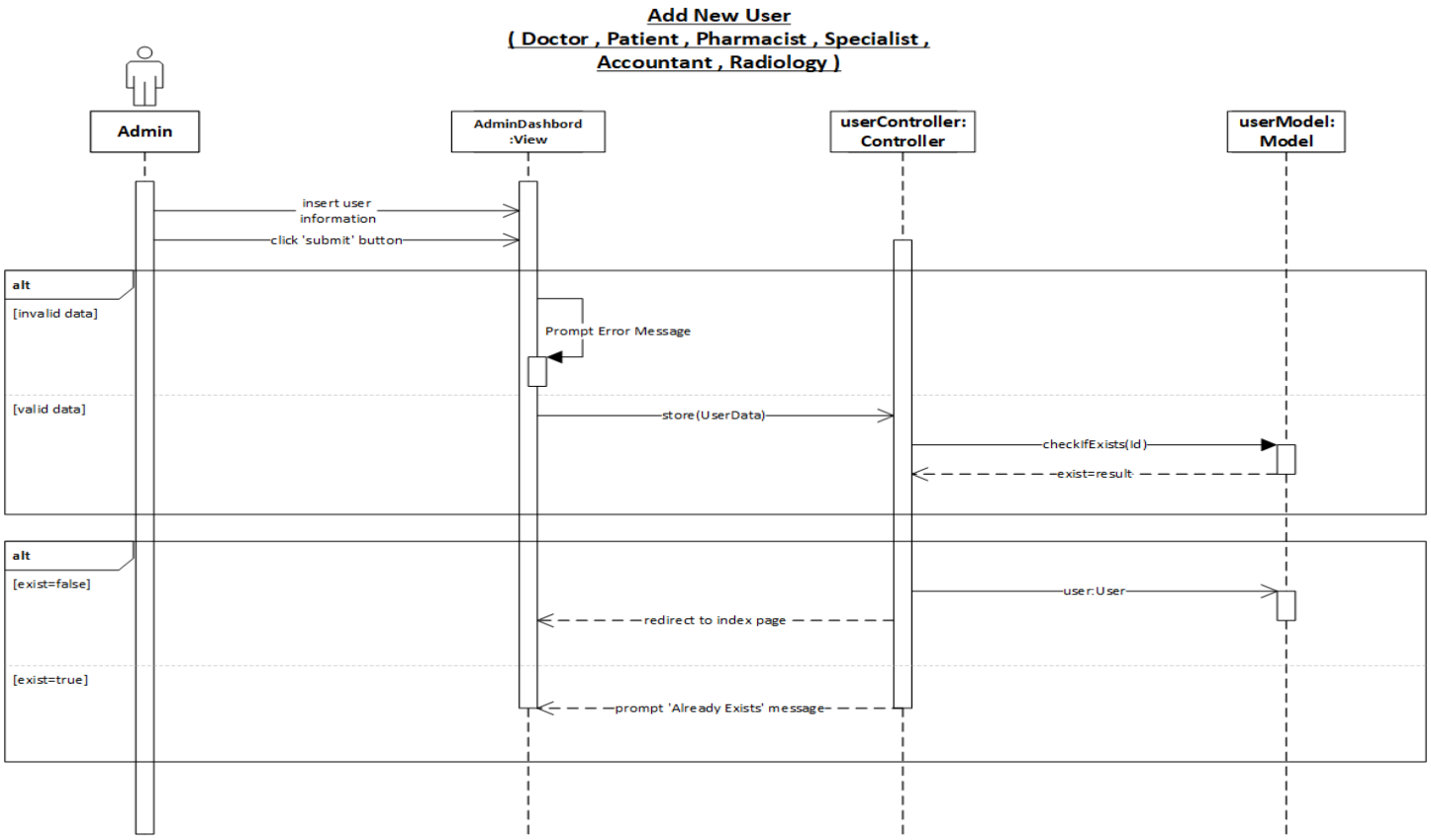


Figure 28: Admin Add Specific User Sequence Diagram

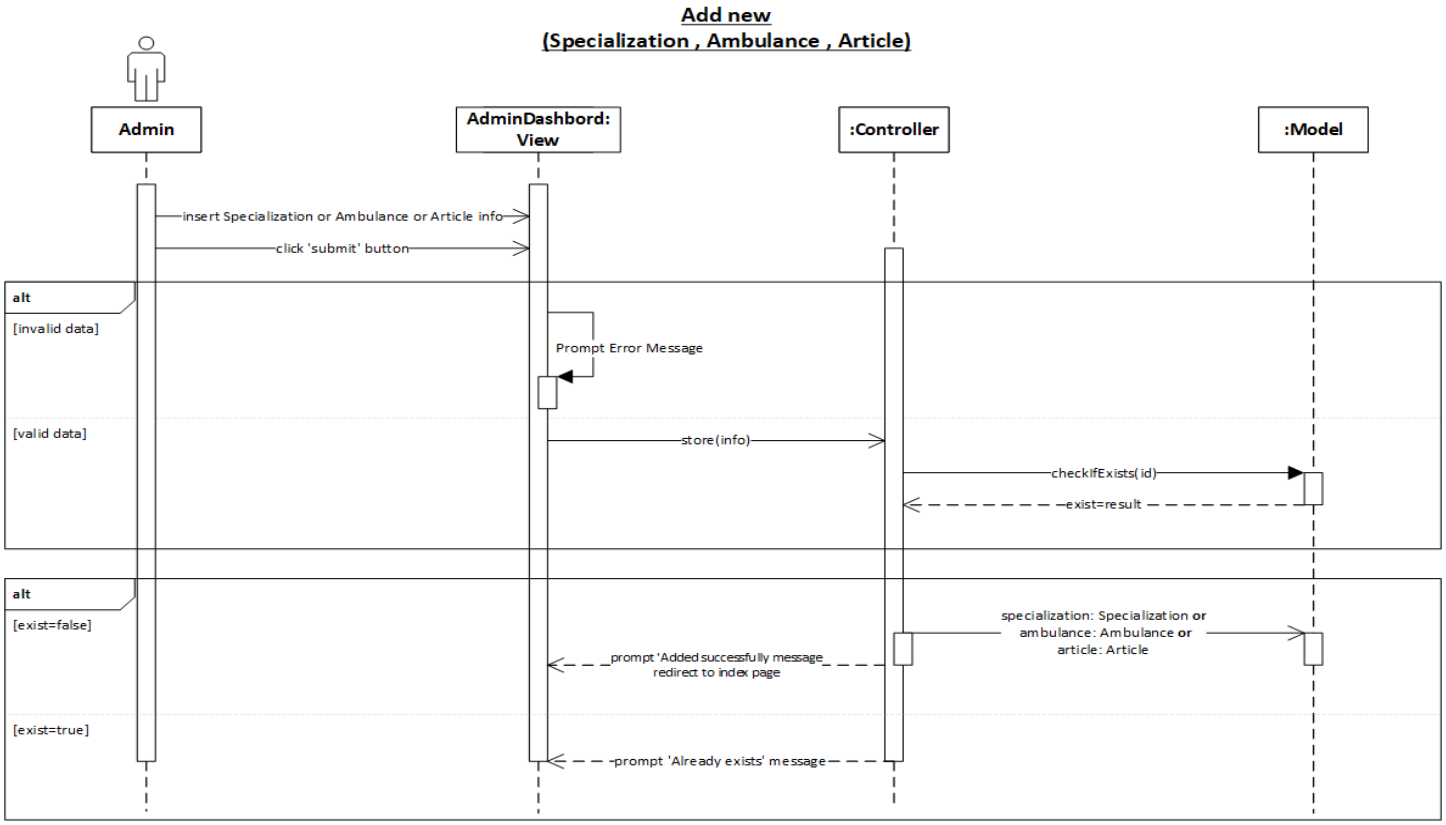


Figure 29: Admin Add Specialization / Ambulance / Article Sequence Diagram

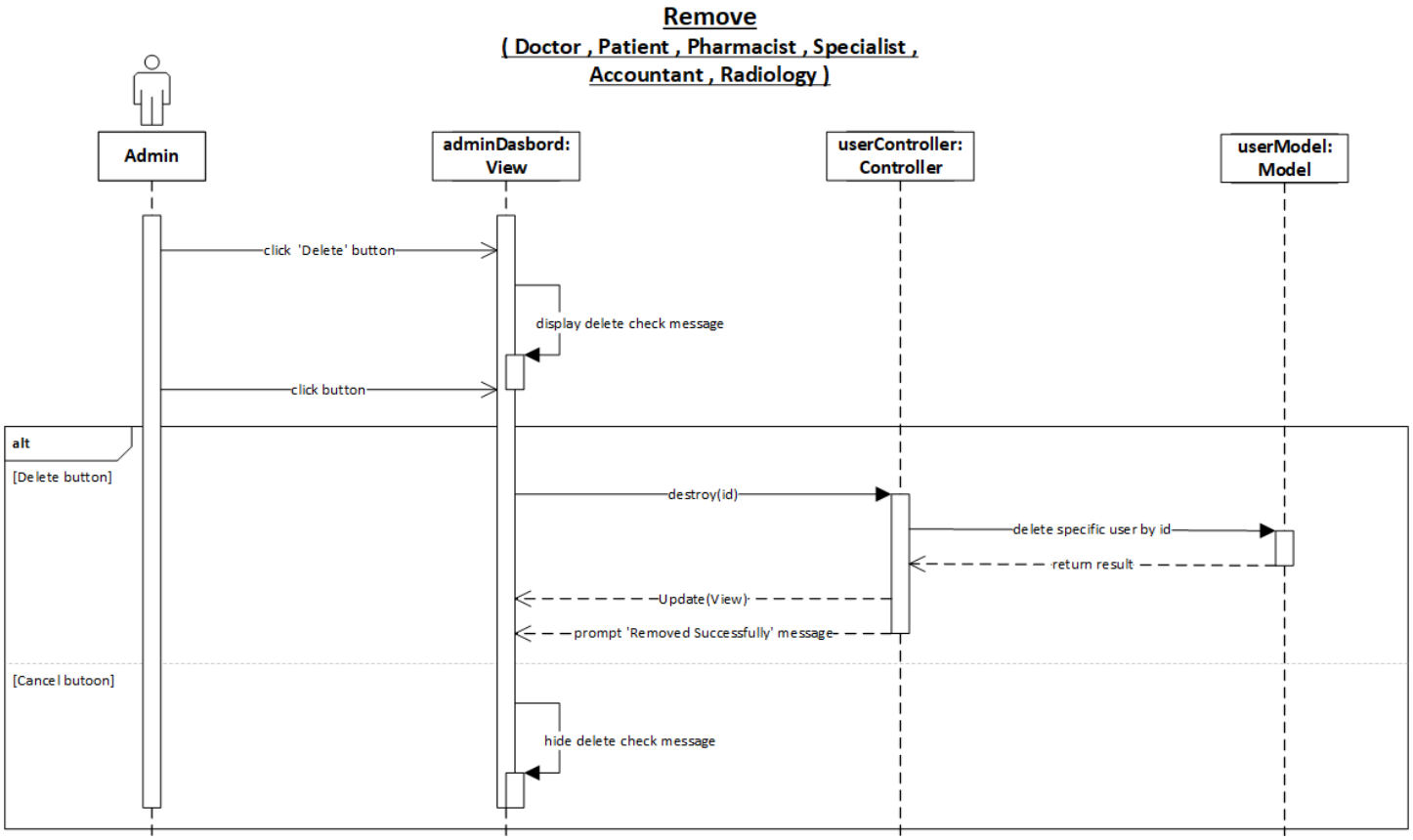


Figure 30: Admin Remove Specific User Sequence Diagram

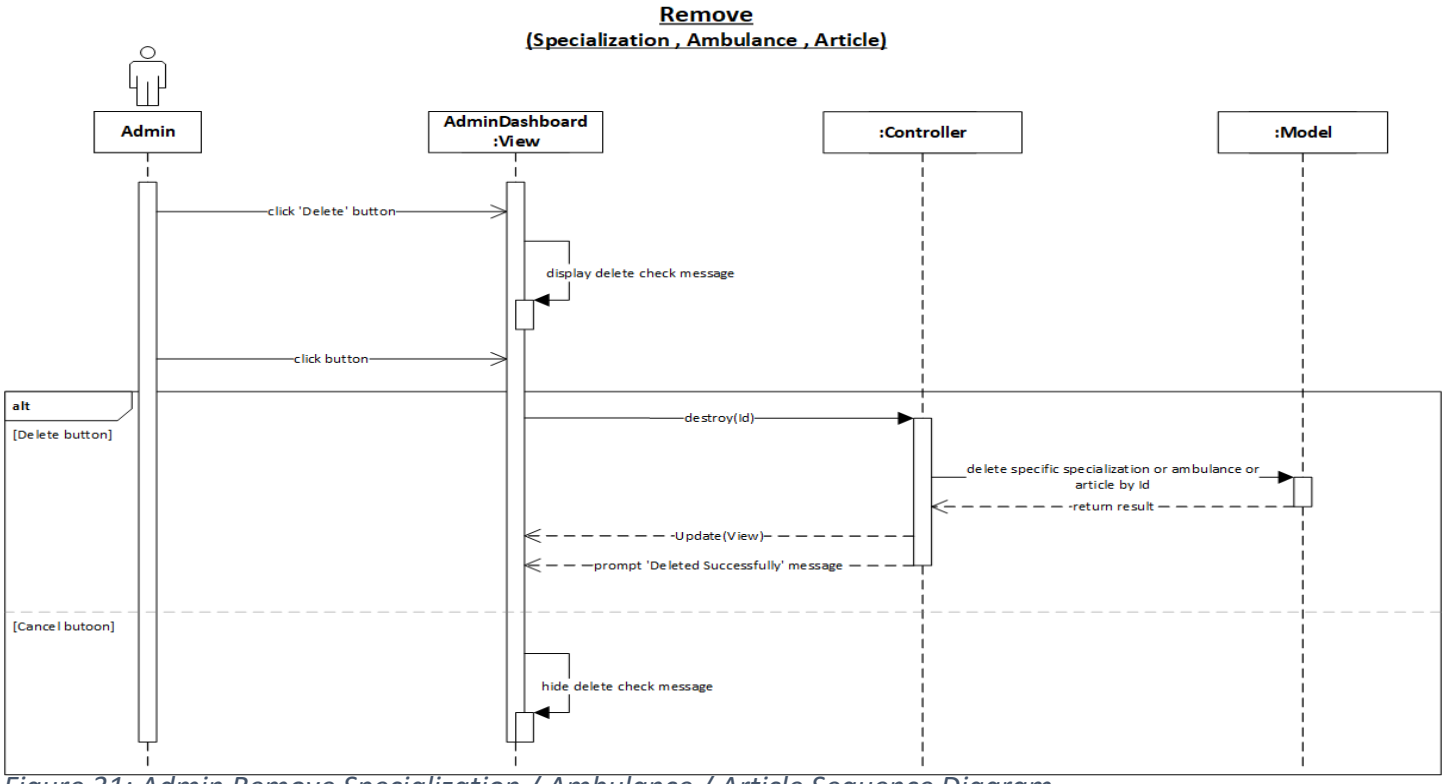


Figure 31: Admin Remove Specialization / Ambulance / Article Sequence Diagram

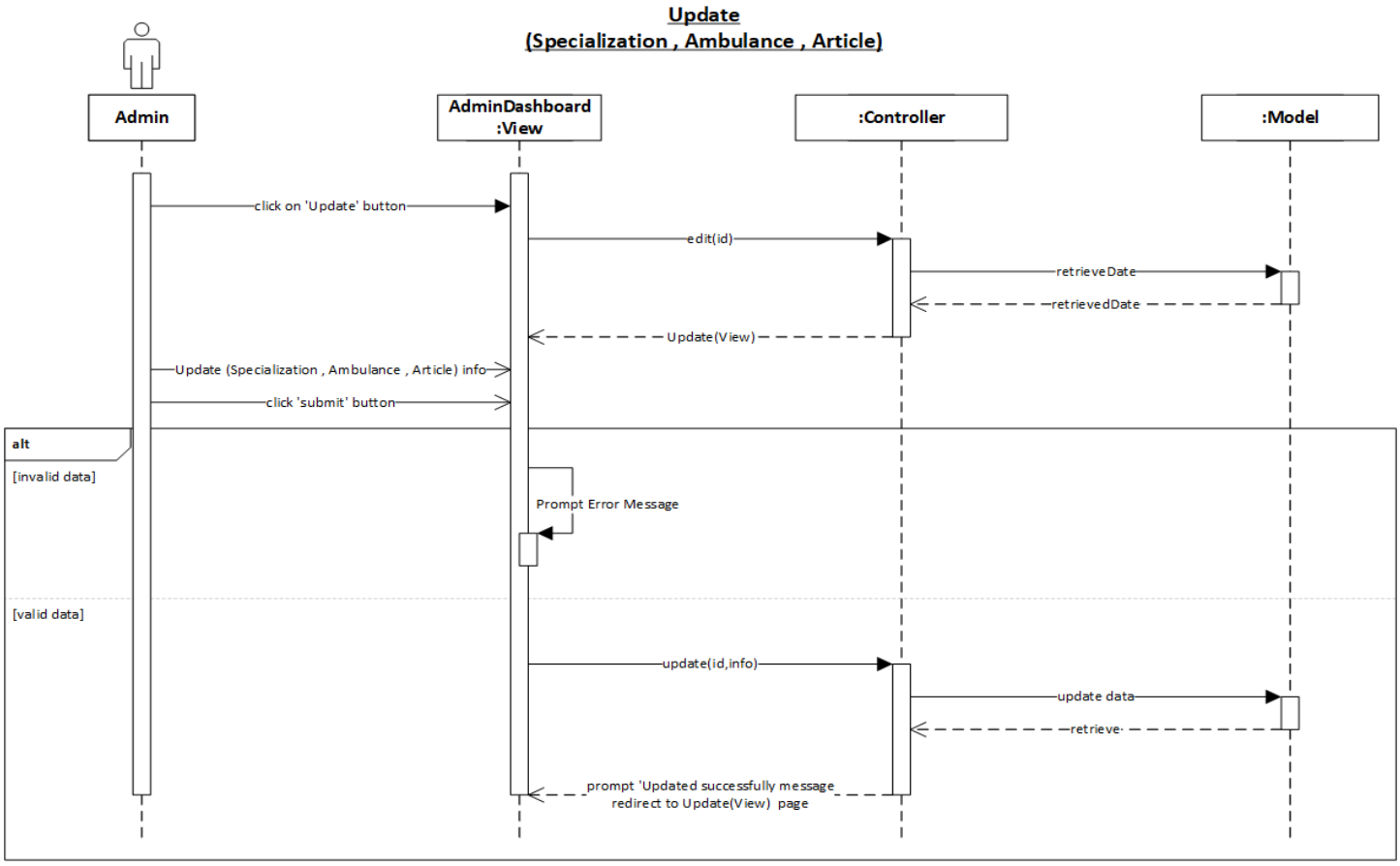


Figure 33: Admin Update Specialization / Ambulance / Article Sequence Diagram

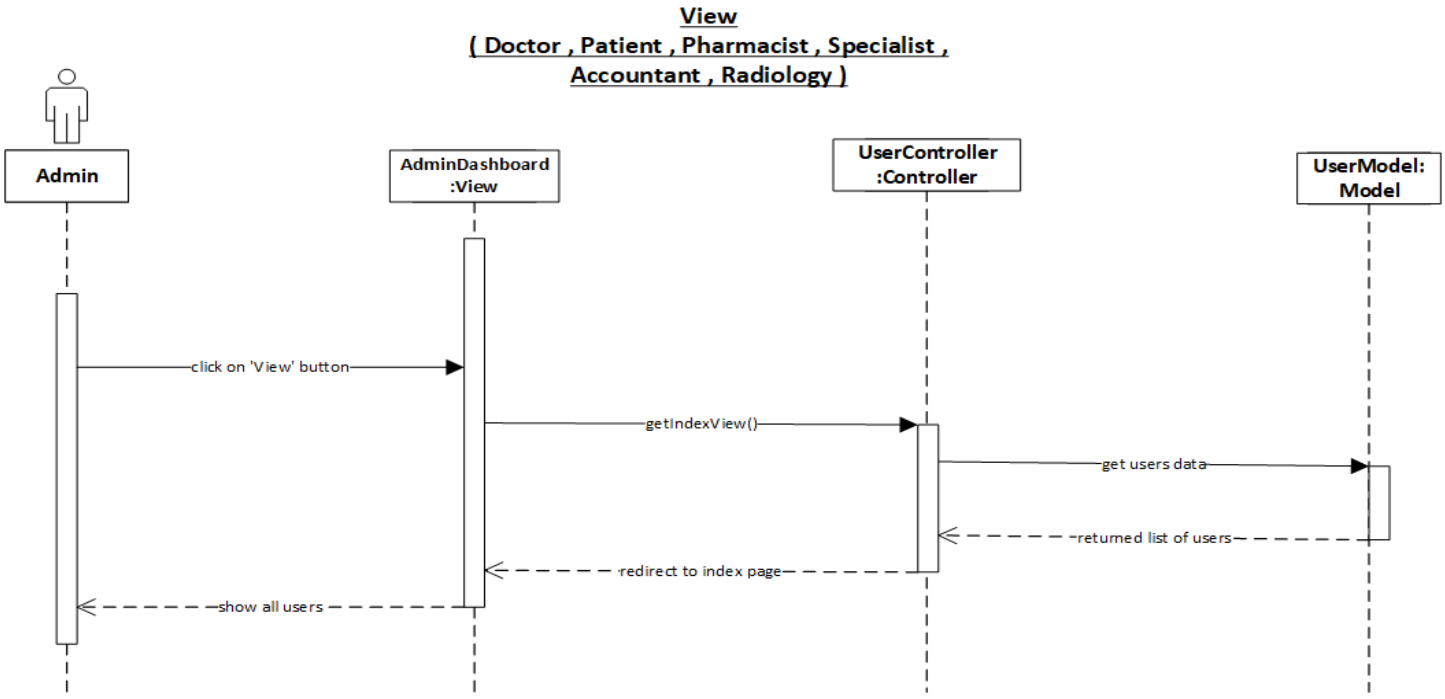


Figure 32: Admin View Specific User Sequence Diagram

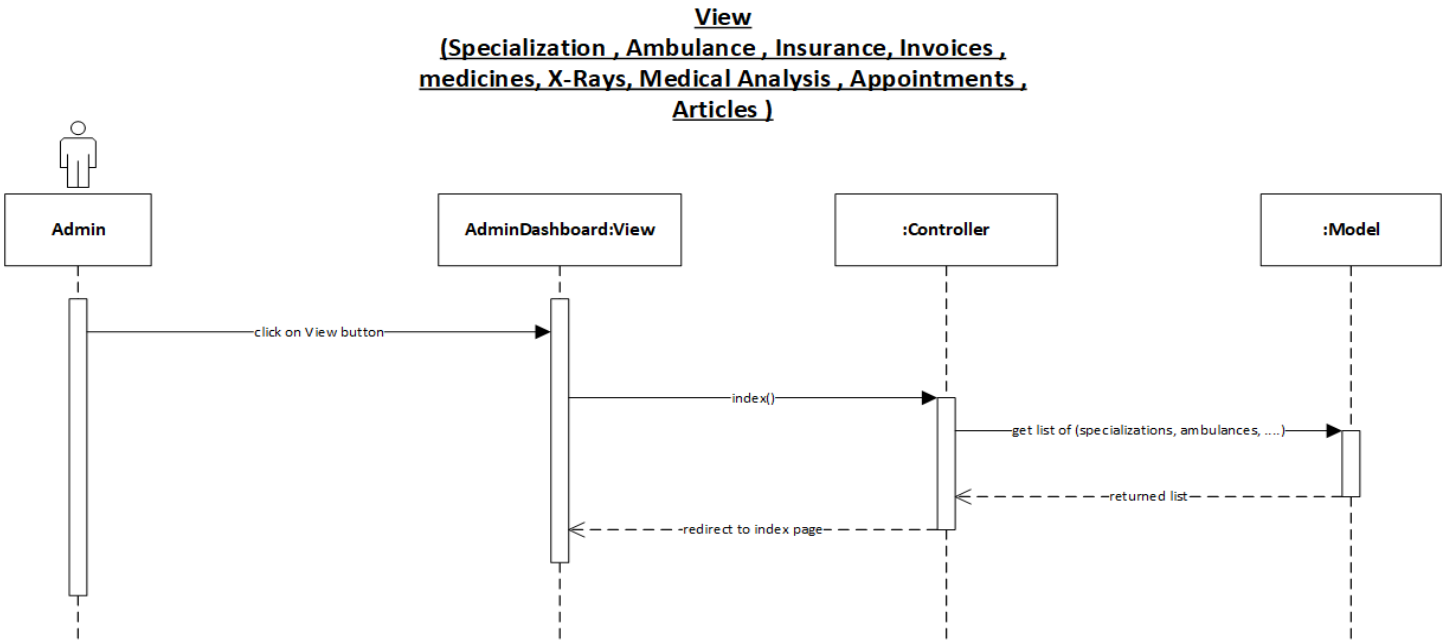


Figure 34: Admin View Something Sequence Diagram

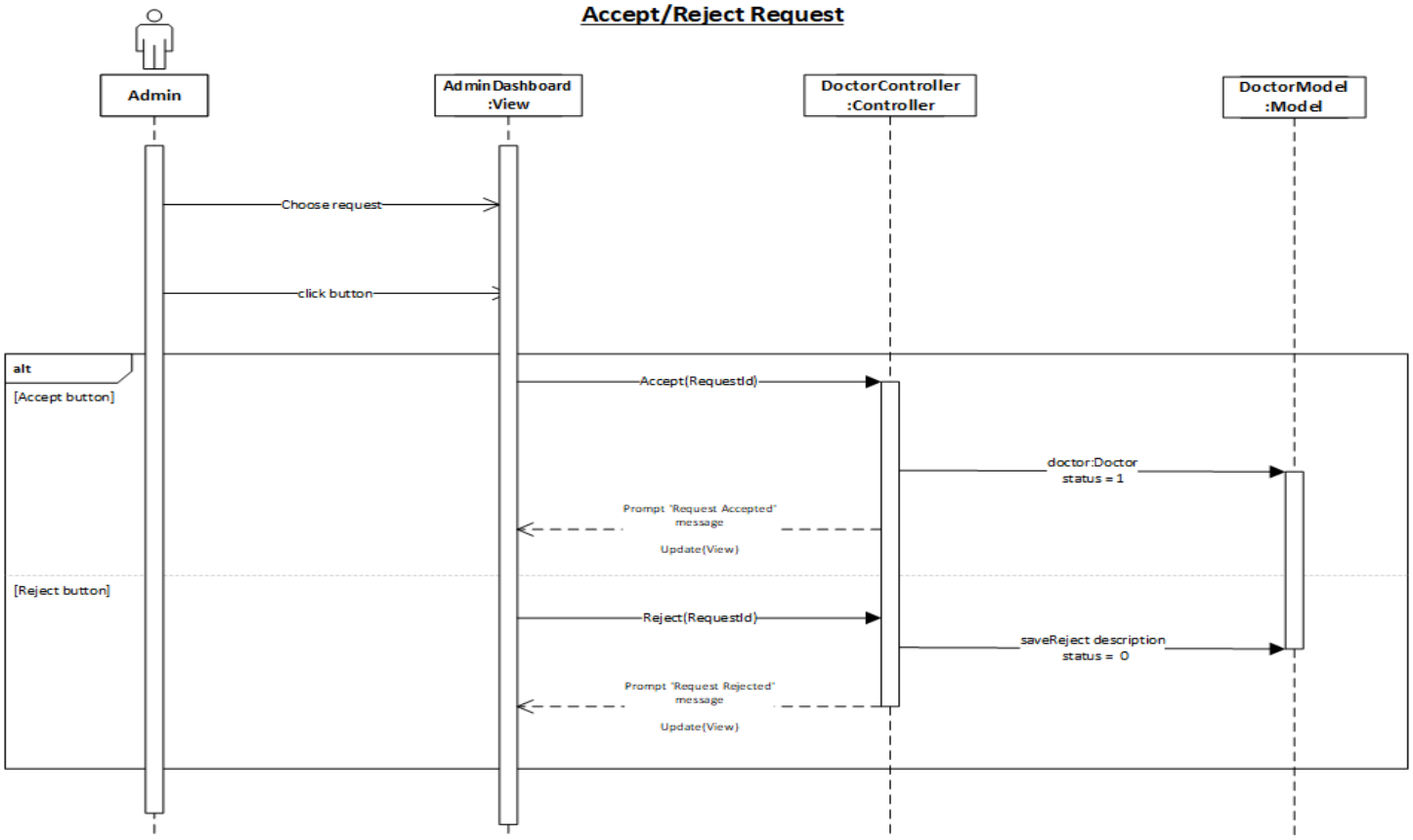


Figure 35: Admin Accept/Reject Request Sequence Diagram

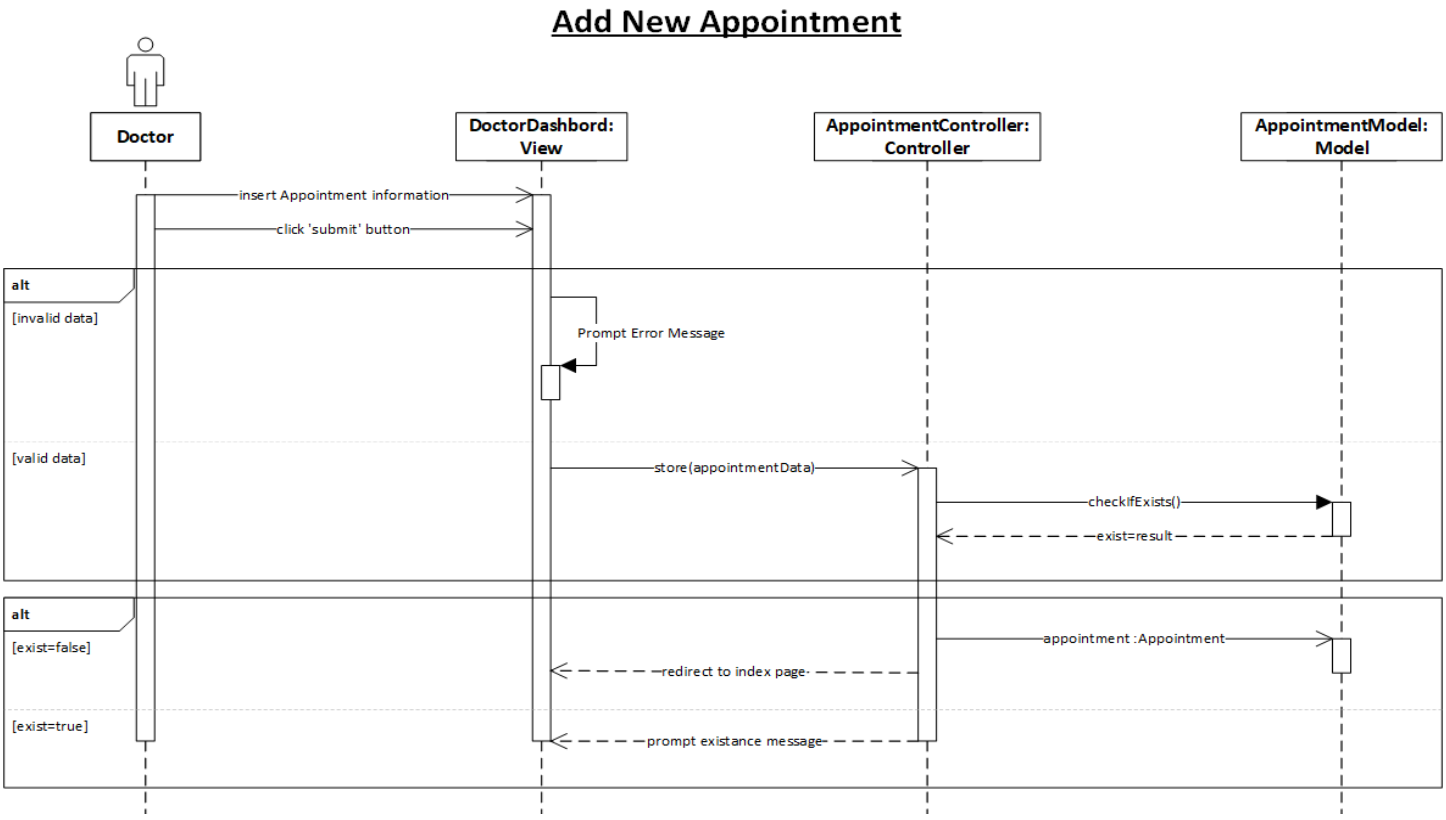


Figure 36: Doctor Add New Appointment Sequence Diagram

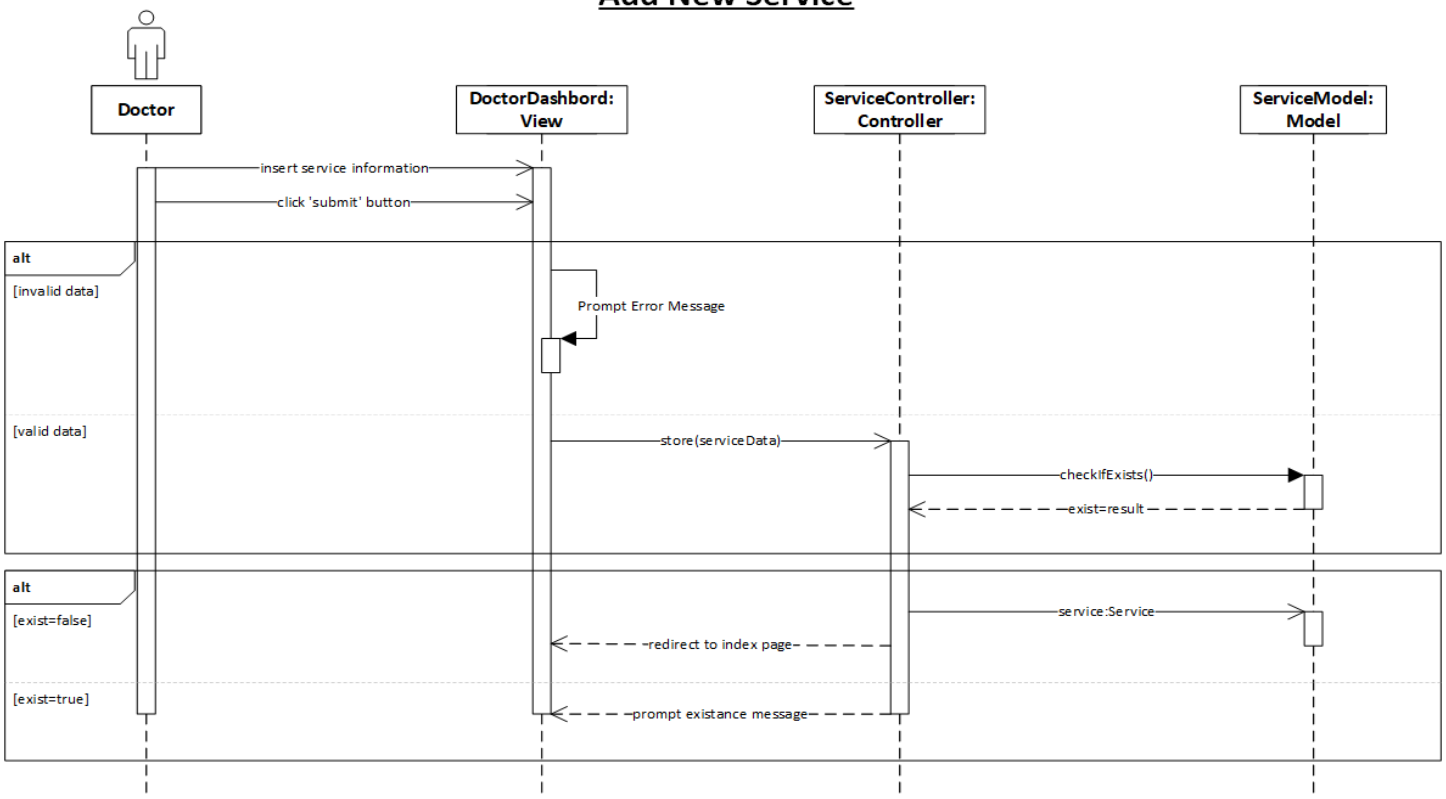


Figure 37: Doctor Add New Service Sequence Diagram

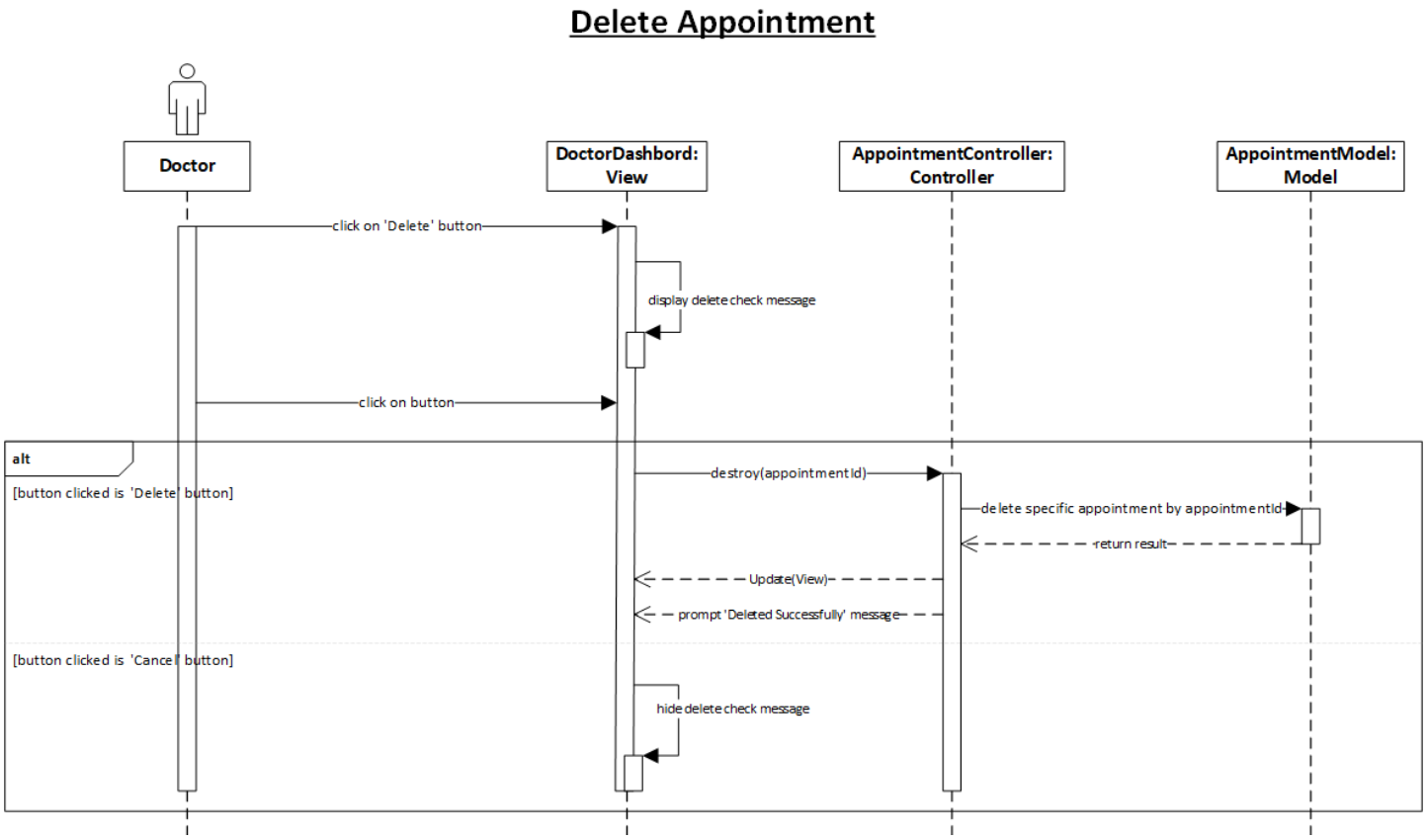


Figure 38: Doctor Delete Appointment Sequence Diagram

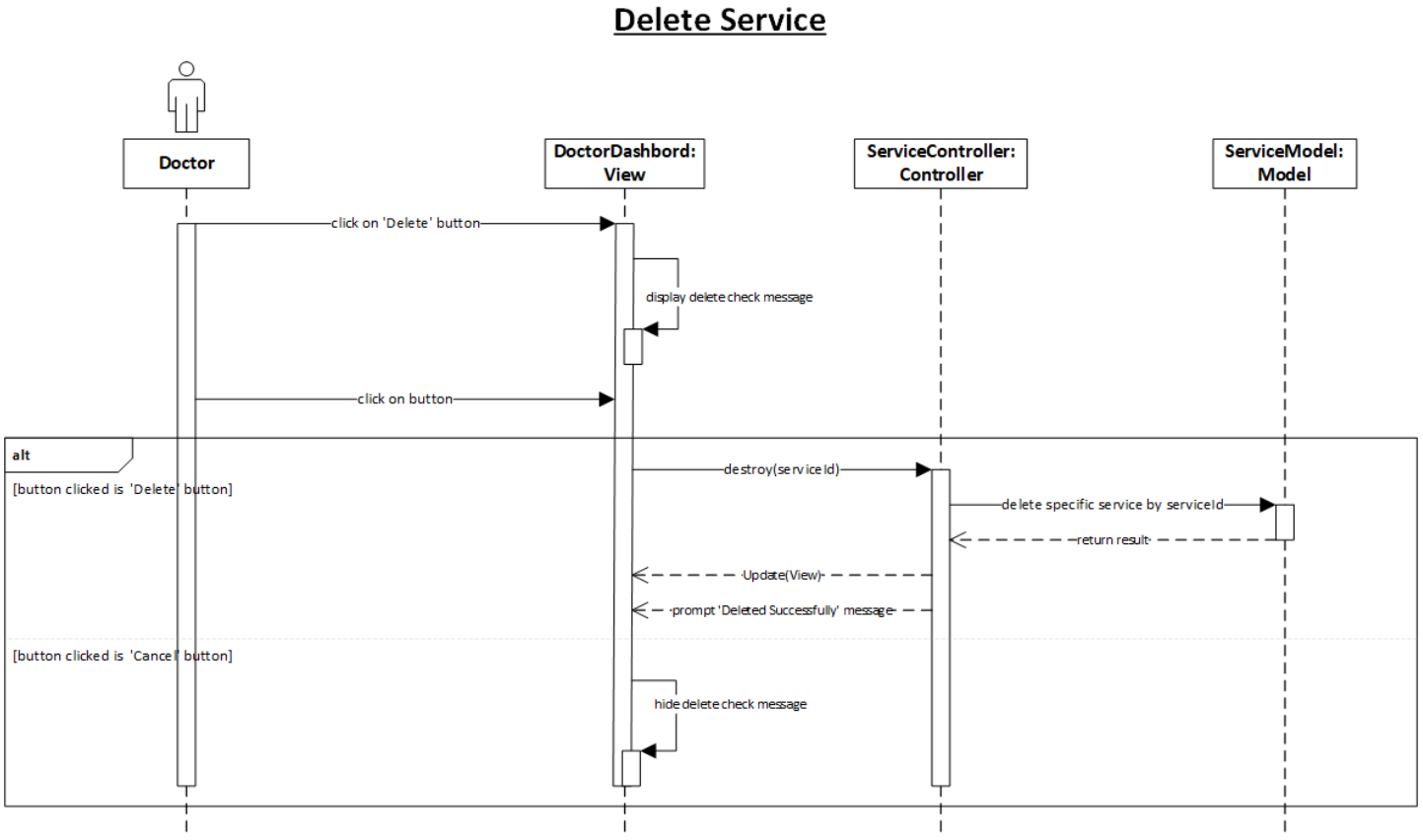


Figure 39: Doctor Delete Service Sequence Diagram

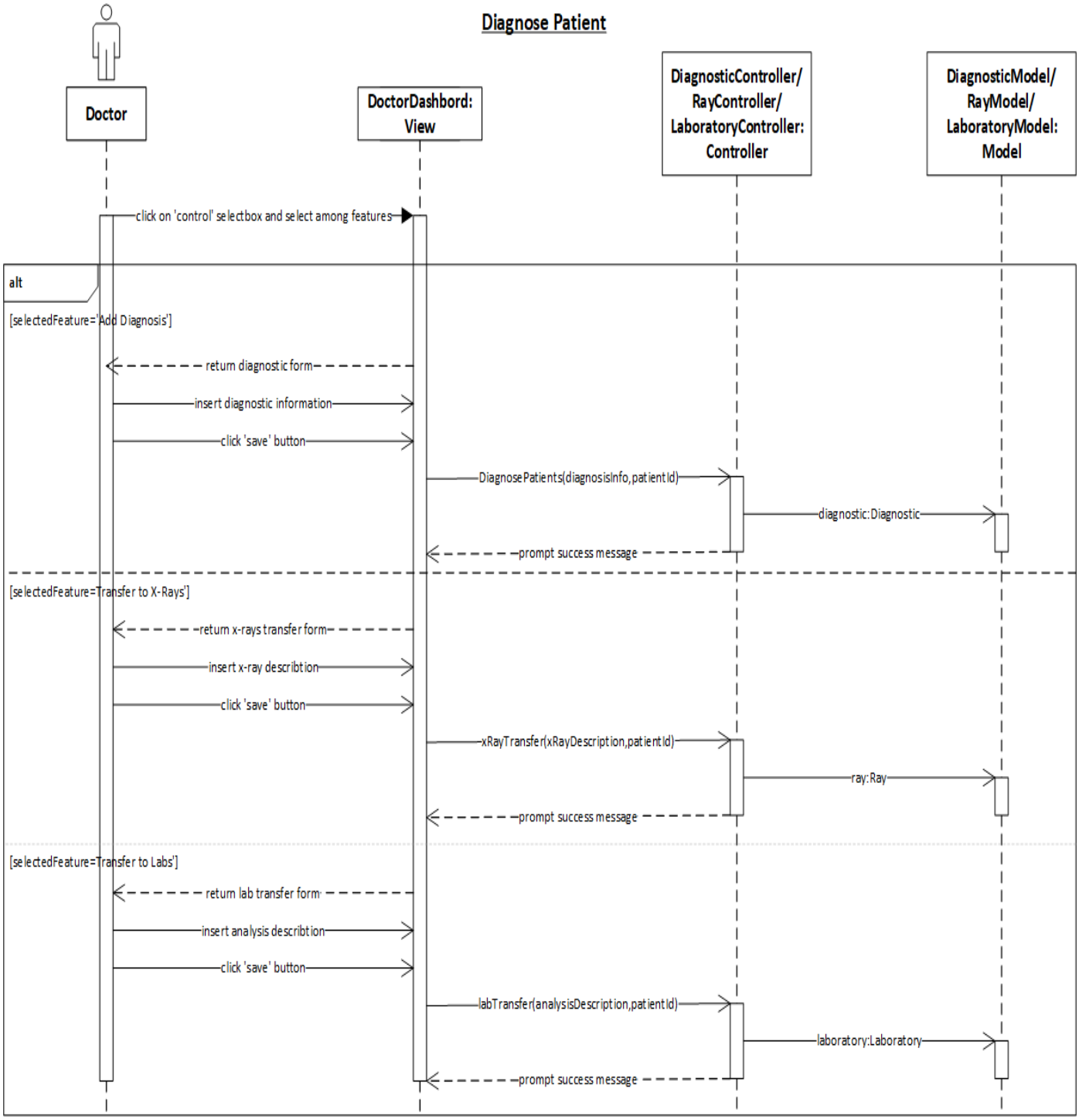


Figure 40: Doctor Diagnose Patient Sequence Diagram

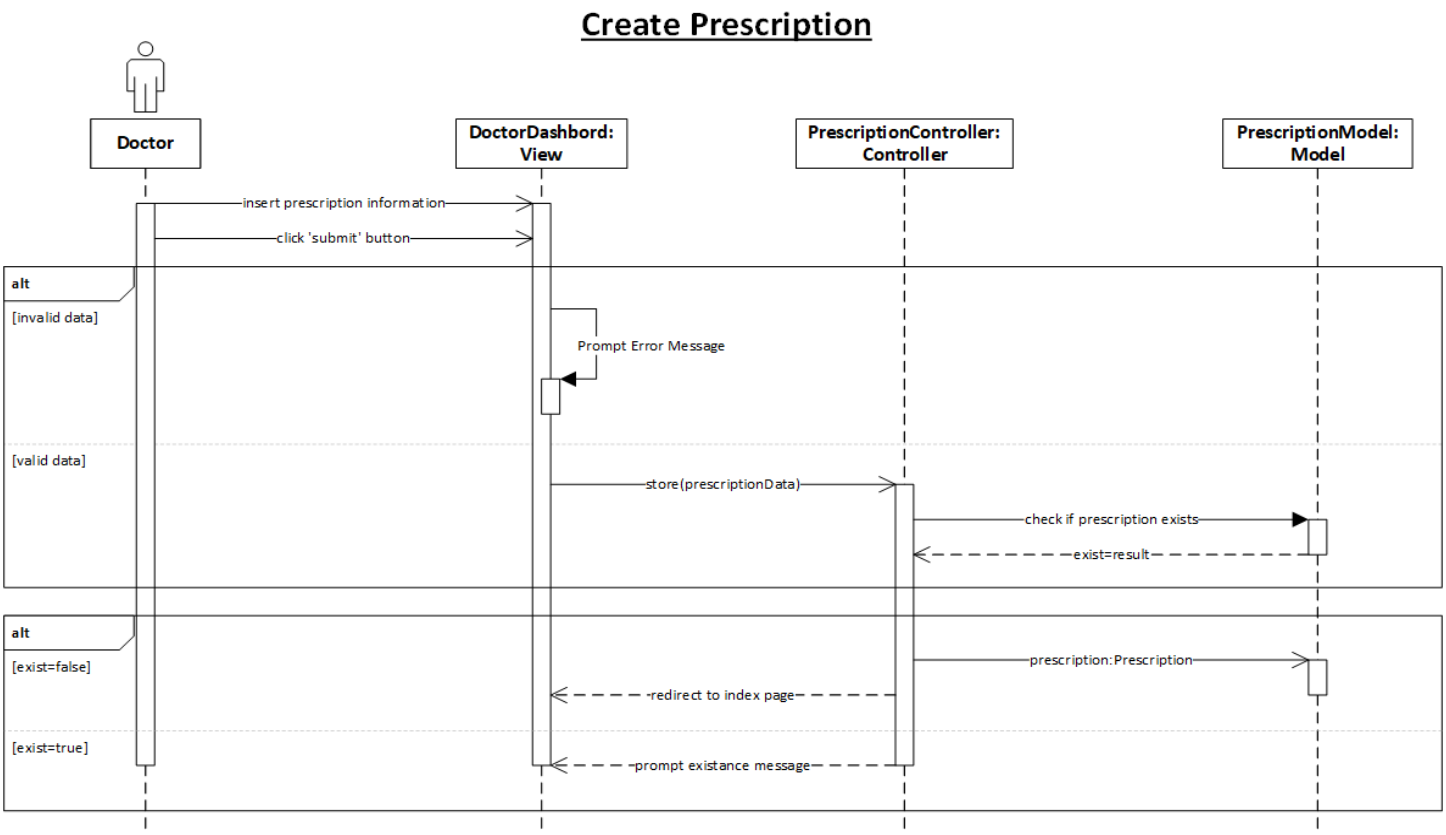


Figure 41: Doctor Create Prescription Sequence Diagram

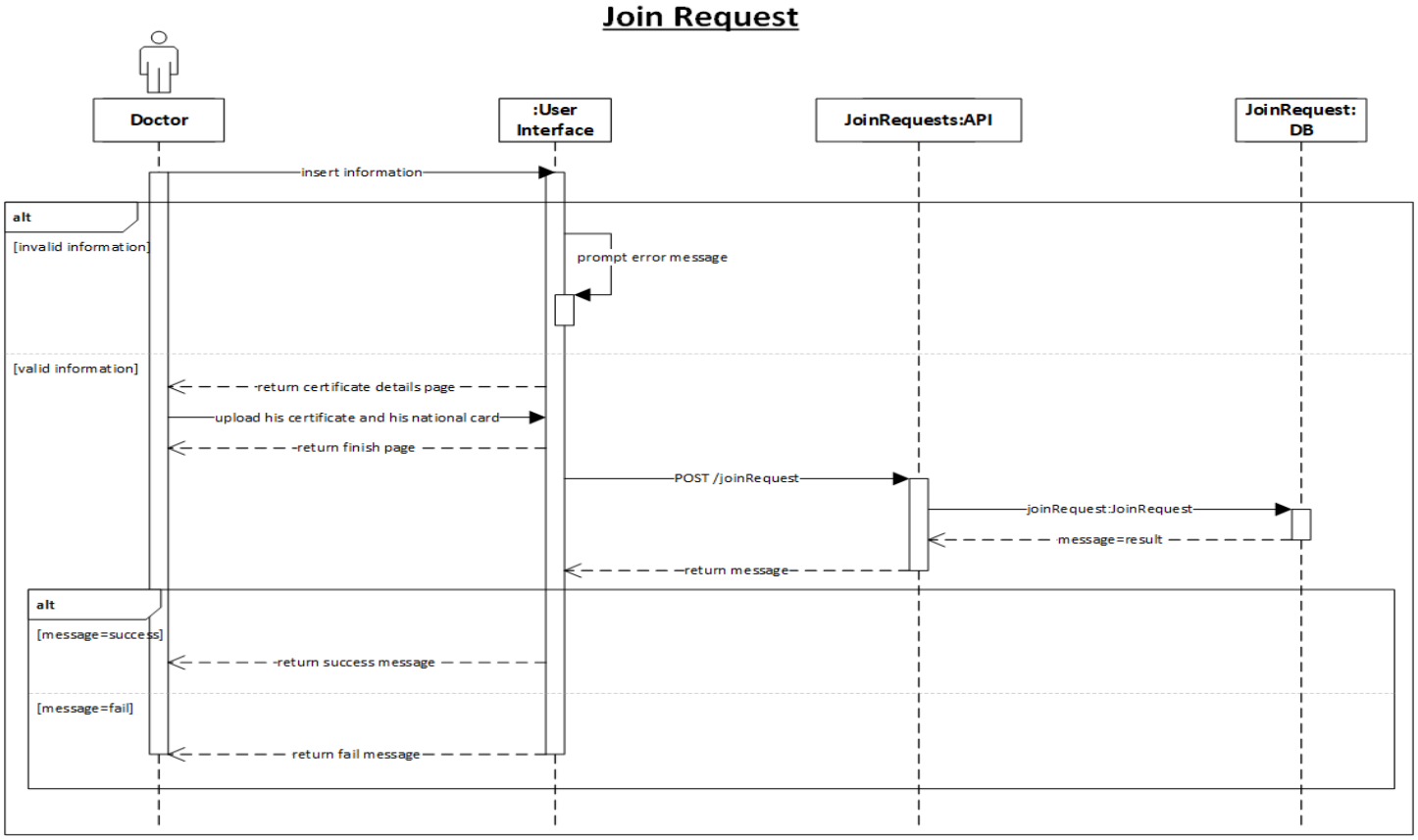


Figure 42: Doctor Join Request Sequence Diagram

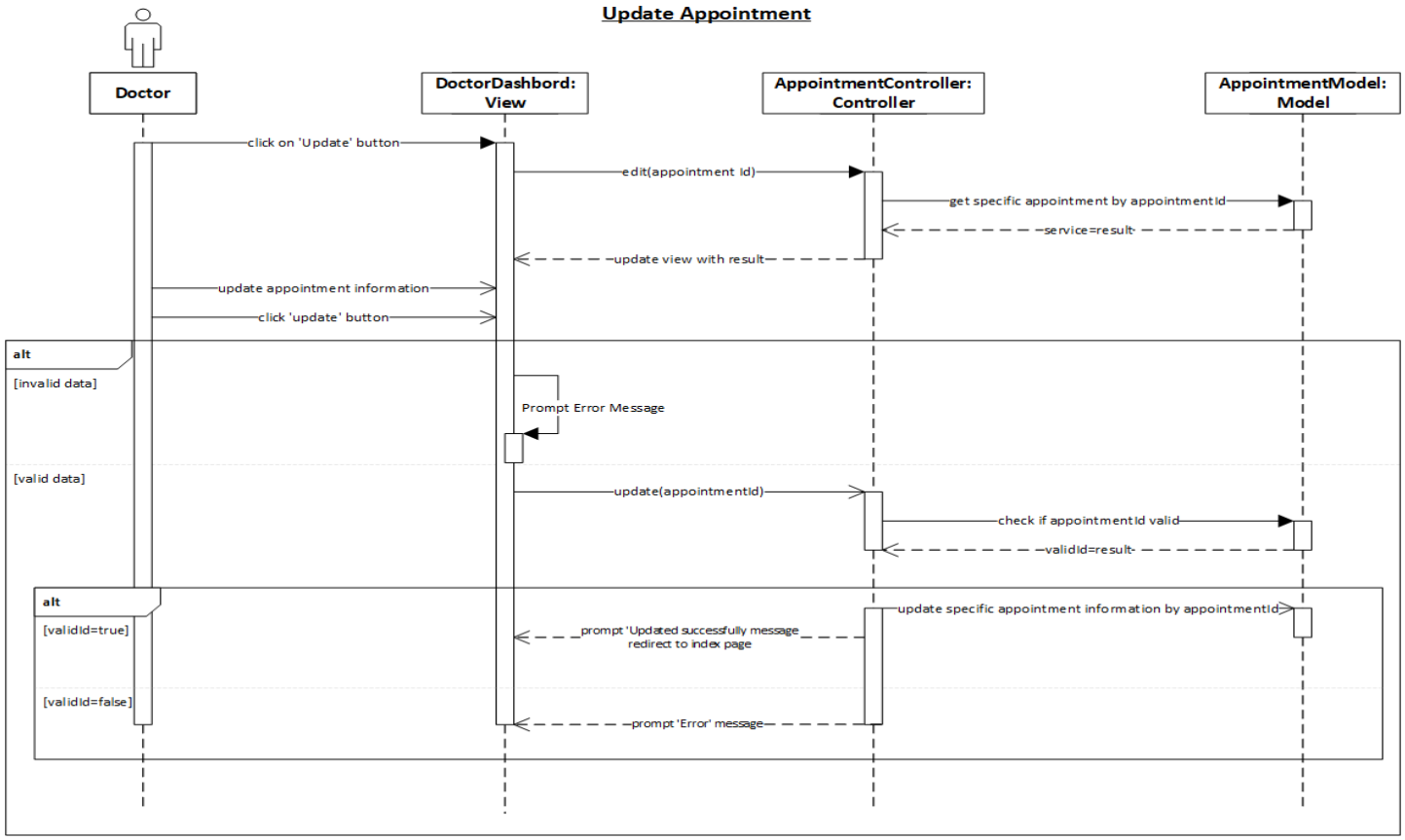


Figure 43: Doctor Update Appointment Sequence Diagram

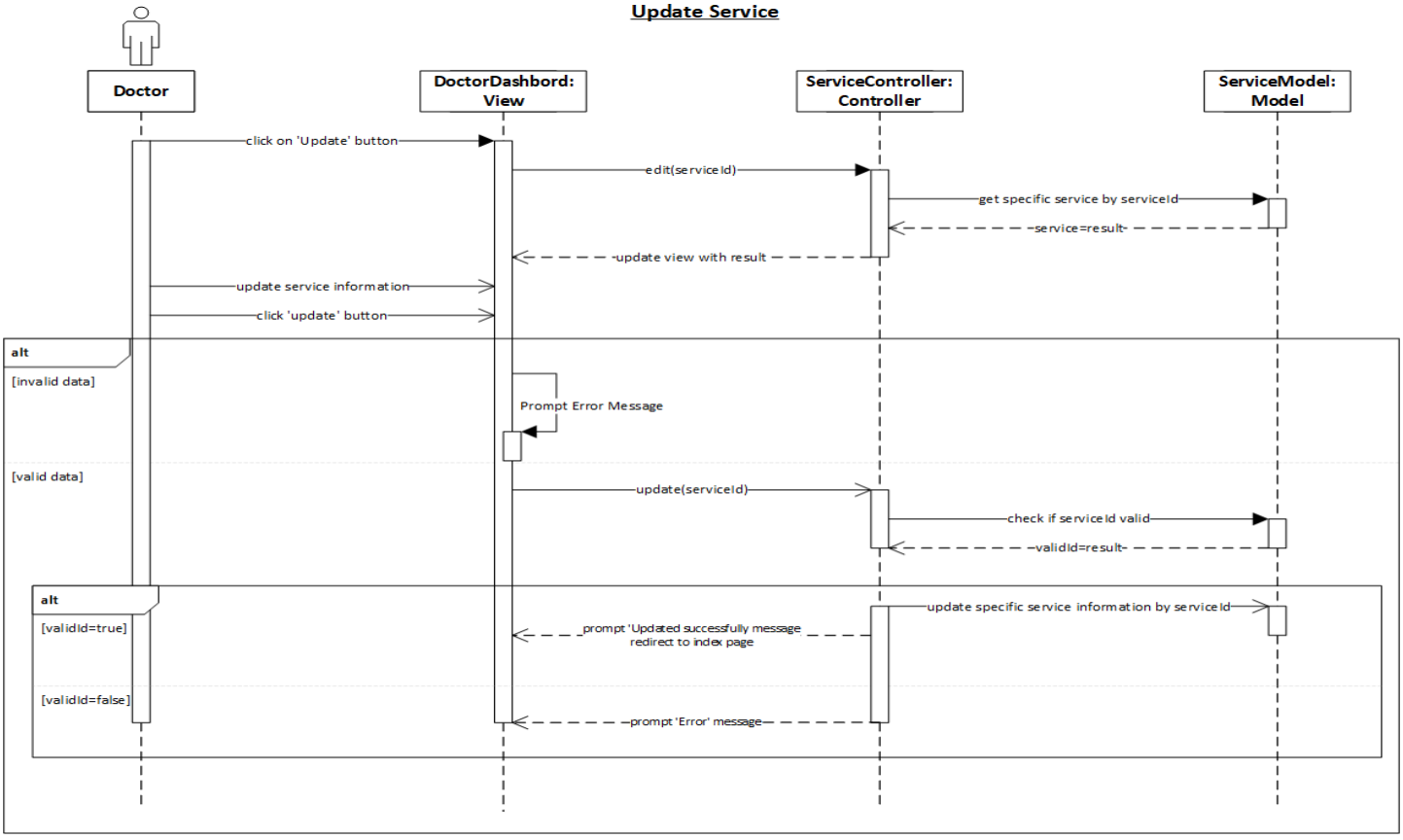


Figure 44: Doctor Update Service Sequence Diagram

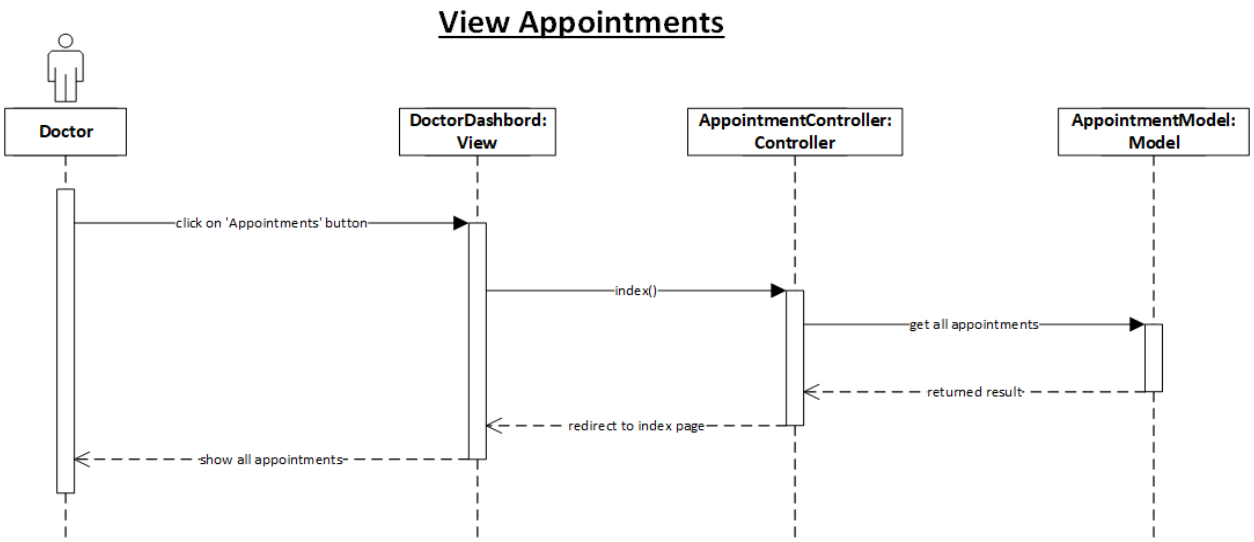


Figure 46: Doctor View Appointments Sequence Diagram

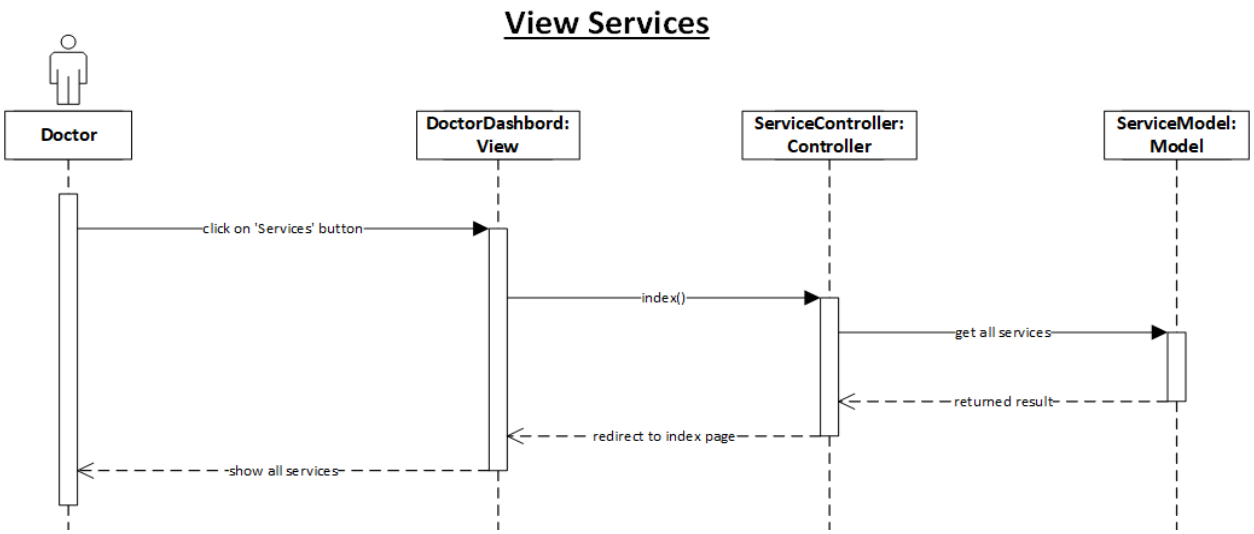


Figure 47: Doctor View Services Sequence Diagram

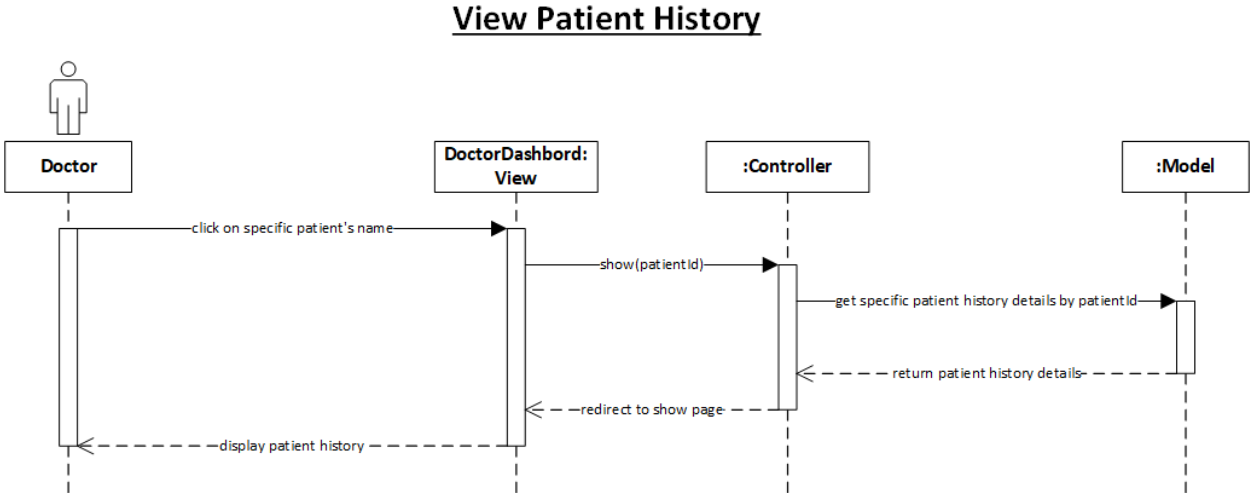


Figure 45: Doctor View Patient History Sequence Diagram

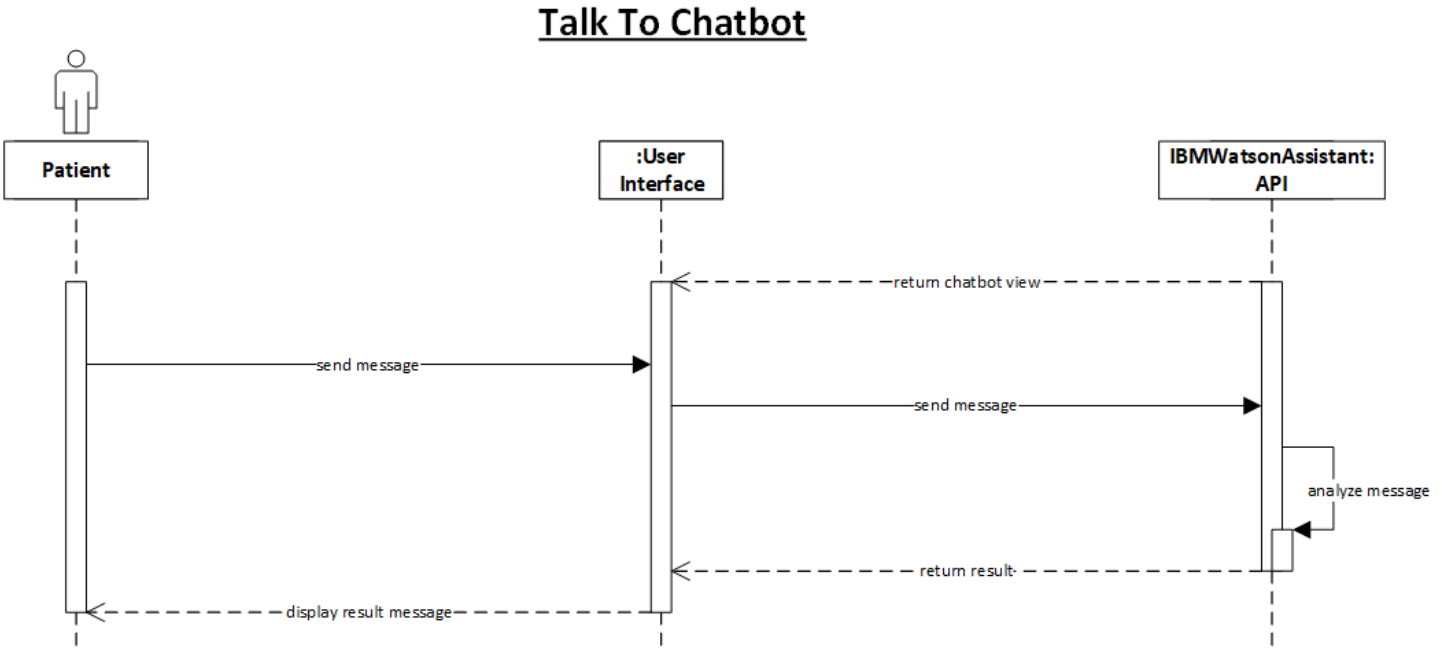


Figure 49: Patient talk to chatbot Sequence Diagram

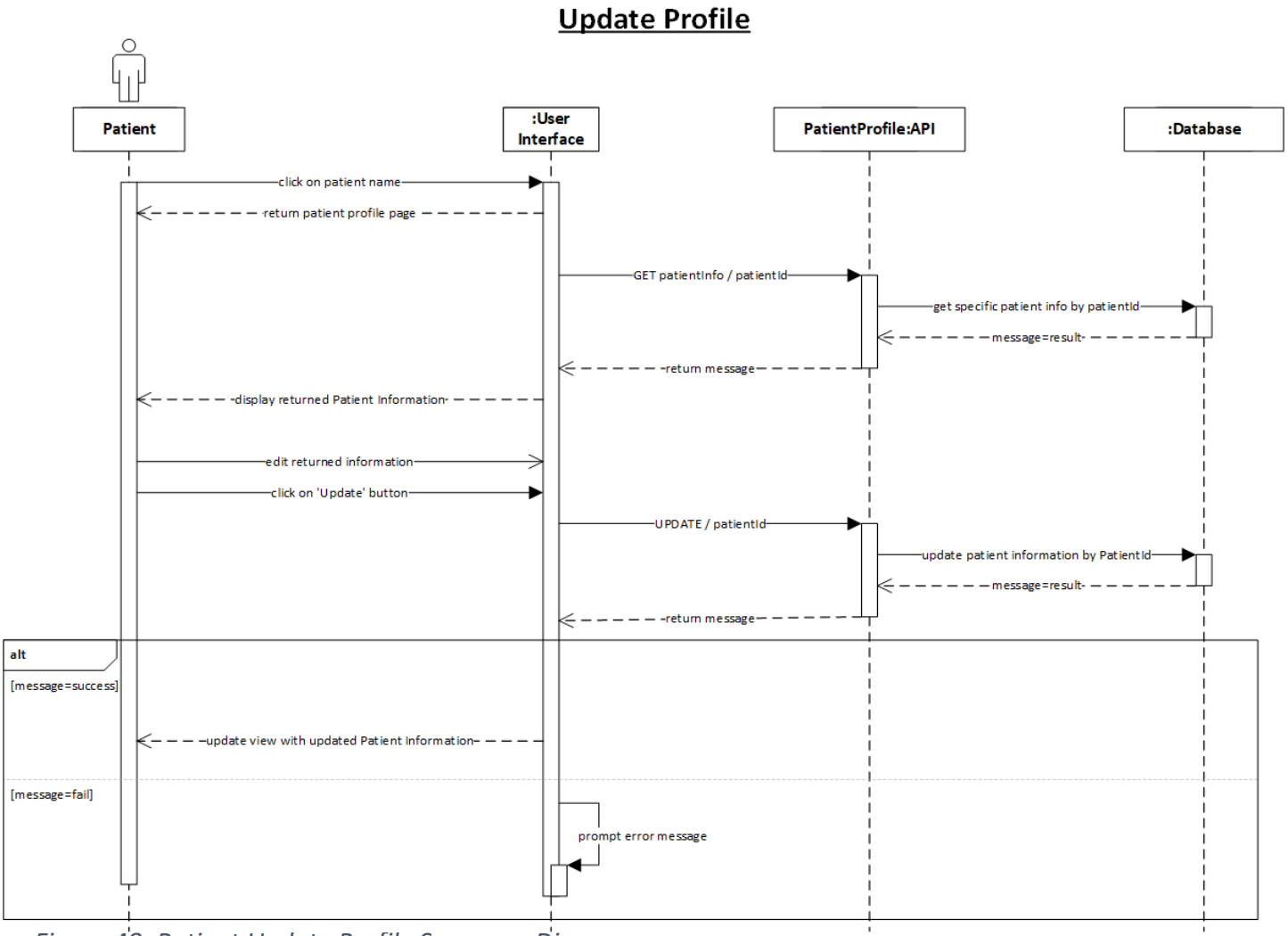


Figure 48: Patient Update Profile Sequence Diagram

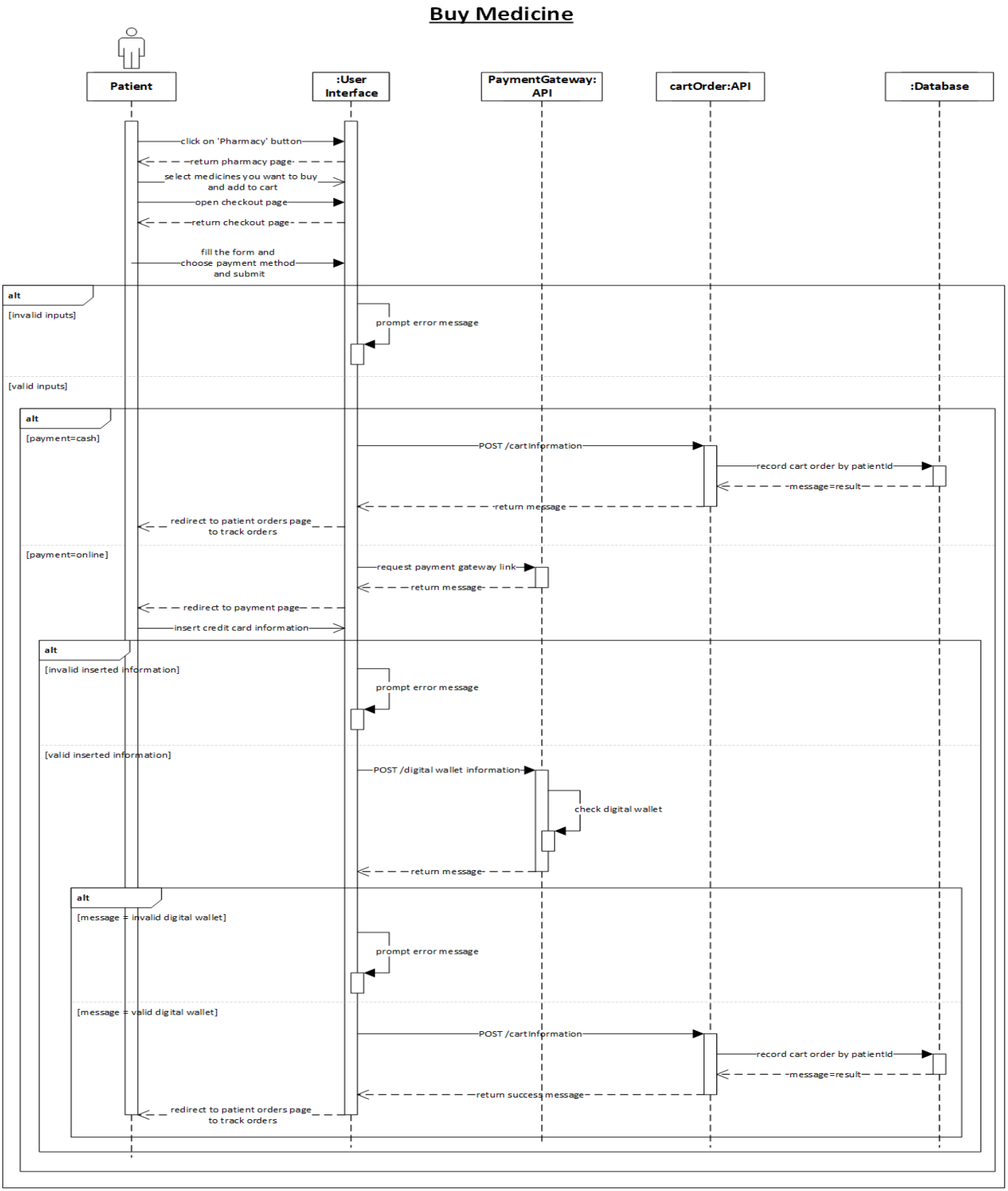


Figure 50: Patient Buy Medicine Sequence Diagram

View Doctor(s) / Specialization(s) / Medicine(s)

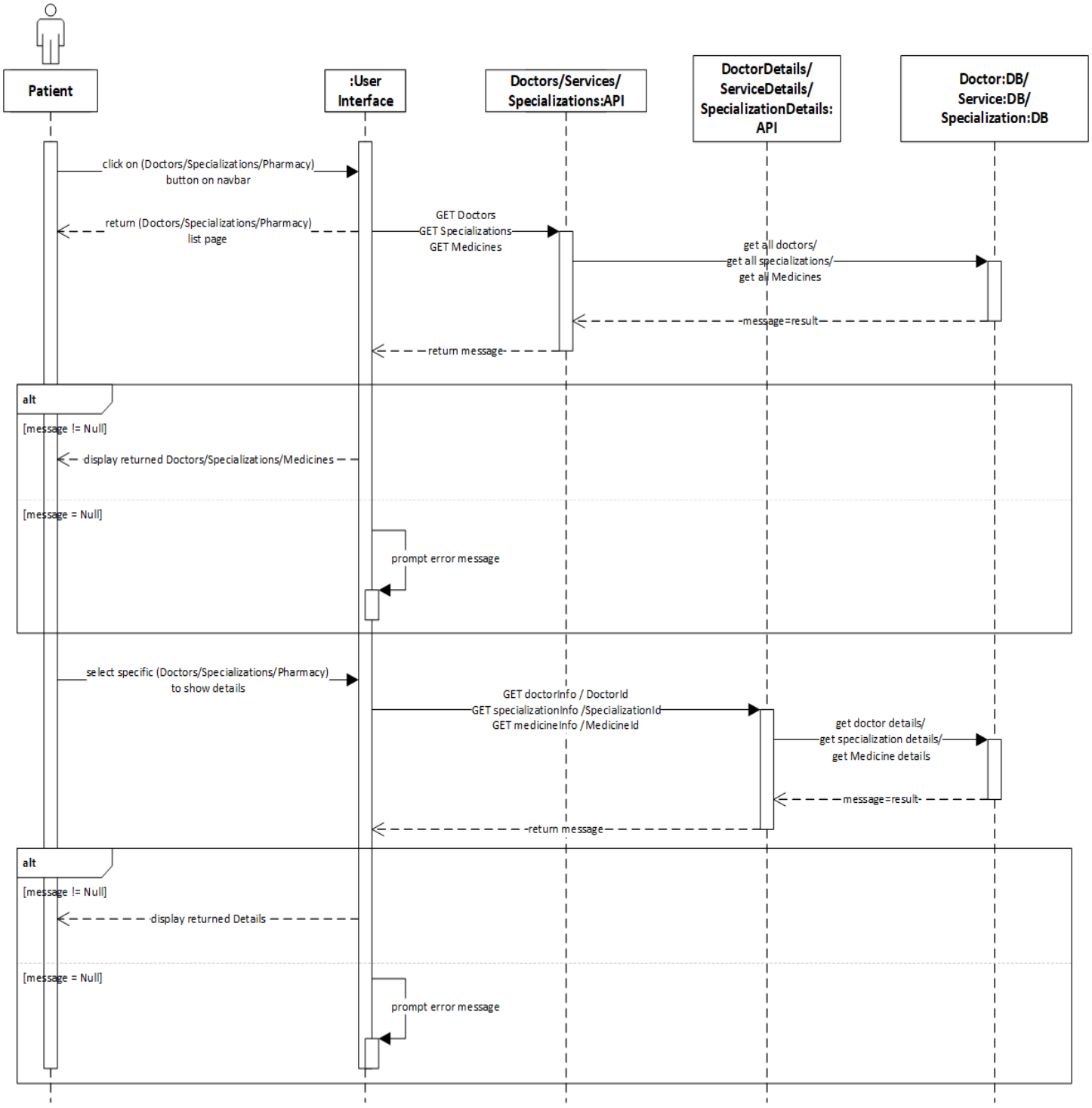


Figure 51: Patient View Doctor_Specialization_Medicine Sequence Diagram

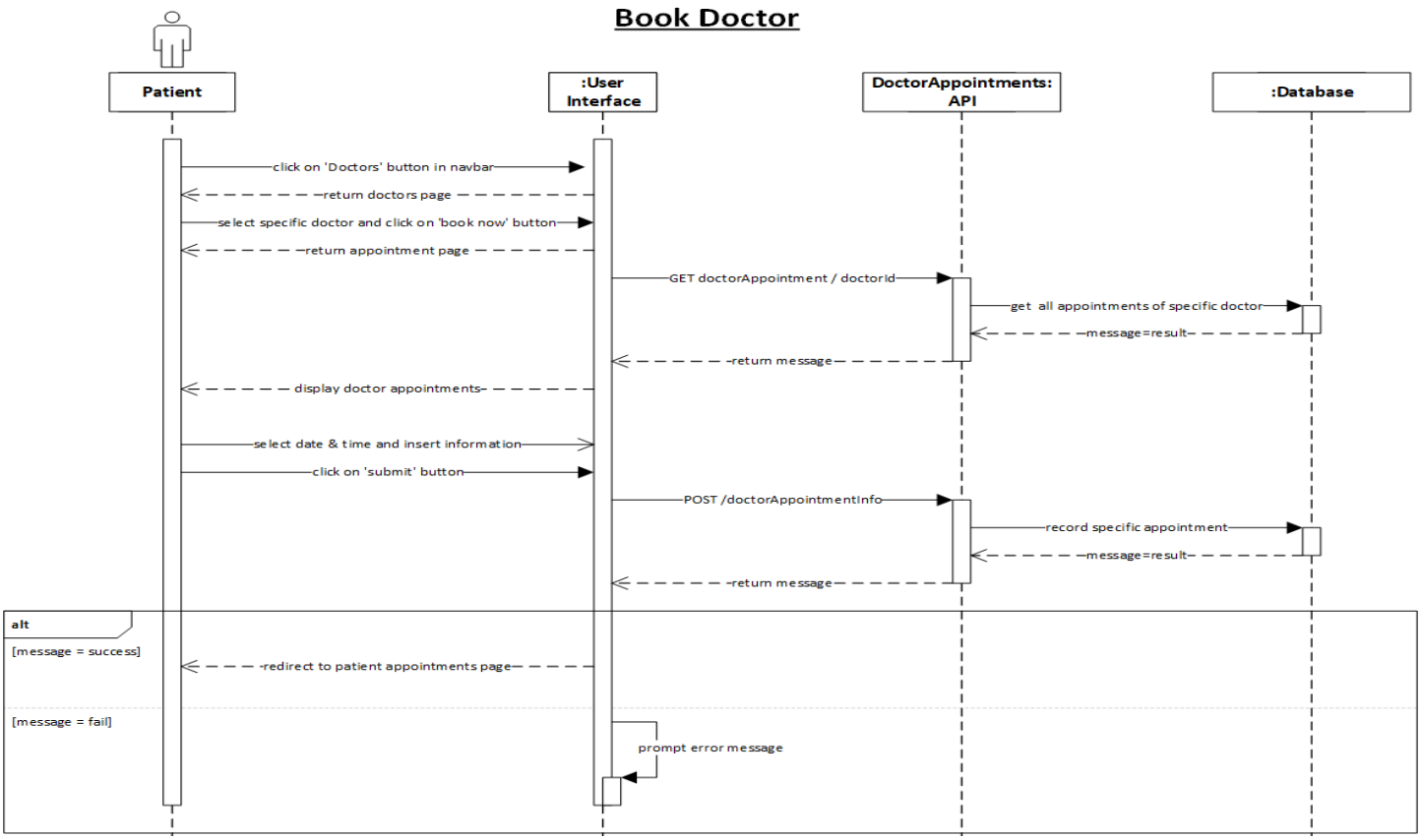


Figure 52: Patient Book Doctor Sequence Diagram

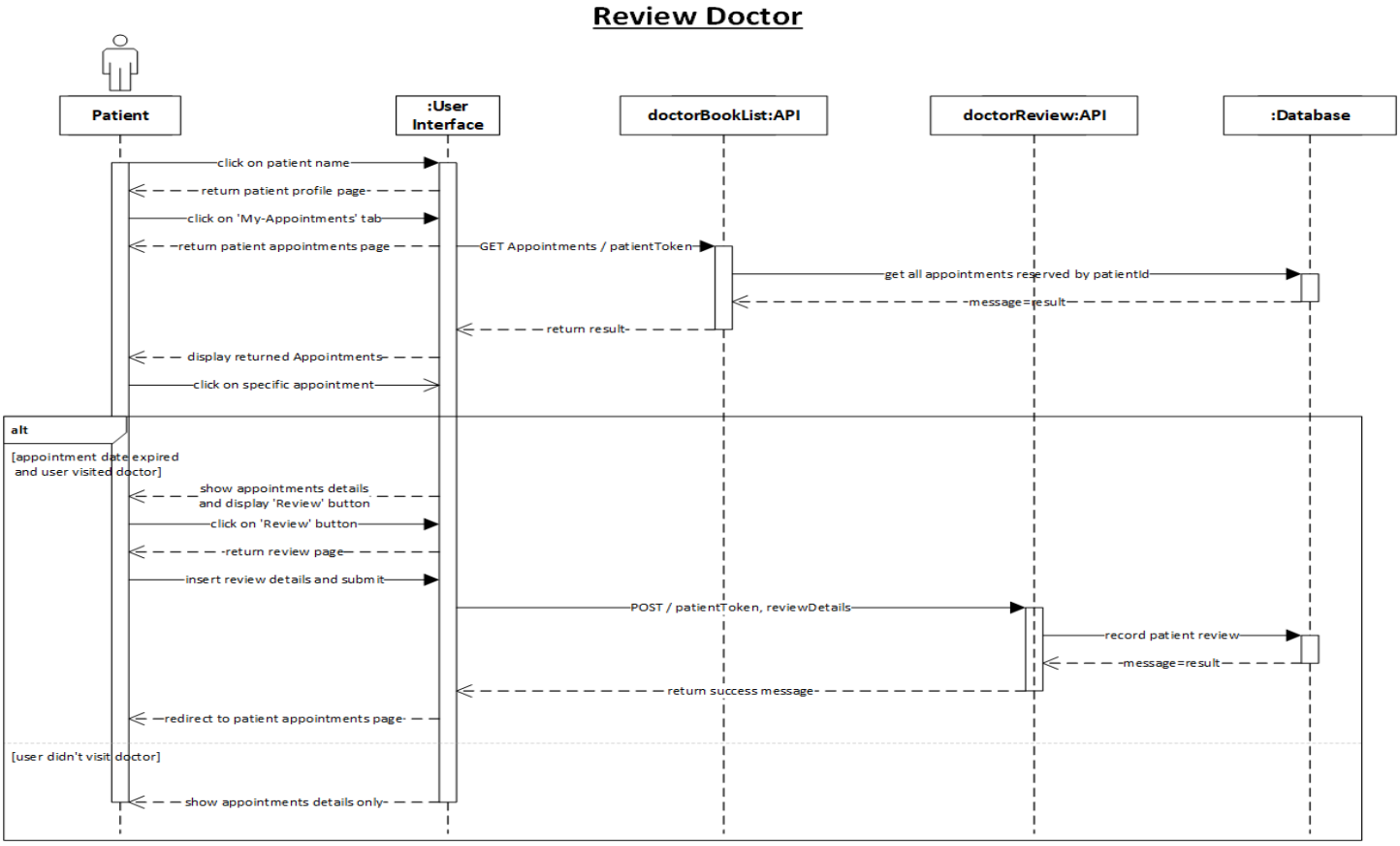


Figure 53: Patient Review Doctor Sequence Diagram

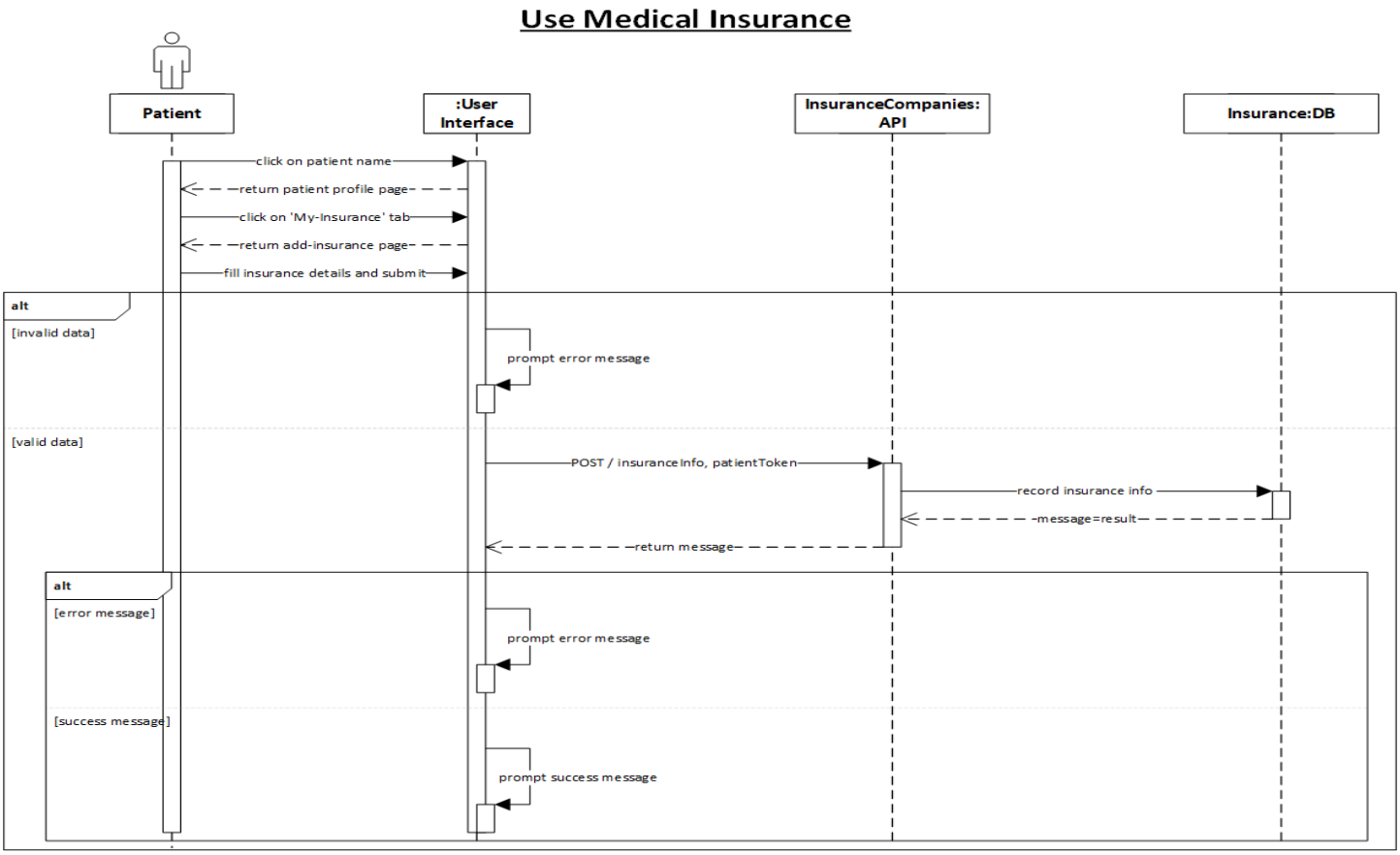


Figure 55: Patient use Medical Insurance

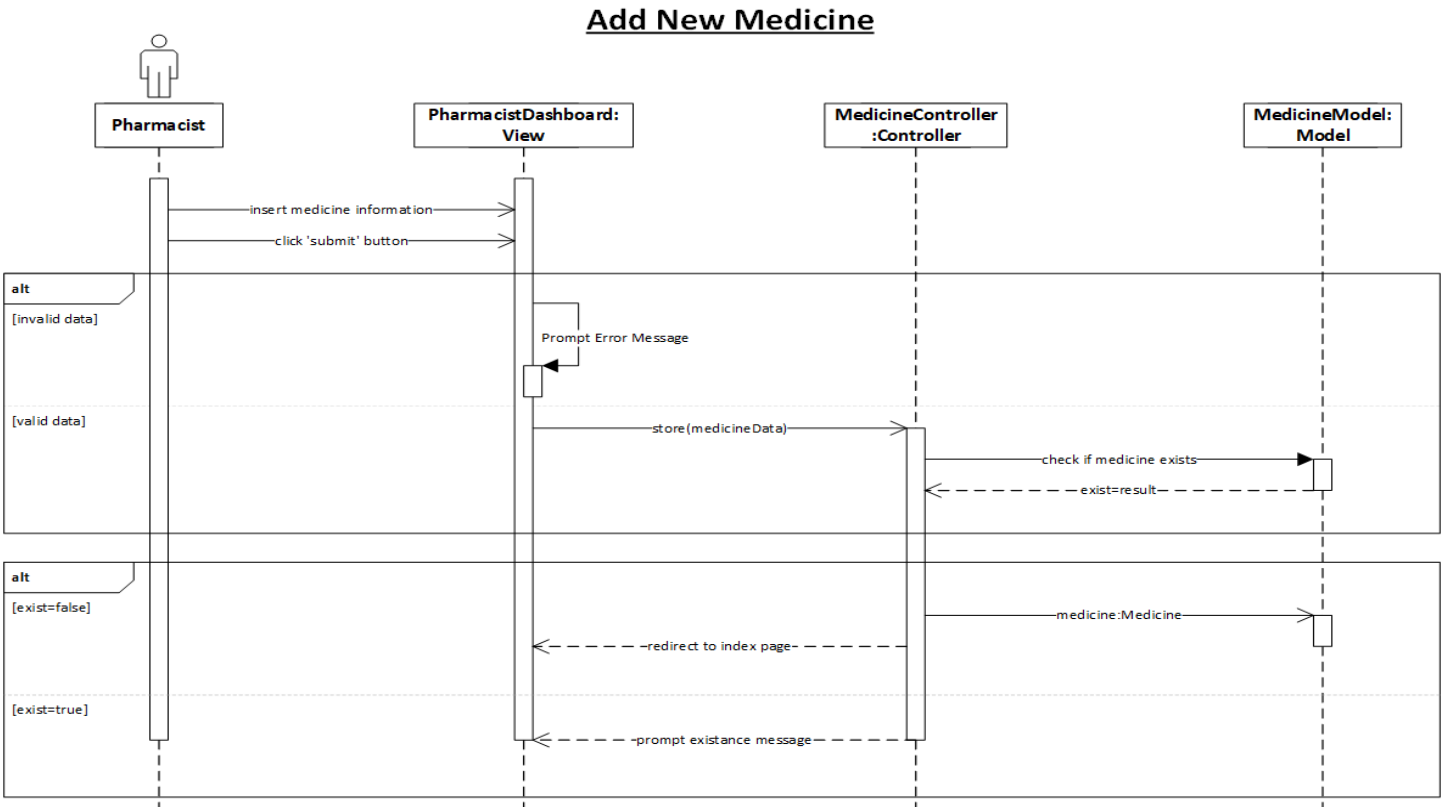


Figure 54: Pharmacist Add Medicine

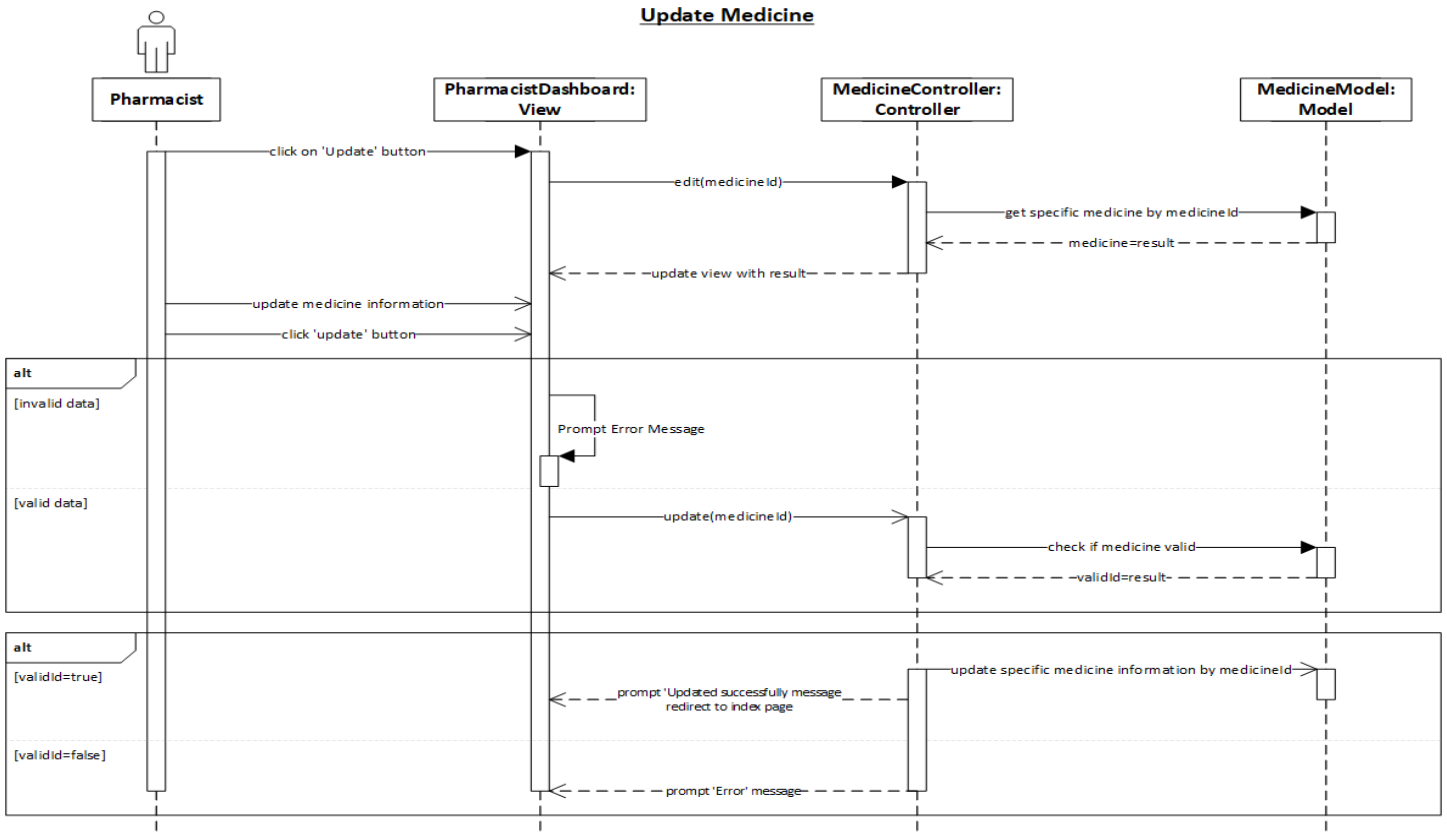


Figure 56: Pharmacist Update Medicine Sequence Diagram

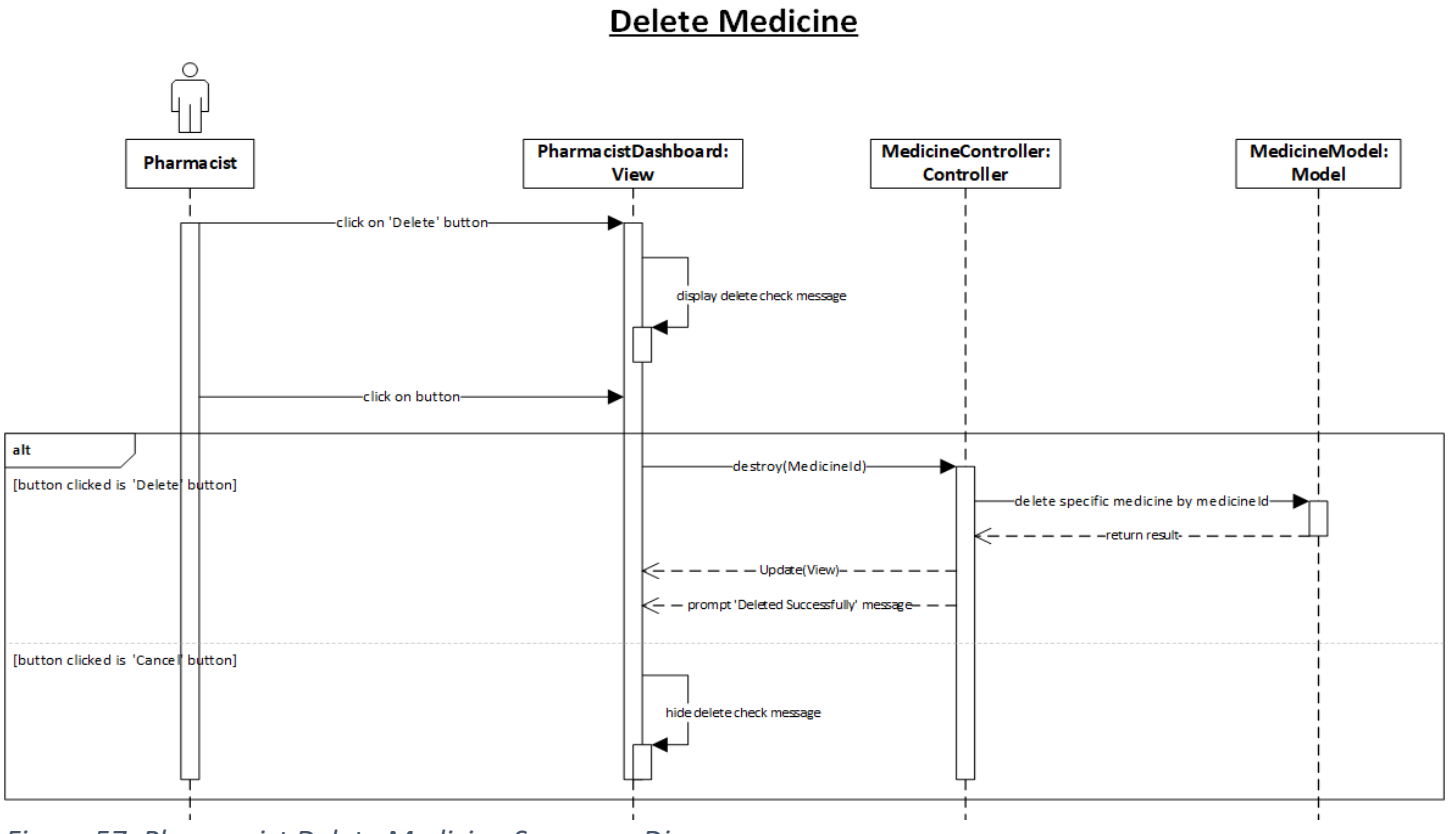


Figure 57: Pharmacist Delete Medicine Sequence Diagram

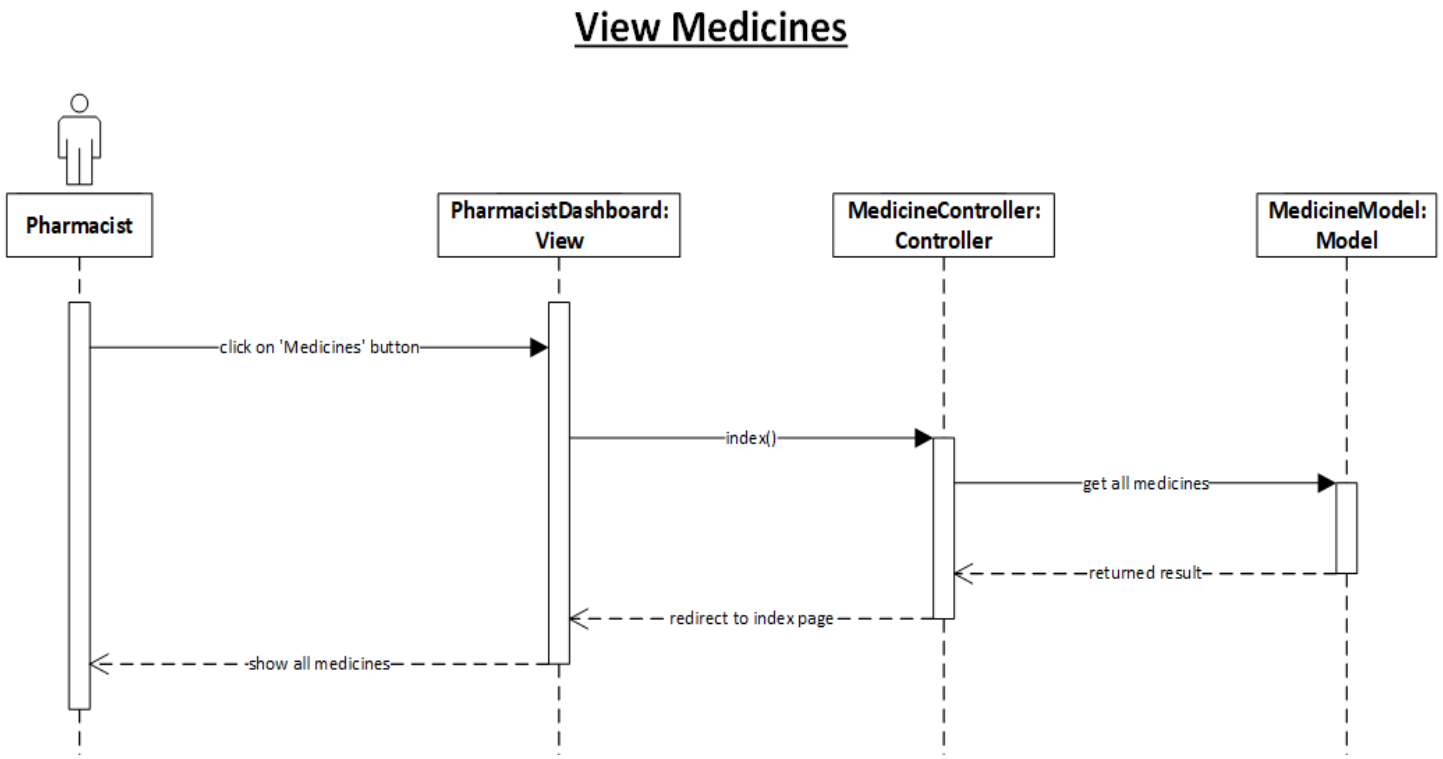


Figure 59: Pharmacist View Medicines Sequence Diagram

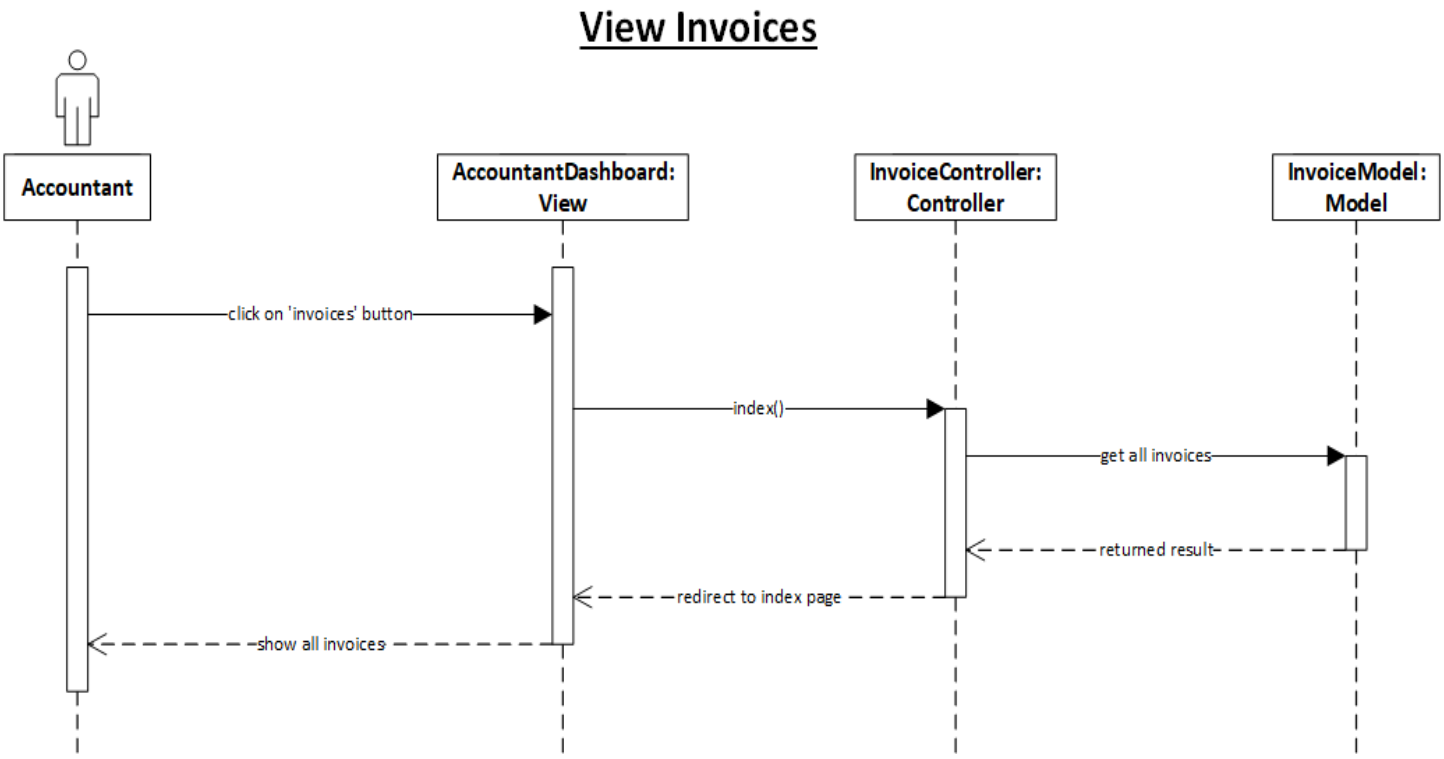


Figure 58: Accountant View Invoices Sequence Diagram

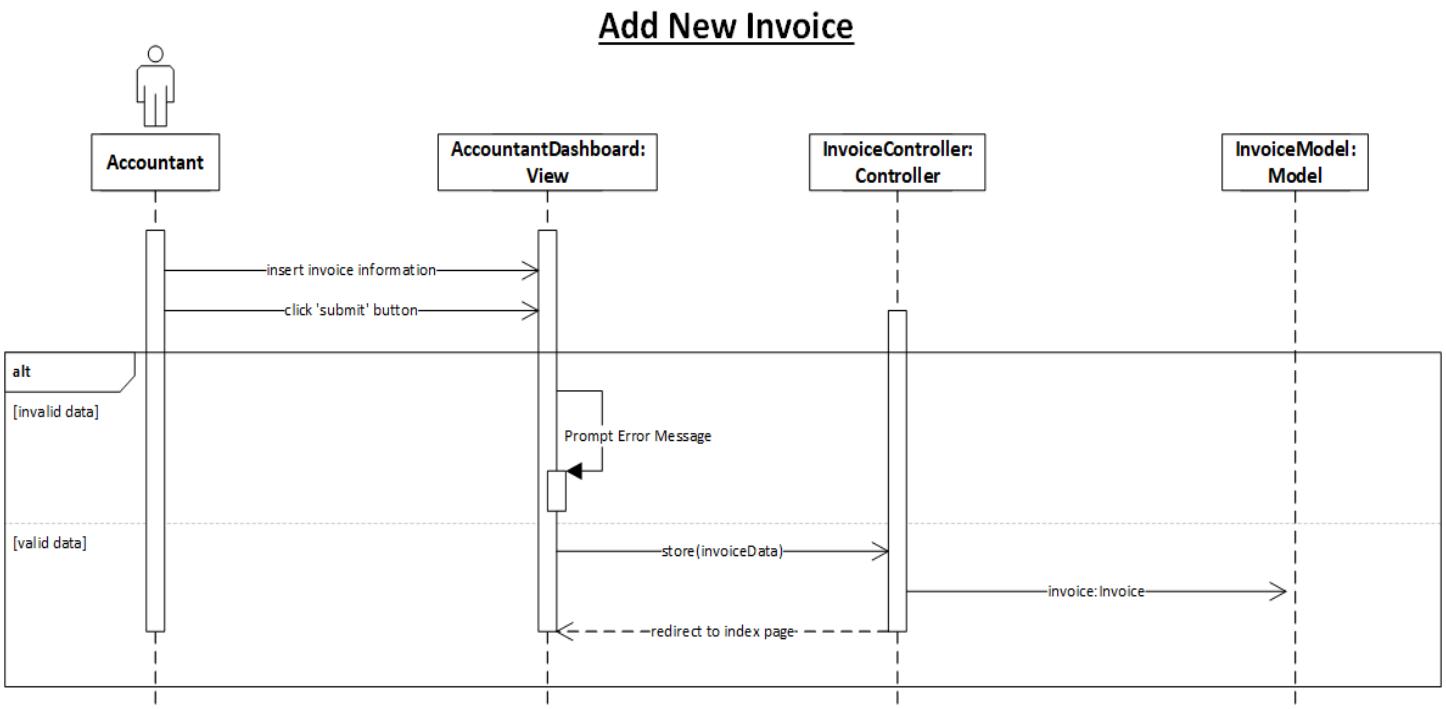


Figure 61: Accountant Add Invoice Sequence Diagram

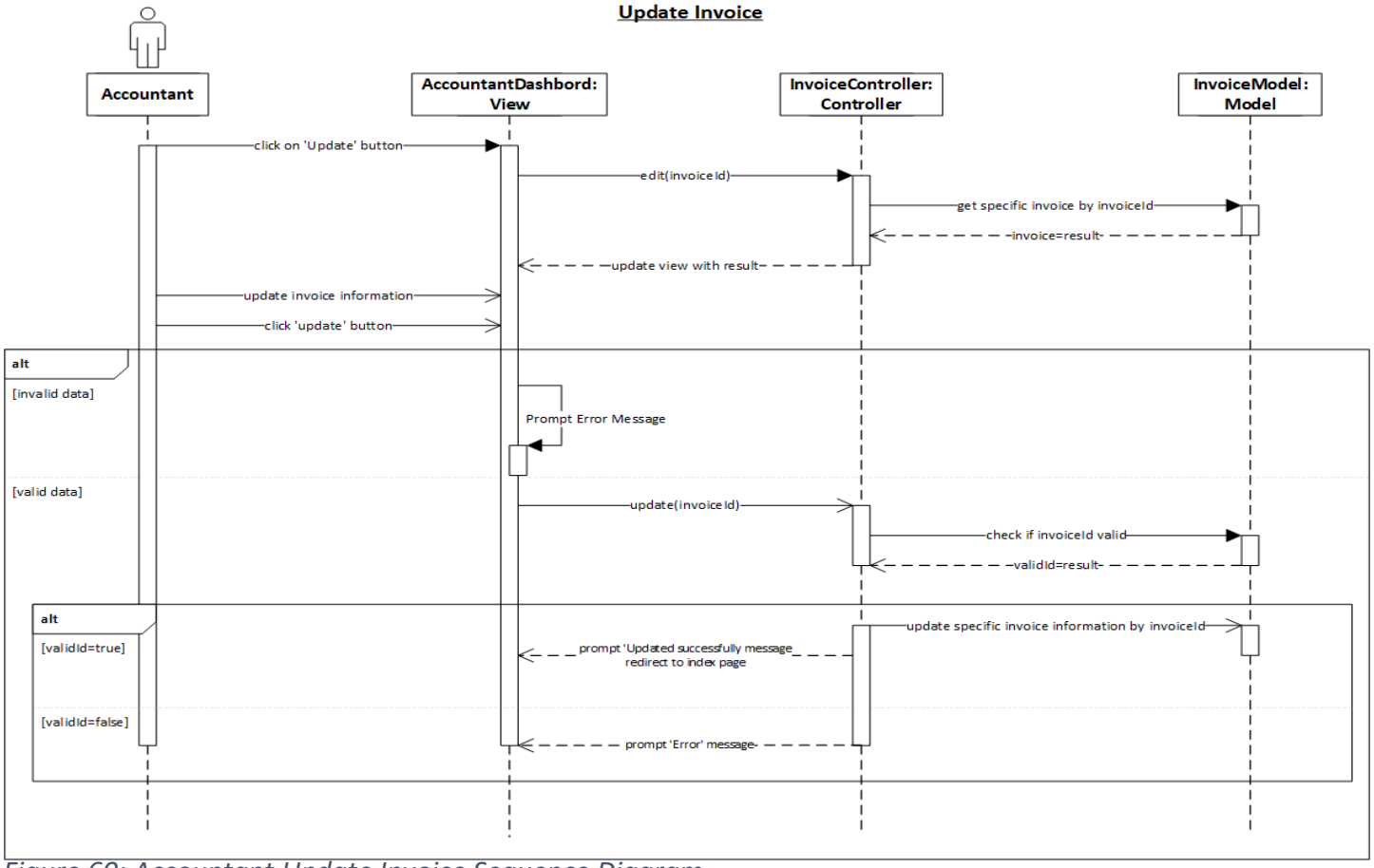


Figure 60: Accountant Update Invoice Sequence Diagram

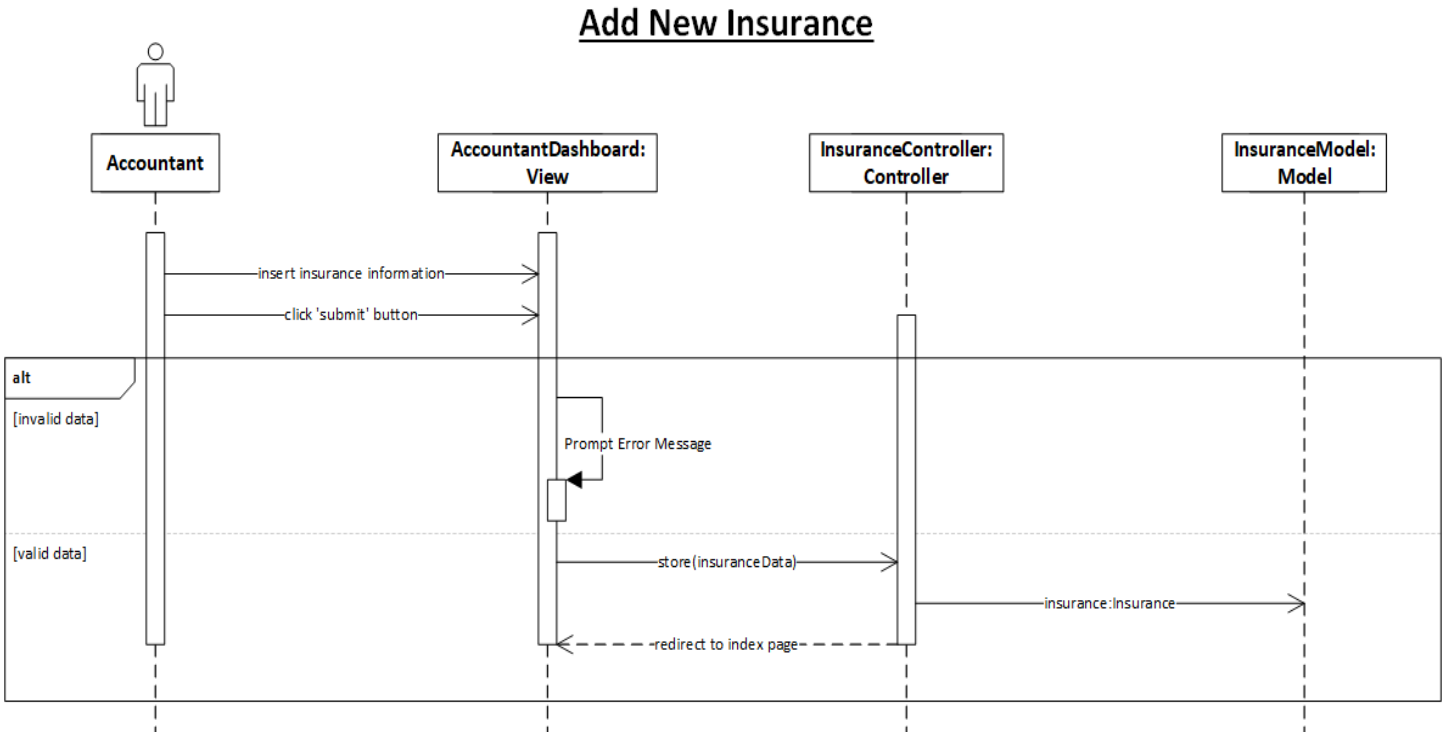


Figure 63: Accountant Add Insurance Sequence Diagram

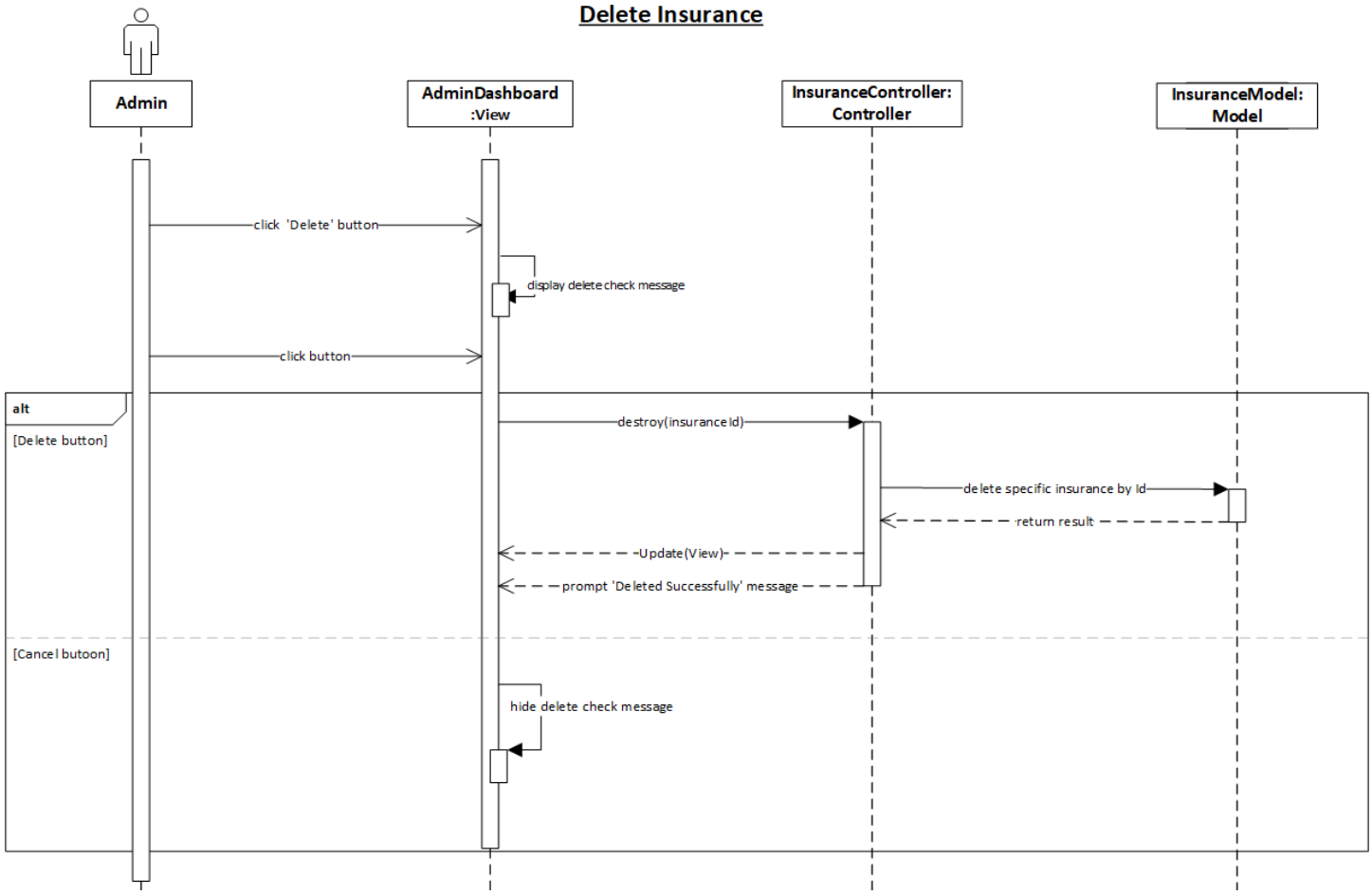


Figure 62: Accountant Delete Insurance Sequence Diagram

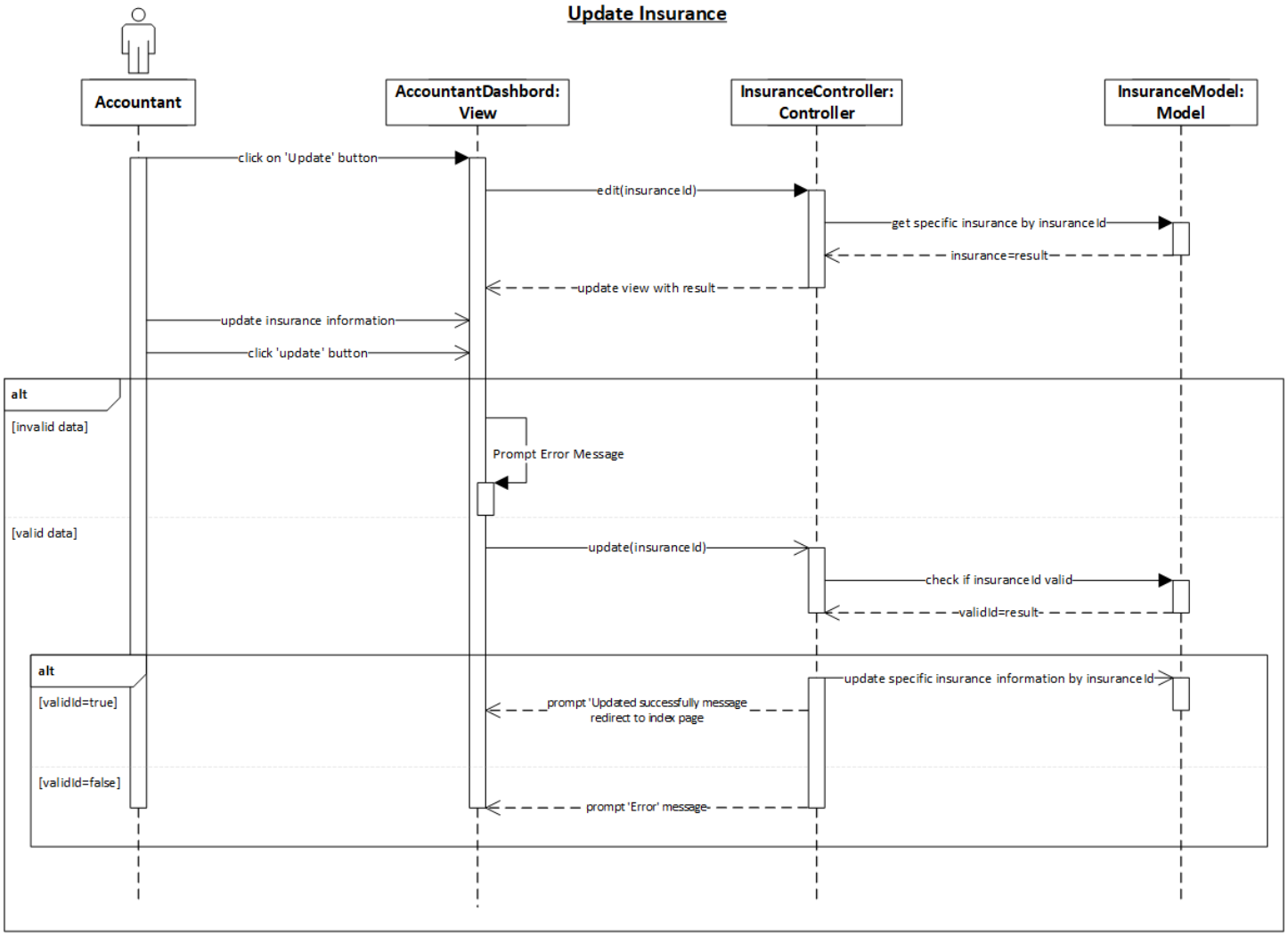


Figure 65: Accountant Update Insurance Sequence Diagram

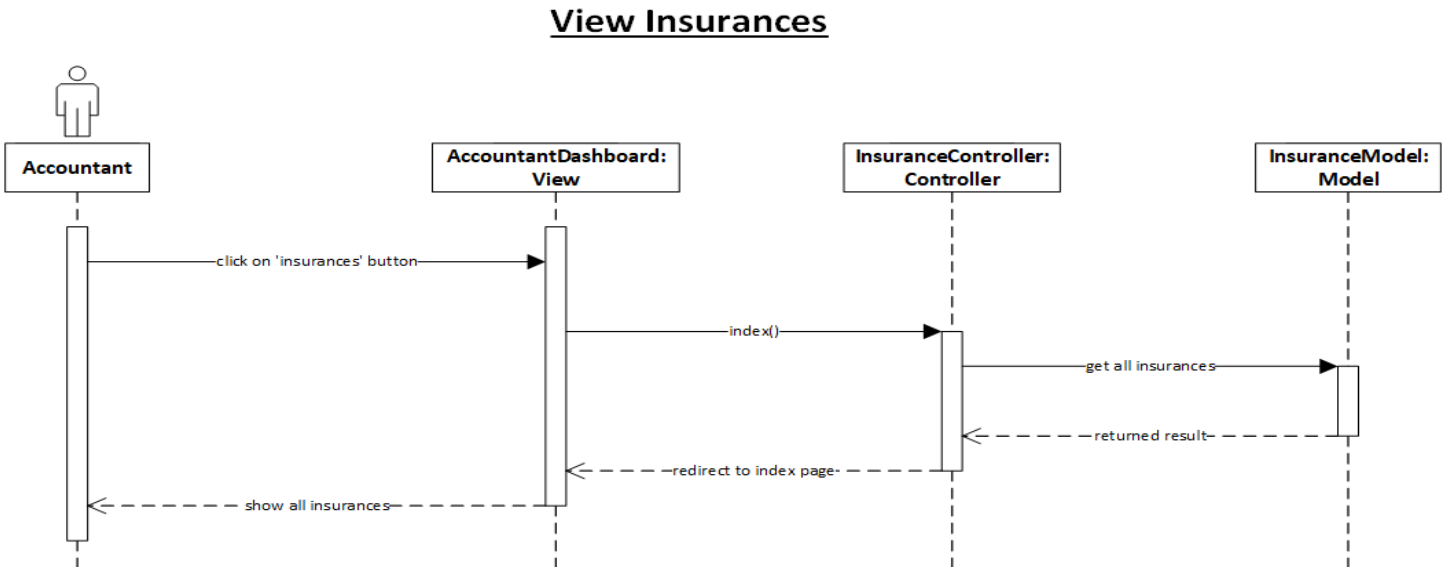


Figure 64: Accountant View Insurances Sequence Diagram

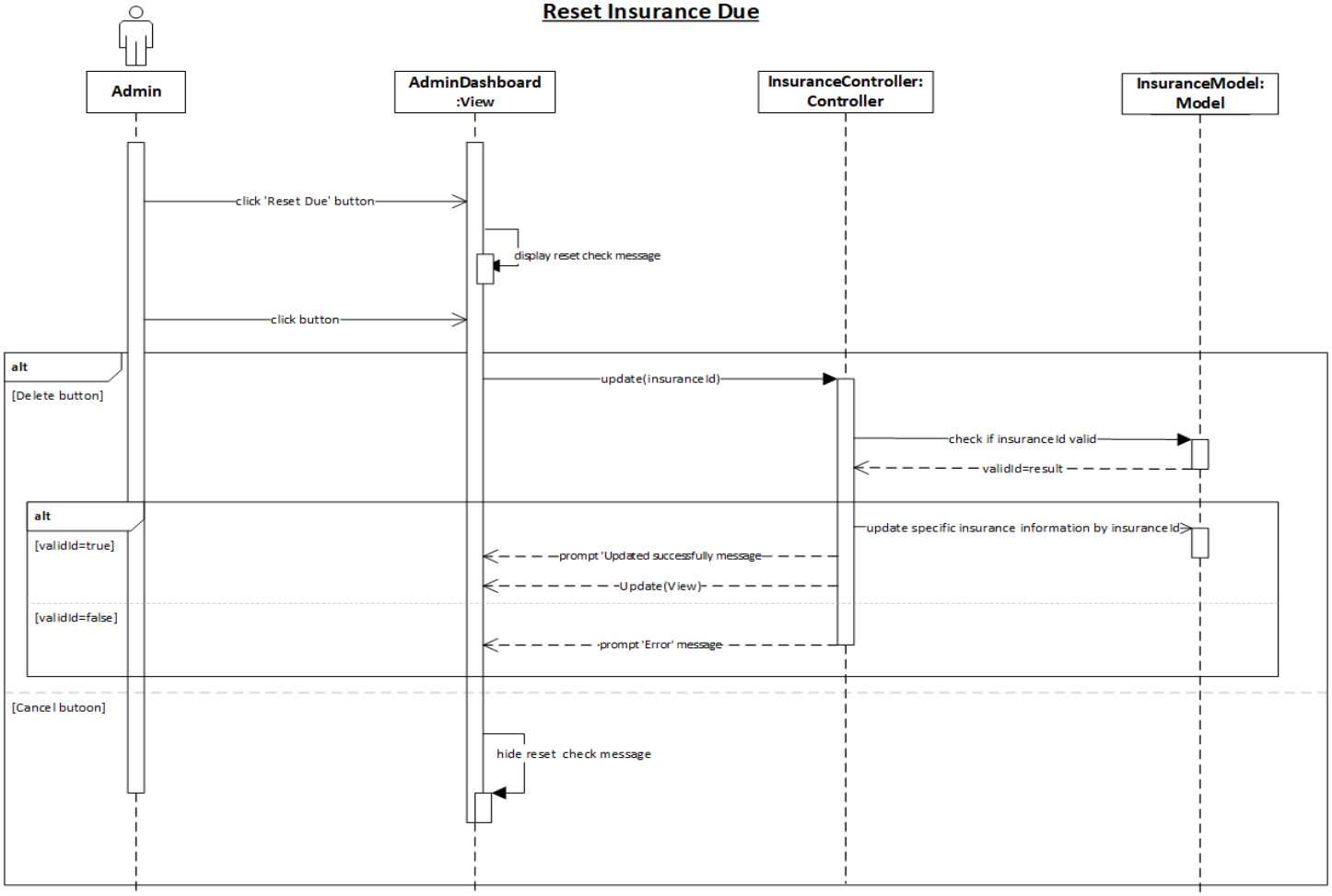


Figure 67: Accountant Reset Insurance Due Sequence Diagram

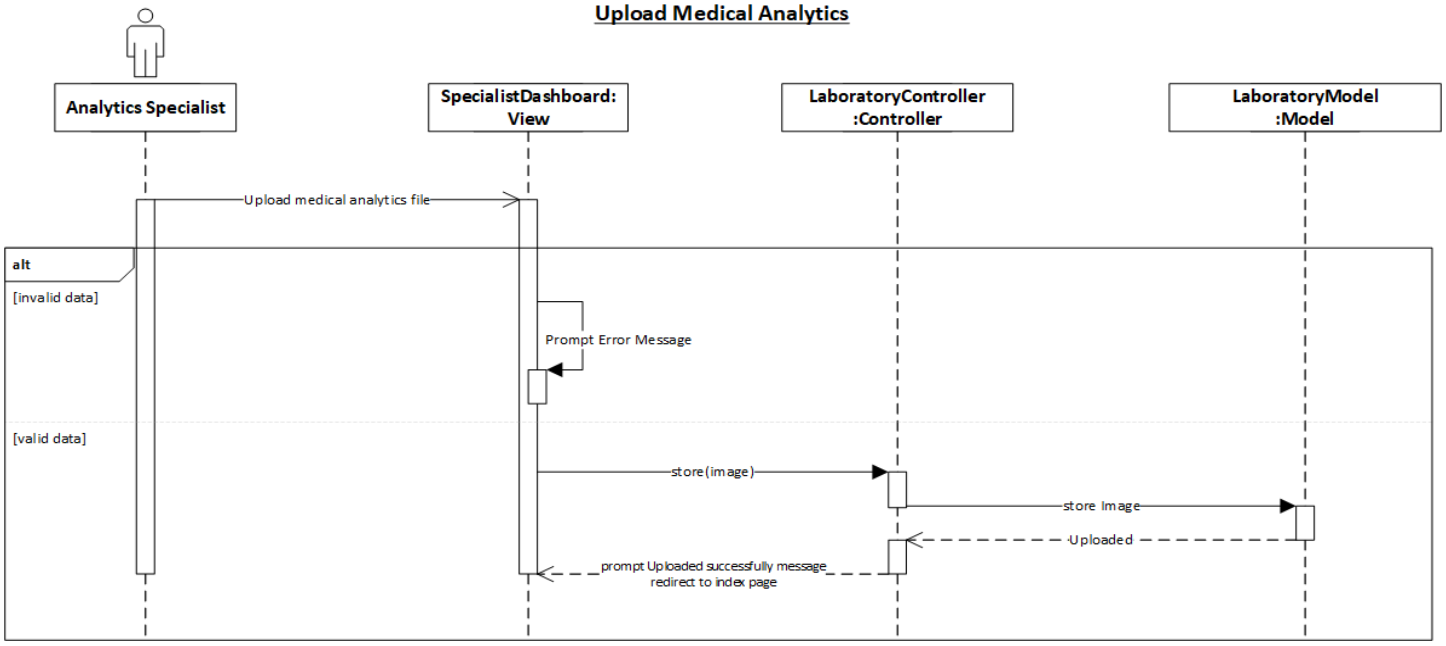


Figure 66: Analytics specialist upload medical analysis Sequence Diagram

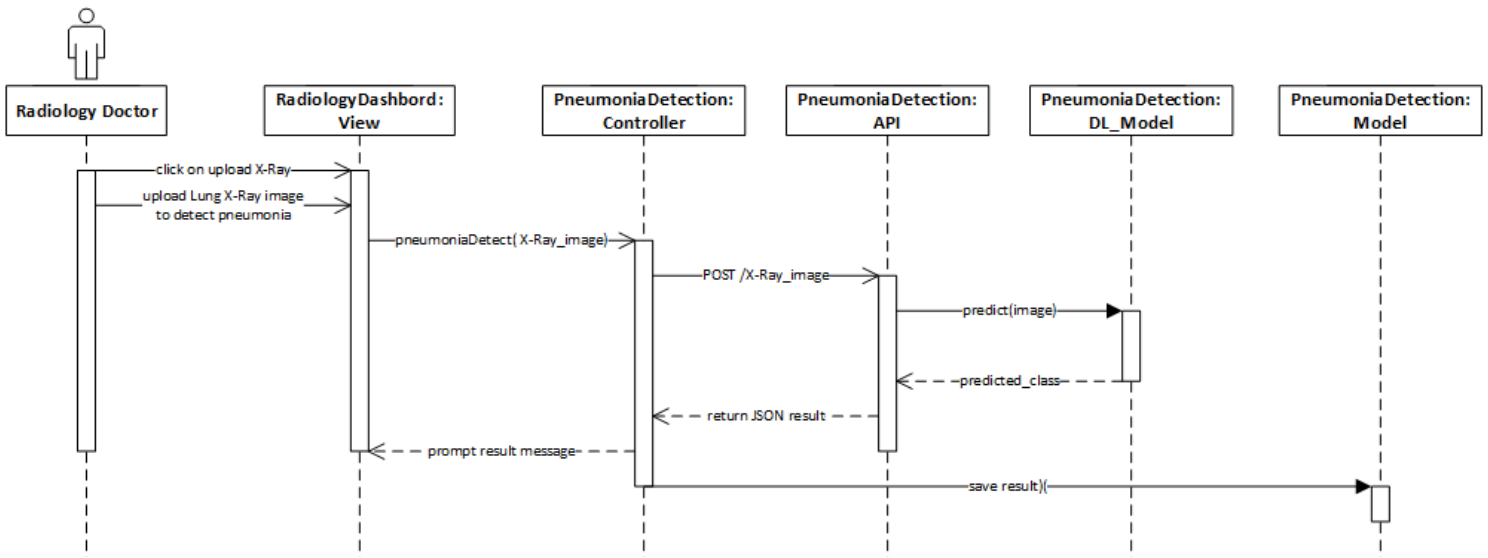


Figure 68: Radiology use Pneumonia Detection Service

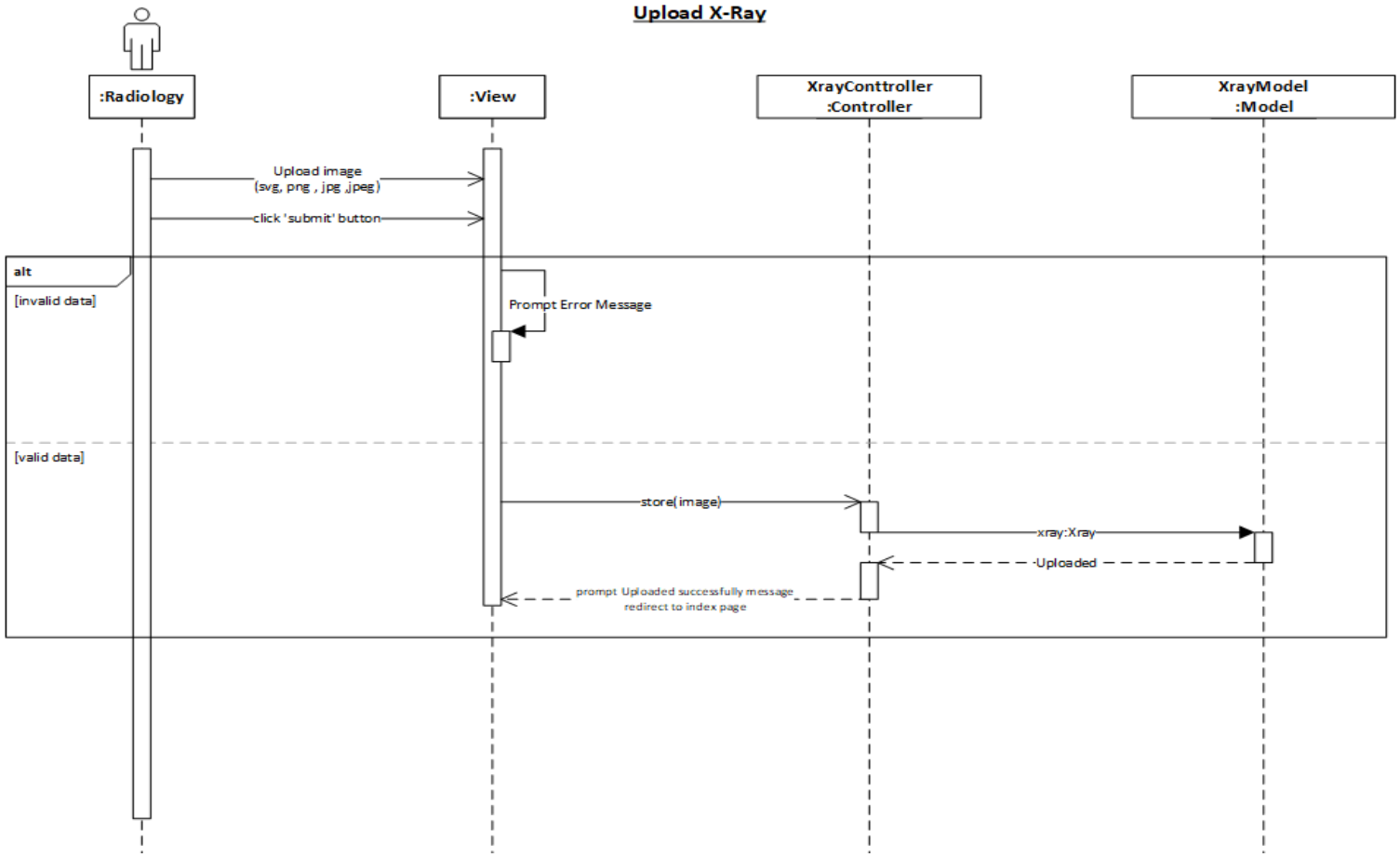


Figure 69: Radiology Upload X-Ray

3.3. System architecture

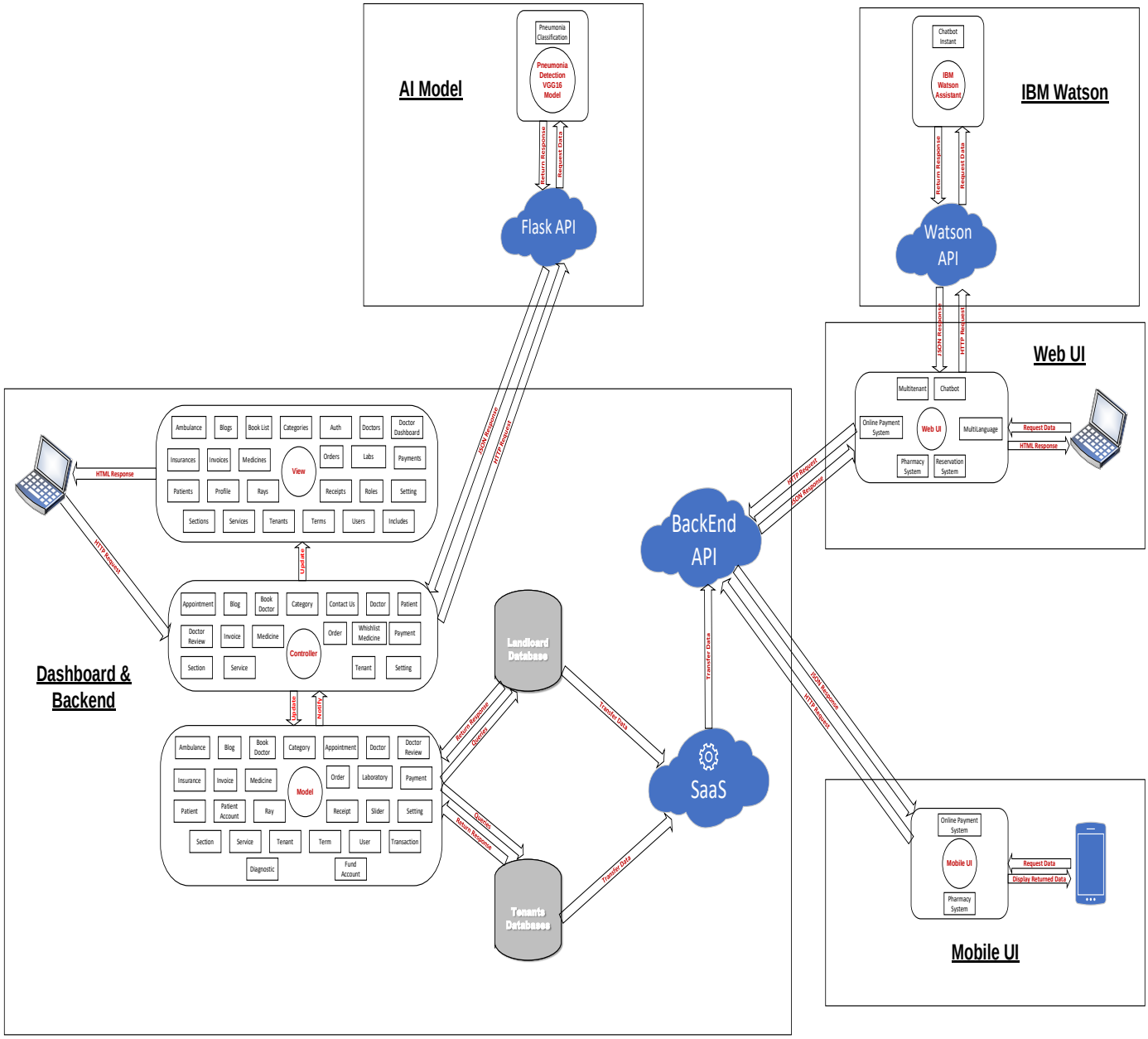


Figure 70: System Architecture

Chapter 4: Implementation

4.1. Description of Implementation

Implementation is the realization of a technical specification or algorithm as a software component or program. It involves the careful conversion of a program's design or design into some executable program code using any preferred programming language. Design can be implemented in different ways depending on the developer's priorities. In this work, many factors were taken into consideration during implementation.

The implementation of the hospital system involved creating a dedicated website to display available services and enable users to book appointments with doctors. Additionally, users could place purchase orders for available medicines and access informative articles about health. The website also included various sections, and an insurance feature was implemented to provide discounts on medicine purchases. To facilitate mobile access and medicine purchases, a mobile application was developed.

For the website's front-end, we used Angular, a framework that leverages TypeScript, HTML, and SCSS. The back-end was implemented using the PHP programming language with the Laravel framework. MySQL was chosen as the database management system for storing data. APIs were utilized to connect the front-end and back-end, enabling data retrieval and presentation to users. The mobile application was developed using the Flutter programming language, which also utilized APIs to connect to the back-end. Hosting the website was necessary for users to access its services.

4.2. Programming language and technology

The hospital system consists of two primary components: the front-end and the back-end, each employing different programming languages and technologies.

For the front-end, Angular framework was used, employing scripting languages such as JavaScript, TypeScript, and SCSS to support the creation of HTML.

The back-end was implemented using the PHP programming language, the Laravel framework. The MySQL database served as the data storage utilizing system. APIs were used to establish a connection between the front-end and back-end, enabling data retrieval and display to users.

On the mobile application side, Flutter framework was used, employing object-oriented client-optimized programming language such as Dart for mobile application development. It served as the client, allowing the creation of input and output forms. The mobile application also utilized the MySQL database as the back-end data storage system.

By utilizing these programming languages and technologies, we were able to develop a comprehensive hospital system with a website and a mobile application, providing users with easy access to services and medicine purchases.

4.2.1. The factors that influenced the choice of Angular

- **Reusability**. Components of similar nature are well encapsulated, in other words, self-sufficient. Developers can reuse them across different parts of an application. This is particularly useful in enterprise-scope projects where different systems may have many similar elements like search boxes, date pickers, sorting lists, etc.
- **Readability**. Encapsulation also ensures that new developers – who've been recently onboarded to a project – can read code better and eventually reach their plateau of productivity faster.
- **Unit-test friendly**. The independent nature of components simplifies unit tests, and quality assurance procedures aimed at verifying the performance of the smallest parts of the application, units.
- **Maintainability**. Components that are easily decoupled from each other can be easily replaced with better implementations. Your engineering team will be more efficient in maintaining and updating the code within the iterative development workflow.

4.2.2. The factors that influenced the choice of PHP

- **Speed**: Being a compiled language, it is very fast, and speed is important in database application.
- **Environment**: It can run in windows.
- **Efficient**: The final code tends to be compact and runs quickly.
- **Portability**: If compiled, it can be executed in different machines with alteration of source code.
- **Maintainability**: To ensure maintainability, this program is broken into modules and each module is assigned a specific function. This will make maintenance of the system easier.
- **Security**: it has proper backups, quality control mechanism for all modules and unauthorized access to sensitive data is prohibited.
- PHP supports full **object-oriented programming** giving us more control over the graphic user interface

- PHP supports all the new **AJAX and CSS**. It makes the graphic user interface friendly.

4.2.3. Why MySql database was chosen?

- MYSQL maintains a high level of security.
- MYSQL database ensures maximum data throughput (i.e., accepting of data with
- MYSQL database has a very high data storage capacity limit, several Nano byte and terabytes.
- MYSQL is multiplatform working on all platforms, Linux, Os X and mobile platform.
- MYSQL together with built-in front end (client) and back end (DB server) such as MYSQL workbench or PhpMyAdmin has several data management and administrative services.
- MYSQL has data backup and recovery management services.
- MYSQL is an open sources application.
- MYSQL can be installed as a cluster server- this makes it possible for two or more MYSQL database servers to be united as a common server in a cluster server.

4.2.4. Why did we use Flutter in Mobile App

- **Reduced Development Time:** The requirements for Flutter application development are much lower. So, the positive outcome is that there are no additional maintenance charges. Flutter makes it possible to create larger apps that use unique features.
- **Native-like Performance:** This is one of the Flutter advantages that stands out the most. Flutter works with Skia, a graphics engine which enables quick and well optimized development. It also is indistinguishable from native apps as it doesn't rely on interpreters or intermediary code representations.
- **Powerful community:** According to statista, Flutter has become one of the most popular frameworks and a first choice by developers globally. Over 40 percent of software developers have chosen Flutter over the course of the last three years. The following chart shows the growing interest in Flutter in comparison with other cross-platform app tools.
- **Hot Reload Feature:** The ability to hot reload is one of the main benefits of using Flutter. This is for effective cross-platform development so it can complement the nature of Flutter. This feature's function speeds up application development.

4.3. Part of Implementation

Frontend Login Method

```
login() {
    const model: Login = {
        email: this.loginForm.value.email,
        password: this.loginForm.value.Password
    }
    this.isVisibleSpinner = true;
    this._AuthService.login(model).subscribe((res: any) => {
        localStorage.setItem("token", "Bearer " + res.data.token)
        localStorage.setItem("patient_id", res.data.id)
        this._DataService.is_login.next(true);
        this.router.navigate(['/home'])
        this.toastr.success(!this.rtlDir?`Signed in successfully`:`تم تسجيل الدخول بنجاح`)
        this.isVisibleSpinner = false;
    }, (error) => {
        this.toastr.error(!this.rtlDir?`Email Or Password is incorrect`:`البريد الالكتروني او كلمة المرور خاطئة`,error.error.data.error )
        this.isVisibleSpinner = false;
    })
}
```

Figure 71: Frontend Angular Login Method

Frontend Add to Cart Method

```
// add medicine to cart when clicking button
addCart(medicineId:number) {
    if (localStorage.getItem("token") == null) { // in case that user isn't login
        this.toastr.info(!this.rtlDir?`You Should Login First!`:`يجب أن تسجل الدخول أولا!`)
    } else {
        this.isVisibleAddSpinner = true;
        this._CartService.addCart(medicineId).subscribe({ // calling API in Service
        next:(message)=>{
            if(message.success == true) {
            this._CartService.calculateTotalQty(); // calculate cart medicines total quantity
            setTimeout(() => {
                this.isVisibleAddSpinner = false;
                this.toastr.success(!this.rtlDir?`This Medicine Added to Cart`:`تم اضافة الدواء الى عربة الشراء`, !this.rtlDir?`Cart Result`:`ناتج عربة الشراء`)
            }, 700);
            } else {
            this.isDisableBtn = true;
            setTimeout(() => {
                this.isVisibleAddSpinner = false;
                this.toastr.info(!this.rtlDir?`You have reached the Max amount of this medicine`:`لقد وصلت إلى الحد الأقصى المسموح لهذا الدواء`)
            }, 700);
            }
        },
        error:(error)=>{
            this.toastr.error(this.rtlDir?`An Error has occured`:`حدث خطأ ما` , `${error}`)
        }
        })
    }
}
```

Figure 72: Frontend Angular Add Medicine to Cart Method

Frontend Get Doctor's Appointments Dates Method

```
//This code is used to get a list of bookings for a particular doctor on a given date
getBookDoctorList(convertDate:any) {

    const model: any = {
        doctor_id: parseInt(this.singleDoctorId),
        date: convertDate,
    }
    console.log(model);
    // passing in the model object as a parameter
    this.bookDaySubscription = this._AppointmentsService.getAllTimeBookeed(model).subscribe({
        //The response from this method is stored in the BookDayAppointment variable
        next: (BookDayAppointment) => {
            console.log(BookDayAppointment)
            this.timeBookList=[];
            this.listOfDayBook = BookDayAppointment.data;
            this.filterintervals = this.intervals
            for(let x = 0  ; x < this.listOfDayBook.length; x++){
                const new_day =  this.listOfDayBook[x].date
                const edit_timeBook = this.listOfDayBook[x].time?.substring(0,5);
                console.log(edit_timeBook)
                if(new_day === convertDate && parseInt( this.singleDoctorId) === this.listOfDayBook[x].doctor.id){
                    this.filterintervals = this.filterintervals.filter(time => time !== edit_timeBook)
                    if(!this.timeBookList.includes(edit_timeBook)){
                    console.log(!this.timeBookList.includes(edit_timeBook))
                    this.timeBookList.push(edit_timeBook)
                    }
                }
            }
        }
    })
}
```

Figure 73: Frontend Angular Get Doctors Appointments Date Method

Frontend Review Doctor Guard Logic

```
canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
        return new Observable<boolean>((observer) => {
            this._DataService._lang.subscribe({next:(language)=>{
                // to get all diagnosis
                this.AppointmentsSubscription = this._PatientProfileService.getPatientAppointments(language).subscribe({
                    next: (Appointments) => {
                        this.appointmentData = Appointments.data;
                        this.doctorId = route.paramMap.get('id')
                        const currentDate = new Date();
                        let flag = false
                        this.appointmentData?.forEach((appointment:any) => {
                            const appointmentDate = new Date(appointment.date);
                            if(appointment.doctor.id === parseInt(this.doctorId)  && (appointmentDate) <= (currentDate)  ){
                                observer.next(true);
                                observer.complete();
                                flag = false
                            } else {
                                flag = true
                            }
                        });
                        if(flag){
                            flag=false
                            this.toastr.error(`please book this doctor to review this doctor`)
                            this.router.navigate(['/appointments/'+this.doctorId])
                            observer.next(false);
                            observer.complete();
                        }
                    }
                });
            }});
        })
    }
```

Figure 74: Frontend Angular Review Doctor's Guard Logic

Backend Tenant Logic

```php
namespace App\services;

use App\Models\Tenant;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\DB;
use Illuminate\Validation\ValidationException;

class TenantService
{

    private $tenant;
    private $domain;
    private $database;


    public function switchToTenant(Tenant  $tenant)
    {
       if($tenant instanceof Tenant) {
        DB::purge('landlord');
        DB::purge('tenant');
        Config::set('database.connections.tenant.database',$tenant->database);
        $this->tenant = $tenant;
        $this->domain = $tenant->domain;
        $this->database = $tenant->database;
        DB::reconnect('tenant')->reconnect();
        DB::setDefaultConnection('tenant');
    } else {
        throw ValidationException::withMessages(['field_name'=>'This Value is incorrect']);
    }
        // dd($tenant->database);
    }

    public function switchToDefault()
    {
        DB::purge('landlord'); //remove cache in sql
        DB::purge('tenant');
        DB::reconnect('landlord')->reconnect();
        DB::setDefaultConnection('landlord');
    }

    public function getTenant(){
        return $this->tenant;
    }
}
```

Figure 75: Backend Laravel Tenants Logic

Backend Store Insurance Method

```php
public function store($request)
{
    DB::beginTransaction();
     // حفظ في جدول الفواتير
    $invoice = new Invoice();
    $invoice->patient_id = $request->patient_id;
    $invoice->doctor_id = $request->doctor_id;
    $invoice->type =  $request->type;
    $invoice->service_id =  $request->service_id;
    $invoice->price =  $request->price;
    $invoice->discount_value =  $request->discount_value;
    $invoice->tax_rate =  $request->tax_rate;
    $invoice->tax_value =  $request->tax_value;
    $invoice->total_with_tax =  $request->total_with_tax;
    $invoice->invoice_date =  date('Y-m-d');
    $invoice->save();
    // فاتورة نقدي
    if($request->type == 1)
    {
        // حفظ في جدول الصندوق
        $fund_account = new FundAccount();
        $fund_account->date =  date('Y-m-d');
        $fund_account->invoice_id = $invoice->id;
        $fund_account->debit = $invoice->total_with_tax;
        $fund_account->credit = 0.00;
        $fund_account->save();
    }
    // فاتورة اجل
    else {
        // حفظ في جدول حسابات المريض
        $patient_account = new PatientAccount();
        $patient_account->date =  date('Y-m-d');
        $patient_account->invoice_id = $invoice->id;
        $patient_account->patient_id = $invoice->patient_id;
        $patient_account->debit = $invoice->total_with_tax;
        $patient_account->credit = 0.00;
        $patient_account->save();

    }
    DB::commit();
    return redirect()->route('invoices.index')->with(['success' => 'Invoice Added Successfully']);
}
```

Figure 76: Backend Laravel Store Insurance Method

Add to cart

```dart
class ProductDetailsPage extends StatelessWidget {
  final String title;
  final String imageUrl;
  final double price;
  final String description;
  final int id;
  ProductDetailsPage(
      {required this.title,
      required this.imageUrl,
      required this.price,
      required this.description,
      required this.id});
  final storage = FlutterSecureStorage();
  bool isFav = false;
  Future<String?> getToken() async {
    String? token = await storage.read(key: 'Token');
    return token;
  }

  void addToCart() async {
    String baseUrl = 'http://medibookidashbord.test/api/patient/orders/';
    int medicineId = id;
    final token = await getToken();
    final url = Uri.parse(baseUrl + '$medicineId');
    final response =
        await http.get(url, headers: {'Authorization': 'Bearer $token'});
    if (response.statusCode == 200) {
      // Successful request
      Fluttertoast.showToast(
        msg: "Added to cart Successfully",
        toastLength: Toast.LENGTH_SHORT,
        gravity: ToastGravity.TOP,
        backgroundColor: Colors.green,
        textColor: Colors.white,
        timeInSecForIosWeb: 2,
      );
      final res = jsonDecode(response.body);
      print(res);
      print('Medicine added to cart successfully!');
    } else {
      // Handle the error response
      Fluttertoast.showToast(
        msg: "Added to cart Failed",
        toastLength: Toast.LENGTH_SHORT,
        gravity: ToastGravity.TOP,
        backgroundColor: Colors.green,
        textColor: Colors.white,
        timeInSecForIosWeb: 2,
      );
      print('Error: ${response.statusCode}');
    }
  }
```

Figure 77: Mobile Flutter Add Medicine to Cart Logic

4.4. Project Mockup:

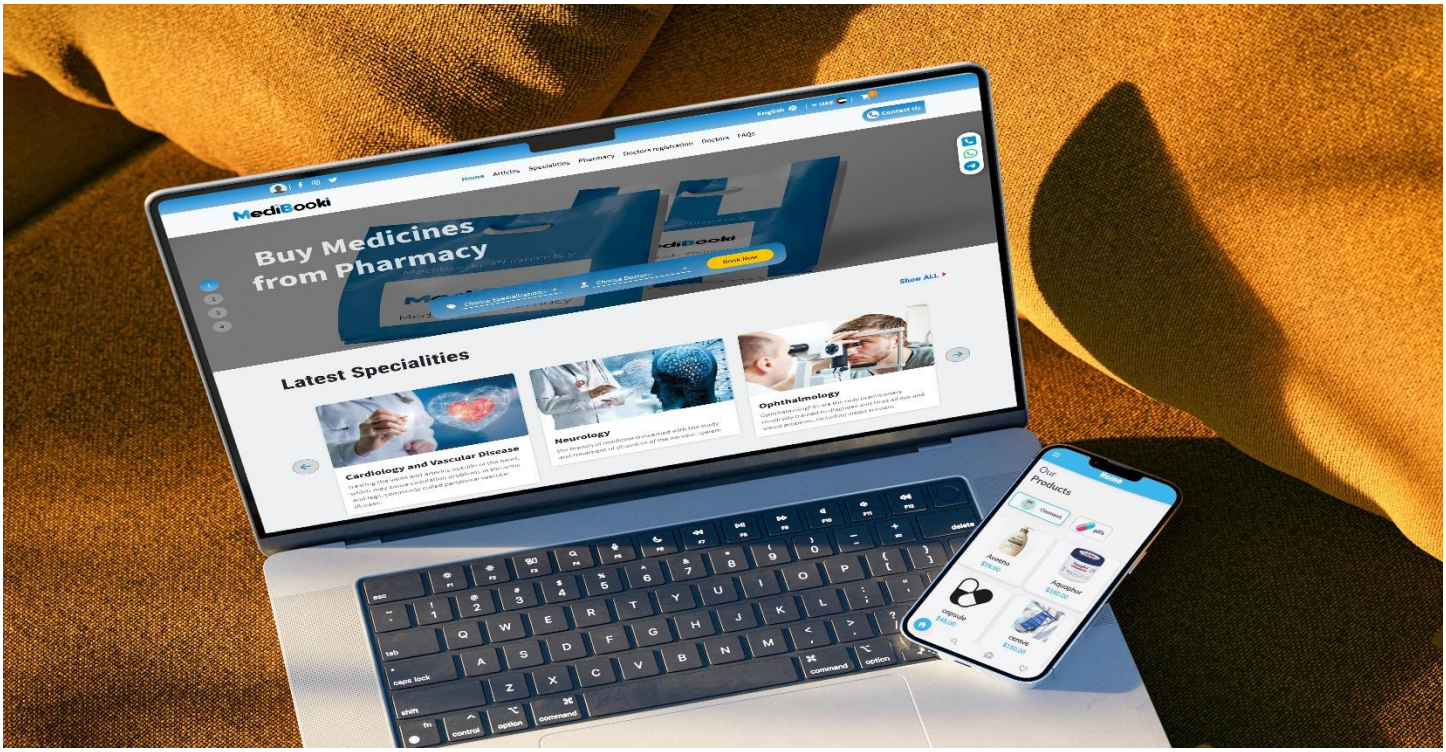


Figure 78: Website & Mobile User Interface Mockup

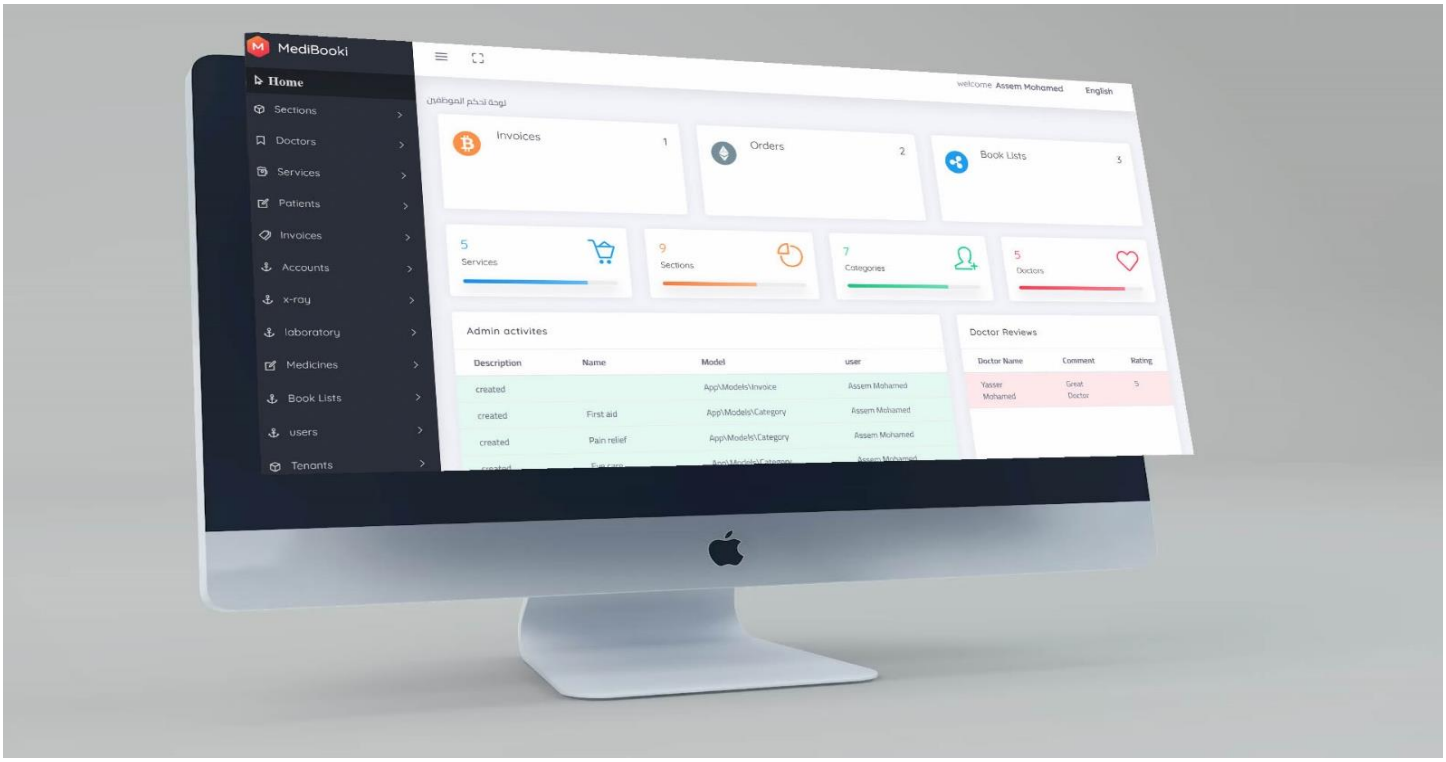


Figure 79: Dashboard Mockup

Chapter 5: Testing

5.1. Introduction: -

Testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. its Identifies bugs and issues in the development process so they're fixed prior to product launch. This approach ensures that only quality products are distributed to consumers, which in turn elevates customer satisfaction and trust.

5.2. Testing Objectives: -

The main objectives of testing in this project are to verify that the software functions as intended, validate its compliance with the specified requirements, and identify and resolve any defects or issues that may impact its usability or performance and generate high-quality test cases, perform effective tests, and issue correct and helpful problem reports. It is important also because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

5.3. Test Scope:

The testing scope includes all modules and interfaces of the software. This encompasses functional testing of individual components, integration testing to validate interactions between modules, and system-wide testing to ensure that the software functions as a cohesive unit.

5.4. Testing Approach:

The testing approach will consist of several types of testing. This includes functional testing to validate individual functionalities, usability testing to assess user-friendliness, performance testing to measure system response times, and security testing to identify and mitigate potential vulnerabilities.

5.5. Test Deliverables:

The testing deliverables will include a Test Plan document outlining the overall testing strategy, Test Cases specifying detailed steps and expected outcomes, a Test Execution Log to track test results, and a Test Summary Report summarizing the overall testing process and outcomes.

5.6. Test Schedule:

The testing activities will be conducted in parallel with the development process. such as completion of unit testing before integration testing begins, to ensure timely progress and effective coordination.

5.7. Types of Testing Used:

5.7.1. White box Testing

White box testing, alternatively referred to as clear box testing or structural testing, requires testers to possess knowledge about the internal structure, design, and implementation specifics of the software under examination. Testers have access to the source code and leverage this information to create test cases that specifically target particular paths, branches, or conditions within the code.

```
import unittest
import json
from unittest.mock import patch
from flask import Flask
from app import app, model_predict
 class TestApp(unittest.TestCase):
     def setUp(self):
        self.app = app.test_client()
     def test_invalid_request_format(self):
        response = self.app.post("/", data="Not a JSON")
        self.assertEqual(response.status_code, 400)
        self.assertEqual(json.loads(response.data), {"Error": "Request must be in JSON format"})
     def test_invalid_image_extension(self):
        response = self.app.post("/", json={"url": "https://example.com/image.txt"})
        self.assertEqual(response.status_code, 400)
        self.assertEqual(json.loads(response.data), {"Error": "URL has an unsupported image extension"})
     def test_url_404(self):
        with patch("requests.get") as mock_get:
            mock_get.return_value.status_code = 404
            response = self.app.post("/", json={"url": "https://example.com/image.jpg"})
            self.assertEqual(json.loads(response.data), {"Error": "The URL is not valid or could not be found"})
     @patch("app.model_predict")
    def test_positive_prediction(self, mock_predict):
        mock_predict.return_value = [[0.3, 0.7]]
        with patch("requests.get") as mock_get, patch("app.tf.keras.utils.get_file") as mock_get_file, \
                patch("app.tf.keras.preprocessing.image.load_img") as mock_load_img, \
                patch("tempfile.NamedTemporaryFile") as mock_tempfile, \
                patch("os.remove") as mock_remove:
            response = self.app.post("/", json={"url": "https://example.com/image.jpg"})
            self.assertEqual(json.loads(response.data), {"The prediction is": "Positive"})
     @patch("app.model_predict")
    def test_negative_prediction(self, mock_predict):
        mock_predict.return_value = [[0.8, 0.2]]
        with patch("requests.get") as mock_get, patch("app.tf.keras.utils.get_file") as mock_get_file, \
                patch("app.tf.keras.preprocessing.image.load_img") as mock_load_img, \
                patch("tempfile.NamedTemporaryFile") as mock_tempfile, \
                patch("os.remove") as mock_remove:
            response = self.app.post("/", json={"url": "https://example.com/image.jpg"})
            self.assertEqual(json.loads(response.data), {"The prediction is": "Negative"})
     def test_model_predict(self):
        # You can use a real image file here or create a mock image
        img_path = "path/to/test/image.jpg"
        result = model_predict(img_path, model)
        self.assertIsInstance(result, np.ndarray)
        self.assertEqual(result.shape, (1, 2))
 if __name__ == "__main__":
    unittest.main()
```

Figure 80: Flask API Testing

Appointments Integration Test (Angular)

```
import { TestBed } from '@angular/core/testing';
import { AppointmentsPatient } from 'src/app/core/interfaces/patients';
import { AppointmentsService } from './appointments.service';
import { HttpClientModule, HttpResponse } from '@angular/common/http';
import { HttpClientTestingModule,HttpTestingController } from '@angular/common/http/testing';
import { SharedModule } from 'src/app/layout/shared/shared.module';
describe('AppointmentsService', () => {
  let service: AppointmentsService;
  let httpMock: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports:[HttpClientModule,HttpClientTestingModule,SharedModule]
    });
    service = TestBed.inject(AppointmentsService);
    httpMock = TestBed.inject(HttpTestingController)
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
  it('should return the doctor with the given id', () => {
    service.getDoctorById('en', 1).subscribe(doctor => {
      expect(doctor.id).toBe(1);
    });
  });
  it('should throw an error if the id is not found', () => {
    service.getDoctorById('en', 0).subscribe(
      () => {},
      error => {
        expect(error).toBeTruthy();
      }
    );
  });
  it('should create an appointment for a patient', () => {
    const model: AppointmentsPatient = {
      patient_id: 0,
      doctor_id: 0,
      price: 0,
      date: "",
      time: ''
    };
    service.createAppointmentPatient(model).subscribe((res) => {
      console.log(res)
      expect(res).toBeTruthy();
      expect(res).toBe(200);
      expect(res).toEqual(model);
    });

  });

});
```

Figure 81: Angular Appointments System Integration Test

```php
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Artisan;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Storage;
use Tests\TestCase;
 class TenantTest extends TestCase
{
    public function testStoreWithValidData()
    {
        Storage::fake('media');
        $data = [
            'name' => 'Test Tenant',
            'domain' => 'test.com',
            'database' => 'test_database',
            'logo' => UploadedFile::fake()->image('logo.png'),
        ];
        $response = $this->post(route('tenants.store'), $data);
        $response->assertRedirect(route('tenants.index'));
        $response->assertSessionHas('success');
        $this->assertDatabaseHas('tenants', [
            'name' => 'Test Tenant',
            'domain' => 'test.com',
            'database' => 'test_database',
        ]);
        $this->assertDatabaseHas('media', [
            'model_type' => Tenant::class,
            'model_id' => 1,
            'collection_name' => 'logo',
        ]);
        Storage::disk('media')->assertExists(Tenant::first()->getFirstMediaPath('logo'));
        DB::statement('DROP DATABASE IF EXISTS test_database');
    }
     public function testStoreWithInvalidData()
    {
        $data = [
            'name' => '',
            'domain' => '',
            'database' => '',
            'logo' => UploadedFile::fake()->create('document.pdf', 100),
        ];
        $response = $this->post(route('tenants.store'), $data);
        $response->assertSessionHasErrors(['name', 'domain', 'database', 'logo']);
        $this->assertDatabaseMissing('tenants', [
            'name' => '',
            'domain' => '',
            'database' => '',
        ]);
    }
     public function testStoreWithExistingDatabase()
    {
        $existingTenant = Tenant::factory()->create();
        $data = [
            'name' => 'Test Tenant',
            'domain' => 'test.com',
            'database' => $existingTenant->database,
            'logo' => UploadedFile::fake()->image('logo.png'),
        ];
        $response = $this->post(route('tenants.store'), $data);
        $response->assertSessionHasErrors(['database']);
        $this->assertDatabaseMissing('tenants', [
            'name' => 'Test Tenant',
            'domain' => 'test.com',
            'database' => $existingTenant->database,
        ]);
    }
}
```

Figure 82: Laravel Tenant Testing

Login Integration Test (Angular)

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { AbstractControl, FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';
import { LoginComponent } from './login.component';
import { SharedModule } from 'src/app/layout/shared/shared.module';
import { AuthService } from '../../services/auth.service';
import { ToastrService } from 'ngx-toastr';
import { of, throwError } from 'rxjs';
import { AppModule } from 'src/app/app.module';

// Create a mock router object
class MockRouter {
  navigate = jasmine.createSpy('navigate');
}

// Create a mock DataService
class MockDataService {
  is_login = {
    next: jasmine.createSpy('next'),
  };
}

describe('LoginComponent', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>;
  let authService: jasmine.SpyObj<AuthService>;
  let toastrService: jasmine.SpyObj<ToastrService>;
  let router: MockRouter;
  let dataService: MockDataService;

  beforeEach(async () => {
    const authSpy = jasmine.createSpyObj('AuthService', ['login']);
    const toastrSpy = jasmine.createSpyObj('ToastrService', ['success', 'error']);

    await TestBed.configureTestingModule({
      declarations: [LoginComponent],
      providers: [
        FormBuilder,
        { provide: AuthService, useValue: authSpy },
        { provide: ToastrService, useValue: toastrSpy },
        { provide: MockRouter, useClass: MockRouter }, // Provide the mock router
        { provide: MockDataService, useClass: MockDataService }, // Provide the mock data service
      ],
      imports: [SharedModule, ReactiveFormsModule,AppModule],
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(LoginComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();

    authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
    toastrService = TestBed.inject(ToastrService) as jasmine.SpyObj<ToastrService>;
    router = TestBed.inject(MockRouter) as MockRouter; // Inject the mock router
    dataService = TestBed.inject(MockDataService) as MockDataService; // Inject the mock data service
  });

  it('should call login API and perform successful login', () => {
    // Mock loginForm value
    const email = 'test@example.com';
    const password = 'password123';
    component.loginForm.setValue({ email, Password: password });

    // Mock AuthService.login to return a successful response
    const loginResponse: any = {
      data: {
        token: 'test-token',
        id: 'test-patient-id',
      },
    };
    authService.login.and.returnValue(of(loginResponse));

    // Trigger the login function
    component.login();

    // Expectations
    expect(authService.login).toHaveBeenCalledWith({ email, password });
    expect(localStorage.setItem).toHaveBeenCalledWith('token', 'Bearer test-token');
    expect(localStorage.setItem).toHaveBeenCalledWith('patient_id', 'test-patient-id');
    expect(localStorage.removeItem).toHaveBeenCalledWith('reset_token');
    expect(localStorage.removeItem).toHaveBeenCalledWith('email_patient');
    expect(router.navigate).toHaveBeenCalledWith(['/home']);
    expect(toastrService.success).toHaveBeenCalledWith(
      !component.rtlDir ? 'Signed in successfully' : 'تم تسجيل الدخول بنجاح'
    );
    expect(toastrService.error).not.toHaveBeenCalled();
  });

  it('should handle login error', async () => {
    // Mock loginForm value
    const email = 'test@example.com';
    const password = 'password123';
    component.loginForm.setValue({ email, Password: password });

    // Create a mock error response
    const errorResponse = {
      error: {
        data: {
          error: 'Invalid credentials',
        },
      },
    };
    authService.login.and.returnValue(throwError(errorResponse));


    // Trigger the login function
    component.login();

    // Expectations
    expect(authService.login).toHaveBeenCalledWith({ email, password });
    expect(localStorage.setItem).not.toHaveBeenCalled;
    expect(localStorage.removeItem).not.toHaveBeenCalled;
    expect(router.navigate).not.toHaveBeenCalled;
    expect(toastrService.error).toHaveBeenCalledWith(
      !component.rtlDir ? `Email Or Password is incorrect` : `البريد الالكتروني او كلمة المرور خاطئة`,
      errorResponse.error.data.error
    );
  });


  afterEach(() => {
    fixture.destroy();
  });
});
```

Figure 83: Angular Login System Testing

5.7.2. Black box Testing

Black box testing, alternatively referred to as functional testing or input/output testing, centers on assessing the software's external behavior while remaining oblivious to its internal structure or implementation specifics. Testers treat the software as an opaque entity, scrutinizing inputs and outputs, and evaluate its functionality in alignment with the defined requirements.

- Test Case ID: 1

  Test Case Name: Login Functionality Verification

  Description: This test case verifies the login functionality of the software and verify the results.

  Preconditions: The software is running. A valid username and password are available.

  Test Steps:

  - Launch the software.
  - Enter a valid username.
  - Enter a valid password.
  - Click on the "Login" button.

  Expected Result: The software should successfully authenticate the user and display the home screen.

  Result: displayed successfully message and redirected to home page

- Test Case ID: 2

  Test Case Name: Login Functionality Verification

  Description: This test case verifies the login functionality of the software and verify the results.

  Preconditions: The software is running. An invalid username and password are available.

  Test Steps:

  - Launch the software.
  - Enter an invalid username.
  - Enter an invalid password.
  - Click on the "Login" button.

  Expected Result: The software should display error message and still in the login page.

Result: prompted ‘invalid email or password’ error message

- Test Case ID: 3

  Test Case Name: Register Functionality Verification

  Description: This test case verifies the register functionality of the software that add accounts to the system and verify the results.

  Preconditions: The software is running. Add the information of the patient like name, email, password …….. etc.

  Test Steps:

  - Launch the software.
  - Enter a valid username.
  - Enter a valid password.
  - Enter a valid gender.
  - Enter a valid phone number.
  - Enter a valid blood group.
  - Enter a valid birth of date
  - Click on the "Login" button.

  Expected Result: The software should successfully authenticate the user and display the login screen.

  Result: displayed successfully message and redirected to login page

- Test Case ID: 4

  Test Case Name: Add medicine to favorite

  Description: This test case adds specific medicine to favorites

  Preconditions: The software is running and login verified and had permission

  Test Steps:

  - Launch the software.
  - Login to the system.
  - Choose the specific medicine that you want to add
  - When you click on it, it will open medicine details then you should click on “add to favorite”

  Expected Result: The software should successfully add medicine to favorite and it will be displayed in favorites page

  Result: “This medicine added to favorite message”

- Test Case ID: 5

  Test Case Name: Store the order information and payment method

  Description: This test case store order information to continue the order successfully and add payment method

  Preconditions: The software is running, login verified, had permission and add the medicine you want to buy in cart

  Test Steps:

  - Launch the software.
  - Login to the system.
  - Choose the specific medicine that you want to add
  - When you click on it, it will open medicine details then you should click on “add to cart”
  - The medicine will be put in the cart
  - You should choose the quantity as you like
  - Click on “proceed to checkout”
  - Fill the information and click the payment method
  - You can choose “cash on delivery” or “pay online”

  Expected Result:

  1. If you click on “pay on delivery” the software should successfully do the function and display a message to confirm it and display “checkout success” and navigate to order page to track order
  2. If you click on “pay online” the software should successfully do the function and display a message to confirm it and navigate to pay mob web site to pay with credit card

  Result: “This order complete success message”

- Test Case ID: 6

  Test Case Name: talk to chatbot

  Description: This test case talk to chatbot to help the users to get their wants or needs

  Preconditions: The software is running

  Steps:

  - Launch the software.
  - Login to the system.
  - Click you the icon of chatbot
  - It will open and will ask you “how can I help you?”

- It will help you in 3 sections “Emergency, FAQs, Insurance “
- You should send any message to enable to choose the section you want

Expected Result:

1. If you click on “Emergency” the software should successfully display a message that have the number of emergency number
2. If you click on “FAQs” the software should successfully display a message that have the most frequently questions
3. If you click on “Insurance” the software should successfully display a message that have the companies that give us their services for insurance

Result: all functions complete successfully

5.8. Test Execution:

During test execution, we will follow the documented test cases and record the test results, including any defects or issues encountered. Defects will be logged in a Defect Tracking System, assigned to the development team for resolution, and retested after fixes are implemented.

Chapter 6: Conclusion & Future Work

6.1. Conclusion

From the beginning, our first and last mission and goal was to make it easier for all those involved in the health field, from patients to doctors and administrators, and all hospital departments. so we made three things that are divided into web app, mobile app, and dashboard, each one serving a specific category on the following approach:

- The Web App serves patients and doctors, so the doctor can apply to join the hospital, and the patient can book an examination and see all doctors and specialties, and he can also buy medicines, and he can add his medical insurance, and he can track all his transactions with the hospital, evaluate doctors, and the most important part is the presence of chatbot that helps the user on the site.
- As for the dashboard, it is divided into two parts. A part serves doctors by detecting patients and viewing all his bills and accounts. He can also add a service to his

personal account, and he can see patients' reservations. He can also see patient reviews. The second part is for administrators, and this is the main control panel in which he can add a branch to the hospital. Also, adding specialties and managing doctors' accounts, and he can accept or reject the doctor, manage patients, manage services, and there is a special part for accounting services and billing, and the largest and most important part is the detection of pneumonia, which is in the radiology section, and also the pharmacy and its medicines can be controlled, and patients' requests from the pharmacy can be controlled and ambulances managed .

- As for the mobile app, it serves patients by purchasing medicines and paying online.

6.2. Future Work

What we mentioned previously is what was actually done. As for our recommendation how to enhance the project if we are given the right resources, we want to connect smart watches to our system, and from them we track the user's health status such as heartbeat and such things, so we process this data and suggestion The patient should visit the specialist doctor in case of something abnormal. Also, we will not stop at detecting pneumonia, but we will use artificial intelligence to detect more diseases to make it easier for doctors and help them with modern technologies, for example Alzheimer's Disease Detection using Deep Learning

Pneumonia Detection using VGG16

Overview

Pneumonia is a common and potentially deadly lung infection that can be difficult to diagnose. In this graduation project, we developed two deep learning models for pneumonia detection in chest X-ray images: one based on a **custom CNN architecture** and the other using the **VGG16 architecture**. **Both** models were trained on a dataset of **5,863** chest X-ray images obtained from the Chest X-Ray Images (pneumonia) dataset available on Kaggle.

Requirements

Python 3.6 or higher

TensorFlow 2.0 or higher

Keras 2.0 or higher

Installation

Clone this repository to your local machine.

Install the required packages using pip install -r requirements.txt.

Download the dataset from Kaggle and extract the files to the data directory.

https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia.

Data

The dataset consists of 5,863 chest X-ray images, including 3,799 images with pneumonia and 1,157 normal images. We performed data augmentation by randomly rotating, zooming, and flipping the images to increase the size of the dataset and improve the model's ability to generalize.

Data Preprocessing

The dataset was preprocessed using the following steps*:*

Resizing: All images were resized to 224 x 224 pixels.

Data augmentation: The training dataset was augmented using various techniques, including random rotation, horizontal flipping, and zooming.

The augmentation was performed using the Keras ImageDataGenerator class with the following hyperparameters:

Rotation Range: 20

Width Shift Range: 0.2

Height Shift Range: 0.2

Horizontal Flip : True

Zoom Range: 0.2

Normalization: The pixel values of the images were normalized to the range [0, 1].

Model Architecture

Custom CNN:

The custom CNN architecture consists of 11 convolutional layers organized into 5 blocks. Each block consists of convolutional layers followed by max pooling and batch normalization layers. The output from the convolutional layers is flattened and passed through fully connected layers, and finally, a softmax output layer is applied to predict the pneumonia condition.

VGG16:

Architecture: The VGG16 architecture is a widely used convolutional neural network architecture for image classification. It consists of 13 convolutional layers organized in 5 blocks and 3 fully connected layers. We used the pre-trained VGG16 model and added a few additional layers on top for pneumonia detection. We added two fully connected layers and a SoftMax output layer to the base VGG16 model. The model was trained using the Adam optimizer and a categorical cross-entropy loss function.

Training

We split the dataset into training, validation, and test sets with a ratio of 70:15:15. The model was trained for 10 epochs with a batch size of 32. We used early stopping to prevent overfitting and reduce training time.

Results

Custom CNN Model

Our model achieved an **Accuracy score of 88.4%** on the 624 images test set, with a **Precision of 87.5%**, **Recall of 95%**, and **F1 score of 91%**. These results suggest that our model can effectively detect pneumonia in chest X-ray images. The precision and recall scores for each class are **shown in the confusion matrix**.

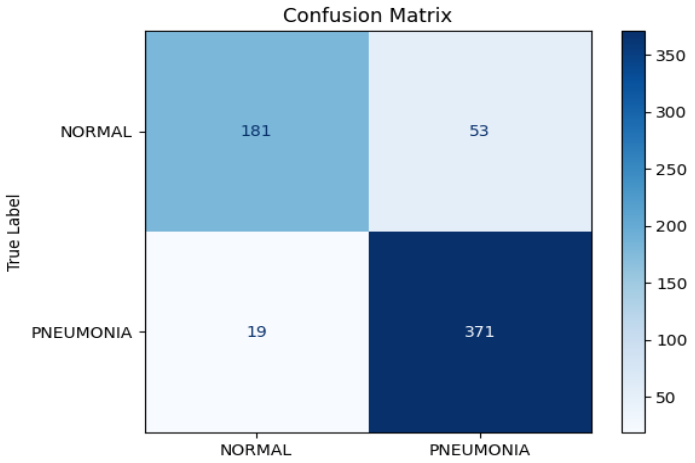


Figure 84: Confusion matrix for the custom CNN model

VGG16 Model

Our model achieved an **Accuracy score of 92%** on the 624 images test set, with a **Precision of 91%**, **Recall of 98%**, and **F1 score of 94%**. These results suggest that our model can effectively detect pneumonia in chest X-ray images. The precision and recall scores for each class are **shown in the confusion matrix**.

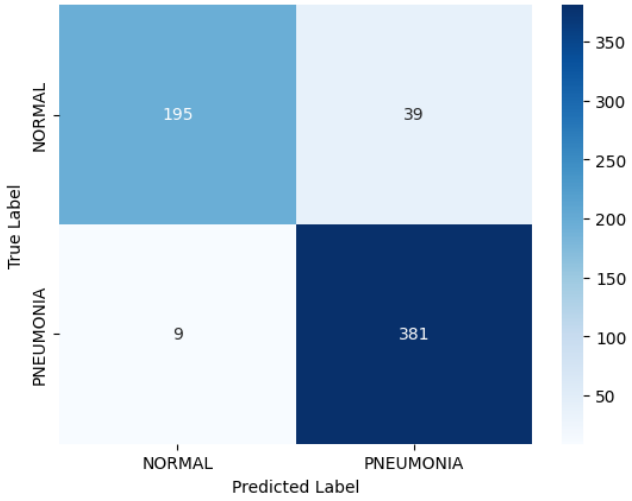


Figure 85: Confusion matrix for the VGG16 model

Comparison:

In this project, we successfully developed and evaluated two deep learning models, a custom CNN and VGG16, for pneumonia detection in chest X-ray images. **The VGG16 model achieved higher accuracy**, **precision**, **recall**, and **F1-score** compared to the custom **CNN model**. Both models show promise in automating the detection of pneumonia, which can aid in early diagnosis and treatment. Further advancements and improvements in the models and data collection can contribute to enhancing the accuracy and practicality of pneumonia diagnosis in real-world medical scenarios.

Limitations

Our model was trained on a relatively small dataset and may not generalize well to other types of chest X-ray images or medical imaging tasks.

The model's performance may be affected by factors such as the quality of the X-ray image or the skill of the radiologist who took the image.

Future Work

Train the model on a larger and more diverse dataset to improve its generalizability.

Investigate the use of transfer learning and other deep learning techniques to further improve the model's performance.

Incorporate additional clinical and demographic data into the model to improve its accuracy and relevance to real-world medical settings.

Conclusion

In this project, we developed a deep learning model based on the VGG16 architecture to automatically detect pneumonia in chest X-ray images. Our model achieved a high level of accuracy and provides a promising approach for automated pneumonia diagnosis.