VIETNAM INTERNATIONAL UNIVERSITY – HO CHI MINH CITY

INTERNAITONAL UNIVERSITY

# WEB APPLICATION DEVELOPMENT PROJECT PHARMACY MANAGEMENT - MEDIMASTER


By

Huỳnh Ngọc Anh Thư – Student ID: ITCSIU21034

Trần Thế Phong – Student ID: ITCSIU21215

Phạm Nguyễn Đăng Khôi – Student ID: ITCSIU21196

Advisor: Dr. Nguyen Van Sinh

A report submitted to the School of Computer Science and Engineering in partial fulfillment of the requirements for the Final Project

in Web Application Development course - Fall 2015

Ho Chi Minh city, Vietnam. 2024

Table of Contents

Table of Figure

# 1. INTRODUCTION

## 1.1 ABOUT US

### 1.1.1 THE PRODUCT INFORMATION

In Vietnam, the culture of purchasing medicine from local pharmacies is deeply ingrained, as people often prefer visiting pharmacies over hospitals for non-critical health concerns. This trend has made retail pharmacies like Long Châu and others integral to daily healthcare.

To streamline operations and improve management for pharmacy owners and staff, we have developed a **Pharmacist Management System** tailored to the unique needs of Vietnamese pharmacies. The system is exclusively used by pharmacists and administrators, providing tools for inventory management, sales tracking, customer invoicing, and efficient medicine restocking.

Our solution simplifies day-to-day tasks, enhances productivity, and ensures accurate record-keeping, enabling pharmacies to serve their customers more effectively while maintaining compliance with regulatory standards. With an intuitive web-based interface, the system supports seamless operations on both desktop and mobile platforms, meeting the growing demands of modern retail pharmacy management.

### 1.1.2 WORK BREAKDOWN STRUCTURE

The structure of our project can be expressed in the figure:

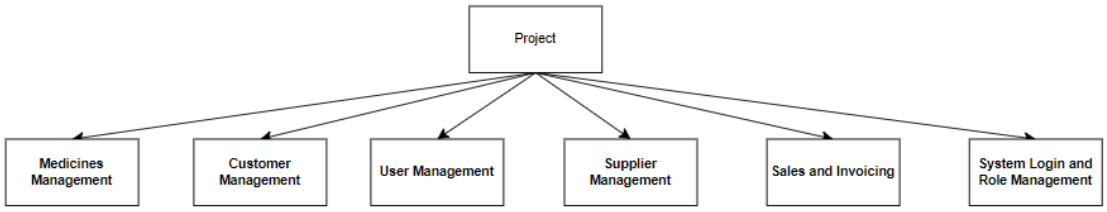



*Figure 1*

In each branch of this tree, we also have subtrees which describes the tasks needed to be accomplished of each team member.

The tasks for the **Medicines Management** is described in the Figure

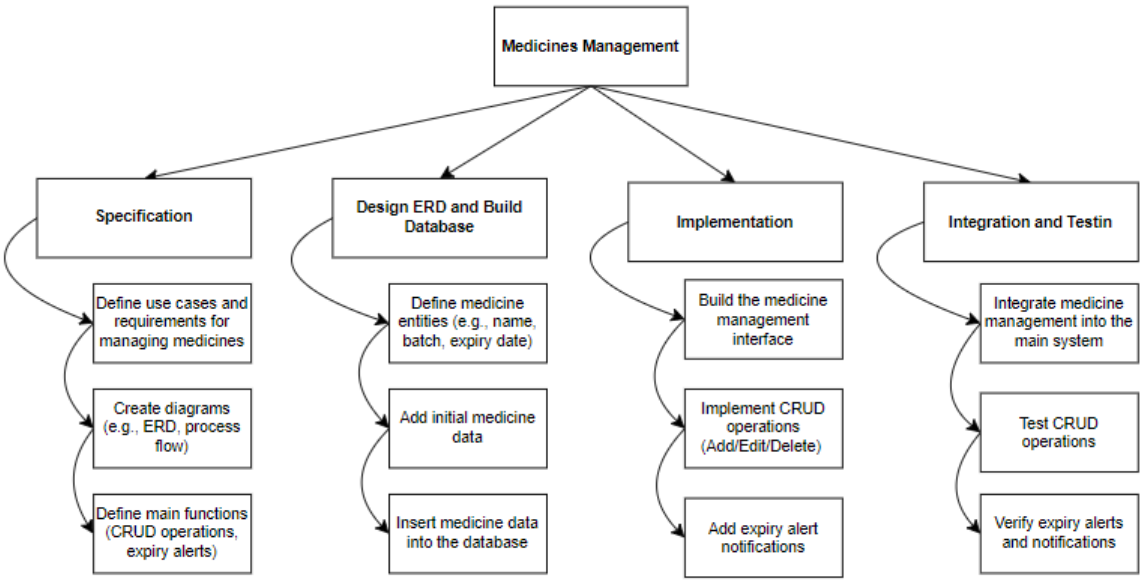



*Figure 2*

The tasks for the **Customer Management** is described in the Figure

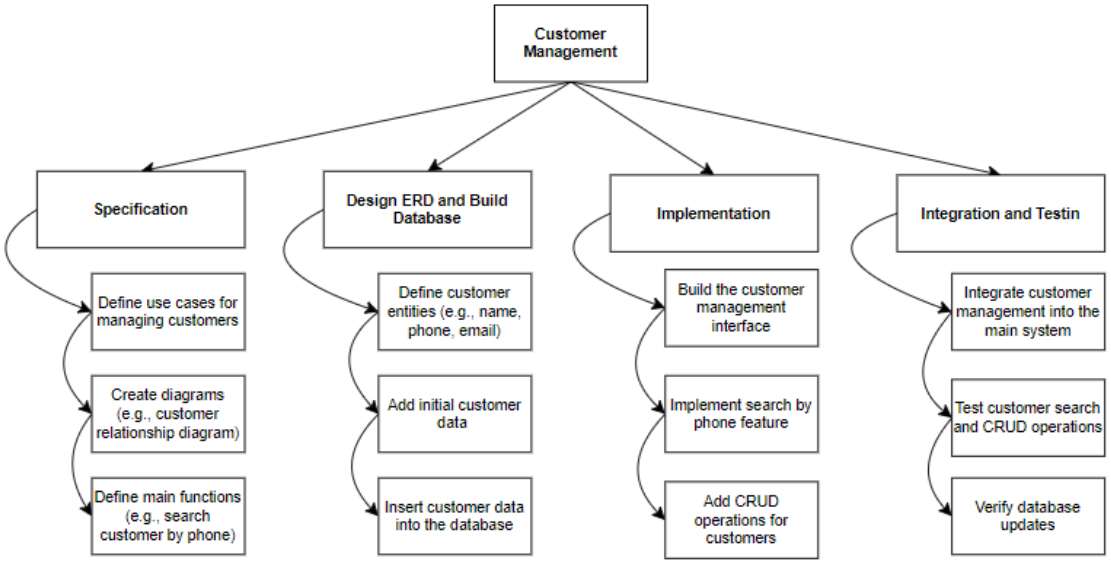



*Figure 3*

The tasks for the **User Management** is described in the Figure

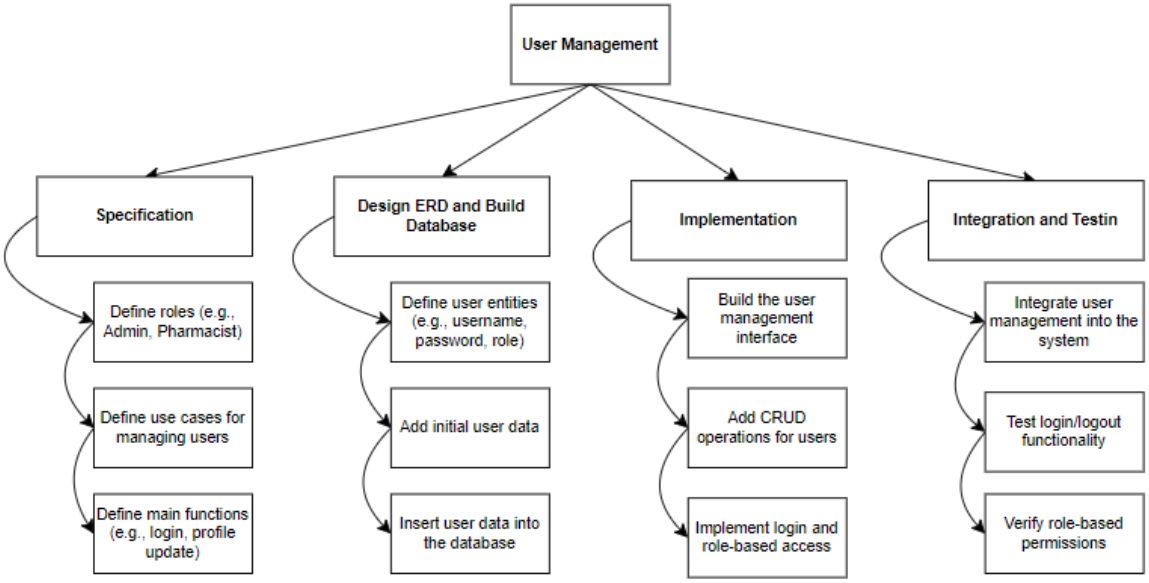



*Figure 4*

The tasks for the **Supplier Management** is described in the Figure

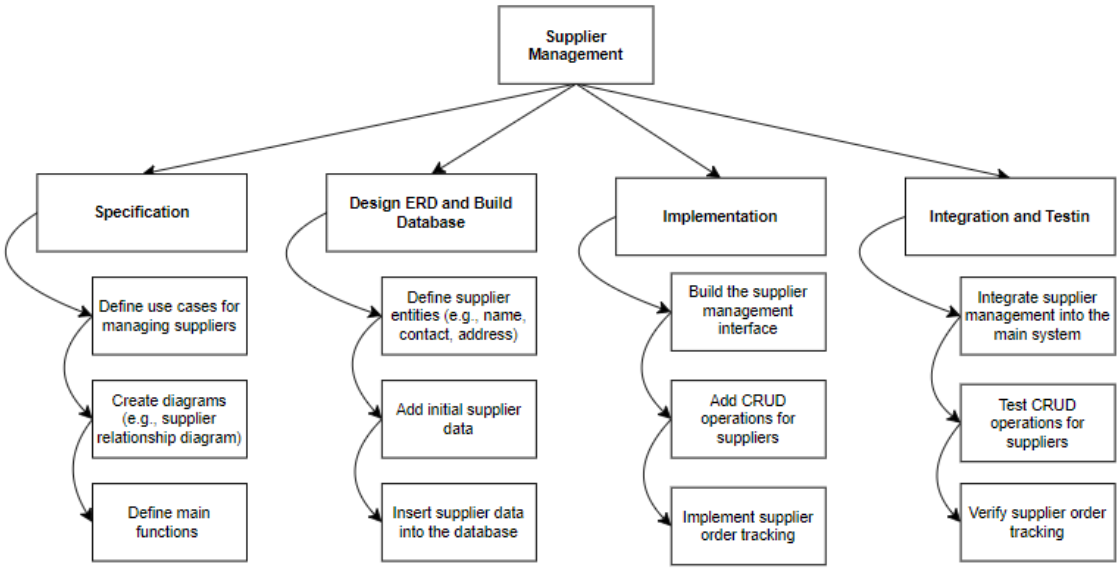



*Figure 5*

The tasks for the **Sales and Inovoicing** is described in the Figure

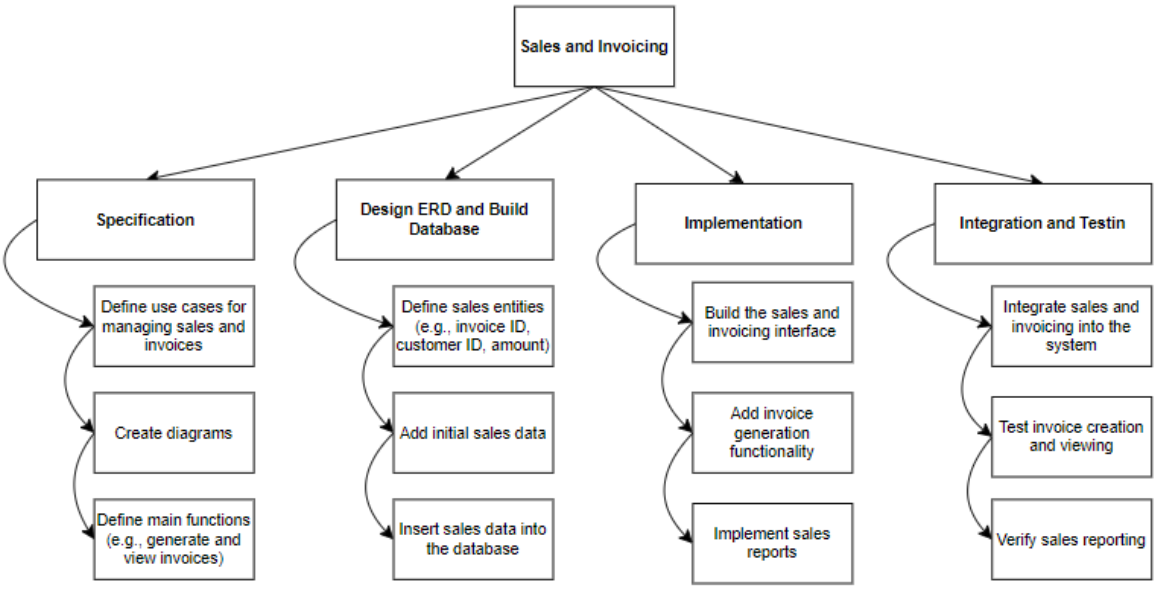



*Figure 6*

The tasks for the **System Login and Role Management** is described in the Figure

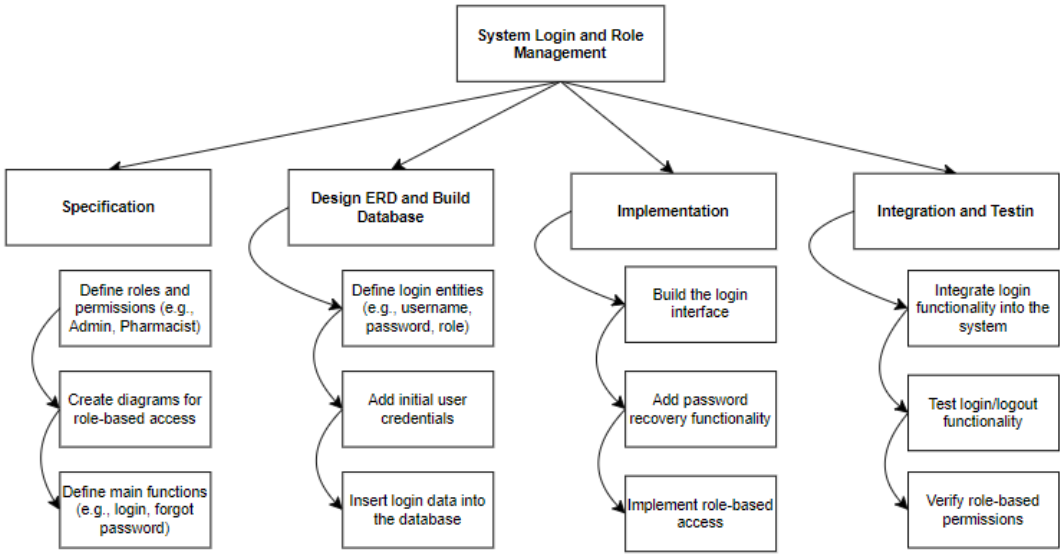



*Figure 7*

## 1.2 DEVELOPMENT PROCESS

For this project, our team adopted a **phased development approach**, ensuring systematic progression through key components before integration. This process emphasizes building a strong foundation, starting with the database, followed by the backend, and finally the frontend.

**Phase 1: Planning and Tasks Allocation**

| Full name | ID | Tasks |
|---|---|---|
| Huynh Ngoc Anh Thu | ITCSIU21034 | 1. Front-end<br>2. Back-end<br>3. Use case diagram<br>4. Report review<br>5. Deployment<br>6. Domain management<br>7. Presentation |
| Pham Nguyen Dang Khoi | ITCSIU21196 | 1. Back-end **(invoiceController, invoiceItemController, supplierController)**<br>2. Front-end **(addInvoice, editInvoice, addSupplierForm, editSupplierForm)**<br>3. Diagrams<br>4. Report writing<br>5. Slide design<br>6. Presentation |
| Tran The Phong | ITCSIU21215 | 1. Back-end **(invoiceController, invoiceItemController, supplierController)**<br>2. Front-end **(addInvoice, editInvoice, addSupplierForm, editSupplierForm)**<br>3. Diagrams<br>4. Report writing<br>5. Slide design<br>6. Presentation |

**Phase 2: Database Development**

The database is the backbone of our system. In this phase:

- The schema was designed, and tables were created to accommodate system requirements.
- SQL scripts for data storage, retrieval, and manipulation were finalized.

**Phase 3: Backend Development**

The backend was developed to provide robust APIs and handle business logic. Key tasks in this phase included:

- Designing and implementing RESTful APIs.
- Connecting the backend to the database and ensuring secure operations.
- Writing middleware for authentication, authorization, and data validation.

**Phase 4: Frontend Development**

Once the backend was stable, we focused on the user interface to ensure a seamless user experience. Key tasks in this phase included:

- Developing responsive pages with React.
- Integrating APIs for real-time data exchange.
- Conducting usability testing to optimize the interface

**Phase 5: Deployment**

The deployment phase focused on transitioning the system from development to a live environment, ensuring it was fully operational and accessible to end users. This phase followed a systematic workflow to maintain a streamlined and error-free process:

- Pre-development Preparation:
  - Verifing all the system components were stable.
  - Performing final testings on local environtment.
- Application Development:
  - Deploying the system components sequentially (database → backend → frontend).
  - Configuring the application to align with the system architecture.
- Post-development Validation:
  - Perfoming end-to-end testing in live environment.
  - Validating and recording all the core workflows, including bugs and errors.
  - Fixing and retesting.

- Launch and Monitoring:
    - Launching the system officially, making the website accessible via a certified domain.
    - Monitoring the application post-launched, addressing any issues and optimizing the performance.

This process allowed us to work sequentially while maintaining flexibility for adjustments during development. Each phase concluded with testing to ensure reliability before progressing to the next stage.

By following this process, we ensured that the system was built on a solid foundation, with each component properly integrated for optimal performance.

## 1.3 DEVELOPMENT ENVIRONMENT

This project is a web-based pharmacy management system, developed using modern web technologies and following the Model-View-Controller (MVC) architecture. The system comprises both backend and frontend components, designed to provide an efficient and dynamic user experience.

### 1.3.1 Technologies and Tools:

#### 1.3.1.1 Frontend:

- **HTML**: To structure the webpages and provide the basic layout.
- **CSS**: To style and enhance the visual appearance of the webpages.
- **JavaScript** (React)**:** For building an interactive and dynamic user interface.
- **React Router**: For seamless navigation across different pages in the application.
- **Ant Design (Antd)**: For pre-built UI components and design consistency.
- **Moment**.js: For handling date and time-related functionalities.
- **ReactJS**: For building the user interface and managing the component-based architecture.
- **Axios**: For making HTTP requests to communicate with the backend.

1.3.1.2 Backend:

- **Node.js**: To handle server-side logic and build scalable backend services.
- **Express.js**: As the web framework for managing API endpoints and middleware.
- **JavaScript:** For developing.
- **MySQL**: For managing the system's database, including user and product data.
- **Sequelize**: An ORM (Object Relational Mapping) tool for database operations.
- **JWT (JSON Web Tokens)**: For secure user authentication and authorization.
- **BCrypt**: To securely hash and manage passwords.
- **Body**-Parser: To parse incoming request bodies.
- **Cors**: For handling Cross-Origin Resource Sharing to allow communication between the frontend and backend (used for deployment preparation).
- **Dotenv**: For managing environment variables securely.

1.3.1.3 Database:

- **MySQL**: For storing and monitoring the data of the system.

1.3.1.4 Development:

- **Database**: Deployed on Railway.
- **Backend**: Deployed and hosted on Render.
- **Frontend**: Deployed and hosted on Netlify.
- **Domain**: Certified domain was bought from tenten.vn.

1.3.1.5 Tools:

To streamline team collaboration and maintain project quality:

- **GitHub**: To manage version control and facilitate code review.
- **Google Drive**: For sharing documents and resources among team members.
- **Microsoft Visio and Draw.io**: For creating UML diagrams, including Entity Relationship Diagrams (ERDs) and Use Case diagrams.
- **Postman**: For tracking progress and managing tasks effectively.
- **MySQL Workbench**: For storing and monitoring the database.

- **WebStorm, Visual Studio Code**: IDE for coding processes.

### 1.3.2 Project Architecture

The system follows a **Client-Server Architecture** that incorporates elements of the MVC design pattern, as illustrated in the system's design:

- **Model:**
  - Represents the data layer, handling the structure and interaction with the database.
  - This is implemented using **Sequelize ORM** in the models directory of the backend. Each Sequelize model corresponds to a table in the database, managing the schema and providing logic for accessing or modifying the data.
- **View:**
  - Represents the user interface layer, built with **React**.
  - It provides an interactive and seamless experience for users, including dynamic UI updates based on interactions and data fetched from the backend.
  - The frontend communicates with the backend through RESTful APIs.
- **Controller:**
  - Manages the application logic and serves as the bridge between the **Model** and **View** layers.
  - Controllers are implemented in the backend within the routes directory, where logic for handling HTTP requests and responses is defined.
  - For example, the categoryController processes API calls related to categories by interacting with the Sequelize models and sending appropriate responses to the frontend.

In this architecture, the **Frontend (React)** and **Backend (Node.js + Express)** work together to deliver a responsive and dynamic user experience:

1. The **Frontend** (View) sends API requests to the backend to perform actions such as retrieving data, creating new entries, or updating existing information.
2. The **Backend** processes these requests using **Controllers**, which:
   - Interact with **Models** to execute the necessary database operations.
   - Return data in JSON format to the frontend.
3. The **Frontend** updates the user interface dynamically based on the received data, ensuring a smooth user experience.

This architecture ensures a clear separation of concerns, enabling scalability, maintainability, and ease of development for both frontend and backend components.

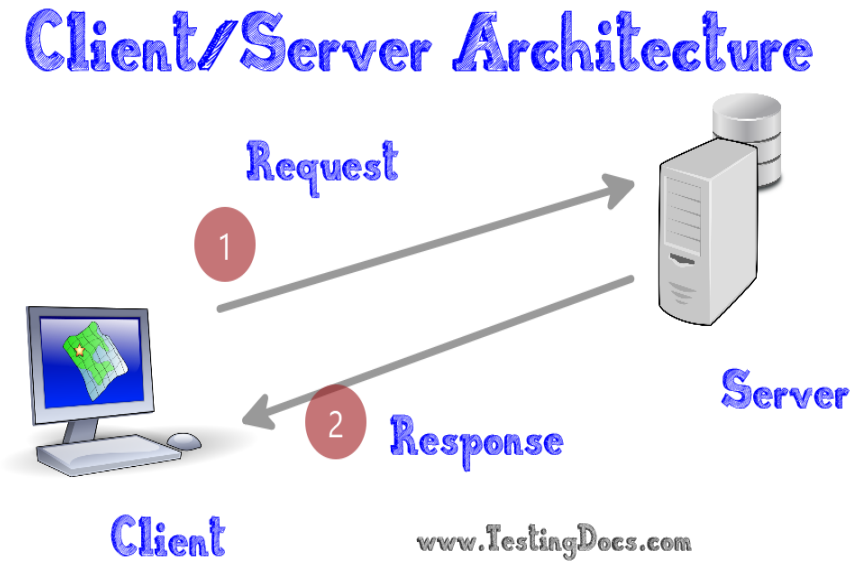



*Figure 8*

### 1.3.3 Collaboration Tools

To streamline team collaboration and maintain project quality:

- **GitHub**: To manage version control and facilitate code review.
- **Google Drive**: For sharing documents and resources among team members.
- **Microsoft Visio and Draw.io**: For creating UML diagrams, including Entity Relationship Diagrams (ERDs) and Use Case diagrams.
- **Postman**: For tracking progress and managing tasks effectively.

### 1.3.4 Database Design and Documentation

The database schema was designed using tools like MySQL Workbench. All database operations are centralized in the **pharmacy_management.sql** file.

### 1.3.5 Development and Testing

- Unit testing is implemented using **Postman** in the backend.
- The frontend is tested for responsiveness and cross-browser compatibility.

By combining these technologies and tools, the project achieves an efficient, robust, and user-friendly system tailored to the needs of a modern pharmacy management application.

# 2 REQUIREMENT ANALYSIS AND DESIGN

## 2.1 Reqirement Analysis

### 2.1.1 USE CASE DIAGRAM

#### 2.1.1.1 Sumary Use Case Diagram:

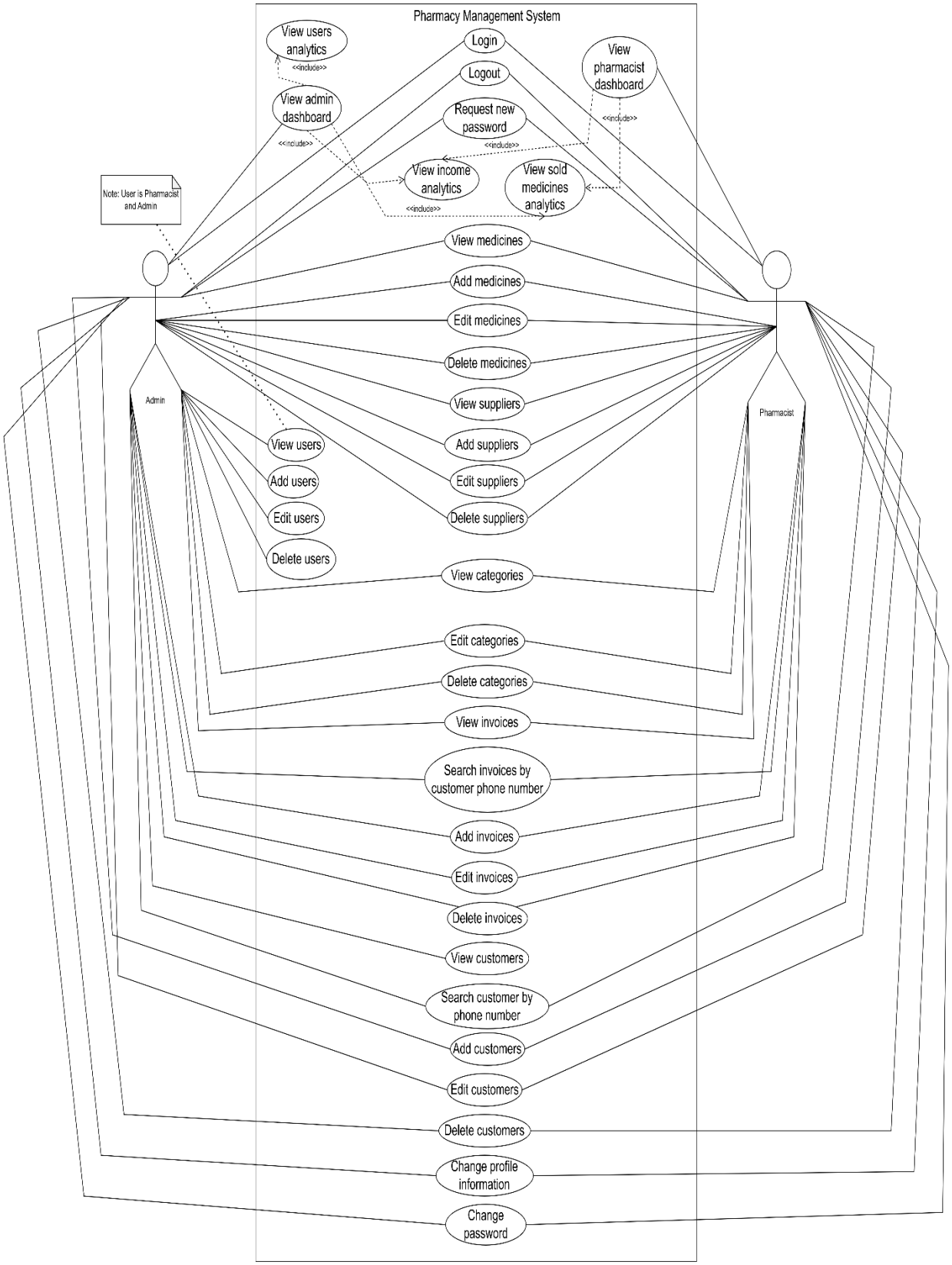

Pharmacy Management System
View users analytics
<<include>>
View admin dashboard
<<include>>
Login
Logout
Request new password
<<include>>
View pharmacist dashboard
<<include>>
View income analytics
View sold medicines analytics
<<include>>
Note: User is Pharmacist and Admin
View medicines
Add medicines
Edit medicines
Delete medicines
View suppliers
Add suppliers
Edit suppliers
Delete suppliers
Admin
Pharmacist
View users
Add users
Edit users
Delete users
View categories
Edit categories
Delete categories
View invoices
Search invoices by customer phone number
Add invoices
Edit invoices
Delete invoices
View customers
Search customer by phone number
Add customers
Edit customers
Delete customers
Change profile information
Change password

*Figure 9*

### 2.1.1.2 Admin:

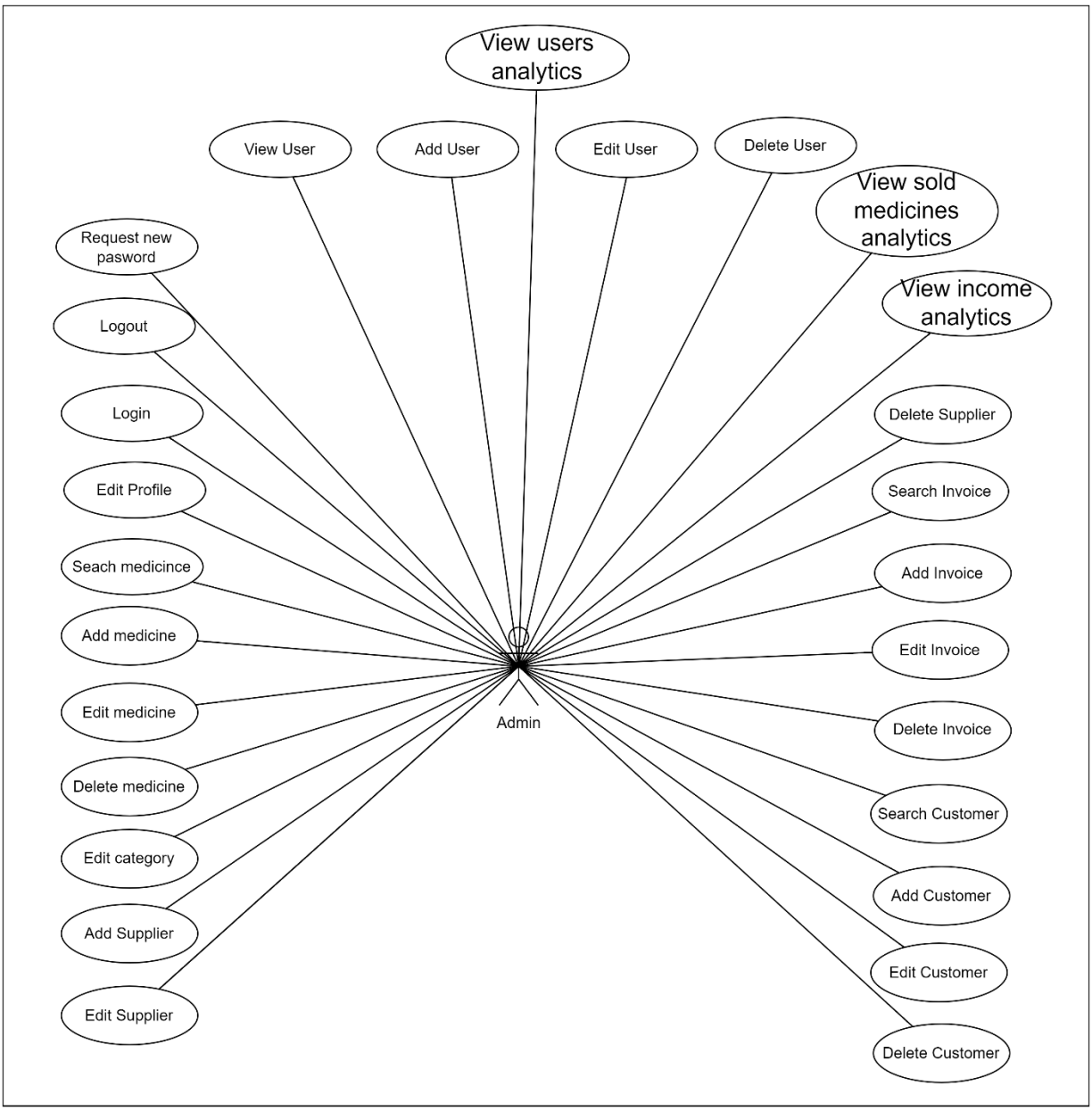



*Figure 10*

2.1.1.3 Pharmacist:

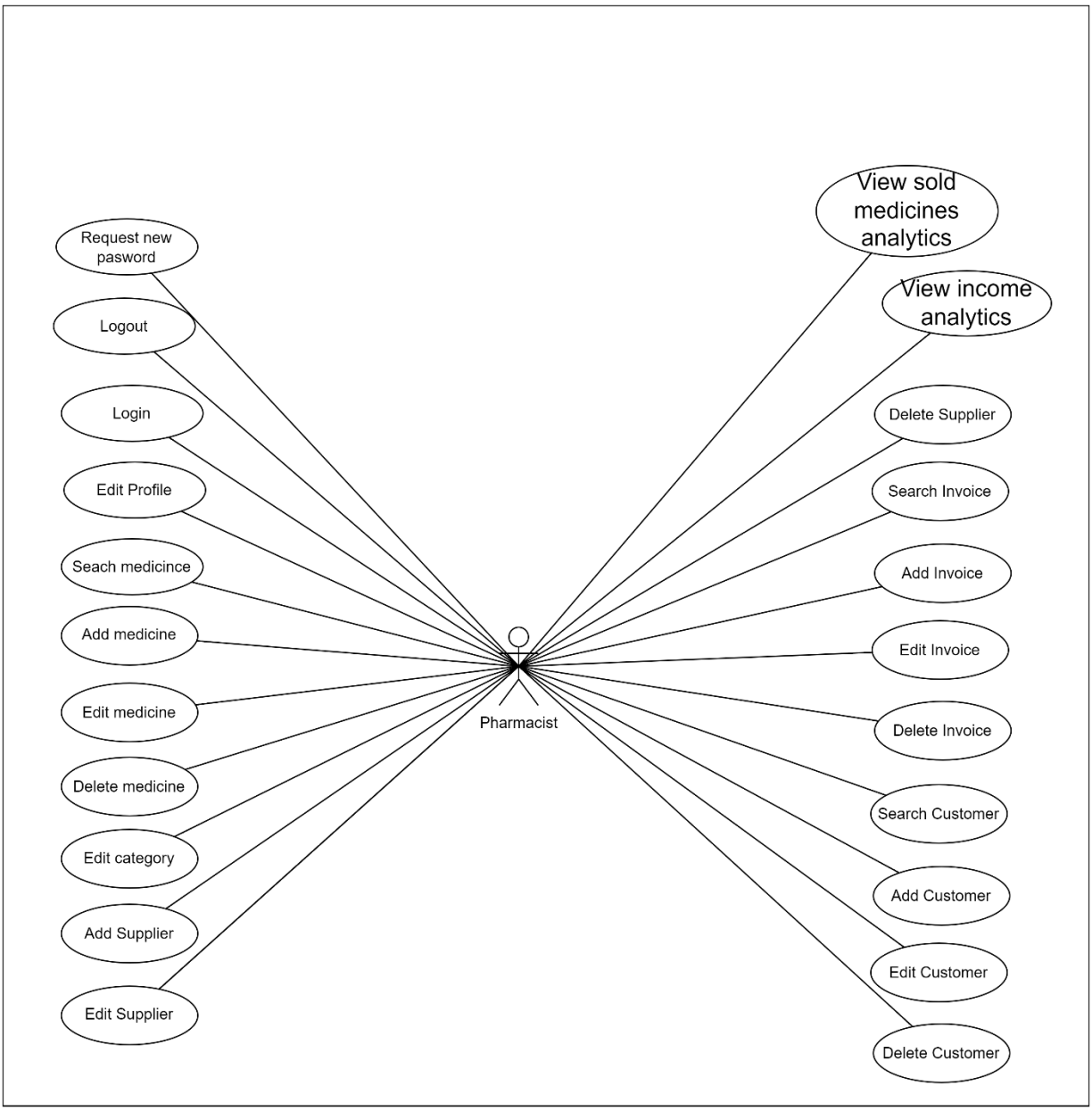



*Figure 11*

2.1.1.4 Use Case 1:

**Name: Add user account**

**Identifier** *UC1*

**Inputs:**

1. User name
2. Password
3. Personal information from user

**Outputs:**

1. Notification on top: “User added successfully” [If success]
2. Notification on top: “Failed to add user” [If fail]

**Basic Course**

| Actor: Admin | System |
|---|---|
| 1. Access Users page | 1.1. Display the Users page |
| 2. Click Add User | 2.1. Display the input form |
| 3. Input information: username, password, name, email, choose role. | |
| 4. Click OK | 4.1. Check the input information<br>4.2. If success, show notification “User added successfully” and display account in Users page<br>4.3 Else show notification “Failed to add user”<br>4.4 Update the database. |

**Precondition**

1. The username must not be duplicated

**Post condition**

1. None

**User story:**

As an Admin of this web page, I want to create new user account that has 1 of 2 role: Admin or Pharmacist to help me manage this web page.

2.1.1.5 Use Case 2:

**Name: Edit user account**

**Identifier** *UC2*

**Inputs:**

1. Name
2. Email

**Outputs:**

1. Notification on top: “User updated successfully” [If success]
2. Notification on top: “Failed to update user” [If fail]

**Basic Course**

| Actor: Admin | System |
|---|---|
| 1. Access Users page | 1.1. Display the Users page |
| 2. Click Edit | 2.1. Display the input form |
| 3. Input information: name, email, choose role. | |

| | |
|---|---|
| 4. Click OK | 4.1. Check the input information<br>4.2. If success, show notification "User updated successfully" and display updated account in Users page<br>4.3 Else show notification "Failed to update user"<br>4.4 Update the database. |

**Precondition**

1. The email must not be duplicated

**Post condition**

1. None

**User story:**

As an Admin of this web page, I want to update my account or other account that needs to be changed for any reason (satisfaction, security, …).

2.1.1.6 Use Case 3:

**Name: Delete user account**

**Identifier** *UC3*

**Inputs:**

1. None

**Outputs:**

1. Notification on top: "User deleted successfully" [If success]
2. The Users page [If fail]

**Basic Course**

| Actor: Admin | System |
|---|---|
| 1. Access Users page | 1.1. Display the Users page |
| 2. Click Delete | 2.1. The system is loading |
| 3. Click OK | 3.1. It will delete user account and show notification “User deleted successfully” .<br>3.2 Update the database. |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin of this web page, I want to delete any user account that is not needed anymore.

2.1.1.7 Use Case 4:

**Name: Register customer account**

**Identifier** *UC4*

**Inputs:**

1. Name
2. Phone
3. Email

**Outputs:**

1. Notification on top: “Customer added successfully” [If success]

2. Notification on top: “Failed to add customer” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Customers page | 1.1. Display the Customers page |
| 2. Click Add Customer | 2.1. Display the input form |
| 3. Input information: name, phone, email (not compulsory) | |
| 4. Click OK | 4.1. Check the input information<br>4.2. If success, show notification “Customer added successfully” and display created account in Customers page<br>4.3 Else show notification “Failed to add customer”<br>4.4 Update the database. |

**Precondition**

1. The phone number must not be duplicated

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to create new accounts for my customers so that I can easily contact them with their information when needed.

2.1.1.8 Use Case 5:

**Name: Update customer account**

**Identifier** *UC5*

**Inputs:**

1. Name
2. Phone
3. Email

**Outputs:**

1. Notification on top: “Customer updated successfully” [If success]
2. Notification on top: “Failed to update customer” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Customers page | 1.1. Display the Customers page |
| 2. Click Edit | 2.1. Display the input form |
| 3. Input information: name, phone, email (not compulsory) | |
| 4. Click OK | 4.1. Check the input information<br>4.2. If success, show notification “Customer updated successfully” and display created account in Customers page<br>4.3 Else show notification “Failed to add customer”.<br>4.4 Update the database. |

**Precondition**

1. The phone number must not be duplicated

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, my customers want to change their information so I want to have an edit function to update their information as desired

2.1.1.9 Use Case 6:

**Name: Delete customer account**

**Identifier** *UC6*

**Inputs:**

1. None

**Outputs:**

1. Notification on top: “Customer deleted successfully” [If success]
2. The Customer page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Users page | 1.1. Display the Users page |
| 2. Click Delete | 2.1. The system is loading |
| 3. Click OK | 3.1. It will delete user account and show notification “Customer deleted successfully” .<br>3.2 Update the database. |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to delete any customer account that is not needed anymore or customers actively want to delete it

2.1.1.10 Use Case 7:

**Name: Add desired medicine to database**

**Identifier** *UC7*

**Inputs:**

1. Image
2. Name
3. Category
4. Description
5. Price
6. Quantity
7. Supplier
8. Location
9. Expiration Date

**Outputs:**

1. Notification on top: “Medicine added successfully” [If success]
2. The Medicines page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|

| | |
|---|---|
| 1. Access Medicines page | 1.1. Display the Medicines page |
| 2. Click Add Medicine | 2.1. Display the input form |
| 3. Input information: image, name, category, description, price, quantity, supplier, location, expiration date. | |
| 4. Click Add | 4.1. Check the input information<br>4.2. If success, show notification "Medicine added successfully" and display created medicine in Medicines page<br>4.3 Else close form and stay in Medicines<br>4.4 Update the database. |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to added new medicine information that is currently available in stock so customers can buy.

2.1.1.11 Use Case 8:

**Name: Add desired supplier to database**

**Identifier** *UC8*

**Inputs:**

1. Supplier Name
2. Contact
3. Address

**Outputs:**

1. Notification on top: “Supplier added successfully” [If success]
2. Notification on top: “Failed to add supplier” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Suppliers page | 1.1. Display the Suppliers page |
| 2. Click Add Supplier | 2.1. Display the input form |
| 3. Input information: supplier name, contact, address. | |
| 4. Click Add | 4.1. Check the input information<br>4.2. If success, show notification “Supplier added successfully” and display created supplier in Suppliers page<br>4.3 Else show notification “Failed to add supplier”<br>4.4 Update the database. |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin of this web page, I want to add new suppliers to database whenever there are new suppliers that provide medicines

2.1.1.12 Use Case 9:

**Name: Add desired invoice to database**

**Identifier** *UC9*

**Inputs:**

1. Customer Phone
2. Customer Name
3. Type
4. Medicine
5. Quantity

**Outputs:**

1. Notification on top: “Invoice created successfully” [If success]
2. The Sales & Invoices page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Sales & Invoices page | 1.1. Display the Sales & Invoices page |
| 2. Click Add Invoice | 2.1. Display the input form |
| 3. Input information: customer phone, name (auto-generated), type, medicine, quantity, then click Add Item | 3.1. System will add each item to that customer’s invoice |

| | |
|---|---|
| 4. Click Save | 4.1. Check the input information<br>4.2. If success, show notification "Invoice created successfully" and display created invoice in Sales & Invoices page<br>4.3 Else return to Sales & Invoices page<br>4.4 Update the database. |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to add new invoice to database whenever there are customers who sale or purchase from our store so that we can keep track of the information.

2.1.1.13 Use Case 10:

**Name: Update medicine information**

**Identifier** *UC10*

**Inputs:**

1. Name
2. Category
3. Description
4. Price
5. Quantity
6. Supplier

7. Location
8. Expiration Date

**Outputs:**

1. Notification on top: “Medicine updated successfully” [If success]
2. Notification on top: “Failed to update medicine” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Medicines page | 1.1. Display the Medicines page |
| 2. Click Edit | 2.1. Display the input form |
| 3. Input information: name, category, description, price, quantity, supplier, location, expiration date | |
| 4. Click Save | 4.1. Check the input information<br>4.2. If success, show notification “Medicine updated successfully” and display created account in Medicines page<br>4.3 Else show notification “Failed to add medicine”.<br>4.4 Update the database. |

**Precondition**

1. None

**Post condition**

1. Item updated in the medicine list

2. Database updated

**User story:**

As an Admin/Pharmacist of this web page, I want to change the information of medicine when current information is not correct.

2.1.1.14 Use Case 11:

**Name: Update category information**

**Identifier** *UC11*

**Inputs:**

1. Category Name
2. Description

**Outputs:**

1. Notification on top: “Category updated successfully” [If success]
2. The Categories page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Categories page | 1.1. Display the Categories page |
| 2. Click Edit | 2.1. Display the input form |
| 3. Input information: category name, description | |
| 4. Click Save | 4.1. Check the input information<br>4.2. If success, show notification “Category updated successfully” and display updated category in Categories page |

| | |
|---|---|
| | 4.3 Else return to Categories page<br>4.4 Update the database. |

**Precondition**

1. None

**Post condition**

1. Item updated in the category list

2. Database updated

**User story:**

As an Admin/Pharmacist of this web page, I want to update category information whenever there are new changes with the name or description of these current categories in the world.

2.1.1.15 Use Case 12:

**Name: Update supplier information**

**Identifier** *UC12*

**Inputs:**

1. Supplier Name
2. Contact
3. Address

**Outputs:**

1. Notification on top: “Supplier updated successfully” [If success]
2. The Suppliers page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Categories page | 1.1. Display the Categories page |
| 2. Click Edit | 2.1. Display the input form |
| 3. Input information: supplier name, contact, address | |
| 4. Click Save | 4.1. Check the input information<br>4.2. If success, show notification "Supplier updated successfully" and display updated supplier in Suppliers page<br>4.3 Else return to Suppliers page<br>4.4 Update the database. |

**Precondition**

1. None

**Post condition**

1. Item updated in the supplier list

2. Database updated

**User story:**

As an Admin/Pharmacist of this web page, I want to update supplier information whenever suppliers have new changes with their information or they actively ask for updating

2.1.1.16 Use Case 13:

**Name: Update invoice information**

**Identifier** *UC12*

**Inputs:**

1. Customer Phone
2. Type
3. Medicine
4. Quantity

**Outputs:**

1. Notification on top: “Invoice updated successfully” [If success]
2. The Invoice page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Sales & Invoices page | 1.1. Display the Sales & Invoices page |
| 2. Click Edit | 2.1. Display the input form |
| 3. Input information: customer phone, type, medicine, quantity | |
| 4. Click Save | 4.1. Check the input information<br>4.2. If success, show notification “Invoice updated successfully” and display updated invoice in Sales & Invoices page<br>4.3 Else return to Sales & Invoices page<br>4.4 Update the database. |

**Precondition**

1. If customer want to changes their phone or name information, they must update in the Customers page, then insert phone number so the name will be automatically generated

**Post condition**

1. Item updated in the invoice list

2. Database updated

**User story:**

As an Admin/Pharmacist of this web page, this page is useful whenever customers want to change their information or Admin/Pharmacist want to correct the information

2.1.1.17 Use Case 14:

**Name: Delete medicine/supplier/invoice object**

**Identifier** *UC14*

**Inputs:**

1. None

**Outputs:**

1. Display message that “Medicine/Supplier/Invoice deleted successfully”

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Click Delete button in Medicines page | 1.1. Display the success message<br>1.2. The system removed item and updated the list<br>1.3. Database updated |
| 2. Click Delete button in Supplier page | 2.1. Display the success message<br>2.2. The system removed item and updated the list<br>2.3. Database updated |

| | |
|---|---|
| 3. Click Delete button in Sales & Invoices page | 3.1. Display the success message<br>3.2. The system removed item and updated the list<br>3.3. Database updated |

**Precondition**

1. None

**Post condition**

1. Database updated

**User story:**

As an Admin/Pharmacist of this web page, I want to delete any medicine/supplier/invoice object that is not needed anymore.

2.1.1.18 Use Case 15:

**Name: Search medicine**

**Identifier** *UC15*

**Inputs:**

1. Name of medicine

**Outputs:**

1. Notification on top: “Found ‘number’ result(s) for “name”” [If success]
2. Notification on top: “No medicines found for “name”””” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Medicines page | 1.1. Display Medicines page |

| | |
|---|---|
| 2. Click “Search medicines…” box and insert name of medicine | |
| 3. Press Enter button or click Search icon | 3.1. Check the available medicine<br>3.2. If success, show a list of related medicine and a message<br>3.3. If not found, show fail message |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to search the available medicine for easily viewing and managing whenever this page has large amounts of medicine

2.1.1.19 Use Case 16:

**Name: Search invoice**

**Identifier** *UC16*

**Inputs:**

1. Customer phone

**Outputs:**

1. Object of invoice that has the same searched number phone [If success]
2. Notification on top: “No invoices found for this phone number” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Sales & Invoices page | 1.1. Display Sales & Invoices page |
| 2. Click “Search by customer phone” box and insert phone number of customer | |
| 3. Press Enter button or click Search icon | 3.1. Check the available phone number<br>3.2. If success, show a list of related invoice and a message<br>3.3. If not found, show fail message |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to search the available invoice for easily viewing and managing whenever this page has large amounts of invoice

2.1.1.20 Use Case 17:

**Name: Search Customers**

**Identifier** *UC17*

**Inputs:**

1. Customer phone

**Outputs:**

1. Object of customer that has the same searched number phone [If success]

2. Notification on top: “No customer found for phone number ‘number’” [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Access Customers page | 1.1. Display Customers page |
| 2. Click “Search customers…” box and insert phone number of customer | |
| 3. Press Enter button or click Search icon | 3.1. Check the available phone number<br>3.2. If success, show related customer and a message<br>3.3. If not found, show fail message |

**Precondition**

1. None

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to search the available customers for easily viewing and managing whenever this page has large amounts of customers

2.1.1.21 Use Case 18:

**Name: Login to the system**

**Identifier** *UC18*

**Inputs:**

1. Username

2. Password

**Outputs:**

1. The home page with user's authorization [If success]
2. The login page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Open login page | 1.1. Display login page |
| 2. Enter user name and password | |
| 3. Click Sign in | 3.1. Check the user's information<br>3.2. If success, return to the home page<br>3.3. If not found, show fail message and return to the login page |

**Precondition**

1. User has an registered account of online store that is created earlier (username and password)

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to login to the system so that I can start viewing and managing medicines, categories, suppliers, …

2.1.1.22 Use Case 19:

**Name: Change profile's information**

**Identifier** *UC19*

**Inputs:**

1. Full Name
2. Username
3. Email
4. Old password
5. New password

**Outputs:**

1. Notification “Profile updated successfully” [If success]
2. The profile page [If fail]

**Basic Course**

| Actor: Admin/Pharmacist | System |
|---|---|
| 1. Open profile page by clicking on avatar | 1.1. Display profile page |
| 2. Enter informations that are needed to be changed | |
| 3. Click Save Changes | 3.1. Check the new information<br>3.2. If success, show success message<br>3.3. If not found, show fail message and return to the profile page |

**Precondition**

1. User has already login

**Post condition**

1. None

**User story:**

As an Admin/Pharmacist of this web page, I want to modify my account any reason (security, new personal information, …)

### 2.1.2 FUNCTIONAL REQUIREMENTS

#### 2.1.2.1 Use Case 1: Login

**The Scope of the Work**

- This occurs during the Backend Development Phase of the process.
- 4 tasks are required to implement this functionality.
- 15 hours of effort is estimated for this functionality.

**The Scope of the Product**

- This functionality is part of the authentication system, enabling users to securely log in to their accounts.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall display a login form for users to input their credentials.
  - Shall validate the provided credentials against the database.
  - Shall encrypt and securely store user passwords in the database.
  - Shall notify users of any invalid login attempts.
- **Data Requirements:**
  - Login credentials (Username/ID and Password) must be valid.
  - All user information must be securely encrypted in the database.

#### 2.1.2.2 Use Case 2: View Pharmacist Account

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 3 tasks are required.
- 12 hours of effort is estimated.

**The Scope of the Product**

- Displays a list of pharmacists registered in the system.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall query the database for pharmacist accounts.
  - Shall display a paginated list of pharmacists.
- **Data Requirements:**
  - Updated pharmacist data (name, ID, contact).

2.1.2.3 Use Case 3: Add Pharmacist Account

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 5 tasks are required.
- 20 hours of effort is estimated.

**The Scope of the Product**

- Allows admins to add a new pharmacist account.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall display a form for inputting details.
  - Shall validate input data (e.g., unique ID).
  - Shall store new account data in the database.
- **Data Requirements:**
  - Valid pharmacist data (name, email, role).

2.1.2.4 Use Case 4: Edit Pharmacist Account

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.

- 4 tasks are required.
- 15 hours of effort is estimated.

**The Scope of the Product**

- Allows admins to update pharmacist details.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall retrieve pharmacist data for editing.
  - Shall validate updated information.
  - Shall store the modified data in the database.
- **Data Requirements:**
  - Updated data must conform to constraints.

2.1.2.5 Use Case 5: Delete Pharmacist Account

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 3 tasks are required.
- 10 hours of effort is estimated.

**The Scope of the Product**

- Removes pharmacist accounts from the system.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall provide a delete option for each pharmacist.
  - Shall confirm the action before deletion.
  - Shall update the database after deletion.
- **Data Requirements:**
  - Deleted records are no longer accessible.

2.1.2.6 Use Case 6: Edit Profile

**The Scope of the Work**

- Occurs during the Frontend Development Phase.
- 3 tasks are required.

- 8 hours of effort is estimated.

**The Scope of the Product**

- Enables users to update their profile information.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall display editable profile details.
  - Shall validate changes.
  - Shall save updated profile data to the database.
- **Data Requirements:**
  - Profile data must comply with validation rules.

2.1.2.7 Use Case 7: Search Medicine

**The Scope of the Work**

- Occurs during the Frontend Development Phase.
- 3 tasks are required.
- 10 hours of effort is estimated.

**The Scope of the Product**

- Allows users to search medicines by name.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall provide a search bar for input.
  - Shall display search results from the database.
- **Data Requirements:**
  - Medicine information must be updated in the database.

2.1.2.8 Use Case 8: Add Medicine

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 5 tasks are required.
- 15 hours of effort is estimated.

**The Scope of the Product**

- Allows admins/ pharmacist to add new medicine to the system.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall display a form for inputting medicine details.
  - Shall validate and store medicine information in the database.
- **Data Requirements:**
  - Medicine details (name, stock, price, expiry date) must be accurate.

#### 2.1.2.9 Use Case 9: Edit Medicine

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 4 tasks are required.
- 12 hours of effort is estimated.

**The Scope of the Product**

- Allows users to update existing medicine details.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall retrieve existing details for editing.
  - Shall validate updated information.
- **Data Requirements:**
  - Updated medicine data must conform to constraints.

#### 2.1.2.10 Use Case 10: Delete Medicine

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 3 tasks are required.
- 8 hours of effort is estimated.

**The Scope of the Product**

- Allows admins to remove medicines from the inventory.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall provide a delete option for each medicine.
  - Shall confirm the action before deletion.
  - Shall update the database to reflect the deletion.
- **Data Requirements:**
  - Deleted records must not appear in the search results or lists.

#### 2.1.2.11 Use Case 11: Edit Category

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 4 tasks are required.
- 12 hours of effort is estimated.

**The Scope of the Product**

- Enables admins to edit existing medicine categories.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall retrieve the category details for editing.
  - Shall validate the updated category name or description.
  - Shall save the changes in the database.
- **Data Requirements:**
  - Updated category data must comply with naming rules and constraints.

#### 2.1.2.12 Use Case 12: Add Supplier

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 5 tasks are required.
- 15 hours of effort is estimated.

**The Scope of the Product**

- Enables admins to add supplier information to the system.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall display a form for inputting supplier details.
  - Shall validate the supplier's name, address, and contact information.
  - Shall store the new supplier details in the database.
- **Data Requirements:**
  - Supplier information must be unique and valid.

#### 2.1.2.13 Use Case 13: Edit Supplier

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 4 tasks are required.
- 12 hours of effort is estimated.

**The Scope of the Product**

- Enables admins to update supplier information.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall retrieve existing supplier details for editing.
  - Shall validate the updated data.
  - Shall save the changes in the database.
- **Data Requirements:**
  - Updated supplier details must not conflict with existing records.

#### 2.1.2.14 Use Case 14: Delete Supplier

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 3 tasks are required.
- 8 hours of effort is estimated.

**The Scope of the Product**

- Allows admins to delete supplier records.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall display a delete option for each supplier.
  - Shall confirm the action before deletion.
  - Shall remove the supplier record from the database.
- **Data Requirements:**
  - Deleted supplier records must not appear in searches or reports.

#### 2.1.2.15 Use Case 15: Search Invoice

**The Scope of the Work**

- Occurs during the Frontend Development Phase.
- 3 tasks are required.
- 10 hours of effort is estimated.

**The Scope of the Product**

- Allows users to search for invoices by ID, customer, or date.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall provide a search bar for invoice queries.
  - Shall display results in a paginated list.
- **Data Requirements:**
  - Invoice data must be accurate and up-to-date in the database.

#### 2.1.2.16 Use Case 16: Add Invoice

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 5 tasks are required.
- 15 hours of effort is estimated.

**The Scope of the Product**

- Enables users to add new invoices to the system.

**Functional and Data Requirements**

- **Functional Requirements:**
    - Shall display a form for inputting invoice details.
    - Shall calculate totals and apply discounts or taxes.
    - Shall store invoice data in the database.
- **Data Requirements:**
    - Invoice details (customer, date, items) must be valid.

#### 2.1.2.17 Use Case 17: Edit Invoice

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 4 tasks are required.
- 12 hours of effort is estimated.

**The Scope of the Product**

- Allows users to update existing invoice details.

**Functional and Data Requirements**

- **Functional Requirements:**
    - Shall retrieve invoice data for editing.
    - Shall validate updated data (e.g., totals).
- **Data Requirements:**
    - Updated invoice data must be consistent with constraints.

#### 2.1.2.18 Use Case 18: delete Invoice

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 3 tasks are required.
- 8 hours of effort is estimated.

**The Scope of the Product**

- Enables users to remove invoices from the system.

**Functional and Data Requirements**

- **Functional Requirements:**

    - Shall confirm deletion actions.
    - Shall update the database to reflect deletions.
- **Data Requirements:**
    - Deleted invoices must not appear in reports or summaries.

#### 2.1.2.19 Use Case 19: Search Customer

**The Scope of the Work**

- Occurs during the Frontend Development Phase.
- 3 tasks are required.
- 10 hours of effort is estimated.

**The Scope of the Product**

- Enables users to search for customer details.

**Functional and Data Requirements**

- **Functional Requirements:**
    - Shall provide a search bar for queries.
    - Shall display customer details in a list format.
- **Data Requirements:**
    - Customer data must be accurate in the database.

#### 2.1.2.20 Use Case 20: Add Customer

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 5 tasks are required.
- 15 hours of effort is estimated.

**The Scope of the Product**

- Allows admins to add new customer records.

**Functional and Data Requirements**

- **Functional Requirements:**
    - Shall display a form for customer details.

- Shall validate and store customer information in the database.

- **Data Requirements:**
  - Customer details (name, contact) must be unique and valid.

#### 2.1.2.21 Use Case 21: Edit Customer

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 4 tasks are required.
- 12 hours of effort is estimated.

**The Scope of the Product**

- Allows users to update customer records.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall retrieve customer data for editing.
  - Shall validate changes.
- **Data Requirements:**
  - Updated customer data must conform to validation rules.

#### 2.1.2.22 Use Case 22: Delete Customer

**The Scope of the Work**

- Occurs during the Backend and Frontend Development Phases.
- 3 tasks are required.
- 8 hours of effort is estimated.

**The Scope of the Product**

- Enables admins/ pharmacist to delete customer records.

**Functional and Data Requirements**

- **Functional Requirements:**
  - Shall confirm deletion actions.
  - Shall remove customer records from the database.

- **Data Requirements:**
    - Deleted records must not appear in searches or lists.

### 2.1.3 NON-FUNCTIONAL REQUIREMENTS

#### 2.1.3.1 Operational Requirements

- Ensure all components (database, backend, and frontend) function seamlessly in an integrated manner.
- Maintain hosting with sufficient capacity to handle concurrent user requests efficiently.
- Assign an administrator to regularly monitor the system, apply updates, and handle error reporting.

#### 2.1.3.2 Legal Requirement

- Cite all third-party libraries, frameworks, and tools used in the system to avoid copyright infringement.
- Protect all custom-written source code to prevent unauthorized access or replication.

#### 2.1.3.3 Usability Requirements

- The system must be easy to use for users with minimal technical knowledge.
- Provide:
    - Clear user manuals for system functionalities.
    - Informative and user-friendly error messages, including solutions where applicable.
    - Help features integrated into the interface.
    - Responsive and intuitive user interfaces for all devices and screen sizes.
- Efficiency: Tasks (e.g., searching medicines or adding invoices) should be completed quickly and with minimal errors.

#### 2.1.3.4 Humanity Requirements

- Provide detailed documentation and training materials for non-technical users (e.g., store managers).
- Design an intuitive graphical user interface (GUI) that is both attractive and easy to navigate.
- Minimize the learning curve for new users while maintaining responsiveness and usability.

#### 2.1.3.5 Performance Requirements

Here's the tailored version of Non-Functional Requirements to match your project:

### 2.1.4 NON-FUNCTIONAL REQUIREMENTS

#### 2.1.4.1 Operational Requirements

Ensure all components (database, backend, and frontend) function seamlessly in an integrated manner.

Maintain hosting with sufficient capacity to handle concurrent user requests efficiently.

Assign an administrator to regularly monitor the system, apply updates, and handle error reporting.

#### 2.1.4.2 Legal Requirements

Cite all third-party libraries, frameworks, and tools used in the system to avoid copyright infringement.

Protect all custom-written source code to prevent unauthorized access or replication.

#### 2.1.4.3 Usability Requirements

The system must be easy to use for users with minimal technical knowledge.

Provide:

Clear user manuals for system functionalities.

Informative and user-friendly error messages, including solutions where applicable.

Help features integrated into the interface.

Responsive and intuitive user interfaces for all devices and screen sizes.

Efficiency: Tasks (e.g., searching medicines or adding invoices) should be completed quickly and with minimal errors.

#### 2.1.4.4 Humanity Requirements

Provide detailed documentation and training materials for non-technical users (e.g., store managers).

Design an intuitive graphical user interface (GUI) that is both attractive and easy to navigate.

Minimize the learning curve for new users while maintaining responsiveness and usability.

#### 2.1.4.5 Performance Requirements

a. Response Requirements

- Ensure fast response times:
  - Login for both customers and administrators must occur within 2 seconds.
  - Database updates (e.g., adding medicines, customers) must be processed in under 3 seconds.
  - Generate reports (e.g., invoices or inventory summaries) seamlessly without noticeable delays.
- The website should load pages (e.g., product pages with up to 50 items) within 2-3 seconds.

b. Throughput Requirements

- Handle simultaneous requests efficiently:
  - Support up to 100 concurrent database modifications (e.g., adding invoices or updating customer profiles).
  - Support 1,000 simultaneous visitors browsing the system.

c. Availability Requirements

- Ensure the system is available 24/7 with minimal downtime.

- Implement efficient memory management and garbage collection to avoid system crashes during high traffic periods.

#### 2.1.4.6 Maintainability Requirements

- Design the system for easy updates and modifications:
  - Enable users to update/manage products, customers, and invoices without requiring technical assistance.
  - Write clean, well-documented code with clear comments for future maintenance.
  - Use a scalable database schema that can be expanded or modified easily.

#### 2.1.4.7 Support Requirements

- Provide the following to assist non-technical users (e.g., store managers):
  - A dedicated support hotline for immediate assistance.
  - Remote support tools (e.g., TeamViewer) for troubleshooting.
  - Scheduled maintenance checks (monthly or bi-monthly) to ensure system stability.

#### 2.1.4.8 Security Requirements

- Protect sensitive data and maintain system integrity:
  - Encrypt all communications between clients and servers (e.g., use SSL for HTTP).
  - Restrict login attempts: After five failed attempts, lock the account for 24 hours.
  - Limit data modification permissions to authorized administrators.
  - Log all system modification events (e.g., user ID, timestamp, action).
  - Encrypt customer information (e.g., passwords, billing details) before storing it in the database.
  - Perform daily backups of all system data, storing copies in a secure offsite location.

- Implement session management with timeouts for inactive users.

#### 2.1.4.9 Interface Requirements

- Ensure smooth integration between all system components:
  - Backend APIs must respond reliably to frontend requests.
  - Use well-defined error handling and buffer management protocols.
- Interface-specific non-functional requirements:
  - Include language options for both native and foreign users.
  - Use recognizable icons and labels on buttons for intuitive navigation.
  - Display sales reports automatically after successful transactions.
  - Maintain consistent design and responsive layouts across devices.

## 2.2 Design

### 2.2.1 System Architecture Model

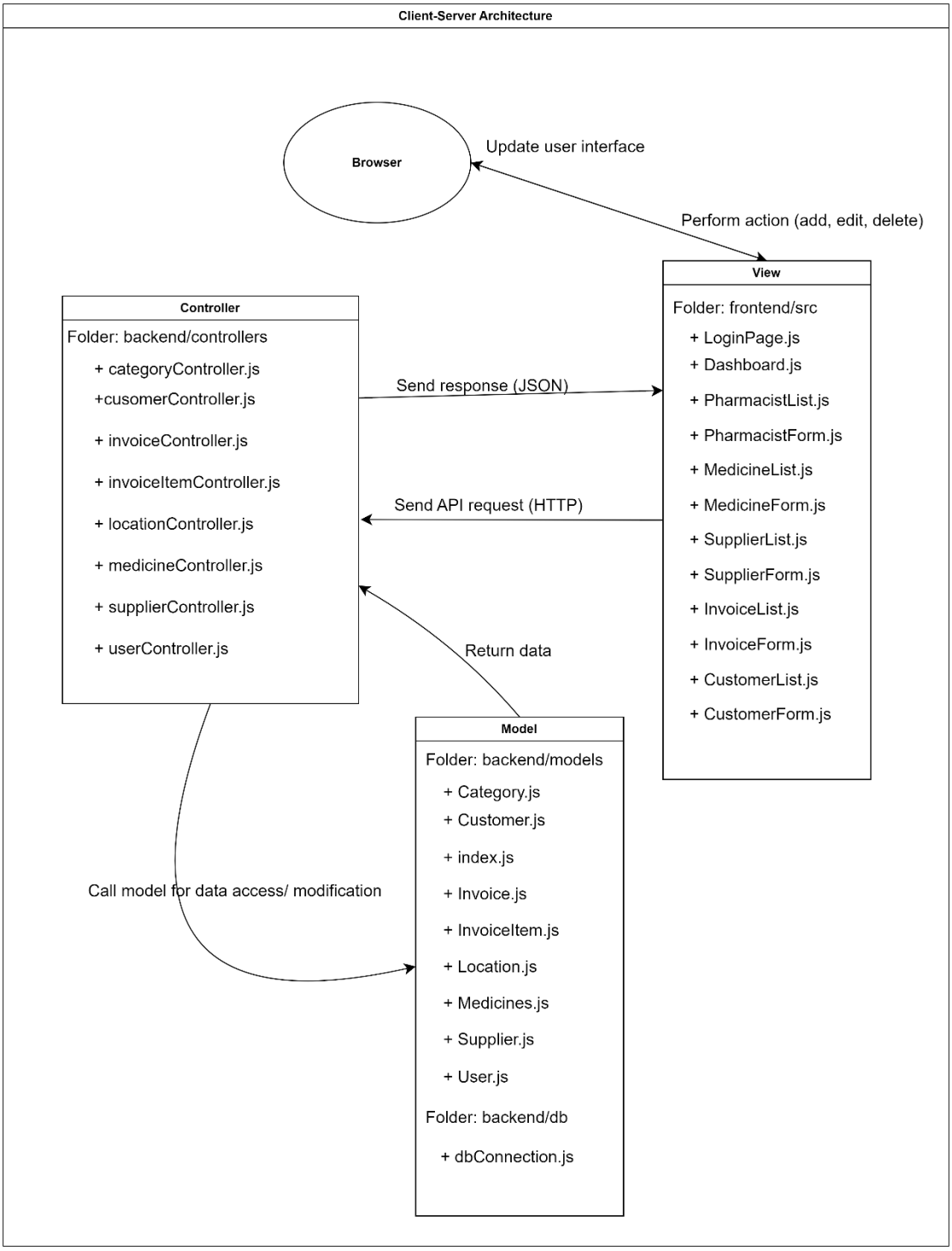



*Figure 12*

## 2.2.2 Entity – Relationship Diagram (ERD)

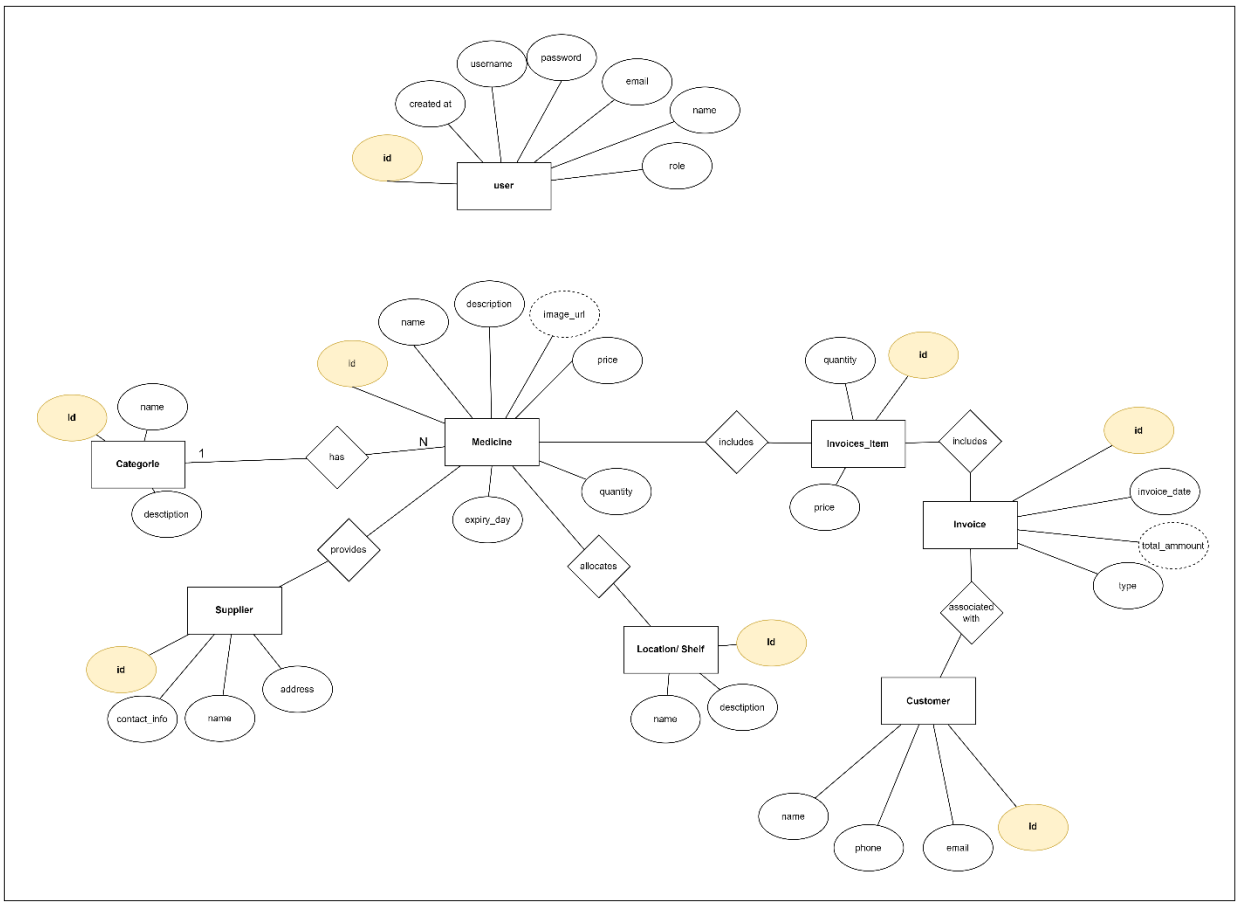



*Figure 13*

### 2.2.3 Data Model

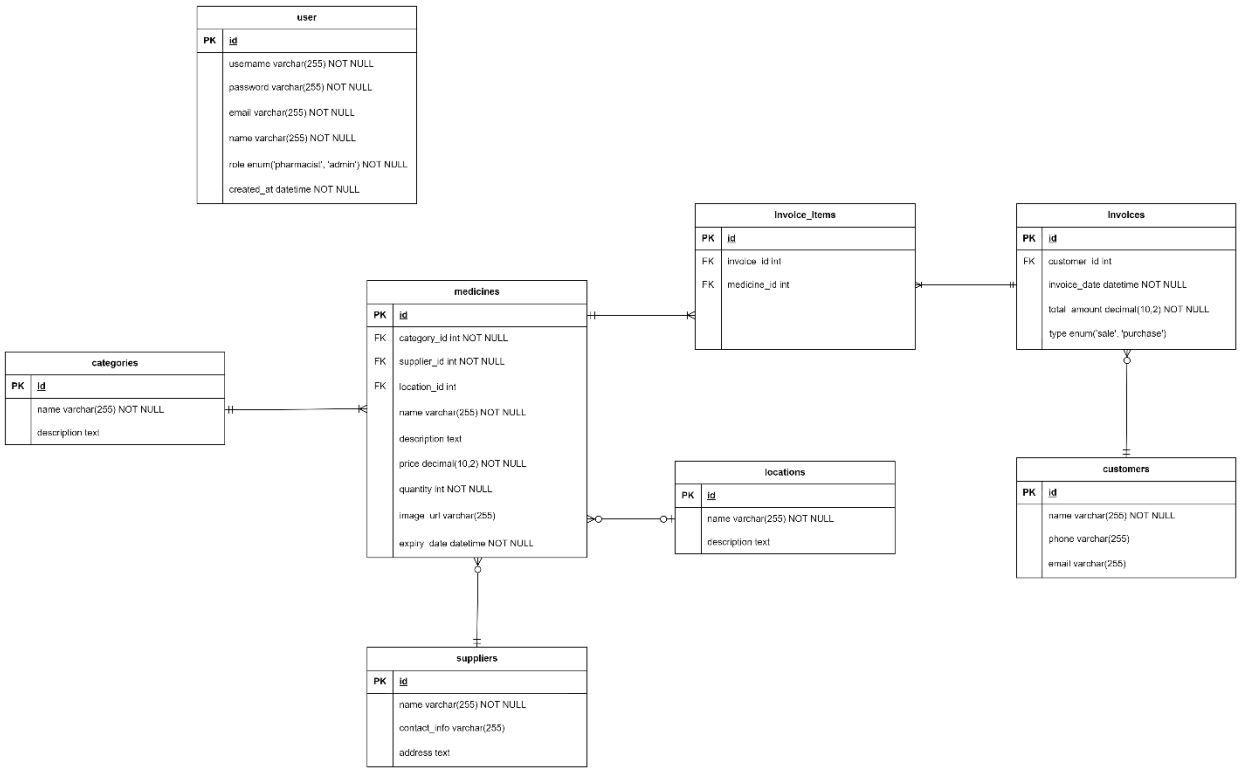



*Figure 14*

### 2.2.4 Class Diagram

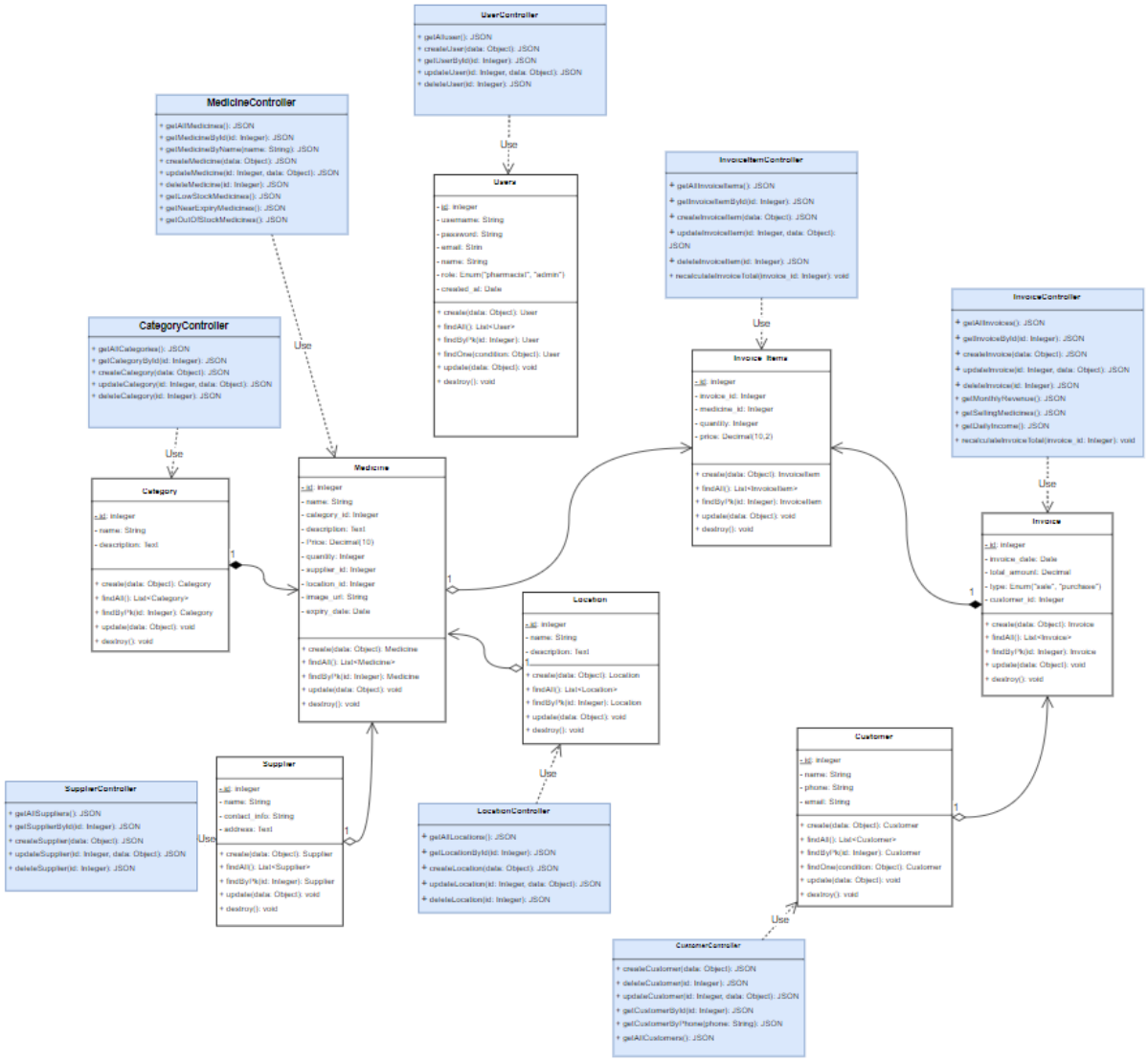



*Figure 15*

### 2.2.5 Use Case Diagram

#### 2.2.5.1 Use case diagram:

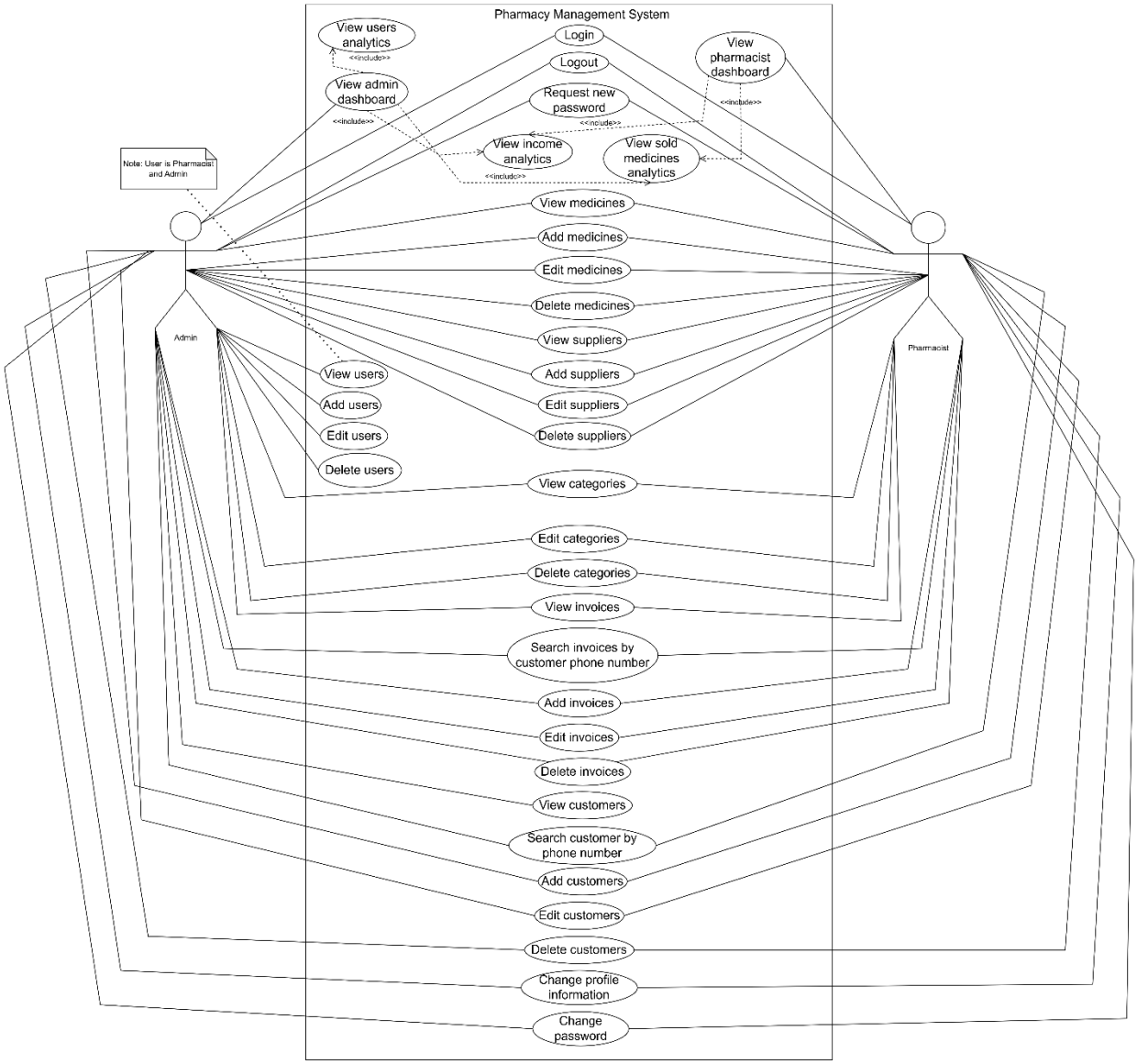



*Figure 16*

### 2.2.5.2 User goal use case diagram

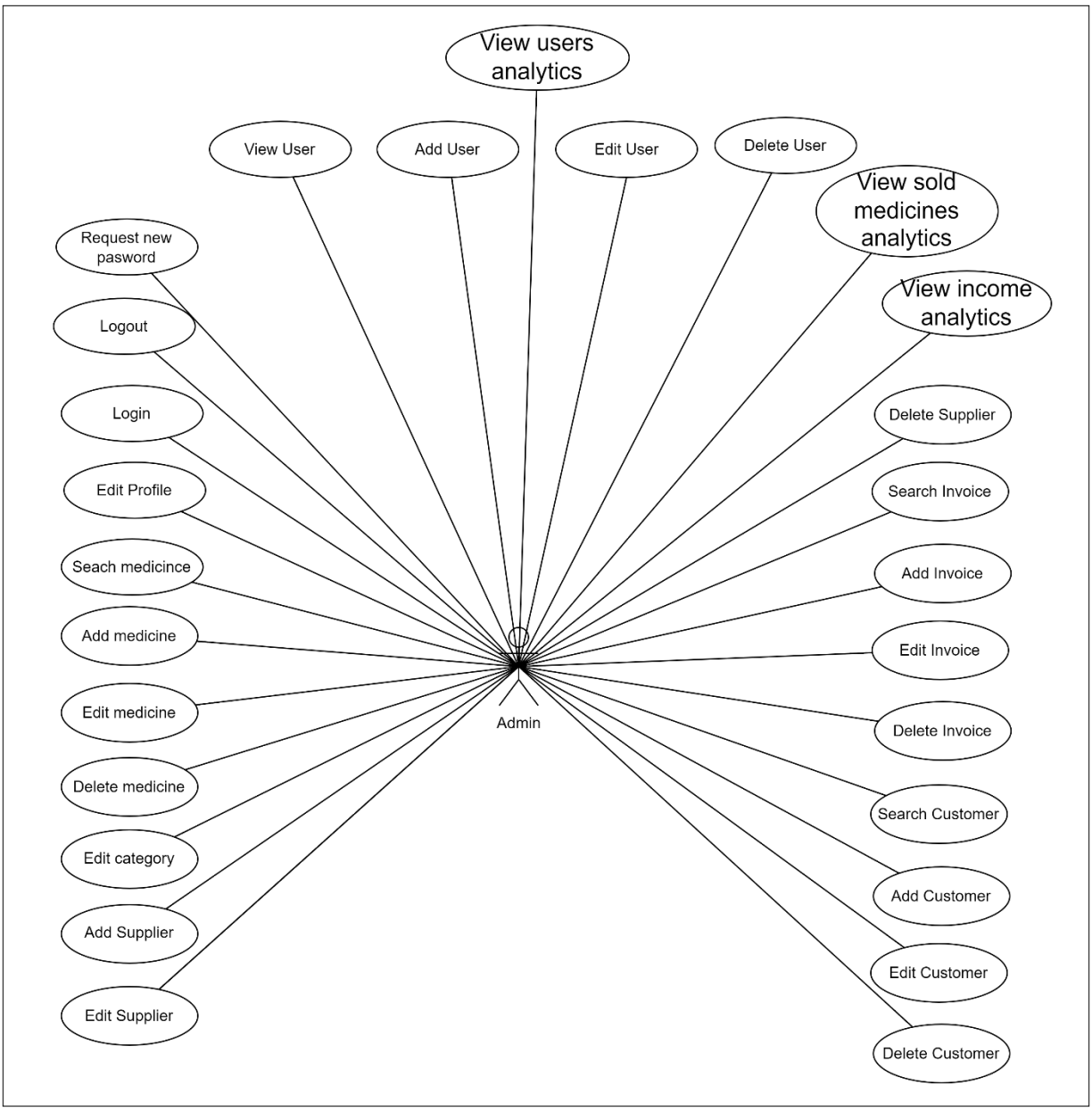



*Figure 17*

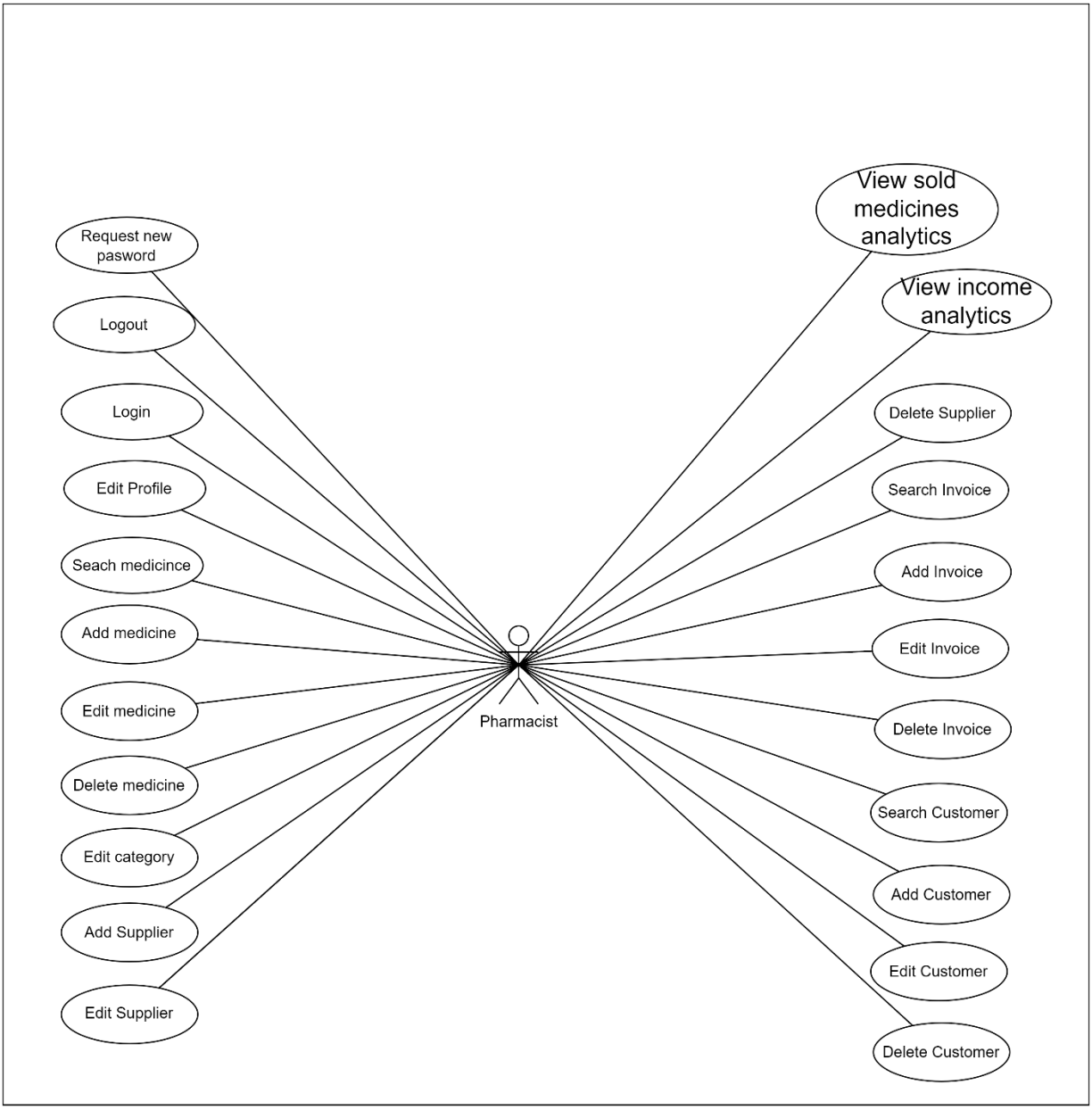



*Figure 18*

### 2.2.5.3 Sequence Diagram for Usecases:

a. Admin

1. Login

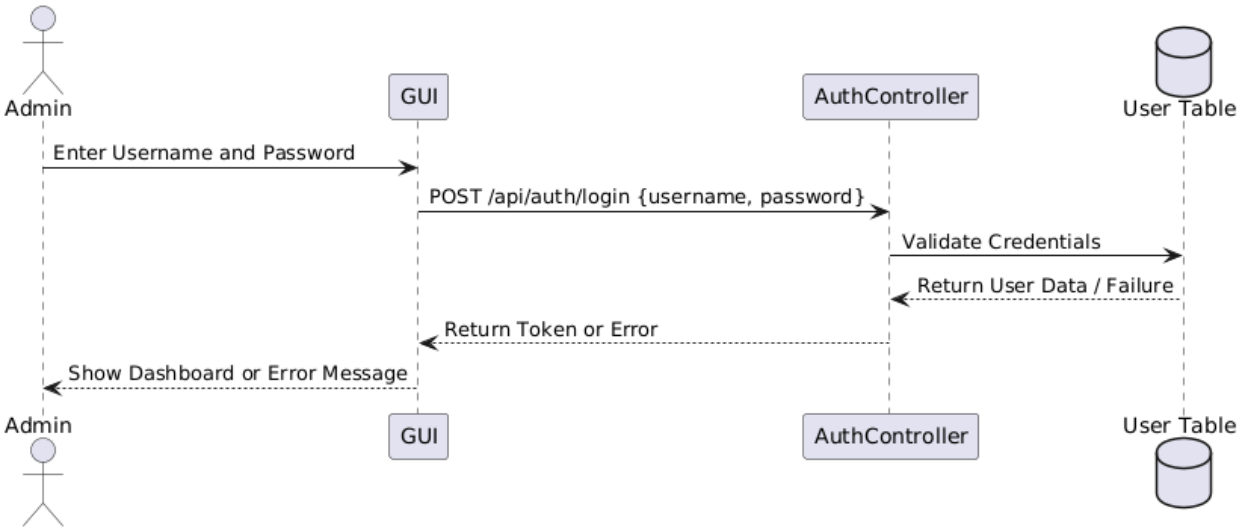



*Figure 19*

2. Edit profile

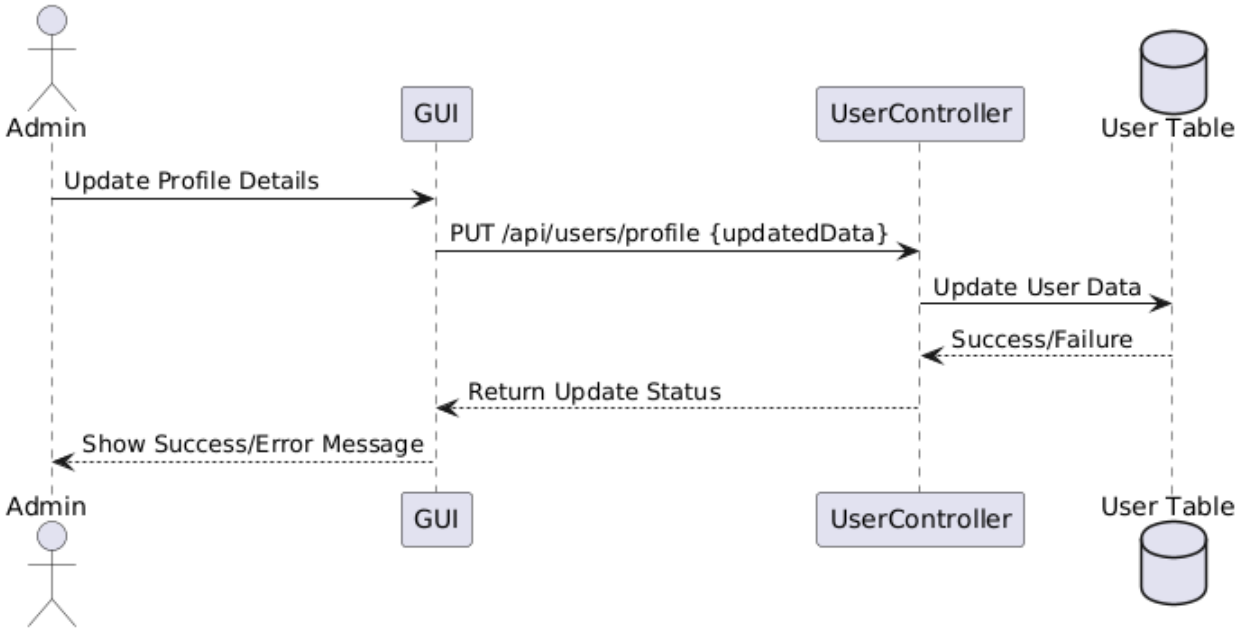



*Figure 20*

3. Search medicine

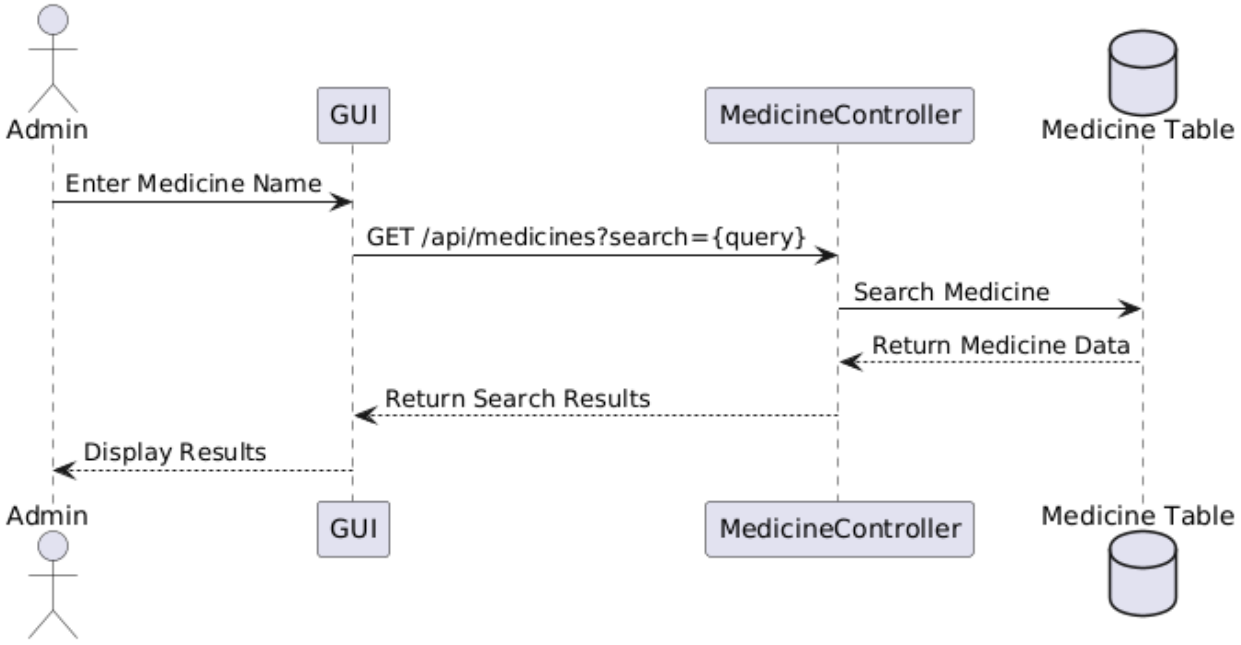



*Figure 21*

4. Add medicine

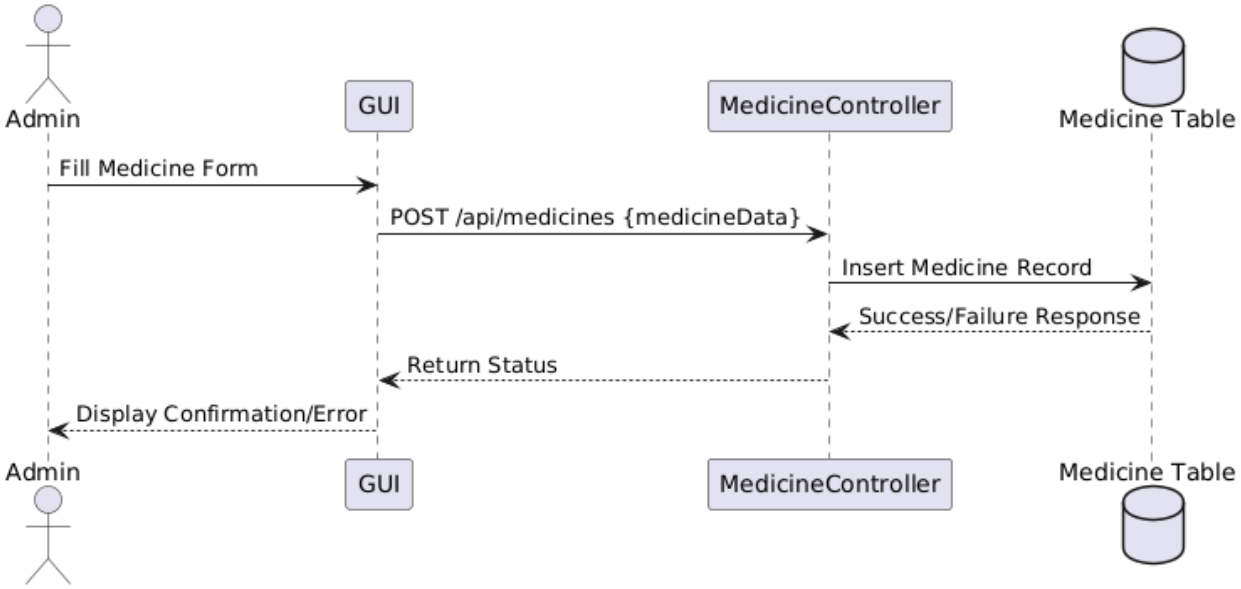



*Figure 22*

5. Edit medicine

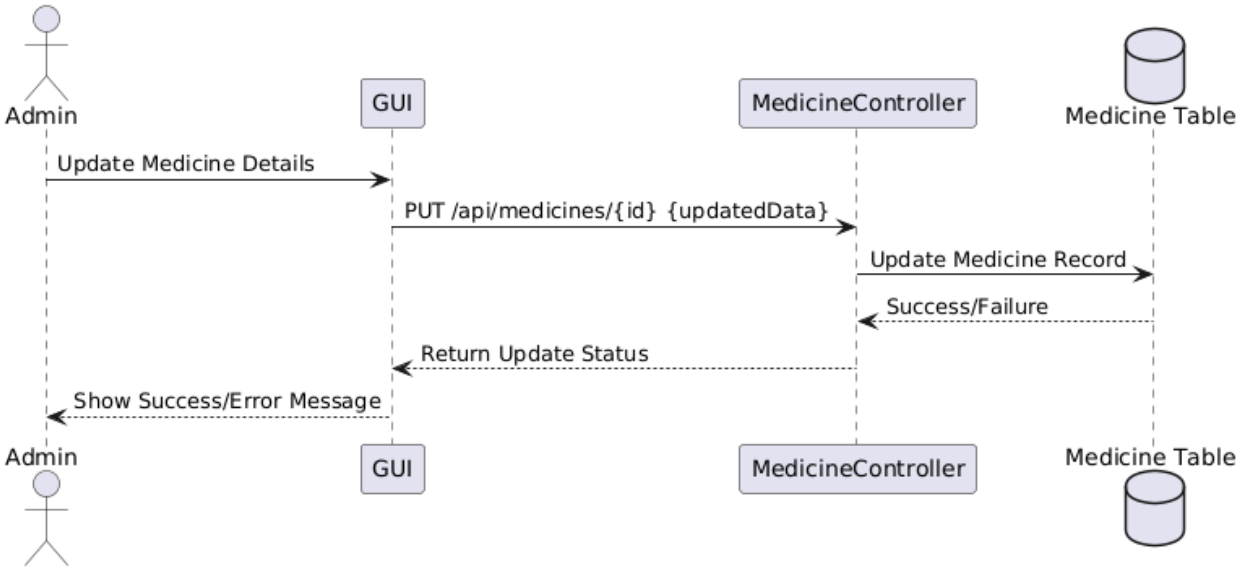



*Figure 23*

6. Delete medicine

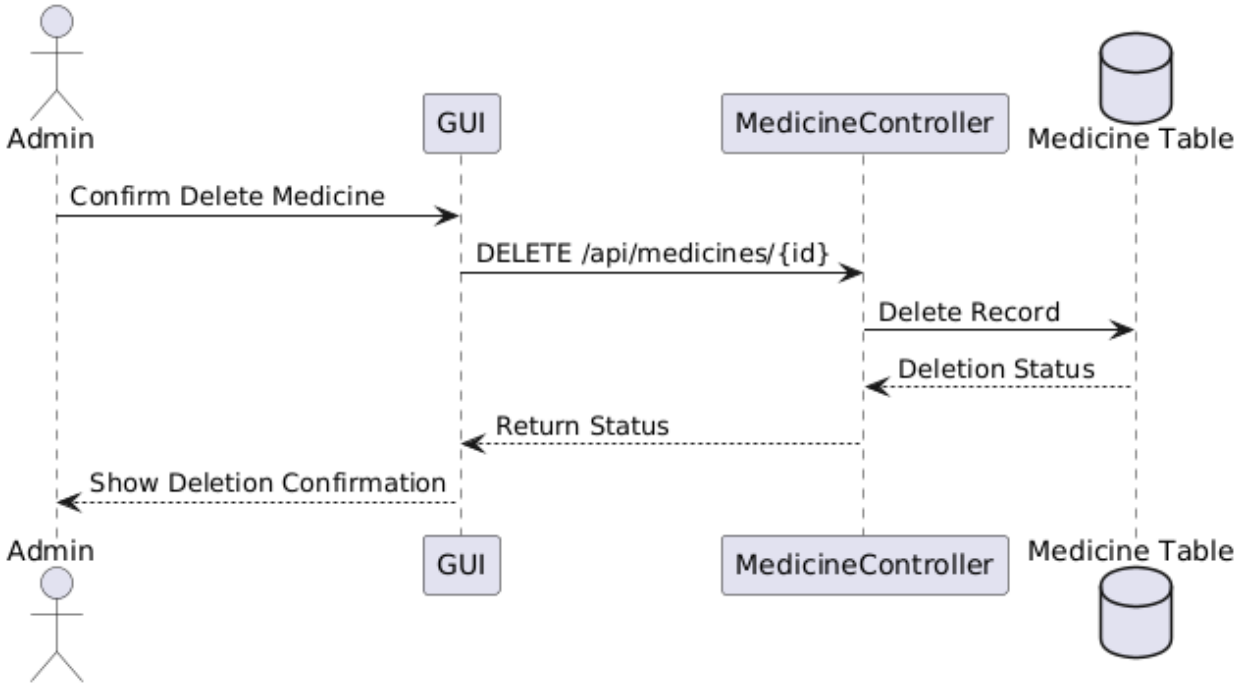



*Figure 24*

7. Edit category

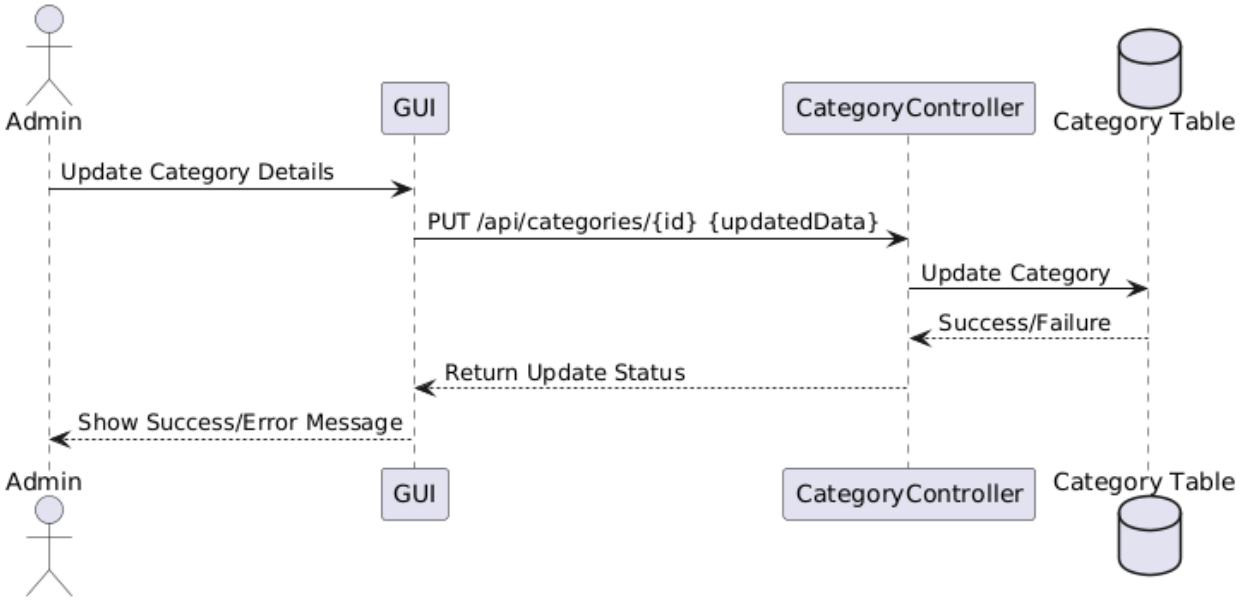



*Figure 25*

8. Add supplier

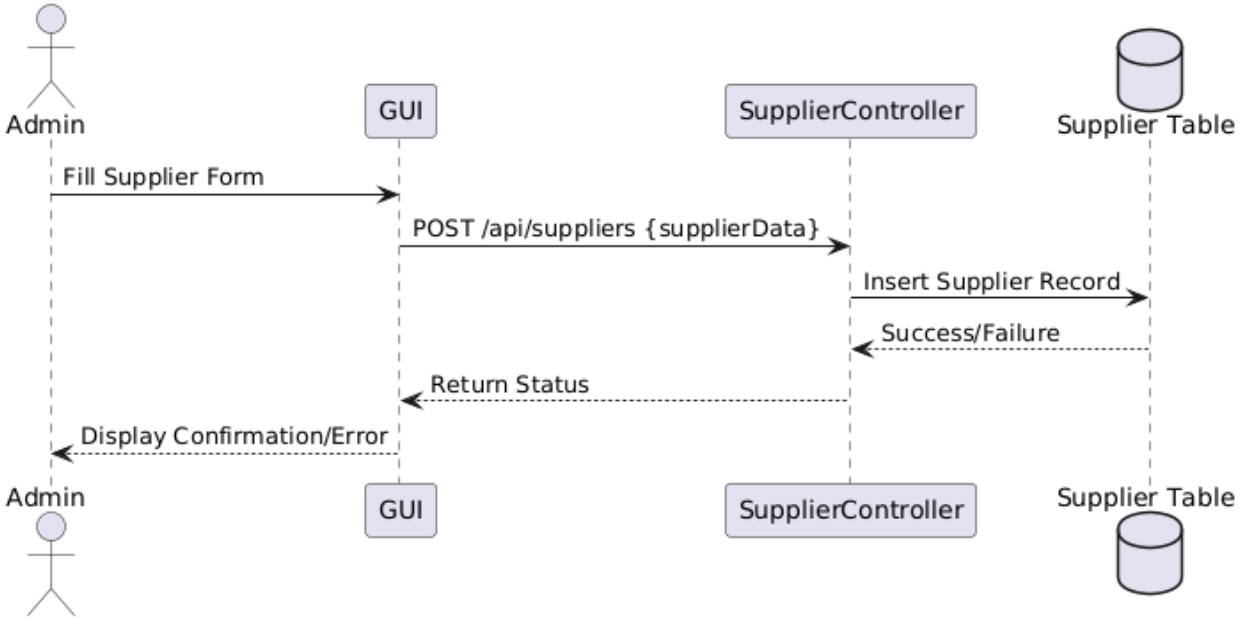



*Figure 26*

9. Edit supplier

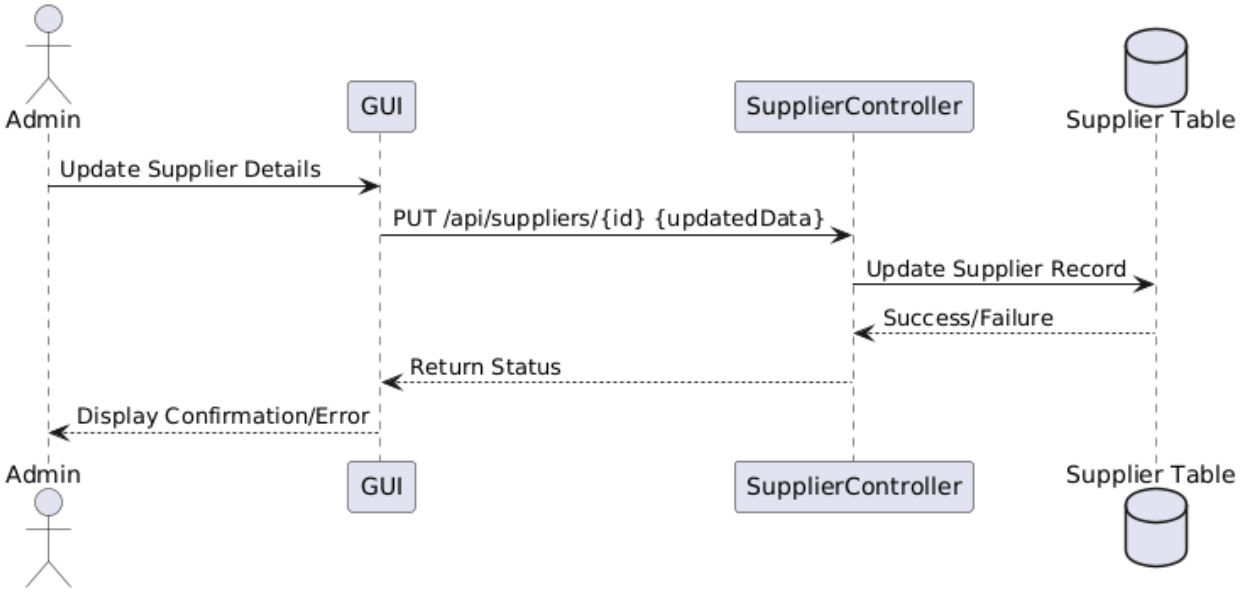



*Figure 27*

10. Delete supplier

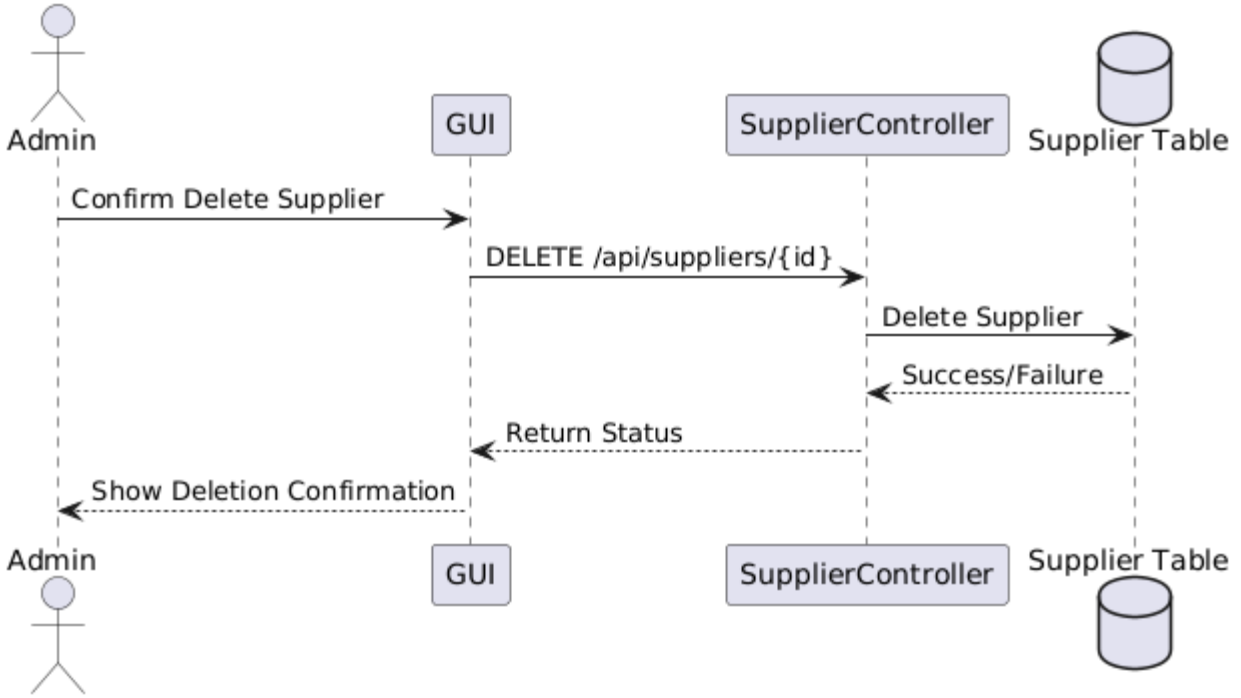



*Figure 28*

11. Search invoice

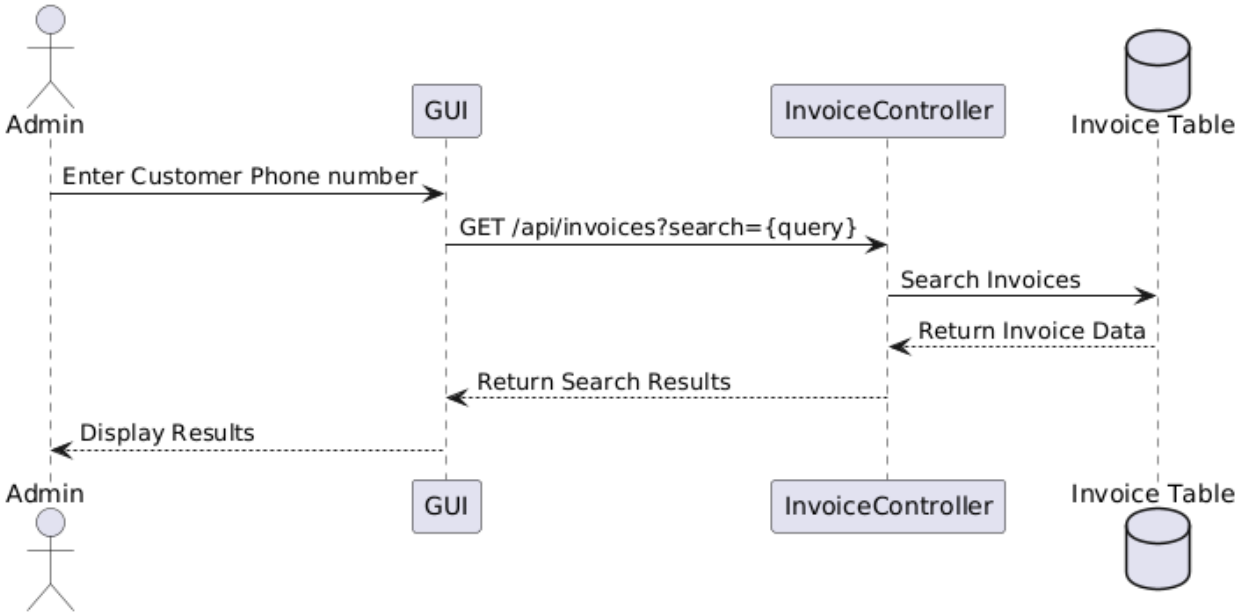



*Figure 29*

12. Add invoice

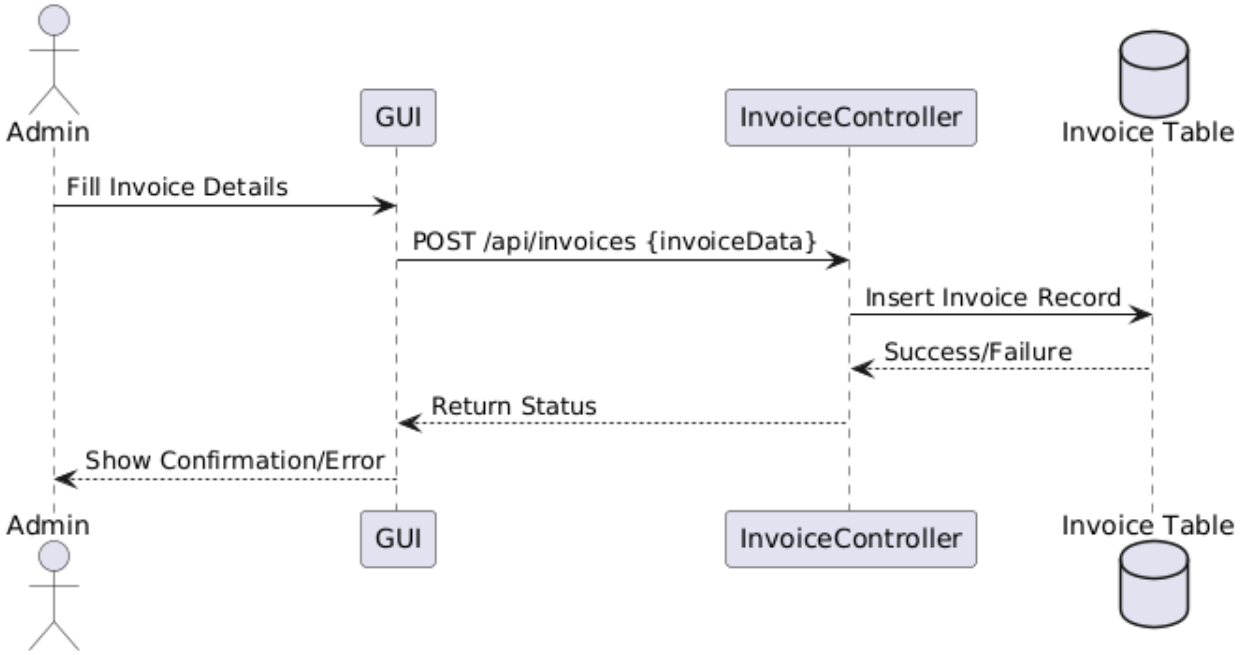



*Figure 30*

13. Edit invoice

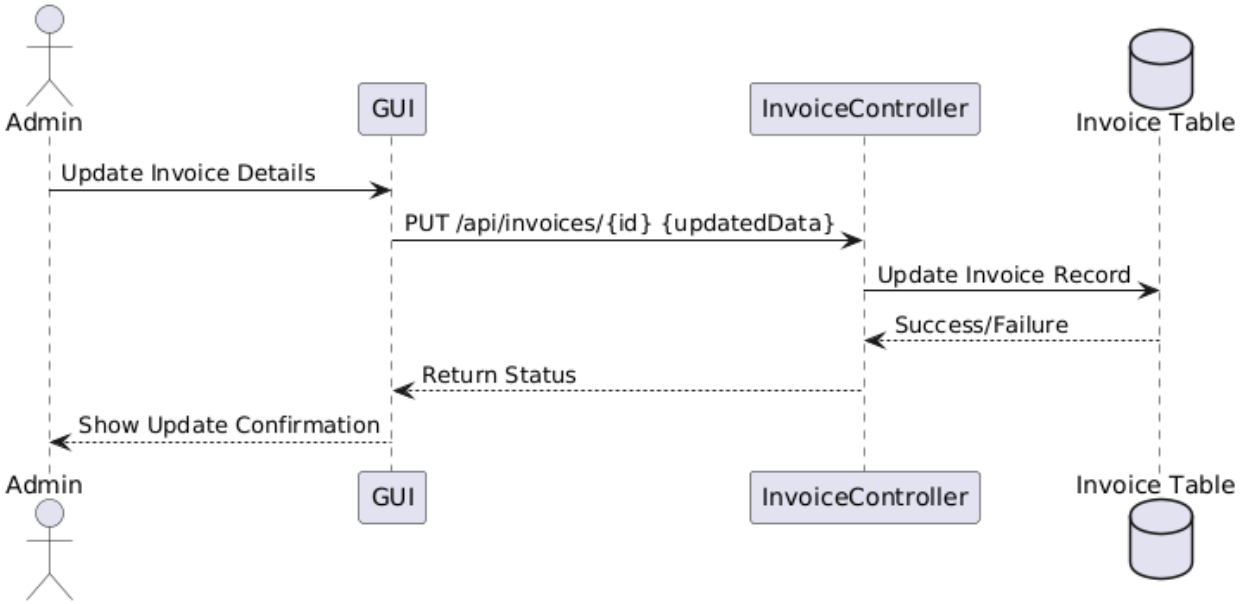



*Figure 31*

14. Delete invoice

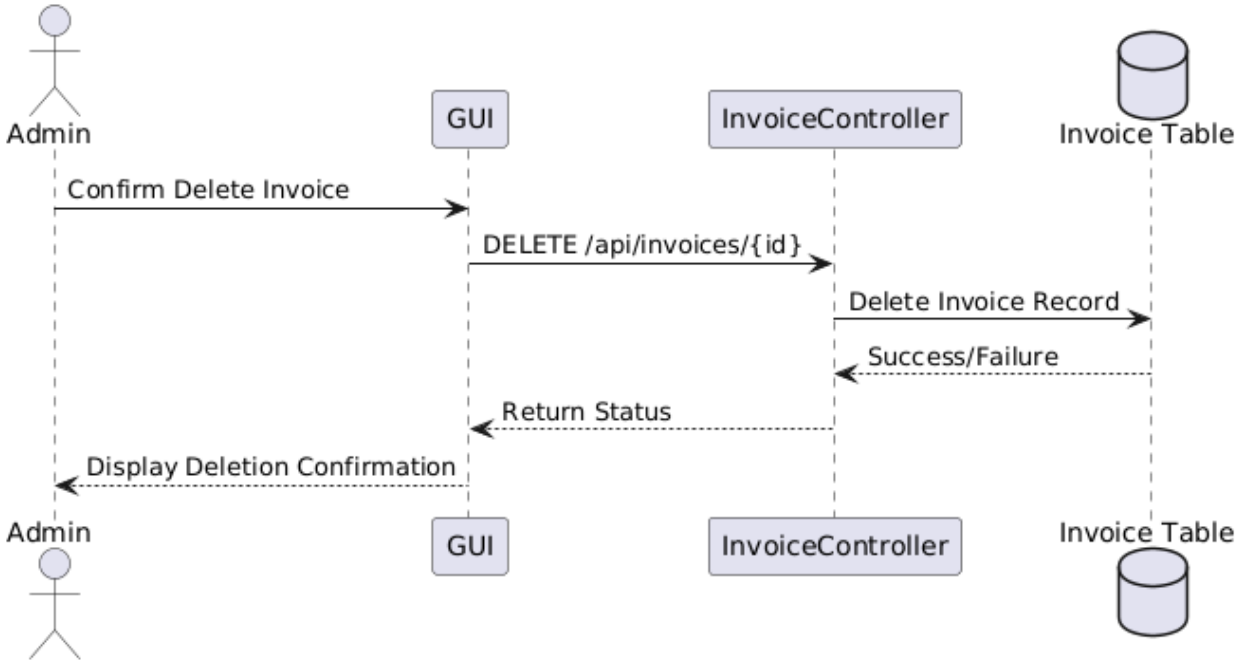



*Figure 32*

15. Search customer

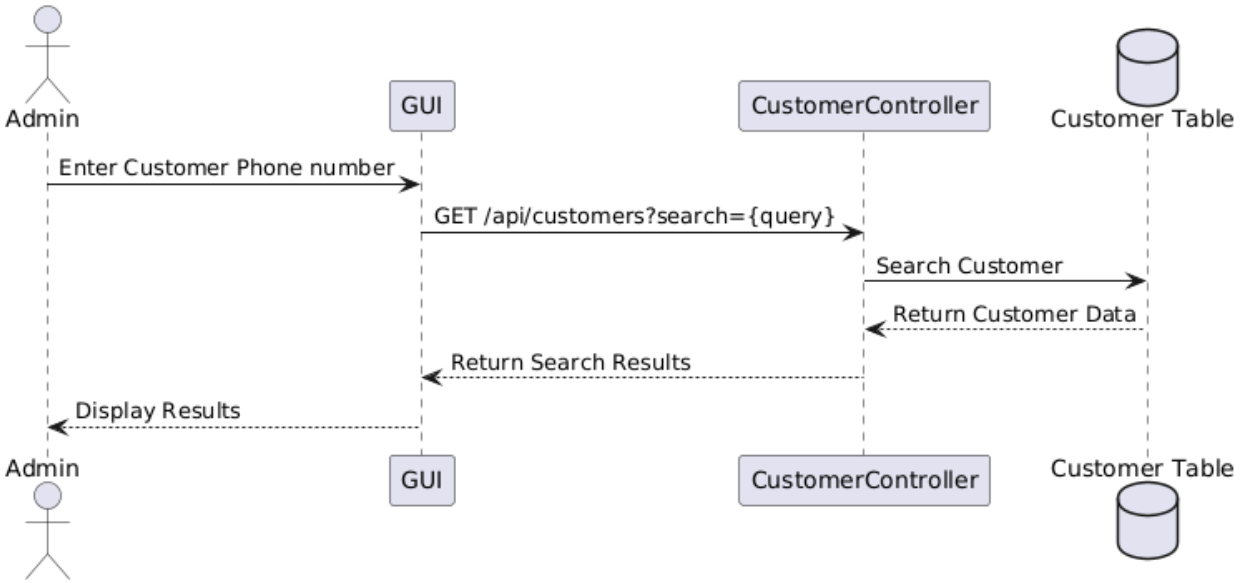



*Figure 33*

16. Add customer

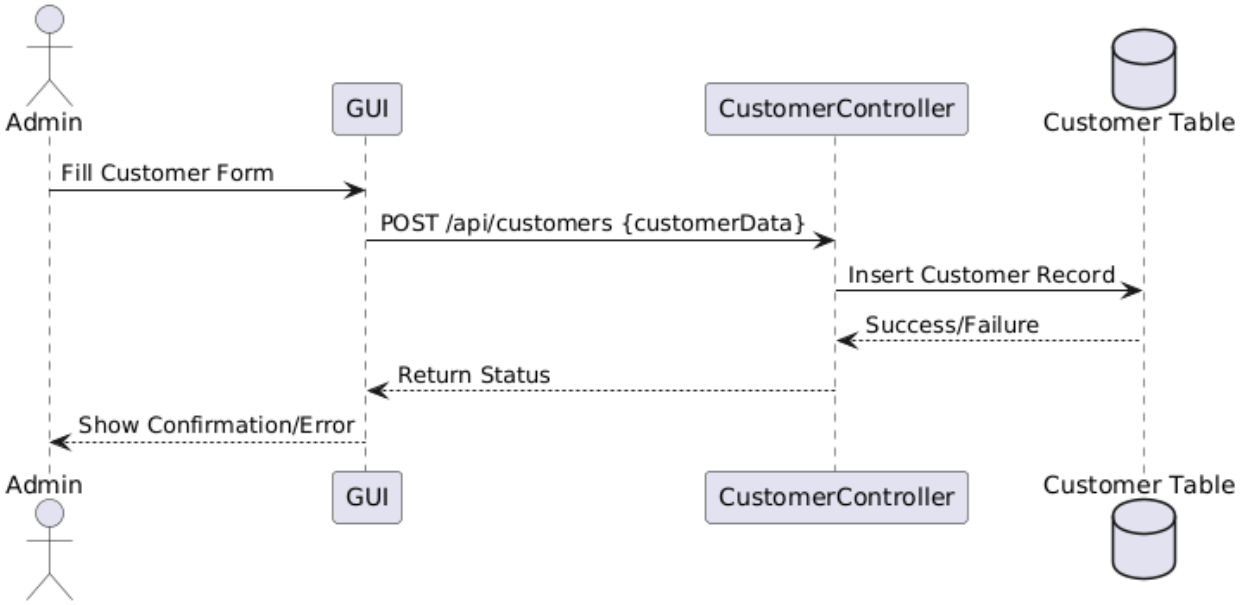



*Figure 34*

17. Edit customer

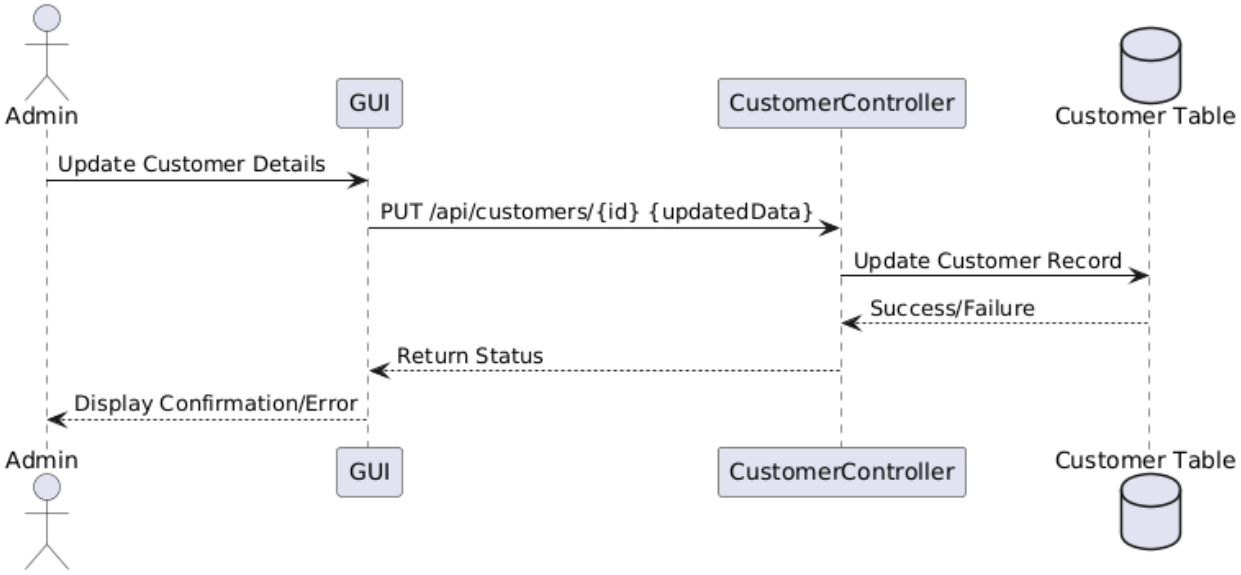



*Figure 35*

18. Delete customer

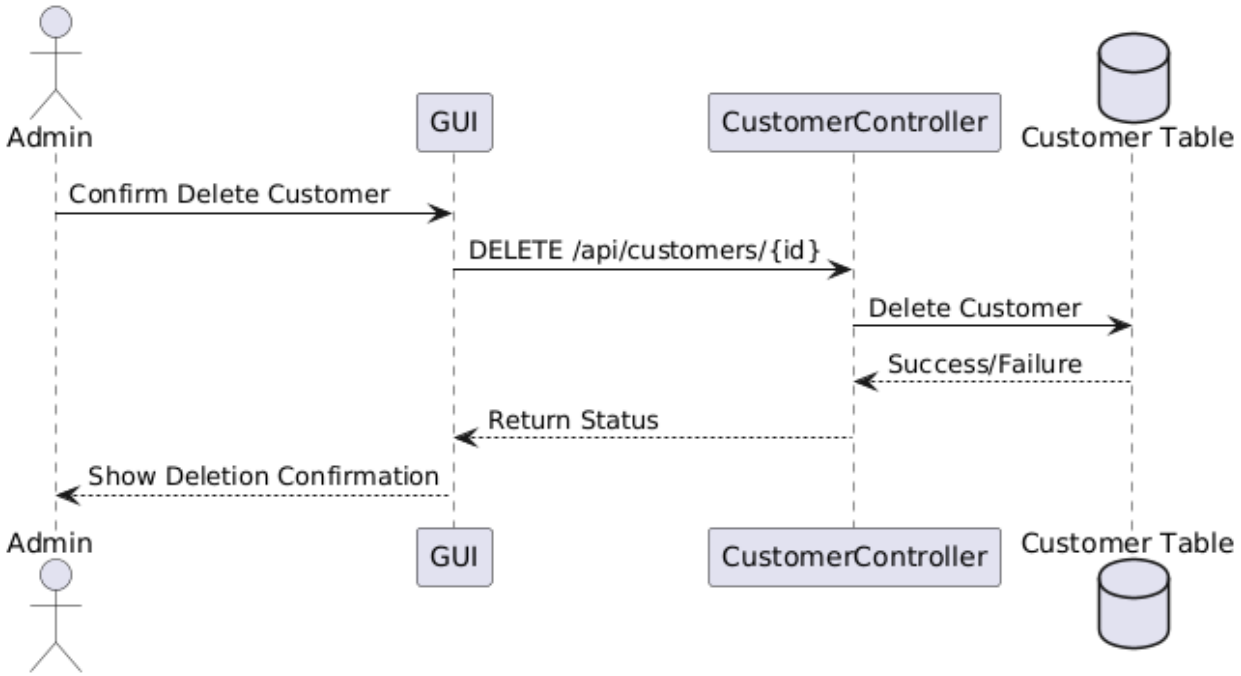



*Figure 36*

19. View user

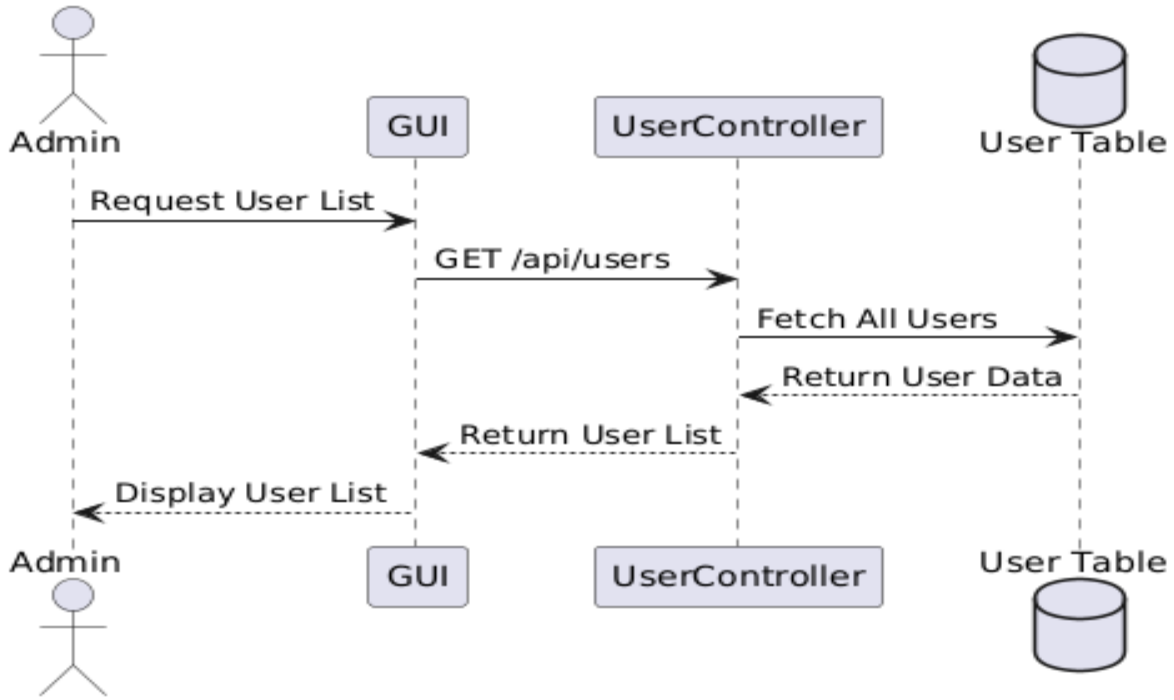



*Figure 37*

20. Add user

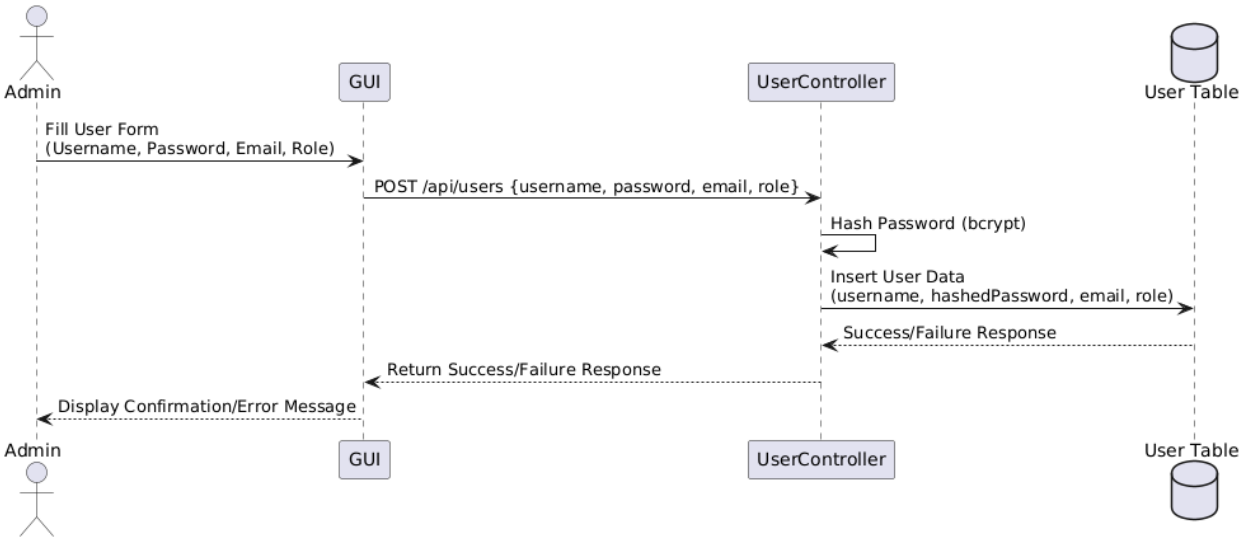



*Figure 38*

21. Edit user

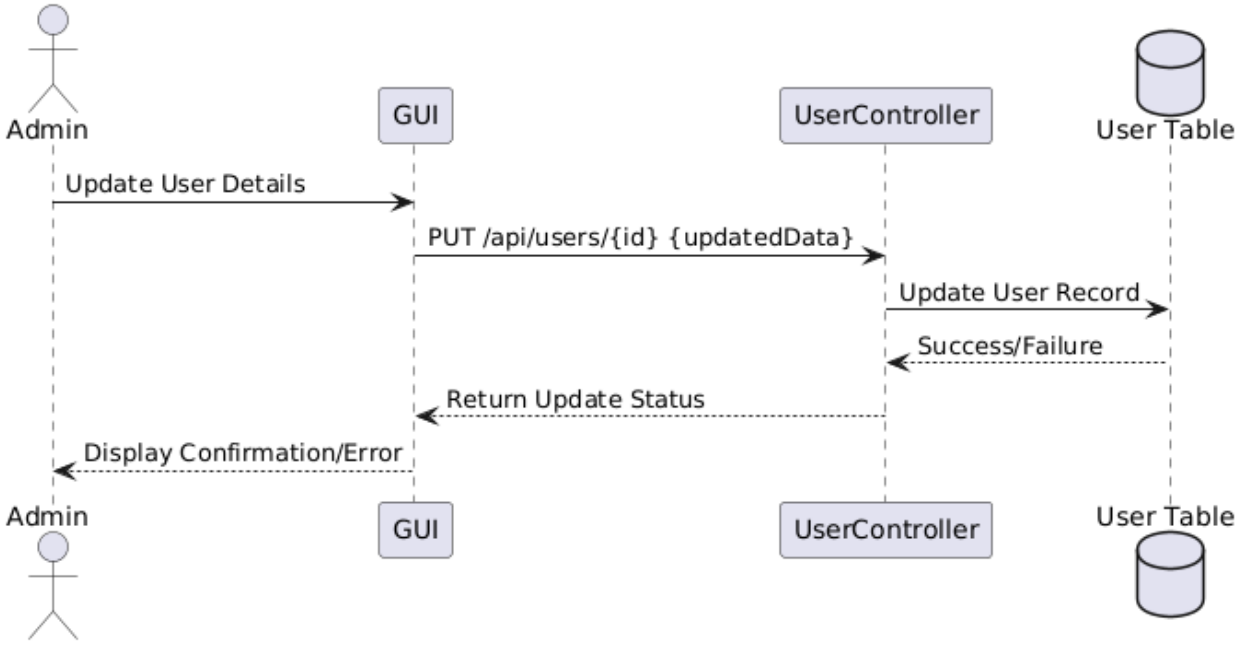



*Figure 39*

22. Delete user

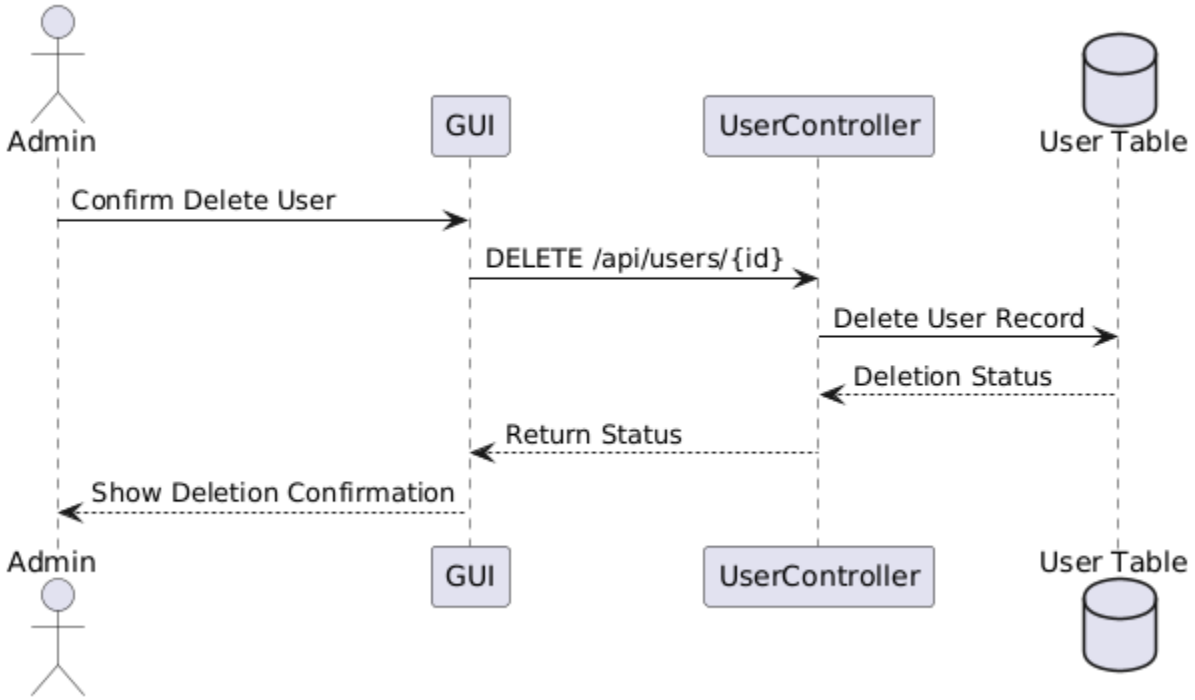



*Figure 40*

b. Pharmacist

1. Login

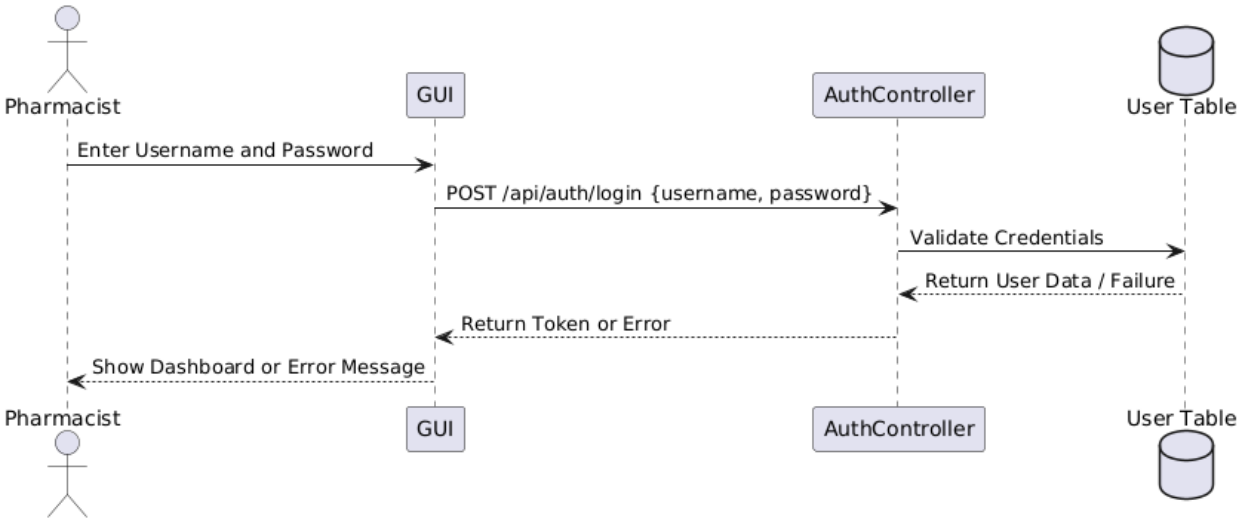



*Figure 41*

2. Edit profile

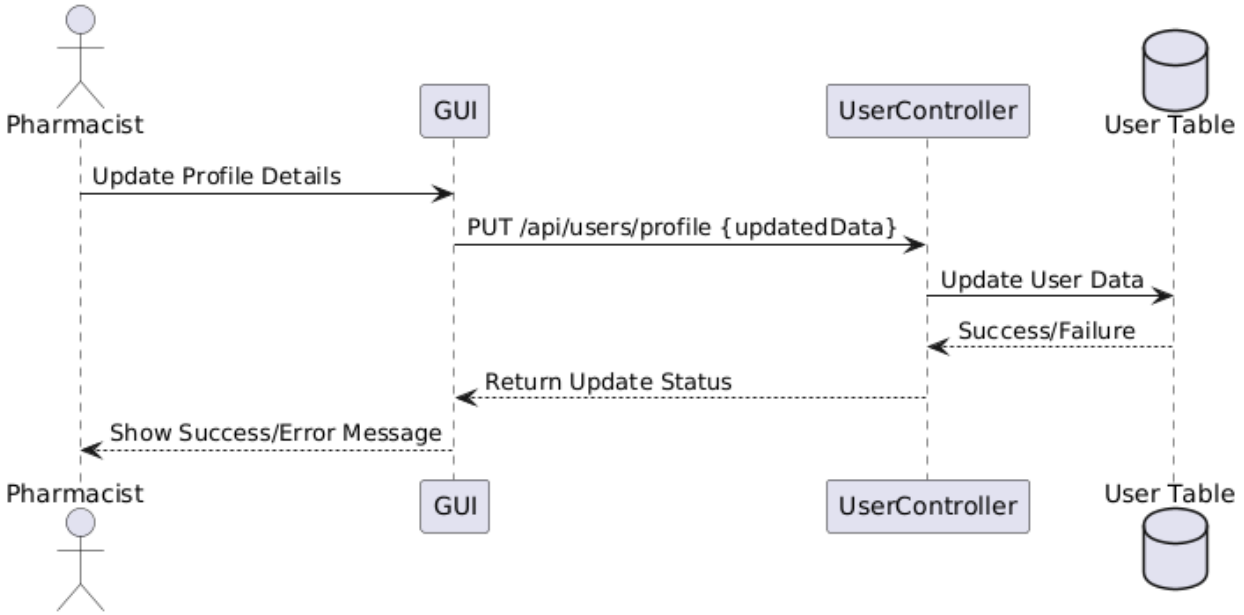



*Figure 42*

3. Search medicine

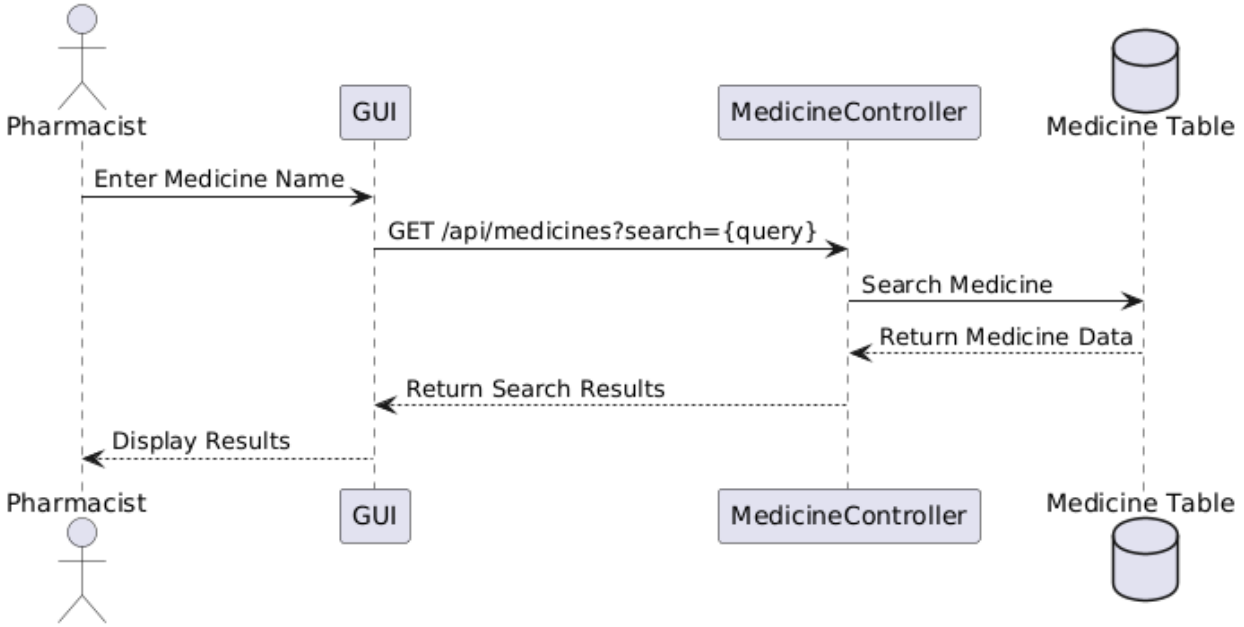



*Figure 43*

4. Add medicine

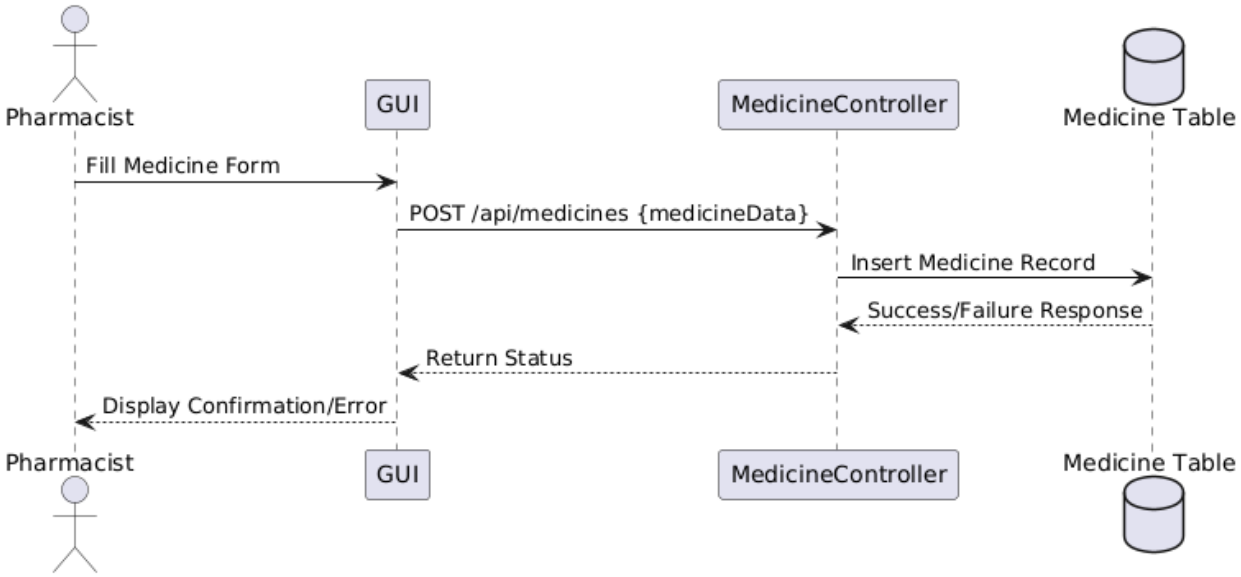



*Figure 44*

5. Edit medicine

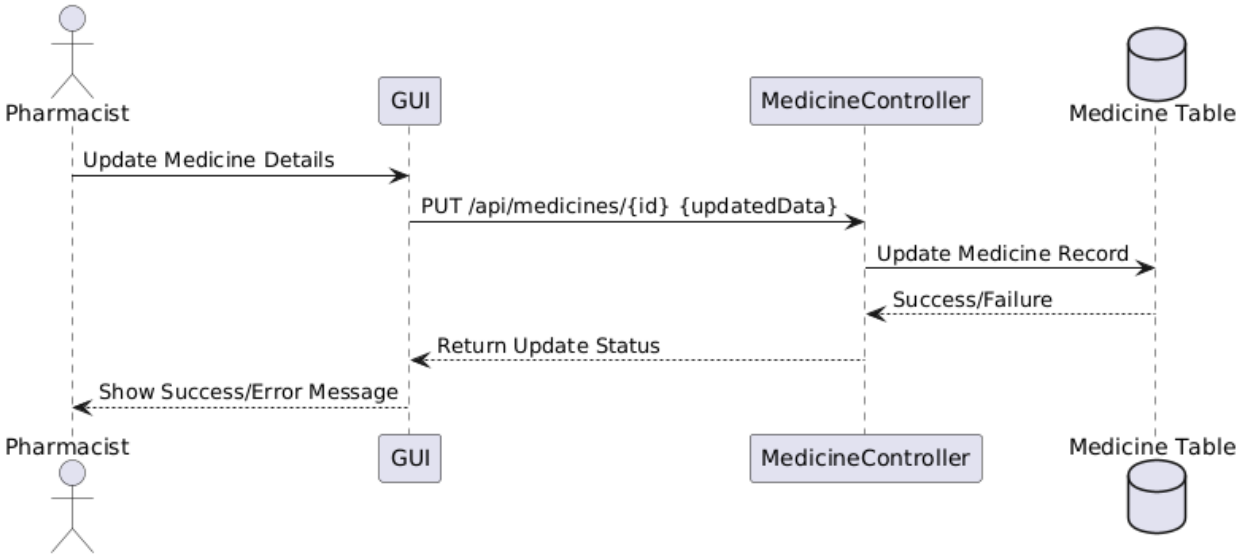



*Figure 45*

6. Delete medicine

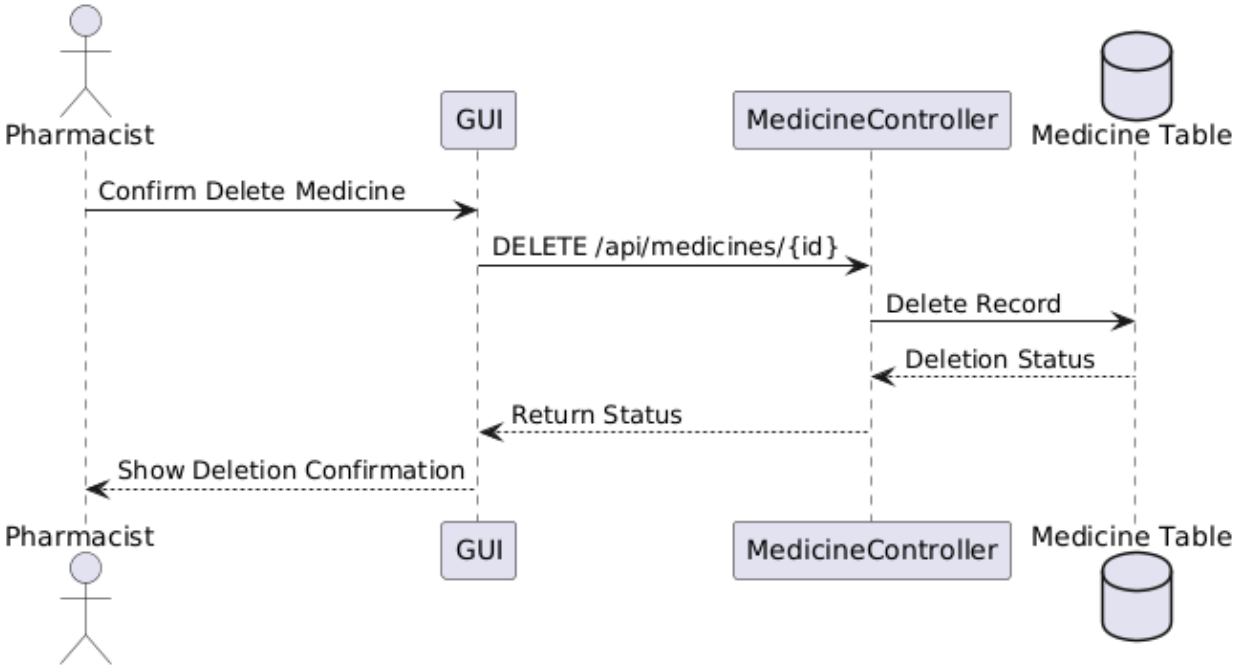



*Figure 46*

7. Edit category

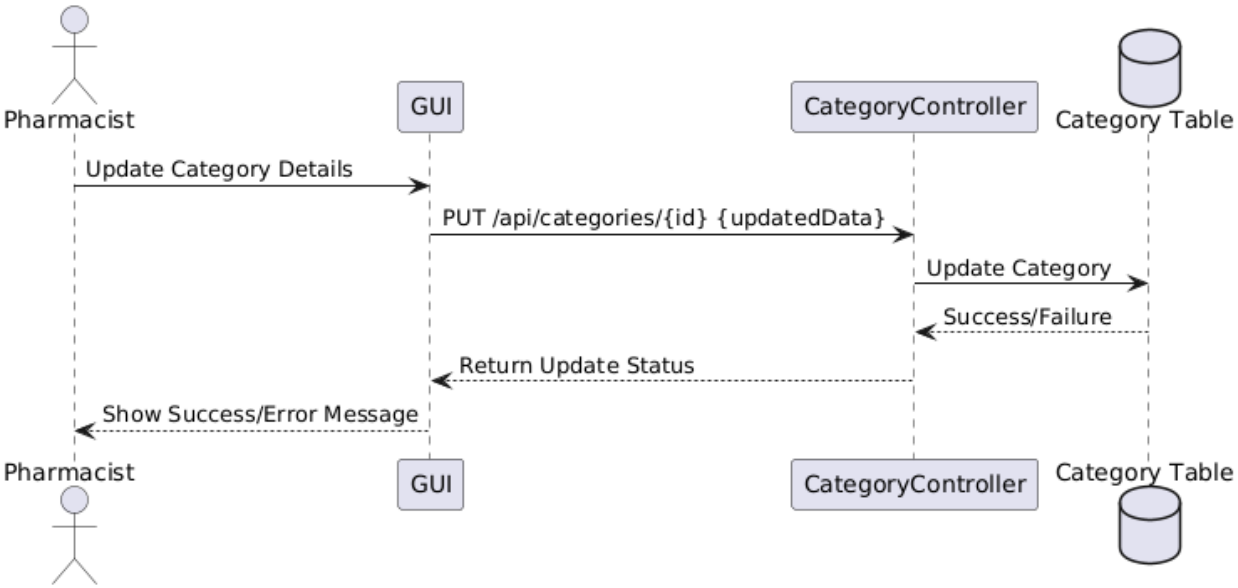



*Figure 47*

8. Add supplier

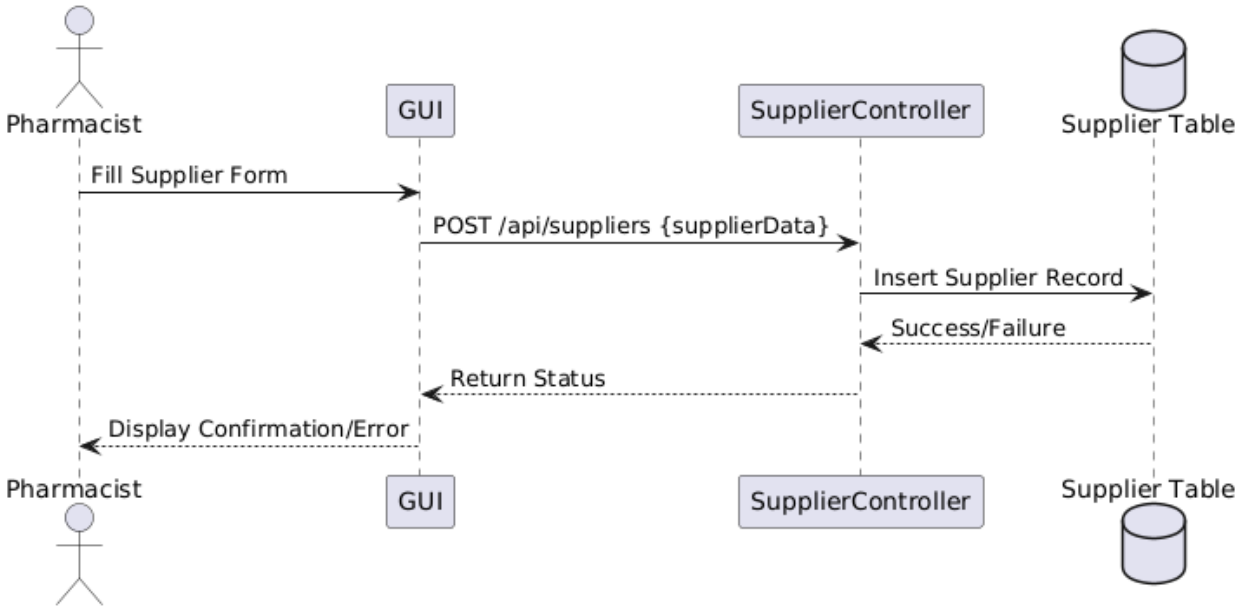



*Figure 48*

9. Edit supplier

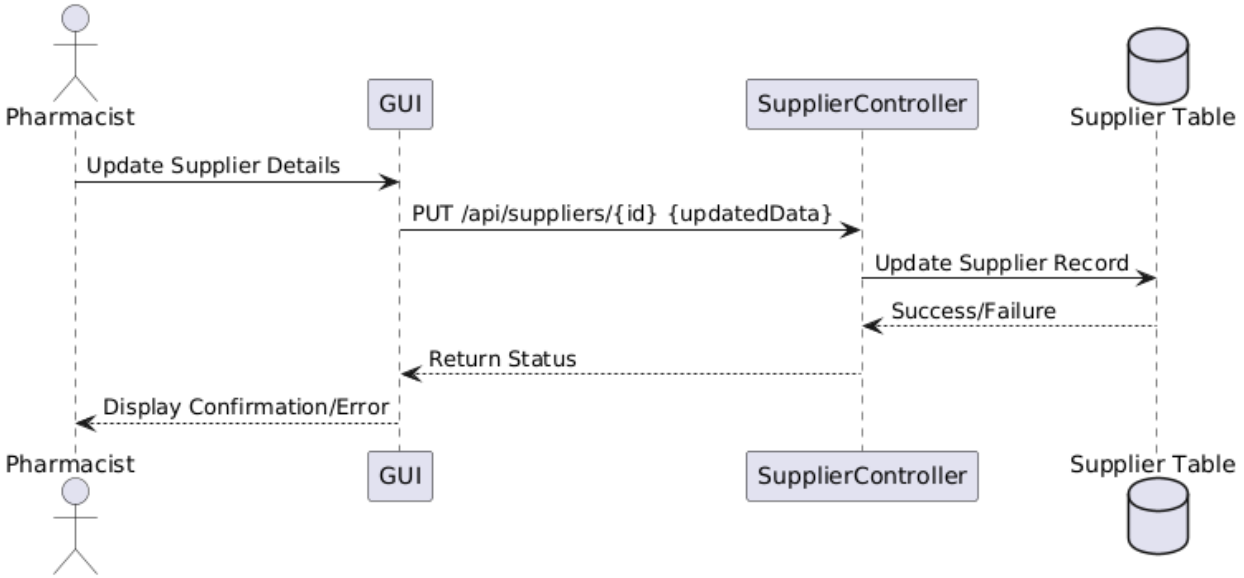



*Figure 49*

10. Delete supplier

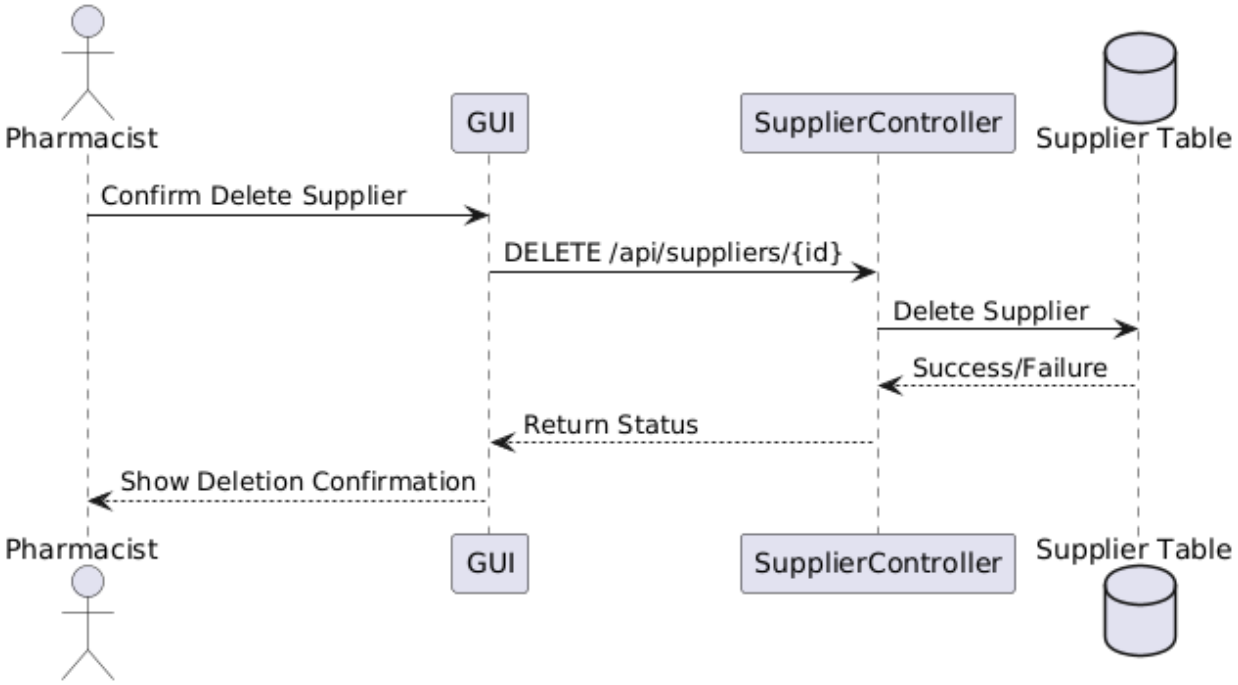



*Figure 50*

11. Search invoice

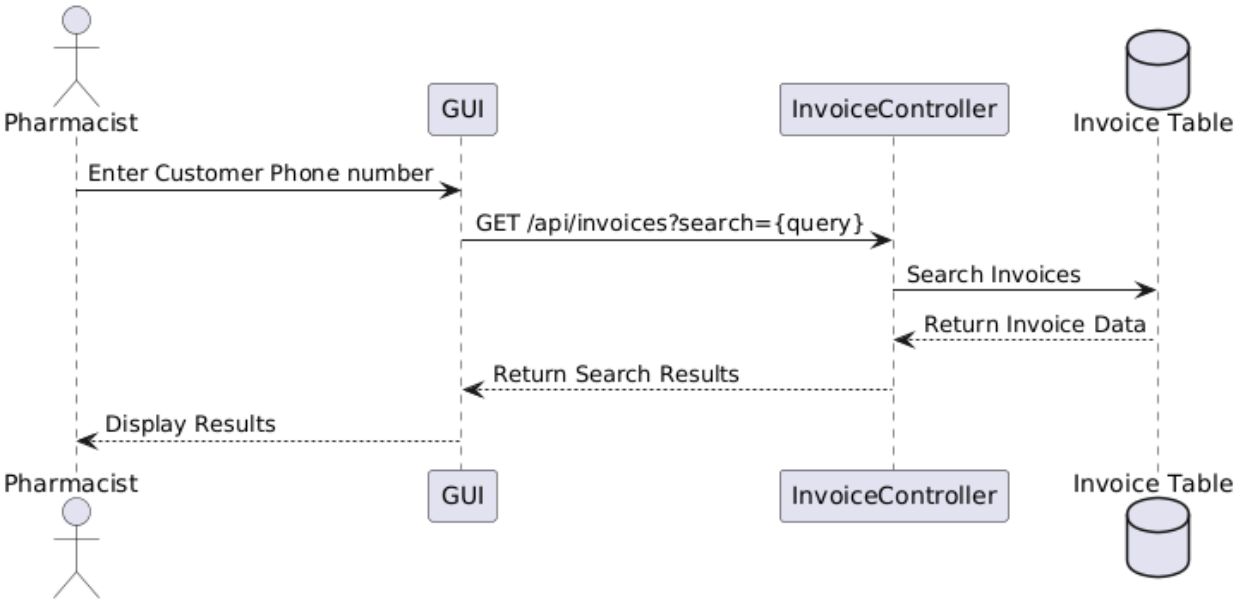



*Figure 51*

12. Add invoice

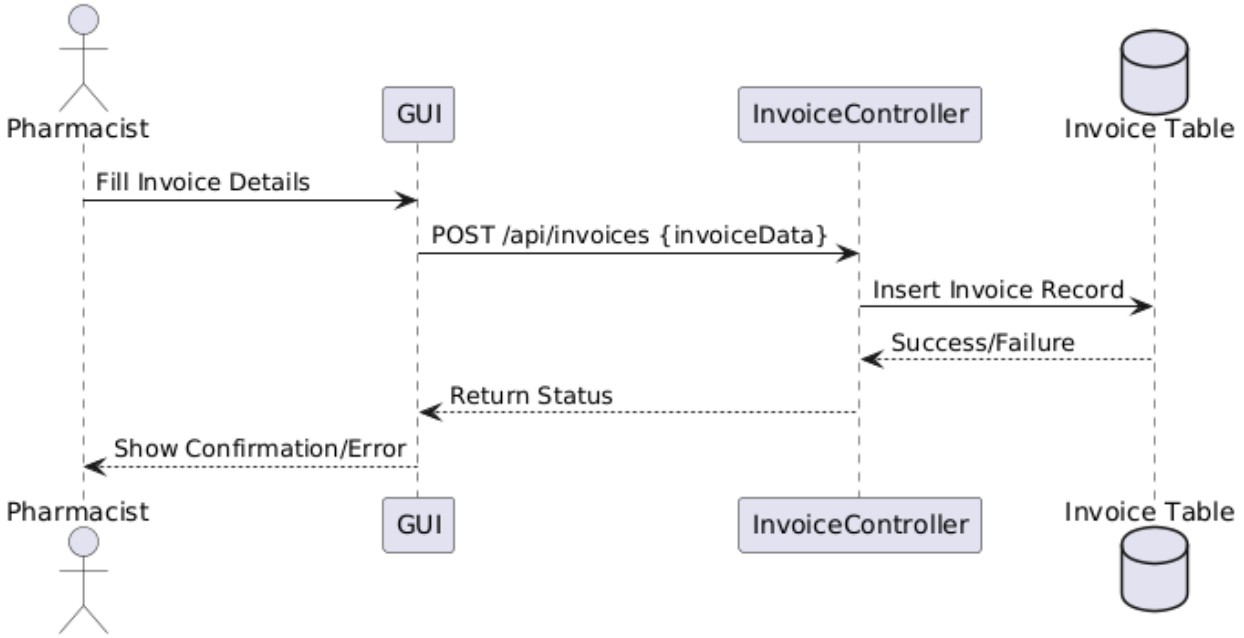



*Figure 52*

13. Edit invoice

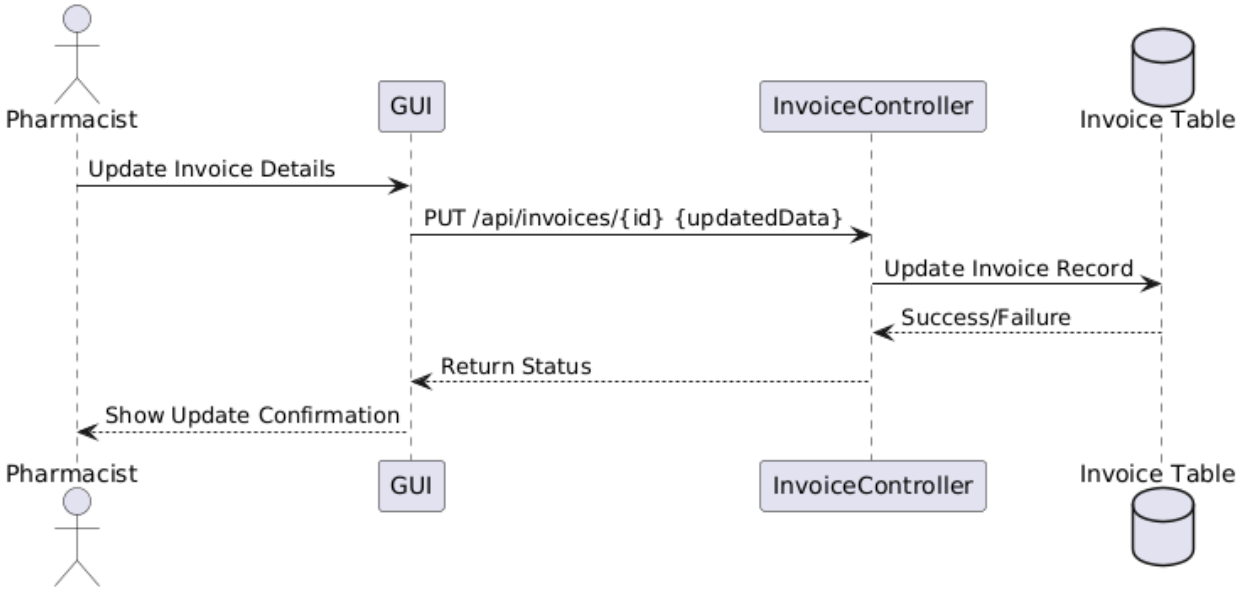



*Figure 53*

14. Delete invoice

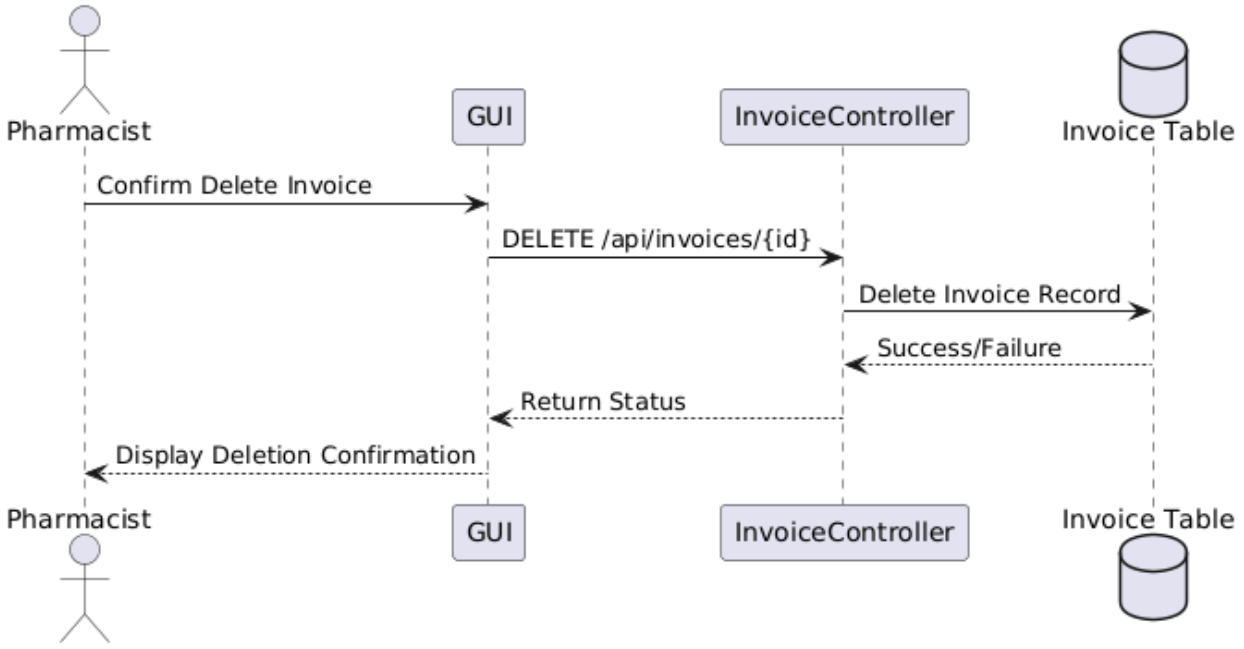



*Figure 54*

15. Search Customer

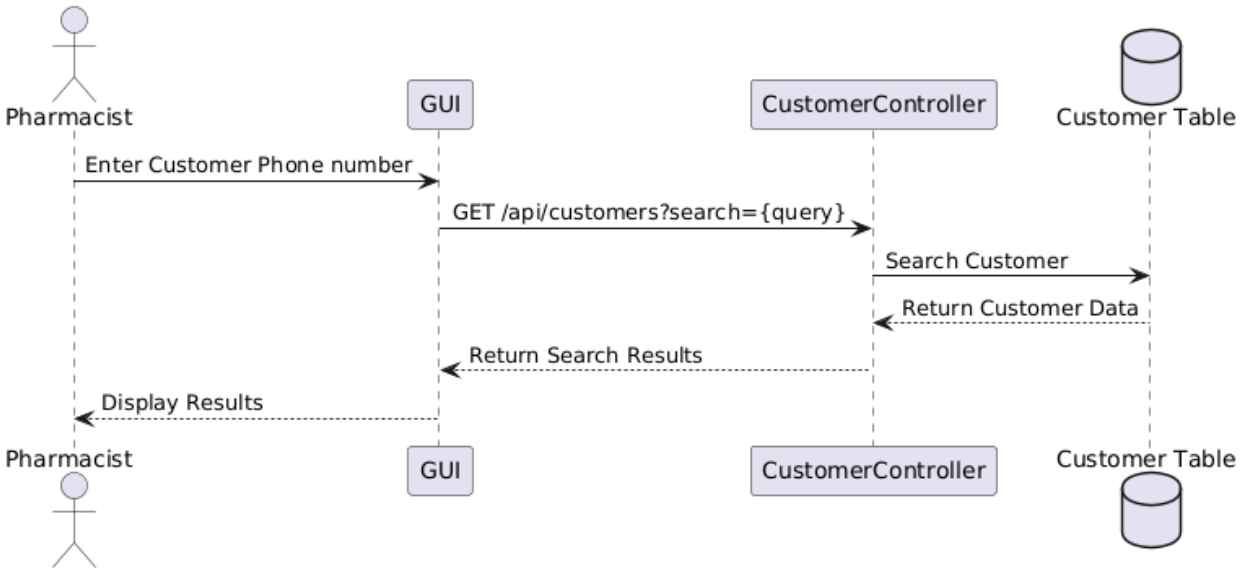



*Figure 55*

16. Add customer

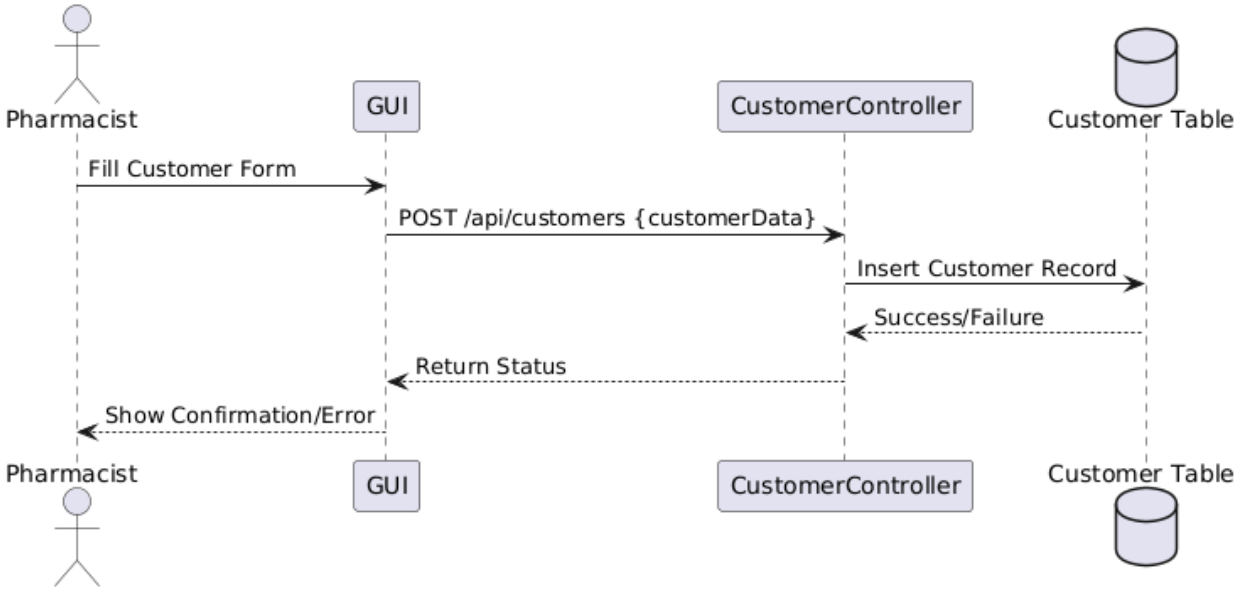



*Figure 56*

17. Edit customer

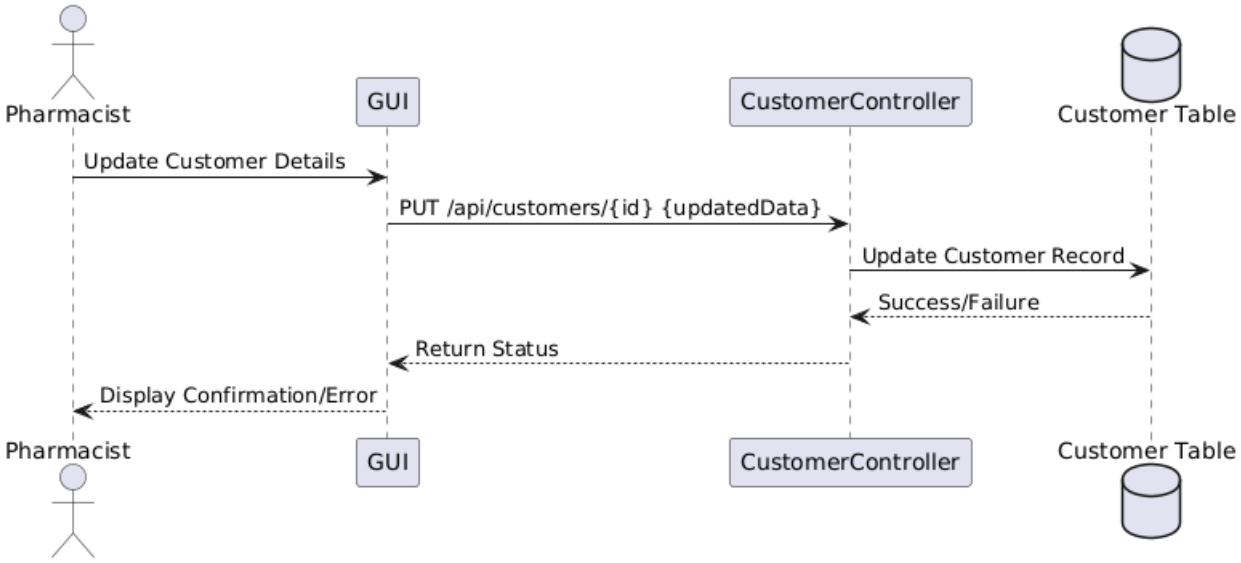



*Figure 57*

18. Delete customer

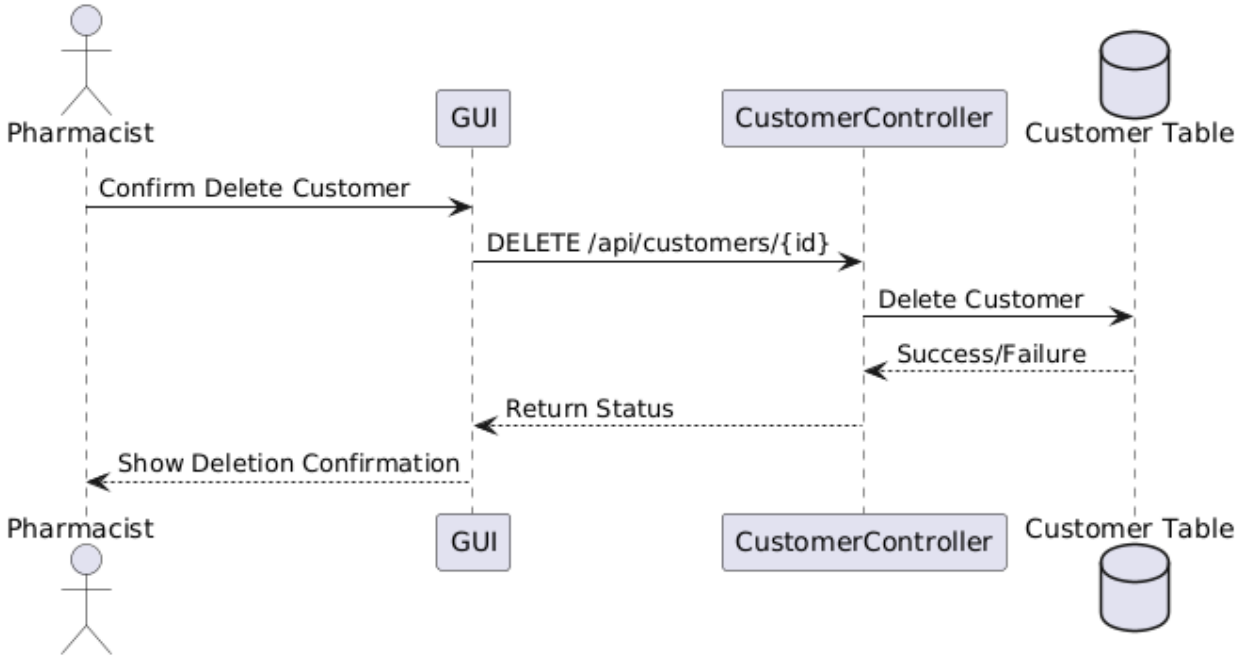



*Figure 58*

# 3 IMPLEMENTATION

## 3.1 USER'S ACCOUNT MANAGEMENT FUNCTIONS

### 3.1.1 Admin/ Pharmacist Login:

Step 1: Go to the Pharmacist website via the link

Step 2: Fill the username, password (If you are an admin, username: "admin" password:"1234", if you are a pharmacist, username in this case is "pharmacist", password: 123).

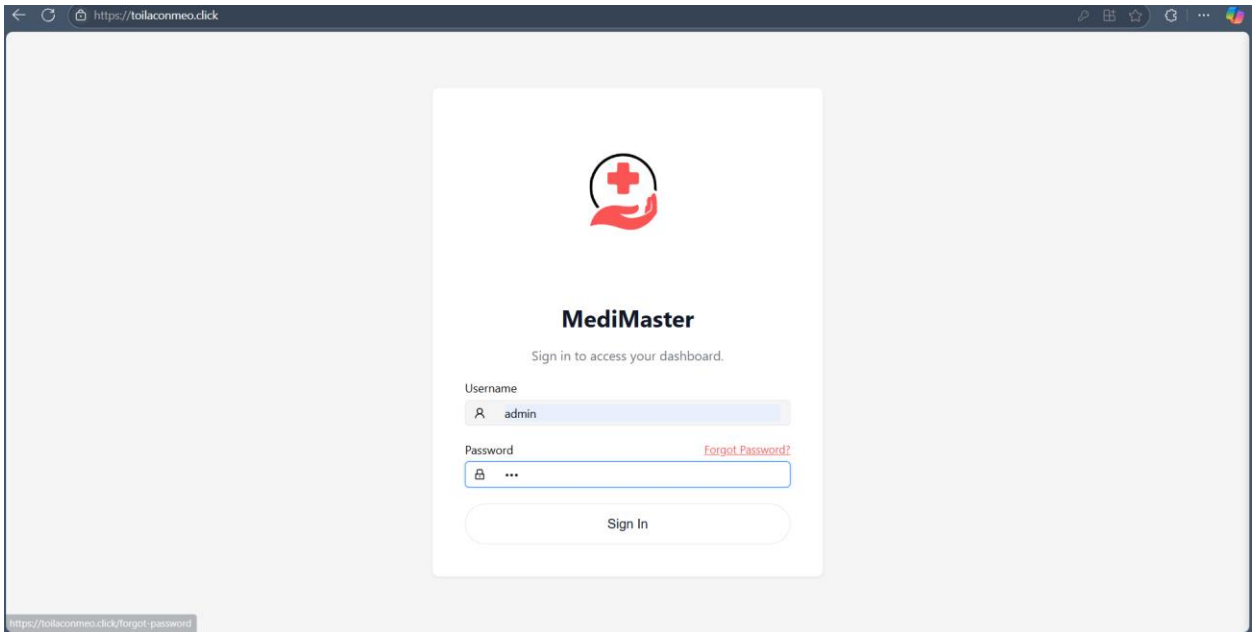



*Figure 59*

Pharmacist:

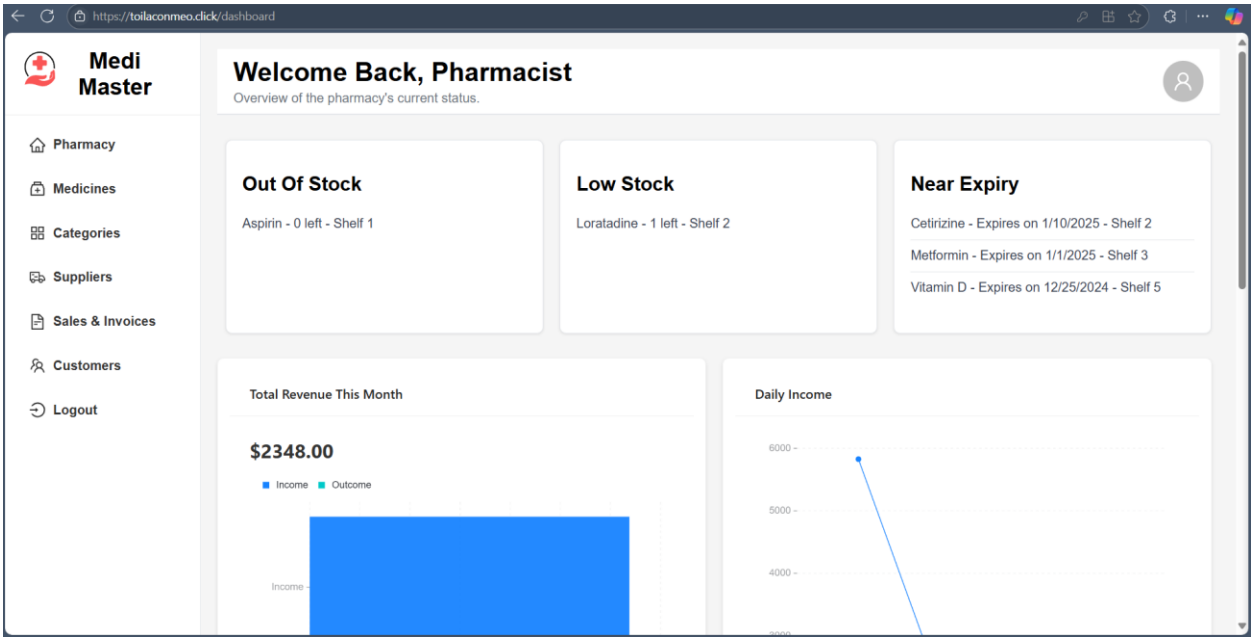



*Figure 60*

Admin:

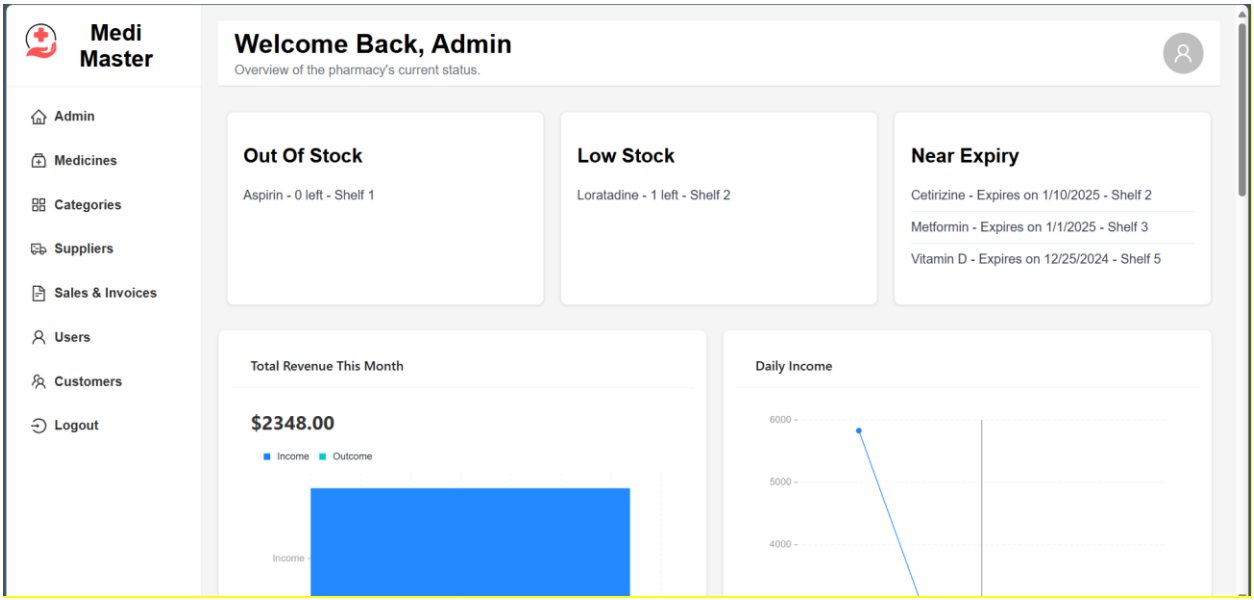



*Figure 61*

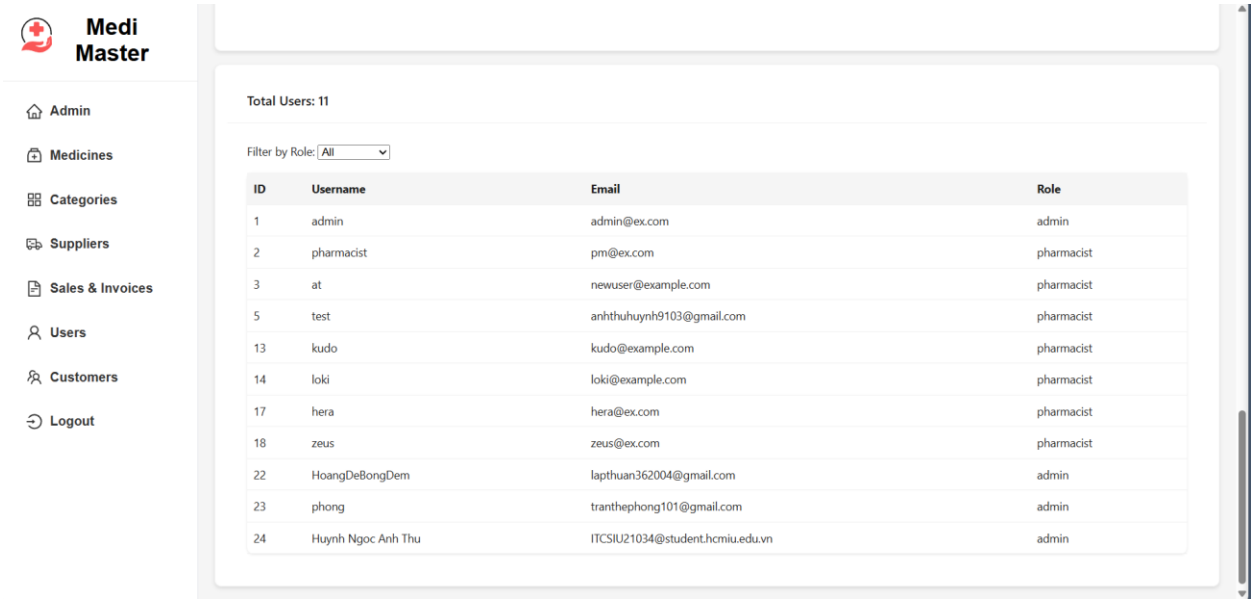



*Figure 62*

As we can see, the dashboard of the Admin is different from the Pharmacist, the Admin can view, edit, and delete accounts as a pharmacist.

The rest of the features, admin and pharmacist are the same.

### 3.1.2 Edit Profile

Step 1: Click on the profile figure at the top – right corner.

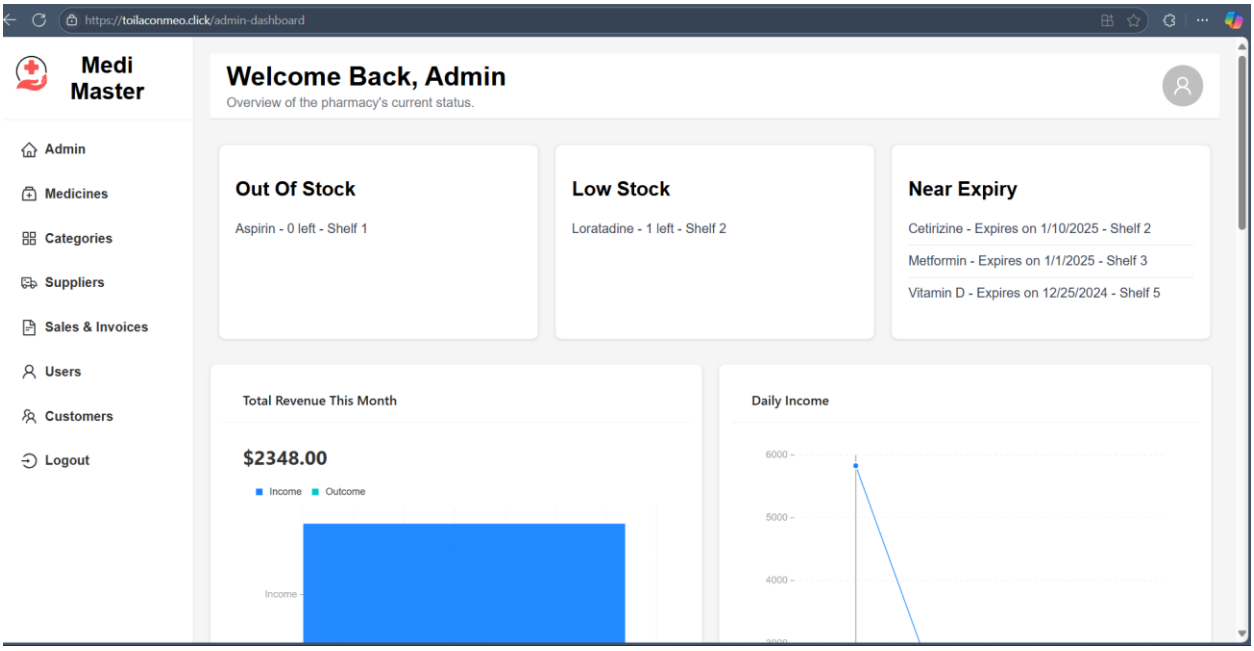



*Figure 63*

Step 2: Update the information, that we want to change, in this case, we will change the pass world and Fullname:

Step 3: click on change savechange and password respectively.

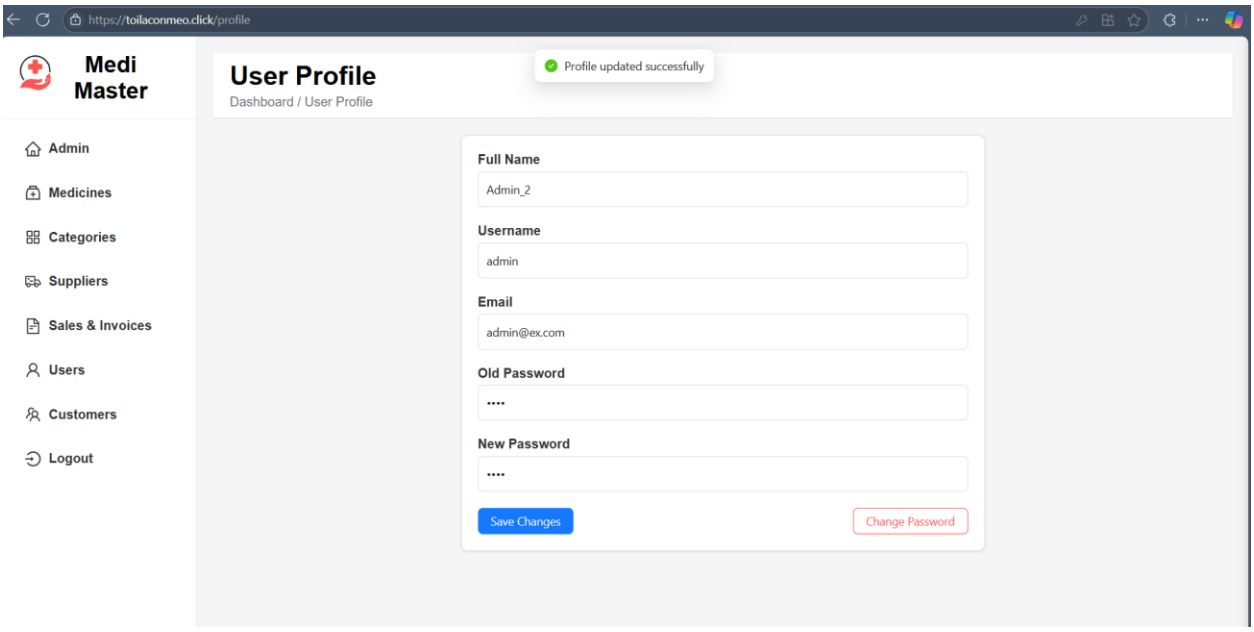



*Figure 64*

The success message will pop up.

### 3.1.3 Forgot Password

Step 1: Click on forgot password at the login screen.

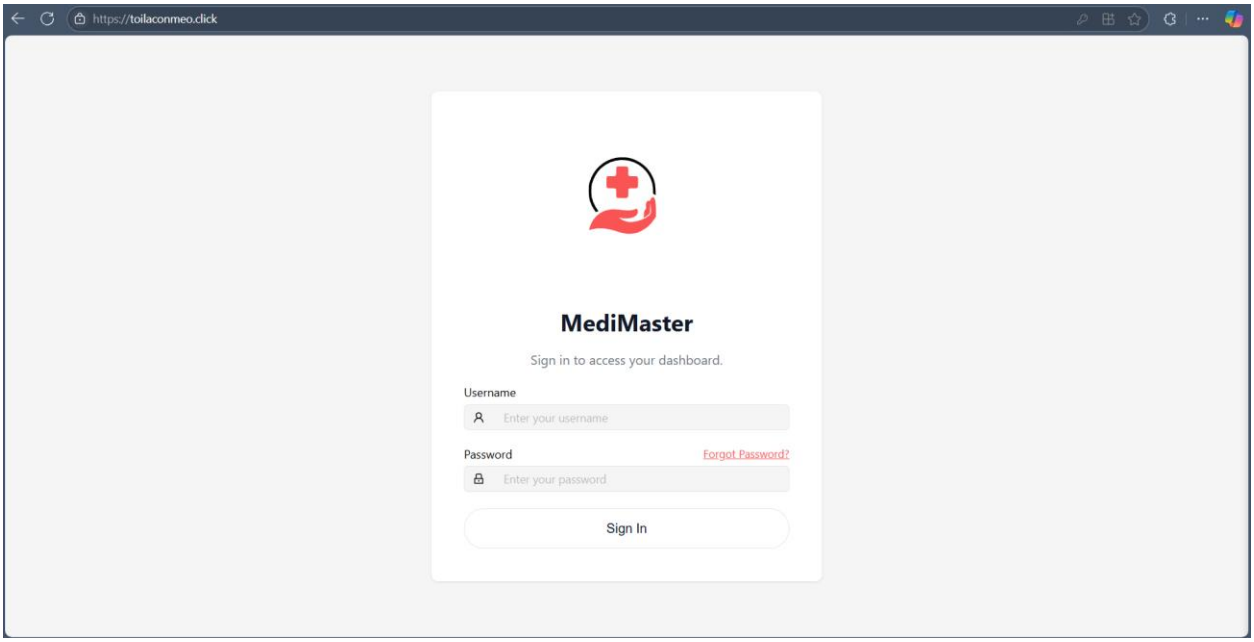



*Figure 65*

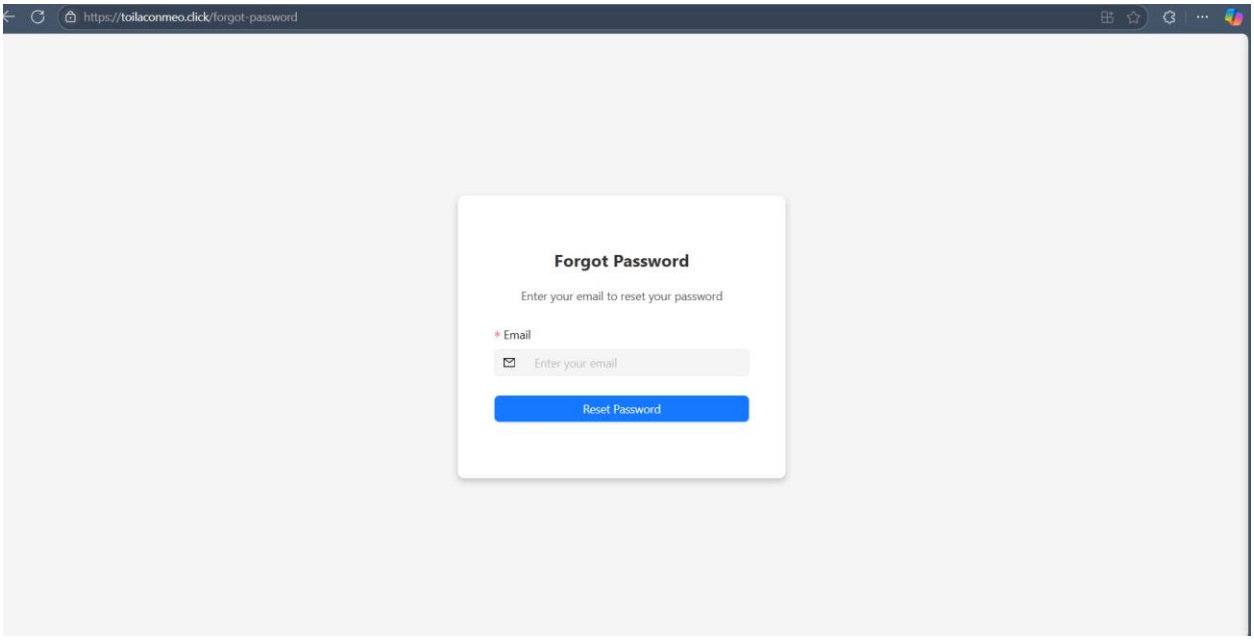



*Figure 66*

Step 2: Fill in the blanks with your email, the system will search the database if the sent email exists, if it does, the system will send a new random password to your email.

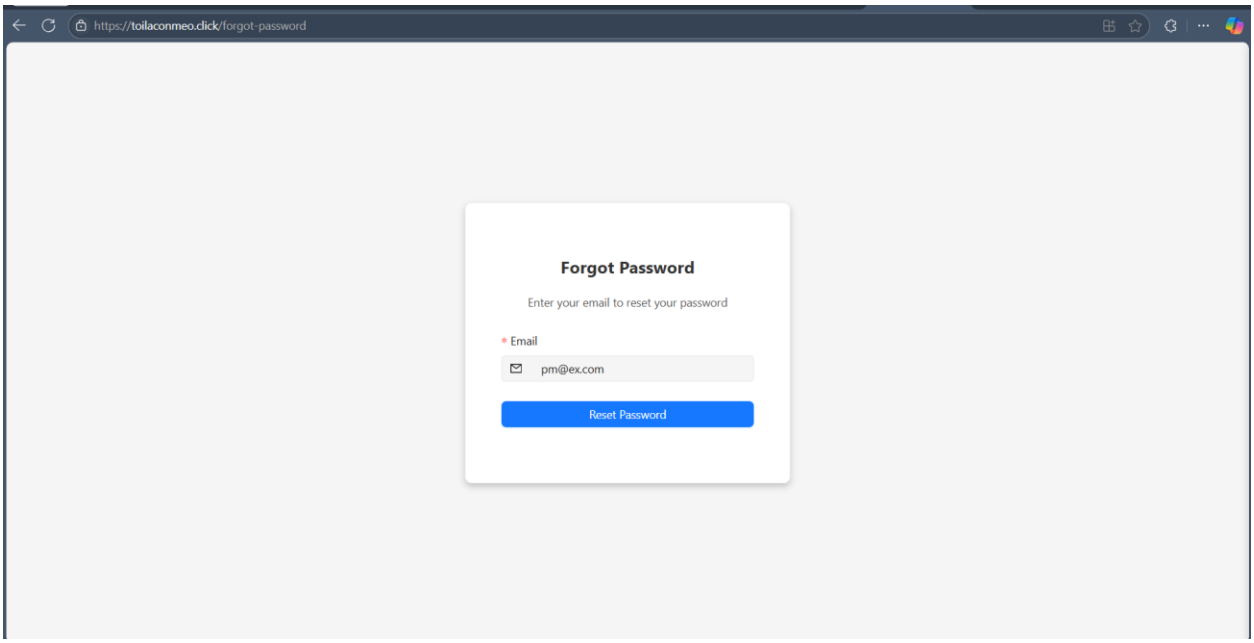



*Figure 67*

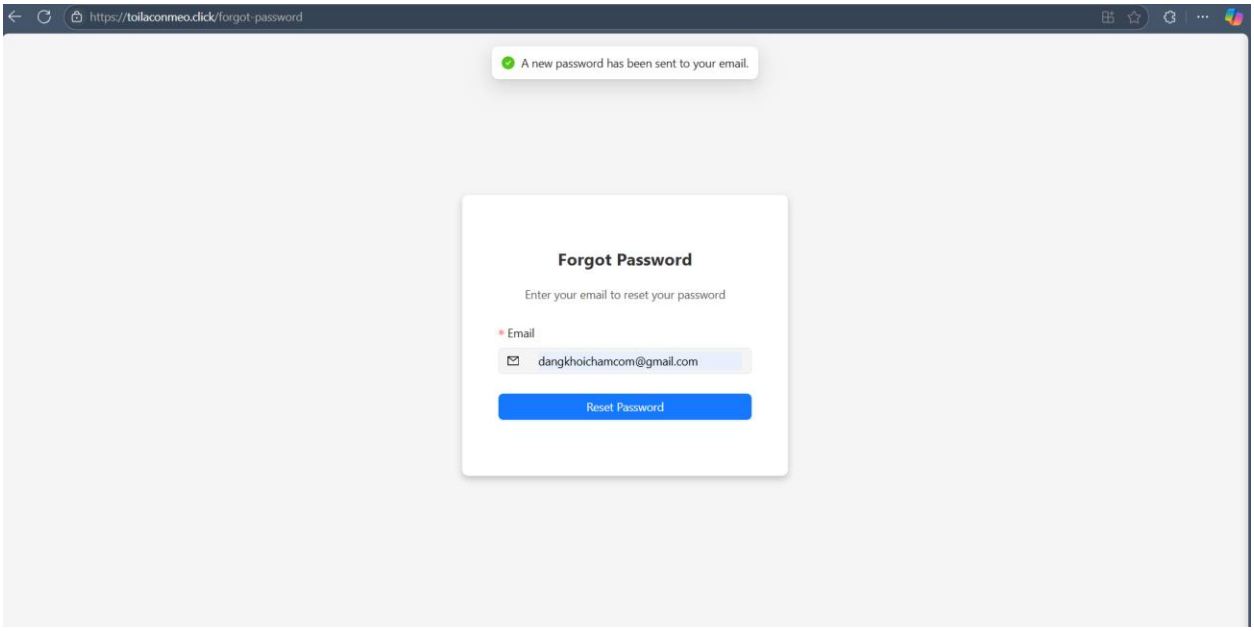
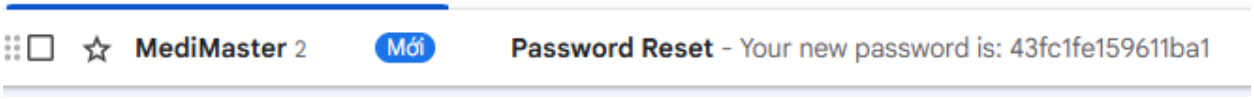



*Figure 68*



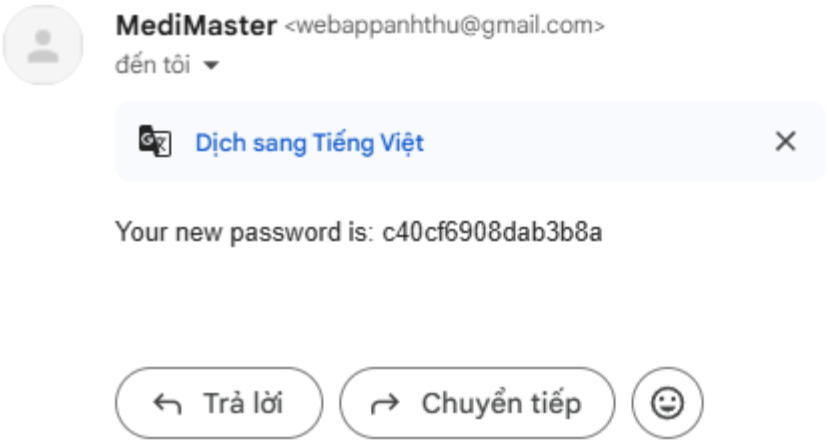



*Figure 69*

If user not found in the database:

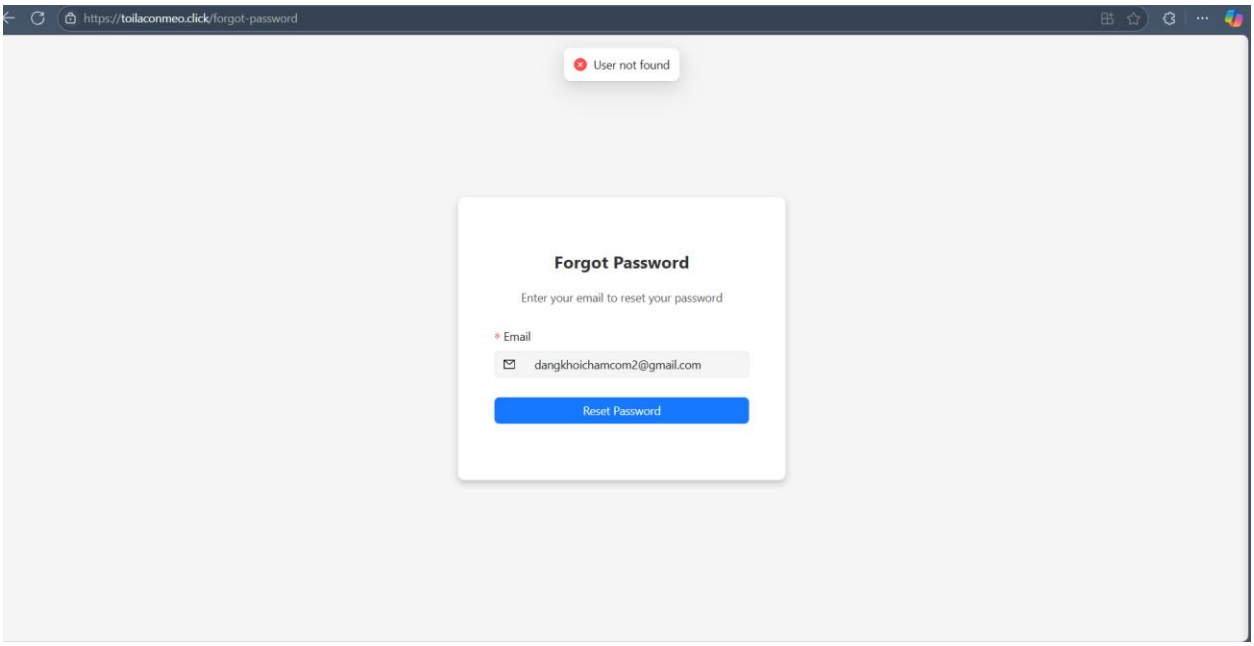



*Figure 70*

### 3.1.4 View User

Step 1: Login with the “admin” role (“admin”,1234)

Step 2: Click on Admin button to navigate to Admin Dashboard

Step 3: Scroll down and view the User

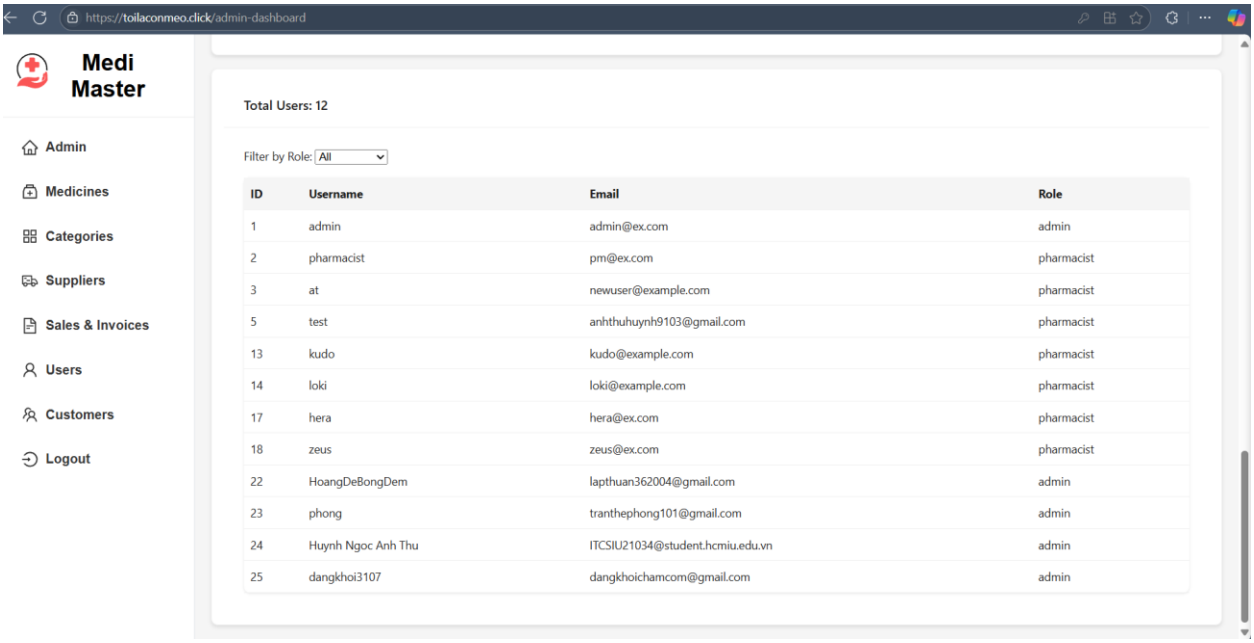



*Figure 71*

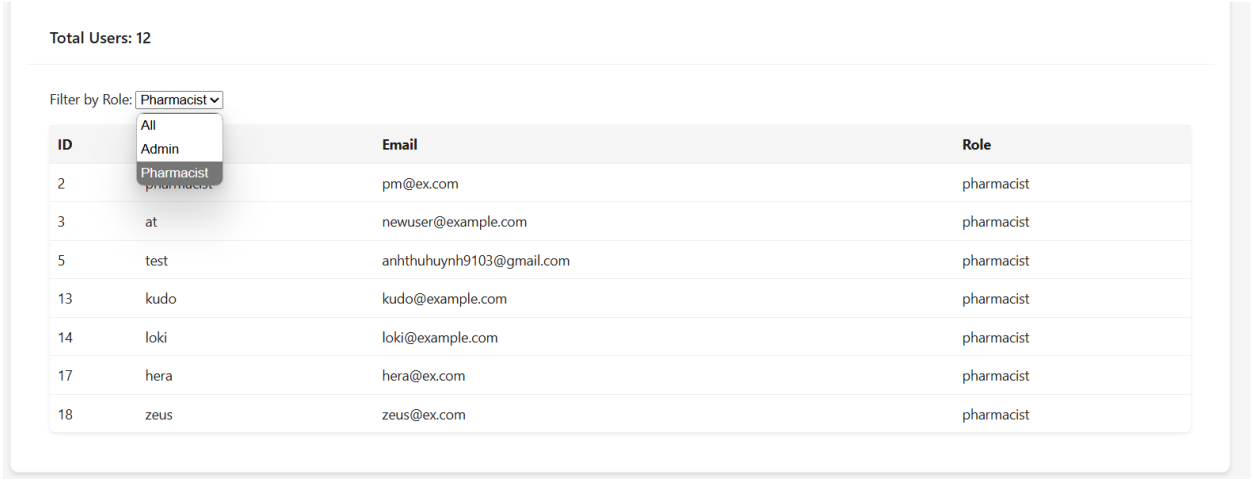



*Figure 72*

We can use filter for each role.

### 3.1.5 Edit User

Step 1: Login with “admin” role.

Step 2: Click on “Users” button and navigate to User Management screen

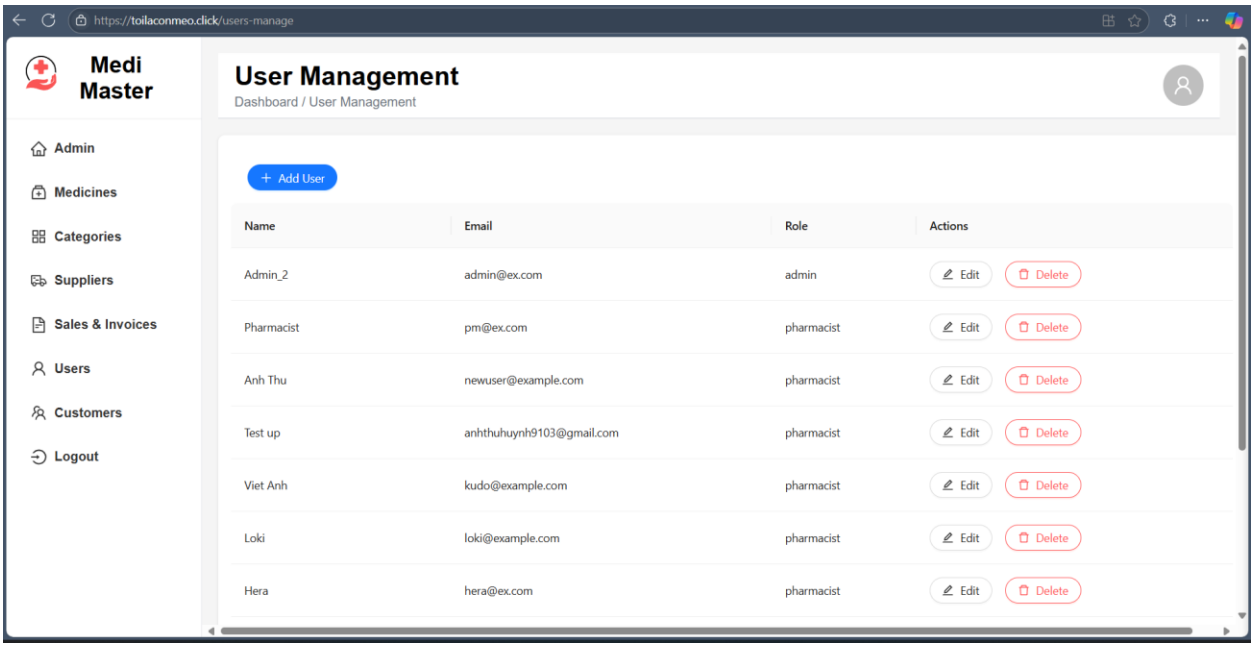



*Figure 73*

Step 3: Click on Edit and change the informations, role, and email address

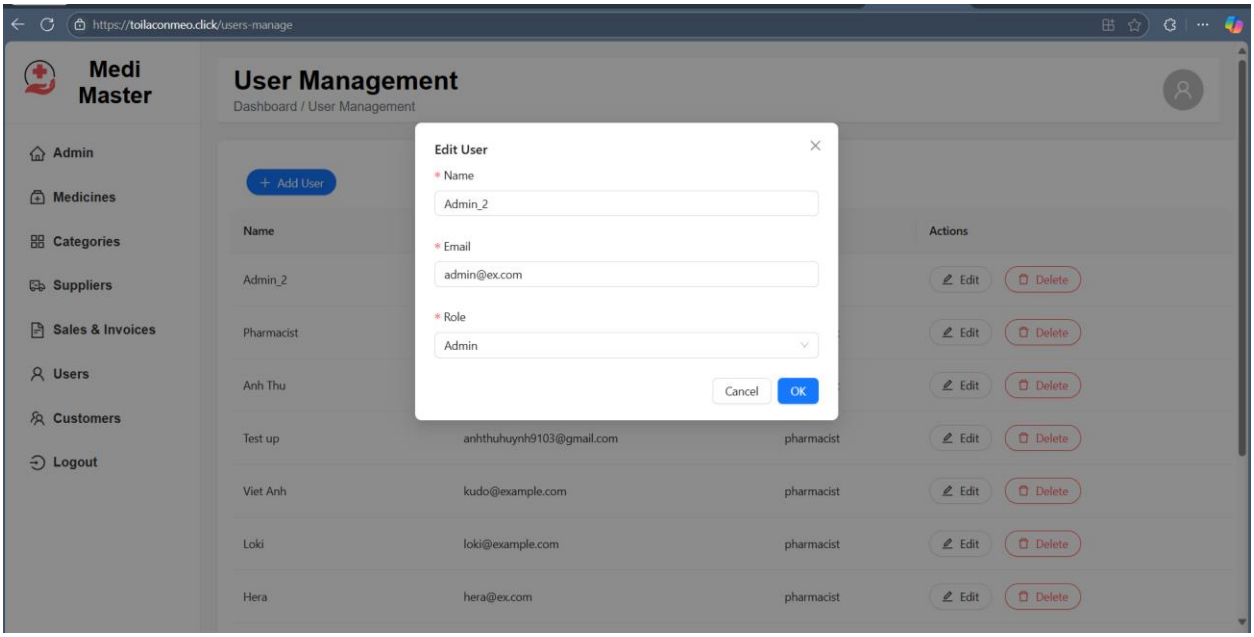



*Figure 74*

Step 4: Click on “OK”

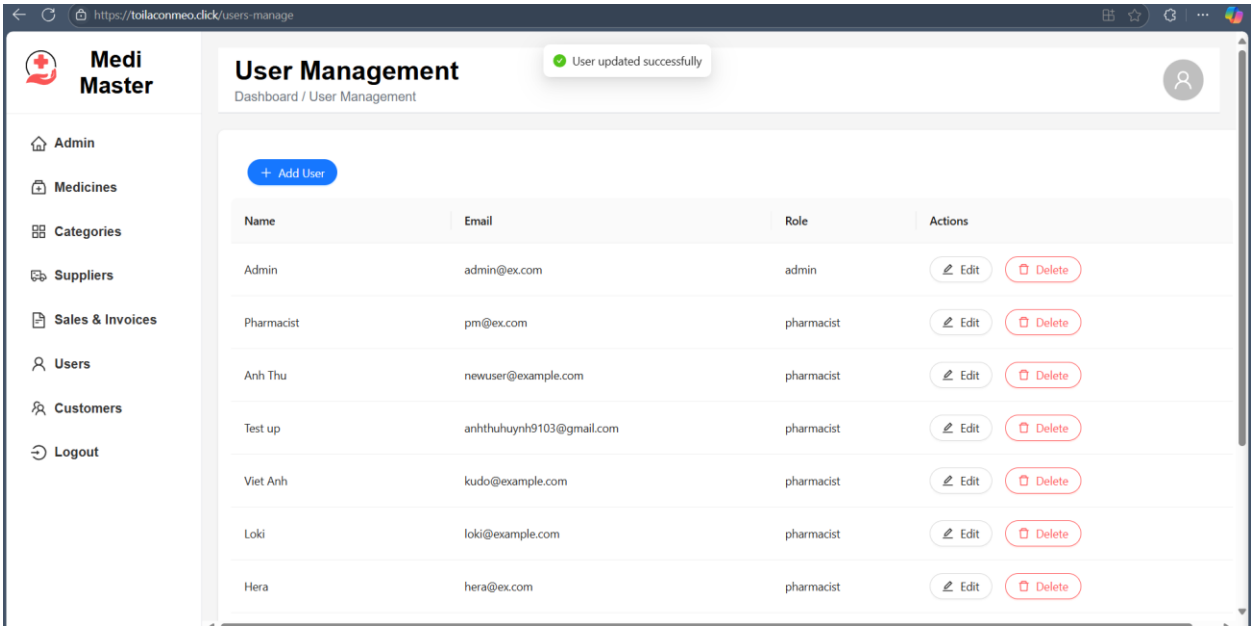



*Figure 75*

### 3.1.6 Edit User

Step 1: At the User Management Screen with admin's role

Step 2: Click on "Add User" and open the form.

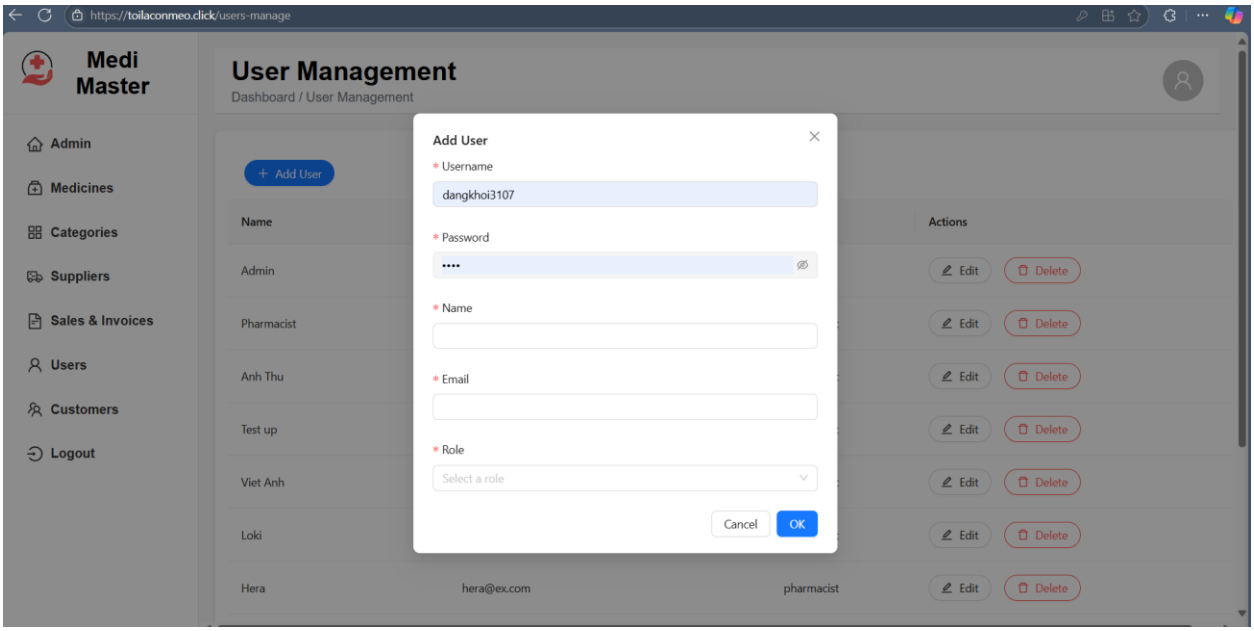



*Figure 76*

Step 3: Fill the information then click on "OK"

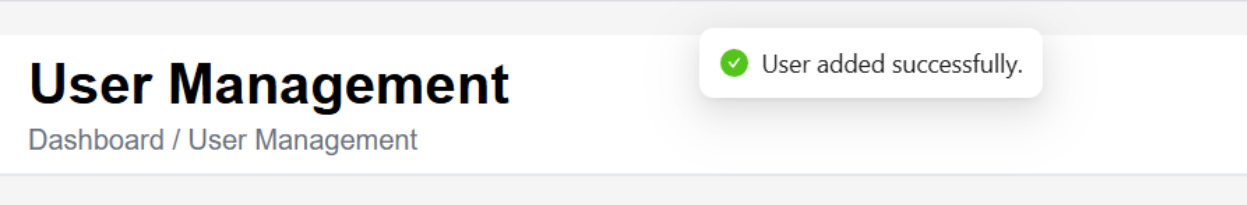



*Figure 77*

### 3.1.7 Delete User

Step 1: At the User Management screen with admin's role

Step 2: Click on "Delete User"

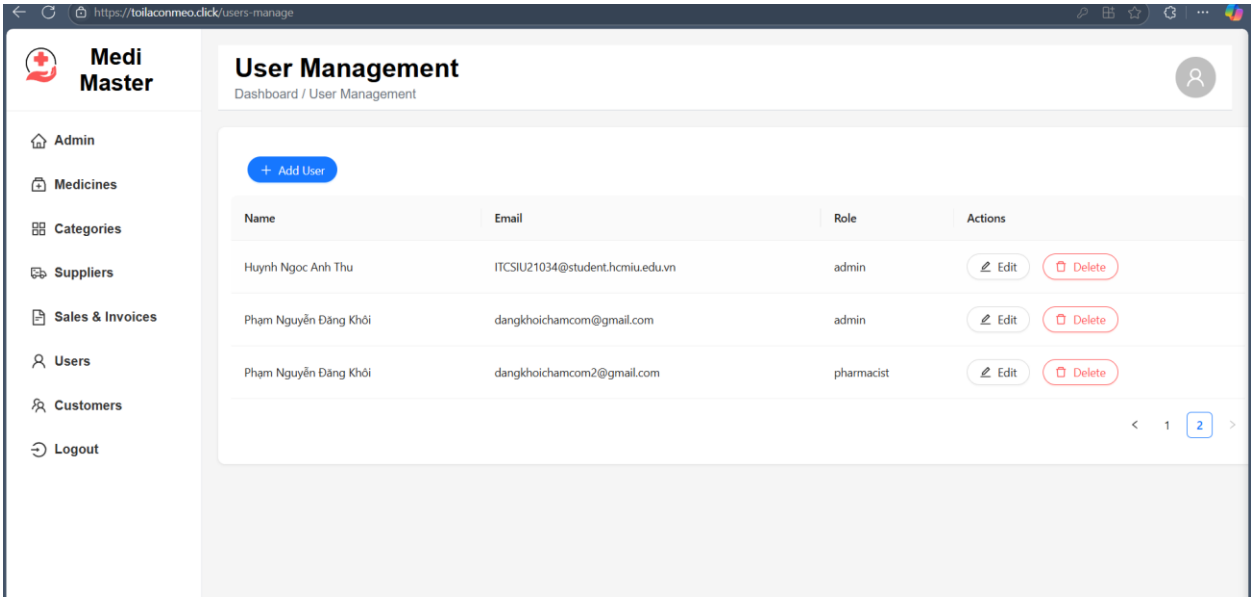



*Figure 78*

Step 3: Click "OK":

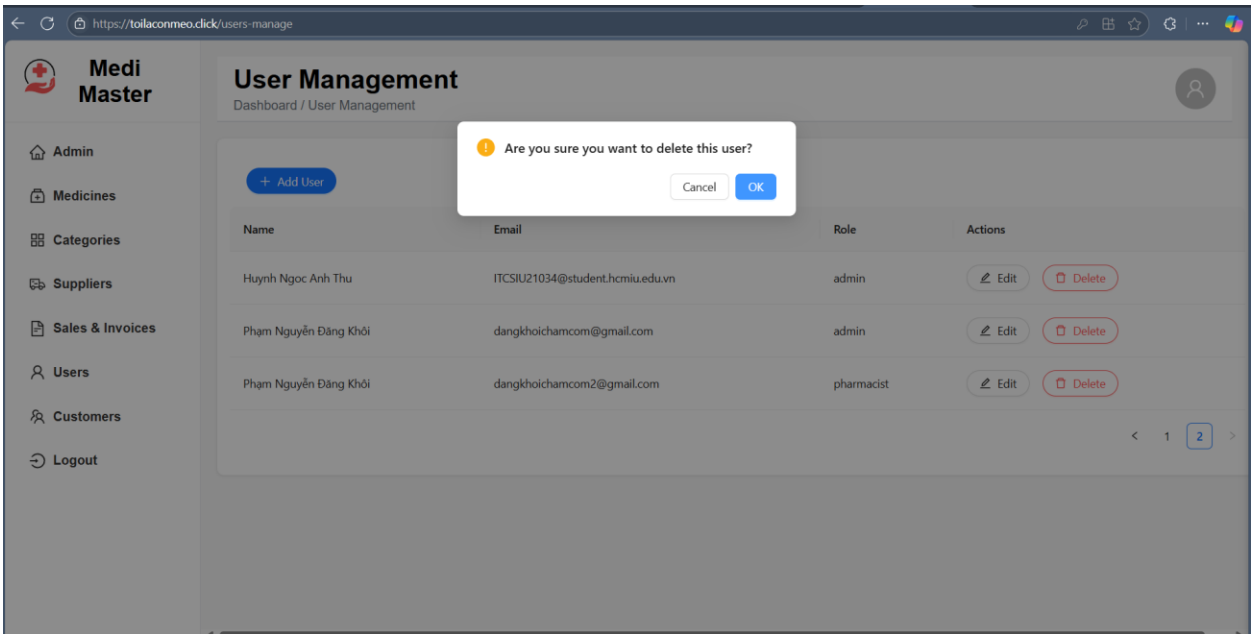



*Figure 79*

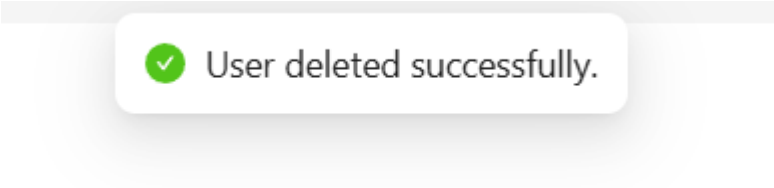



*Figure 80*

### 3.1.8 Log Out

Step 1: Already login to the system.

Step 2: Click on “Logout” button.

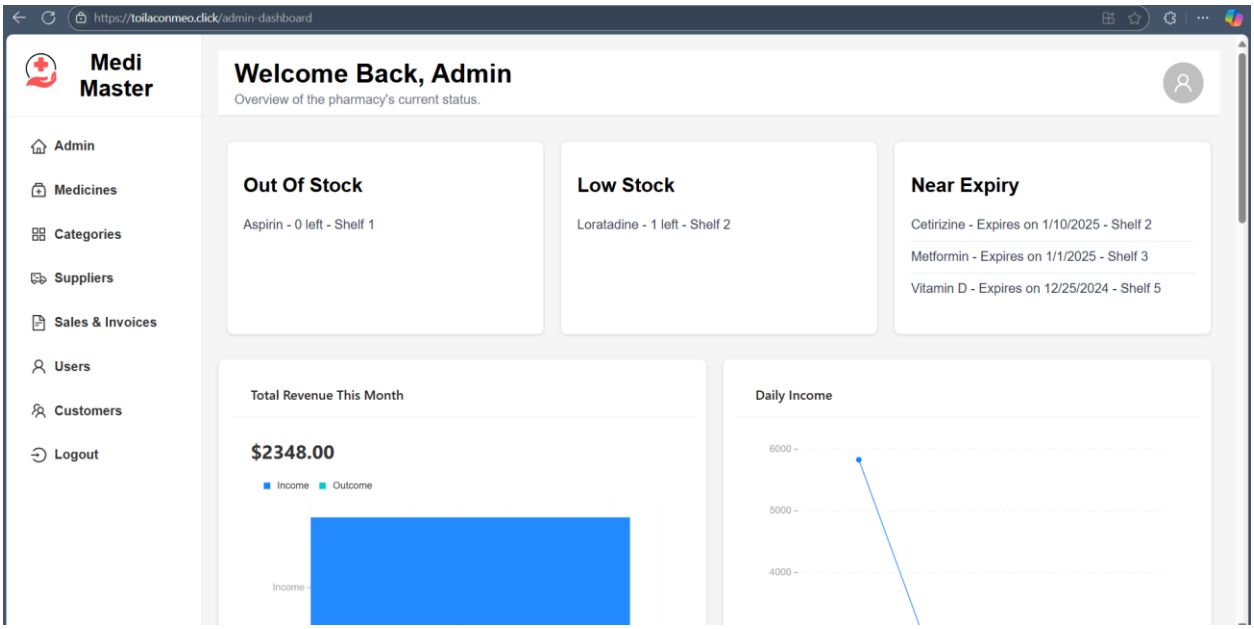



*Figure 81*

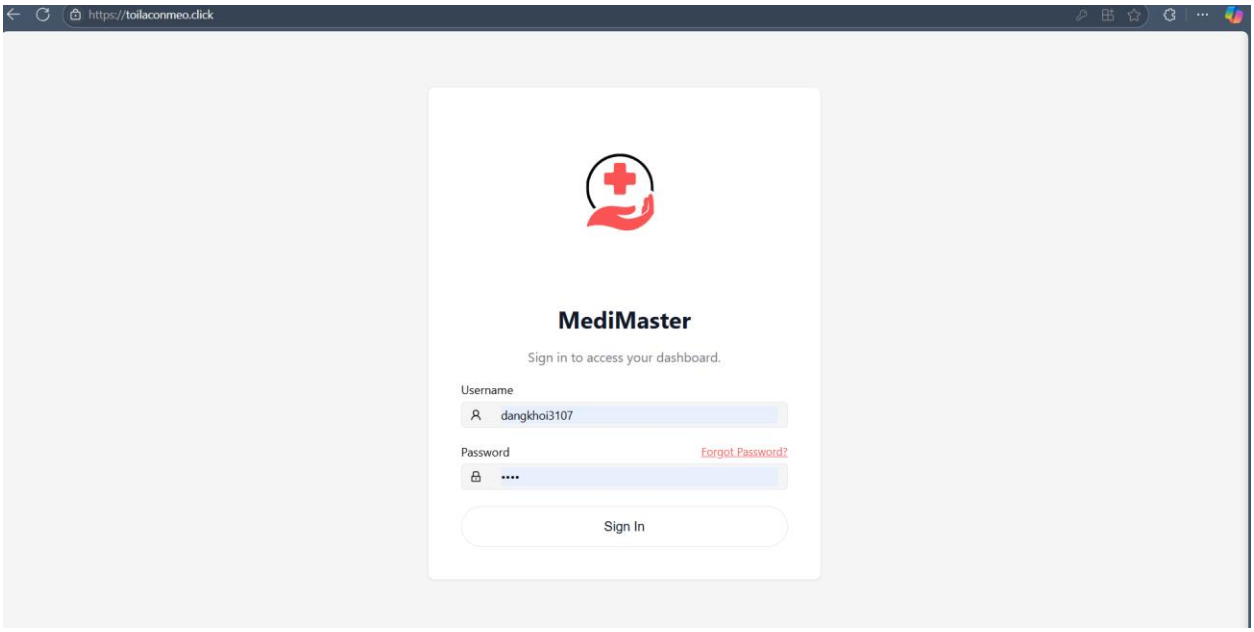



*Figure 82*

## 3.2 MEDICINCES MANAGEMENT FUNCTIONS

### 3.2.1 Add New Medicince

Step 1: Login the system with any role.

Step 2: Click on the button Medicine to navigate to Medicines Management screen.

Step 3: Click on Add Medicines

Step 4: Fill the form then click on “ADD”.

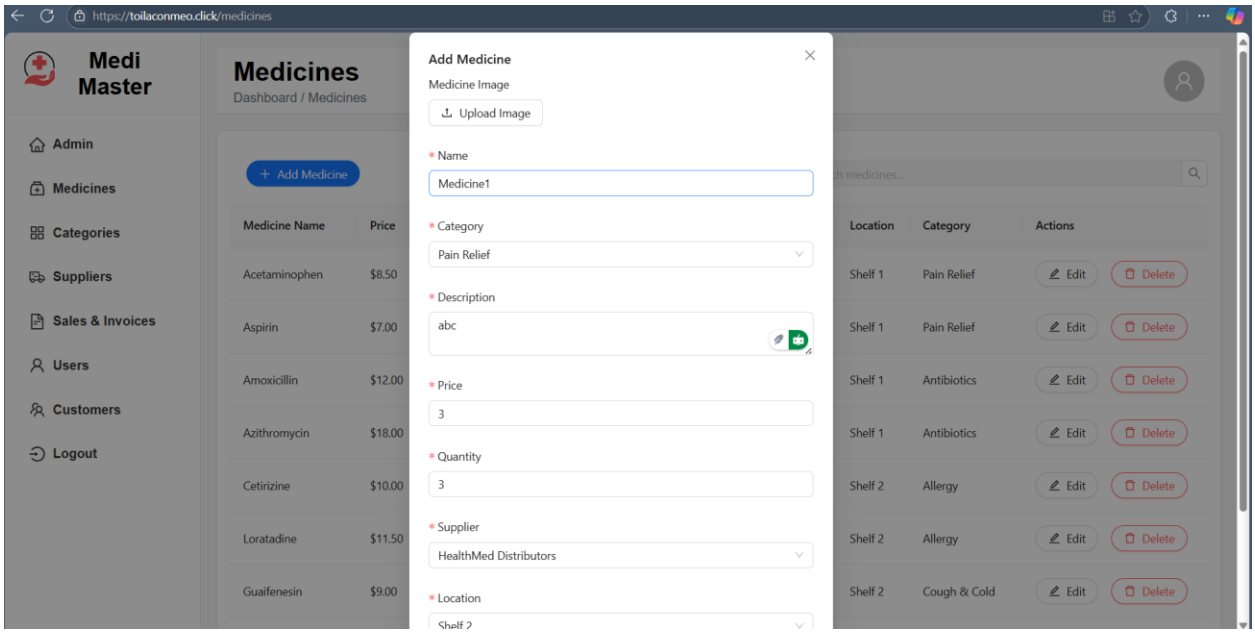



*Figure 83*

### 3.2.2 Edit Medicine

Step 1: Login the system with any role.

Step 2: Click on the button Medicine to navigate to Medicines Management screen.

Step 3: Click on “Edit”.

Step 4: Fill the form then click on “Save”.

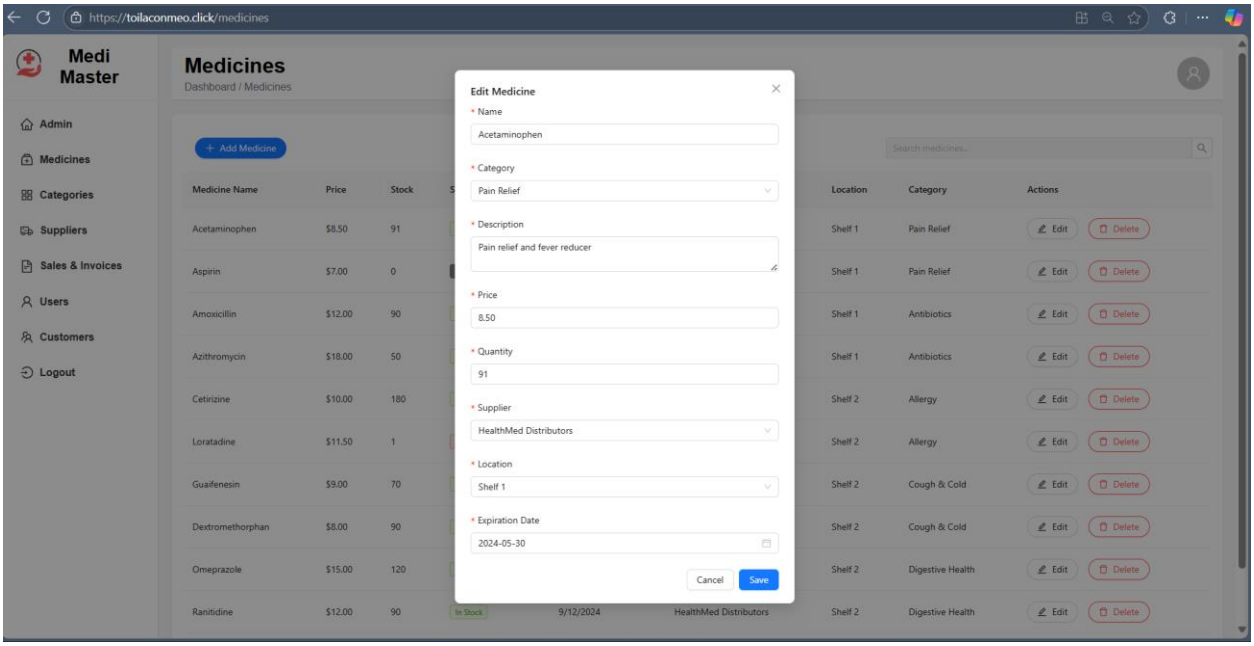



*Figure 84*

### 3.2.3 Delete Medicine

Step 1: Login the system with any role.

Step 2: Click on the button Medicine to navigate to Medicines Management screen.

Step 3: Click on “Delete”.

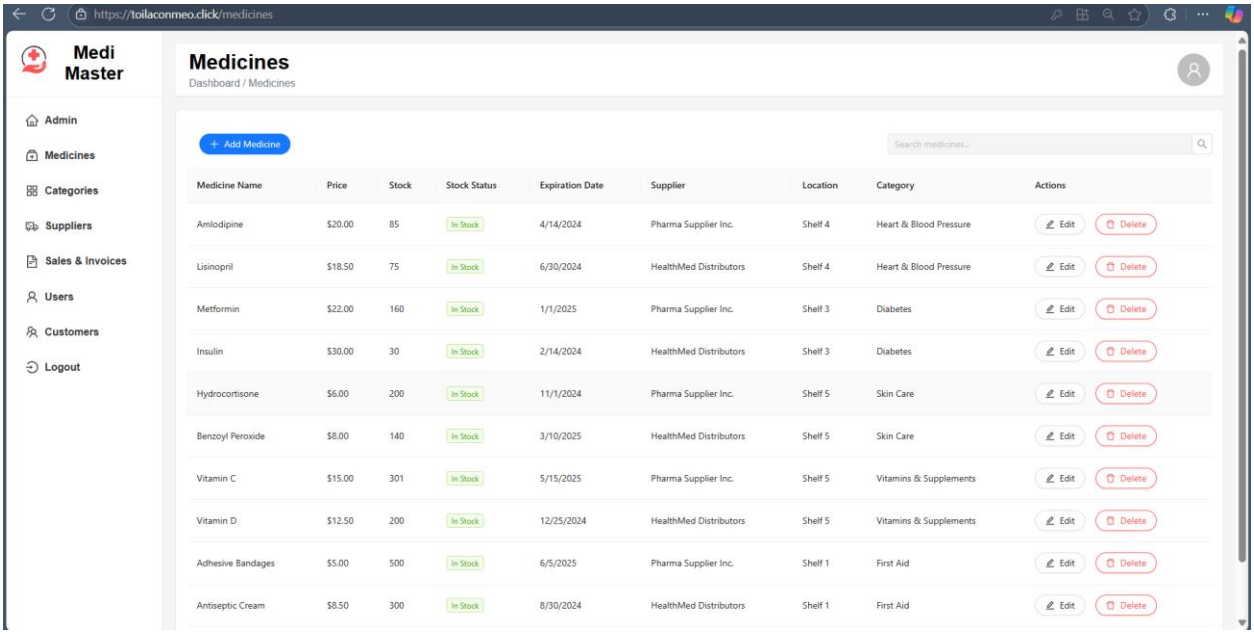



*Figure 85*

### 3.2.4 Search Medicine

Step 1: Login the system with any role.

Step 2: Click on the button Medicine to navigate to Medicines Management screen.

Step 3: Click on “Search bar”.

Step 4: fill the blank with medicines name, or some first letters.

Step 5: Click the search button or press enter.

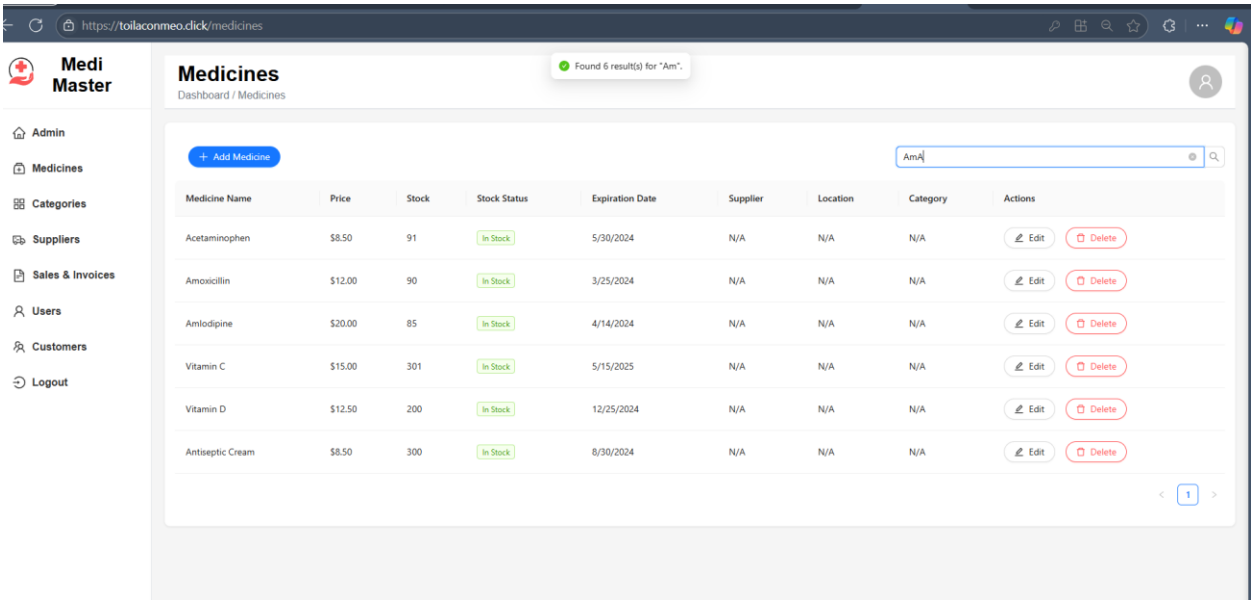



*Figure 86*

## 3.3 SUPPLIERS MANAGEMENT FUNCTIONS

### 3.3.1 Add New Supplier

Step 1: Login the system with any role.

Step 2: Click on the button Suppliers to navigate to Suppliers Management screen.

Step 3: Click on “Add Supplier”.

Step 4: Click “Add”.

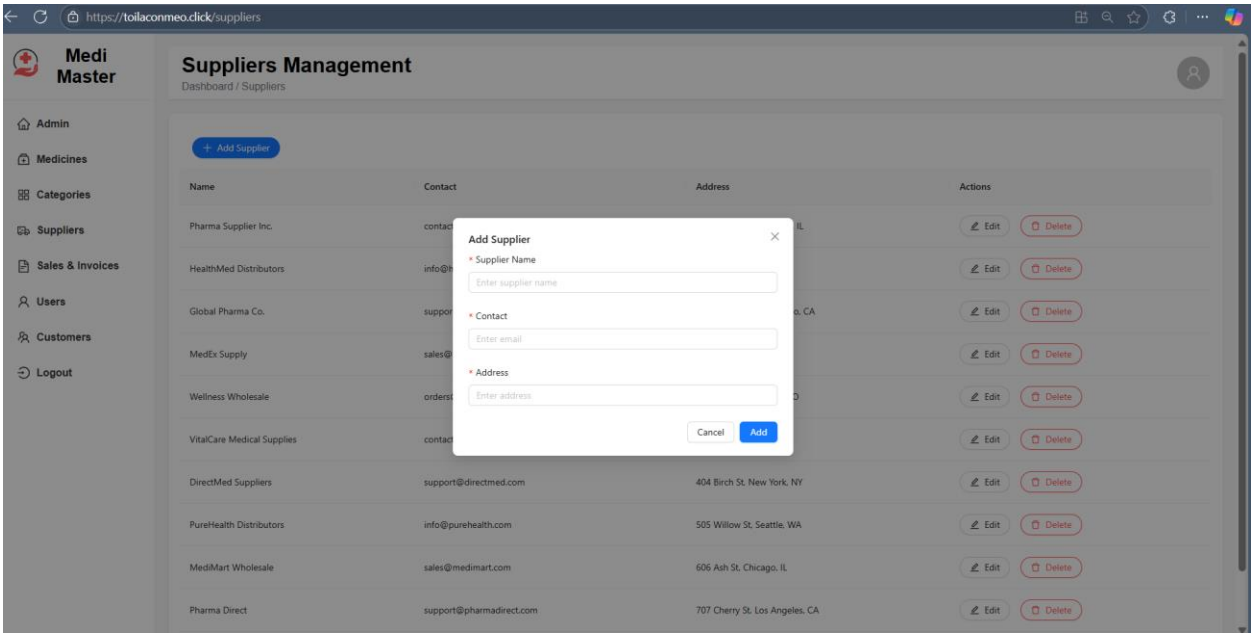



*Figure 87*

### 3.3.2 Edit Supplier

Step 1: Login the system with any role.

Step 2: Click on the button Suppliers to navigate to Suppliers Management screen.

Step 3: Click on “Edit”.

Step 4: Click “Save”.

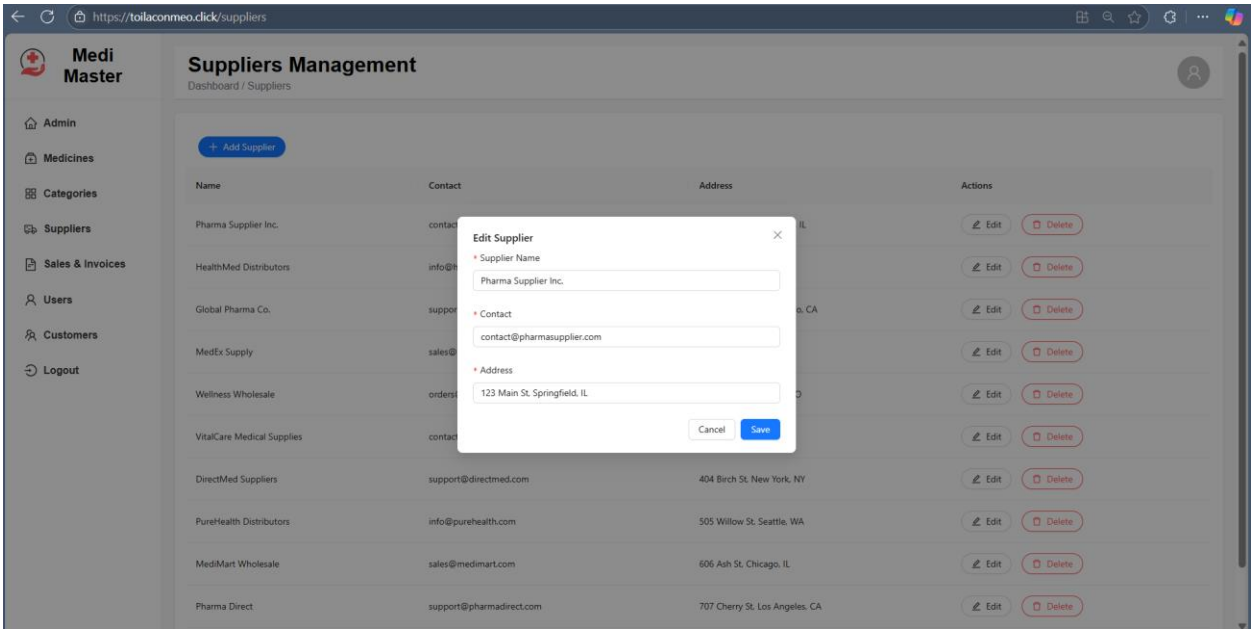



*Figure 88*

### 3.3.3 Delete Supplier

Step 1: Login the system with any role.

Step 2: Click on the button Suppliers to navigate to Suppliers Management screen.

*Step 3: Click on "Delete".Figure 89*

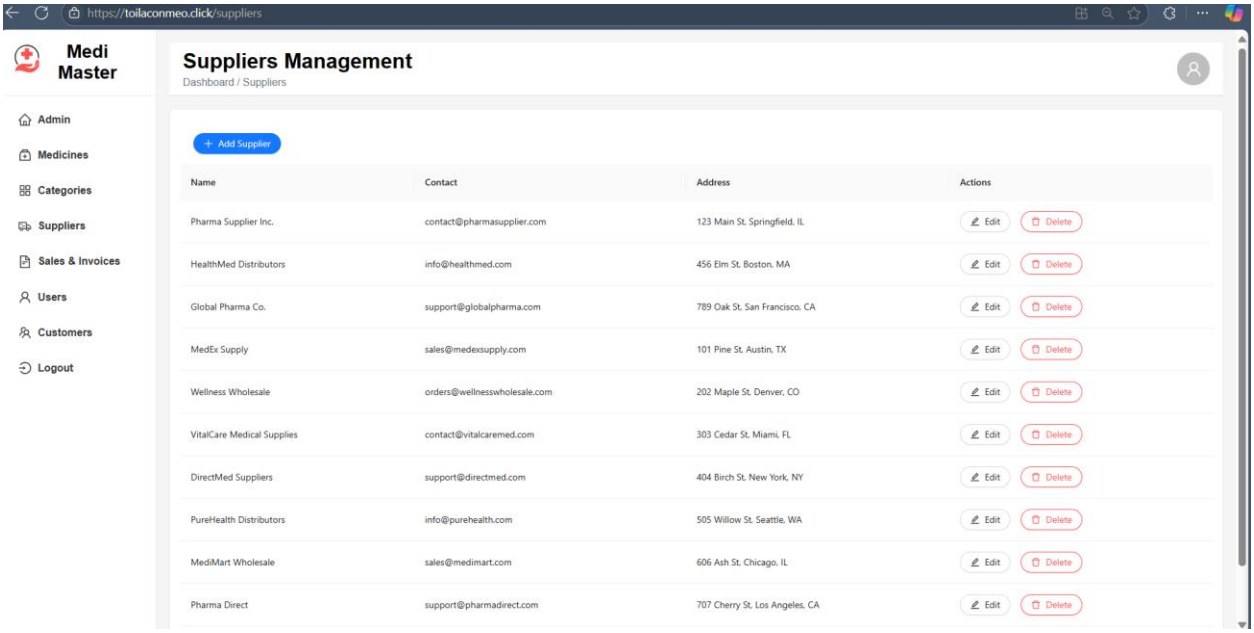



*Figure 90*

## 3.4 SALE & INVOICES MANAGEMENT FUNCTIONS

### 3.4.1 Add New Invoice

Step 1: Login the system with any role.

Step 2: Click on the button "Sales & Invoices" to navigate to Sales & Invoices Management screen.

Step 3: Click on "Add Invoice".

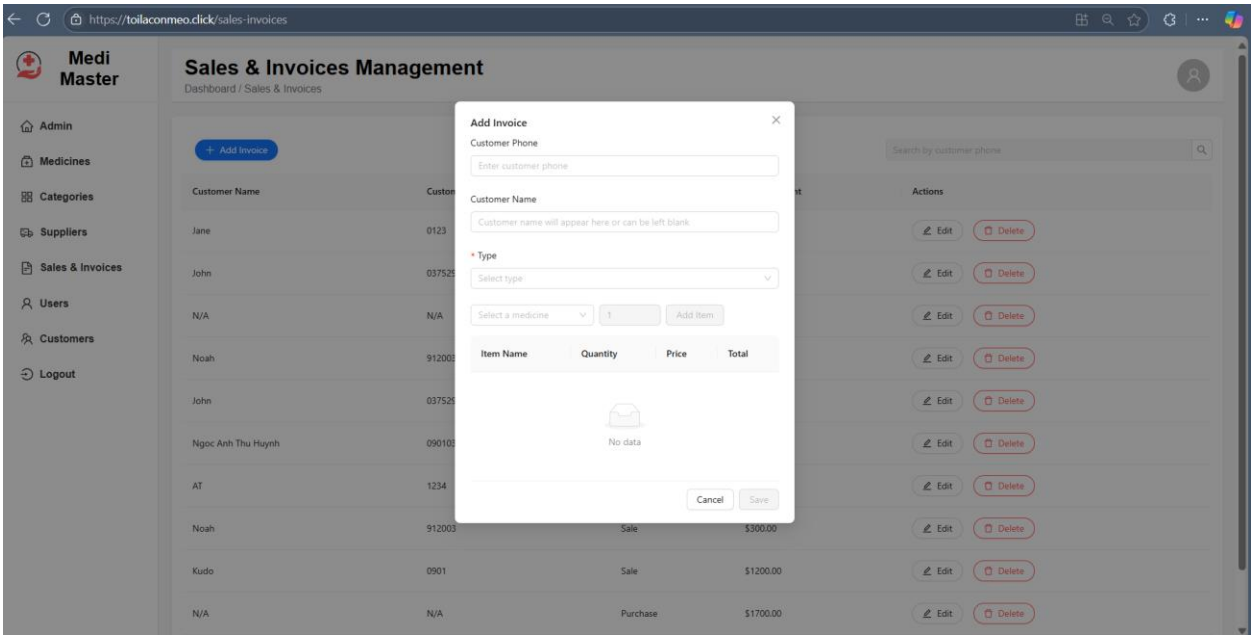



*Figure 91*

Case 1: The customer has made a previous purchase and has phone number information in the data

Step 4: Insert the customer number to search

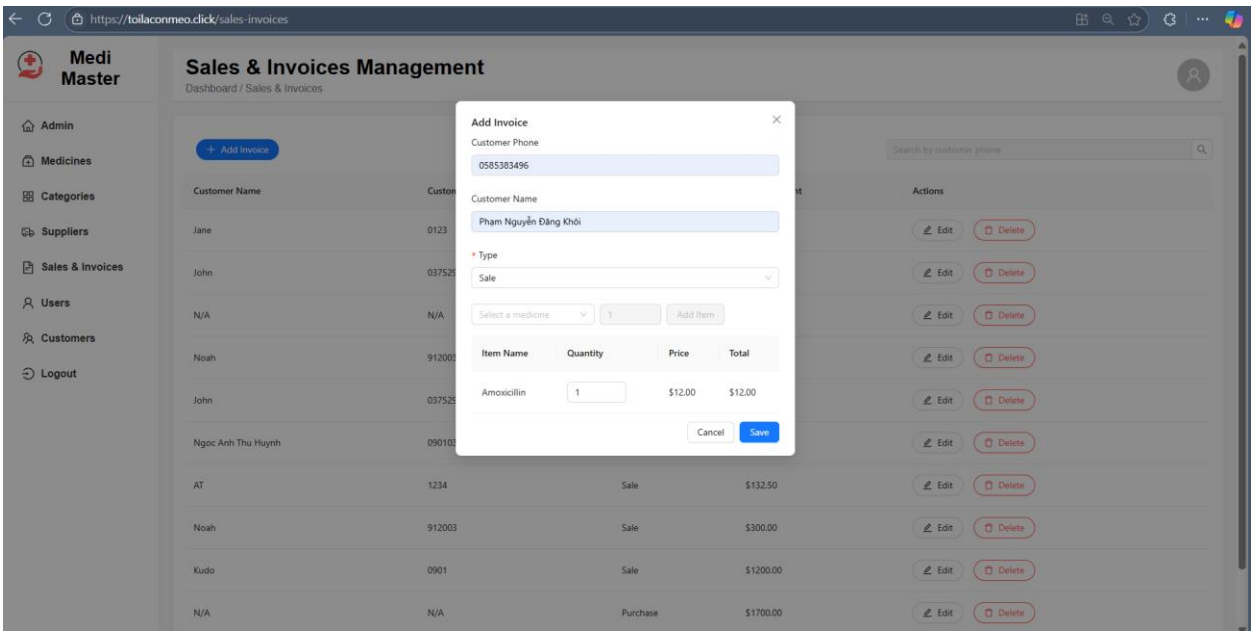



*Figure 92*

Step 5: Add medicines and click “Save”.

Case 2: The customer did not provide a phone number.

Step 4: Let the phone and customer name in the form being blank.

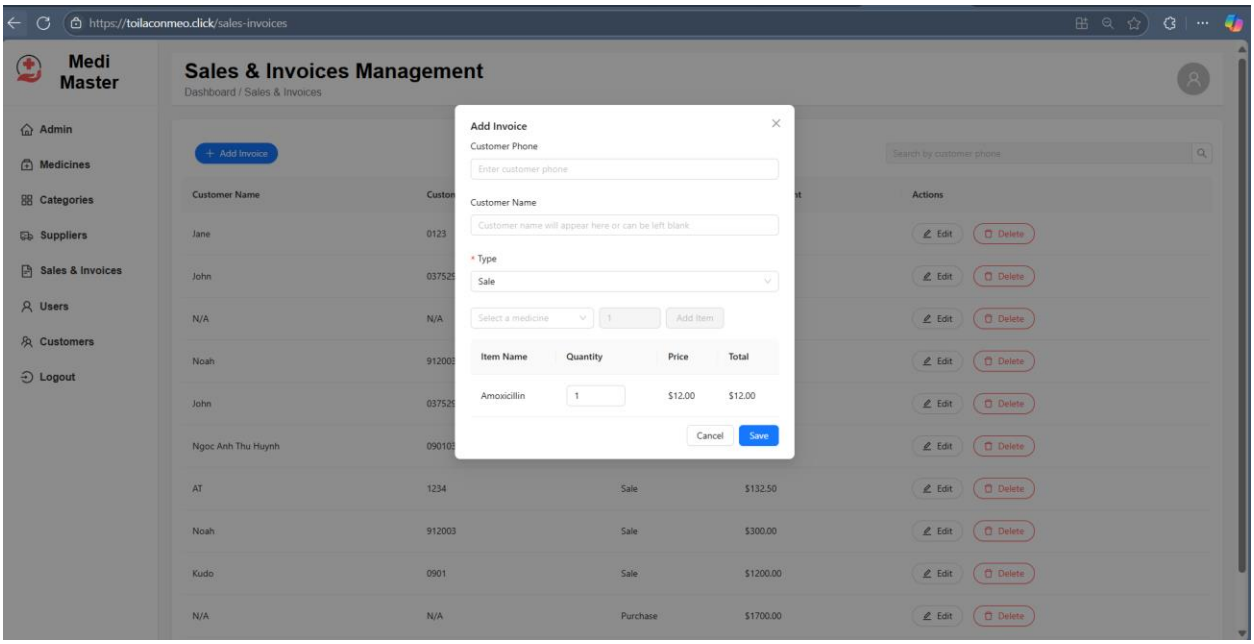



*Figure 93*

Step 5: Add medicines then click “Save”

Thus, the system will save the invoice without the customer's name, and this invoice will not be searchable by customer id.

### 3.4.2 Edit Invoice

Step 1: Login the system with any role.

Step 2: Click on the button ”Sales & Invoices” to navigate to Sales & Invoices Management screen.

Step 3: Click on “Edit Invoice”.

Step 4: Fill the form and click “Save”

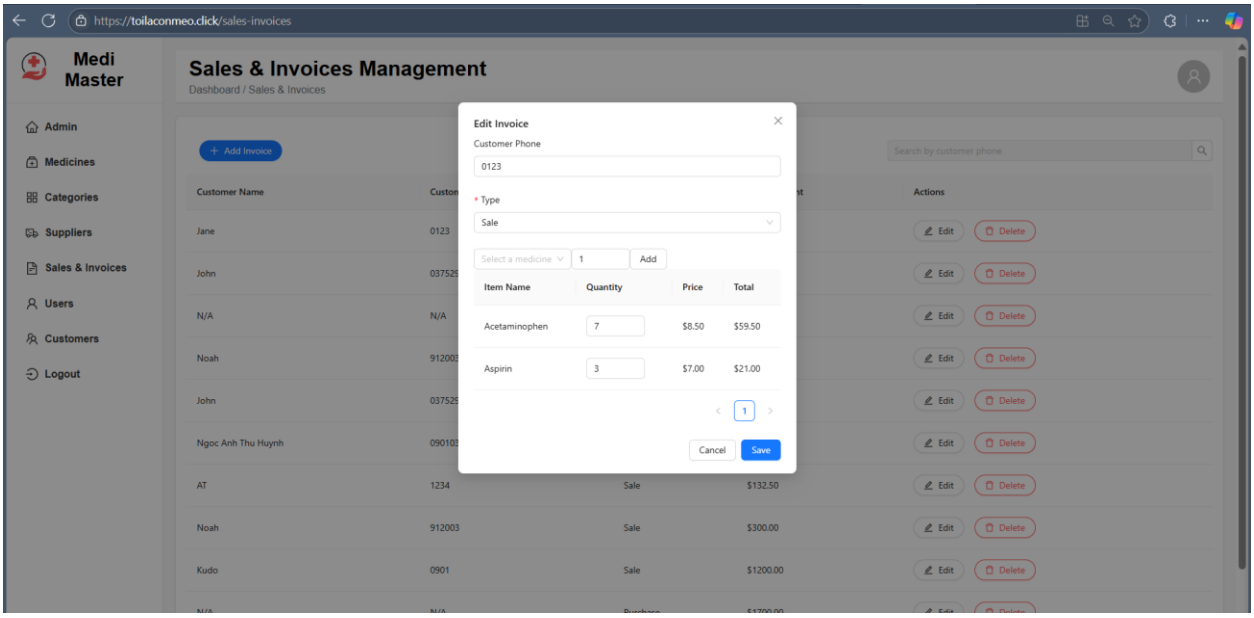



*Figure 94*

### 3.4.3 Delete Invoice

Step 1: Login the system with any role.

Step 2: Click on the button ”Sales & Invoices” to navigate to Sales & Invoices Management screen.

Step 3: Click on “Delete”.

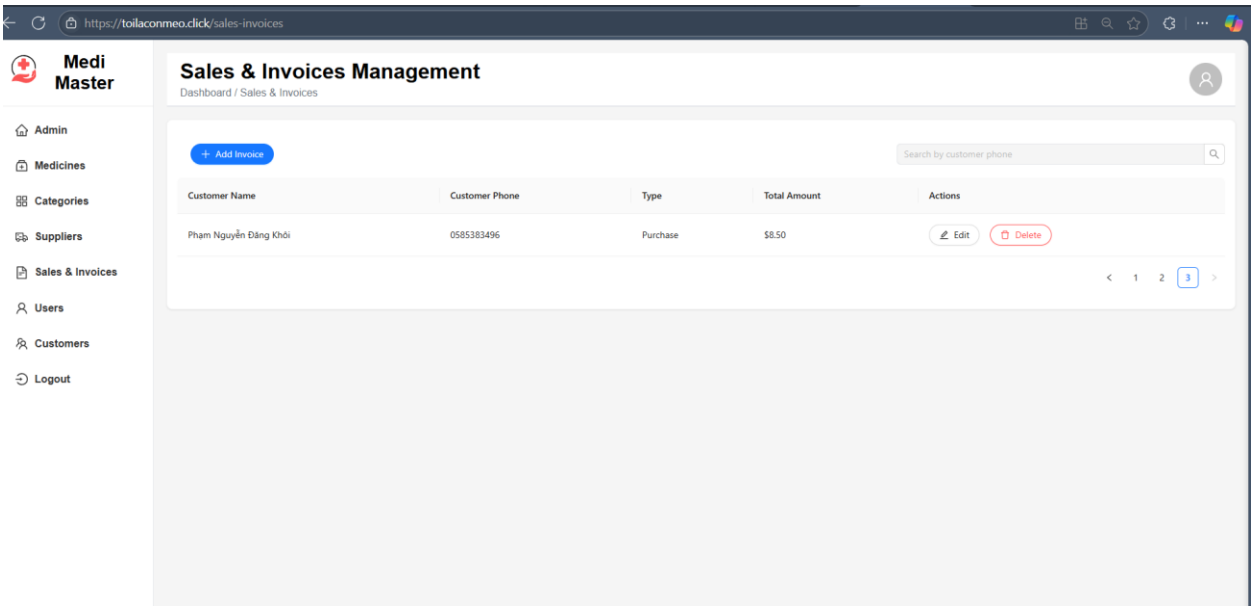



*Figure 95*

### 3.4.4 Search Invoice via Customer's phone

Step 1: Login the system with any role.

Step 2: Click on the button "Sales & Invoices" to navigate to Sales & Invoices Management screen.

Step 3: Click on the search bar then insert the phone number.

Step 4: Click search button or press Enter.

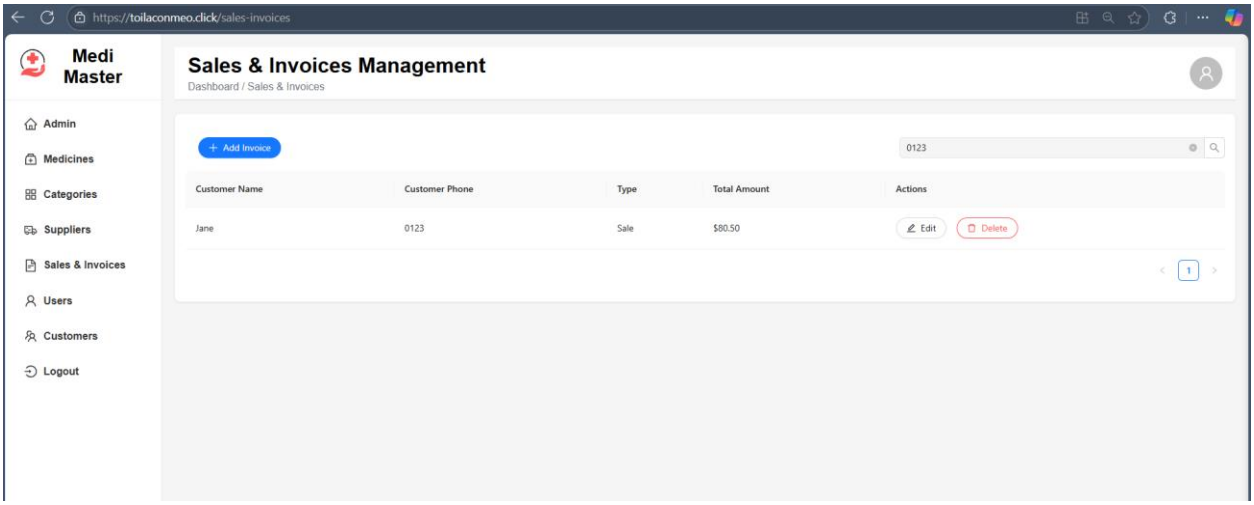



*Figure 96*

## 3.5 CUSTOMER MANAGEMENT FUNCTIONS

### 3.5.1 Add New Customer

Step 1: Login the system with any role.

Step 2: Click on the button "Customers" to navigate to Customers Management screen.

Step 3: Click on the "Add Customer" button.

Step 4: Fill the information then click "OK".

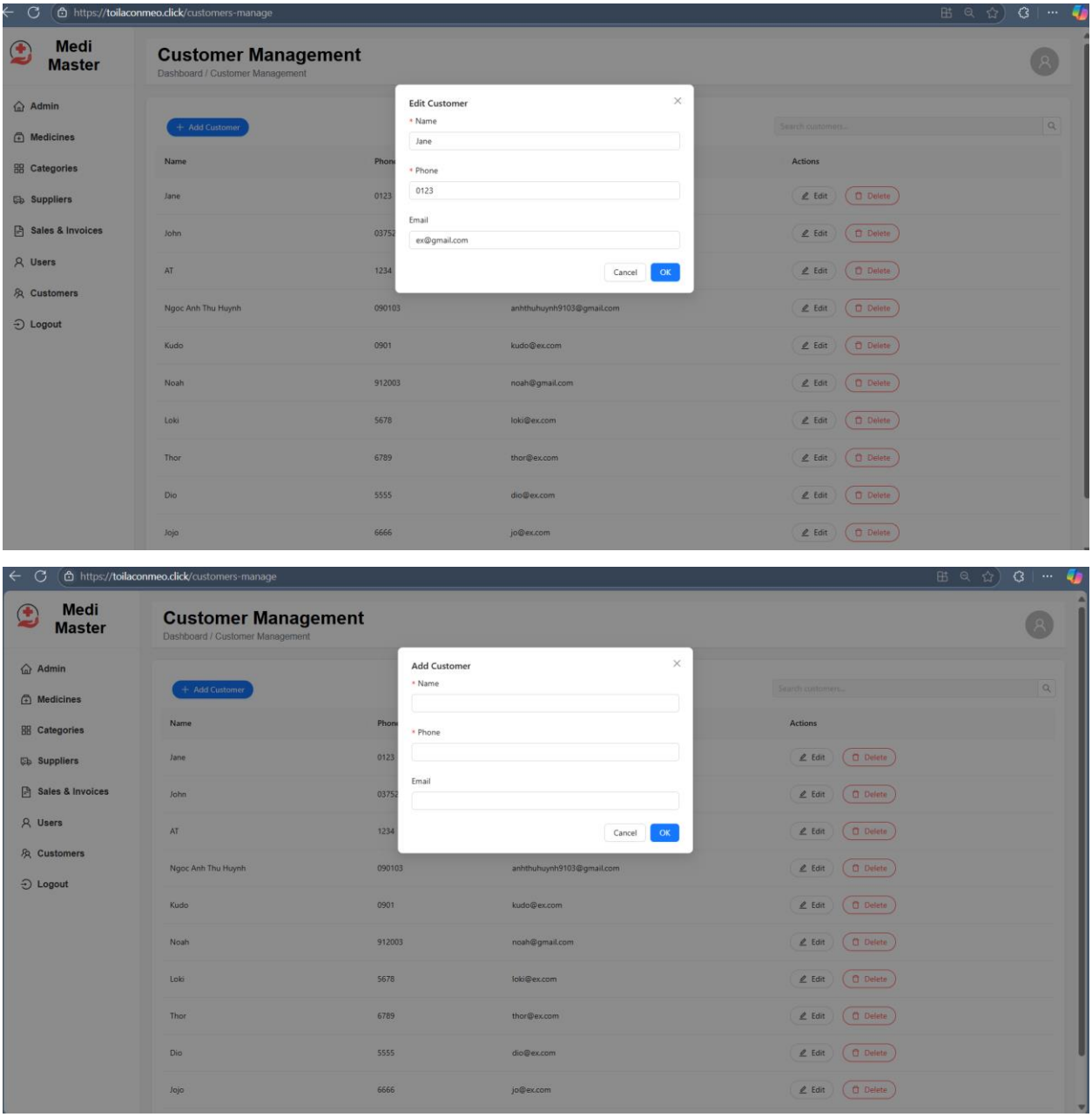






*Figure 97*

### 3.5.2 Edit Customer

Step 1: Login the system with any role.

Step 2: Click on the button ”Customers” to navigate to Customers Management screen.

Step 3: Click on the “Edit” button.

Step 4: Fill the information then click “OK”.

*Figure 98*

### 3.5.3 Delete Customer

Step 1: Login the system with any role.

Step 2: Click on the button ”Customers” to navigate to Customers Management screen.

Step 3: Click on the “Delete” button.

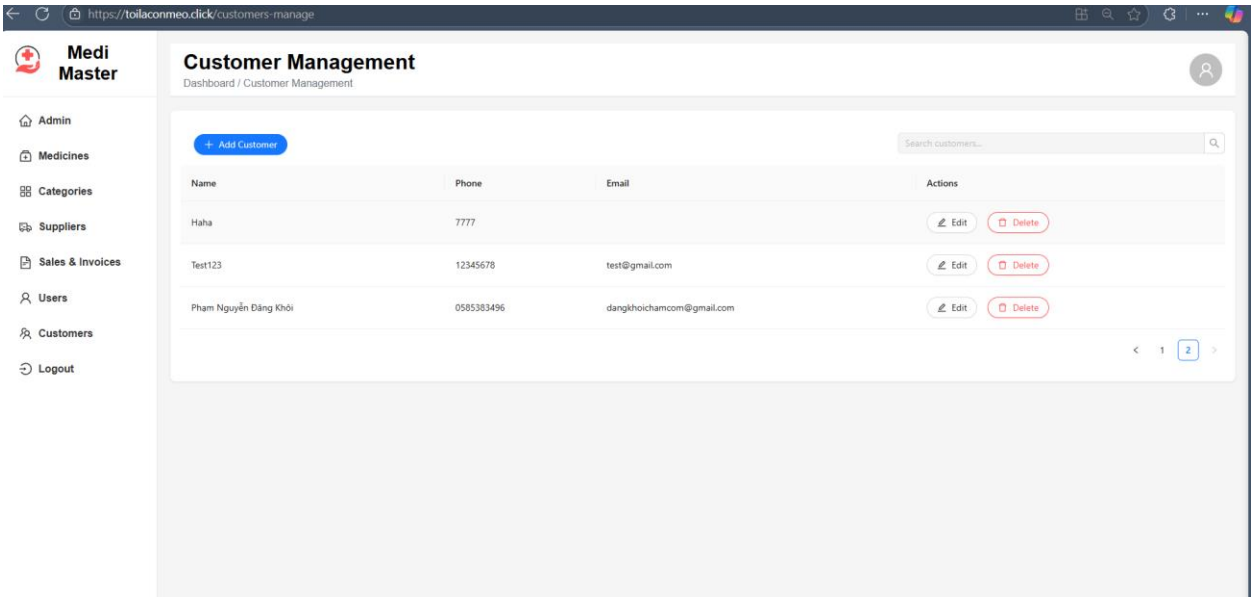



*Figure 99*

Step 4: Click “OK”.

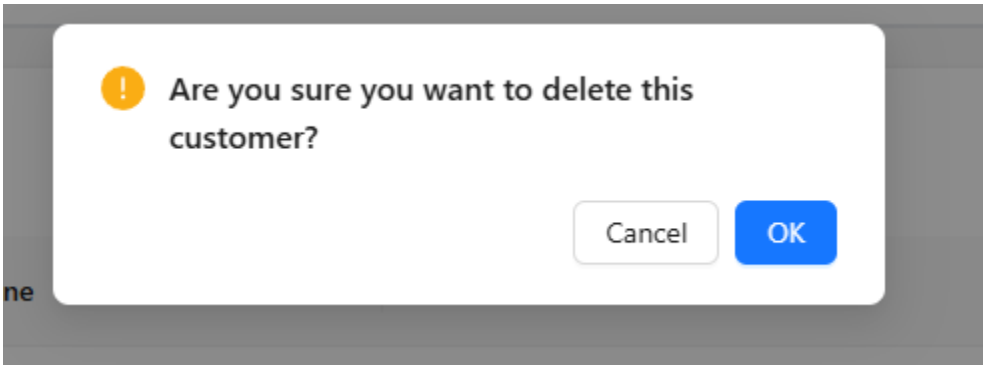



*Figure 100*

### 3.5.4 Search Customer via Customer's phone

Step 1: Login the system with any role.

Step 2: Click on the button "Customers" to navigate to Customer Management screen.

Step 3: Click on the search bar then insert the phone number.

Step 4: Click search button or press Enter.

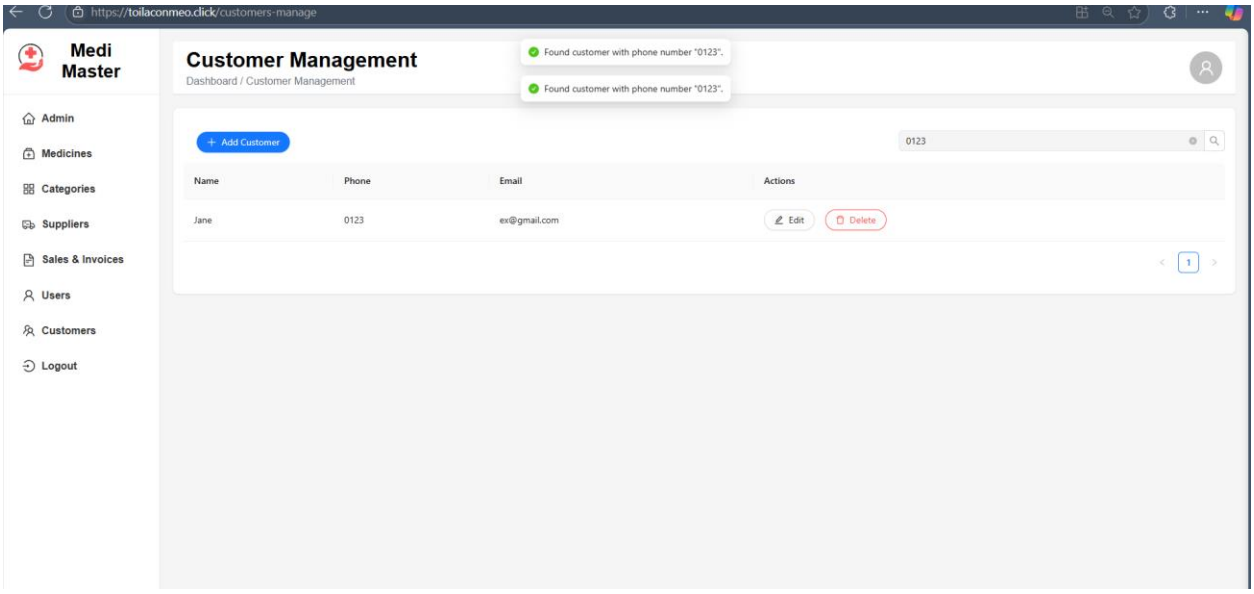



*Figure 101*

## 4 DISCUSSION AND CONCLUSION

The successful deployment of the Pharmacist Management System on a certified domain, https://medimaster.io.vn , represents an important milestone in the project. This deployment provides user accessibility while also demonstrating the system's readiness for real-world applications. The recognized domain strengthens the application's professionalism and credibility, making it more safe and reliable for end users.

This project establishes a solid basis for pharmacy operations by streamlining processes such as inventory management, sales monitoring, and invoicing. The system has been developed to facilitate scalability and integration, making it responsive to future changes.

Looking forward, the project has enormous potential for expansion with the deployment of new features, including:

### 4.1 Introducing New User Roles:

The application will be extended to include customers as end users, accessible through both the web application and a dedicated mobile application. Customers will be able to:

- Manage their personal information.
- Access their purchasing history at pharmacies using our system.
- Maintain records of their personal prescriptions, enhancing healthcare transparency and accessibility.

### 4.2 AI-Driven Enhancements:

To further improve user experience, artificial intelligence will be integrated into both the mobile and web applications. This feature will enable customers to:

- Search for their symptoms (e.g., headache, toothache, nausea) and receive recommendations for non-prescription medicines.
- Ensure that suggestions are aligned with input from professional pharmacists, enhancing the accuracy and safety of the feature.

These future upgrades aim to raise the application from a pharmacy management system to a full-service healthcare solution that benefits both pharmacy employees and clients. By employing innovative technology and addressing real-world demands, especially with Viet Nam culture, this project will continue to improve and bring value to its users.
Finally, the deployment of this technology demonstrates the team's commitment to innovation and excellence. The project is a big step toward modernizing pharmacy operations, laying the framework for future developments that will improve user involvement and healthcare efficiency. We would also modified the website structure to responsive.

# 5 REFERENCES


**Online Tutorials**

1. **React Documentation**: https://reactjs.org/docs/getting-started.html
2. **MySQL Official Documentation**: https://dev.mysql.com/doc/

**Case Studies / Reports**

1. **"Improving Pharmacy Management Systems through Digital Solutions" by WHO** *(https://www.who.int/health-topics/pharmaceutical-products)*

**Tools and Platforms**

1. **ERD Tools**: Draw.io, Lucidchart
2. **Version Control**: Git, GitHub

**Tutorials for Tech Stack**

1. **React with Ant Design**: https://dev.to/dangol/ant-design-react-how-to-start-creating-your-website-2k51
2. **Node.js and MySQL**: https://www.w3schools.com/nodejs/