Device Manager Web Application

1. Introduction

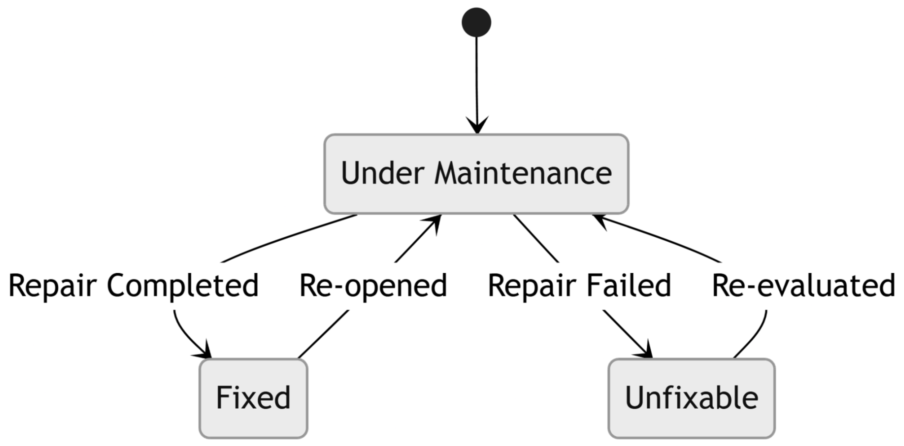
Device Manager is a full-stack web application designed to streamline the process of managing device repairs for small to medium-sized repair shops. It provides a user-friendly interface for technicians and administrative staff to log, track, and manage the entire lifecycle of a device repair, from intake to completion.

This application is built with a modern technology stack, featuring a Node.js and Express.js backend, a lightweight and efficient SQLite database, and a dynamic vanilla JavaScript frontend. The primary goal of this application is to offer a simple, yet powerful, tool for repair shops to organize their workflow, improve customer service, and maintain a clear record of all repair jobs.

2. Features

The Device Manager application comes with a comprehensive set of features to handle all aspects of repair management:

- **User Authentication:** A secure login and registration system ensures that only authorized personnel can access the application's data. User credentials are encrypted for enhanced security.
- **Device Management:**
  - **Add New Devices:** Easily add new repair jobs by entering customer details, device information, the quoted amount, the intake date, and the initial repair status.
  - **View All Devices:** A clear and organized table displays all registered devices with their relevant details.
  - **Edit Device Information:** Modify the details of an existing repair job, such as updating the status, changing the cost, or correcting customer information.
  - **Delete Devices:** Remove completed or canceled repair jobs from the system.



- **Search and Filter:**
  - **Real-time Search:** A powerful search functionality allows users to quickly find specific devices by customer name or device name.
  - **Status Filtering:** Filter the device list based on their current repair status (e.g., "Fixed", "Under Maintenance", "Unfixable").
- **Shareable Device Details:** A "Share" feature allows users to copy the essential details of a repair job to the clipboard, making it easy to share with customers or other technicians.
- **Responsive User Interface:** The application is designed to be responsive and accessible on various screen sizes, from desktops to tablets.

3. Technical Stack

The application is built using a combination of modern and reliable technologies:

- **Backend:**
  - **Node.js:** A JavaScript runtime environment that allows for the execution of JavaScript code on the server-side.
  - **Express.js:** A minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications.
  - **SQLite:** A C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.
  - **bcrypt:** A library for hashing passwords.
  - **jsonwebtoken (JWT):** A compact, URL-safe means of representing claims to be transferred between two parties.

- **Frontend:**
  - **HTML5:** The standard markup language for documents designed to be displayed in a web browser.
  - **CSS3:** A stylesheet language used to describe the presentation of a document written in a markup language like HTML.
  - **Vanilla JavaScript:** The standard scripting language for web pages, used to create dynamic and interactive content.

4. High-Level Architecture

The Device Manager application follows a classic client-server architecture:

- **Client (Frontend):** The user interface is a single-page application (SPA) built with HTML, CSS, and JavaScript. It runs in the user's web browser and is responsible for presenting data and handling user interactions. It communicates with the backend via RESTful API calls to fetch and manipulate data.
- **Server (Backend):** The backend is a Node.js application powered by the Express.js framework. It exposes a set of RESTful API endpoints that the frontend consumes. Its responsibilities include handling business logic, authenticating users, and interacting with the SQLite database to perform CRUD (Create, Read, Update, Delete) operations.
- **Database:** A SQLite database is used for data persistence. It stores information about users and the devices they manage.

5. Getting Started

To get the Device Manager application up and running on your local machine, follow these steps:

5.1. Prerequisites

Make sure you have the following software installed on your system:

- Node.js (which includes npm, the Node.js package manager)

5.2. Installation and Setup

1. **Clone the Repository:**

   ```
   git clone <https://github.com/mohammedtakween/device-manager.git>
   cd device-manager
   ```

2. **Install Dependencies:**
   Navigate to the project's root directory in your terminal and run the following command to install the necessary Node.js packages:

   ```
   npm install
   ```

5.3. Running the Application

To start both the server and open the application in your default web browser, you can use the provided batch file:

```
start_manager.bat
```

Alternatively, you can start the server manually by running:

```
npm start
```

This will start the Node.js server, and you can then access the application by navigating to http://localhost:3000 in your web browser.

6. API Endpoints

The backend provides a RESTful API for managing users and devices. All device-related endpoints require a valid JSON Web Token (JWT) in the Authorization header.

6.1. Authentication Endpoints

- **POST /api/register**
  - **Description:** Registers a new user.
  - **Request Body:**
    ```json
    {
      "username": "your_username",
      "password": "your_password"
    }
    ```
  - **Response:**
    - **201 Created:** { "message": "User registered successfully", "userId": 1 }
    - **400 Bad Request:** { "message": "Username and password are required" }
    - **409 Conflict:** { "message": "Username already exists" }
- **POST /api/login**
  - **Description:** Authenticates a user and returns a JWT.
  - **Request Body:**
    ```json
    {
      "username": "your_username",
      "password": "your_password"
    }
    ```
  - **Response:**
    - **200 OK:** { "message": "Logged in successfully", "token": "your_jwt_token" }
    - **400 Bad Request:** { "message": "Invalid credentials" }

6.2. Device Endpoints

- **GET /api/devices**
  - **Description:** Retrieves all devices associated with the authenticated user.
  - **Response (200 OK):** An array of device objects.
- **POST /api/devices**
  - **Description:** Adds a new device for the authenticated user.
  - **Request Body:** A device object.
  - **Response (201 Created):** The newly created device object.
- **PUT /api/devices/:id**
  - **Description:** Updates an existing device.
  - **Request Body:** A device object with the updated information.
  - **Response:**
    - **200 OK:** The updated device object.
    - **404 Not Found:** { "message": "Device not found" }
- **DELETE /api/devices/:id**
  - **Description:** Deletes a device.
  - **Response:**
    - **204 No Content:** Successful deletion.
    - **404 Not Found:** { "message": "Device not found" }

7. Database Schema & Tables

The application uses an SQLite database with two main tables: users and devices.

- **users table:**
  - id: INTEGER PRIMARY KEY AUTOINCREMENT
  - username: TEXT UNIQUE NOT NULL
  - password: TEXT NOT NULL
- **devices table:**
  - id: INTEGER PRIMARY KEY AUTOINCREMENT
  - customerName: TEXT NOT NULL
  - deviceName: TEXT NOT NULL
  - amount: REAL NOT NULL
  - date: TEXT NOT NULL
  - status: TEXT NOT NULL
  - userId: INTEGER, FOREIGN KEY (userId) REFERENCES users(id)

Data Dictionary Tables

Users Table

- **Description:** Stores user account information for authentication.

Field Name	Data Type	Constraints	Description	Example
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	The unique identifier for the user.	1
username	TEXT	UNIQUE, NOT NULL	The user's chosen login name.	"tech_user"
password	TEXT	NOT NULL	The user's hashed password.	"$2b$10$..."

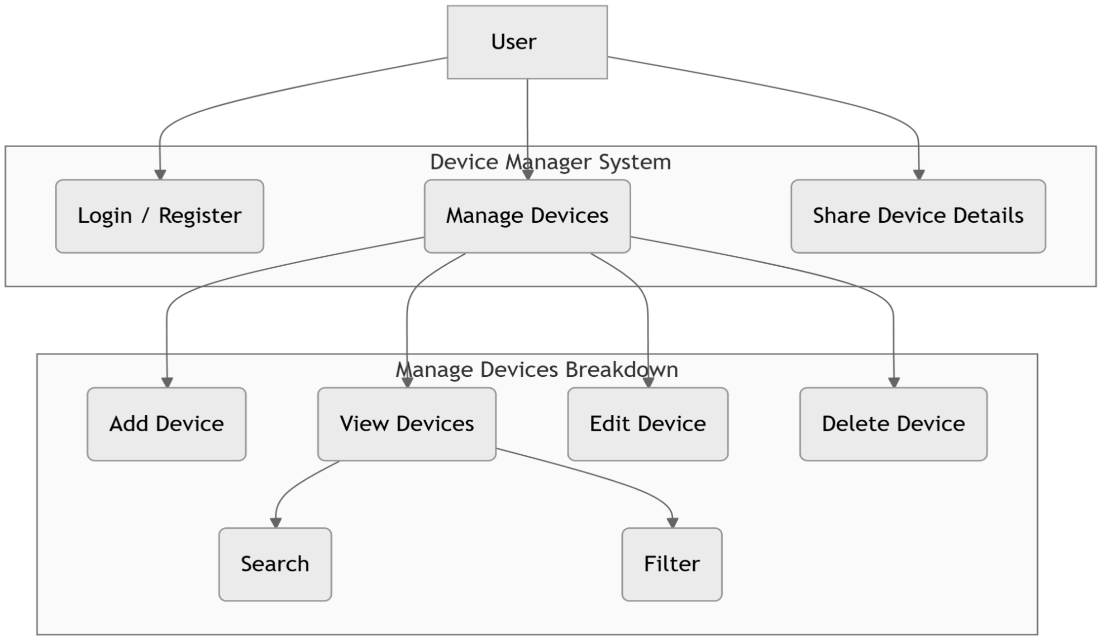
Devices Table

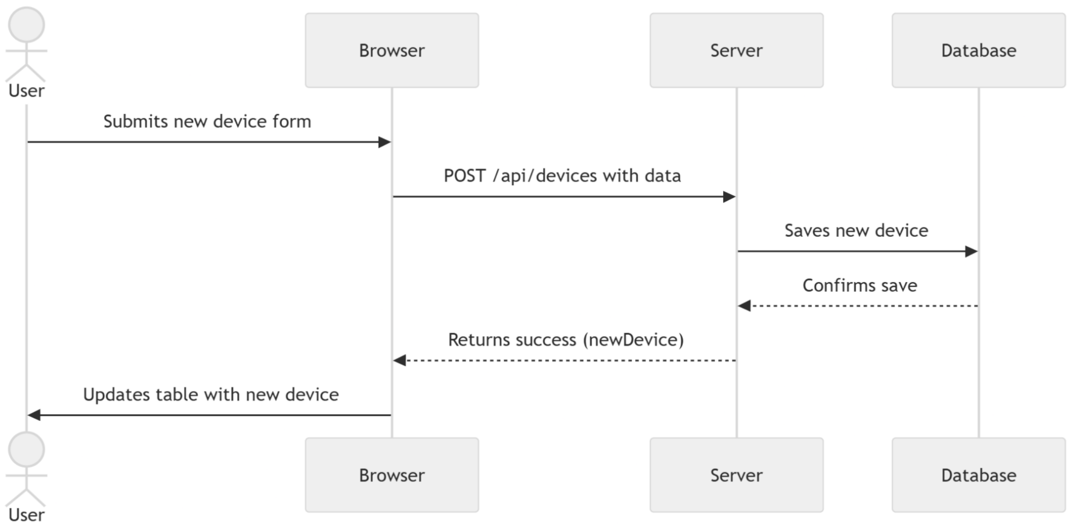
- **Description:** Stores information about each device repair job.

Field Name	Data Type	Constraints	Description	Example
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	The unique identifier for the device record.	101
customerName	TEXT	NOT NULL	The full name of the device's owner.	"Mohammed"
deviceName	TEXT	NOT NULL	The name or model of the device being repaired.	"iPhone 12 Pro"
amount	REAL	NOT NULL	The quoted or final cost for the repair.	15000
date	TEXT	NOT NULL	The date the device was registered into the system.	"2025-07-26"
status	TEXT	NOT NULL	The current status of the repair.	"Under Maintenance"
userId	INTEGER	FOREIGN KEY (REFERENCES users.id)	The ID of the user who registered the device.	1

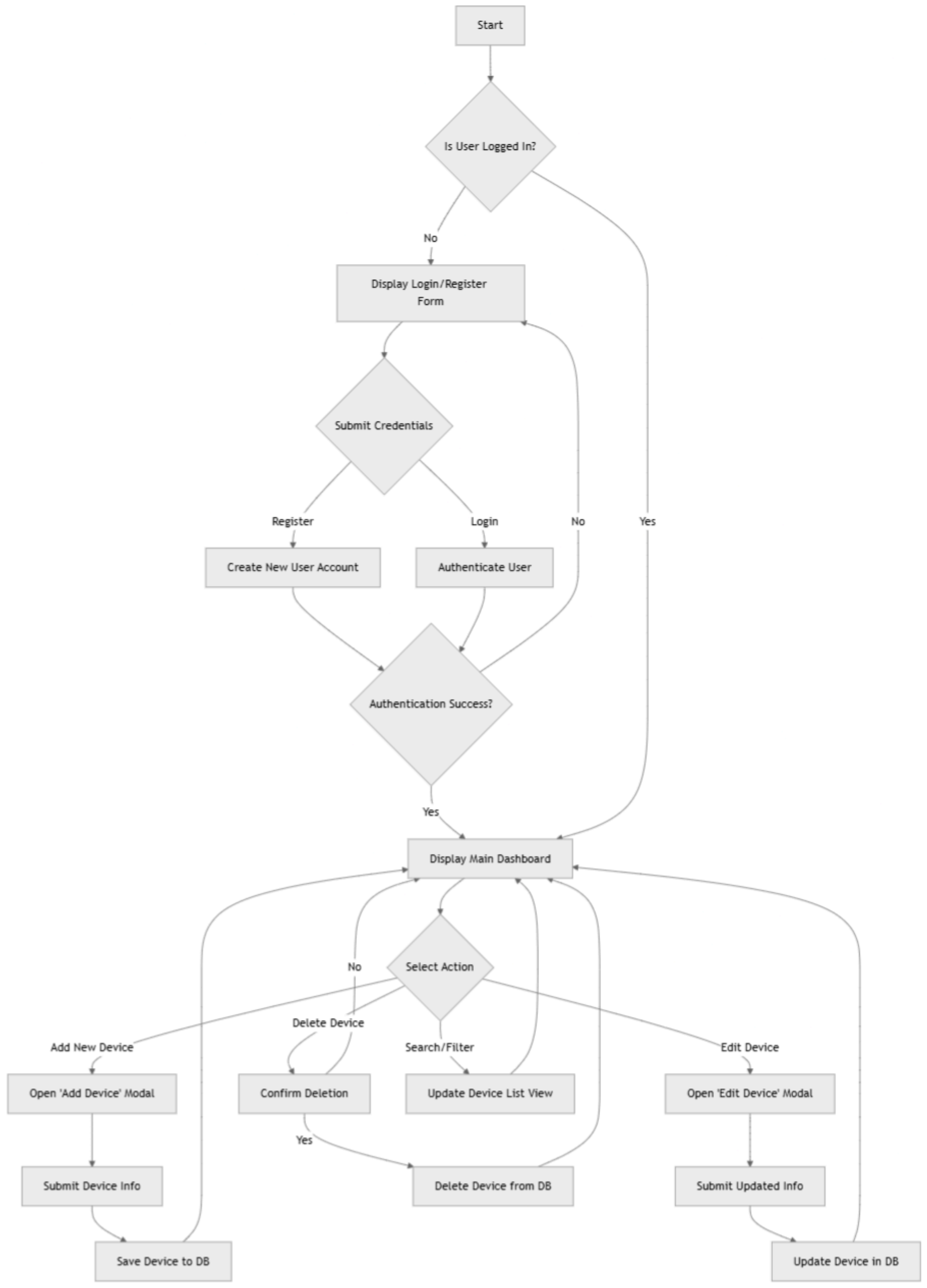
8. UML For This System

Use Case/Sequence Diagram

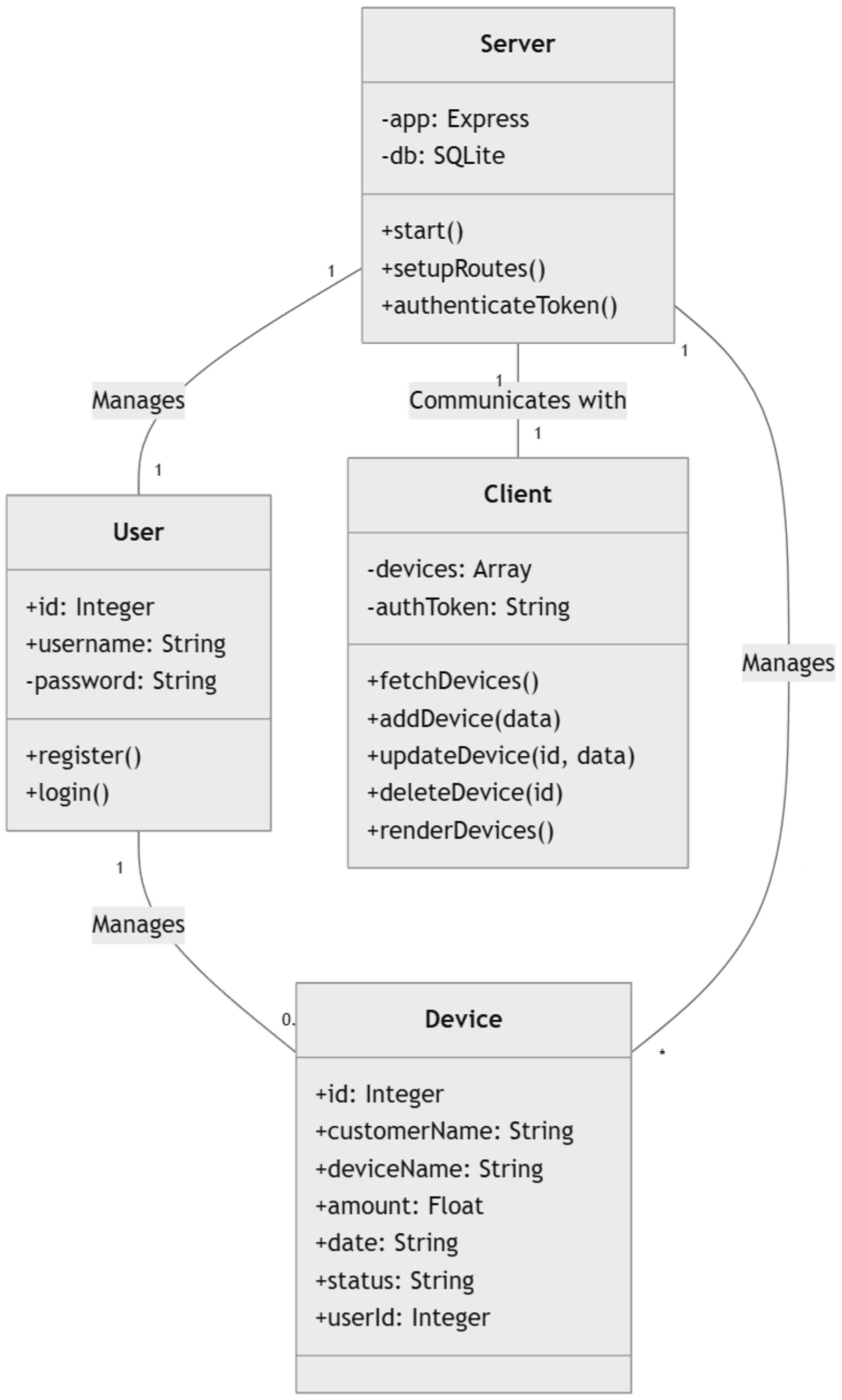




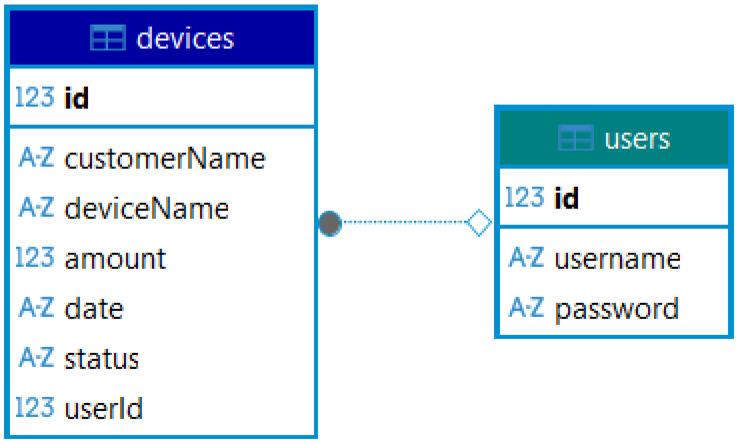
Activity Diagram



Class Diagram

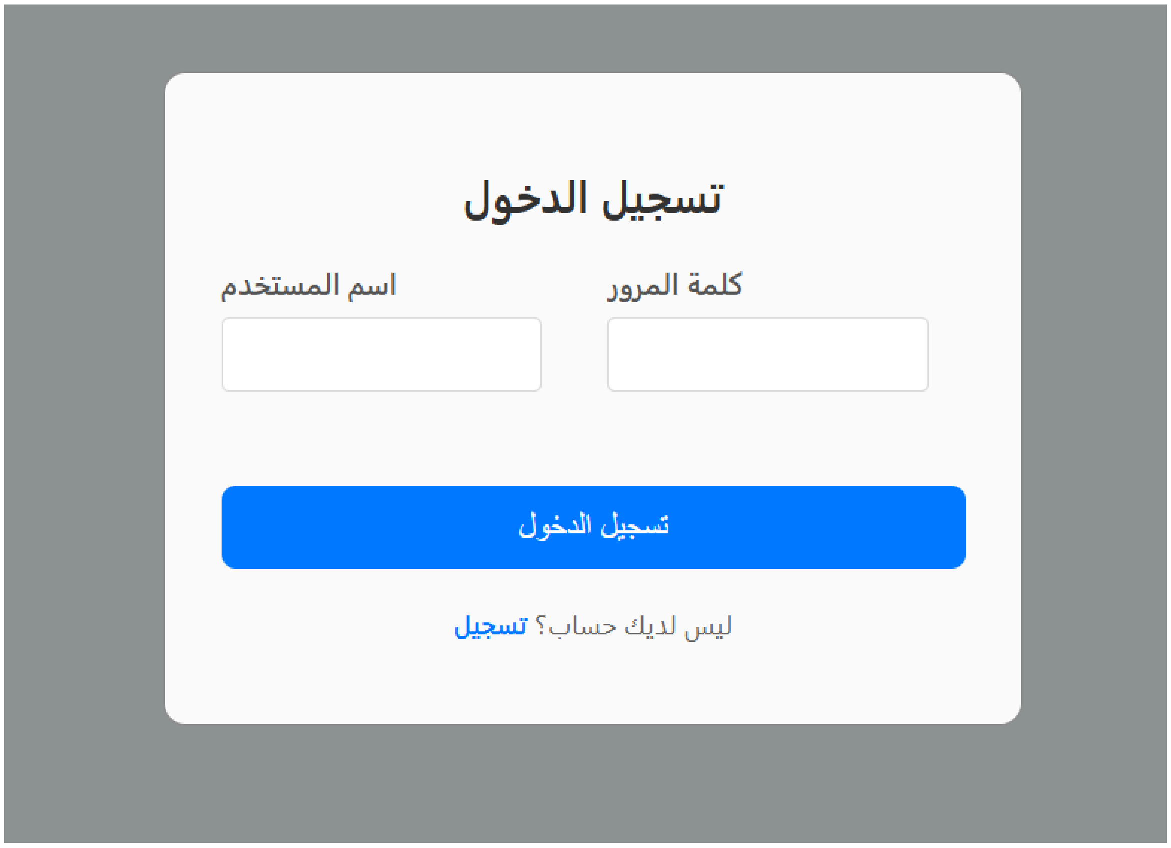


9. Entity-Relationship Diagram (ERD)

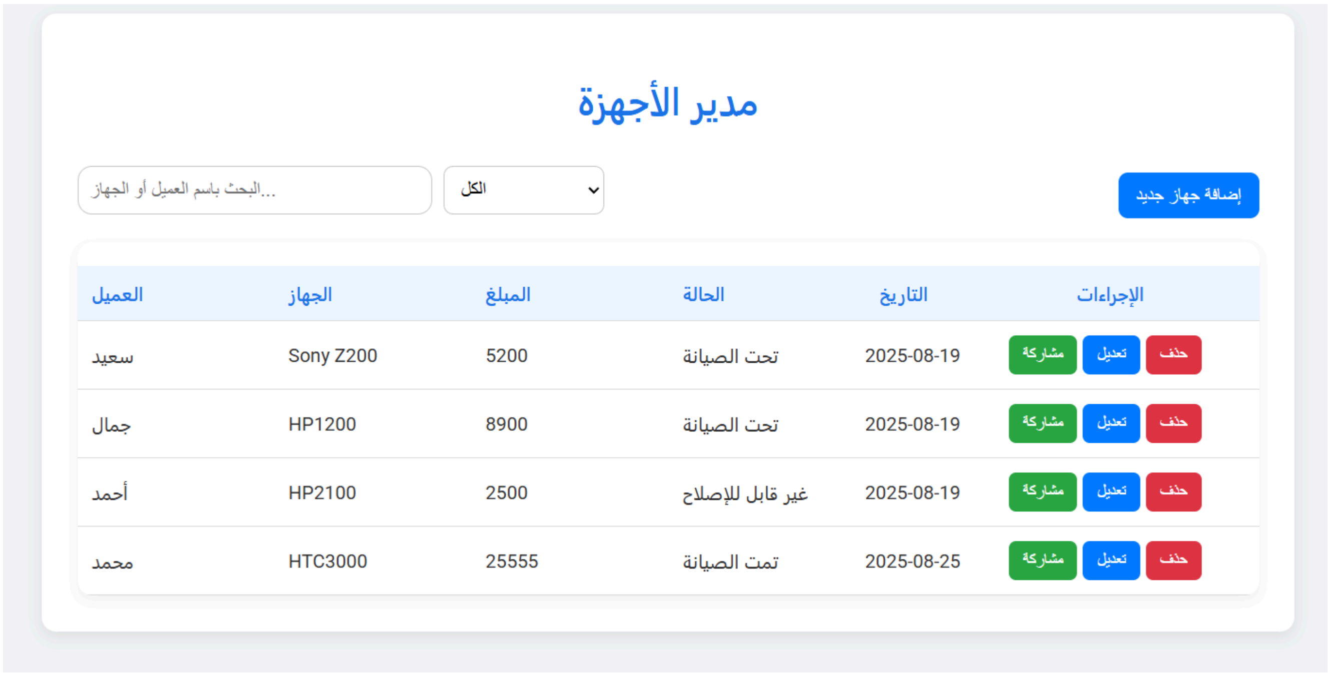


10. User Interface (UI/UX) Design

Login Screen



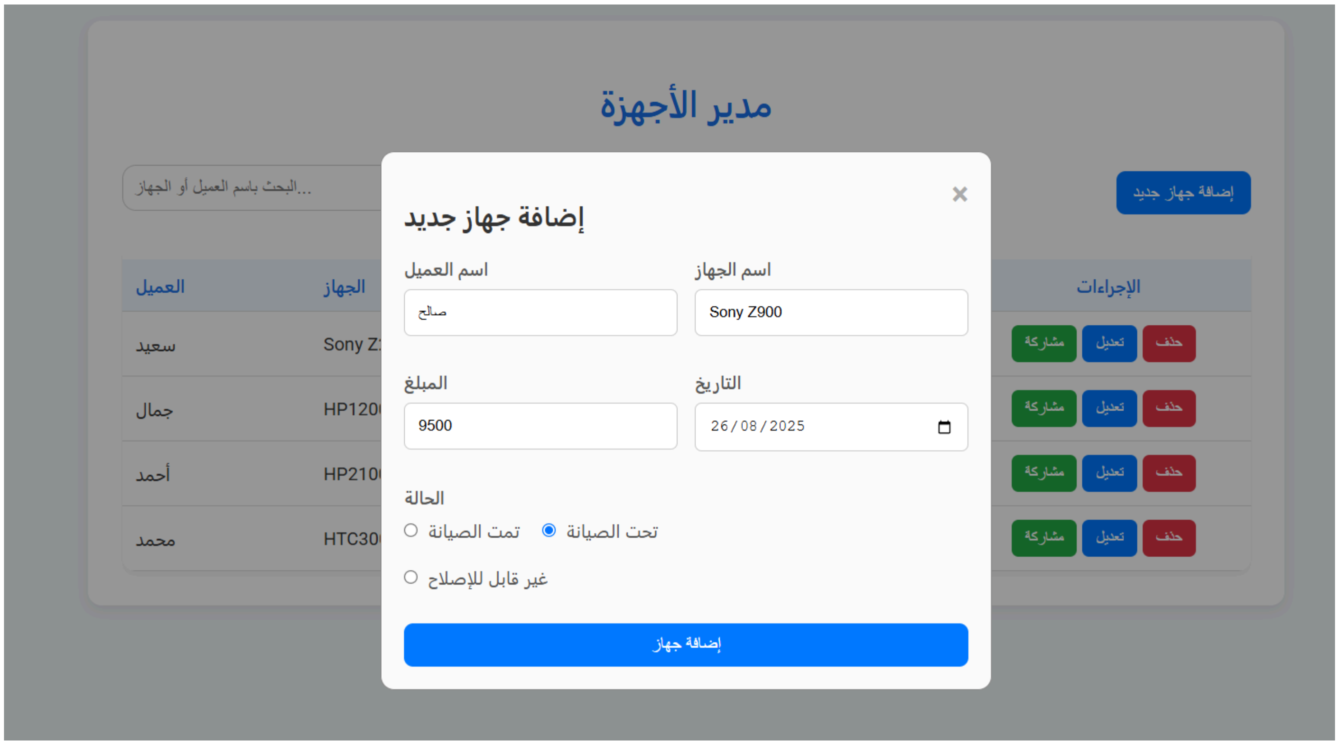
Main Screen



مدير الأجهزة

العميل	الجهاز	المبلغ	الحالة	التاريخ	الإجراءات
سعيد	Sony Z200	5200	تحت الصيانة	2025-08-19	مشاركة تعديل حذف
جمال	HP1200	8900	تحت الصيانة	2025-08-19	مشاركة تعديل حذف
أحمد	HP2100	2500	غير قابل للإصلاح	2025-08-19	مشاركة تعديل حذف
محمد	HTC3000	25555	تمت الصيانة	2025-08-25	مشاركة تعديل حذف

Add New Device



مدير الأجهزة

البحث باسم العميل أو الجهاز...

إضافة جهاز جديد

الإجراءات	الجهاز	العميل
حذف تعديل مشاركة	Sony Z	سعيد
حذف تعديل مشاركة	HP120	جمال
حذف تعديل مشاركة	HP210	أحمد
حذف تعديل مشاركة	HTC30	محمد

×

إضافة جهاز جديد

اسم الجهاز: Sony Z900	اسم العميل: صالح
التاريخ: 26/08/2025	المبلغ: 9500

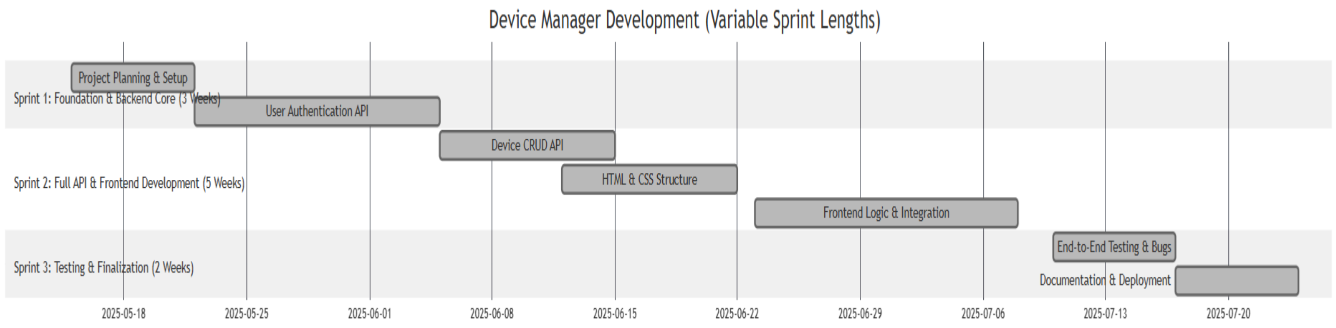
الحالة

[✓ تحت الصيانة] تمت الصيانة

غير قابل للإصلاح

إضافة جهاز

11. GANT Chart



12. Conclusion

The Device Manager web application offers a practical and efficient solution for repair shops seeking to digitize and streamline their device management process. With its intuitive user interface and robust feature set, the application empowers businesses to enhance their operational efficiency and improve customer satisfaction.