Project Report: Service Management Web Application

1. Introduction

The **Service Management Web Application** is designed to facilitate the interaction between **Admins**, **Customers**, and **Service Professionals** in managing service requests for various household services. The application provides role-based navigation, secure access, and functionalities such as service creation, booking, request assignment, and status updates.

2. Objectives

- To provide a platform for **admins** to manage services and monitor service requests.
- To allow **customers** to browse available services, make bookings, and track the status of their service requests.
- To enable **service professionals** to view, accept, and mark service requests as completed.
- To allow the **"Completed"** service status to trigger the display of a **"Closed"** button for the customer.
- To ensure secure access with role-based authentication.

3. Technologies Used

- **Backend**:
    - **Flask**: A Python web framework used to create the RESTful API and handle service management tasks.
    - **Flask-Security**: For token-based authentication and role-based access control.
    - **SQLAlchemy**: An ORM (Object Relational Mapping) for interacting with the SQLite database.
    - **SQLite**: A lightweight database for storing service request and user data.
- **Frontend**:
    - **Vue.js**: A progressive JavaScript framework for building the user interface.
    - **Bootstrap**: A front-end framework for responsive design and styling.
- **Deployment**:
    - The application can be deployed on any platform supporting Python and Vue.js.

4. System Design

4.1 User Roles and Permissions

- **Admin Role**:
    - Create, edit, and delete services.
    - View and manage service requests.
    - Assign professionals to service requests.

- **Customer Role**:
    - Browse available services.
    - Book and cancel service requests.
    - Review completed services.
- **Service Professional Role**:
    - View assigned service requests.
    - Accept or reject service requests.
    - Mark requests as completed when the service is finished.

4.2 Key Features

- **Service Management**: Admins can manage services by adding, editing, and deleting service records. Services include attributes such as name, description, price, and time required.
- **Service Requests**: Customers can book services, cancel bookings, and track the status of their service requests. Service professionals can accept, complete, or reject requests.
- **Authentication**: The system uses **Flask-Security** to secure access to the application, ensuring that only authenticated users can interact with the platform.
- **Service Status Management**: Once a service professional marks a service as completed, the system updates the status, and the customer can see the "Closed" button.
- **Role-based UI**: The user interface is tailored based on the role of the user (Admin, Customer, or Service Professional), allowing them to access only the functionalities relevant to their role.

4.3 Database Structure

- **Users**: Stores information about users, including role, username, email, and password.
- **Services**: Stores details about the services offered, such as service name, description, price, and estimated time.
- **ServiceRequests**: Stores information about customer service requests, including the service ID, customer ID, professional ID, status, and completion date.

5. Workflow

1. **Admin Role**:
    - Admin creates new services and assigns them a price, description, and time duration.
    - Admin can view all service requests and assign professionals to them.
2. **Customer Role**:
    - Customers browse available services and make bookings.
    - They can view the status of their request and cancel if necessary.
    - Upon completion of the service, they see a "Closed" button on the UI if the status is `"completed"`.
3. **Service Professional Role**:

    - Service professionals accept requests assigned to them.
    - They mark the service as completed once the job is done.
    - After completion, the service status is updated to `"completed"`, and the customer can see the update.

6. API Endpoints

- **GET /api/service/{service_id}**: Retrieves details of a specific service request (status, date of completion, etc.).
- **PUT /api/service/{request_id}/complete**: Marks the service request as completed. This is available only to the service professional assigned to the request.
- **GET /api/services**: Fetches the list of all available services.
- **POST /api/service**: Admin endpoint to create a new service.
- **PUT /api/service/{request_id}/close**: Allows the service professional to mark the service as closed.

7. Frontend Design

The frontend is designed using **Vue.js** to ensure a responsive and interactive UI. The UI elements change dynamically based on the user's role and the status of the service requests.

7.1 UI Components:

- **Service List**: A table displaying the services available to customers.
- **Service Form**: A form for admins to create or update services.
- **Service Request Table**: A table displaying customer service requests, their status, and options to cancel or mark as completed.

8. Security

The application uses **Flask-Security** for authentication and authorization. Token-based authentication ensures that only authorized users can access specific resources. Role-based access ensures that users only have access to the functionality appropriate to their role.

9. Challenges and Solutions

- **Challenge**: Handling service status updates and making sure the "Closed" button appears correctly.
    - **Solution**: The service status is updated in the backend (Flask) when the professional marks the service as completed, and the frontend dynamically checks the status to show or hide the "Closed" button.
- **Challenge**: Managing role-based access and ensuring that users can only access their relevant resources.
    - **Solution**: Role-based authorization was implemented with **Flask-Security**, restricting access to certain endpoints based on the user's role.

10. Future Enhancements

- **Real-time Notifications**: Implementing real-time updates using WebSockets to notify customers when their service status changes.
- **Customer Reviews**: Adding a feature for customers to leave reviews and ratings for the services provided.
- **Mobile App**: Expanding the platform to include a mobile application for ease of access.
- **Service History**: Allow customers to view the history of all their service requests.

11. Conclusion

The Service Management Web Application successfully integrates role-based access, secure authentication, and dynamic UI updates based on service statuses. It provides a streamlined platform for managing service requests and offers a user-friendly experience for customers, service professionals, and admins.

Video link
https://www.loom.com/share/3c8eca9dffdc43d3a00e12932c8fcdd2?sid=a564bec8-d7b4-465a-8f24-4406cca14774