MAD-II Project Report

Name: Sai Ashwin Kumar C
Email ID: 21f2000619@ds.study.iitm.ac.in
(Click here to view the video explaining the project)

Description

This project report details the development of a household services web application designed to efficiently manage user roles (Admin, User, Professional) with a focus on seamless operations, including service booking, professional approval, order management, and feedback handling. The application is built using Flask and Python for the backend, and Vue.js for the frontend.

Backend Architecture

Models

1. **Users:** Represents all registered users of the system, including Admins, Professionals, and regular Users. Each user has attributes like name, email_id, password, role, address, city_region, and status.
2. **Inactive Users:** Tracks deactivated users with attributes like user_id, deactivated_date, and reason.
3. **Service Category:** Represents categories of services available in the system (e.g., Plumbing, Electrical). Each category has attributes like category_name and is linked to multiple services.
4. **Service:** Represents individual services under specific categories. Each service has attributes like service_name, service_category_id, price, and time_required_hrs. It is linked to a category and the professionals providing the service.
5. **Professional:** Represents professionals offering services. Each professional is linked to a specific user registered as professional and the services they are qualified to provide.
6. **Order:** Tracks service orders placed by users. Attributes include order_placed_date, order_request_date, order_completion_date, order_status, remarks, and cost_of_order.
7. **User Visit:** Logs user visits to the platform with attributes like user_id and visit_time.
8. **Review:** Records feedback provided by users and professionals. Attributes include review_by_id, review_for_id, ratings, remarks, and review_type (e.g., user-to-professional or professional-to-user).

Relationships

- **Users-Orders:** A user can place multiple orders (one-to-many).
- **Users-Reviews:** Users can give or receive multiple reviews (one-to-many).
- **Users-Visits:** Each user can log multiple visits (one-to-many).
- **Inactive Users-Users:** A one-to-one link tracks deactivated users.
- **Service Category-Services:** A category has multiple services (one-to-many).
- **Services-Professionals:** Services are linked to professionals (many-to-one).
- **Orders-Reviews:** An order can have one review each from the user and the professional
- **Orders-Users:** Orders link to users as creators and professionals as providers (many-to-one).

Frontend Architecture

The frontend of the application is developed using Vue.js, a versatile JavaScript framework for building dynamic user interfaces. The project structure is organized with Vue components stored in the components directory. Key views include:

- **Authentication Views:** Includes signup and login components, allowing users to register and access the platform based on their role (Admin, User, or Professional).

- **Admin Views:** The admin-home component provides a comprehensive interface for managing professionals, services, and orders, along with administrative actions like approving or deactivating users and professionals.
- **User Views:** The user-home component allows regular users to manage their profiles, browse services, place orders, and provide feedback for services received.
- **Professional Views:** The professional-home component offers professionals tools to manage their assigned services, view and update order statuses, and provide feedback for users.
- **Feedback Views:** The feedback component supports a context-sensitive interface for users or professionals to provide reviews and ratings for services or interactions.

The application uses VueRouter for managing navigation between these views, with route guards ensuring secure and role-based access to restricted areas. This structured approach ensures a seamless and secure user experience tailored to the needs of each role.

Notable Features

Client-Server Architecture:

The application leverages a client-server model for seamless functionality:

- **Client-side:** Vue.js ensures a dynamic and responsive user interface, with modular components interacting with the backend via HTTP APIs.
- **Server-side:** Flask manages requests, executes business logic, and interacts with the PostgreSQL database through SQLAlchemy, ensuring robust and secure operations.

Role-Based Authentication:

The application implements JWT-based Token Authentication for secure role-based access control. Tokens are securely stored in the browser's localStorage, facilitating smooth user authentication and authorization.

Database Design:

SQLite serves as the relational database, managing data for users, roles, services, orders, and feedback. SQLAlchemy ORM handles complex relationships and ensures efficient database transactions.

Batch and Scheduled Jobs:

Scheduled tasks are handled using Celery and Redis, with testing via Mailhog. Key tasks include:

- **Daily Reminders** for professionals with pending service requests.
- **Monthly Activity Reports** summarizing user orders in HTML format.
- **CSV Reports** for detailed professional insights on admin’s request.

User Wise capabilities:

1. **Admin Capabilities:**
   - **Manage users and professionals**: View all users, review and approve/reject professional proof of experience, activate/deactivate accounts.
   - **Oversee platform activity:** Monitor past and ongoing orders.
   - **Manage services:** Add, update, or delete service categories and offerings.
2. **Customer Capabilities:**
   - **Explore and book services:** View region-specific services, select professionals based on ratings, and book services.
   - **Manage orders:** Reschedule or cancel pending orders and provide feedback post-service.
   - **Profile management:** Update personal details and deactivate their account.
3. **Professional Capabilities:**
   - **Service selection:** Choose a specific service to work with.
   - **Manage orders:** Accept/reject customer requests and view customer ratings.
   - **Feedback:** Provide feedback on customers post-service.
   - **Profile management:** Edit personal details and deactivate their account.

Conclusion

This project successfully addresses the challenges of managing a service-oriented platform by providing robust functionality for each role, ensuring data integrity, and delivering a seamless user experience. For further enhancements, considerations might include improving authentication and authorization mechanisms, enhancing frontend design, implementing more robust error handling, and scaling the system for increased user traffic.