**PROJECT REPORT**

Modern Application Development II

Household Services App

**Author:** Maan Kaur

**Email:**[22f3002399@ds.study.iitm.ac.in](mailto:22f3002399@ds.study.iitm.ac.in)

**Project Statement :**

It is a multi-user app (requires one admin and other service professionals/ customers) which acts as platform for providing comprehensive home servicing and solutions.

**Technologies Used :**

* Python
* Flask
* Flask SQLAlchemy
* Vue.js
* CSS
* SQLite for data storage
* Redis

**Database Schema :**

The Database schema comprises following Classes/Entities:

1. **User** - Stores user account details with attributes such as id, email, password, active status, and role ('admin', 'professional', or 'customer'). This is the base model for authentication and authorization.
2. **Professional** - Stores service provider information with fields such as professional\_id, user\_id (Foreign Key to User table), name, service\_type, mobile, experience, pin code, creation date, description, and flags for approval and blocking status. Represents service providers in the system.
3. **Customer** - Stores service consumer information with fields such as customer\_id, user\_id (Foreign Key to User table), name, address, pin code, creation date, and blocked status. Represents users who request services.
4. **Service** - Stores available service categories with fields such as id, name, price, time\_req (time required), description, and service\_type. This defines the types of services that professionals can provide.
5. **ServiceRequest**- Stores service booking information with fields such as id, service\_id (Foreign Key to Service table), customer\_id (Foreign Key to Customer table), professional\_id (Foreign Key to Professional table), request date, completion date, status ('requested', 'assigned', 'closed'), and remarks. Tracks the full lifecycle of service requests.
6. **Review** - Stores customer feedback with fields such as id, service\_request\_id (Foreign Key to ServiceRequest table), customer\_id (Foreign Key to Customer table), rating, comment, and date created. Enables quality assessment of services provided.

**Architecture Design :**

**Back-end Architecture**

The Household Services application's back-end architecture is designed with modularity, security, and performance in mind.

● **Modular Structure**: The application employs a modular architecture separating core functions like user management, service booking, and professional management. This approach allows for independent development and maintenance of each module, enhancing overall application flexibility.

● **Model-View-Controller (MVC) Pattern**: The backend follows the MVC pattern with Flask providing the controller layer, SQLAlchemy models for data representation, and RESTful APIs serving the view layer, ensuring clean separation of concerns.

● **Database Design**: The application uses SQLite with a well-defined schema consisting of six primary models (User, Professional, Customer, Service, ServiceRequest, and Review), establishing clear relationships for data integrity.

● **JWT Authentication**: Security is implemented through JWT token-based authentication, ensuring secure and stateless authorization across user roles (admin, professional, and customer).

● **Redis for Caching**: The application leverages Redis for caching frequently accessed data like service listings and professional profiles, significantly improving response times and reducing database load.

● **Role-Based Access Control (RBAC)**: The system implements comprehensive role-based permissions ensuring users only access appropriate functionality based on their designated role.

**Client-Side Design**

The Household Services application's client-side is developed with usability and responsiveness as primary goals:

● **Component-Based Architecture with Vue.js**: The frontend utilizes Vue.js with CLI to create modular, reusable components that enhance development efficiency and maintenance.

● **Responsive UI with Bootstrap**: Bootstrap powers the responsive design, ensuring consistent user experience across various devices and screen sizes while maintaining the application's accessibility.

● **Dashboard Interfaces**: Custom dashboards are implemented for each user role (admin, professional, customer) with role-specific functionalities and data visualization.

● **Search and Filter Capabilities**: Advanced search functionality allows customers to find services by location, type, and professional ratings, while admins can efficiently manage users through similar filtering mechanisms.

**Video Link :**

https://drive.google.com/file/d/1ZQ27UeYhiGGzBv7LRc0nSpthYtUTnjyr/view?usp=sharing