Modern Application Development – II Project Report
(Household Services Application – V2)

Student Details

Name: Siddharth Ghosh
Student Email ID: 21f3002603@ds.study.iitm.ac.in
Roll Number: 21f3002603

Project Details

Question Statement
Household Services Application – V2 (H.S.A.) **–** 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.

Approach to the Problem

To build the **Household Services Application (H.S.A.)**, I adopted a modular approach with a clear separation of concerns between the frontend and the backend. The following modular approach was adopted:

Frontend Development: The application features a **Vue.js 2** frontend, ensuring a responsive and interactive Single Page Application (SPA) experience. **Vue Router** handles navigation, while **Vuex** is used for state management.

Backend Development: The backend is developed using **Flask**, focusing on a RESTful architecture. **Flask-RESTful** defines and manages APIs that act as the communication layer between the frontend and backend. **Flask SQLAlchemy** enables object-relational mapping (ORM) for interacting with the database.

Security and Role Management: **Flask Security** Too ensures secure authentication and implements **role-based access control (RBAC)** with predefined roles, such as Admin, Service Professional, and Customer. This approach provides a secure login mechanism and enforces access restrictions based on user roles.

Task Automation: **Celery** is used for scheduling and executing background tasks, such as:
Sending **daily email reminders**
Generating and emailing **monthly activity reports**
Enabling admins to download CSV files

Caching for Performance Optimization: Caching mechanisms are implemented in the backend using **cache.get** and **cache.set** which minimizes redundant database queries by storing serialized responses temporarily, ensuring faster responses for frequently requested resources, such as services.

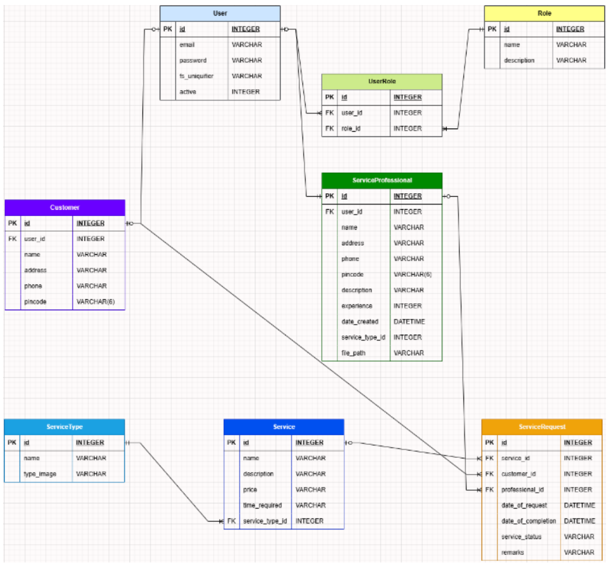
Technology Stack

Front-end**:** Built with **HTML5** for structure, **CSS** and **Bootstrap 5.3** for styling and layout, **JavaScript (ES6)** for dynamic interactions, and **VueJS 2** for creating a Single Page Application with **Vuex** for state management and **Vue Router** for navigation. **Chart.js** was used for data visualization on the admin summary page.

Back-end**:** Developed using **Flask (3.0.3)** with **Flask-SQLAlchemy (3.1.1)** for database models, **Flask-RESTful (0.3.10)** for API resources, and **Jinja2 (3.1.4)** for rendering monthly reports sent via email. Other tools include **Flask-Caching (2.3.0)** for caching, **Flask-Security-Too (5.5.2)** for authentication and RBAC, **Celery (5.4.0)** for handling background tasks, and **Flask-Excel (0.0.7)** for exporting Service Request CSVs. **Redis (5.2.0)** acts as the broker for Celery tasks and supports caching. Emails are sent using **smtplib** and tested via the **MailHog** server (port 1025 for sending, 8025 for receiving).

Database**: SQLite3**.

ER Diagram (Crow’s Foot Notation)



API Resource Endpoints

1. UserAPI

- **Endpoint:** /api/users/<int:user_id>/<string:action>
- **Methods:** POST
- **Description:** Approve or block a user based on the action parameter (approve or block). Accessible to admins only.

2. ServiceAPI

- **Endpoint:** /api/services/<int:service_id>
- **Methods:** GET, POST, PUT, DELETE
- **Description:**
  - GET: Retrieve a service by ID (supports caching).
  - POST: Create a new service.
  - PUT: Update an existing service by ID.
  - DELETE: Delete a service by ID.

3. ServiceListAPI

- **Endpoint: /api/services**
- **Methods: GET**
- **Description: Fetch all available services (cached for 10 seconds).**

4. CustomerAPI

- **Endpoints:**
  - /api/customers
  - /api/customers/<int:customer_id>
- **Methods: GET, DELETE**
- **Description:**
  - GET: Retrieve a specific customer by ID, a customer by user_id, or all customers.
  - DELETE: Delete a customer by ID (also deletes the associated user).

5. ServiceProfessionalAPI

- **Endpoints:**
  - /api/professionals
  - /api/professionals/<int:professional_id>
- **Methods: GET, DELETE**
- **Description:**
  - GET: Retrieve a service professional by ID, by user_id, or all professionals.
  - DELETE: Delete a professional by ID (also deletes the associated user).

6. ServiceTypeAPI

- **Endpoints:**
  - /api/service-types
  - /api/service-types/<id>
- **Methods: GET, POST, PUT, DELETE**
- **Description:**
  - GET: Retrieve service types (by ID or all).
  - POST: Create a new service type.
  - PUT: Update a service type by ID.
  - DELETE: Delete a service type by ID.

7. ServiceRequestAPI

- **Endpoints:**
  - /api/service-requests
  - /api/service-requests/<id>
- **Methods: GET, POST, PUT, DELETE**
- **Description:**
  - GET: Retrieve a service request by ID or all requests.
  - POST: Create a new service request.
  - PUT: Update a service request (status, rating, remarks).
  - DELETE: Delete a service request by ID.

Presentation Video

https://drive.google.com/file/d/1ybJuOlHt7MKD3uE_qmKafU2gdblRWy07/view?usp=sharing