Project Report
Modern Application Development – 1
Household Services Application Version 1

Author Details
Name – Shahbaaz Singh
LinkedIn – https://www.linkedin.com/in/shahbaazsingh/

As someone passionate about programming and data science, working on the Household Services Application Version 1 for the Modern Application Development 1 course by IIT Madras was an enriching experience. While there were challenges along the way, overcoming them brought a sense of accomplishment and further fueled my enthusiasm for developing practical and impactful applications.

Description –
In this project, we developed a multi-user *Household Services Application* using Flask, Jinja2 templates, Bootstrap, and SQLite. The app provides a platform for managing and delivering home services. It includes three roles:

- **Admin**: Handles service creation, user monitoring, service professional approval, and user blocking for fraudulent activities.
- **Service Professionals**: Accept/reject service requests based on their expertise and provide services.
- **Customers**: Book service requests, search for services, and leave reviews.

Core functionalities include user authentication, service creation and management, service request handling, search features, and the ability for service professionals to take actions on requests. The application also supports CRUD operations for services and service requests.

Technologies Used

1. **Flask**: Used for building the web application and creating RESTful APIs.
2. **Flask-SQLAlchemy**: An extension of Flask, used for database connections and ORM (Object Relational Mapping).
3. **Flask-Migrate**: Handles database migrations seamlessly.
4. **Flask-Login**: Used for user authentication, session handling, and login/logout management.
5. **Flask-RESTful**: Simplifies the creation of REST APIs for backend services.
6. **Flask-WTF**: Facilitates form handling with validation and Flask integration.
7. **Jinja2 Templates**: Provides dynamic HTML content rendering.
8. **Bootstrap**: A framework for front-end styling and creating responsive designs.
9. **SQLAlchemy**: Core ORM for database operations.
10. **matplotlib**: Used for creating visualizations and charts within the application.
11. **datetime** (from Python's standard library): Used for managing timestamps and date-related functionalities.
12. **os** (from Python's standard library): Used to manage file paths, particularly for handling the documents directory.
13. **request** (from Flask): Extracts and processes data from incoming HTTP requests.

14. **wraps** (from functools): Used to create a *no-cache* decorator to prevent browser caching of specific pages.
15. **BytesIO** (from io): Facilitates the in-memory handling of binary data streams, specifically for image processing.
16. **base64** (from Python's standard library): Used for encoding images into base64 format for embedding them in HTML.
17. **Werkzeug**: Handles routing, debugging, and application security.
18. **WTForms**: Assists with form handling and validation.

Architecture and Structure:

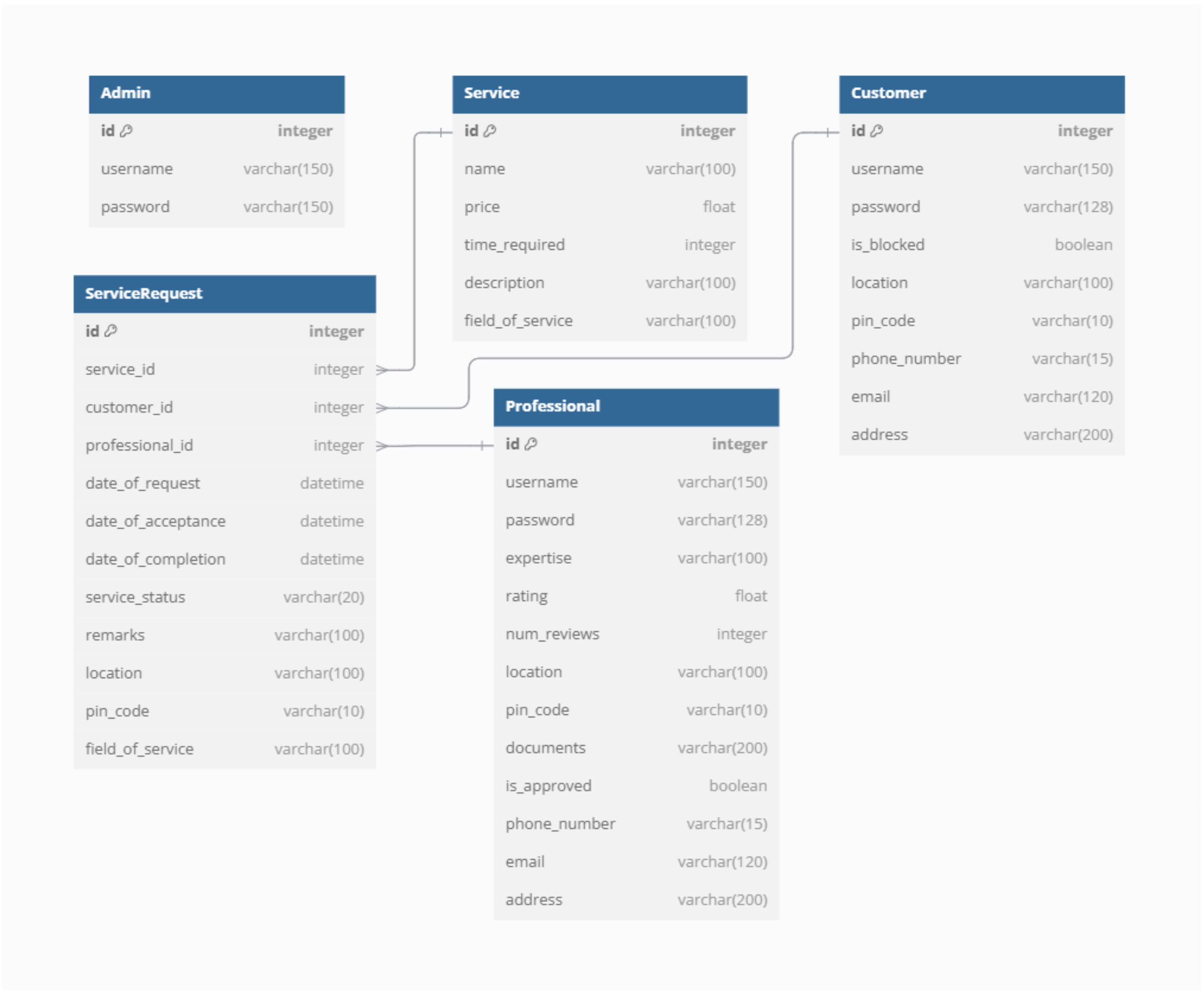
The **MAD1** folder contain another folder **Household_Services_Application**. This folder contains the folders **Household Services Application Version 1**, **Include, share** and **var.** The **Household Services Application Version 1** folder is structured into several key components for efficient organization and functionality. Inside the "Household Services Application Version 1" folder, the **app** folder contains subdirectories for **static** files, such as **uploads** for professional documents, and **templates** which are further divided into **admin**, **professional**, and **customer** folders, each containing specific HTML files. The application logic resides in **routes.py**, **models.py**, **forms.py**, and initialization files such as **init.py**, alongside important documents like **Important Commands.txt** and **curl_commands.txt**. The "Household Services Application Version 1" folder also includes a **Migrations** folder, a **run.py** file for starting the app, and a **Project_Report.pdf** for documentation. The **var** folder contains the **app-instance** and **app.db**, which stores the SQLite database. Additional system files are found in the **Share** and **Include** folders.

DB Schema Design:

The database schema is designed with the following relationships and structures:

- Admin: The admin has root privileges to manage users and services. Admin has unique attributes like *username* and *password* with *id* as the primary key.
- Customer: A customer can create multiple service requests, establishing a one-to-many relationship between the Customer and ServiceRequest tables. The *id* is the primary key, and the table includes attributes like *username*, *email*, and *location*, etc.
- Professional: A professional can handle multiple service requests, creating a one-to-many relationship between the Professional and ServiceRequest tables. The table tracks details like *expertise*, *rating*, and *documents*, etc.
- Service: Each service can have multiple service requests associated with it, forming a one-to-many relationship between the Service and ServiceRequest tables. It includes attributes like *name*, *price*, and *description*, etc.
- ServiceRequest: This table acts as a bridge linking services, customers, and professionals. It includes foreign keys referencing the service, customer, and professional tables, along with attributes like *date_of_request*, *status*, and *remarks*, etc

For more details on the database schema, relationships, and other components, please refer to the image below:



API Design

- **User_api (Registration)**: Implemented using GET and POST methods. The GET method retrieves user details, including their associated services and service statuses from the database. The POST method is used to create new users, storing their information such as name, email, and role (customer or professional) in the system.

- **Admin_api**: Implemented using GET, POST, PUT and DELETE methods. The GET method is used by admins to fetch customer and professional details. The POST method allows admins to add new services, by specifying price, description, time required, field of service, name. The PUT method is used to update professional and customer status of being approved, blocked, etc., updating services, while the DELETE method enables admins to delete services.

- **Customer_api**: Implemented using GET, POST, and PUT methods. The GET method retrieves a customer's profile, service requests, and any ongoing services they have requested. The POST method allows customers to submit new service requests, specifying details like service name, pincode, and location. The PUT method is used by customers to update the status of a service request or modify any details of the service, such as rescheduling or providing additional information.

- **Professional_api**: Implemented using GET and POST methods. The GET method retrieves a list of professionals, and their specific service expertise. The POST method allows professionals to accept and reject services. The PUT method is used by professionals to update the status of a service request (e.g., "Accepted," "Completed," etc.) or to mark a service requests as completed.

- **Session_api**: Implemented using GET and POST methods. The GET method is used to fetch session details for a user, such as their login status and role. The POST method is used to manage user login and logout processes, securely establishing and clearing user sessions with token-based authentication.

Wireframe:

The controllers that were used:

@main.route('/')
@main.route('/register', methods=['GET', 'POST'])
@main.route('/admin/login', methods=['GET', 'POST'])
@main.route('/admin/dashboard')
@main.route('/admin/customer_info', methods=['GET', 'POST'])
@main.route('/admin/professional_info', methods=['GET', 'POST'])
@main.route('/admin/services')
@main.route('/admin/approve/<int:id>', methods=['POST'])
@main.route('/admin/unapprove/<int:id>', methods=['POST'])
@main.route('/admin/block_user/<int:id>', methods=['POST'])
@main.route('/admin/unblock_user/<int:id>', methods=['POST'])
@main.route('/admin/create_service', methods=['GET', 'POST'])
@main.route('/admin/update_service/<int:id>', methods=['GET', 'POST'])
@main.route('/admin/delete_service/<int:id>', methods=['POST'])
@main.route('/customer/login', methods=['GET', 'POST'])
@main.route('/customer/dashboard')
@main.route('/customer/create_request', methods=['GET'])
@main.route('/customer/search_services', methods=['GET'])
@main.route('/customer/request_service', methods=['POST'])
@main.route('/customer/service_requests')
@main.route('/customer/update_request/<int:id>', methods=['POST'])
@main.route('/customer/close_request/<int:id>', methods=['POST'])
@main.route('/professional/login', methods=['GET', 'POST'])
@main.route('/professional/dashboard')
@main.route('/professional/pending_requests')
@main.route('/professional/accepted_requests')
@main.route('/professional/accept_service/<int:id>', methods=['POST'])
@main.route('/professional/reject_service/<int:id>', methods=['POST'])
@main.route('/professional/update_service_status/<int:id>', methods=['POST'])
@main.route('/admin/logout')
@main.route('/customer/logout')
@main.route('/professional/logout')

API Endpoints:

('/api/login', endpoint='login') # Authentication

('/api/admins', endpoint='admins') # For POST
('/api/admins/<int:admin_id>', endpoint='admin') # For GET, PUT, DELETE

('/api/customers', endpoint='customers') # For POST
('/api/customers/<int:customer_id>', endpoint='customer') # For GET, PUT, DELETE

('/api/professionals', endpoint='professionals') # For POST
('/api/professionals/<int:professional_id>', endpoint='professional') # For GET, PUT, DELETE

```
('/api/services', endpoint='services')  # For POST (create)
('/api/services/<int:service_id>', endpoint='service')  # For GET, PUT, DELETE

('/api/requests', endpoint='service_requests')  # For POST
('/api/requests/<int:request_id>', endpoint='service_request')  # For GET, PUT, DELETE
```

Video:
Google Drive Links:

Short Demo Video –
https://drive.google.com/file/d/1vbFJmSOv7FcIMYXn21FQkzHt5sRzHkzf/view?usp=drive_link

Long Demo Video –
https://drive.google.com/file/d/1ZJLlAVfdWpCmIDZWAm_DMjAjdyfjmplX/view?usp=drive_link