Project Report

Author

- **Name**: SADIYA MAHEEN SIDDIQUI
- **Roll Number**: 23f3001208
- **Email**: 23f3001208@ds.study.iitm.ac.in

I am a student passionate about learning modern web application development and exploring full-stack frameworks. This project gave me a hands-on experience in developing a multi-user application with various roles and functionalities.

Description

The project aims to create a multi-user application, *A-Z Household Services*, which connects customers, service professionals, and administrators. The application enables customers to book household services, professionals to manage service requests, and administrators to monitor and control the platform.

Technologies Used

- **Flask**: For backend development and routing.
- **Jinja2**: To dynamically generate HTML templates.
- **Bootstrap**: For responsive design and user interface styling.
- **SQLite**: For database management.
- **ChartJS**: To render graphical analytics (optional feature).

Purpose:

The chosen technologies are lightweight and easy to integrate, making them ideal for local machine demonstrations. Flask's modular architecture and extensions like Flask-SQLAlchemy provide seamless interaction between database models and views.

DB Schema Design

Database Tables:

1. Users Table

- **Fields:**
  - id: Integer, Primary Key.
  - username: String, Unique, Not Null.
  - password: String, Not Null.
  - full_name: String, Not Null.
  - address: String, Not Null.
  - pincode: Integer, Not Null.
  - role: String, Not Null (admin, customer).
- **Constraints:**
  - username is unique.
  - Validates role to ensure it matches predefined values (admin or customer).

2. Professional Table

- **Fields:**
  - id: Integer, Primary Key.
  - username: String, Unique, Not Null.
  - password: String, Not Null.
  - full_name: String, Not Null.
  - service_name: String, Not Null (Type of service offered by the professional).
  - experience: Integer, Not Null (Years of experience).
  - address: String, Not Null.
  - pincode: Integer, Not Null.
  - status: String, Not Null (Approved, Rejected, In Progress).
- **Constraints:**
  - username is unique.

3. Services Table

- **Fields:**
  - id: Integer, Primary Key.
  - name: String, Unique, Not Null (Service name).
  - description: String, Optional (Description of the service).
  - base_price: Float, Not Null.
- **Constraints:**
  - name is unique.

4. Service Requests Table

- **Fields:**
  - id: Integer, Primary Key.
  - service_id: Integer, Foreign Key referencing Services.id, Not Null.
  - customer_id: Integer, Foreign Key referencing Users.id, Not Null.
  - professional_id: Integer, Foreign Key referencing Professional.id, Optional (Assigned professional).
  - date_requested: DateTime, Not Null (Default to current date).
  - date_completed: DateTime, Optional.
  - status: String, Not Null (Requested, Accepted, Rejected, Closed).
- **Constraints:**
  - service_id is a foreign key referencing Services.id.
  - customer_id is a foreign key referencing Users.id.
  - professional_id is a foreign key referencing Professional.id.

5. Reviews Table

- **Fields:**
  - id: Integer, Primary Key.
  - service_id: Integer, Foreign Key referencing Services.id, Not Null.
  - customer_id: Integer, Foreign Key referencing Users.id, Not Null.
  - professional_id: Integer, Foreign Key referencing Professional.id, Not Null.

    - rating: Integer, Not Null.
    - comment: String, Optional (Customer review of the service).
- **Constraints:**
    - service_id is a foreign key referencing Services.id.
    - customer_id is a foreign key referencing Users.id.
    - professional_id is a foreign key referencing Professional.id

Design Rationale:
This schema ensures normalization, modularity, and scalability. Each table has well-defined foreign keys to maintain relationships between entities.

API Design
Implemented APIs:

1. /api/services: Create, update, and delete services.
2. /api/users: Manage user authentication and details.
3. /api/requests: CRUD operations for service requests.

APIs return JSON responses and were implemented using Flask-RESTful. This ensures smooth interaction with the frontend or potential third-party integrations.

Architecture and Features
Project Organization:

- **Controllers**: Defined in views.py to manage routes and business logic.
- **Templates**: Located in the /templates directory, using Jinja2 for dynamic rendering.
- **Static Files**: CSS, JavaScript, and images are stored in /static.
- **Database Models**: Defined in models.py using Flask-SQLAlchemy.

Implemented Features:

1. **Role-based Authentication**: Separate login flows for admin, customers, and professionals.
2. **Admin Dashboard**: Manage users, services, and service requests.
3. **Service Management**: Customers can search, book, and review services; professionals can manage assigned requests.
4. **Responsive UI**: Using Bootstrap for cross-device compatibility.

Video
https://drive.google.com/file/d/18YcOeENESogp0GHZj9YcvEjGjlJthtPz/view?usp=sharing