PROJECT DOCUMENTATION

Author

Name: Astha Shukla
Roll No.: 22f1000725
Email: 22f1000725@ds.study.iitm.ac.in

Description

Introducing ServiceSphere – a Household services App, a lightweight service booking app designed for people to book services easily. With it you can easily explore and enjoy various services from trained and verified professionals. The app also lets you rate and add remarks the services you avail for to improve the service quality.
You can register as a user who wants to avail services or as a professional who will appear on the platform.
Designed to be user-friendly, It offers a seamless experience for both users and professionals with constant feedback and monitoring by admin.

Technologies Used

The application harnesses Flask, a potent web framework, for backend development. Key Flask extensions like Flask-Security for user authentication and Flask-SQLAlchemy for database management are seamlessly integrated. For frontend development, VueJS is employed, leveraging CDN for efficient resource loading, while Bootstrap enhances visual aesthetics. Vue-router ensures smooth navigation, while Pinia effectively manages state. Performance optimization is achieved through Redis for caching, supplemented by Flask-Caching. flask_mail facilitates email sending, while flask_cors enables cross-origin resource sharing. HTTP requests are managed using the 'requests' library. Additionally, Redis combined with Celery streamlines batch job handling.

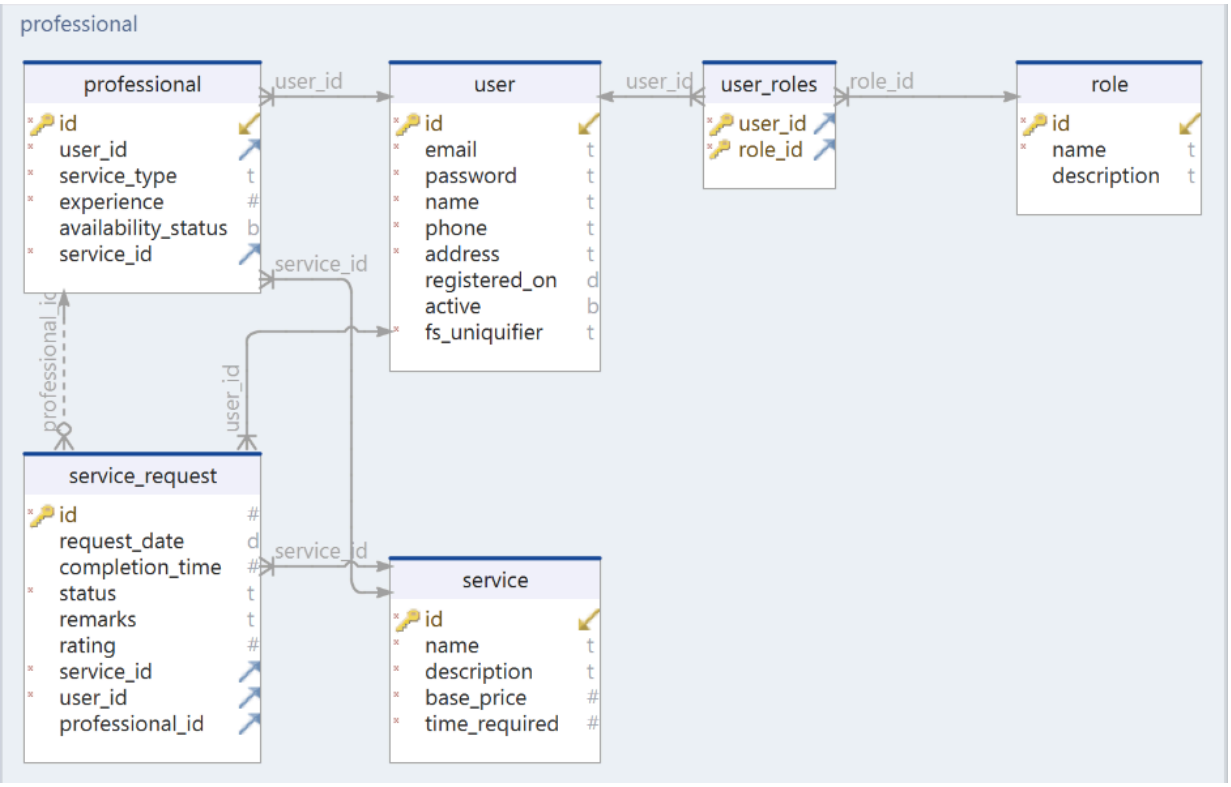
API Design

The Flask-based API architecture is organized for efficient functionality management through distinct endpoints. The **Services API** handles operations like adding, updating, and retrieving service information. The **Users API** manages user creation and retrieval, emphasizing role-specific actions. The **Service Request API** supports handling service request creation, updates, and administration. The **Professionals API** focuses on managing professional profiles and their availability. The **Admin Summary API** aggregates system statistics for admins, while **User and Professional Summary APIs** provide personalized insights. Using Flask-RESTful, Flask-SQLAlchemy, and caching, the architecture ensures streamlined interactions and robust data handling in the service management platform.

db Schema Design

The database model for the service management system includes tables for User, Role, User_Roles, Professional, Service, and Service_Request. Users are linked to roles through User_Roles, while professionals are associated with users and offer services. Services store details like name, description, price, and time. Service_Request tracks user requests, including status, remarks, and ratings.
The schema links users, roles, professionals, services, and requests, enabling efficient

service management using Flask-SQLAlchemy.



Architecture and Features

In the root folder we have app.py, config.py, initial_data.py, a readme.md file, a requirements.txt file, the instance folder with the SQLite database file, in application folder we have the views.py, models.py, resources.py, worker.py. tasks.py, etc, the template folder which contains index.html and other html files. The static folder which contains two files: index.js and router.js and components which contain the JS templates.

1. **views.py :** This file acts as a controller for the app and has all the routes to different pages.
2. **models.py :**This file contains a Python code for defining database models using SQLAlchemy and Flask-Login for the music streaming web application..
3. **resources.py :** All the api endpoints have been created and managed here.
4. **mail_service.py:** For Sending Monthly Report using Celery.
5. **workers.py :** For managing Cache using redis.
6. **app.**py: this file runs the flask application.
7. **config.**py: configures all the requirements.

There's a registration form for users with proper front-end and back-end validation. There's a login form for customer, professional and admin. The Search bar on top helps the user, professional and admin to search for different service requests. The app has CRUD features for Service and Service request. Customers can request/close their service requests. Users/Professionals can be flagged or deleted by the admin if they breach any company Policy.

Presentation Video : [Mad2_Presentation]