

ServiceHivePro: Professional Service Marketplace

Project Report

Student Details

- Name: Ashutosh Singh
- Roll Number: 23f2001233
- Course: MAD I Sept 2024

Project Details

Question Statement

The goal of this project was to develop a web-based platform called ServiceHivePro that connects customers with professional service providers. The platform aims to streamline the process of finding, booking, and reviewing various services, while also providing an efficient management system for service professionals and administrators.

Approach to the Problem Statement

To address the problem statement, we adopted the following approach:

1. User Roles and Authentication: Implemented a robust user authentication system with role-based access control (RBAC) to distinguish between customers, service professionals, and administrators.
2. Service Management: Created a flexible service model that allows administrators to add and manage various service types, including details such as description, base price, and estimated time required.
3. Service Request Workflow: Developed a comprehensive workflow for service requests, from initial customer booking to completion and review.
4. Review and Rating System: Implemented a system for customers to leave reviews and ratings for service professionals, enhancing trust and quality control on the platform.
5. Admin Dashboard: Created an administrative interface for user management, service oversight, and review moderation.
6. Responsive Design: Ensured the application is accessible and functional across various devices and screen sizes.
7. Data Security and Privacy: Implemented secure password hashing and protected routes to ensure user data safety and privacy.

Frameworks and Libraries Used

1. Backend Framework:

1. Flask: A lightweight WSGI web application framework in Python, chosen for its simplicity and flexibility.

2. Database and ORM:

1. SQLAlchemy: An SQL toolkit and Object-Relational Mapping (ORM) library for Python, used for database operations and management.

2. Flask-SQLAlchemy: An extension for Flask that adds support for SQLAlchemy, simplifying database integration.

3. Authentication and Security:

1. Flask-Login: Provides user session management for Flask, handling the common tasks of logging in, logging out, and remembering users' sessions.

2. Werkzeug: A comprehensive WSGI web application library, used for password hashing and security features.

4. Form Handling:

1. Flask-WTF: An extension for Flask that integrates WTForms for form handling and CSRF protection.

2. WTForms: A flexible forms validation and rendering library for Python web development.

5. Frontend:

1. Jinja2: A modern and designer-friendly templating engine for Python, used for rendering HTML templates.

2. Bootstrap: A popular CSS framework for developing responsive and mobile-first websites, used for styling and layout.

6. Date and Time Handling:

1. pytz: A Python library for accurate timezone calculations, used for handling different timezones in the application.

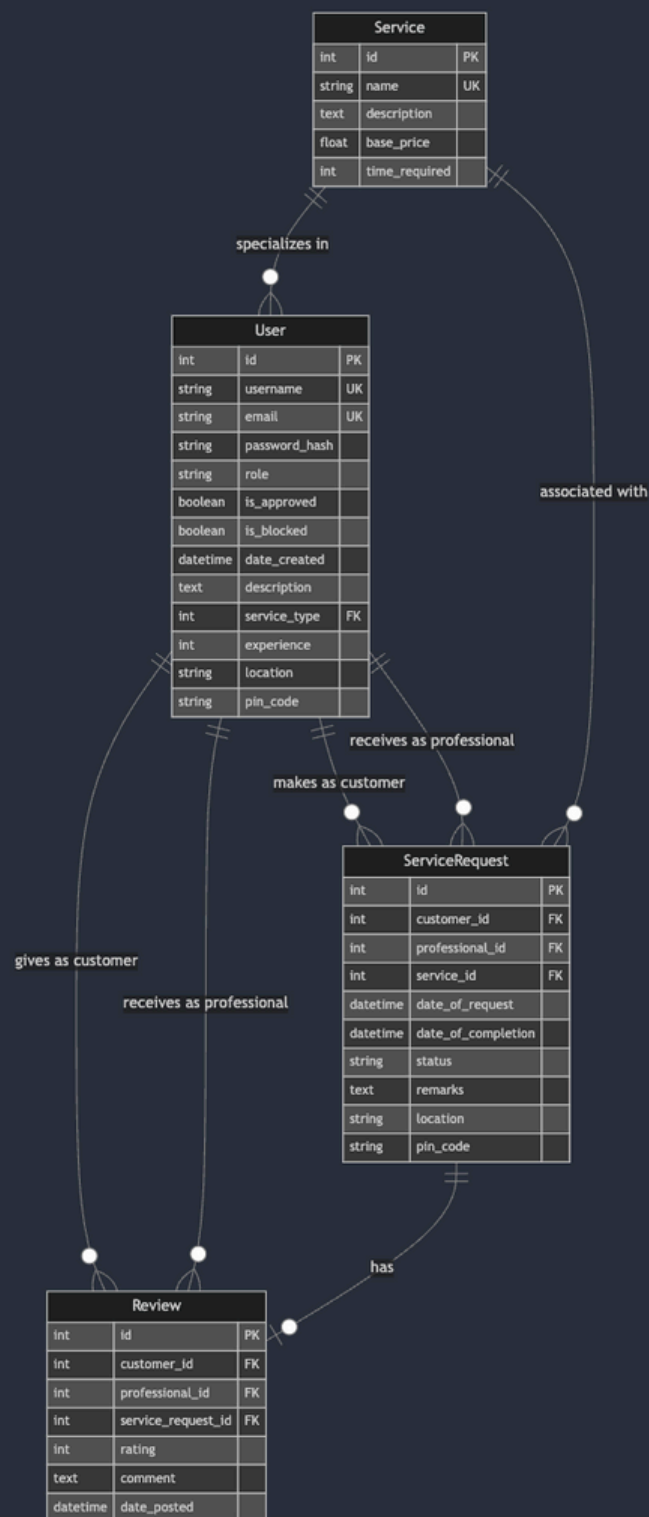
7. Development and Deployment:

1. python-dotenv: A Python library that allows you to specify environment variables in traditional UNIX-like ".env" (dotenv) files, used for configuration management.

2. Flask-Migrate: An extension that handles SQLAlchemy database migrations for Flask applications

using Alembic.

ER Diagram of the Database



API Resource Endpoints

While our application primarily uses server-side rendering with Flask and Jinja2 templates, we have implemented some API endpoints for specific functionalities. Here are the main API resource endpoints:

1. User Management:

1. POST `/api/register`: Register a new user
2. POST `/api/login`: Authenticate a user and return a session token
3. GET `/api/user/<int:user_id>`: Retrieve user details
4. PUT `/api/user/<int:user_id>`: Update user details

2. Service Management:

1. GET `/api/services`: Retrieve all services
2. POST `/api/services`: Create a new service (admin only)
3. GET `/api/services/<int:service_id>`: Retrieve details of a specific service
4. PUT `/api/services/<int:service_id>`: Update a service (admin only)
5. DELETE `/api/services/<int:service_id>`: Delete a service (admin only)

3. Service Requests:

1. POST `/api/service-requests`: Create a new service request
2. GET `/api/service-requests/<int:request_id>`: Retrieve details of a specific service request
3. PUT `/api/service-requests/<int:request_id>`: Update the status of a service request

4. Reviews:

1. POST `/api/reviews`: Submit a new review
2. GET `/api/reviews/<int:professional_id>`: Retrieve reviews for a specific professional

These API endpoints return JSON responses and use appropriate HTTP status codes to indicate the success or failure of operations.

Drive Link of the Presentation Video :

https://drive.google.com/file/d/1q51Md4n3M5AThHOrJouqWseSBXzQ-ImN/view?usp=drive_link

CODE



app

> routes

static

> css

> images

> js

templates

admin

dashboard.h...

edit_service....

manage_ser...

manage_use...

professional...

professional...

auth

login.html

register.html


customer

dashboard.h...


request_serv...

review_servi...

professional

 dashboard.h...

 update_profi...


 base.html

 index.html

 __init__.py

 forms.py

 models.py

>  instance

 config.py

 flask_app.py

 run.py