Author

Malipeddi Sudeep

22f1001895

22f1001895@ds.study.iitm.ac.in

Description

Household Services Application is a multi-user application where there are 3 roles (Admin, Professional, Customer). This application is used to allow customers to book services and professionals can accept or reject the request of the customer based on their schedule

Technologies used

Backend

Flask, Flask-RESTful, Flask-JWT-Extended, Flask-SQLAlchemy, Flask-Migrate, Flask-CORS, Werkzeug

SQLite: Database systems

Frontend

Vue.js 3, Vue Router, Pinia, Axios, Bootstrap 5, Chart.js

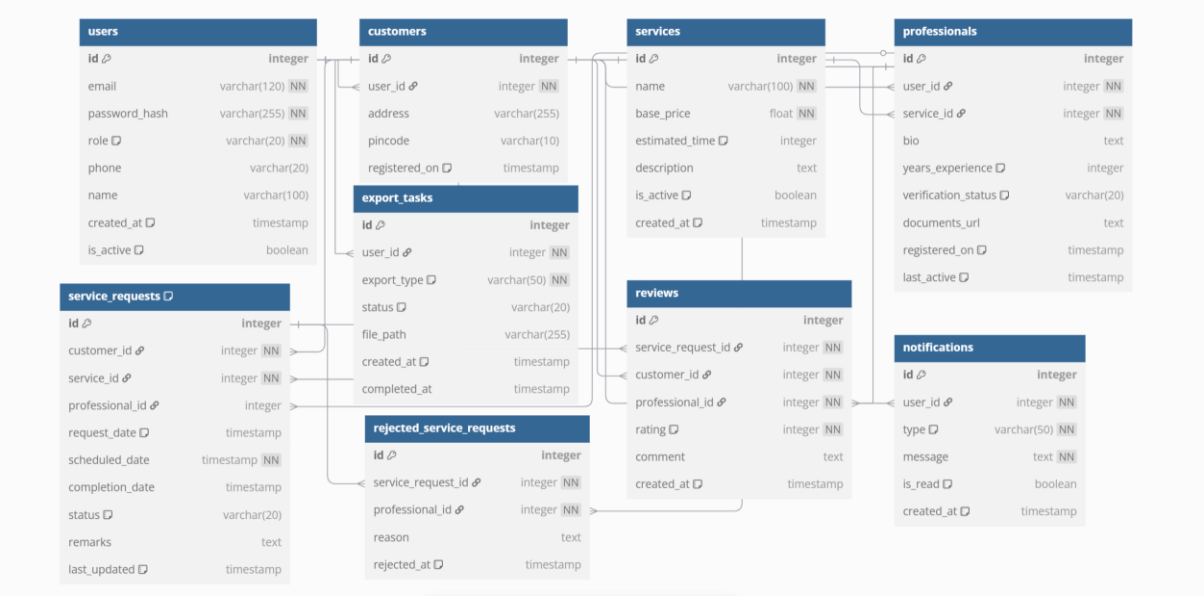
DB Schema Design

Role Based User Model: Single users table with role differentiation allows for a single authentication endpoint while maintaining distinct user types

Service Request Lifecycle: The status field in Service Requests enables tracking the entire lifecycle from request to completion

Professional Verification: Dedicated verification workflow ensures quality control before professionals can accept service requests.

Notification System: Generic notification table that can accommodate various notification types across all user roles.



API Design

The API is designed with these endpoints:

Authentication Endpoints

- POST /api/register: User registration for customers and professionals
- POST /api/login: User authentication and token generation
- POST /api/refresh: Refresh access token
- POST /api/logout: User logout and token invalidation

Admin Endpoints

- GET /api/admin/dashboard: Retrieve system statistics
- GET, PUT /api/admin/professionals: Manage professionals
- GET, PUT /api/admin/customers: Manage customers

Customer Endpoints

- GET, PUT /api/customers/<id>: Retrieve or update customer profile
- GET /api/customers: List all customers (admin only)

Professional Endpoints

- GET, PUT /api/professionals/<id>: Retrieve or update professional profile
- PUT /api/professionals/<id>/verify: Upload verification documents
- POST /api/professionals/<id>/verify: Approve or reject professional profile

Service Endpoints

- GET, PUT, DELETE /api/services/<id>: Manage individual services
- GET, POST /api/services: List all services and create new ones

Service Request Endpoints

- GET, PUT, DELETE /api/service-requests/<id>: Manage individual service requests
- GET, POST /api/service-requests: List service requests and create new ones
- POST /api/service-requests/<id>/action: Perform actions on service requests (accept, reject, complete, close)
- GET /api/rejected-requests: List rejected service requests

Review Endpoints

- GET, PUT, DELETE /api/reviews/<id>: Manage individual reviews
- GET, POST /api/reviews: List reviews and create new ones

Notification Endpoints

- GET, PUT, DELETE /api/notifications/<id>: Manage individual notifications
- GET, PUT /api/notifications: List notifications and mark all as read

Architecture

Backend (Flask)

Core: Flask RESTful API with SQLAlchemy ORM

Resources: Modular endpoints based on domain entities (admin, auth, customer, etc.)

Tasks: Background processes using Celery for reports, reminders, and exports

Static: Document storage and file management

Utils: Shared helper functions and authentication decorators

Frontend (Vue.js)

API Layer: Modular API clients for backend communication

Stores: Pinia stores for state management by domain

Views: Role-segregated UI components (admin, customer, professional)

Utils: Shared utilities for formatting, validation, and authentication

Router: Navigation with route guards based on user roles

Features

Form validation: Client-side validation for all forms

Rejection tracking: Track and manage rejected service requests

Real-time statistics: Dashboard with up-to-date counts and status metrics

Video - https://drive.google.com/file/d/1Ej9RzOAO0qBr79zb8y23VFVP_ESvvk6d/view?usp=sharing