Household Services Application - V2

A project report for the Modern Application Development II

Author

SANJAI NATARAJAN

23F1002165

23f1002165@ds.study.iitm.ac.in

IITM Online BS Degree Program

Indian Institute of Technology Madras, Chennai

Tamil Nadu, India 600036

Description

The goal of this project is to develop a web-based platform that seamlessly connects customers with service professionals for household-related services. The platform enables customers to search for services, request assistance, and track service progress. Service professionals can manage, accept, and complete service requests, while administrators oversee user roles, service listings, and platform operations.

The system ensures a smooth and transparent user experience through service tracking, real-time notifications, scheduled tasks, and comprehensive reports. Designed for efficiency and scalability, the application effectively manages user roles, enhances performance through caching, and supports automated reminders and activity reports.

Technologies used

Backend:
- Flask (Python)
- Flask-RESTful (API development)
- Flask-SQLAlchemy (ORM for database management)
- Flask-Migrate (Database migrations)
- Flask-Security (Token-based authentication)
- Celery (For background tasks)
- Redis (For caching and job queueing)

Frontend:
- Vue.js (CDN-based)
- Bootstrap (For responsive UI)
- HTML, CSS, JavaScript

Database:
- PostgreSQL (Primary database)
- SQLite (For local development and testing)

Task Scheduling & Background Jobs:
- Celery Beat (Task scheduling)
- Crontab (Automated task execution)

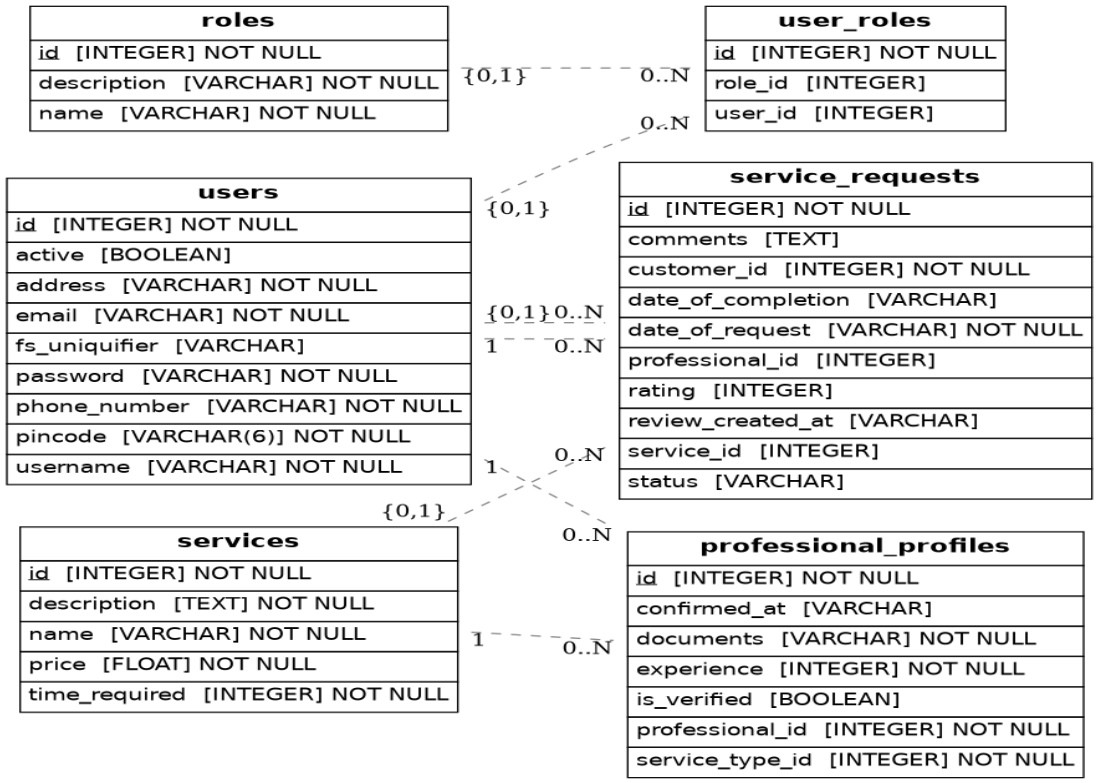
Email & Notifications:
- Flask-Mail, smtplib (For email handling)
- Google Chat Webhooks (For notifications)

DB Schema Design

Database Tables and Relations:

- **Users** (id, username, email, password, active, address, phone_number, pincode, fs_uniquifier)
- **Roles** (id, name, description)
- **User Roles** (id, user_id, role_id)
- **Professional Profiles** (id, professional_id, service_type_id, experience, is_verified, documents, confirmed_at)
- **Services** (id, name, description, price, time_required)
- **Service Requests** (id, customer_id, professional_id, service_id, date_of_request, date_of_completion, status, rating, review_created_at, comments)

The ER diagram below represents the database schema:



Relations: Service Requests are linked to Users, Professional Profiles, and Services (Many-to-One with each). Professional Profiles are linked to Users (One-to-One) and Services (Many-to-One).

API Design

- **Authentication:**
  - `POST /api/login` - Login for admin, professionals, and customers
  - `POST /api/register` - Register a new user (Customer)
  - `POST /api/professional/register` - Register a new user (Professional)
- **Admin Endpoints:**
  - `POST /api/services` - Create a new service
  - `POST /api/edit_service/<int:id>` - Update a service
  - `DELETE /api/delete_service/<int:id>` - Delete a service
  - `GET /api/customers` - View all customers
  - `POST /api/status/customer/<int:id>` - Block/Unblock a customer
  - `GET /api/user?name=` - Search professional
  - `GET /api/professional` - View all professionals
  - `POST /api/status/professional/<int:id>` - Block/Unblock a professional
  - `POST /api/verify/professional/<int:id>` - Approve a professional
  - `POST /api/deny/professional/<int:id>` - Reject a professional
  - `GET /api/request` - View all service requests
- **Customer Endpoints:**
  - `GET /api/services` - View available services
  - `POST /api/profile/edit/<int:id>` - Update profile
  - `POST /api/request/service` - Create a service request
  - `POST /api/request/edit/<int:id>` - Update a service request
  - `POST /api/request/close/<int:id>` - Submit a review
- **Service Professional Endpoints:**
  - `POST /api/re_verify` – Submit re-verification request

- `GET /api/servicerequest/<int:id>` - View assigned service requests
- `POST /api/request/edit/<int:id>` - Accept /Reject/Complete a request

- **Admin Dashboard Management Endpoints:**
  - `GET /api/service/<string:name>` - Fetch service details
  - `GET /api/customer/<int:id>` - Fetch customer details
  - `GET /api/professional/<int:professional_id>` - Fetch professional details
  - `GET /api/service_request/<string:name>` - Fetch all requests linked to a service
  - `GET /api/request/service/<int:customer_id>` - Fetch all requests made by the customer
- **Batch Jobs (Celery Tasks):**
  - `send_service_reminders` - Sends reminders to professionals for new and pending requests
  - `send_monthly_report` - Generates and emails customer activity reports
  - `create_csv` - Allows admin to download service requests as CSV

Architecture and Features

User Roles: Implemented three primary roles—Admin, Service Professional, and Customer—with role-based access control (RBAC).
Authentication System: Developed a secure login and registration system using token-based authentication.
Customer Features: Customers can search for services based on location, name, and pin code and request services from available professionals, and browse ratings and reviews before booking.
Professional Features: Professionals can accept/reject service requests, mark them as completed, and receive automated reminders about pending requests, and start services using OTP-based verification.
Admin Features: Admins can add, update and manage services, search for professionals, and block/unblock them if needed. Admins have access to detailed reports and analytics of service requests.
OTP-Based Service Initiation: Customers receive a one-time password (OTP) when starting a service request. Professionals must enter the OTP to complete the service, ensuring authenticity.
Dummy Payment Portal: Implemented a dummy payment portal that takes payment details from customers for a service request.
Automated Email Reports: Used Celery to generate and send monthly activity reports to customers.
Task Scheduling: Configured Celery Beat to automate periodic tasks such as report generation and email distribution.
Backend Jobs:

1. **Daily Reminders:** Send alerts to professionals for pending service requests.
2. **Monthly Reports:** Generate and send service activity reports via email.
3. **CSV Export:** Generate a downloadable CSV report for all service requests.

Performance Optimization: Implement caching using Redis to enhance API performance and reduce response time.

Video

Link to the video presentation
https://drive.google.com/file/d/1DK6dcCSv2NaWkWKrA2HuNdfiePrjxEM8/view?usp=sharing

---

Conclusion

This project successfully demonstrates a functional household services platform with secure user authentication, service request management, automated reporting, and real-time data visualization. The system efficiently integrates role-based access control, background task automation, and performance optimizations to enhance user experience and operational efficiency. Future enhancements may include online payment integration and AI-based service recommendations.
