Household Services Application -V2

Submitted by:
Harsh Badala
Roll No. : 22F2000474

Table of Contents

---

1. Introduction

The Household Services Application is a full-stack web platform designed to connect customers with professional service providers for various household needs. It provides an intuitive interface for customers to request services, professionals to manage service requests, and administrators to oversee platform operations. The system is developed to ensure seamless interactions between all users while incorporating automated background tasks and analytics.

---

2. Technologies Used

Backend:

- Flask (RESTful API development)
- SQLAlchemy (ORM for database interactions)
- Redis (Caching and Celery task management)
- Celery (Background task processing)
- JWT (Authentication & Security)

Frontend:

- HTML, CSS, Bootstrap (UI design)
- Vue.js (Client-side rendering & dynamic components)
- Fetch API (Asynchronous communication)

Database:

- SQLite (Lightweight database for data storage)

Libraries:

- Flasgger (API documentation)
- Chart.js (Data visualization)

---

3. System Architecture

The application follows a three-tier architecture:

- **Presentation Layer (Frontend):** Developed using Vue.js and Bootstrap for a modern, responsive UI.
- **Business Logic Layer (Backend):** Implemented with Flask, providing RESTful API endpoints.
- **Data Layer (Database):** SQLite database managed via SQLAlchemy ORM.

Additionally, Redis and Celery handle caching, scheduled background tasks, and asynchronous processing.

---

4. Installation Guide

Backend Setup:

1. Clone the repository:

```
git clone <repository-url>
cd backend
```

2. Create a virtual environment:

```
python -m venv venv
source venv/bin/activate   # macOS/Linux
venv\Scripts\activate      # Windows
```

3. Install dependencies:

```
pip install -r requirements.txt
```

4. Initialize the database:

```
python populate_db.py
```

5. Start the Flask server:

```
python app.py
```

Backend will be available at `http://127.0.0.1:5000/`

Frontend Setup:

1. Navigate to the frontend directory:

```
cd frontend
```

2. Install dependencies:

```
npm install
```

3. Run the frontend:

```
npx serve .
```

The application will be available at `http://localhost:5000`

---

5. Features and Functionalities

Customer Features:

- Account creation & profile management
- Service browsing by category
- Booking services & tracking requests
- Viewing service history & submitting reviews
- Receiving monthly activity reports

Professional Features:

- Profile creation with service details
- Accepting/rejecting service requests
- Managing schedules & appointments
- Tracking earnings & performance metrics
- Receiving daily reminders

Admin Features:

- Managing service categories & professionals
- Approving or blocking professionals
- Exporting service data & generating reports
- Monitoring platform analytics
- Sending notifications to users

---

6. API Documentation

The API is documented using Swagger/OpenAPI and can be accessed at `/apidocs` when the backend is running.

Authentication:

- JWT authentication required for most endpoints
- Token must be included in requests as `Bearer <token>`

Example Endpoints:

Endpoint	Method	Description
`/register`	POST	Register a new user
`/login`	POST	Authenticate & get JWT token
`/customer/profile`	GET	Retrieve customer profile
`/customer/professionals/<service_type>`	GET	Get professionals by service
`/professional/requests`	GET	View service requests
`/admin/service`	POST	Create a new service

`/admin/export/<professional_id>`	GET	Export service requests

---

7. Testing and Debugging

Backend Testing:

- **Celery Reminders:** `python test_reminders.py`
- **Monthly Reports:** `python test_reports.py`
- **Data Exports:** `python test_exports.py <professional_id>`

Frontend Testing:

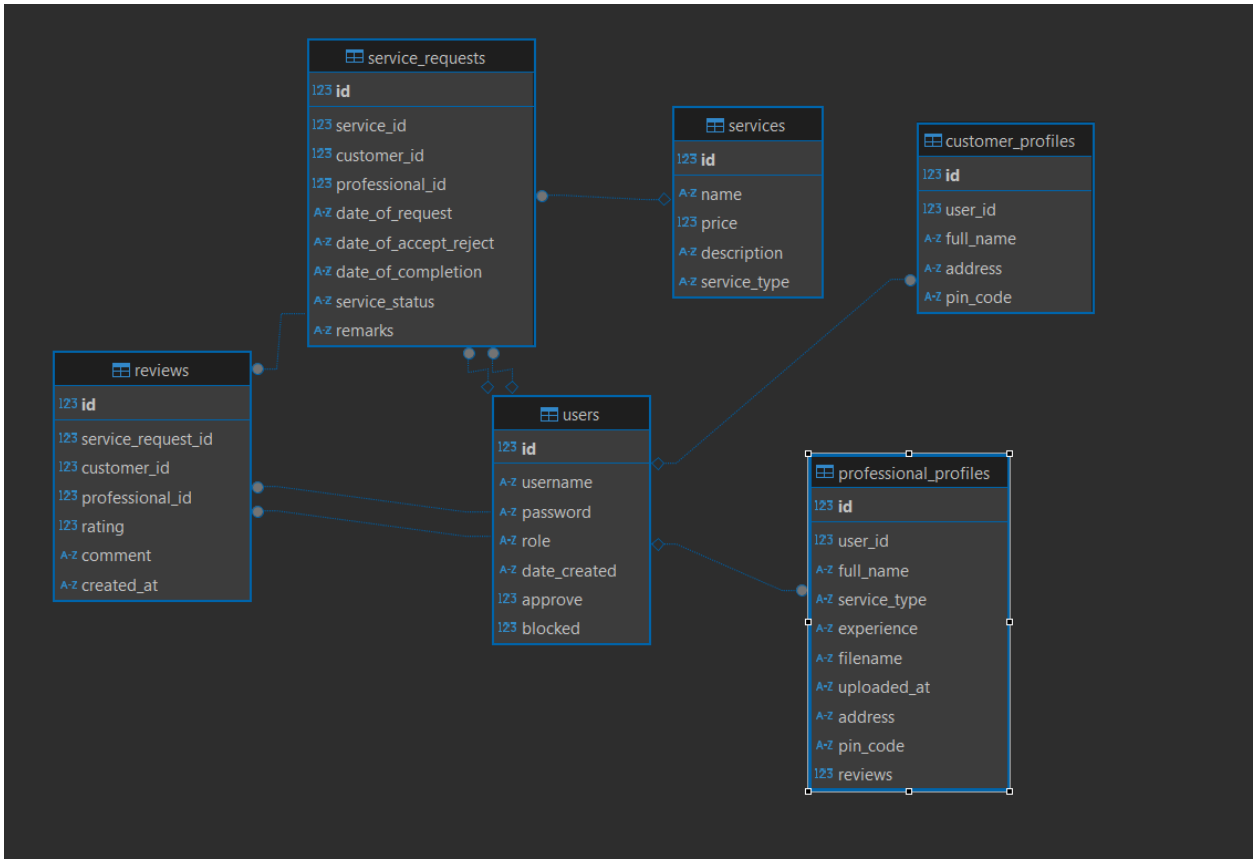
- Ensure correct UI functionality for customer registration, service booking, professional approvals, and admin tasks.

---

8. Conclusion

The Household Services Application effectively bridges the gap between customers and service professionals by providing a streamlined platform for managing service requests. With its structured architecture, efficient background task handling, and well-defined API, the system ensures smooth operations for all users. Future improvements may include integrating machine learning for service recommendations and enhancing user experience with real-time chat functionalities.

ER - Diagram



Drive Link to project demonstration(MAD2).mp4 or https://drive.google.com/file/d/1c-7d3OywORPsT_KQwLlWjyWmezUe3-Ke/view?usp=sharing