


Household Services Application

Student Details

- Name: Rahul Maurya
- Email: 23f1002653@ds.study.iitm.ac.in
- Presentation video:  MAD1-PROJECT-RAHUL-MAURYA.mp4

Project Details

Platform for household services. Professionals from multiple cities provide citywide services. Customers browse, book, and get services. Professionals accept bookings, fulfill requests, and customers close them upon completion. Admin manages users, professionals, and services, including blocking/unblocking users and verifying professionals.

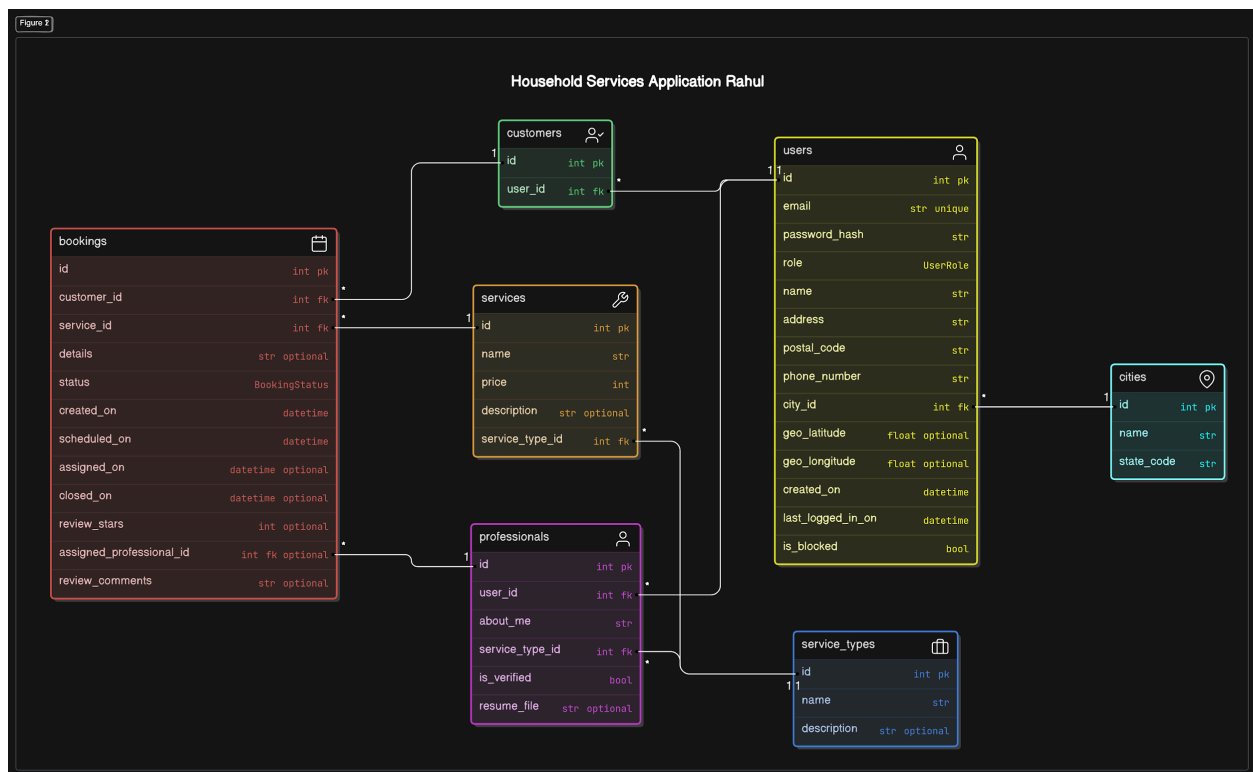
Implementation

- ER modeling: entities, attributes, relationships.
- Flask app setup: init, extensions, blueprints, and routes.
- Secure login sessions via Flask-Login. Role-based route access.
- Bootstrap for UI, Charts.js for summaries, and DataTables.js for tables.

Insights and Learnings

The implementation began with creating an ER model to structure entities and relationships. Flask was set up using blueprints for modularity, ensuring each user role had a clear route structure. Secure login sessions were integrated with Flask-Login for role-based access. Bootstrap streamlined UI design, while Charts.js and DataTables.js added interactive summaries and tables. Developing this system highlighted the importance of clean architecture, modular coding, and aligning database design with functional needs.

ER Model



Libraries

- Backend: Flask, Flask-SQLAlchemy, Flask-Login, SQLite.
- Frontend: Bootstrap, Bootstrap Icons, DataTables (with jQuery), Charts.js.

Folder Structure

|—app # App configuration and initialization.

|—controllers # Flask blueprints for handling routes.

|—database # ER diagram, models, and sample data.

|—instance # Local SQLite database storage.

|—static # Uploads and static files like CSS/JS.

|—templates # HTML templates for all pages.

Flask Routes

Public Blueprint (/)

- /: Home
- /login: Login (GET, POST).
- /logout: Logout.
- /signup: Customer signup (GET, POST).
- /signup-as-professional: Professional signup (GET, POST).

Admin Dashboard Blueprint (/admin)

- /dashboard: Admin homepage.
- /search: Search users/services.
- /search-results: Display search results.
- /summary: Admin summary view.
- /block_user/<int:user_id>: Block a user.
- /unblock_user/<int:user_id>: Unblock a user.
- /verify_professional/<int:professional_id>: Approve a professional.
- /update_service/<int:service_id>: Edit service (GET, POST).
- /delete_service/<int:service_id>: Remove service.

- `/create_service`: Add service (GET, POST).
- `/create_service_type`: Add service type (GET, POST).
- `/create_city`: Add city (GET, POST).

Customer Dashboard Blueprint (`/customer`)

- `/dashboard`: Customer homepage.
- `/search`: Search services.
- `/search_results`: Show filtered services.
- `/summary`: Customer summary view.
- `/book/<int:service_id>`: Book a service (GET, POST).
- `/cancel_booking/<int:booking_id>`: Cancel a booking.
- `/close_booking/<int:booking_id>`: Close booking after service (GET, POST).

Professional Dashboard Blueprint (`/professional`)

- `/dashboard`: Professional homepage.
- `/search`: Search bookings.
- `/search_results`: Filtered booking results.
- `/summary`: Professional summary view.
- `/accept_booking/<int:booking_id>`: Accept a booking.
- `/cancel_booking/<int:booking_id>`: Decline a booking.

Profile Blueprint (`/profile`)

- `/customer/<int:customer_id>`: View customer profile.
- `/professional/<int:professional_id>`: View professional profile.
- `/edit_customer/<int:customer_id>`: Edit customer info (GET, POST).
- `/edit_professional/<int:professional_id>`: Edit professional info (GET, POST).

How to Run

1. Install dependencies: `pip install -r requirements.txt`.
2. Start the app: `python run.py`.