Author

Adithya Pradeep
21f1002580
21f1002580@ds.study.iitm.ac.in
FullStack Software Engineer currently working at Oracle

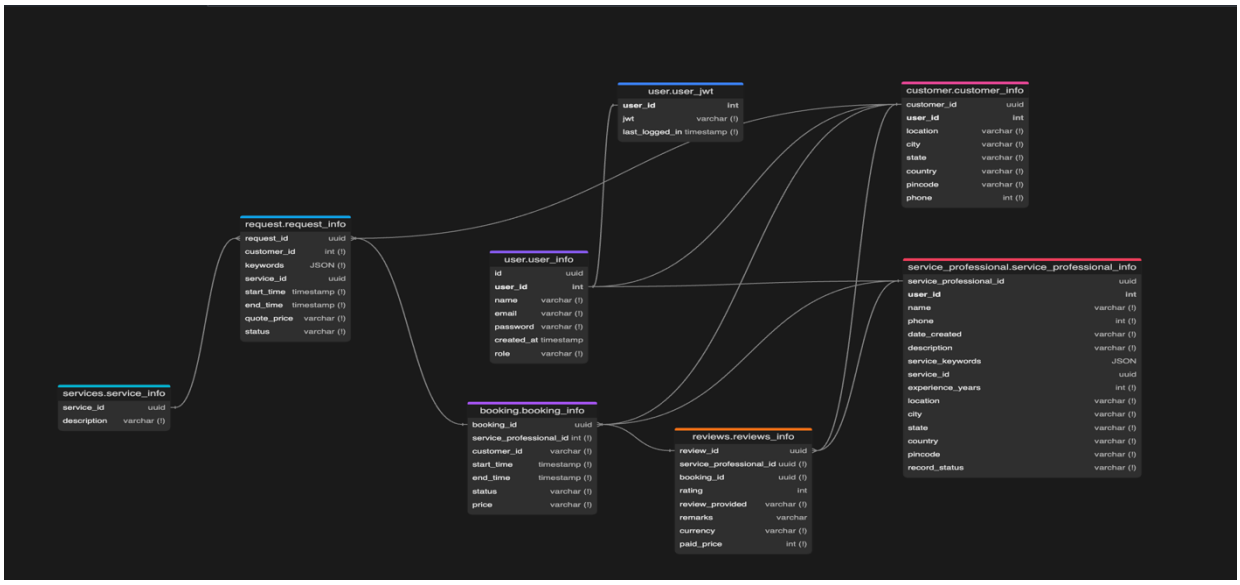
Description

Build an application focused on providing users with household services like plumber, electrician or cleaner on a dynamic basis. Users should be able to book services based on their schedule such that professionals can accept or reject the request. Professionals should be able to handle the user booking information, sending reminder emails to collecting customer feedback.

Technologies Used

Flask (Backend), Vue.js (Frontend), Redis (Caching), Celery (Job Scheduling), SQLite (Database), Resend (Emails)

DB Schema Design



Primary Keys & Unique Constraints: user_info.user_id is the primary key and auto-increments. service_professional_info.user_id and customer_info.user_id are foreign keys that link to user_info.user_id. UUIDs are used for most primary keys to ensure uniqueness and scalability.

Foreign Key Constraints (Relationships between Tables):
user_id in user_jwt, service_professional_info,
customer_info references user_info.user_id
customer_id in booking_info references customer_info.customer_id
service_professional_id in booking_info references service_professional_info.service_professional_id.
service_id in service_professional_info
request_info references service_info.service_id.
booking_id in reviews_info references booking_info.booking_id.service_professional_id in reviews_info

references service_professional_info.service_professional_id.
customer_id in reviews_info references customer_info.customer_id

Reasons for the Design Choices:

Separation of Concerns: The database is structured with separate schemas (user, services, service_professional, customer, booking, request, reviews) to ensure modularity. This allows for easy scaling and future modifications
Scalability & Performance: UUIDs are used for customer_id, service_id, and booking_id to ensure global uniqueness. Using JSON fields for service_keywords and keywords allows dynamic storage of flexible attributes
Security and Access Control: Storing JWTs separately from user information enhances security
Role-based access control: (role column in user_info) ensures different permission levels
Efficient Data Retrieval: Indexing user_id, service_id, and booking_id improves query performance. Timestamp columns allow sorting based on activity

API Design

This API is designed to facilitate an on-demand service platform, enabling interactions between administrators, customers, and service professionals. It is implemented using Flask and structured into modular route groups for better maintainability and scalability

1. Admin Routes:
Service Management: Create, edit, and delete services
User Data Access: Retrieve customer and service professional verification data
2. Authentication Routes:
User Management: Registration, login, logout, and token-based authentication
Protected Routes: Access control for restricted endpoints
3. Customer Routes:
Booking & Reviews: View and update booking details, submit reviews
Search Services: Retrieve service listings based on queries
4. Request Info Routes:
Service Requests: Insert, view, and cancel service requests
5. Service Info Routes:
Location-Based Service Fetching: Retrieve service details based on user location
6. Service Professional Routes:
Verification & Documents: Upload, view, and verify professional credentials.
Job Management: Accept and manage service requests, bookings, and reviews.
Report Generation: Export service requests and track report download status

Architecture and Features

The project follows a modular architecture using Flask, with separate controllers for different user roles and functionalities. The controllers are organized within distinct route files, such as admin_routes.py, authentication_routes.py, customer_routes.py, request_info_routes.py, service_info_routes.py, and service_professional_routes.py, ensuring clean separation of concerns. Service logic is handled within corresponding service classes like AdminService, AuthService, CustomerService, and ServiceProfessionalService.

User Authentication & Authorization: Implemented via JWT, with login, logout, and protected routes
Admin Management: Service creation, modification, and user verification
Customer Services: Booking management, service search, and review submission
Service Professional Management: Registration, document upload, verification, and job management

Service Request Handling: Insertion, retrieval, cancellation of service requests
Rate Limiting: Protects against API abuse with predefined request limits
Asynchronous Report Generation: Uses Celery for exporting and downloading service request reports
File Handling: Secure document uploads and safe file downloads
Role-Based Access: Ensures proper authorization for different users

Video

https://drive.google.com/file/d/1z6jAkxAeQUUH3m003jIiscbDyPDgcPoZ/view?usp=sharing