Name : Risshab Srinivas Ramesh

Roll No : 23F3002157

Student Email : 23f3002157@ds.study.iitm.ac.in

Project report – MAD2 Project

Project Description

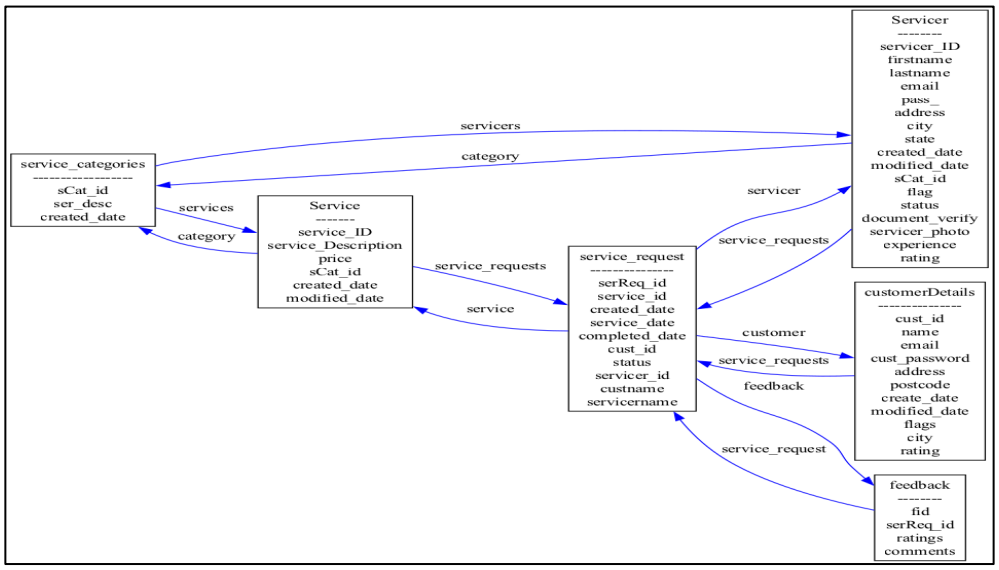
This project aims to create a dynamic service platform designed to connect customers with skilled service providers for a wide range of everyday needs. The platform empowers customers to effortlessly browse, select, and book trusted professionals for services tailored to their requirements. Service providers, in turn, gain access to a steady stream of opportunities to showcase their expertise and grow their business.

At the heart of system is a robust admin system that ensures seamless management of both customers and service providers. The admin oversees operations, maintaining quality and trust by monitoring interactions and performance. With the ability to block either party based on predefined parameters—such as compliance, behavior, or service standards—the admin ensures a safe, reliable, and efficient experience for all. The project is more than just a marketplace, it's a community built on convenience, accountability, and excellence

Technologies Used:

1. Flask-RestFul - APIs
2. SQLAlchemy – For Database Interaction
3. Email (MIME) – For sending emails
4. flask_jwt_extended – for JWT token
5. Jinja2 for templates
6. flask_caching for caching
7. celery - for backend jobs
8. SQLite for Database storage

DB Schema design:



API Design

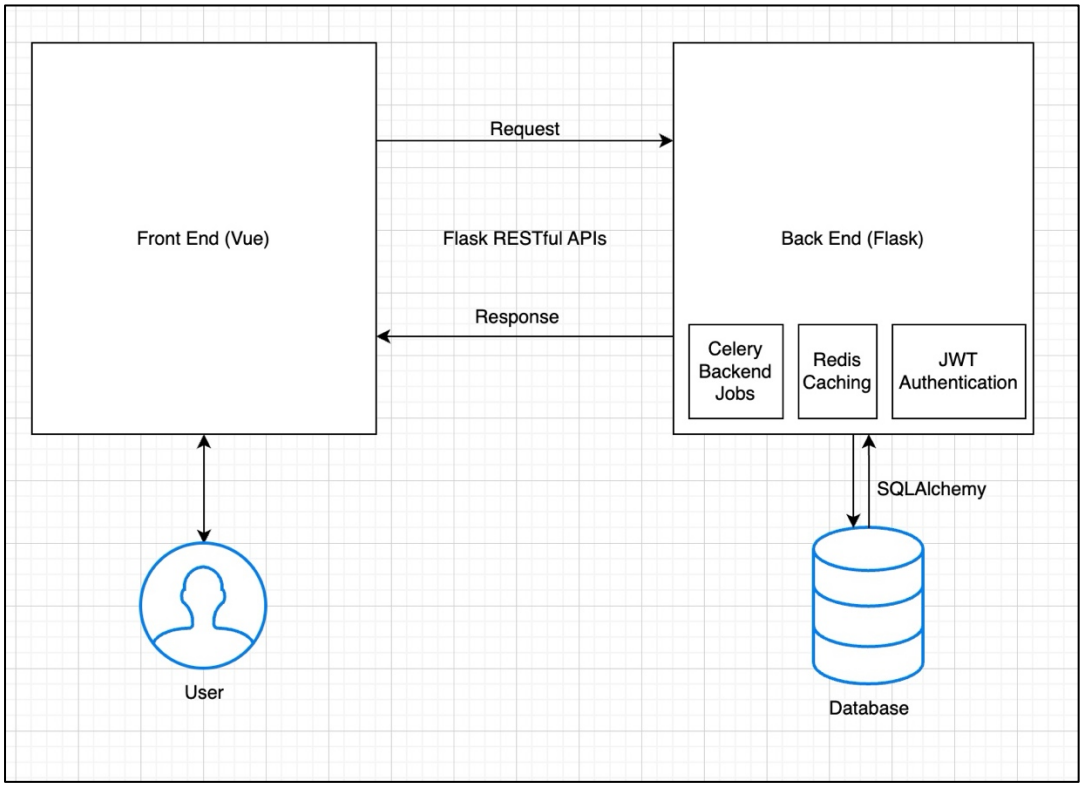
The API designed facilitates the interactions between customers, servicers (service providers), and administrators. It manages key entities and operations such as user authentication, service categorization, service requests, and administrative oversight. Below are the main elements for which APIs were created:

1. **Customers**:
    - **Purpose**: Allows customers to sign up, log in, manage their profiles, request services, and track their service requests.
    - **Endpoints**:
        - /customerSignUp: Register a new customer.
        - /customerLogin: Authenticate and provide a JWT token.
        - /customerDashboard: Access customer dashboard.
        - /customerDashboard/getDetails: Retrieve customer profile details.
        - /customerDashboard/serviceRequest: Create a service request.
        - /customerDashboard/deleteRequest/{serReq_id}: Delete a service request.
        - /customerDashboard/updateRequest: Update service request date.
        - /customerDashboard/search: Search for services or servicers.
        - /customerDashboard/summary: View summary of service requests.
2. **Servicers (Service Providers)**:
    - **Purpose**: Enables servicers to sign up, log in, view assigned service requests, update request statuses, and close completed services.
    - **Endpoints**:
        - /servicerSignUp: Register a new servicer.
        - /servicerLogin: Authenticate and provide a JWT token.
        - /servicerDashboard/serviceRequests: Retrieve all assigned service requests.
        - /servicerDashboard/updateRequestServicer: Update request status (accept/reject).
        - /servicerDashboard/closeServiceCutomer: Close a service request and add feedback.
3. **Services**:
    - **Purpose**: Manages service categories and individual services, allowing customers to browse and admins to create/edit them.
    - **Endpoints**:
        - /getServices: List all service categories.
        - /getServicesAdmin: List all services with category details (admin view).
        - /adminDashboard/new_service: Create a new service (admin only).
        - /adminDashboard/new_service/{service_id}: Delete or update a service (admin only).
4. **Service Requests**:
    - **Purpose**: Handles the lifecycle of service requests from creation to completion.
    - **Endpoints**:
        - /getServiceRequest: Retrieve customer's service requests.
        - /customerDashboard/serviceRequest: Create a service request.
        - /customerDashboard/deleteRequest/{serReq_id}: Delete a service request.
        - /customerDashboard/updateRequest: Update service request date.
        - /servicerDashboard/serviceRequests: Retrieve servicer's assigned requests.
        - /servicerDashboard/updateRequestServicer: Update request status.
        - /servicerDashboard/closeServiceCutomer: Close a request with feedback.
5. **Admin Functions**:
    - **Purpose**: Provides administrative control over customers, servicers, services, and reports.

- **Endpoints**:
  - /adminLogin: Authenticate admin and provide a JWT token.
  - /adminDashboard: Access admin dashboard.
  - /adminDashboard/customer: List all customers.
  - /adminDashboard/customerBlock/{cust_id}: Block/unblock a customer.
  - /adminDashboard/servicerToggle/{servicer_id}: Toggle servicer flag, delete, or approve.
  - /adminDashboard/new_service: Create a new service.
  - /adminDashboard/new_service/{service_id}: Edit or delete a service.
  - /adminDashboard/search: Search servicers.
  - /adminDashboard/summary: View system summary.
  - /adminDashboard/exportReport: Export service request report as CSV.

6. **General Utilities**:
   - **Purpose**: Provides basic system access and servicer browsing.
   - **Endpoints**:
     - /api/welcome: Welcome message and customer list (for testing/demo).
     - /getServicers: List all servicers.
     - /getServicersCustomer/{sCat_id}: List servicers by category.

Architecture



1. **Framework and Libraries**:
   - **Flask**: Core web framework for routing and request handling.
   - **Flask-RESTful**: Used to define API resources as classes (e.g., homePageAPI, customerSignUp).
   - **Flask-JWT-Extended**: Implemented JWT-based authentication for securing endpoints (e.g., @jwt_required()).
   - **SQLAlchemy**: Managed database interactions with models like CustomerDetails, Service, Servicer, and ServiceRequest.

    - **Flask-Caching**: Added caching (e.g., homePageAPI GET method) for performance optimization.
2. **Database Models**:
    - Defined models (CustomerDetails, Service, ServiceCategory, Servicer, ServiceRequest, Feedback) with SQLAlchemy to store and query data.
    - Each model includes a convert_to_json() method to serialize data into JSON responses.
3. **HTTP Methods**:
    - **GET**: Retrieve data (e.g., /getServices, /customerDashboard/getDetails).
    - **POST**: Create resources or authenticate (e.g., /customerSignUp, /adminLogin).
    - **PUT**: Update resources (e.g., /customerDashboard/updateRequest, /adminDashboard/servicerToggle/{servicer_id}).
    - **PATCH**: Partial updates (e.g., unblock customer in /adminDashboard/customerBlock/{cust_id}).
    - **DELETE**: Remove resources (e.g., delete servicer in /adminDashboard/servicerToggle/{servicer_id}).
4. **Authentication**:
    - JWT tokens generated via create_access_token() and validated with get_jwt_identity() for secure access to customer, servicer, and admin endpoints.
    - Admin-only endpoints (e.g., /adminDashboard/*) check for a specific identity (e.g., admin@1234).
5. **Routing**:
    - Routes defined using api.add_resource() to map endpoints to resource classes (e.g., api.add_resource(customerSignUp, '/customerSignUp')).
    - Path parameters (e.g., {cust_id}, {service_id}) used for dynamic routing.
6. **Error Handling**:
    - Returns appropriate HTTP status codes (200 for success, 400 for bad requests, 401 for unauthorized, 404 for not found) with JSON error messages.
7. **Additional Features**:
    - **Search**: Implemented in /customerDashboard/search and /adminDashboard/search using filters on database queries.
    - **Summaries**: Aggregated data in /customerDashboard/summary and /adminDashboard/summary for quick insights.
    - **File Export**: /adminDashboard/exportReport generates a CSV file using an async task (admin_asyncReport).

Project Video Link

https://drive.google.com/file/d/1SuaRNt1fmeOkQgNs0ezEjT_VtW-HILmN/view?usp=sharing

YAML File Link

https://drive.google.com/file/d/17twyYE3O4jpJ2_-VYL2Ks86L3Juq0P30/view?usp=sharing