AZ
HOUSEHOLD
SERVICES
Quality Service at your doorsteps • 100% Verified Professionals

A PROJECT REPORT FOR

HOUSEHOLD SERVICES APPLICATION

BY

MAYANK KUMAR PODDAR

23f3004197 | Modern Application Development – I | IIT Madras

Author:

Name: Mayank Kumar Poddar
Email: 23f3004197@ds.study.iitm.ac.in
Website: https://mynkpdr.github.io

I am currently pursuing in diploma level of BS degree in Data Science from IIT Madras. My passion for computers and technology began in my childhood, and this enthusiasm has driven me ever since. I am dedicated to pushing my boundaries and continuously expanding my knowledge to achieve impactful and meaningful growth.

Description:

This project is from the Modern Application Development - I course which is offered by IITM BS Degree during the Diploma in Programming.
It is a multi-user app where verified professionals can provide essential home servicing and solutions to customers. Services include Electrical, Carpentry, Plumbing, Appliance Repair, etc. The project enables customers to request services from professionals of their choice. Upon receiving a service request, the professional has the option to either accept or reject it. Once the service is completed, the customer can mark the request as closed and provide a review. Admins have comprehensive control over the platform, including the ability to create new services, and manage and edit customer and professional profiles. The system is designed to be scalable and extendable, allowing for the future integration of additional secure features as needed.

Technologies used:

Below are some of the important backend modules and libraries that I've used in this project.

- Flask: The core framework for this web application, Flask provides essential tools and features, allowing for seamless routing, request handling, and template rendering.
- PyJWT: Used for creating and verifying JSON Web Tokens (JWT) for secure API authentication and user session management.
- WTForms: This library helps prevent Cross-Site Request Forgery (CSRF) attacks by incorporating CSRF tokens automatically.
- Requests: This library simplifies interactions with APIs, making it easier to send and receive data in a readable format.
- Flask_SQLAlchemy: It provides a straightforward way to work with databases using Python objects.
- Flasgger: This library assists in generating a Swagger UI based on OpenAPI specifications, making API documentation accessible and interactive

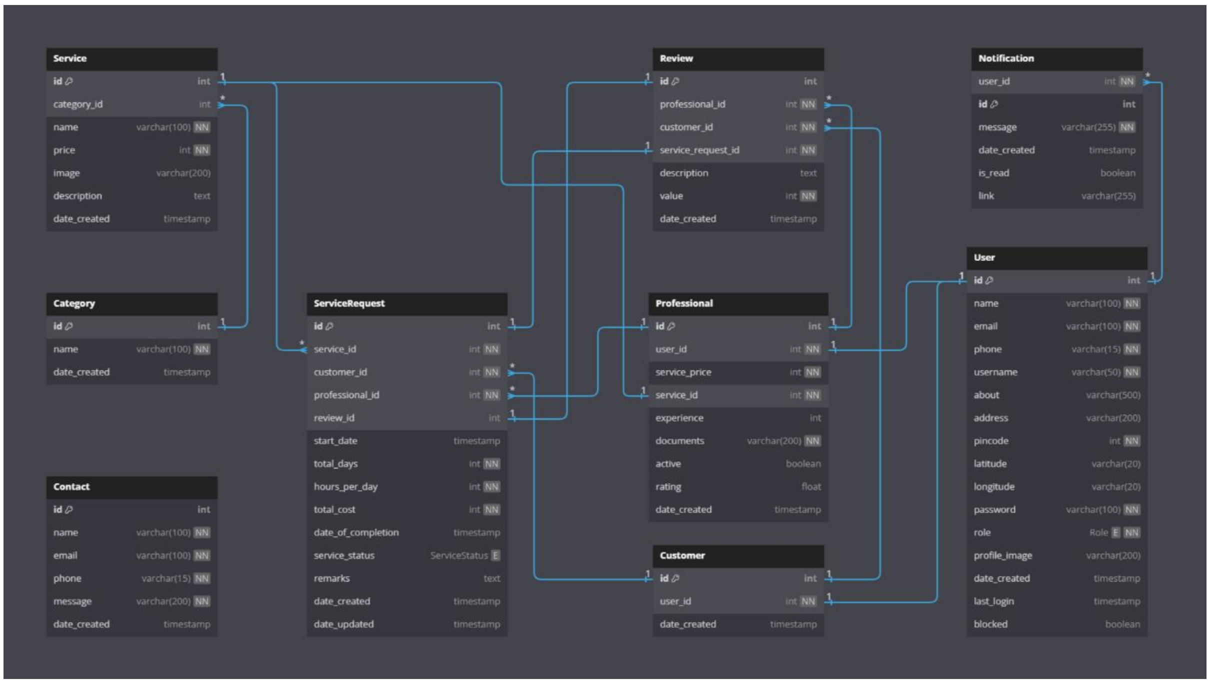
And for the frontend, the technologies that I've used are:

- HTML, CSS, JS: The core of the frontend. Basic CSS and JS are used.
- Bootstrap 4: It is a front-end framework with pre-styled components and a flexible grid system.
- Chart.js: It is a JavaScript library for creating interactive and customizable data visualizations like charts and graphs.

In addition to these, several other modules and libraries contribute to the functionality and security of this web application.

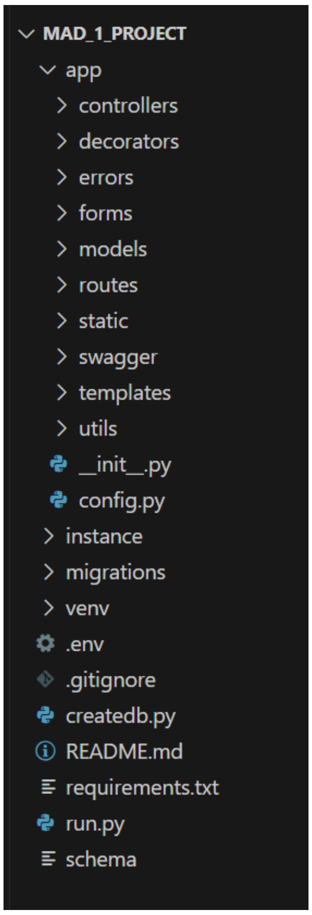
Database Schema Design:

There are total of 9 database tables. It includes features like notifications, role-based access (Admin, Customer, Professional), and timestamps for tracking records. Foreign keys and enums ensure data integrity and role management. Overall, it facilitates managing users, services, requests, and reviews. There are no multi-valued or transitive dependencies, and all relationships are properly defined with foreign keys. The following schema diagram can explain a lot of database schema details.



- User: Represents a system user (could be a customer, professional, or an admin). It stores personal information, login credentials, and a reference to the user's role. A user can have associated professional or customer data.
- Professional: Represents a professional offering services. A professional can handle service requests and receive reviews.
- Customer: Represents a customer in the system, linked to a User. A customer can make multiple service requests and leave reviews for services or professionals.
- Category: Represents a service category (e.g. Electrical, Appliance Repair). Each category can have multiple associated services.
- Service: Represents a specific service offered (e.g., CCTV Installation, Washing Machine Repair). A service belongs to a category and can be requested multiple times.
- ServiceRequest: Represents a customer's request for a service.
- Review: Stores feedback from customers for services or professionals.
- Notification: Stores notifications for users, containing a message, timestamp, and whether it has been read, along with a link.
- Contact: Stores contact form submissions.

Architecture and Features:

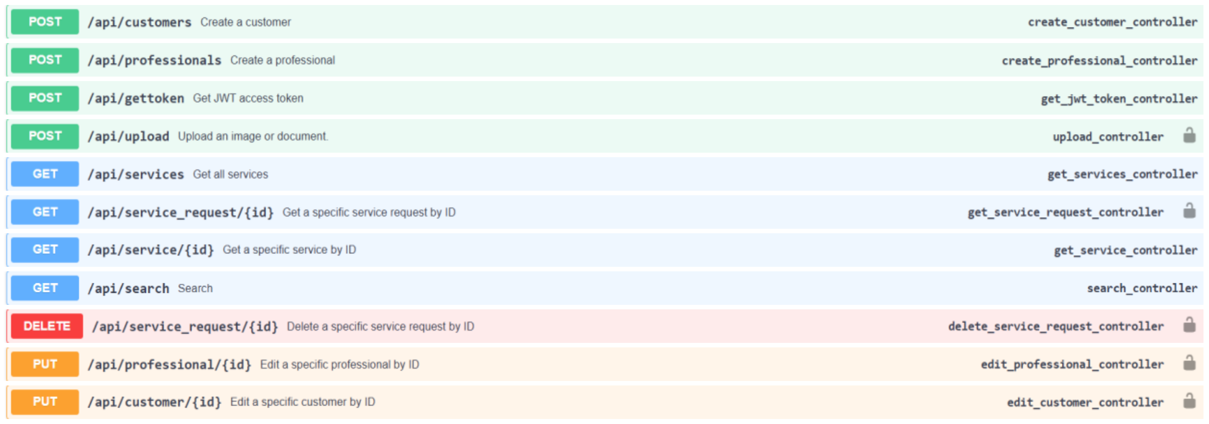


```
MAD_1_PROJECT
  app
    controllers
    decorators
    errors
    forms
    models
    routes
    static
    swagger
    templates
    utils
    __init__.py
    config.py
  instance
  migrations
  venv
  .env
  .gitignore
  createdb.py
  README.md
  requirements.txt
  run.py
  schema
```

|-- app	
|-- |-- controllers	Contains all API controllers
|-- |-- decorators	Contains decorators like jwt_required, handle_errors.
|-- |-- errors	Contains custom error handlers
|-- |-- forms	Contains Flask_WTForms
|-- |-- models	Contains 9 Database Models
|-- |-- routes	Contains routes for users and the API
|-- |-- static	Contains images, documents, CSS and JS files
|-- |-- swagger	Contains custom OpenAPI dictionaries for flasgger
|-- |-- templates	Contains HTML files for the website.
|-- |-- utils	Contains helpers and file utils.
|-- |-- __init__.py	Contains flask app, blueprint and other route handlers
|-- |-- config.py	Contains configuration variables
|-- instance	Contains website's database
|-- migrations	Contains database migration details
|-- venv	Contains virtual environment files
|-- .env	Contains secret keys and other sensitive variables
|-- .gitignore	Contains files to exclude from Git
|-- createdb.py	Contains script to populate the database
|-- README.md	Contains documentation for the website
|-- requirements.txt	Contains required modules and extensions
|-- run.py	Contains the code to initialize the app
|-- schema	Contains the DB schema (dbdiagram)

API Design and Endpoints:

In the project, you will see API playing a major role in the data transmission. There is Swagger UI integration using flasgger module for efficient documentation and testing. It includes a total of 53 API routes each designed to fulfil specific functionality. It uses JWT Bearer key authentication to secure access to its API endpoints. Proper authorization is mandatory to avail the use of all features with proper functionality. Below are some of the API routes:



POST	/api/customers	Create a customer	create_customer_controller	
POST	/api/professionals	Create a professional	create_professional_controller	
POST	/api/gettoken	Get JWT access token	get_jwt_token_controller	
POST	/api/upload	Upload an image or document.	upload_controller	🔒
GET	/api/services	Get all services	get_services_controller	
GET	/api/service_request/{id}	Get a specific service request by ID	get_service_request_controller	🔒
GET	/api/service/{id}	Get a specific service by ID	get_service_controller	
GET	/api/search	Search	search_controller	
DELETE	/api/service_request/{id}	Delete a specific service request by ID	delete_service_request_controller	🔒
PUT	/api/professional/{id}	Edit a specific professional by ID	edit_professional_controller	🔒
PUT	/api/customer/{id}	Edit a specific customer by ID	edit_customer_controller	🔒

Video:

https://drive.google.com/file/d/1y1uLCuosbxgqoy0LxiZMbmPQMIXdLov-/view?usp=sharing