Project Report

M.R.Hikmathun Nisha

I am a college student with a passion for programming and data science. Working on this project was a delightful experience. Although I faced some challenges along the way, the sense of accomplishment after overcoming them made the effort truly rewarding.

Description:

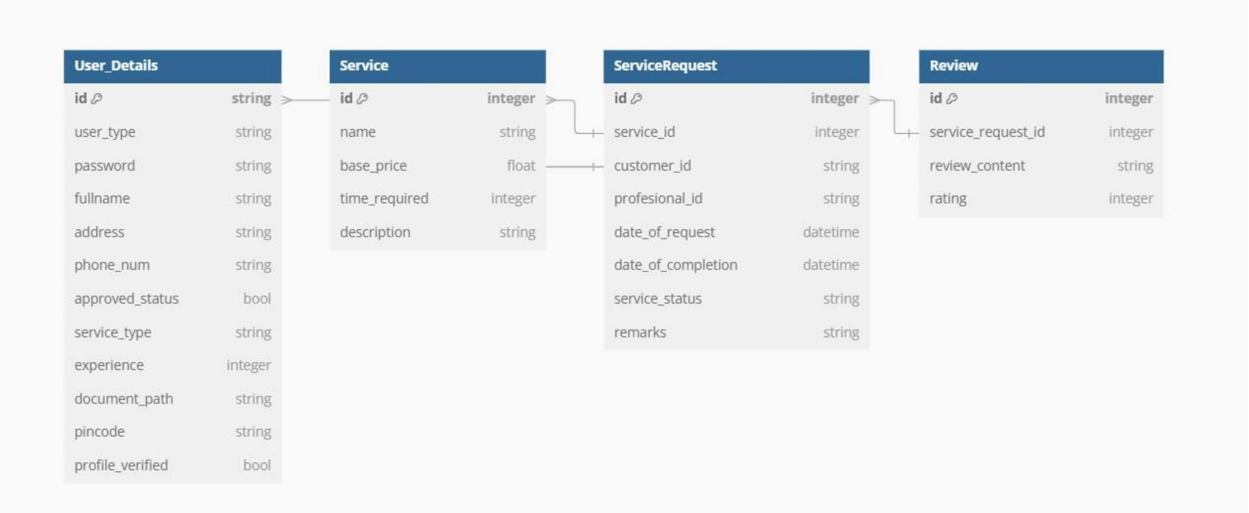
The User_Details class is a comprehensive representation of all users in the system, encompassing Admins, Professionals, and Customers. Each user is uniquely identified by an id and is categorized by a user_type, which specifies their role. The password field securely stores user credentials, while the full name field captures the names of Professionals and Customers. For Professionals, additional details such as address, phone_num, and pincode are stored to facilitate service assignments. The approved_status field tracks whether a Professional's profile has been approved by an Admin, and service_type and experience detail the type of services offered and their years of experience. The document_path field allows for uploading relevant verification documents. Lastly, the profile_verified field ensures the profile has been reviewed and approved by an Admin. This model also supports relationships for managing service requests and accepted jobs through the service_requests and accepted_requests attributes.

Technologies used

1. Flask: used for building the web application.
2. Flask-SQLAlchemy: extension of Flask, used to handle database connections across the
app.
3. Flask-RESTful: an extension of Flask, used as the back-end which builds and handles APIs
for the endpoints.
5. Datetime: This python module is used for time stamping the class operations.
6. security: for username and password
7. Requests:This python module is used to handle HTTP requests and responses.

DB Schema Design

The User_Details class establishes two key relationships with the ServiceRequest model. The service_requests relationship links a customer (user) to multiple service requests they create, using customer_id as the foreign key. The accepted_requests relationship connects a Professional to service requests they accept, using professional_id as the foreign key.



User_Details	
id	string
user_type	string
password	string
fullname	string
address	string
phone_num	string
approved_status	bool
service_type	string
experience	integer
document_path	string
pincode	string
profile_verified	bool

Service	
id	integer
name	string
base_price	float
time_required	integer
description	string

ServiceRequest	
id	integer
service_id	integer
customer_id	string
profesional_id	string
date_of_request	datetime
date_of_completion	datetime
service_status	string
remarks	string

Review	
id	integer
service_request_id	integer
review_content	string
rating	integer

API Design

This API design implements CRUD operations for managing UserCustomer (customers) and Service resources. It provides RESTful endpoints for creating, reading, updating, and deleting customer and service data, as well as searching specific records by ID. Resources are added to the Flask-RESTful API with clear and structured routes for operations like /api/get_customer, /api/add_service, and /api/search_service/<id>.

Architecture and Features:

The project code is organized based on its utility in different files. I named my project HomeCare the Home app folder: there are the Python pages named __init__ , API, models, and views. There is also the sqlite page. Further, there is a static folder that contains all the images that were used for the background, The templates folder contains all the HTML files. The app.py folder is situated outside the FlashyPep folder. The HTML files include the landing page, signup page, login page, dashboard page, and review page. On the review page, hover over the flashcard to see the backside. The HTML pages are rendered through User, customer, professional