Project Report

**Introduction:** The project is a comprehensive web application designed to streamline home service management. It facilitates seamless interactions among Admins, Professionals, and Customers. With features like dynamic dashboards, feedback systems, and responsive design, it ensures efficient service delivery, user engagement, and insightful analytics, promoting an organized and user-friendly experience.

**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. **Bootstrap**: A CSS framework for responsive and mobile-first web design.
4. **Jinja2 Templates**: A templating engine used to render dynamic content in the frontend.
5. **Datetime**: A Python module used for handling timestamps for various database operations.
6. **Matplotlib**: A Python library used for generating visual insights like graphs and charts.
7. **SQLite**: A lightweight database engine used for storing application data.
8. **Security**: Implements secure password hashing and validation mechanisms for user authentication.
9. **Requests**: A Python module used for handling HTTP requests and responses within the application.

**System Overview:** The system is a multi-role web application with three key roles: Admin, Professional, and Customer. Admins oversee service operations, user management, and system performance. Professionals offer services, manage requests, and interact with customers. Customers can browse, request, and provide feedback on services. The system integrates features like service tracking, feedback collection, and data visualization through interactive dashboards, ensuring efficiency and engagement.

**System Architecture:** The system follows an MVC (Model-View-Controller) architecture for scalability and maintainability:

1. **Model**: The database layer manages entities like User, Service, ServiceRequest, and Feedback, ensuring proper relationships and constraints. SQLAlchemy ORM is used for efficient data manipulation.
2. **View**: Frontend templates built with Flask, Jinja2, and Bootstrap offer responsive designs for user interfaces. Dynamic components like search and accordion menus enhance user interaction.
3. **Controller**: Flask routes handle API requests, user interactions, and data processing. Controllers manage user authentication, service requests, and feedback submissions.

The architecture ensures clean separation of concerns, facilitating ease of development and extensibility.

**Data Models**

The system comprises the following data models:

**User Model**

* **Attributes**: id, username, name, email, phone, street, city, state, postal\_code, password, is\_admin, is\_professional, is\_customer, photo, description, experience, created\_at
* **Description**: Represents users of the system, including their personal details and roles. The roles differentiate between customers, professionals, and administrators.

**Service Model**

* **Attributes**: id, name, ServiceType, price, description, status\_service, time\_required, created\_at, professional\_id
* **Description**: Represents the services offered by professionals, including details such as type, price, and professional association.

**ServiceRequest Model**

* **Attributes**: id, service\_id, customer\_id, professional\_id, service\_status, messages, date\_of\_request, date\_of\_completion
* **Description**: Represents requests made by customers for specific services, tracking their status and communication.

**Feedback Model**

* **Attributes**: id, service\_request\_id, rating, remarks
* **Description**: Captures customer feedback for completed service requests, including ratings and comments.

**Conclusion**: the system efficiently manages users, services, and service requests with roles for admins, professionals, and customers. It enables seamless service bookings, feedback collection, and data handling, offering an intuitive experience. This project provides a scalable and functional solution for service management and user interaction.

**Wireframe:**

The controllers that were used:

* @app.route("/")
* @app.route("/<user\_type>/register", methods=['GET', 'POST'])
* @app.route('/login', methods=['GET', 'POST'])
* @app.route("/logout")
* @app.errorhandler(404)
* @app.route('/<user\_type>/dashboard', methods=['GET', 'POST'])
* @app.route('/<user\_type>/dashboard/<int:id>', methods=['GET', 'POST'])
* @app.route('/<user\_type>/feedback/<int:service\_id>', methods=['GET', 'POST'])
* @app.route('/<user\_type>/create\_request/<int:service\_id>', methods=['GET', 'POST'])
* @app.route('/<user\_type>/profile', methods=['GET', 'POST'])
* @app.route('/<user\_type>/password\_change', methods=['GET', 'POST'])
* @app.route('/<user\_type>/editprofile', methods=['GET', 'POST'])
* @app.route('/<user\_type>/profile\_action', methods=['GET', 'POST'])
* @app.route('/<user\_type>/myservices', methods=['GET', 'POST'])
* @app.route('/<user\_type>/addservice', methods=['GET', 'POST'])
* @app.route('/<user\_type>/service\_action/<int:service\_id>', methods=['GET', 'POST'])
* @app.route("/<user\_type>/search", methods=["GET",'POST'])
* @app.route('/<user\_type>/summary', methods=['GET', 'POST'])