Project Report for Final Project Modern Application Development - I

Author :

Sanjay B

Roll No: 22f3001023

IITM email: 22f3001023@ds.study.iitm.ac.in

Hi, I'm Sanjay. I'm 21 years old and have been working as a sales executive for the past three years while pursuing this degree. Although I didn't study computer science until 12th grade, I learned programming through this course and have enjoyed acquiring these skills from IITM. This project was both challenging and rewarding, testing my coding abilities thoroughly. Despite the obstacles, I had fun tackling them and successfully completed the assignment.

Detailed Project Report

Project Overview

This project is a service management application developed using Flask, SQLAlchemy, HTML, CSS, and other associated modules. It includes features for user authentication, management of professional services, and customer interaction with service providers. Below are the specific details:

Technologies Used

1. **Flask**: Web framework for handling routing and views.
2. **Flask-SQLAlchemy**: ORM tool to manage the database schema and operations.
3. **Flask-Sessions**: For session management.
4. **Datetime Module**: For handling timestamps related to user activity.
5. **HTML, CSS, JavaScript**: For the frontend.

The extracted project structure includes the following elements:

- **static**: Contains static assets like images
- **instance**: A folder for configurations or database files.
- **routes.py**: Contains the route definitions and possibly the logic for API endpoints.
- **models.py**: Defines the database schema and relationships.
- **app.py**: The main entry point of the Flask application.
- **templates**: Contains HTML files for rendering the frontend.

Project Report for Final Project Modern Application Development - I

Features

1. **User Management**:
    - Two types of users: Professionals and Customers.
    - Login/logout functionality with session-based access control.
    - User authentication using a decorator (login_required) to protect certain routes.
2. **Service Management**:
    - Professionals can define services they offer.
    - Customers can browse and request services.
    - Requests are managed through a relational database model.
3. **CRUD Operations**:
    - Create, Read, Update, and Delete operations are supported for users, services, and requests.
    - API endpoints facilitate backend interactions.
4. **Database Design**:
    - The schema supports user roles, services, and service requests.
    - SQLite is used for the database.

Code Structure

1. **app.py**:
    - Main entry point of the application.
    - Configures the Flask application, initializes the database, and imports routes.
2. **models.py**:
    - Contains ORM definitions for tables:
        - User: Stores details of both professionals and customers.
        - Service: Stores details of services provided by professionals.
        - ServiceRequest: Tracks requests made by customers.
3. **routes.py**:
    - Handles endpoints for user actions, authentication, and service management.
    - Implements decorators for access control and features like flash messaging.

4. **templates**:
    - HTML files include:
        - Login page (login.html, adminlogin.html, login_professoinal.html)
        - Registration page (register.html, register_professional.html)
        - Dashboard (index.html, customer_dashboard.html, professional_dashboard.html)
        - Service request interface (createservices.html, editservices.html)
5. **static**:
    - Have images

API Design

- **User Management**:
    - Login, logout, and registration endpoints.
    - Flash messages for feedback (e.g., invalid login).
- **Service Handling**:
    - Professionals can manage their services.
    - Customers can view available services and make requests.

Database Schema

Explanation of the Database Schema

This project implements a database schema for a service management application. Below is a detailed breakdown of the schema and relationships:

1. Users Table

- **Purpose**: Stores information about both professionals and customers.
- **Attributes**:
    - id: Primary key, unique identifier for a user.
    - username: User's name.
    - email: Unique email address for login.
    - password: Hashed password for security.
    - mobile_no, address, pincode: Optional personal details.

  - role: Defines if the user is a "professional" or a "customer."
  - active: Boolean indicating if the user account is active.
- **Relationships**:
  - Links to **Professionals** and **Customers** tables with one-to-one relationships.
  - Deleting a user cascades deletions in related tables.

2. Services Table

- **Purpose**: Defines the services offered by professionals.
- **Attributes**:
  - id: Primary key, unique identifier for a service.
  - name: Name of the service (e.g., Plumbing, Cleaning).
  - price: Cost of the service.
  - time_required: Estimated time to complete the service.
  - description: Details about the service.
- **Relationships**:
  - Linked to the **Professionals** and **Service Requests** tables.
  - Deleting a service cascades deletions in related records.

3. Professionals Table

- **Purpose**: Links professional users to their offered services.
- **Attributes**:
  - id: Primary key, unique identifier for a professional.
  - user_id: Foreign key linking to the **Users** table.
  - service_id: Foreign key linking to the **Services** table.
  - description: Additional details about the professional.
- **Relationships**:
  - Linked to **Service Requests** to track services provided by professionals.
  - Deleting a professional cascades deletions in related records.

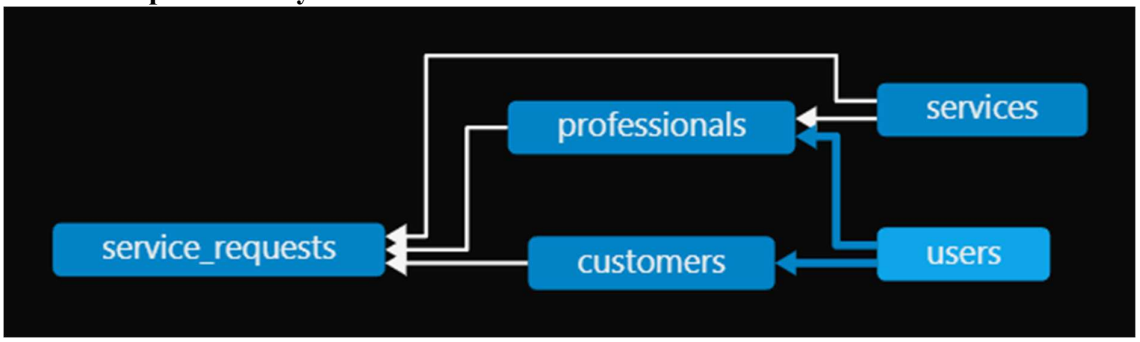
4. Customers Table

- **Purpose**: Stores customer-specific details.
- **Attributes**:
  - id: Primary key, unique identifier for a customer.
  - user_id: Foreign key linking to the **Users** table.
  - address, phone: Personal details for communication.
- **Relationships**:
  - Linked to **Service Requests** for tracking customer requests.
  - Deleting a customer cascades deletions in related records.

5. Service Requests Table

- **Purpose**: Manages service requests made by customers.
- **Attributes**:
  - id: Primary key, unique identifier for a service request.
  - service_id: Foreign key linking to the **Services** table.
  - customer_id: Foreign key linking to the **Customers** table.
  - professional_id: Foreign key linking to the **Professionals** table (nullable for unassigned requests).
  - status: Current status of the request (e.g., Pending, Completed).
- **Relationships**:
  - Acts as the central table connecting services, customers, and professionals.

Relationships Summary



1. **Users Table**:
   - One-to-one relationship with both Professionals and Customers.
2. **Services Table**:

   - One-to-many relationship with Professionals.
   - One-to-many relationship with Service Requests.
3. **Service Requests Table**:
   - Many-to-one relationship with Services.
   - Many-to-one relationship with Customers.
   - Many-to-one relationship with Professionals.

Highlights of the Schema Design

- **Scalability**: Designed to support multiple professionals offering the same service and multiple customers requesting services.
- **Flexibility**: The nullable professional_id in Service Requests ensures unassigned requests can exist.
- **Cascading**: Deleting a user, service, or professional cascades the deletions to dependent records, ensuring database integrity.

Architecture

- The application follows the MVC architecture:
  - **Models**: Defined in models.py.
  - **Views**: Rendered by Flask using templates.
  - **Controllers**: Defined in routes.py.

Execution Steps

1. Run the Flask application via app.py.
2. Use the database auto-created by SQLAlchemy to store user and service data.
3. Access the application through defined routes to manage services and requests.

Project presentation Video: https://youtu.be/t8In0UjHDoI?si=hr3ULsswhfLk4mhH

Project URL:

Github: https://github.com/spideysanjay007/A-Z-Services-App-MAD-1-.git

Drive:
https://drive.google.com/drive/folders/1WA1nF3bhLaom0EuLWnGnTb9qNv1UCoWR?usp=drive_link