MAD-1 PROJECT FINAL REPORT

Name: Shruti Sinha
Roll Number: 23F2002761
Student Email: 23f2002761@ds.study.iitm.ac.in
About Me: I am an undergraduate student who is fascinated by the world of web and how its has come to existence. My interest in creating web pages began when I first learned HTML and built my very first webpage in 8th grade—it felt like magic to me back then. Unfortunately, I couldn't continue exploring this field due to a lack of guidance. This project has given me the opportunity to reconnect with that passion and refine my skills in web development.

Description

This project is a **Vehicle Parking Management System** that allows users to register, book parking spots, and track their parking history, while admins can manage parking lots, spots, and monitor reservations. It uses **Flask** as the backend framework with **SQL Alchemy ORM** for database interaction and Jinja2 for templating.

AI/LLM Used: Approximately 20–30%.
AI was used for debugging and code optimization not for writing the full code.

Technologies Used

- **Flask** – Core backend framework to handle routing and application logic.
- **Flask-SQL Alchemy** – ORM for managing database models and queries efficiently.
- **SQLite** – Lightweight relational database for easy integration and development.
- **Jinja2** – Templating engine for rendering dynamic HTML pages.
- **Werkzeug** – Used for password hashing and authentication security.
- **Bootstrap** – Front-end framework for building responsive and clean UI.

DB Schema Design

Tables

User

- **id:** Integer, Primary Key, Auto Increment
- **full_name:** String (100), Nullable
- **username:** String (100), Unique, Not Null
- **email:** String (120), Unique, Nullable
- **password:** String (100), Not Null
- **is_admin:** Boolean, Default = False

ParkingLot

- **id:** Integer, Primary Key
- **location_name:** String (100), Not Null
- **address:** String (200), Not Null
- **pin_code:** String (6), Not Null
- **price_per_hour:** Float, Not Null
- **max_spots:** Integer, Not Null

ParkingSpot

- **id:** Integer, Primary Key
- **lot_id:** Foreign Key -> ParkingLot.id (Cascade Delete)
- **status:** String (1), Default = 'A' (A = Available, R = Reserved, O = Occupied)

Reservation

- **id:** Integer, Primary Key
- **spot_id:** Foreign Key ->ParkingSpot.id
- **user_id:** Foreign Key -> User.id
- **start_time:** DateTime, Nullable
- **end_time:** DateTime, Nullable
- **status:** String (20), Default = 'active'
- **cost:** Float, Nullable (calculated at checkout)

Design Reasoning

- Used **normalized tables** to maintain data integrity and scalability.
- Established **relationships**:
    - One ParkingLot -> Many ParkingSpots
    - One User -> Many Reservations
- Stored cost in Reservation for **billing and historical tracking**.
- status fields allow quick availability and reservation state checks.

API Design

I created APIs for user authentication, admin operations, and user-related actions. These were implemented using **Flask routes** and organized using **Blueprints** to keep everything modular. Data mostly flows through HTML forms for the web interface, but the logic behind them is structured like an API.

What I Built

- **Authentication:**
    - Users can register by entering their details (username, email, password), and login using email and password. Passwords are stored in a hashed format for security.
- **Admin Functions:**
    - Admin can create, update, or delete parking lots.
    - Auto-generate parking spots when a lot is created or updated.
    - A search feature is added so admin can quickly find users or check spot status.
- **User Functions:**
    - Users can reserve an available parking spot in a lot.
    - They can mark parking in and out. When they check out, the app calculates cost based on duration and lot's price per hour.
    - Users can also view their reservation history.

How I Implemented It

- I used **Flask Blueprints** for clear separation: one for authentication, one for admin tasks, and another for user dashboard.
- **Flask-SQL Alchemy** handles all database queries and relationships.
- The API endpoints are fully documented in a separate **YAML file (OpenAPI spec)** so they can be tested with tools like Swagger or Postman.

Architecture and Features

For this project, I organized my Flask application in a modular way to keep everything neat and manageable. I used Blueprints to separate different parts of the app: authentication.py handles login and registration, admin.py is for managing parking lots and spots, and user.py covers reservations and user dashboard actions. All these route files are placed inside a routes folder.

The HTML templates live in the templates folder and are rendered using Jinja2, while CSS files and images are stored in the static folder. The database models are all defined in models.py, and the app's configuration and initialization logic is kept cleanly in __init__.py.

Features Implemented

The main features include **user authentication** (register/login) with password hashing for security, and **role-based dashboards** for admin and users. Admin can create, update, and delete parking lots, auto-generate spots based on the lot's capacity, and search for users or spot details. Users can reserve parking spots, mark check-in and check-out, and see their reservation history. The app also calculates the parking cost based on duration and price per hour.

Additional Features:

- Responsive UI using **Bootstrap** for better user experience.

- Search functionality for admin to quickly find users or spots.
- Status management for each parking spot (Available, Reserved, Occupied).
- Reservation summary for both users and admin, showing history and cost.

Video Link:
https://drive.google.com/file/d/1P-M7QV6eLQ7g7fsztcRJNTjSWLsYkfHy/view?usp=drive_link