Author

Bhargav

23f3003696

23f3003696@ds.study.iitm.ac.in

I'm currently pursuing the Online Degree Program and have always been curious about how web applications work behind the scenes. This project was a great learning experience where I got to build something from scratch, make mistakes, fix them, and see it all come together.

Description

This project is about building a web-based vehicle parking management system. It includes admin and user roles, where admins manages parking operations. The goal is to simplify parking operations and provide clear summaries and reports.

AI/LLM (e.g., ChatGPT, Copilot)

Purpose / Role in Project: Helped with JS logic, debugging and code suggestions(15%)

Technologies used

Flask – Lightweight web framework to build the application.

Flask-SQLAlchemy – ORM to manage database models and queries easily.

Flask-Login – Handles user session management and authentication.

SQLite – Lightweight, serverless database for local development and storage.

HTML/CSS + Bootstrap – For designing responsive, user-friendly interfaces.

Chart.js – To generate interactive summary charts for admin and users.

Jinja2 – Template engine for rendering dynamic content in HTML.

JavaScript (JS) – Enables dynamic behaviour in forms, modals, and charts (e.g., modal popups, real-time updates).

DB Schema Design

- Users table stores login credentials along with user details like full name, address, and pincode.
- Lots table represents each parking lot with fields like name, location, address, price, and max number of spots.
- Spots are individual parking spaces linked to lots, tracking status (empty or occupied), vehicle info, and timestamps.

Why this design?

This structure keeps the data clean and organized. Using foreign keys maintains clear relationships between entities. It also makes it easier to track parking usage, calculate billing, and generate admin reports without redundant data.

API Design

This project primarily uses Flask routes for handling user actions and rendering dynamic content via server-side templates. While not structured as a public REST API, form submissions and JavaScript-based chart rendering internally rely on route endpoints that act like APIs in behaviour. For example, parking data is processed and returned via Flask routes to dynamically update dashboards.

Architecture and Features

The project is structured using Flask's Blueprint system for separation of routes (controllers/routes.py), models (models/models.py), and configuration (app.py). Templates are organized in the templates directory, while static assets like CSS are placed under static/.

Core features include user/admin login, parking spot booking and releasing, lot management by admin, and auto-assignment of spots. Admins can view all users, edit lots, and monitor parking activity through summary charts. Another feature is search that enables filtering lots and users efficiently.

Video

Google-drive video link:
https://drive.google.com/file/d/1B7U_vOPreuI8obgDZFAVb3l8YqfdV31X/view?usp=sharing