Vehicle Parking App

Author

Name: Vedant Deepak Gawande
Roll Number: 23f2003038
Student Email: 23f2003038@ds.study.iitm.ac.in

Description

This project is about creating a simple web app to manage 4-wheeler parking efficiently. It should let admins create and monitor parking lots, while users can log in, book a spot, and release it when done. The goal is to make the whole parking process smooth, automated, and easy to use.

AI assistance was used only in the backend, mainly for writing and refining syntax in Flask routing and SQLAlchemy models. It contributed to about 10–15% of the total work, strictly limited to backend logic and database structure—no AI help was used in the frontend, styling, or templates.

Technologies used

- **Backend Framework:**
  - **Flask (v2.3.3)**: Lightweight Python web framework for routing, templating, and backend logic
- **Database & ORM**
  - **Flask-SQLAlchemy (v3.0.5)**: Integrates SQLAlchemy with Flask for simplified model management and relationships.
  - **SQLAlchemy (v2.0.21)**: ORM layer offering a powerful query interface and schema handling.
  - **SQLite**: Lightweight, file-based DB requiring no external setup — perfect for local development
- **Security & Authentication**
  - **Werkzeug (v2.3.7)**: Handles password hashing and session utilities; core dependency of Flask.
- **Time & Date Handling**
  - **pytz:** Enables timezone-aware datetime operations (used for Indian Standard Time).
  - **tzdata:** Provides accurate timezone data for calculations and billing.
- **Frontend Technologies**
  - **Bootstrap (v5.1.3)**: CSS framework for building responsive, mobile-first UI.
  - **Font Awesome (v6.0.0)**: Provides clean, scalable icons for UI enhancements.
  - **Jinja2**: Templating engine for rendering dynamic HTML using Flask data.

DB Schema Design

1. **User**

- Columns: id, username, email, password_hash, created_at
- Constraints: username and email are unique and required.
- Purpose: Stores registered user credentials and links to their reservations.

2. **Admin**

- Columns: id, username, password_hash, created_at
- Constraints: One-time seeded; no registration. Username is unique.
- Purpose: Superuser with privileges to manage parking lots and users.

3. **ParkingLot**

- Columns: id, location_name, hourly_rate, address, pin_code, total_spots, created_at
- Constraints: location_name and pin_code are required.
- Purpose: Defines each parking location and its pricing/capacity.

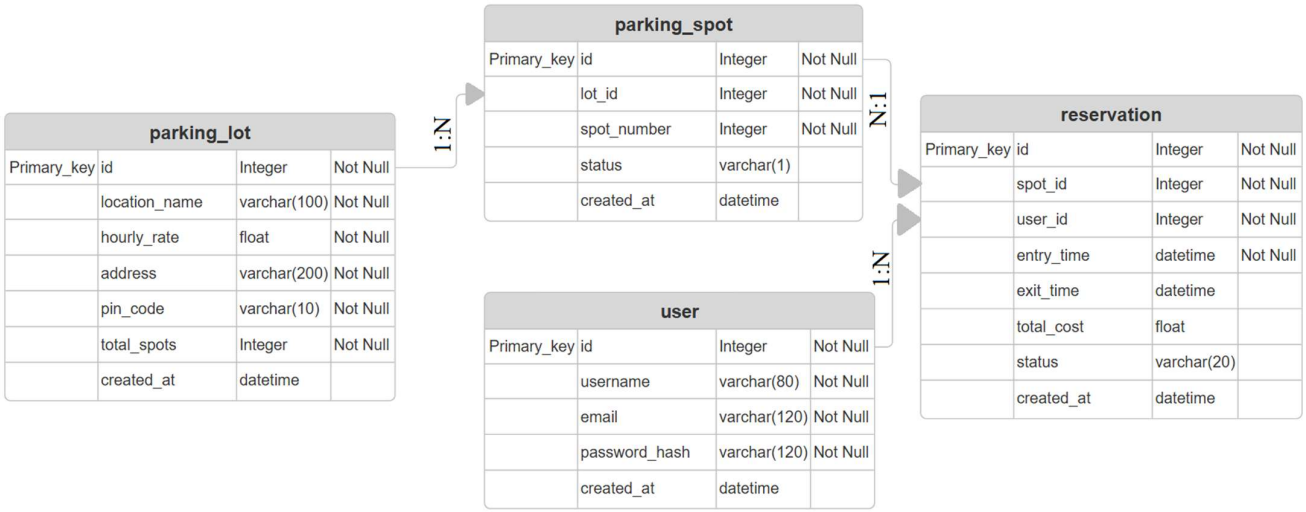
4. **ParkingSpot**

- Columns: id, lot_id (FK), spot_number, status, created_at
- Constraints: Linked to a parking lot via lot_id; status is 'A' (available) or 'O' (occupied).
- Purpose: Represents individual spots within each lot.

5. **Reservation**

- Columns: id, spot_id (FK), user_id (FK), entry_time, exit_time, total_cost, status, created_at
- Constraints: Tracks active or completed bookings.
- Purpose: Records user bookings, timestamps, and costs.

Entity-Relation Diagram



API Design

To make the application more dynamic and interactive, I created an API that lets the backend share real-time data with the frontend — without needing to reload the page. This helps improve the user experience, especially for admins who want quick insights into what's happening in the parking system

Example: Admin Parking Statistics API

- Endpoint: /api/parking_stats
- Purpose: To provide real-time parking lot data for the admin dashboard.
- Data Returned:
    - Total spots
    - Occupied and available counts
    - Per-lot status summary
- Implementation Details:
    - Built using Flask's @app.route() decorator.
    - Queried the database using SQLAlchemy.
    - Used jsonify() to return structured JSON data.
    - Access restricted to logged-in admins via session checks.

Architecture and Features

The project is organized in a clean and simple way that makes it easy to understand and work with. All the main backend logic — like routing, session handling, and database operations — lives inside the app.py file. That’s basically the brain of the application. The HTML files that users see are kept in the templates/ folder and use Jinja2 to display dynamic content. Styling and interactivity are handled through the static/ folder, which holds CSS, JavaScript, and image files. I used Flask-SQLAlchemy to define my database models, and the database is created automatically when the app runs — no need to set it up manually. This setup keeps things neat and organized while making development smooth.

Core Features

- Separate login for Admin and Users
  The admin doesn't need to register — they can log in directly. Users can register with a username, email, and password. I used Flask sessions to manage logins securely.
- Admin Dashboard
  The admin can see all parking lots, the number of available and occupied spots, and even details of registered users. It's basically their control panel.
- Create, Edit, and Delete Parking Lots
  The admin can set up new parking lots by just entering details and the number of spots — the app automatically generates the spots. Editing and deletion are also allowed, but only when no spots are in use.
- User Dashboard
  Users can see their current and past parking activity, including how long they parked and what they paid.

- Booking a Parking Spot
  Users just pick a parking lot — the app finds the first available spot for them. It's all automatic, no need to choose manually.
- Releasing a Spot
  When users leave, they release their spot. The app records the exit time and calculates the total cost based on how long they were parked.

Extra Features

- Live Admin Stats via API
  I created a custom API endpoint (/api/parking_stats) so the admin can get real-time updates on parking availability and usage, perfect for adding charts or dashboards.
- Search Function While Booking
  Users can search for parking lots using a location name, address, or pin code — makes it easier to find a nearby spot.
- Accurate Time & Cost Calculations
  I used timezone-aware datetime with the pytz library to make sure the cost calculations are spot-on, no matter when users check in or out.

Video

Link: Mad 1 Project Video