Vehicle Parking --- V1

Author

Vaibhav Pandey
Roll No: 23f3004149
Email: 23f3004149@ds.study.iitm.ac.in
I am a student at IIT Madras, pursuing a Bachelor of Science (BS) in Data Science and Applications. I enjoy solving real-world problems through technology and love building end-to-end systems.

Description

This project is a web-based vehicle parking management system that enables users to reserve and release parking spots while allowing administrators to manage parking lots and monitor spot usage. The application provides a seamless, role-based interface and ensures real-time availability and historical tracking of parking data.

Users can search for lots, reserve available spots, and track their reservation history. Admins have the capability to manage cities, lots, and spots and oversee user activities. The system ensures data consistency, secure access, and a clean user experience.

Technologies Used

Backend: Flask

Flask Extensions:
-Flask-Login (Authentication)
- Flask-Bcrypt (Password Hashing)
- Flask-SQLAlchemy (ORM)
- Flask-Migrate (Database Migrations)
- Flask-CORS (CORS headers)
- Flask-RESTful (API routing)

Frontend: HTML, CSS, Bootstrap, Jinja2, JavaScript

Database: SQLite

API Format: REST (JSON)

Purpose

Flask provides a lightweight yet powerful backend framework ideal for rapid development. Its modularity and extension ecosystem enabled secure authentication, structured database management, and seamless API design. Jinja2 made rendering dynamic HTML content intuitive. Bootstrap was used to design a responsive and clean user interface. SQLite was chosen for simplicity and efficient local development.

The project aims to simulate a practical parking reservation system that could be scaled and deployed in real-world settings.

DB Schema Design

User: Stores full name, email, address, pin code, role (user/admin), and authentication data. Related to reservations.

Admin: Predefined login with separate role and credentials.

City: Manages parking lots by geographical location (name, state). One city has many lots.

ParkingLot: Belongs to a city; contains details like prime location, address, capacity, and rate. Related to spots.

ParkingSpot: Created automatically per lot; each spot has status (Available/Occupied). Linked to reservations.

Reservation: Stores reservation metadata such as spot, user, timestamps, vehicle number, cost, and status. Duration and cost are auto-calculated.

Key Design Highlights:
- One-to-many and many-to-one relationships are implemented using SQLAlchemy.
- Auto-spot generation when creating a parking lot.
- Cascade deletes to ensure cleanup of dependent entities.
- Role-based access via custom session checks.
- Spot status dynamically updates on reservation/release.
- Parking cost is calculated based on timestamps and hourly rate.

Architecture and Features

The application follows a modular folder structure:
- app.py: Main application runner. Initializes extensions and registers blueprints.
- models.py: Defines all SQLAlchemy models and their relationships.
- controllers/: Contains blueprints for user, admin, auth, and API logic.
- templates/: Jinja2 HTML templates for user/admin interfaces.
- static/: Contains static files including Bootstrap and images.
- api_doc.yaml: OpenAPI documentation for the RESTful API.

Admin Features:
- Login with predefined credentials.
- Create, edit, and delete parking lots.
- Auto-generate parking spots based on lot capacity.
- View status of all spots (Available/Occupied).
- Search for lots and users.
- Filter spots based on occupancy.

User Features:
- Register/login with full name, email, address, pin.
- View and search lots by pincode, name, city, and state.
- Reserve the first available spot in a lot.
- Release the spot — auto-update timestamps and cost.
- Track current reservations and full history.

Frontend Highlights:
- Responsive UI with Bootstrap Cards, Badges, Alerts.
- Clean dashboards for user and admin views.
- Flash messages for feedback.
- AJAX-powered search for lots and filters.

Backend & API:
- Flask Blueprints used to modularize routes.
- JSON APIs created manually.
- SQLAlchemy relationships ensure data consistency.
- Security via Flask-Login and session-based role checks.

AI Declaration

I declare that AI tools (like ChatGPT) were used only for:
- Debugging Flask and Bootstrap issues
- UI/UX suggestions

The overall contribution of AI assistance is estimated at less than 10%. All core logic, database design, authentication flow, and API implementation were developed independently.

Api Document   Demo Video