Project Report - Vehicle Parking Management System

Author

Name: Aryan Patil
Roll Number: 23f1000968
Email: 23f1000968@ds.study.iitm.ac.in
About: BS Degree student specializing in Data Science and Applications at IIT Madras. Passionate about building practical web applications that solve real-world problems.

Description

This project implements a multi-user vehicle parking management system that digitizes parking lot operations. The system enables administrators to manage multiple parking lots and allows users to book and release parking spots with automatic cost calculation based on duration.

AI/LLM Usage: Approximately 35% - Used for debugging Flask routes, understanding SQLAlchemy relationships, and implementing search functionality. All code was reviewed and modified to match project requirements.

Technologies Used

- **Flask 3.1.1** - Lightweight web framework for backend logic and routing
- **SQLAlchemy ORM** - Database abstraction layer for cleaner database operations
- **SQLite** - File-based database for simple deployment and testing
- **Jinja2** - Template engine for dynamic HTML generation
- **Bootstrap 5.1.3** - CSS framework for responsive and modern UI
- **Werkzeug** - Security utilities for password hashing

Purpose: Flask was chosen for its simplicity and flexibility. SQLAlchemy provides protection against SQL injection while SQLite offers zero-configuration database setup perfect for academic projects.

DB Schema Design

Tables Structure:

1. **users** (id, username, email, password_hash, created_at)
    - Primary key: id
    - Unique constraints: username, email
2. **admins** (id, username, password_hash)
    - Primary key: id
    - Unique constraint: username

3. **parking_lots** (id, prime_location_name, price_per_hour, address, pin_code, maximum_spots, created_at)
    - Primary key: id
4. **parking_spots** (id, lot_id, spot_number, status)
    - Primary key: id
    - Foreign key: lot_id → parking_lots.id
    - Status values: 'A' (Available), 'O' (Occupied)
5. **reservations** (id, spot_id, user_id, vehicle_number, parking_timestamp, leaving_timestamp, total_cost, is_active)
    - Primary key: id
    - Foreign keys: spot_id → parking_spots.id, user_id → users.id

Design Rationale: The schema follows normalization principles with proper relationships. Cascade delete ensures data integrity when removing parking lots. The reservation table tracks both active and historical parking data for analytics.

API Design

While the current implementation uses traditional form-based interactions, the architecture supports easy API addition. The search functionality demonstrates JSON-capable endpoints that could be extended to full RESTful APIs for parking operations.

Architecture and Features

The project follows MVC pattern with clear separation:

- **Models** (`/models/database.py`): Database schema and relationships
- **Controllers** (`app.py`): Route handlers and business logic
- **Views** (`/templates/`): HTML templates with Jinja2
- **Static** (`/static/`): CSS and client-side assets

Core Features:

- Automatic admin creation on first run
- Secure authentication with hashed passwords
- Automatic spot allocation (first available)
- Real-time cost calculation based on duration
- Cascade deletion for data integrity
- Search functionality for spots, vehicles, and locations

Additional Features:

- User parking statistics dashboard
- Responsive design for mobile devices
- Session-based access control
- Input validation on all forms

Video

Walkthrough link -
https://drive.google.com/file/d/1aMAwgBHhxLEmnxY_BvQm1dPZgwV2zIKb/view