ParKing App - Application Development 1 Project, May 2025

Project Report

1. Introduction

The ParKing Web Application is a multi-user platform designed to efficiently manage multiple parking lots and parking spots for 4-wheelers, specifically addressing modern urban parking challenges. The system defines two primary user roles: Admin and User. Admins possess superuser privileges, overseeing all lots and users, while regular users can register, book parking spots, and monitor their parking history. This document outlines the system architecture, technical implementation, and core feature set as deployed.

2. Problem Statement & Approach

Urban parking scarcity and poor management of spot allocation often lead to congestion, misuse of spaces, and administrative overload. The core objective was to design a locally hosted, user-friendly application where:

- Admins can create, edit, or delete parking lots and monitor all parking activities.
- Users can register, securely log in, find available lots, and book parking spots instantly.
- Each parking lot configuration supports dynamic pricing, maximum capacity, and real-time status tracking of all spots.

These requirements were addressed through a modular Flask web application employing clear role-based access and a normalized relational database schema

.

3. Frameworks and Libraries Used

- **Flask**: Core web framework, handling routing and session management.
- **Jinja2**: HTML templating, powering dynamic web pages for different user roles.
- **SQLite**: Lightweight, file-based database for persistent data storage and ORM for clear database manipulation.
- **HTML, CSS, Bootstrap**: For responsive and clean front-end interfaces.
- **Security**: Password hashing for admin. (Note: User password hashing can be further improved; included for admin per project requirements.)

4. Database Schema and Entity-Relationship Summary

The system's relational schema comprises the following core entities:

User	id, username, password, associated reservations.
Admin	id, username, password.
ParkingLot	id, location, address, pincode, price_per_hour, max_spots, child relationship to ParkingSpot
ParkingSpot	id, lot_id, spot_number, status (A=Available, O=Occupied), child reservations
Reservation	id, user_id, spot_id, start_time, end_time, cost

5. Major Implementation Details

5.1 User Authentication and Session Management

- Registration and Login: Users create accounts and log in via secure form routes. Sessions are maintained using Flask's built-in session feature. Admin has a preset account created at database initialization.
- Session Security: Admin and user sessions are segregated via session keys, ensuring no privilege overlap across role boundaries.

5.2 Parking Lot & Spot Management (Admin)

- Lot Creation/Editing/Deletion: Admins can add lots with parameters (location, address, pincode, price, max spots) and manage them from a dedicated dashboard.
- Dynamic Spot Assignment: Adding or editing a lot updates its available spots, automatically creating or removing ParkingSpot entries while ensuring spots cannot be edited/removed if currently occupied.
- User Viewing and Payment Calculation: The admin dashboard provides an overview of each user's bookings and total dues, calculated based on spot occupancy durations and applicable rates.

5.3 Parking Spot Booking & Release (User)

- Browse & Book: Users can view all lots, see real-time spot availability, and book directly. The system automatically allocates the lowest-numbered available spot
- Release/Finish Parking: Users end a booking session via dashboard controls, which updates both reservation end time and spot status to available.
- Cost Calculation: The application computes total parking cost by booking duration multiplied by the lot's rate per hour; cost is reflected live as well as in the booking history and on admin overviews.

5.4 Data Integrity and User Experience

- Form Validation: User input is validated both server-side and, optionally, via HTML validation for basic checks.
- Feedback: All significant actions (login, logout, booking, spot release, admin edits) generate UI feedback for user clarity.
- Responsive Dashboards: Jinja2 templates render dashboards with tables for spots, bookings, and payment summaries, tailored to user and admin perspectives.
- AI was used occasionally for error handling and logo generation during the project development.

6. Notable Features & Possible Extensions

Implemented:

- Charts on admin dashboard pages summarizing lot occupancy (facilitated with chart libraries or manual data calculation).
- Automated admin account initialization secured with password hashing for seamless and secured deployment.
- Accurate cost computation and data-driven user payment summaries.
- Modern glass inspired UI design for better customer experience.

Optional/Recommended Future Extensions:

- REST APIs for mobile or third-party integration.
- Enhanced security with hashed passwords for all users and role-based access control via Flask-Login.

7. Conclusion

The developed Vehicle Parking Lot Web Application fulfills all functional requirements: robust admin tools for real-time lot management, user-friendly interfaces for booking, and solid backend design for data integrity and future extensibility. The separation of roles, responsive dashboards, and clear data flow ensure practical usability both for small deployments and as a base for further development.

Attachments:

- Project Source Code and Directory (`code/` folder)
- API Definition (`openapi.yaml`)
- This Report (`21f1004338_MAD1_Project_Report.pdf`)