Student Details

Name: Sushant Nathuram Kadam
Roll Number: 23f1000132
Email: 23f1000132@ds.study.iitm.ac.in
Program: Modern Application Development I

Project Details

Problem Statement

The project involves developing a multi-user vehicle parking management application for 4-wheeler parking with two distinct roles: Administrator and User. The system manages parking lots, parking spots, and vehicle reservations with real-time availability tracking and cost calculation.

Approach to Solution

I approached this problem by designing a comprehensive web application using Flask framework with the following key components:

1. User Management System: Implemented role-based access control with separate dashboards for admin and users
2. Parking Lot Management: Created a hierarchical structure where parking lots contain multiple parking spots
3. Reservation System: Developed an automated booking system with first-available-spot allocation
4. Cost Calculation: Implemented time-based pricing with minimum 1-hour billing
5. Real-time Monitoring: Built dashboards with live status updates and visual charts

Frameworks and Libraries Used

Backend Technologies

1. Flask 2.3.3: Main web framework for handling HTTP requests and routing
2. Flask-SQLAlchemy 3.0.5: ORM for database operations and model definitions
3. Flask-Login 0.6.3: User session management and authentication
4. Werkzeug 2.3.7: Password hashing and security utilities
5. SQLite: Lightweight database for data persistence

Frontend Technologies

1. Jinja2: Template engine for dynamic HTML rendering
2. Bootstrap 5.1.3: CSS framework for responsive UI design
3. Font Awesome 6.0.0: Icon library for enhanced user interface
4. Chart.js: JavaScript library for data visualization and charts
5. HTML5/CSS3: Standard web technologies for structure and styling

Purpose of Technology Choices

1. Flask: Chosen for its simplicity and flexibility in rapid prototyping
2. SQLAlchemy: Provides database abstraction and relationship management
3. Bootstrap: Ensures responsive design across different device sizes
4. Chart.js: Enables interactive data visualization for admin dashboard

Database Schema Design

ER Diagram Description

Users Table:

1. `id` (Primary Key, Integer, Auto-increment)
2. `username` (Unique, String(80), Not Null)
3. `email` (Unique, String(120), Not Null)
4. `password_hash` (String(128), Not Null)
5. `phone` (String(15), Optional)
6. `is_admin` (Boolean, Default: False)
7. `created_at` (DateTime, Default: Current Time)

ParkingLots Table:

1. `id` (Primary Key, Integer, Auto-increment)
2. `prime_location_name` (String(100), Not Null)
3. `address` (Text, Not Null)
4. `pin_code` (String(10), Not Null)
5. `price_per_hour` (Float, Not Null)
6. `maximum_number_of_spots` (Integer, Not Null)
7. `created_at` (DateTime, Default: Current Time)

ParkingSpots Table:

1. `id` (Primary Key, Integer, Auto-increment)
2. `lot_id` (Foreign Key → ParkingLots.id, Not Null)

3. `spot_number` (String(10), Not Null)
4. `status` (String(1), Default: 'A') # A-Available, O-Occupied
5. `created_at` (DateTime, Default: Current Time)

Reservations Table:

1. `id` (Primary Key, Integer, Auto-increment)
2. `spot_id` (Foreign Key → ParkingSpots.id, Not Null)
3. `user_id` (Foreign Key → Users.id, Not Null)
4. `vehicle_number` (String(20), Not Null)
5. `parking_timestamp` (DateTime, Default: Current Time)
6. `leaving_timestamp` (DateTime, Optional)
7. `parking_cost` (Float, Default: 0.0)
8. `is_active` (Boolean, Default: True)

Relationships

1. One-to-Many: ParkingLots → ParkingSpots (CASCADE DELETE)
2. One-to-Many: Users → Reservations
3. One-to-Many: ParkingSpots → Reservations

Design Rationale

The schema follows normalized database design principles to minimize redundancy while maintaining referential integrity. The hierarchical structure (Lot → Spot → Reservation) allows for scalable parking management with efficient queries.

API Design

REST API Endpoints

GET /api/parking_lots

1. Purpose: Retrieve all parking lots with availability status
2. Response: JSON array containing lot details, pricing, and real-time availability
3. Implementation: Flask route returning JSONified SQLAlchemy query results

GET /api/parking_spot/{spot_id}

1. Purpose: Get detailed information about a specific parking spot
2. Response: JSON object with spot status, location, and current reservation details
3. Implementation: Query-based endpoint with conditional reservation data inclusion

API Implementation Strategy

The APIs are implemented using Flask's native JSON support rather than Flask-RESTful to maintain simplicity while providing essential data access for potential mobile applications or third-party integrations.

Architecture and Features

Project Organization

The application follows the Model-View-Controller (MVC) architectural pattern:
Models (`models.py`): Database models using SQLAlchemy ORM with relationship definitions and business logic methods
Views/Templates (`templates/`): Jinja2 templates organized by user roles:

1. `base.html`: Common layout and navigation
2. `admin/`: Administrative interface templates
3. `user/`: User interface templates

Controllers (`app.py`): Flask routes handling business logic, authentication, and data processing
Static Assets (`static/`): CSS styling and JavaScript functionality for enhanced user experience

Implemented Features

Core Features

1. Dual Authentication System: Separate login flows for admin and regular users
2. Admin Dashboard: Complete parking lot management with CRUD operations
3. User Dashboard: Parking lot browsing and spot booking interface
4. Automated Spot Allocation: First-available-spot assignment algorithm
5. Real-time Status Updates: Live parking availability tracking
6. Cost Calculation: Time-based billing with minimum charge logic

Additional Features

1. Data Visualization: Interactive charts showing parking lot utilization
2. Responsive Design: Mobile-friendly interface using Bootstrap
3. Booking History: Complete transaction history for users
4. Search and Filter: Enhanced admin controls for user and lot management
5. Form Validation: Frontend and backend validation for data integrity

6. Session Management: Secure user authentication with Flask-Login

Technical Implementation

1. Database Creation: Programmatic schema generation without manual intervention
2. Password Security: Werkzeug-based password hashing
3. Error Handling: Comprehensive error messages and user feedback
4. RESTful Routes: Clean URL structure following REST conventions

AI/LLM Usage Declaration

Extent of AI Usage

Comprehensive Development Assistance (20-30%)

Video Presentation

Drive Link:
https://drive.google.com/file/d/19mvG41b-uGgUh1_tzfi39C9fdhgD2Gc0/view?usp=sharing