City Park - Project Report

1. Author Details

Name: Kaif Fazal
Roll Number: 24f1002359
Email: 24f1002359@ds.study.iitm.ac.in
About : Student pursuing BS Data Science(IITM) and Interaction design (USDI)

2. Project Overview

Project Title: City Park – Vehicle Parking App

A web application built using Flask for managing 4-wheeler parking lots. It supports admin-level lot management and user-level booking/reservation of parking spots in real time. The system maintains all data in SQLite and uses Jinja2 for frontend rendering.

AI tools were used in this project development, the percentage of assistance was approximately **9%.**

3. Technologies Used

- Python (Flask Framework)
- Jinja2 (Templating)
- HTML, CSS, Bootstrap
- SQLite (Backend DB)
- Chart.js (Admin dashboard charts)

4. Database Schema

- User TABLE (
  id INTEGER PRIMARY KEY,
  username VARCHAR(25) UNIQUE NOT NULL,
  email VARCHAR(50) UNIQUE NOT NULL,
  full_name VARCHAR(35) NOT NULL,
  password VARCHAR(50) NOT NULL,
  phone VARCHAR(15),
  is_admin BOOLEAN DEFAULT FALSE)
- Parking_lot Table (
  id INTEGER PRIMARY KEY,
  prime_location_name VARCHAR(100) NOT NULL,
  price_per_hour FLOAT NOT NULL,
  address TEXT NOT NULL,
  pin_code VARCHAR(10) NOT NULL,
  maximum_number_of_spots INTEGER NOT NULL
- Parking_spot Table(
  id INTEGER PRIMARY KEY,
  lot_id INTEGER NOT NULL,
  spot_number INTEGER NOT NULL,

status VARCHAR(1) DEFAULT 'A',
FOREIGN KEY (lot_id) REFERENCES parking_lot(id)

- Reservation Table (
    id INTEGER PRIMARY KEY,
    spot_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    vehicle_number VARCHAR(20) NOT NULL,
    parking_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    leaving_timestamp DATETIME,
    parking_cost FLOAT,
    status VARCHAR(20) DEFAULT 'active',
    FOREIGN KEY (spot_id) REFERENCES parking_spot(id),
    FOREIGN KEY (user_id) REFERENCES user(id)

Relationships

- **One-to-Many:** User → Reservations (One user can have multiple parking sessions)
- **One-to-Many:** ParkingLot → ParkingSpots (One lot contains multiple spots)
- **One-to-Many:** ParkingSpot → Reservations (One spot can have multiple reservation records)
- **Cascade Delete:** When a parking lot is deleted, all associated spots are automatically removed

Design Rationale: The schema follows normalized database design principles to eliminate redundancy while maintaining data integrity. The foreign key relationships ensure referential integrity, and the status fields enable efficient querying of available spots and active reservations

5. API Design

The application implements RESTful routes following Flask conventions:

Authentication Routes

- **GET/POST /login -** User authentication with session management
- **GET/POST /register -** New user registration with validation
- **GET /logout -** Session termination and cleanup

Admin Routes

- **GET /admin/dashboard -** Administrative overview with statistics
- **GET /admin/manage_lots -** CRUD operations for parking lots
- **POST /admin/create_lot -** Create new parking lot with automatic spot generation
- **POST /admin/edit_lot -** Update lot details with spot count management
- **GET /admin/delete_lot/<id> -** Remove empty parking lots
- **GET /admin/view_spots/<lot_id> -** Detailed spot status and user information
- **GET /admin/reservations -** Comprehensive reservation history
- **GET /admin/charts -** Data visualization dashboard

User Routes

- **GET /user/dashboard -** User interface with available lots and active reservations
- **GET/POST /user/book_spot/<lot_id> -** Vehicle registration and spot booking
- **GET /user/release_spot/<reservation_id> -** Spot release with cost calculation
- **GET /user/history -** Personal parking history and statistics

Landing Route

- **GET / -** Application landing page with navigation

Implementation Details: Routes are organized by user roles with proper session validation. POST routes handle form submissions with server-side validation, while GET routes serve data with appropriate template rendering. The booking system implements first-available-spot allocation logic.

6. Architecture Features

Core Features Implemented

Admin dashboard with:

- Create/Edit/Delete lots
- Auto-create spots
- View user & spot details
- View charts (occupancy, revenue)

User dashboard with:

- Registration & login
- Auto-allot available spot
- Book/release spots
- Cost calculation & history

User Management:

- Secure registration and login system with session management
- Role-based access control (Admin/User)
- Automatic admin user creation during database initialization

Parking Lot Management:

- Complete CRUD operations for parking lots
- Automatic parking spot generation based on lot capacity
- Dynamic spot count adjustment with validation for occupied spots

- Real-time availability tracking

Reservation System:

- First-available-spot allocation
- Vehicle registration integration for enhanced security
- Timestamp-based parking duration calculation
- Automatic cost computation using time difference × hourly rate formula
- Status tracking (active/completed) for reservation lifecycle

Administrative Analytics:

- Interactive Chart.js visualizations showing occupancy distribution
- Revenue potential analysis across all parking lots
- Comprehensive reservation logs with user and vehicle details
- Real-time dashboard with key performance indicators

Additional Features

- **Enhanced User Experience:** Vehicle number collection during booking for realistic parking management
- **Responsive Design:** Mobile-first approach using Bootstrap for cross-device compatibility
- **Professional UI:** Clean, modern interface with hover effects and smooth transitions

Technical Architecture: The application follows the Model-View-Controller (MVC) pattern with clear separation of concerns. Database models handle data persistence, Jinja2 templates manage presentation logic, and Flask routes control application flow and business logic.

Folder Structure

```
City_Park_24f1002359/
├── app.py       # Main application with route definitions
├── models.py   # Database models and relationships
├── database.py  # Database initialization logic
├── static/
│   └── style.css     # Custom styling
├── templates/     # Jinja2 HTML templates
│   ├── login.html
│   ├── register.html
│   └── …..
├── README.md
└── Report.pdf
```

7. Video Demo Link

🎬 City Park Demo.mkv