Vehicle Parking App – Project Documentation

Author

- **Name:** Sachin Maurya
- **Roll Number:** 24f2000305
- **Email:** 24f2000305@gmail.com
- **Program:** IITM Online BS Degree Program, Indian Institute of Technology Madras, Chennai, India 600036

Description

This project implements a web-based Vehicle Parking App using Flask, enabling two roles: Admin and User. Admins can manage parking lots and monitor usage statistics, while Users can view available spots and make reservations. The system persists data in SQLite and provides a RESTful API for all core operations.

Technologies Used

- **Flask:** Python microframework for request handling and routing.
- **Jinja2, HTML, CSS, Bootstrap:** Frontend templating and responsive UI design.
- **SQLite:** Lightweight relational database for storing users, parking lots, spots, and reservations.
- **Python 3.x:** Core application language.
- **Flask-SQLAlchemy:** ORM for database modeling and queries.
- **Flask-Migrate:** (if used) for database schema migrations.

DB Schema Design

User

- `id` (Integer, PK)
- `username` (String, unique, not null)
- `password_hash` (String, not null)
- `role` (String, default 'user')

ParkingLot

- `id` (Integer, PK)
- `name` (String, not null)
- `address` (String)
- `price` (Float, not null)
- `max_spots` (Integer, not null)

ParkingSpot

- `id` (Integer, PK)
- `lot_id` (Integer, FK → ParkingLot.id, not null)
- `status` (String(1), default 'A')
- 'A' = Available
- 'O' = Occupied

Reservation

- `id` (Integer, PK)
- `spot_id` (Integer, FK → ParkingSpot.id, not null)
- `user_id` (Integer, FK → User.id, not null)
- `start_time` (DateTime, default current UTC)
- `end_time` (DateTime)
- `cost` (Float)

Constraints: Foreign keys enforce referential integrity. Unique constraint on username.

API Design

The application exposes the following RESTful endpoints:

- **POST /api/auth/register:** Create a new user account.
- **POST /api/auth/login:** Authenticate and obtain session token.
- **GET /api/lots:** List all parking lots with availability.
- **POST /api/lots:** (Admin) Create a new parking lot.
- **PUT /api/lots/\<lot_id>:** (Admin) Update lot details.
- **DELETE /api/lots/\<lot_id>:** (Admin) Remove a parking lot.
- **GET /api/spots:** Query all spots or filter by lot/status.
- **POST /api/reservations:** Reserve a specific spot.
- **GET /api/reservations:** (Admin/User) View reservations; admin sees all, user sees own.
- **PUT /api/reservations/\<res_id>:** Cancel or update reservation details.

Authentication uses session cookies or token headers. All admin routes require `role='admin'`.

Architecture and Features

- **Project Structure:**

```
project_root/
├─ app.py            # Application entrypoint, route definitions
├─ config.py         # Configuration (DB URI, secret key)
├─ models.py         # SQLAlchemy models and schema setup
├─ controllers/      # Request handlers and business logic
├─ templates/        # Jinja2 HTML templates
```

```
├─ static/           # CSS, JavaScript, images
└─ requirements.txt  # Python dependencies
```

- **Key Features:**
- **Admin Dashboard:** Overview of lots, occupancy rates, and user management.
- **Spot Monitoring:** Real-time display of available and occupied spots.
- **Reservation Flow:** Users can reserve, view, and cancel bookings.
- **Cost Calculation:** Automatic cost based on duration and lot pricing.
- **Responsive UI:** Mobile-friendly design using Bootstrap.

Video

The project demonstration video (≈3 minutes) is available here:\ https://drive.google.com/file/d/1kOjcsMi7h7s3wqVWDbfrbJ8aSAWMWbJJ/view?usp=drivesdk

---