AUTHOR

Name : Nihaal Ahmed R

Roll no: 22f2000914

Email: 22f2000914@ds.study.iitm.ac.in

About Me: Hello, I am Nihaal, I am 20 years old and I love to code!

Description

The Vehicle Parking Application is a web-based platform which aims to solve the problem of real-time vehicle parking management by providing a role-based system for both users and administrators. It allows users to reserve parking spaces, track their usage history, and calculate costs based on usage. Administrators can manage parking lots and spots, oversee user activity, and analyze revenue and usage data.

The system is developed using Flask for the backend, Vue.js for the frontend, Redis for caching, and a SQLite database.

System Architecture

The backend is implemented in Python using the Flask framework. SQLAlchemy ORM is used for database modeling, with Alembic for schema migrations.

Key components:

Flask-RESTful API for client-server interaction.

Flask-Security for secure authentication and session management.

Flask-SQLAlchemy for object-relational mapping.

Redis integrated with Flask-Caching to optimize performance for frequently accessed endpoints.

Frontend

The frontend is developed using Vue.js (Version 2 via CDN) and Vue Router for navigation. Bootstrap is used for responsive and consistent styling.

Functional Overview

The Vehicle Parking Application supports two primary roles: Users and Administrators. Registered users can log into the application and are able to view available parking lots and automatically reserve the first available spot. Once a spot is reserved, users can mark it as occupied, and upon leaving, release it—both actions are timestamped to calculate the total duration and corresponding cost based on lot-specific pricing. Users can view a complete history of their reservations, including entry and exit times, parking costs, and license plate information.

Administrators, who are predefined and cannot register themselves, have access to a dedicated dashboard from which they can create, update, and delete parking lots. When a new lot is created, spots are automatically generated based on the defined capacity. Admins can view real-time data on spot availability, user details, and historical reservation records. In addition, the system presents analytical summaries such as parking usage trends and revenue insights using integrated charting tools. To enhance performance, commonly accessed endpoints like available lots and spots are cached using Redis, ensuring quick response times and reduced server load.

Database Design

The application uses SQLAlchemy ORM for object-relational mapping and Alembic for schema migrations. The relational schema includes six main tables, outlined below:

User

The User model extends UserMixin from Flask-Security to enable authentication and role management. Each user has a unique username and email, a password hash, a creation timestamp, and a one-to-many relationship with the Reservation table. Users are associated with roles through the UsersRoles association table.

Role and UsersRoles

The Role model defines user roles such as "user" and "admin". The UsersRoles table implements a many-to-many relationship between User and Role using foreign keys. This structure allows each user to be assigned multiple roles and vice versa.

ParkingLot

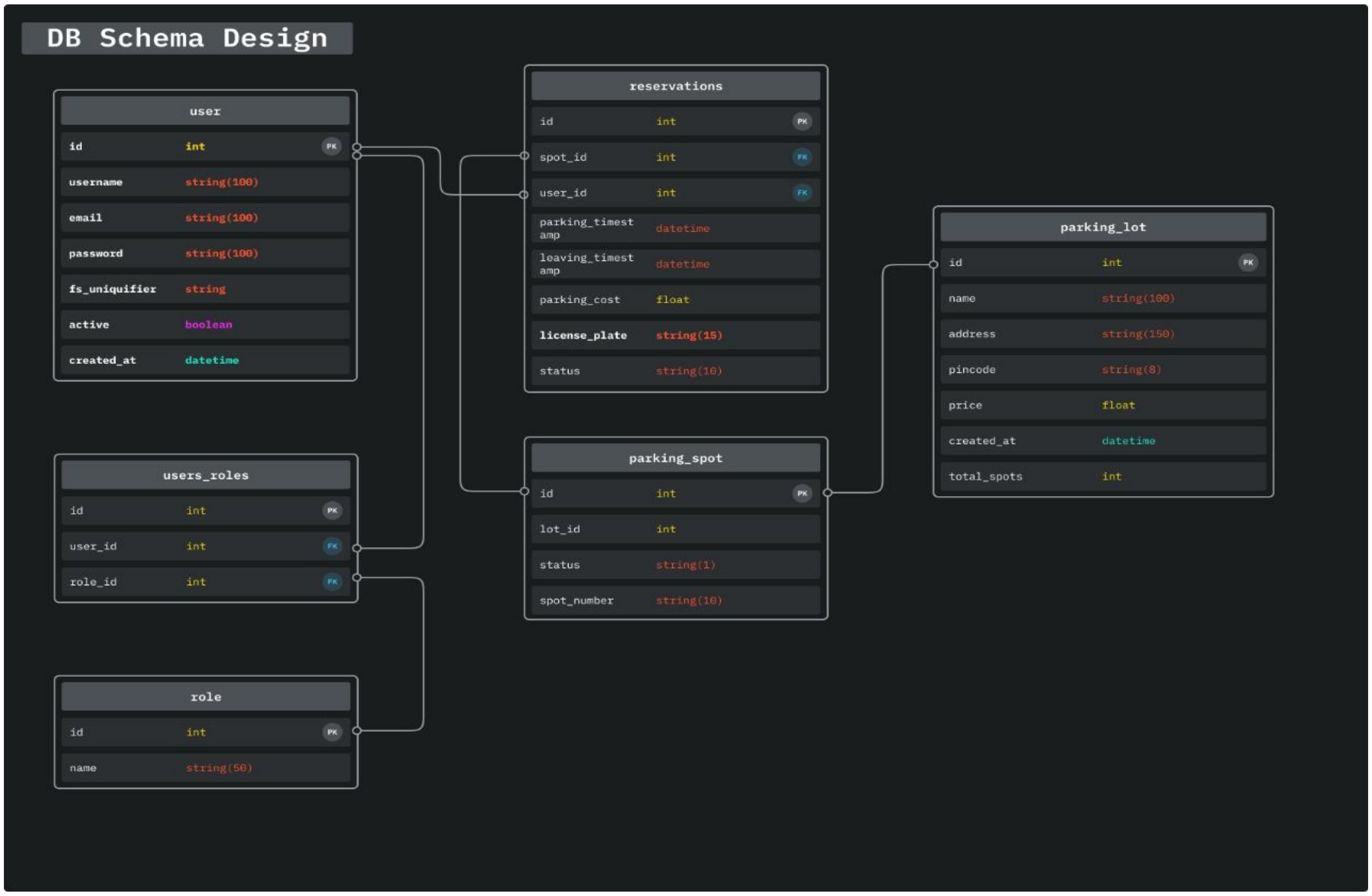
Each ParkingLot represents a physical parking facility with fields including name, address, pincode, hourly price, and total spot capacity. A one-to-many relationship exists between ParkingLot and ParkingSpot, enabling cascade deletion to maintain referential integrity.

ParkingSpot

A ParkingSpot is linked to a specific lot and includes a status field indicating whether it is available or occupied. Each spot has a spot number and can be reserved multiple times through its one-to-many relationship with the Reservation table.

Reservation

The Reservation table stores data related to each user's parking activity. It includes foreign keys to both User and ParkingSpot, timestamps for when a vehicle enters and exits, the total cost, license plate information, and a status field (e.g., active, completed). Cost is dynamically calculated based on time duration and lot pricing.



user		
id	int	PK
username	string(100)	
email	string(100)	
password	string(100)	
fs_uniquifier	string	
active	boolean	
created_at	datetime	

reservations		
id	int	PK
spot_id	int	FK
user_id	int	FK
parking_timestamp	datetime	
leaving_timestamp	datetime	
parking_cost	float	
license_plate	string(15)	
status	string(10)	

parking_lot		
id	int	PK
name	string(100)	
address	string(150)	
pincode	string(8)	
price	float	
created_at	datetime	
total_spots	int	

users_roles		
id	int	PK
user_id	int	FK
role_id	int	FK

parking_spot		
id	int	PK
lot_id	int	
status	string(1)	
spot_number	string(10)	

role		
id	int	PK
name	string(50)	

Application Architecture

The system follows a modular architecture comprising:

A RESTful API backend developed in Flask.

A client-side rendered interface using Vue.js.

Role-based routing and access controls enforced by Flask-Security.

Caching layers handled via Redis to reduce load on database for frequent queries.

Analytic charts powered by Chart.js, embedded in both admin and user dashboards.

AI Usage Declaration

Estimated percentage of AI involvement: approximately 10–15%.

Debugging and styling purposes.

Video Demo Link

https://drive.google.com/file/d/1ZGVDB3dKeh8YEpxzEa8-hvCjJc1mxcOZ/view?usp=sharing