PROJECT REPORT

Author

Mili Parashar
23f2004291
23f2004291@ds.study.iitm.ac.in

I am a passionate tech enthusiast currently pursuing a BS in Data Science at IIT Madras. My interests lie in exploring creativity, new ideas, and new perspectives, which enhances my problem–solving approach. Web Development and Data analysis are the areas where my energy thrives.

Description

The Vehicle Vault (Vehicle Parking App - V2) is a multi-user web application designed to manage 4-wheeler parking across various parking lots. It provides role-based access for admin and users, where an admin can create, edit, and monitor parking lots, spots available or occupied, users and revenue generated. On the other hand, users can reserve, release parking spots, and make payments for their car parking reservations. The application supports automated job scheduling (daily reminders, monthly reports), data caching, and report exports. It is built using Flask for the backend API, VueJS for the frontend, SQLite for the database, Redis for caching, and Celery for background jobs.

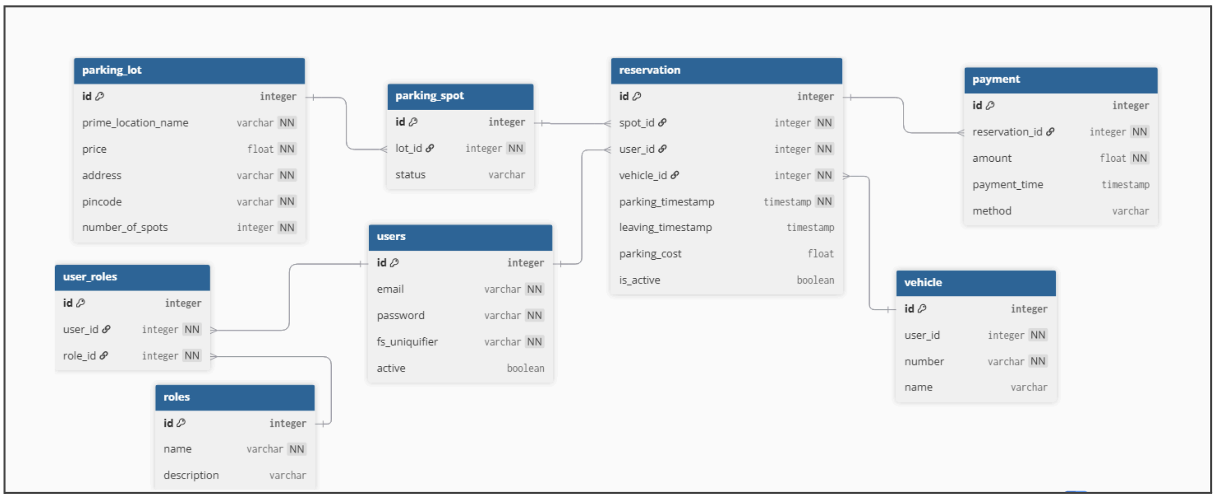
AI/LLM Used (Approx. 10-15%)
AI assistance was used primarily for debugging JavaScript functionality and implementing Celery tasks when manual attempts were unsuccessful.

Technologies used

Frameworks Used	**Purpose**
Flask	Primary backend framework for route handling and app configuration.
Flask-RESTful	Provides an organized structure for creating REST APIs using Flask.
Flask-SQLAlchemy	ORM for interacting with the SQLite database efficiently and cleanly.
Flask-Security	Manages authentication, authorization, password hashing, and role-based access.

Flask-Caching	Implements caching to improve performance and reduce redundant DB queries.
Flask-Excel	Enables import/export of data in Excel/CSV format via endpoints.
Matplotlib	Used for generating analytical charts in the background for reports.
Celery	Handles asynchronous and periodic tasks like email reports, reminders, and exports.
Redis	Serves as the broker for Celery tasks and a cache backend.
VueJS	Used to build a reactive, component-based, and dynamic frontend.
Vue Router	Manages navigation between user/admin views in the frontend.
SQLite	Lightweight, file-based database for easy integration with Flask.
pyexcel	Supports working with Excel files for export functionality.
ReportLab	Creates well-designed PDF reports for monthly activity summaries.
smtplib / email.mime	Sends HTML emails and PDF attachments to users as part of report functionality.
CSS	Styling framework for responsive and clean UI design.

DB Schema Design



Entity Relationship - Diagram

Database Structure and ER Explanation:

- **User Table:** Includes standard login credentials and is integrated with Flask-Security. Relationships to ‘Reservations’ and ‘Vehicles’ tables allow tracking a user’s activity and assets.
- **Role Table:** Defines roles to control access using Flask-Security. Ensures role-based behavior in the system.
- **UserRoles Table:** Implements a many-to-many relationship between users and roles via a linking table.
- **ParkingLot Table:** Captures the basic configuration and identity of each parking area. The number of spots determines how many ‘ParkingSpot Table’ entries are generated.
- **ParkingSpot Table:** Designed for dynamic allocation. The status field makes real-time availability tracking efficient. 'A' (Available) or 'O' (Occupied), indexed for fast lookups.
- **Reservation Table:** Tracks real-time and historical reservation data. Flexible design allows cost computation and report generation.
- **Payment Table:** Separates financial details from reservation logic. Helps in generating detailed payment records and summaries.
- **Vehicle Table:** Allows a user to register and manage multiple vehicles. Helpful in linking specific vehicle usage to reservations.

Relationships:

- One User can own multiple Vehicles.
- One Vehicle can have multiple Reservations.
- Each Reservation links a User, Vehicle, and a ParkingSpot.
- Each ParkingSpot belongs to a ParkingLot.
- Each Reservation has one Payment.

API Design

In the context of Application Programming Interface, a full-featured RESTful has been implemented for Vehicle Vault, the vehicle parking management system. Built using **Flask**, **Flask-RESTful**, **Flask-Security**, and **SQLAlchemy**, the system enables a secure and scalable communication between the frontend and backend in the form of requests and responses.

Authentication & Roles:

- **@auth_required('token'):** Token-based authentication using Flask-Security.
- **@roles_required('admin')** / **@roles_required('user')**: Role-based access control for **admin** and **user** roles, respectively.

Parking Lot Management: Only an admin can perform CRUD operations.

/api/parking_lots

- GET: List all parking lots for both admin and users

- POST: Admin adds a new parking lot and generates some parking spots for each lot, which initially are marked available.

/api/parking_lots/<lot_id>

- GET: Fetch details of a specific parking lot.
- PUT: Admin can update parking lot details and dynamically edit the quantity of parking spots
- DELETE: Admin can delete a parking lot (only if all spots are available).

Parking Spot Management

/api/parking_spots

- GET: List all spots or filter by lot_id

/api/parking_spots/<spot_id>

- GET: Admin can get spot details and reservation info if occupied by any user whereas users get the number of spots available for them to occupy.
- PUT: Only users can update parking spot status as available (when they confirm a reservation) and occupied (when they end a reservation)

Reservation System

/api/reservations

- GET: Admin can fetch a list of all active reservations made by the users.
- POST: Users can create a reservation out of the listed available parking spots and mark it occupied.

/api/reservations/<reservation_id>

- GET: Get a specific reservation
- PUT: When a user ends the reservation, payment has to be made.
- DELETE: When a reservation is deleted by an admin, spot is freed and no cost is to be paid by the user.

/api/active-reservations

- GET: Get active reservations for the current user

Vehicle Management: Only users can perform CRUD operations.

/api/vehicles

- GET: Get all vehicles for the logged-in user
- POST: A user can add a vehicle and its details

/api/vehicles/<vehicle_id>

- GET: Get vehicle information.
- PUT: Update it (only if owned by the current user)
- DELETE: Delete (only if not linked to an active reservation)

Data Visualization Charts

/api/parking-summary: Plots summary graphs for the admin to visualize and analyze data.

- The amount of revenue generated by each parking lot shows the popularity of the parking lots.
- Real-time status of occupancy (available vs. occupied spots) per parking lot and even track the number of spots each lot has.

/api/user-summary: Plots personalized summary graphs for each user to visualize and analyze data of their activity.

- Total Spend Over Time (Line graph)
- Reservations by Weekday (Bar chart)
- Vehicle Usage (Pie chart showing frequency of each vehicle used)

Architecture and Features

Vehicle Vault, the Vehicle Parking App, follows a **modular MVC-inspired architecture** tailored for a **Flask-VueJS full-stack** implementation. It organized into the following key components:

- **Model:** SQLAlchemy models define entities like User, Admin, Lot, Spot, Reservation, and Payment. The database is handled by SQLite with programmatic schema initialization with no manual setup. Asynchronous jobs such as reports, reminders and background exports are taken care by Redis and Celery. Additionally, Redis caches key API data for performance, with controlled expiration.
- **View:** The UI is built using Vue, Vue Router, and Vuex. All three of them were tailored together to make a beautiful, responsive user interface.
    - **Vue** is the Javascript framework that enables two-way data binding, controls what appears on screen and reacts to user input or data changes.
    - **Vue Router** is the official routing library for Vue.js. It maps URL paths (/dashboard, /login, etc.) to Vue components and supports nested routes, dynamic routes, and navigation guards.
    - **VueX** is the state management library for Vue. It manages shared data (state) across components (like user info, booking status, etc.) and ensures predictable state updates via mutations and actions.
- **Controller:**
    - **Flask-RESTful APIs** are located in api/ or routes/ and they handle endpoint logic, CRUD operations, and validations.
    - **Caching:** Key API data (e.g., spot availability) cached using Redis with timeout controls.
    - **Async Controllers:** Daily reminders, monthly reports, and CSV exports handled via scheduled tasks

Video

Project Video Explanation Link: Click here or Mad 2 Project Video - Vehicle Vault.mp4