App Development- 2 Project Report

1. Student Details

Name: Shashwat Pandey

About Me: I am a dual degree student who is also pursuing BS in Data Science at IIT Madras BS Degree program with a deep interest in modern web application development and data-driven technologies. I enjoy building meaningful applications that combine learning, analytics, and user experience.

2. Project Details

Project Title: PAVE – Vehicle Parking Management System

Problem Statement: Managing parking spaces manually leads to confusion, delays, and poor user experience for both parking authorities and vehicle owners. Users often struggle to find available spots, track parking duration, or calculate costs accurately. Administrators face difficulties maintaining multiple lots, tracking occupancy, and generating reports.

This web-based app project addresses these gaps by providing a complete, automated, and real-time parking management system accessible through a web interface.

Approach: This web app was made using Flask as the backend framework with a modular structure and Vue.js was used to create frontend that helps admin and users to manage and book parking spaces within a location.

3. Technologies and Frameworks Used

Technology / Framework	Description
Vue 3	Core frontend framework (Composition API)
Vite	Used as Frontend CLI & Bundler
Custom CSS	Entire user interface styled using handcrafted CSS without external frameworks, focus on Glass morphism style.
Font Awesome Icons	Used to display clean UI icons for actions, menu items, and status indicators across dashboards.
Chart.js	Generates dynamic charts for analytics, including 30-day spending trends and per-lot revenue visualizations.
Flask	Core Backend Web Framework - Handles API, business logic, reservations, analytics, and backend processes.
SQLAlchemy ORM	Provides structured interaction with the SQLite database.
SQLite Database	Lightweight local database used to store users, lots, spots, and reservation details.
Token Authentication	Manages secure login and role-based access (admin/user) via tokens stored in localStorage.

Fetch API	Custom apiFetch Wrapper for handling API calls, headers, tokens, and JSON data exchange.
Email Service (CSV Export)	Backend service that sends a formatted CSV of the user's parking history to their email on request.

4. Database Schema / ER Diagram

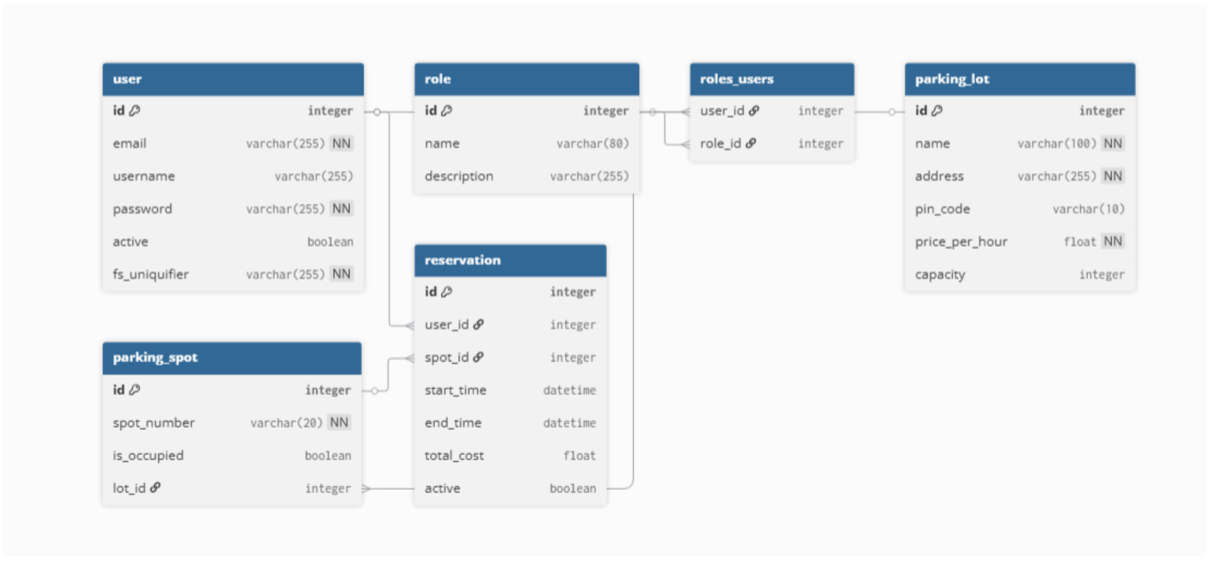
Tables:

1. **User**: stores *(id (PK), email, username, password, active, fs_uniquifier)*
2. **Role**: stores *(id (PK), name, description)*
3. **roles_users** (association): stores *(user_id (FK → user.id), role_id (FK → role.id))*
4. **ParkingLot**: stores *(id (PK), name, address, pin_code, price_per_hour, capacity)*
5. **ParkingSpot** : stores *(id (PK), spot_number, is_occupied, lot_id (FK → parking_lot.id))*
6. **Reservation**: stores *(id (PK), user_id (FK → user.id), spot_id (FK → parking_spot.id), start_time, end_time, total_cost, active)*

Relationships:

- User ↔ Role — Many-to-Many *via roles_users*
- User → Reservation — One-to-Many
- ParkingLot → ParkingSpot — One-to-Many
- ParkingSpot → Reservation — One-to-Many

ER Diagram:



(Generated using dbdiagram.io)

5. API Resource Endpoints

Endpoint Method Description

Authentication

- POST /login – login with email & password
- POST /register – create new user

Parking Lots

- GET /available-lots – used by user dashboard
- POST /parking-lot – add new lot (admin)
- POST /parking-lot/<id>/analytics – get 30-day analytics

Reservations

- POST /my-reservations – fetch user's active & past bookings
- POST /reserve – book a spot
- POST /release-spot – end session & calculate cost

Reports

- POST /export-csv – email CSV
- Client-side CSV download via Blob

User Summary

- POST /user-summary – lifetime spending, per-lot stats, timeseries data

YAML API Definition File:

Included separately in the submission ZIP as openapi.yaml.

6. Architecture and Features

Overall Architecture:

The system follows a modern client–server architecture. The frontend is a Vue 3 SPA structured using modular components and router-based navigation. All backend interactions use REST endpoints with JSON data exchange. Authentication relies on tokens stored in localStorage.

Chart.js handles analytics and modals manage booking/release workflows. Auto-refresh every 30 seconds keeps data live without requiring page reloads. The backend handles business logic, cost calculations, and summary aggregation. Together, the architecture ensures a reliable end-to-end parking management.

Key Features:

User Side

- Live active bookings
- Grouped and sortable active session cards
- "Find a Spot" section with availability bar

- Reservation modal with stepper (1–10 spots)
- End-session modal with estimated cost
- Parking history list & grouped cards
- CSV export (download + email)
- Summary tab with 30-day expense graph
- Per-lot spending list and analytics modal

Admin Side

- Add parking lots + auto spot generation
- View users & their histories
- Search across lots
- Summary analytics and revenue charts
- Per-lot 30-day breakdown