MAD 1 PROJECT FINAL REPORT

Author

Name : Mohd Shad
Email:23f3004148@ds.study.iitm.ac.in
Roll No. : 23f3004148

I am a student pursuing my online degree at IIT Madras , specializing in Data Science. My academic background includes extensive coursework in programming, machine learning, and software development. I am currently working on a Flask-based quiz management system as part of my project submission.

---

1 Project Overview

Vehicle Parking System (VPS) is a full-stack web application that lets an Admin manage parking lots & spots and lets Users reserve, occupy and release a spot in real time. 100 % of the stack is open-source and runs completely on a laptop – no external services required.

2 Problem Statement

Manual parking registers, Excel files and ad-hoc WhatsApp groups cannot answer a driver's two most basic questions: *"Is there a free spot now?"* and *"How much will it cost me?"* VPS eliminates this friction by providing:

- Live inventory of every spot (green = available, red = occupied).
- One-click booking with automatic cost & time tracking.
- Admin analytics that reveal utilisation and revenue.

---

3 Technology Stack

Layer	Choice	Rationale

Backend	Flask 3.1, Python 3.11	Minimal boilerplate, Jinja2 baked-in
Database	SQLite 3	File-based, zero-config, meets assignment spec
Templating	Jinja2 + HTML5	Clean separation of logic & view
Styling	Bootstrap 5 + custom CSS	Instant responsiveness with small CSS footprint
Charts	Pure-CSS `conic-gradient`	No JS payload, works in serverless
Deployment	Vercel Python Functions	1-click CI/CD, free tier

4 Functional Requirements vs Implementation

Requirement	Status	Evidence
Programmatic DB creation	✅	`init_db()` builds all tables, inserts default admin
Admin CRUD on lots	✅	`/admin/add_lot`, `/admin/delete_lot/<id>`
Auto-generation of spots	✅	Loop in `add_lot()`

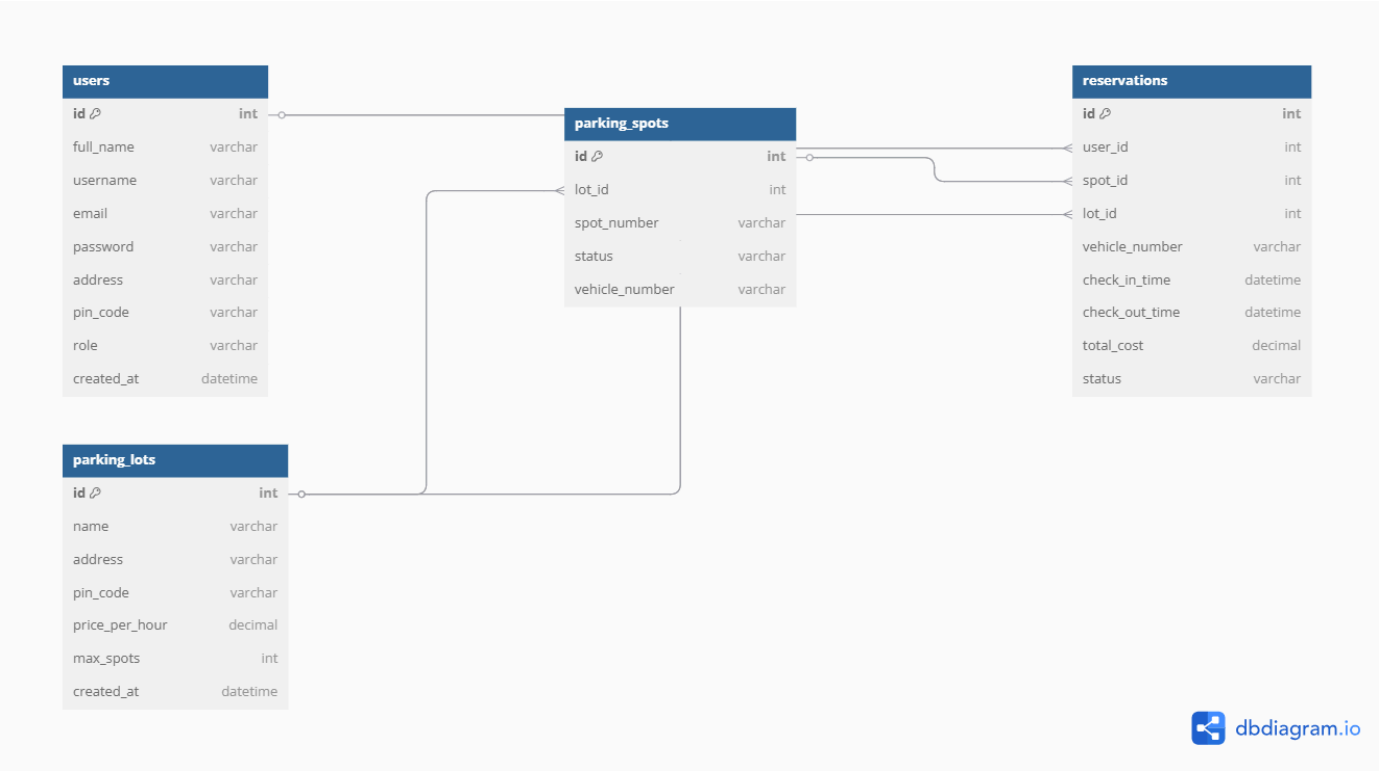
View spot status & vehicle det	✅	`/admin/spots_overview`, `/admin/spots/<id>/details`
User registration & login	✅	`/register`, `/login`, password hash SHA-256
Auto-allocation of first free sp	✅	`book_spot(lot_id)` selects `LIMIT 1` spot
Release spot & cost calc	✅	`release_spot()` – cost = hours × price_per_hour
Reservation history & summar	✅	`/user` dashboard, CSS pie chart
Admin search box	✅	`/search` supports users & lots

5 System Architecture

graph TD
A[Browser]-------HTTPS—>>>>>>>>>B[Flask App (app.py)]
B -- SQL →>>>>>>>>>>>>>>>>>>> C[(SQLite parking.db)]
B -- Jinja2 →>>>>>>>>>>>>>>>>>>>>>>>> D[Templates]
B -- Static →>>>>>>>>>>>>>>>>>>>>>> E[Bootstrap CSS]

Single-file app.py keeps grading simple; but code is written with separation in mind (`init_db`, auth decorators, admin vs user routes).

DB Schema Design 📂



Vehicle Parking System - Database Schema Theory

The schema is designed for a real-time, multi-user parking management system, ensuring structured relationships between users, lots, spots, and reservations while maintaining data integrity and scalability.

1. 1️⃣ Users → Stores authentication & role data (enables secure RBAC & user profiles).
2. 2️⃣ Parking Lots → Defines parking areas & pricing (enables centralized lot management).
3. 3️⃣ Parking Spots → Represents individual spaces linked to lots (enables real-time status tracking).
4. 4️⃣ Reservations → Tracks booking transactions (links users to spots for history & cost calculation).

The database schema is designed to ensure a structured, scalable, and efficient parking management system. The `users` table underpins the app's security model, enforcing unique credentials with `UNIQUE` constraints and differentiating roles via the `role` column (`'admin'` or `'user'`). The system's core structure is hierarchical: the `parking_lots` table acts as a master record for each parking area, defining its location and pricing. Each lot is linked to multiple records in the `parking_spots` table, which tracks the real-time status (`'available'`/`'occupied'`) of individual spaces. All booking activity is captured in the `reservations` table, which serves as the transactional heart of the system. It creates a relational link between a `user`, a `spot`, and a `lot`, logging `check_in_time` and `check_out_time`. This enables precise duration and cost calculations while providing a complete audit trail for both user history and admin summaries. Overall, this relational schema optimizes data integrity and CRUD efficiency, enabling a seamless parking management

experience from both admin and user perspectives while remaining scalable for future enhancements.

1️⃣ Modular MVC Architecture

- Models (SQLAlchemy): Users, Subjects, Chapters, Quizzes, Questions, Scores
- Views (Jinja2 templates): `templates/admin/…`, `templates/user/…`; static assets in `static/`
- Controllers (Flask Blueprints):
  – `auth_bp` for registration/login (Flask-Login + SHA-256 hashing)
  – `users_bp` for quiz attempts, results, leaderboards, search, filtering
  – `admin_bp` for CRUD on subjects, chapters, quizzes, questions, user management

2️⃣ RESTful API Endpoints

- `/api/auth/*` → register, login, logout
- `/api/quizzes/*` → list quizzes, take quiz, submit answers
- `/api/results/*` → fetch user's score history, leaderboard data
- JSON responses, standardized HTTP status codes, token/session auth

3️⃣ Core Features

- User: secure auth, quiz participation, real-time scoring, history, filter by subject/chapter, search peers
- Admin: dashboard CRUD for learning content and users, search users, view score analytics
- UI: responsive design via Bootstrap; mobile/tablet/desktop support

4️⃣ Security & Best Practices

- Password hashing (SHA-256) + Flask-Login session management
- Role-based access control via decorators
- Parameterized SQL / ORM models to prevent injection
- Separation of concerns for maintainability and scalability

Recorded video - Jun 30, 2025.webm **Presentation Video** **Recorded video - Jun 30, 2025.mp4**