# Vehicle Parking App – V2

Author: Parth Tiwari
Roll No: 24f3002941
Email: 24f3002941@ds.study.iitm.ac.in
Course: Modern Application Development II, IIT Madras

---

## Description

A full-stack vehicle parking web application for 4-wheeler parking that supports two roles: admin and users. The system manages parking lots, parking spots, and parked vehicles with automated spot allocation, background jobs, and analytic summaries. Features include secure login, user and admin dashboards, parking history, monthly reports, and CSV export of usage data. AI/LLM assistance was used mainly for design suggestions, debugging help, and minor refactoring; core code and architecture were implemented manually (estimated use ~15–20%).

---

## Key Features

- Role-based authentication: Separate flows for admin and users using JWT tokens; a single superuser admin is auto-created when the database is initialised.
- Admin dashboard: Create, edit, and delete parking lots; auto-generate spots based on max_spots; view per-lot total / available / occupied spots; view registered users; see current parked vehicles and per-spot status with vehicle + user details.
- User dashboard: Register/login, view available lots, park into the first available spot of a lot (user cannot pick spot manually), view currently parked vehicles, and unpark to compute cost using duration × hourly price.
- Parking history and analytics: Detailed history table with timestamps, duration, and cost per session; monthly summary page showing total sessions, total amount, most used lot, and recent activity.
- User-triggered CSV export (async job): From the user dashboard, a CSV export job can be triggered; the backend runs a Celery task that compiles full history and notifies the UI when the file is ready for download.
- Daily reminder job: A scheduled Celery Beat task checks which users have not parked in the last few days and sends them HTML email reminders to book a spot if needed.
- Monthly HTML email report: On a scheduled interval, a Celery task sends each user a monthly parking report via email summarising sessions, total amount spent, most used lot, and recent visits.
- Caching and performance: Redis cache is integrated for frequently accessed data (such as parking lot listings / dashboard summary), with expiry configured to keep data fresh while reducing database load.
- Responsive UI: All main pages (home, admin dashboard, user dashboard, history, export, monthly report) are styled with simple CSS and designed to work on desktop and smaller screens.

---

## Technologies Used

- Backend: Flask (Python), Celery for background jobs.
- Frontend: Vue 3 (Composition API), Vue Router for SPA-style navigation.
- Database: SQLite using SQLAlchemy / Flask-SQLAlchemy ORM.
- Caching & Message Broker: Redis (used both as cache and as Celery broker/result backend).

- Auth & Security: JWT-based token authentication, password hashing via Werkzeug.
- Environment & Config: python-dotenv for configuration management.

Purpose of stack: Flask + Vue cleanly separate API and UI concerns, SQLite provides simple local persistence suitable for the course, and Redis + Celery implement reliable async jobs for reminders, monthly reports, and CSV export, as recommended in the MAD 2 specification.

---

Database Schema Design

- User – stores user details and role.
    - Columns: id (PK), name, email (unique), password (hashed ), is_admin (boolean), timestamps.
- ParkingLot – represents each parking lot with pricing info.
    - Columns: id (PK), name, location, max_spots, price.
- Spot – individual spots within a lot.
    - Columns: id (PK), lot_id (FK → ParkingLot), spot_number, is_available (boolean).
- Vehicle – tracks each parking session.
    - Columns: id (PK), user_id (FK → User), lot_id (FK → ParkingLot), spot_id (FK → Spot), number_plate, entry_time, exit_time, derived fields such as duration and cost are computed on the fly.

Design reasoning: The schema models clear relationships between users, lots, spots, and individual parking sessions, keeping it normalised and aligned with the MAD 2 project tables (User, ParkingLot, ParkingSpot, ReserveParking).

---

Architecture and Project Structure

- backend_app/ – Flask and Celery backend
    - __init__.py – app factory, DB and cache initialisation, admin seeding.
    - models.py – SQLAlchemy models for User, ParkingLot, Spot, Vehicle.
    - controllers/ – blueprints for user and admin APIs (dashboard, park/unpark, history, summaries).
    - auth.py – registration, login, JWT token generation and validation.
    - tasks.py – Celery tasks for daily reminders, monthly reports, and CSV export.
    - email_utils.py – helper to send HTML emails (can be wired to SMTP).
- frontend/ – Vue single-page frontend
    - pages/ – Home, AdminDashboard, AdminCreateLot, AdminLotsManagement, UserDashboard, UserParkingHistory, UserExportHistory, MonthlyReport.
    - components/ – Navbar, LoginForm, RegisterForm, shared UI components.
    - services/api.js – Axios client for talking to Flask APIs.
- celery_worker.py – entry point for Celery worker using the Flask app factory.

Interactions happen via JSON APIs; Vue consumes these endpoints and renders dynamic pages, matching the MAD 2 requirement of a Flask API + JS frontend instead of server-rendered HTML.

---

**Backend Jobs (Batch Processing)**

- **Scheduled Daily Reminder:**
    - **Implemented using Celery Beat to run approximately every 24 hours.**
    - **For each non-admin user, finds the last parking entry; if the user never parked or has been inactive beyond a threshold (e.g., 3 days), an HTML reminder email is sent.**
- **Scheduled Monthly Activity Report:**
    - **A Celery task computes statistics for the previous month per user: total sessions, total amount, most used parking lot, and a table of recent visits.**
    - **Generates an HTML report and emails it to each active user.**
- **User-Triggered Async CSV Export:**
    - **From the user dashboard, an "Export" page triggers a Celery task that aggregates all Vehicle records for that user into a CSV with lot/spot/time/cost columns.**
    - **The frontend polls export status and enables a "Download CSV" button once the background task completes, satisfying the "batch job + alert once done" requirement.**

---

**Caching and Performance**

- **Redis cache is used for selected endpoints such as parking lot listings or dashboard summaries to improve response time under repeated access.**
- **Cache entries are configured with expiry so that updates to lots and spots eventually propagate while still reducing repeated database hits.**

---

**Features Implemented**

- **Two roles (single admin + multiple users) with role-based dashboards.**
- **Admin can create/edit/delete lots; spots auto-created from max_spots; lots can only be deleted when all spots are free.**
- **User parking flow with automatic first-available spot allocation, recorded timestamps, and cost calculation.**
- **User dashboard shows active parkings and history; admin dashboard shows lot/spot status, parked vehicle details, registered users, and summary stats.**
- **Background jobs: daily reminders, monthly HTML report via mail, and CSV export as async user-triggered batch job using Redis + Celery.**

    ---

-