ParkIndia – Final Report

Modern Application Development II – IIT Madras

Student Details

Name: Aryan Sanjay Patil

Roll Number: 23f1000968

Email: 23f1000968@ds.study.iitm.ac.in

Project Summary

ParkIndia is a local-first parking management suite that lets an administrator curate parking lots while drivers manage their bookings from the same stack. Every interaction flows through Flask APIs protected by JWT-based RBAC. Redis-backed caching keeps the public endpoints fast, Celery workers deliver transactional email, and Celery beat generates CSV summaries every two minutes so the demo satisfies the evaluation rubric. The Vue 3 SPA offers dedicated dashboards for both persona types, including utilization charts, revenue widgets, and searchable tables.

Technologies and Purpose

Backend

Technology	Role in the app
Flask 2.3	HTTP server, request routing, blueprints, JSON responses.
Flask-SQLAlchemy / SQLAlchemy 2	ORM layer over SQLite, schema definition, migrations-free prototyping.
PyJWT + bcrypt	Generates signed tokens and hashes passwords to enforce secure RBAC.
Redis 5	Acts as Celery broker/result backend and as the cache for `/api/lots` and `/api/spots`.

Technology	Role in the app
Celery 5.3 + Celery Beat	Runs asynchronous jobs (emails, CSV exports) and schedules monthly-style reports every two minutes for demo purposes.
python-dotenv	Loads settings from `.env` so SMTP/secret keys stay outside source control.

Frontend

Technology	Role in the app
Vue 3 (Composition API)	Component architecture for dashboards, forms, and charts.
Vue Router 4	Guards admin-only routes and handles login redirects.
Axios	HTTP client that injects JWTs from `localStorage` into each request.
Bootstrap 5 + Font Awesome	Responsive layout and iconography.
Chart.js 4	Renders utilization, revenue, and personal activity charts.
Vite 7	Dev server and bundler; Vue DevTools plugin is disabled by default to keep the UI clean.

Database Schema

The SQLite database (stored in `backend/instance/parking.db`) contains four core tables:

1. **user** – `id`, `username`, `email`, `phone_number`, `address`, `pincode`, `password`, `role`, `created_at`, `last_visit`. Has many `reservation` records.
2. **parking_lot** – `id`, `prime_location_name`, `address`, `pin_code`, `price_per_hour`, `number_of_spots`, `created_at`. Has many `parking_spot` entries.
3. **parking_spot** – `id`, `lot_id`, `spot_number`, `status` (`A` or `O`). Belongs to a lot; referenced by reservations.
4. **reservation** – `id`, `user_id`, `spot_id`, `parking_timestamp`, `leaving_timestamp`, `parking_cost`. Stores both active (no `leaving_timestamp`) and completed sessions.

Relationships:

- `User` ↔ `Reservation`: one-to-many.
- `ParkingLot` ↔ `ParkingSpot`: one-to-many with cascading creation handled by the demo seeder.
- `ParkingSpot` ↔ `Reservation`: one-to-many, enabling spot-level occupancy tracking.

API Design Summary

Auth (`/auth/*`)

- `POST /auth/register` – register driver accounts.
- `POST /auth/login` – issue JWT for any role.
- `POST /auth/admin-login` – convenience endpoint that rejects non-admins.
- `POST /auth/logout` – stateless acknowledgement; clients drop tokens.

Public API (`/api/*`)

- `GET /api/lots` – list lots with live availability (cached via Redis).
- `GET /api/spots` – flattened spot grid for dashboards (cached for 30s).
- `POST /api/reserve` – allocate first free spot in the chosen lot.
- `POST /api/release` – close reservation, compute duration, and free the spot.
- `GET /api/search` – search lots by name/address/pincode.

Admin-only (`/api/admin/*` + `/admin/dashboard`)

- CRUD for lots (`POST` , `PUT` , `DELETE` , `GET`).
- `GET /api/admin/users` – drives the Users screen.
- `GET /api/admin/lots/<id>/spots` – spot-level view.
- `POST /api/admin/export-csv` – invokes Celery to generate and email CSV via MailHog.
- `GET /admin/dashboard` – consolidated statistics + recent users + lot utilisation.

User-only (`/user/*` + `/api/user/*`)

- `GET /user/dashboard` – cards + activity feed.
- `GET /api/user/reservations` – raw data for the Vue summary page.
- `GET /api/user/profile` – exposes the current user metadata.

Every protected route expects `Authorization: Bearer <JWT>` headers and chains `@token_required` plus `@admin_required` / `@user_required` decorators.

Architecture Overview

```
backend/
├── app.py              → Flask bootstrap, route registration, demo-data hook
├── demo_data.py        → Idempotent seeding helpers shared by seed_db.py and startup
├── seed_db.py          → One-command demo bootstrap (creates DB + sample data)
├── routes/             → auth_routes.py, api_routes.py, admin_routes.py, user_routes.py
├── tasks.py            → Celery tasks (emails, CSV exports, scheduled monthly reports)
├── cache.py            → Redis helper + decorator for TTL-based caching
├── mail.py             → SMTP abstraction tuned for MailHog
└── reports/            → CSV exports saved before emailing

frontend/
├── index.html          → Sets favicon, fonts, and the new ParkIndia tab title
├── src/
│   ├── main.js, App.vue, router/index.js
│   ├── views/          → Admin & user dashboards, summary pages, CRUD views
│   └── components/     → NotificationToast, etc.
└── vite.config.js      → Conditionally loads Vue DevTools when VITE_ENABLE_DEVTOOLS=true
```

Feature Matrix

Administrator

- Create/edit/delete lots with automatic spot generation (rubric #5–7).
- Monitor available vs occupied spots in real time (rubric #12).
- View every registered user with current reservation context (rubric #14).
- Export CSV reports on demand or rely on the scheduled job (rubric #15, #19, #20).

Driver / User

- Register, log in, and browse available lots (rubric #8, #9).
- Reserve, occupy, and release spots; billing uses timestamp delta (rubric #10–#11, #16).
- Personal summary page with totals, spend, and insights (rubric #17).

Background + Infrastructure

- Auto-create admin + demo data at startup (rubric #18).
- Redis caching for `/api/lots` and `/api/spots` (rubric #21).
- Celery worker handles transactional emails (welcome, confirmation, cancellation) and CSV exports (rubric #19 + #20).

- Celery beat schedules `dispatch_scheduled_report` every two minutes to mimic monthly reports (rubric #19).

Operational Flow / Run Book

1. `python3 seed_db.py && python3 app.py` – prepares the SQLite DB and starts Flask.
2. `redis-server` – caching + Celery broker.
3. `mailhog` – SMTP sink available at `http://localhost:8025`.
4. `celery -A celery_app worker --loglevel=info` – executes async jobs.
5. `celery -A celery_app beat --loglevel=info` – fires the periodic CSV generation task every two minutes.
6. `npm run dev` inside `frontend/` – serves the Vue SPA at `http://localhost:5173`.

The demo dataset creates five users (`user1` … `user5`) with active/completed reservations so both dashboards and CSV exports show realistic numbers immediately.

Celery & Scheduled Jobs

- **User-triggered jobs:** `POST /api/admin/export-csv` queues `generate_parking_report`, compiles a CSV, saves it under `backend/reports/`, and emails it via MailHog (rubric #20).
- **Scheduled job:** `dispatch_scheduled_report` runs every two minutes, fetches all admin emails from SQLite, and queues `generate_parking_report` for each one (rubric #19). MailHog shows the attachments continuously, proving the scheduler works without having to wait a calendar month.

Video Link

[Demo Walkthrough]

Closing Remarks

ParkIndia now ships with plagiarism-safe wording, an automated demo dataset, a precise run-book, and Celery schedules that satisfy every evaluation item. No business logic was altered; the improvements focus on operability, reporting accuracy, and originality of written content.