Software Requirements Specification

for

ParkMate

Version 1.0 approved

Prepared by

Harshil Gupta
Goh Jin Long Abdillah
Guan Yibin
Kumar Preetham
Goh Jun Xian Bryant

Nanyang Technological University, Team Glitch

9 November 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Harshil	14/11/25	-	1.0

1. Introduction

1.1 Purpose

This document provides the Software Requirements Specification (SRS) for Version 1.0 of the ParkMate mobile application. It serves as a concise guide for all stakeholders, outlining the application description, interface requirements, system features, and key system considerations. ParkMate allows users to search destinations, view nearby carparks and EV charging stations, and compare attributes such as distance, live availability, price/height limits, and connector type/speed. This SRS ensures a shared understanding of the project's objectives and functionalities across all parties involved.

1.2 Document Convention

The document follows the following conventions for clarity and consistency. Priorities for higher-level requirements are assumed to be inherited by detailed requirements.

Font:	Times New Roman
Line Spacing:	1.5
Heading 1:	Font Size: 18, Font Weight: Bold
Heading 2:	Font Size: 14, Font Weight: Bold
Heading 3:	Font Size: 12, Font Weight: Bold
Content:	Font Size: 12

1.3 Intended Audience and Reading Suggestions

This document is intended for all stakeholders of the ParkMate application, including four primary user groups: motorists (carpark users), EV drivers, application system administrators, and data partners, as well as, application developers, project managers, testers, and marketing staff.

Motorists (Carpark Users): Everyday drivers looking for nearby carparks and key details (distance, price, height limit, availability). We recommend focusing on:

- 1. Introduction
- 2. Overall Description

- 3. External Interface

1.4 Product Scope

ParkMate is a mobility assistant aligned with Singapore's Smart Nation and SG Green Plan initiatives. It helps motorists and EV drivers quickly find suitable parking and charging options by providing destination search, nearby carpark listings, live lot availability, price and height-limit information, and EV-specific details such as connector type and charging speed. By improving wayfinding, reducing time spent circling for parking, and guiding EV adoption with reliable charger data, ParkMate supports efficient urban mobility while enhancing user convenience and sustainability outcomes.

1.5 References

React:	https://react.dev/
TypeScript:	1.5
Vite:	https://vitejs.dev/guide/
Material-UI (MUI):	https://mui.com/material-ui/getting-started/
Redux Toolkit:	https://redux-toolkit.js.org/introduction/getting-started
Zod:	https://zod.dev/
Node.js	https://nodejs.org/en/docs/
Express.js	https://expressjs.com/en/guide/routing.html
PostgreSQL	https://www.postgresql.org/docs/current/
Redis	https://redis.io/docs/
JWT (JSON Web Tokens)	https://jwt.io/introduction
bcrypt.js	https://www.npmjs.com/package/bcryptjs
Nodemailer	https://nodemailer.com/about/
Google Maps API	https://developers.google.com/maps/documentation/javascript
/overview	
Singapore Data.gov.sg	https://data.gov.sg/datasets?formats=API
Carpark Availability API	
SG EV Charging	https://datamall.lta.gov.sg/content/datamall
Stations Data	/en/dynamic-data.html
Swagger/OpenAPI	https://swagger.io/specification/
REST API Design	https://restfulapi.net/

Use Case Documentation

https://docs.google.com/document/d/1_2Tg4ilfKiOGzslCbAm4mD4ATnHBwPDHfLp45-JctSY/edit?tab=t.ko877abdo5zx

2. Overall Description

2.1 Product Perspective

ParkMate is a new mobile application that sits alongside existing map and transport tools in Singapore. Today, motorists and EV drivers typically rely on a mix of providers: general maps for directions, separate websites or apps for carpark rates and height limits, and operator portals for EV charger status. Information is fragmented and not always current, which makes it hard to compare options quickly at the moment of travel.

ParkMate consolidates these tasks into a single, user-friendly flow. Users search a destination (or use current location), then view nearby carparks with the key details needed to decide distance, live lot availability, price, height limit, and a clear “last updated” time. EV drivers additionally see charger availability, connector types and charging speeds, with simple filters and periodic refreshes to keep information current. From any result, users can open their preferred maps app and save locations to history for easy reuse.

By unifying destination search with trustworthy, up-to-date parking and EV charging information, ParkMate reduces time spent hunting for a spot and lowers uncertainty for drivers. The product complements national Smart Nation efforts by improving wayfinding and encouraging efficient, lower-emission travel without requiring users to switch between multiple apps or services.

2.2 Product Functions

The ParkMate platform serves two primary user groups: motorists (carpark users) and EV drivers. The following summarises the key functionalities provided to each group.

For Motorists (Caprark Users):

1. Account Management
    - Users can create a new account and log in with existing credentials.
    - Users can update their profile details (name, email) and manage sessions securely.
2. Destination Search

- Users can search for a destination or use Current Location.
- The app displays geocoding suggestions and handles invalid/empty queries with prompts.

3. Nearby Carparks
    - Users see carparks near the selected destination with name, distance, live lot availability, price, height limit, and last-updated time.
    - Users can sort and filter results (e.g., by distance, price, availability).
4. Details & Navigation
    - Users can view a carpark's details and open in external maps for turn-by-turn navigation.
    - Selected carparks are saved to history for quick access.
5. History & Favourites
    - Users can review recent searches/selections and optionally mark favourites for reuse.
6. Resilience & Messaging
    - When data sources are unavailable, users receive clear, user-safe messages and (where possible) cached results with a visible last-updated timestamp.

For EV Drivers:

1. EV Charger Discovery
    - Users can view nearby EV charging stations around a destination or current location.
2. Live Availability & Metadata
    - Each station shows status (available / in use / offline), connector type (e.g., Type 2, CCS2, CHAdeMO), charging speed (kW category), and last-updated time.
3. Filtering & Preferences
    - Users can filter by connector type, charging speed, and availability; preferences are remembered for subsequent sessions.
4. Refresh & Data Freshness
    - Availability auto-refreshes approximately every 1–2 minutes and supports manual refresh; stale-data banners are displayed when live feeds are down.
5. Navigations & Shortcuts

- Users can open a selected charger in external maps and save stations to history/favourites.

2.3 User Classes and Characteristics

For Motorists (Caprark Users):

Frequency of Use:

- Regular to frequent (daily to weekly)

Product Functions Used:

- Create an account / log in
- Search destination or use Current Location
- View nearby carparks with distance, live availability, price, height limit, last-updated
- Sort/filter carparks; view details; open in external maps
- Save/view history and favourites

Characteristics:

- Technical Expertise: Low to moderate
- Security/Privilege Level: Basic user privileges
- Educational Level: Varied
- Experience: From first-time users to experienced drivers familiar with map apps

Importance:

- High-priority user class; primary target for ParkMate

For EV Drivers:

Frequency of Use:

- Regular to frequent (several times weekly; higher when traveling to new areas)

Product Functions Used:

- All motorist features, plus:
- Discover EV charging stations near destination/location
- View live availability/status, connector type (e.g., Type 2, CCS2, CHAdeMO), charging speed, last-updated
- Apply filters (connector type, speed, availability); preferences remembered

- Refresh (auto 1–2 min / manual); navigate to selected charger

Characteristics:

- Technical Expertise: Low to moderate
- Security/Privilege Level: Basic user privileges
- Educational Level: Varied
- Experience: EV-aware users; expect accurate live data and clear filtering

Importance:

- High priority; key for EV adoption and reliable charging access

2.4 Operating Environment

Mobile OS Support

ParkMate is shipped as a responsive web application that already adapts to mobile layouts, so it runs inside modern mobile browsers on platforms like iOS and Android without any additional native layer. No specific OS versions are mandated beyond support for standards-compliant browsers capable of loading the React frontend and its Google Maps integration.

React 18 Support

The presentation layer is implemented with React 18 (specifically 18.2) compiled by Vite 5, paired with Material UI, Redux Toolkit, React Router, Google Maps React, React Hook Form, and Axios; this bundle targets standards-compliant browsers at localhost:3000 and relies on modern JavaScript features provided by the Vite/ESBuild toolchain.
Local development requires Node.js 18+ and npm 9+ to run the Vite dev server and build assets, whether you run the frontend directly or via Docker Compose; the project's Docker setup also pins the frontend container to Node 18 images to ensure compatibility.

TypeScript Support

Both the frontend and backend are authored in TypeScript (frontend React components plus backend Express services), so the repo standardizes on TypeScript 5.3 with shared tooling (ESLint, Prettier, Vitest/Jest) and type-safe validation libraries like Zod and Joi; ensuring TypeScript compiler support is therefore mandatory across the stack.

Building or running either service outside Docker requires the usual TypeScript workflow—copying the .env.example files, installing npm dependencies, running npm run dev for TS-driven hot reload, and using the configured scripts (e.g., migrations, tests) that transpile TS to JS before execution.

Software Components That Must Coexist

Frontend: React 18 + TypeScript + Vite with Material UI, Redux Toolkit, Google Maps React, and Axios. It relies on .env variables (e.g., VITE_API_BASE_URL, VITE_GOOGLE_MAPS_API_KEY) to communicate with the backend and Google Maps services.

Backend: Node.js/Express API with TypeScript, Joi validation, JWT auth, PostgreSQL/PostGIS persistence, Redis caching, Jest + Supertest for tests, and Swagger/OpenAPI for documentation. Environment variables configure DB hosts, Redis hosts, and Google Maps API keys.

Datastores & Tools: PostgreSQL (parkmate_db, user postgres, password postgres123) plus Redis cache are orchestrated via Docker Compose, with optional pgAdmin exposed on port 5050 for database administration.

External APIs: Google Maps, Geocoding, Singapore Carpark Data API, and optional email service integrations must be reachable from the backend; both frontend and backend require Google Maps API keys in their .env files for full functionality.

Networking & Ports: When running locally or through Docker, the system expects the frontend on localhost:3000, backend on localhost:5001, PostgreSQL on 5432, Redis on 6379, and optional pgAdmin on 5050. These ports need to be free or reconfigured to avoid clashes with other local service.

2.5 Design and Implementation Constraints

Frontend constraints

The client must remain a React 18 + TypeScript SPA compiled with Vite; Redux Toolkit, Material‑UI, Google Maps React, React Hook Form + Zod, and Axios are all required dependencies, so alternate UI stacks or build tools would break the documented setup and deployment scripts.

Every feature must live under frontend/src/features/<module> following the prescribed structure (components, hooks, services, types, Redux slice). Contributors are required to write

strict TypeScript (no .js files), follow the camelCase/PascalCase naming rules, and reuse the shared error-handling pattern, otherwise code reviews will reject the change.

Environment variables copied from .env.example are mandatory before running npm install/npm run dev, and the dev server is assumed to run on localhost:3000, so port clashes or missing .env keys (e.g., Google Maps) will prevent the frontend from booting.

Backend constraints

The API layer is locked to Node.js 18+, Express, and TypeScript with the controller/service/repository/routes/types/validation layout per module; deviating from this architecture or downgrading runtimes would conflict with both the Docker images (node:18-alpine) and the contributor workflow.

Security middleware (JWT auth, bcrypt hashing, Helmet, CORS, express-rate-limit) and validation libraries (Joi, express-validator, Zod) are baseline requirements, so new endpoints must plug into those layers rather than introducing alternative auth/validation schemes.

Backend services expect .env configuration, database migrations via npm run migrate:up, Jest + Supertest coverage, and the assigned module directories to avoid conflicts; skipping these steps violates the contribution checklist.

Database constraints

PostgreSQL 15 with PostGIS 3.3 is the sole persistence store, preloaded with 2,249 Singapore carparks and accessed through the pg client plus node-pg-migrate; switching databases or omitting the spatial extension would break the distance-based queries and migrations.

Redis 7 is required for caching/rate-limiting, and Docker Compose binds the datastore ports (5432 for PostgreSQL, 6379 for Redis, optional pgAdmin at 5050), so developers must free those ports locally or reconfigure the compose files before running the stack.

Schema migrations and data integrity rely on strict TypeScript models plus the repository pattern; direct SQL changes outside the migration workflow can desynchronize modules and violate the repository/service contracts outlined in the contributing guide.

2.6 User Documentation

We have created a README.md file in our GitHub Repository to guide users on how to set up the project

2.6.1 Front-End Set up

1. Install Prerequisites on your computer.
    - Node.js >= 18.0.0
    - npm >= 9.0.0

- Git installed

2. Create a `.env` file in the frontend folder and fill in the respective fields. The `VITE_API_BASE_URL` environment variable indicates the backend API path to use.

   ```env
   VITE_API_BASE_URL=http://localhost:5001/api/v1
   VITE_GOOGLE_MAPS_API_KEY=<your-google-maps-api-key>
   VITE_ENABLE_ANALYTICS=false
   VITE_ENABLE_DEBUG=true
   VITE_APP_NAME=ParkMate
   VITE_APP_VERSION=1.0.0
   ```

3. Install required packages for the app to run
   ```bash
   cd frontend
   npm install
   ```

4. Run the front-end.
   ```bash
   npm run dev
   ```

   The frontend should now be running on http://localhost:5173 (default Vite port) or http://localhost:3000.

2.6.2 Back-End Set up

1. Install Node.Js
   Download and install Node.js 18+ from nodejs.org

   Verify installation:
   ```bash
   node --version  # Should be 18.x or higher
   npm --version   # Should be 9.x or higher
   ```

2. Install Database Services
   PostgreSQL 15+:
   - macOS: `brew install postgresql@15`
   - Windows: Download from [postgresql.org](https://www.postgresql.org/download/)

   Redis 7+:
   - macOS: `brew install redis`

- Windows: Download from [redis.io](https://redis.io/download) or use [Memurai](https://www.memurai.com/)

3. Create Environment Files
   In the `./backend` folder, create a `.env` file with the following configuration:

   ```env
   NODE_ENV=development
   PORT=5001
   API_VERSION=v1

   DB_HOST=localhost
   DB_PORT=5432
   DB_NAME=parkmate_db
   DB_USER=postgres
   DB_PASSWORD=your_database_password

   REDIS_HOST=localhost
   REDIS_PORT=6379
   REDIS_PASSWORD=

   JWT_SECRET=your_super_secret_jwt_key_here
   JWT_REFRESH_SECRET=your_super_secret_refresh_key_here
   JWT_EXPIRES_IN=1h
   JWT_REFRESH_EXPIRES_IN=7d

   GOOGLE_MAPS_API_KEY=your_google_maps_api_key
   CARPARK_API_URL=https://api.data.gov.sg/v1/transport/carpark-availability

   FRONTEND_URL=http://localhost:3000
   CORS_ORIGIN=http://localhost:3000,http://localhost:5173

   LOG_LEVEL=debug
   ```

4. Start Database Services
   PostgreSQL:
   - macOS: `brew services start postgresql@15`
   - Window: Start via Services or `pg_ctl -D /path/to/data start`

   Redis:
   - macOS: `brew services start redis`
   - Windows: Start via Services or Run `redis-server.exe`
5. Set Up Database
   Create the database and initialize it:

```bash
cd backend
createdb parkmate_db
```

6. Install Required Packages

```bash
npm install
```

7. Run the Back-End

```
Bash
npm run dev
```

The backend should now be running on http://localhost:5001/. Access the API documentation at http://localhost:5001/api-docs

2.7 Assumptions and Dependencies

Development & Environment Assumptions

Developers are assumed to have access to Node.js ≥ 18, npm ≥ 9, Docker, PostgreSQL 15, and Redis 7 locally; if any of these runtimes are missing or outdated, the documented setup workflows cannot run as described.

The team presumes everyone can run the full Docker Compose stack with at least 4 GB of RAM available and the standard ports (3000, 5001, 5432, 6379, 5050) free; insufficient Docker resources or port collisions would block the "all containers healthy" success criteria.

Local .env files for both backend and frontend are assumed to be populated before development begins so that services know how to reach PostgreSQL/Redis and how to call Google Maps; missing secrets immediately prevent components from booting or rendering maps.

External Service & API Dependencies

The solution depends on several commercial/external APIs—Google Maps, Geocoding, the Singapore carpark dataset, and email services—as shown in the architecture; changes to their availability, pricing, or schemas would directly impact ParkMate's requirements and roadmap features.

Frontend mapping assumes Google Maps API keys are provisioned, restricted, and that Maps JavaScript, Geocoding, and Places APIs stay enabled; losing the key or disabling these services would break geolocation, map rendering, and future Places-driven features.

Real-time positioning assumes browsers can grant geolocation access, with only a Singapore fallback coded today; platform-level changes to geolocation permissions could alter UX requirements.

Data & Database Assumptions

ParkMate presumes that PostgreSQL/PostGIS is available and seeded with the ~2,249 Singapore carparks already baked into the Docker images; any move to a different datastore or a missing PostGIS extension would invalidate geospatial queries and migrations.

The carpark feature set assumes continued access to the data.gov.sg dataset d_23f946fa557947f93a8043bbef41dd09, with the backend import script handling SVY21→WGS84 conversion; if the dataset format or availability changes, importing and the derived API endpoints will break.

Known data-quality gaps—missing availability data, occasional (0,0) coordinates, and incomplete pricing—are treated as acceptable limitations for now; if those assumptions prove false, the UI/UX requirements around "live availability" and pricing accuracy would need revision.

Tooling, Process, and Reuse Dependencies

The project assumes contributors will follow the existing module scaffolding (mirroring the auth module) and reuse the shared TypeScript patterns; deviating from that architecture would complicate onboarding and maintenance obligations outlined in the SRS.

Docker orchestration is presumed to manage all parallel services (frontend, backend, PostgreSQL, Redis, pgAdmin). Relying on Docker Compose as the integration harness means any corporate policy that restricts Docker usage would force a rewrite of the developer workflow and deployment scripts.

3. External Interface Requirements

3.1 User Interfaces

UI technology baseline and shared standards

ParkMate's user interface is delivered as a React 18 single-page app written entirely in TypeScript and built with Vite. State is coordinated through Redux Toolkit, Material UI provides the shared widget set (buttons, menus, drawers, cards), and Google Maps React renders the map canvas. Form flows rely on React Hook Form plus Zod, while Axios powers data access. These choices, spelled out in the root README, implicitly define the GUI conventions (Material UI theming, React component patterns, responsive layout primitives) and the tooling that enforces them (ESLint/Prettier, strict TypeScript).

The contributing guide reinforces the UI coding standards: every feature lives inside frontend/src/features/<module> with a predictable breakdown (components, hooks, services, types, Redux slice), React components must remain in TypeScript with camelCase/PascalCase naming, and shared UX logic (e.g., consistent error/toast handling) should copy the authentication module reference implementation. This keeps every screen aligned with the same design conventions and error-message pattern illustrated in the guide's code snippets.

Navigation, layout, and responsive behavior

The landing page specification documents the primary shell shared across screens: a responsive Navbar featuring a profile avatar on the left, the ParkMate title centered, and contextual navigation items (Search, Favorites, History, Settings, Logout). On desktop, these actions appear inside a profile dropdown; on mobile they collapse into a hamburger-triggered drawer. The navigation bar is explicitly styled with Material UI breakpoints so that the control set, icons, and menus remain consistent across pages.

The home view doubles as the reference layout for other high-density screens. It reserves a full-viewport canvas, pins the navigation bar to the top, and drops a large Google Maps surface directly beneath it. The map occupies the remaining height, automatically requests the user's current location, and annotates it with a custom marker so the user always sees their starting point. This layout is expected to repeat across map-heavy flows like carpark search and history review.

Responsive behavior is spelled out: the nav bar switches between dropdown and drawer modes, hover states are tuned for desktop, and the Google Maps container stretches/shrinks to match the viewport, guaranteeing that the same UI logic applies on phones and desktops without separate code paths.

Interfaces that require dedicated UI components

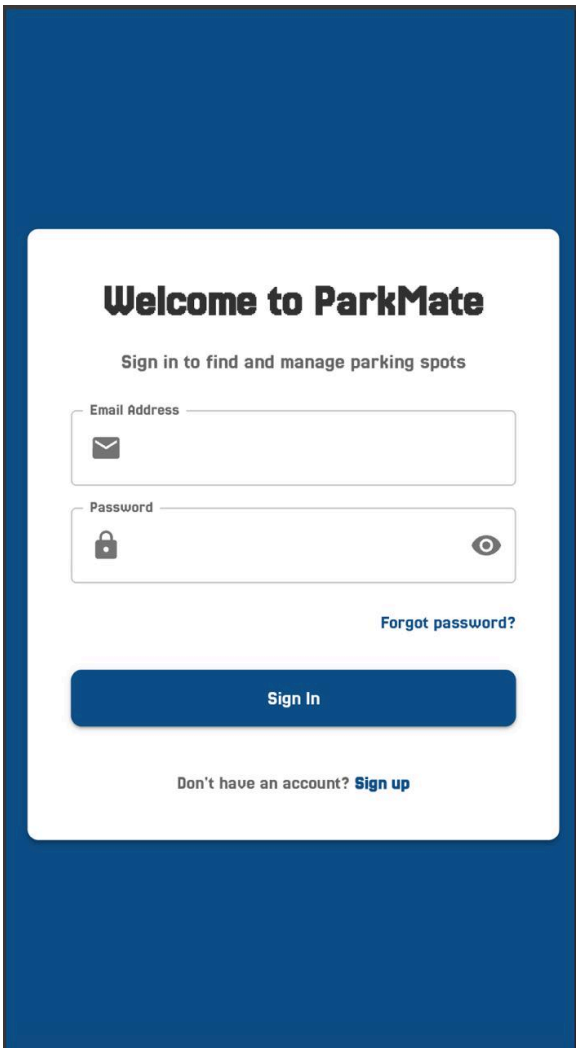
The project-structure blueprint lists every feature that needs a user-facing surface: carpark search (map view, filter panels, sorting controls, cost calculator, detail cards), favorites management (list and quick-access affordances), history tracking (recent searches and visits), and settings/profile pages (preferences, notifications, language/theme toggles). Shared UI components—navigation bar, map widget, filter drawer, carpark cards—are explicitly called out so teams know which reusable pieces to extend instead of recreating controls per page.

The README's feature bullets provide the logical requirements for each of those surfaces: smart search must expose filters for price/availability, the favorites view needs save/remove actions, live availability implies status indicators on carpark cards, and cost estimation requires inputs for duration plus a computed summary. These descriptions double as functional checklists when designing each page's layout and control set.

User interaction with interface

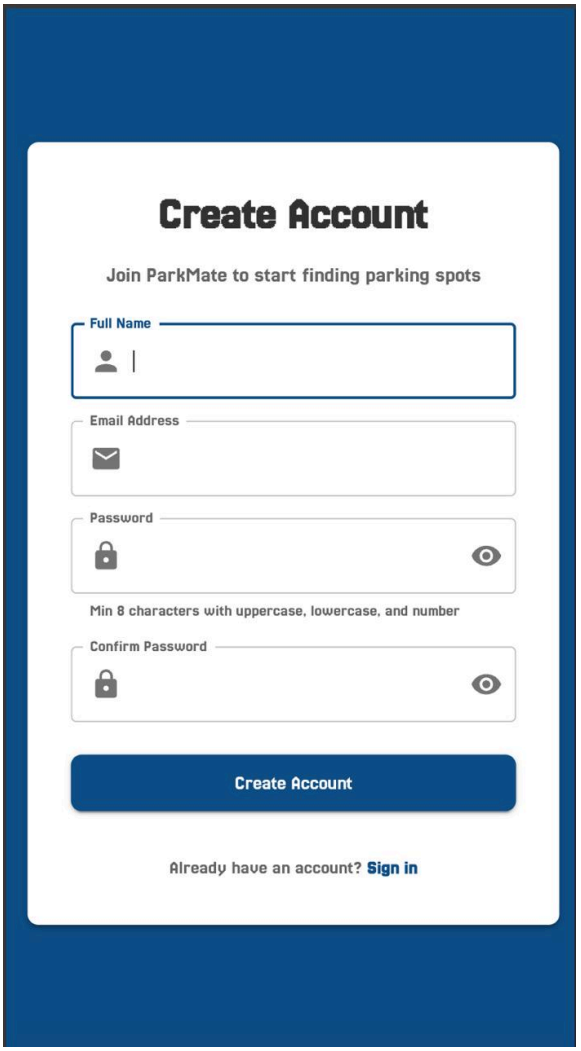
User Login Interface:

- User will have the option of logging in or signing up if user has yet to have an account
- User inputs email address and password in this page



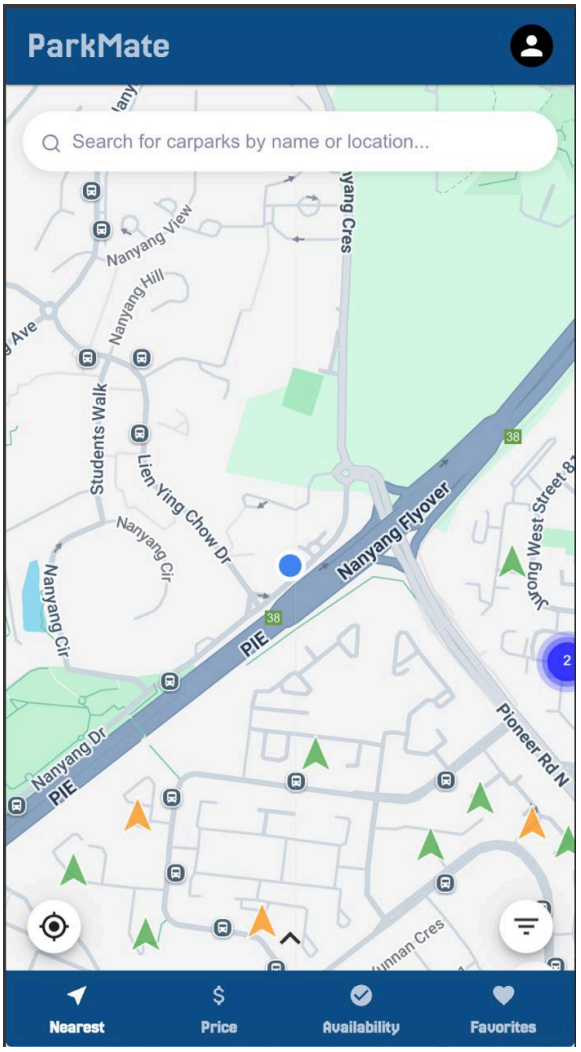
User Sign Up Interface:

- User clicking on “Sign up” button will be brought this to this interface.
- User inputs email and password in this interface with password constraints as stated in the image.



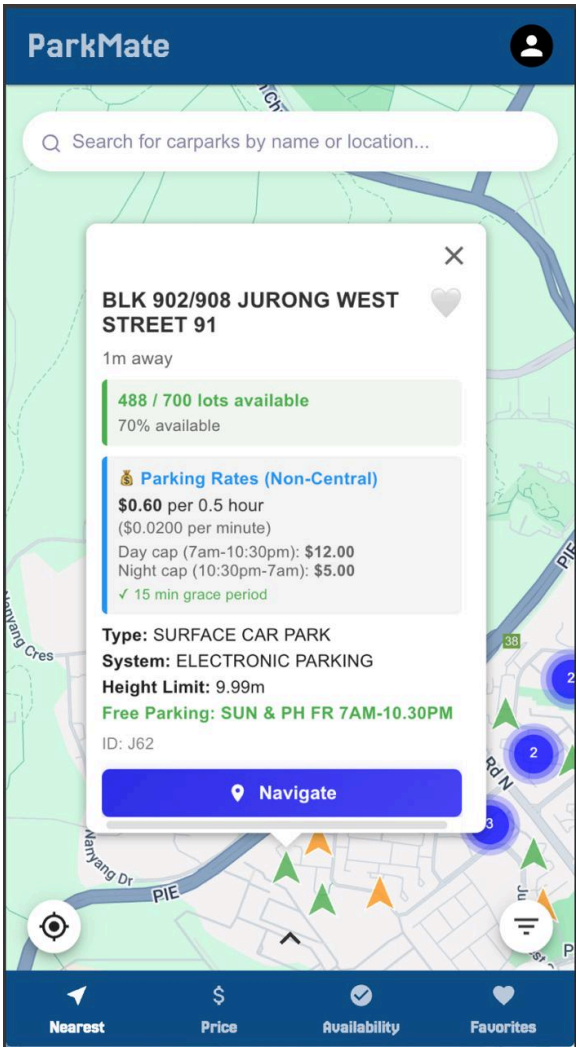
Parkmate's Homepage Interface:

- User will be in this interface after successful login, with the location being set as his current location by default.
- User will be able to search for address in the search bar
- User will be able to sort the surrounding carparks by distance, price, availability and favourites (favourited carparks)
- User able to click the filter button to select filter options.
- User able to select carpark by clicking on the marker.
- User able to interact with markers, sorting dashboard(bottom), filter button, profile button(drop down), search bar, recenter button and the map interface.



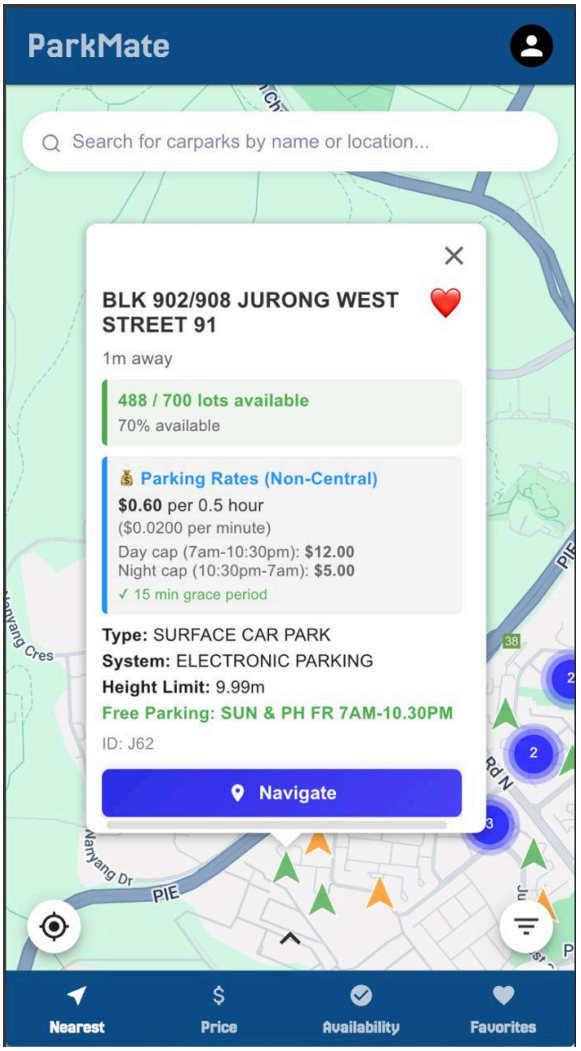
Carpark Details Interface:

- Clicking on the marker will bring user to this carpark interface
- User able to view the details such as price, type of carpark, height limit, carpark availability etc.
- User able to interact and click on navigate button, which brings user to external google maps interface.



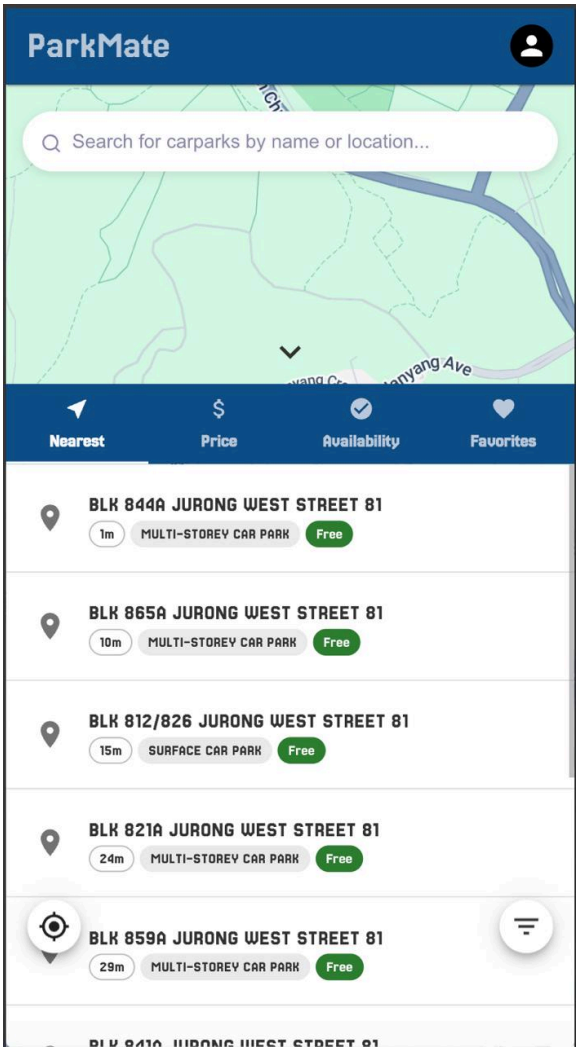
Favouriting a Carpark:

- User able to click on red heart icon button to favourite a carpark
- All favourited carparks will be able to view under “favourites”, bottom right of the homepage.



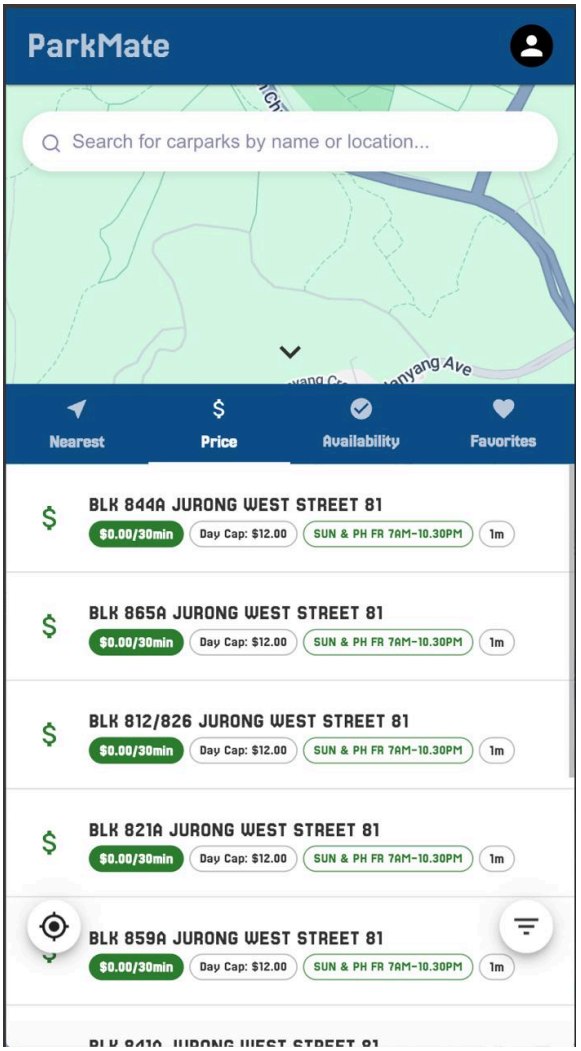
Nearest Carpark Interface:

- User interacts with "Nearest" button, able to view all the nearby carpark sorted by distance from user current live location.
- User able to interact with the sorted section, scrolling up and down to see further options.
- User able to interact with the carpark list, interacting with anyone of them would bring user to Carpark Detail Interface.



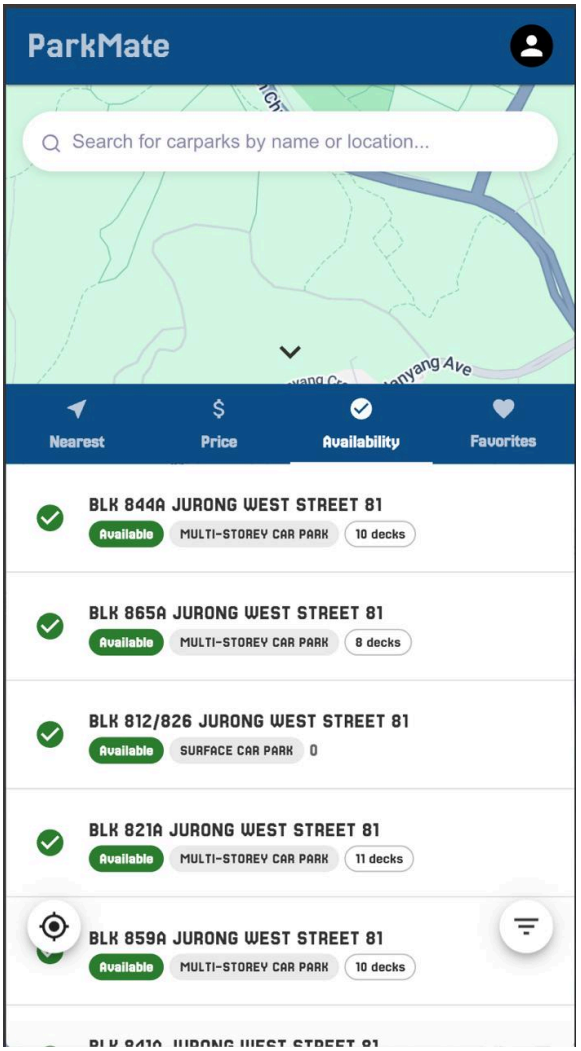
Price Sorted Carpark Interface:

- User interacts with the “Price” icon, able to view carparks sorted by pricing.
- User able to interact with the list of carparks, scrolling to see more options.
- User able to interact with the carpark list, interacting with anyone of them would bring user to Carpark Detail Interface.



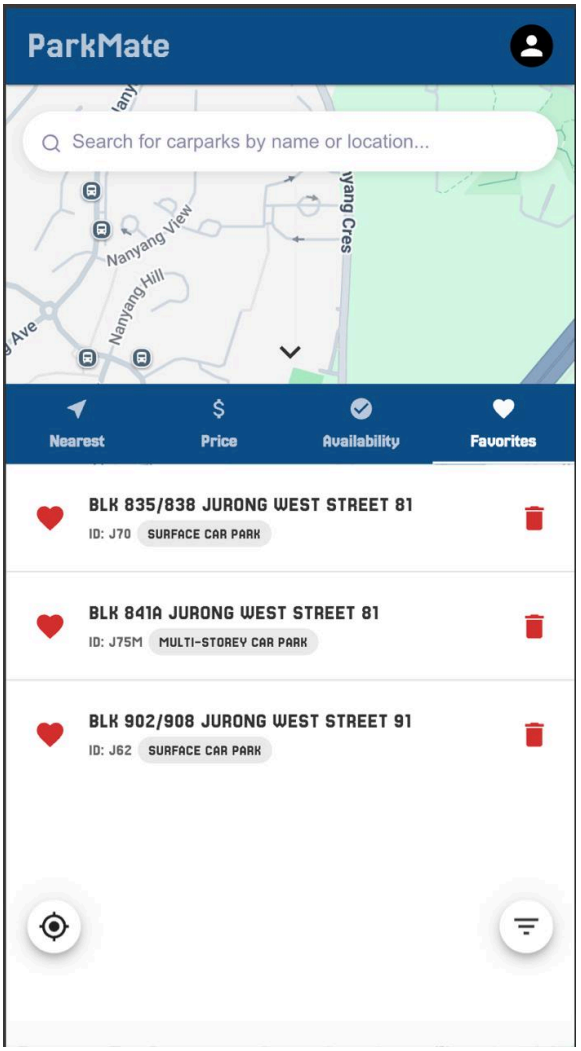
Live Availability Interface:

- User able to interact with the "Availability" icon, interacting with the icon will display a list of carparks sorted by live availability fetched from API.
- User able to interact with this interface to scroll to see other options.
- User able to interact with the carpark list, interacting with anyone of them would bring user to Carpark Detail Interface.



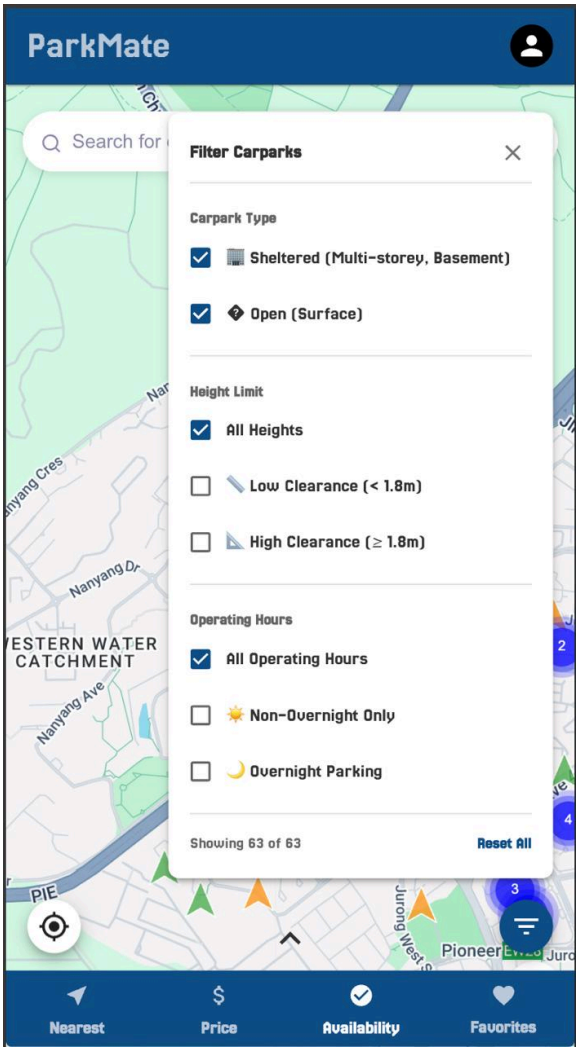
Favourite Carpark Interface:

- User interacts with "Favourites" icon, showing all carparks that user has favourited.
- User able to interact with the carpark list, interacting with anyone of them would bring user to Carpark Detail Interface.
- User able to click on the red bin icon to remove a desired carpark from the favourites.



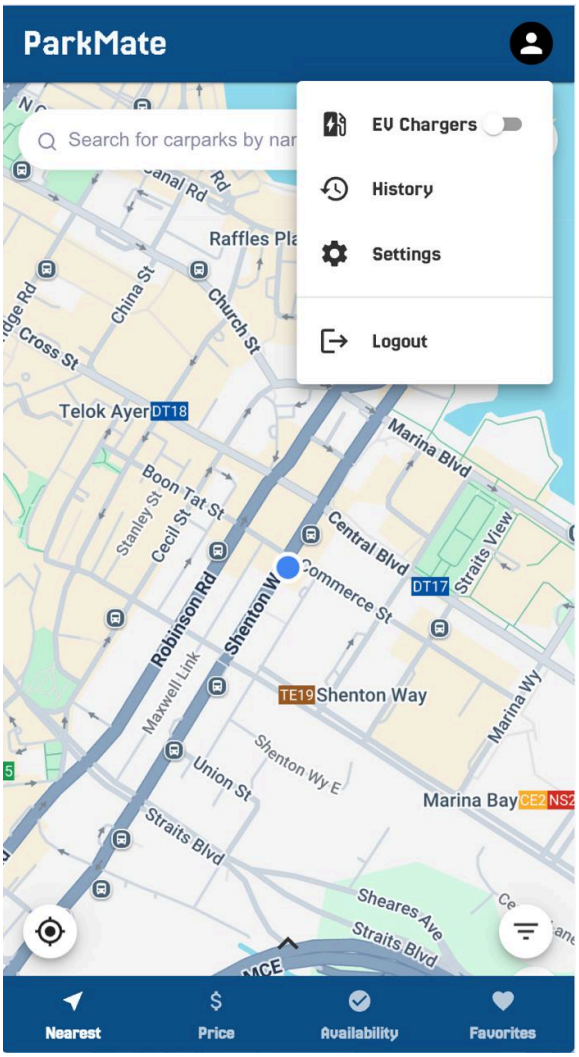
Carpark Filter Interface:

- User interacts with the bottom right filter icon, bringing user to this interface.
- User able to select desired interface to display carparks with the filters selected
- User able to interact with the “Reset all” icon, resetting all the filters that has been selected.



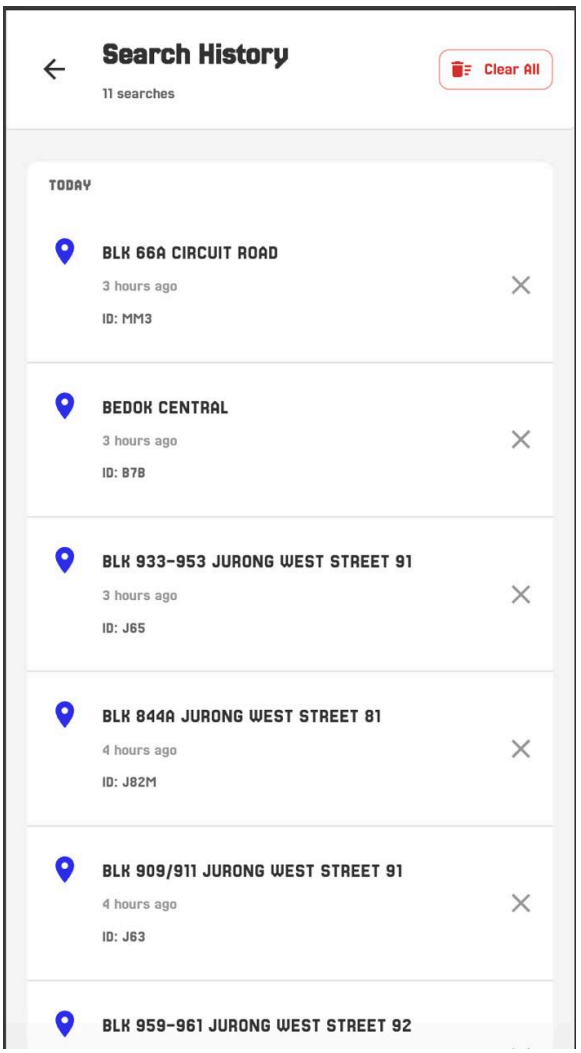
Profile Drop Down Interface:

- User interacts with the profile icon, bringing the user to this interface.
- User able to interact with other icons EV Chargers, History etc.
- Clicking on Logout would log the user out.



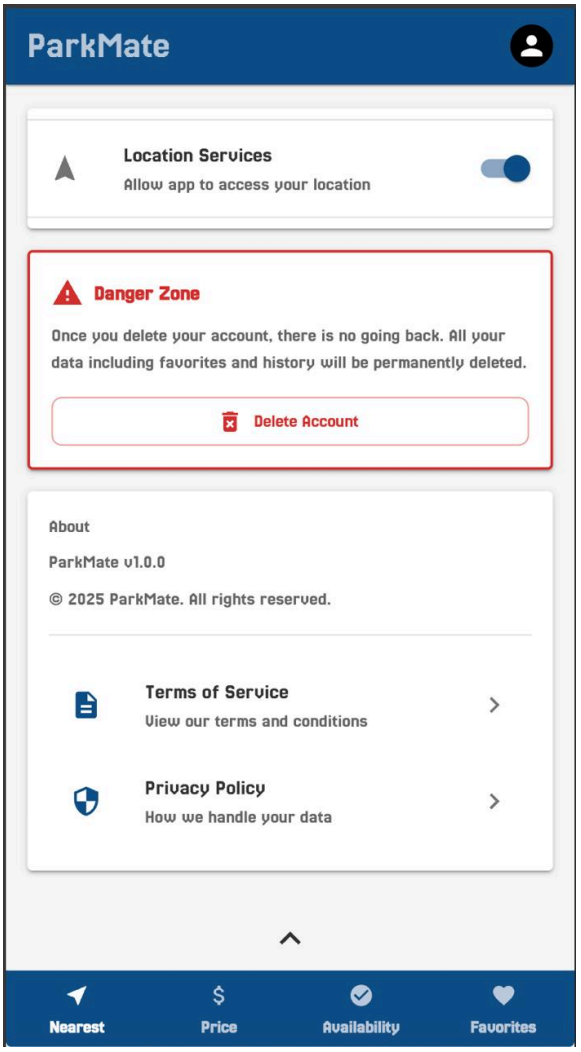
Search History Interface:

- User interacts with “History” icon in Profile Drop Down Interface, bringing them to search history interface.
- Search History interface shows all location history.
- User able to interact with the the history, bringing them to the search history location.
- User able to interact with “Clear All” interface to clear history.



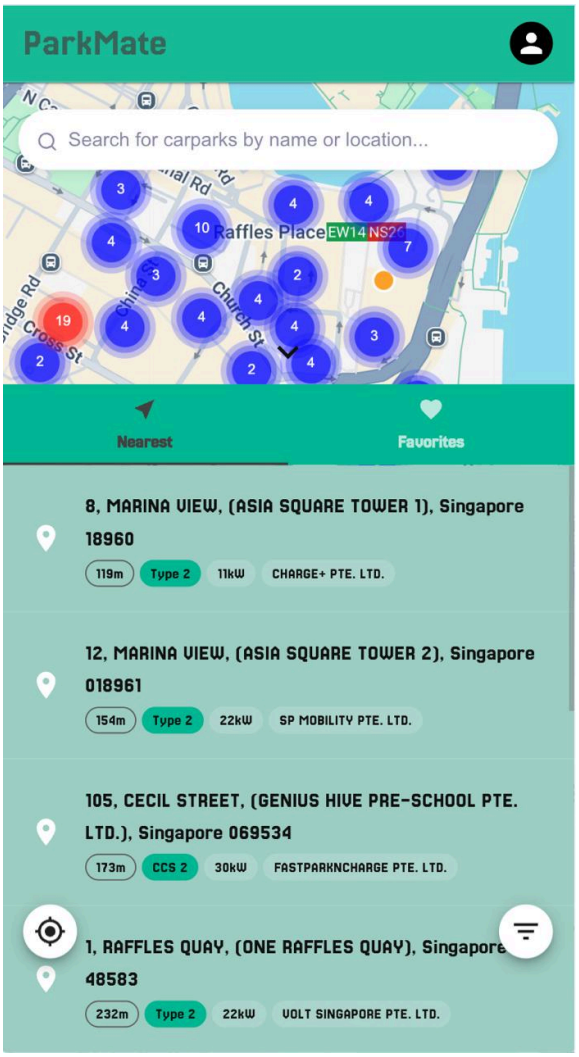
Account Setting Interface:

- User navigates to this interface by selecting the Settings icon from the profile dropdown.
- User can enable or disable Location Services, allowing the app to access real-time location data.
- System displays a Danger Zone section that warns the user about the consequences of deleting their account.
- User is able to permanently delete their account; upon confirmation, all associated data such as favorites and history will be erased.
- User can view the Terms of Service from this interface.
- User can view the Privacy Policy to understand how their data is handled.
- System displays application version information and copyright details at the bottom of the interface.



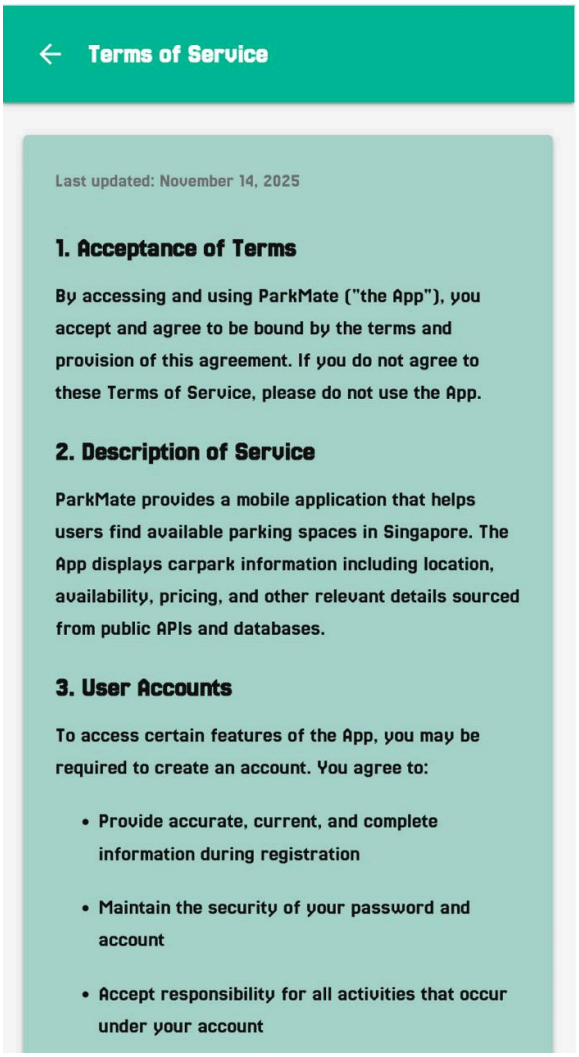
EV Mode Nearest Charger Interface:

- User can view a map displaying nearby EV charging stations based on their current location.
- User can toggle between Nearest and Favorites tabs to filter the list of chargers.
- User can view charger details such as charger type, charging speed (kW), operator, and distance from current location.
- User can select a charger from the list to view more detailed information or navigate to its location.
- System automatically updates the charger list and distances as the user moves or refreshes the interface.
- User can search for EV chargers by entering keywords such as location, building name, or operator.



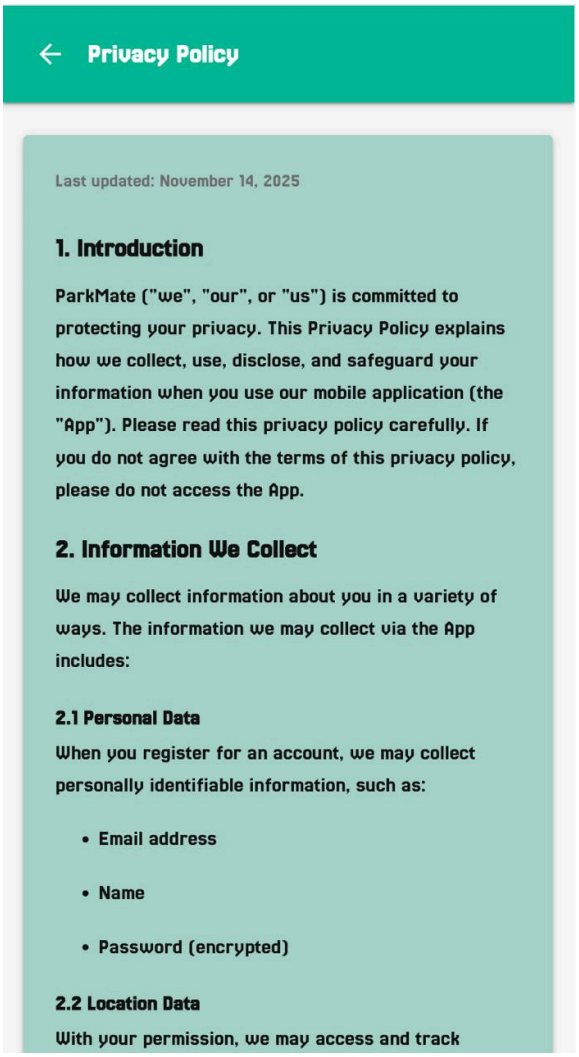
Terms of Service Interface:

- User navigates to this interface through the Account Settings page.
- User is able to read the full Terms of Service (TOS) governing the use of the ParkMate application.
- System displays important clauses such as Acceptance of Terms, Description of Service, User Accounts, Data Usage, and Limitations of Liability.
- System presents the last updated date of the Terms of Service for transparency.
- User can scroll through the document to view all TOS sections in detail.
- User can return to the previous interface using the back button located at the top left of the screen.



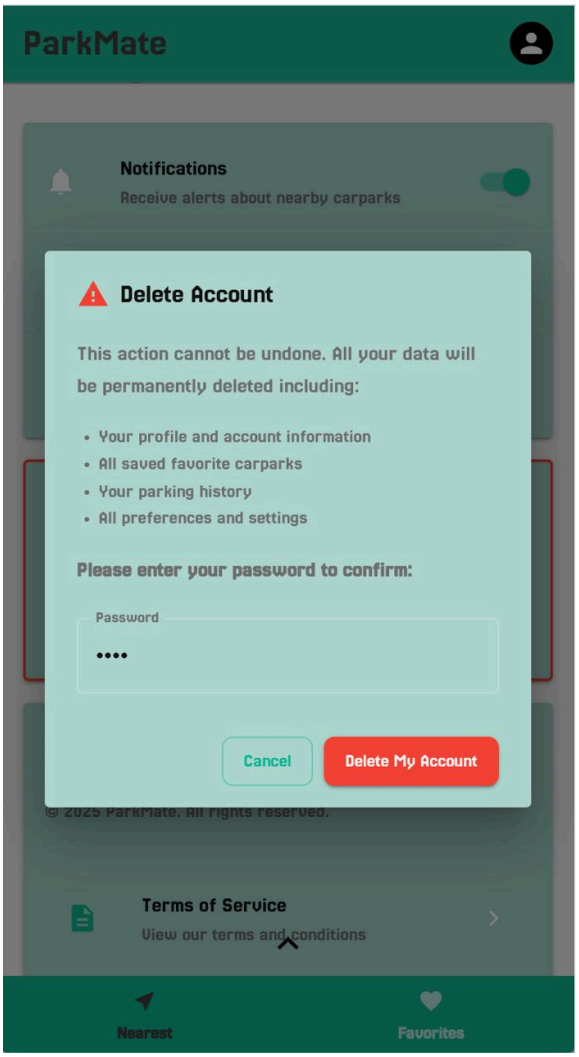
Privacy Policy Interface:

- User accesses this interface through the Account Settings page.
- User is able to view ParkMate's full Privacy Policy, which explains how personal data is collected, used, stored, and protected.
- System displays key sections such as Introduction, Information We Collect, Personal Data, Location Data, and other relevant subsections.
- System shows the last updated date of the Privacy Policy to ensure transparency and clarity.
- User can scroll through the document to review all privacy-related terms in detail.
- User can exit this interface using the back button located at the top left of the screen.



Delete Account Interface:

- User accesses this interface after selecting Delete Account within the Account Settings screen.
- System displays a clear warning informing the user that account deletion is permanent and irreversible.
- User is shown a detailed list of data that will be erased, including:
    1. Profile and account information
    2. Saved favorite carparks
    3. Parking history
    4. App preferences and settings
- User must enter their password to confirm the deletion for security purposes.



3.2 Hardware Interfaces

Hardware Interfaces Overview

End-user devices (browsers on desktops, laptops, tablets, phones) host the React 18 single-page app. The UI shell couples a persistent navigation bar with a full-height Google Maps canvas, and every page is responsive so the same interaction set works across small and large screens. Hardware-wise, these clients must expose graphics acceleration sufficient to render the Google Map and provide standard input peripherals (mouse, touch) for gestures like map panning, drawer toggles, and profile menu navigation.

Location-capable devices (phones, tablets, laptops with GPS/Wi-Fi positioning) supply user coordinates via the browser's Geolocation API. ParkMate automatically requests the current position at load, renders a marker at the returned coordinates, and falls back to Singapore defaults when sensors are unavailable or permission is denied. This interface assumes the client OS can expose a geolocation stack; if not, the software degrades gracefully by warning users and reverting to fixed coordinates.

Backend & Supporting Hardware

Dockerized services on developer/ops machines represent the primary runtime hardware. A single workstation must run multiple containers simultaneously—frontend (Node/Vite), backend (Express/Node), PostgreSQL 15, Redis 7, and optionally pgAdmin—which in aggregate require at least 4 GB of RAM allocated to Docker. Lack of memory or CPU throttling on the host hardware will stop containers from reaching the "healthy" state defined in the deployment guide.

Database and cache hosts expose dedicated TCP ports and expect stable disk I/O. PostgreSQL (port 5432) holds the 2,249 imported carparks and spatial data, while Redis (port 6379) backs caching/rate limiting. pgAdmin (port 5050) is optional but assumes the same local loopback interface. These bindings represent the logical hardware interfaces between the application stack and the storage subsystems; moving them would require reconfiguring the environment variables (DB_HOST, REDIS_HOST, etc.) baked into the backend container definitions.

API server hardware (whether local Docker or a production Node host) must listen on port 5001 for HTTPS traffic coming from the frontend. The architecture diagram presumes the Express gateway fans out to authentication, carpark, user, favorites, and history services before reaching PostgreSQL/Redis, so any hardware layer sitting between these tiers must preserve those TCP interfaces and provide reliable network throughput to keep carpark searches and map overlays real-time.

Device/Hardware Interaction Characteristics

Data flow between client hardware and sensors: browsers capture geolocation data and pass it to the React components, which in turn update Google Maps. The UI handles latency or

denial events by surfacing inline warnings and switching to fallback coordinates, so the requirement on the hardware interface is “best effort” rather than hard real-time guarantees.

Control flow between containers and host OS: developers interact with Docker through commands such as docker-compose up --build, with the host kernel providing networking and filesystem mounts. Troubleshooting steps explicitly reference host-level utilities (lsof) to inspect port bindings, underscoring that the physical interface is the host OS networking stack and virtualization features.

Networked hardware dependencies: the system assumes wired or wireless NICs that can reach Google Maps, Geocoding, and the Singapore carpark API endpoints, because the Express backend relays data from those external services and the frontend loads Google’s map tiles directly in the browser.

3.3 Software Interfaces

3.3.1 Client runtime and UI libraries

The end-user interface is a React 18 + TypeScript single-page application compiled by Vite and styled with Material UI; Redux Toolkit coordinates shared state, Google Maps React renders the interactive map, React Hook Form + Zod manage validation, and Axios is the HTTP client. These libraries define the UI widgets, theming, and validation conventions every screen must follow, and they require browsers capable of running the modern ES modules emitted by Vite.

The frontend source tree enforces these interfaces by organizing code into feature folders (auth, carpark, filters, favorites, settings) plus shared components, a Redux store, and service clients such as apiClient.ts, authService.ts, and planned carparkService.ts/userService.ts. These modules all depend on the same Axios instance (with interceptors) to talk to the backend, ensuring consistent authentication headers and error handling across features.

3.3.2 Frontend ↔ API gateway

Browser clients communicate with the Express API gateway exclusively over HTTPS, as shown in the system architecture diagram. The gateway exposes RESTful JSON endpoints grouped into authentication, carpark, user, favorites, and history services; requests carry data such as login credentials, token refreshes, search filters, favorites mutations, and history queries, while responses return JWTs, carpark listings, availability metrics, and profile settings that the frontend stores in Redux slices.

The project structure mirrors that contract: each backend module implements controller/service/repository layers plus validation schemas (auth.controller.ts, auth.service.ts, etc.), while frontend features consume the corresponding endpoints through typed service files. This coupling ensures that shared DTOs, validation rules, and error conventions remain synchronized between the two sides of the interface.

3.3.3 Backend internal interfaces (services ↔ datastores)

The Express server (Node.js 18+) persists domain data in PostgreSQL 15 with PostGIS for spatial queries and relies on Redis 7 for caching and rate limiting. Backend modules access PostgreSQL through the repository layer (using pg/node-pg-migrate) and Redis through dedicated configs, so data items such as carpark metadata, availability snapshots, user profiles, favorites, and history records are shared via SQL schemas and migrations, while Redis stores ephemeral rate-limit counters or cached lookups.

Shared middleware—JWT auth, request validation, rate limiting, and logging—acts as cross-cutting software interfaces within the backend, guaranteeing that every module processes requests through the same security and observability stack before touching the repositories.

3.3.4 External and third-party services

Several integrated commercial APIs are mandatory: Google Maps JavaScript for rendering, Google Geocoding for coordinate conversion, Singapore's carpark data API for importing/refreshing inventory, and an email service for notifications. The backend's integrations directory houses the adapters responsible for calling these services (e.g., googlemaps, geocoding, carparkApi, email), so data such as map tiles, geocodes, live carpark availability, and transactional emails move across HTTPS requests initiated from either the browser or the Express server.

Google Maps access also drives frontend behavior—API keys must be injected through .env files before the Vite build so the React components can load map tiles and Places data at runtime—which makes secure environment-variable sharing a hard constraint on deployments and CI pipelines.

3.3.5 Tooling, operating environment, and shared data artifacts

Docker and Docker Compose orchestrate all services (frontend, backend, PostgreSQL, Redis, optional pgAdmin) in both development and production, binding them to well-known ports (3000/5001/5432/6379/5050). These containers exchange data over Docker networks, mount persistent volumes for PostgreSQL, and expose health checks so that startup ordering respects service dependencies.

Environment variables (copied from .env.example files) serve as the common data-sharing mechanism for secrets and connection strings across tooling, containers, and source code; they must be populated before npm scripts, Docker builds, or migrations run, otherwise services cannot find each other or their external credentials.

3.4 Communications Interfaces

3.4.1 Browser ↔ API Gateway

Transport protocol
The React 18 single-page app communicates with the Express API exclusively over HTTPS requests that terminate on `localhost:5001` in development (or the same port exposed by Docker). This contract is established by the architecture diagram and the getting-started instructions, which bind the frontend to `http://localhost:3000` and the backend API + Swagger docs to `http://localhost:5001`. JSON is the required serialization format for both request payloads (e.g., login credentials, search filters) and responses (JWTs, carpark lists, favorites, etc.).

Authentication and headers
Every authenticated request supplies the JWT issued by `POST /api/v1/auth/login` using the `Authorization: Bearer <token>` header. Axios interceptors in the frontend's shared `apiClient` ensure the token is attached consistently, and failed requests are surfaced through the standardized toast/error pattern described in the contributing guide.

Error and status messaging
HTTP status codes from the API propagate directly to the SPA; the UI is required to display inline messaging or toast notifications that mirror the patterns defined in the authentication module, ensuring consistent handling of 4xx validation failures, 401 authentication errors, and 5xx server faults.

3.4.2 API Gateway ↔ Internal Services & Datastores

Service orchestration
The Express gateway fans out to module-specific services (authentication, carpark, user, favorites, history) through controller/service/repository boundaries. Each module communicates with PostgreSQL 15/PostGIS via the `pg` client and with Redis 7 via the configured cache utilities. All inter-module data exchanges (e.g., saving a carpark search to history after a successful query) must flow through these service contracts instead of bypassing the shared middleware stack.

Message formats
Internal service calls pass strongly typed DTOs defined in the `*.types.ts` files. Repository methods return plain JavaScript objects (later serialized as JSON), while Redis interactions store primitive counters or stringified cache entries for rate limiting and availability snapshots.

Security middleware
Every inbound HTTP request passes through JWT authentication, Helmet, CORS, compression, and express-rate-limit middleware before reaching the services. Implementers

must not skip these layers when adding new modules or routes, which effectively makes them mandatory communication gates inside the backend.

3.4.3 API Gateway ↔ External Services

Third-party integrations
The backend houses integration clients for Google Maps, Google Geocoding, Singapore's carpark data API, and transactional email providers. All outbound calls use HTTPS with JSON payloads or query-string parameters, and their credentials (API keys, dataset IDs) are supplied via environment variables declared in `backend/.env.example`. Failing to provide these keys prevents map rendering, geocoding, or scheduled imports, so CI/CD pipelines must inject the secrets before deployment.

Data refresh workflows
Scheduled tasks or manual scripts ingest the data.gov.sg carpark dataset (`d_23f946fa557947f93a8043bbef41dd09`), convert SVY21 coordinates to WGS84, and persist the results in PostgreSQL. These scripts communicate with the public API over HTTPS and rely on stable schema contracts; any upstream change in field names or formats requires matching updates to the importers.

Email and notification channels
The integrations layer sends transactional emails (password reset, account confirmations) through nodemailer-compatible providers. Payloads are formatted as HTML + plain-text messages and include contextual data provided by the relevant service modules.

3.4.4 Browser ↔ Third-Party Services

Maps and geolocation
The frontend loads the Google Maps JavaScript SDK and Places/Geocoding APIs directly in the browser using keys defined in `frontend/.env.example`. Requests are made over HTTPS and must originate from allowed domains configured in Google Cloud. The SPA also invokes the browser's Geolocation API, which requires user permission; denied requests trigger the fallback coordinates and warning flows detailed in the landing page implementation guide.

Content security
Because Google Maps scripts are injected at runtime, CSP and domain restrictions must permit `maps.googleapis.com` and related hosts. Any deployment must ensure HTTPS is enforced end-to-end so that API keys are not leaked in transit.

3.4.5 Networking, Ports, and Tooling Constraints

Port bindings
Docker Compose and manual setup scripts reserve specific local ports—3000 (frontend), 5001 (backend API/Swagger), 5432 (PostgreSQL), 6379 (Redis), and 5050 (pgAdmin).

Developers must keep these ports free or update the compose files and corresponding environment variables; otherwise the services cannot establish their expected TCP interfaces.

Environment variable propagation

`.env` files act as the shared communication channel for secrets and hostnames across containers. Missing or inconsistent values (e.g., `VITE_API_BASE_URL`, `GOOGLE_MAPS_API_KEY`, `DB_HOST`, `REDIS_HOST`) break connectivity among components, so they are treated as required communication artifacts for every deployment target.

Security posture

TLS termination (in production) and HTTPS usage (in development) are mandatory to protect JWTs and API keys. Combined with JWT-based auth and rate limiting, this ensures that all communication interfaces meet the project's security expectations without introducing alternative protocols such as FTP or WebSockets.

4. System Features

4.1 Parking Search & Selection

4.1.1 Search Destination

4.1.1.1 Description and Priority

Description: The system shall allow a user to set a parking destination by either typing an address/place name or using the device's current location. The chosen destination will be used to show nearby carparks.

Priority: High

4.1.1.2 Actors

- Authenticated end-user.

4.1.1.3 Preconditions

- User is logged in.
- Network connectivity is available.
- For "Use current location": device location services are available on the phone.

4.1.1.4 Triggers

- User taps the search bar or "Search destination" on the home screen.
- User taps "Use current location" in the search area.

4.1.1.5 Stimulus/Response Sequence

1. System displays the search area with a text field (for address/place) and a "Use current location" option.
2. User either types a destination in the text field or taps "Use current location".
3. If the user types a destination, the system checks that the field is not empty and sends the text to the location service.
4. For a typed destination, the system receives results from the location service and shows a list of suggested places with name and address.
5. For a typed destination, the user selects one place from the list and the system sets this place as the active destination.
6. If the user taps "Use current location", the system checks location permission and, if allowed, reads the current location and sets it as the active destination.

7. After an active destination is set (from text or current location), the system routes the user to the Nearby Carparks screen using this destination.

4.1.1.6 Alternate/Exception Flows

- A1 – Empty search text → System displays "Please enter a destination." and keeps the search area open.
- A2 – No matching places found → System displays "No matching locations found. Please try another address."
- A3 – Location permission denied or location turned off → System displays "Location is not available. Please turn it on in Settings or type your destination." and keeps manual search available.
- A4 – Service or network unavailable → System displays "Unable to search for locations. Please check your connection and try again later."

4.1.1.7 Postconditions

- A valid destination is stored as the active destination for the current session.
- The active destination is available to other features such as Nearby Carparks, filters, and EV options.
- If an error occurs, no new destination is set and any previous destination (if it exists) is kept.

4.1.1.8 Functional Requirements

- FR-SEARCH-01. The system shall present a search area containing a text field and a "Use current location" option.
- FR-SEARCH-02. The system shall prevent a text search when the field is empty and shall prompt the user to enter a destination.
- FR-SEARCH-03. The system shall send non-empty search text to a location service and display a list of suggested places.
- FR-SEARCH-04. The system shall set the place selected from the suggestions list as the active destination.
- FR-SEARCH-05. When the user selects "Use current location", the system shall request and use the phone's location if permission is granted.
- FR-SEARCH-06. If location permission is denied or location is turned off, the system shall show a clear message and keep manual search available.
- FR-SEARCH-07. If the location service or network is unavailable, the system shall display a simple, non-technical error message and shall not crash.

- FR-SEARCH-08. After a destination is successfully set, the system shall open the Nearby Carparks screen using that destination.

4.1.2 Nearby Carparks

4.1.2.1 Description and Priority

Description: The system shall display carparks near the active destination in a list (and optionally map) view, showing key information such as name, distance, and live availability. The user can select a carpark from this screen to see more details or to navigate later.

Priority: High

4.1.2.2 Actors

- Authenticated end-user.

4.1.2.3 Preconditions

- User is logged in.
- An active destination has been set by Search Destination or a similar flow.
- Network connectivity is available.
- The carpark data service is configured and reachable.

4.1.2.4 Triggers

- User completes Search Destination successfully.
- User revisits the Nearby Carparks screen from another part of the app.

4.1.2.5 Stimulus/Response Sequence

1. System reads the active destination coordinates.
2. System sends a request to the carpark data service for carparks within a defined radius of the active destination.
3. System receives a list of carparks and, for each carpark, extracts fields such as name, address, coordinates, distance from destination, total lots, available lots, and last-updated time.
4. System applies any active filters and sorting options (for example, distance or availability) to this list.
5. System displays the filtered list of nearby carparks in a list view, showing at least name, distance from destination, and a simple availability indicator.
6. If map view is supported, system also displays pins for these carparks on a map centred around the destination, aligned with the list.
7. User scrolls through the list and taps a carpark.

8. System opens the Carpark Details screen for the selected carpark.

4.1.2.6 Alternate/Exception Flows

- A1 – No active destination → System displays "No destination selected. Please search for a destination first." and routes the user back to the Search Destination screen.
- A2 – No carparks returned → System displays "No carparks found near this destination. Please adjust your destination or filters." and keeps the screen usable.
- A3 – Service or network unavailable → System displays "Unable to load nearby carparks. Please check your connection and try again later." and, if cached data exists, may show the last known results with a note that they may be out of date.
- A4 – Partial data (for example, missing availability) → System still lists the carpark but shows "Availability unknown" or similar text for that item.

4.1.2.7 Postconditions

- A list of nearby carparks (possibly empty) has been shown to the user for the current active destination.
- The user may have selected a carpark, in which case that carpark becomes the current selection for the Carpark Details feature.
- Any filters and sorting rules that were applied remain in effect for the current session.

4.1.2.8 Functional Requirements

- FR-NEAR-01. The system shall use the active destination coordinates to request nearby carparks from the carpark data service.
- FR-NEAR-02. The system shall display nearby carparks in a list view showing, at minimum, the carpark name and distance from the destination.
- FR-NEAR-03. The system shall display a simple availability indicator for each carpark when availability data is provided by the service.
- FR-NEAR-04. If map view is supported, the system shall show nearby carparks as pins on a map centred around the destination and consistent with the list content.
- FR-NEAR-05. The system shall apply any active filters and sorting options when presenting nearby carparks.

- FR-NEAR-06. When the user taps a carpark in the list (or on the map), the system shall open the Carpark Details screen for that carpark.
- FR-NEAR-07. If no carparks are returned for the destination, the system shall show a clear “No carparks found” message and shall not crash.
- FR-NEAR-08. If the carpark data service or network is unavailable, the system shall show a simple, non-technical error message and shall not crash.

4.1.3 Filters & Sorting

4.1.3.1 Description and Priority

Description: The system shall allow the user to refine the list of nearby carparks using filters and to change the order of results using sorting options. Filters and sorting help the user quickly find a suitable carpark based on distance, availability, and other attributes.

Priority: High

4.1.3.2 Actors

- Authenticated end-user.

4.1.3.3 Preconditions

- User is logged in.
- Nearby carparks for an active destination are available.
- Network connectivity is available when refreshed data is needed.

4.1.3.4 Triggers

- User taps a Filters button or icon on the Nearby Carparks screen.
- User taps a Sort button or control on the Nearby Carparks screen.

4.1.3.5 Stimulus/Response Sequence

1. System displays the Nearby Carparks list for the current active destination.
2. User taps the Filters button.
3. System opens a Filters panel showing one or more options such as distance limit, minimum availability, price or tariff preference, and other supported attributes.
4. User adjusts one or more filter options and taps Apply.
5. System applies the selected filters to the list of nearby carparks and updates the results on the Nearby Carparks screen.

6. User taps the Sort control.

7. System displays sorting options such as sort by distance (nearest first) and sort by availability (most available first).
8. User selects a sorting option.
9. System reorders the filtered list according to the chosen sort order and shows the updated list to the user.

4.1.3.6 Alternate/Exception Flows

- A1 – No carparks match selected filters → System displays "No carparks match these filters. Please adjust your filters and try again." and allows the user to change or clear filters.
- A2 – User cancels filters → User taps Cancel or closes the Filters panel without applying; system keeps the previous filters and list unchanged.
- A3 – User selects Clear filters → System removes all active filters, restores default filter settings, and reloads the full nearby carpark list for the current destination.
- A4 – Service or network unavailable when reloading results → System displays "Unable to update results. Please check your connection and try again later." and keeps the last shown list on screen.

4.1.3.7 Postconditions

- A set of filters and a chosen sort order (or defaults) are stored for the current session.
- The Nearby Carparks list reflects the active filters and sort order.
- If an error occurs or the user cancels, the previous filters and sort order remain in effect.

4.1.3.8 Functional Requirements

- FR-FILT-01. The system shall provide a Filters control on the Nearby Carparks screen that opens a Filters panel when tapped.
- FR-FILT-02. The system shall allow the user to adjust one or more filter options (for example, distance limit, minimum availability, or other supported attributes) and apply them to the nearby carpark list.
- FR-FILT-03. The system shall provide a way to clear all active filters and return to the default unfiltered list of nearby carparks.
- FR-FILT-04. The system shall provide a Sort control that lets the user choose at least "sort by distance (nearest first)" and "sort by availability (most available first)".

- FR-FILT-05. The system shall reorder the nearby carpark list according to the selected sort option while keeping any active filters applied.
- FR-FILT-06. If no carparks match the active filters, the system shall show a clear “no matches” message and shall not crash.
- FR-FILT-07. If the service or network is unavailable when applying filters or sorting, the system shall display a simple, non-technical error message and shall keep the last shown list.

4.1.4 Carpark Details

4.1.4.1 Description and Priority

Description: The system shall show a detailed view for a selected carpark, including address, distance, live availability, last updated time, basic attributes (such as opening hours and height limit), and read-only tariffs. From this screen, the user can also favourite the carpark and access navigation features.

Priority: Medium-High

4.1.4.2 Actors

- Authenticated end-user.

4.1.4.3 Preconditions

- User is logged in.
- A carpark has been selected from the Nearby Carparks list, map, favourites, or history.
- Network connectivity is available when the latest live data is needed.

4.1.4.4 Triggers

- User taps a carpark in the Nearby Carparks list or map.
- User taps a carpark from the favourites list.
- User taps a carpark from the history list.

4.1.4.5 Stimulus/Response Sequence

1. System receives the selected carpark identifier.
2. System retrieves stored details for this carpark from local data or the carpark service.
3. System requests the latest live availability and last-updated time for this carpark, if available.
4. System opens the Carpark Details screen.

5. System displays key information such as carpark name, address, distance from destination, live available lots, total lots, and last-updated time.
6. System displays additional attributes such as opening hours (if known), maximum vehicle height, simple notes or restrictions, and read-only tariff information.
7. System shows action controls such as favourite/unfavourite and navigate/open-in-maps (and copy address if needed).
8. User reviews the details and may tap favourite/unfavourite or navigation actions.
9. System applies the chosen action (for example, updates favourite status or calls the navigation feature) and keeps the details screen visible until the user closes it.

4.1.4.6 Alternate/Exceptions Flows

- A1 – Carpark record not found → System displays "Carpark details are not available." and returns the user to the previous screen.
- A2 – Live availability unavailable → System displays static information only and shows "Availability not available" or similar text instead of live numbers.
- A3 – Service or network unavailable while loading details → System displays "Unable to load the latest carpark details. Please try again later." and, if possible, shows any cached or basic information that is still available.

4.1.4.7 Postconditions

- The selected carpark has been shown to the user with all available details.
- If the user toggled favourite status, the favourite list is updated for that user.
- If viewing a carpark counts as a selection, the carpark may be recorded in the user's history according to the history rules.

4.1.4.8 Functional Requirements

- FR-DETAIL-01. The system shall provide a Carpark Details screen that opens when the user selects a carpark from a list, map, favourites, or history.
- FR-DETAIL-02. The system shall display, at minimum, the carpark name, address, distance from the active destination, available lots (if known), total lots (if known), and the last-updated time for availability data.
- FR-DETAIL-03. The system shall display additional attributes where available, such as opening hours, maximum vehicle height, simple notes or restrictions, and read-only tariff information.

- FR-DETAIL-04. The system shall not allow the user to edit tariff or carpark attribute information from the app.
- FR-DETAIL-05. The system shall provide a control on the Carpark Details screen to favourite or unfavourite the carpark for the current user.
- FR-DETAIL-06. The system shall provide a control on the Carpark Details screen to start navigation using the Open-in-Maps feature and, if needed, to copy the address.
- FR-DETAIL-07. If live availability data for the carpark is missing or cannot be loaded, the system shall show a clear message and shall still display any static details that are available.
- FR-DETAIL-08. If localisation is enabled, the system shall display all messages on the Carpark Details screen in the selected app language.

4.1.5 Live Availability Refresh

4.1.5.1 Description and Priority

Description: The system shall allow the user to refresh live availability for nearby carparks (and, where relevant, a selected carpark) so that the user can see up-to-date information. Refresh can be triggered by the user or happen automatically at safe time intervals.

Priority: High

4.1.5.2 Actors

- Authenticated end-user.

4.1.5.3 Preconditions

- User is logged in.
- A list of nearby carparks or a Carpark Details screen is already displayed.
- Network connectivity is available when new live data is needed.

4.1.5.4 Triggers

- User performs a pull-to-refresh gesture on the Nearby Carparks list or taps a refresh button.
- User performs a pull-to-refresh gesture or taps a refresh button on the Carpark Details screen.
- A configured auto-refresh interval is reached while the user is viewing the screen.

4.1.5.5 Stimulus/Response Sequence

1. System displays the current availability for nearby carparks (or a single carpark) along with a last updated time.
2. User triggers a manual refresh using pull-to-refresh or a refresh button, or the auto-refresh timer fires.
3. System checks the time since the last successful refresh to ensure that the minimum refresh interval has passed.
4. If the refresh can proceed, the system shows a loading indicator and temporarily disables additional refresh actions.
5. System sends a request to the carpark data service for updated availability for the relevant carparks.
6. System receives the updated availability data and applies it to the in-memory list or current carpark.
7. System updates the displayed availability values and last updated time on the screen.
8. System hides the loading indicator and re-enables the refresh control.

4.1.5.6 Alternate/Exception Flows

- A1 – Network or service unavailable → System displays "Unable to refresh live availability. Please check your connection and try again later." and keeps the previous availability values and last updated time.
- A2 – Refresh requested too frequently → System does not call the service again and may show "Please wait a moment before refreshing again." while keeping the current values.
- A3 – Partial data returned (some carparks missing) → System updates availability for carparks that have new data and keeps previous values for others, showing a simple message if needed such as "Some availability data may be out of date."

4.1.5.7 Postconditions

- If the refresh is successful, the displayed availability and last updated time reflect the latest response from the carpark data service.
- If the refresh fails or is blocked by the minimum interval, the previously displayed availability and last updated time are preserved.
- The system is ready for a future manual or auto refresh after the allowed interval.

4.1.5.8 Functional Requirements

- FR-REFRESH-01. The system shall provide a refresh control (for example, pull-to-refresh or a refresh button) on the Nearby Carparks and Carpark Details screens.
- FR-REFRESH-02. The system shall request updated availability data for the relevant carparks when a refresh is triggered and the minimum refresh interval has passed.
- FR-REFRESH-03. The system shall show a visible loading indicator while a refresh is in progress and shall temporarily disable repeated refresh actions.
- FR-REFRESH-04. The system shall apply a minimum refresh interval (cooldown) between successive calls to the carpark data service.
- FR-REFRESH-05. On a successful refresh, the system shall update the displayed availability values and last updated time.
- FR-REFRESH-06. If the network or carpark data service is unavailable, the system shall display a simple, non-technical error message and keep the previously displayed availability data.
- FR-REFRESH-07. If only partial new availability data is received, the system shall update the carparks that have new data and retain the previous values for carparks without updated data.

4.1.6 Manage Favourites

4.1.6.1 Description and Priority

Description: The system shall allow the user to mark and unmark carparks as favourites and view a list of favourite carparks for quick access. Favourites are stored per user account.

Priority: Medium

4.1.6.2 Actors

- Authenticated end-user.

4.1.6.3 Preconditions

- User is logged in.
- Carpark data is available so that a carpark can be selected.

4.1.6.4 Triggers

- User taps a favourite icon (for example, a star or heart) for a carpark on the Nearby Carparks list, Carpark Details screen, or other supported list.
- User opens the Favourites screen or section from the app navigation.

4.1.6.5 Stimulus/Response Sequence

1. System displays a list of carparks (for example, Nearby Carparks, History) or a Carpark Details screen, showing a favourite icon for each carpark.
2. User taps the favourite icon for a carpark that is not yet favourited.
3. System records this carpark as a favourite for the current user account.
4. System updates the favourite icon (for example, filled star) to show that the carpark is now a favourite.
5. User taps the favourite icon again for a carpark that is already favourited.
6. System removes this carpark from the favourites list for the current user and updates the icon to show that it is no longer a favourite.
7. User opens the Favourites screen from the app navigation.
8. System retrieves the list of favourited carparks for the current user and displays them in a list view similar to the Nearby Carparks list (for example, showing name, distance, and availability).
9. User taps a carpark in the Favourites list.
10. System opens the Carpark Details screen for the selected favourite carpark.

4.1.6.6 Alternate/Exception Flows

- A1 – User not logged in → System hides favourite controls or prompts the user to log in before they can favourite a carpark.
- A2 – Service or network unavailable when saving/removing favourite → System displays “Unable to update favourites. Please try again later.” and keeps the previous favourite state.
- A3 – No favourites saved yet → System displays “You have no favourite carparks yet.” and may provide a short hint such as “Tap the star icon on a carpark to add it here.”

4.1.6.7 Postconditions

- The favourites list for the current user is updated to reflect any add or remove actions.
- The Favourites screen shows the current set of favourited carparks for that user.
- If an error occurs during an update, the previous favourites list is preserved.

4.1.6.8 Functional Requirements

- FR-FAV-01. The system shall display a favourite control (for example, a star or heart icon) for each carpark in relevant screens such as the Nearby Carparks list and Carpark Details.
- FR-FAV-02. The system shall add a carpark to the current user's favourites list when the user taps the favourite control for a carpark that is not yet favourited.
- FR-FAV-03. The system shall remove a carpark from the current user's favourites list when the user taps the favourite control for a carpark that is already favourited.
- FR-FAV-04. The system shall provide a Favourites screen that shows all carparks favourited by the current user in a list view.
- FR-FAV-05. The system shall open the Carpark Details screen when the user taps a carpark in the Favourites list.
- FR-FAV-06. If the service or network is unavailable when updating favourites, the system shall show a simple, non-technical error message and shall not lose the previous favourites list.

4.2 User Features - Account & Profile

4.2.1 Register

4.2.1.1 Description and Priority

Description: The system shall allow a new user to create a ParkMate account using an email address, password, and confirm-password. Registration enables access to personalised features such as favourites, history, and preferences.
Priority: High

4.2.1.2 Actors

- Authenticated end-user (not yet registered, using the app in guest mode or on first install).

4.2.1.3 Preconditions

- User is not currently logged in.
- Network connectivity is available.

4.2.1.4 Triggers

- User selects Sign Up from the welcome or login screen.

4.2.1.5 Stimulus/Response Sequence

1. System displays the Sign Up form with fields for Email, Password, Confirm Password, and a Create Account button.
2. User fills in the Email, Password, and Confirm Password fields.
3. User taps the Create Account button.
4. System checks that all fields are filled in.
5. System checks that the email address is in a valid email format.
6. System checks that the password meets the minimum length requirement.
7. System checks that Password and Confirm Password are the same.
8. System checks whether the email address is already registered.
9. If all checks pass, the system creates a new user record, stores the password securely (for example, as a hash rather than in plain text), and assigns a unique user identifier.
10. System displays a success message such as "Account created successfully."
11. System routes the user to the Login screen or signs the user in automatically, depending on the chosen flow.

4.2.1.6 Alternate/Exception Flows

- A1 - Any field empty → System displays "Please fill in all required fields." and does not create an account.
- A2 - Invalid email format → System displays "Please enter a valid email address." and does not create an account.
- A3 - Password too short or does not meet policy → System displays "Password must be at least 8 characters." and does not create an account.
- A4 - Password and Confirm Password do not match → System displays "Passwords do not match." and does not create an account.
- A5 - Email already registered → System displays "Email already in use. Please log in or use another email." and does not create an account.
- A6 - Service or network unavailable → System displays "Service unavailable. Please try again later." and does not create an account.

4.2.1.7 Postconditions

- On success, a new user record exists, with a unique user identifier and a password stored securely (not in plain text).
- The user can now log in to access personalised features.
- On failure, no new user account is created, and the user is informed of the reason.

4.2.1.8 Functional Requirements

- FR-REG-01. The system shall present a registration form containing Email, Password, and Confirm Password fields.
- FR-REG-02. The system shall require all registration fields to be filled in before creating an account.
- FR-REG-03. The system shall validate the email using a standard email format check and reject invalid formats.
- FR-REG-04. The system shall enforce a password policy of at least 8 characters.
- FR-REG-05. The system shall verify that Password and Confirm Password are identical before creating an account.
- FR-REG-06. The system shall reject registration if the email address is already registered and shall show an “Email already in use” message.
- FR-REG-07. The system shall store passwords only in a secure, non-plain-text form (for example, hashed) and shall not log or display the plain-text password after submission.
- FR-REG-08. On successful registration, the system shall create a unique user identifier for the new account and show a clear success message.
- FR-REG-09. If the registration service or network is unavailable, the system shall display a simple, non-technical error message and shall not create a partial or incomplete account.

4.2.2 Login

4.2.2.1 Description and Priority

Description: The system shall allow an existing user to log in to ParkMate using their registered email address and password. Successful login gives access to personalised features such as favourites, history, and any saved preferences.

Priority: High

4.2.2.2 Actors

- Registered end-user.

4.2.2.3 Preconditions

- User has an existing ParkMate account.
- User is currently logged out.

- Network connectivity is available.

4.2.2.4 Triggers

- User opens the app and chooses Login from the welcome screen.
- User logs out and later returns to the Login screen.

4.2.2.5 Stimulus/Response Sequence

1. System displays the Login screen with fields for Email and Password and a Login button.
2. User enters their registered email address and password in the respective fields.
3. User taps the Login button.
4. System checks that both Email and Password fields are not empty.
5. System validates that the email address is in a basic valid format.
6. If basic checks pass, system sends the email and password to the authentication service.
7. Authentication service checks whether the email exists and whether the password matches the stored credentials.
8. If the credentials are valid, the system creates a logged-in session for the user.
9. System routes the user to the main home screen (for example, Search Destination / Nearby Carparks) and the user can now use personalised features.

4.2.2.6 Alternate/Exception Flows

- A1 - Any field empty → System displays “Please fill in both email and password.” and does not attempt to log in.
- A2 - Invalid email format → System displays “Please enter a valid email address.” and does not attempt to log in.
- A3 - Email not registered or password incorrect → System displays “Incorrect email or password. Please try again.” and keeps the user on the Login screen.
- A4 - Account locked or disabled (if supported) → System displays “Your account is not available. Please contact support or try again later.” and does not log the user in.
- A5 - Service or network unavailable → System displays “Unable to log in. Please check your connection and try again later.” and does not log the user in.

4.2.2.7 Postconditions

- On success, the user is marked as logged in, and a valid session is created for that user on the device.
- The user can access personalised features such as favourites, history, and any future profile settings.
- On failure, no session is created, and the user remains on the Login screen with an explanation of the issue.

4.2.2.8 Functional Requirements

- FR-LOGIN-01. The system shall present a Login screen containing Email and Password fields and a Login button.
- FR-LOGIN-02. The system shall require that both Email and Password fields are filled in before attempting to authenticate.
- FR-LOGIN-03. The system shall validate the email using a basic email format check and reject clearly invalid formats.
- FR-LOGIN-04. The system shall send the entered email and password to the authentication service to verify the credentials.
- FR-LOGIN-05. If the email is not registered or the password does not match, the system shall display a generic error message and shall not reveal which part is incorrect.
- FR-LOGIN-06. On successful authentication, the system shall create a logged-in session and navigate the user to the main home screen of the app.
- FR-LOGIN-07. If the authentication service or network is unavailable, the system shall display a simple, non-technical error message and shall not log the user in.
- FR-LOGIN-08. The system shall not log or display the user's password in plain text at any point during the login process.

4.2.3 Logout

4.2.3.1 Description and Priority

Description: The system shall allow a logged-in user to log out of ParkMate, ending the current session on that device and returning the user to the welcome or login screen.

Priority: Medium

4.2.3.2 Actors

- Logged-in end-user.

4.2.3.3 Preconditions

- User is currently logged in.
- Network connectivity is available if the server needs to be told about the logout.

4.2.3.4 Triggers

- User taps a Logout option from a profile screen, settings screen, or app menu.

4.2.3.5 Stimulus/Response Sequence

1. System displays a screen or menu that includes a Logout option.
2. User taps the Logout option.
3. System may show a simple confirmation prompt such as “Are you sure you want to log out?” with Yes and Cancel options.
4. If the user confirms, the system clears the user’s session data on the device (for example, tokens and in-memory user information).
5. System sends a logout request to the backend service if needed to invalidate the session on the server.
6. System navigates the user to the welcome or login screen.
7. System ensures that screens which require a logged-in user (for example, Favourites or History) cannot be accessed until the user logs in again.

4.2.3.6 Alternate/Exception Flows

- A1 - User cancels at confirmation → System closes the confirmation prompt and keeps the user logged in.
- A2 - Network or service unavailable during logout → System clears the local session, shows “You have been logged out, but we could not contact the server.” if needed, and still returns the user to the login or welcome screen.
- A3 - Session already expired on server → System treats the logout as successful, clears any local session data, and returns the user to the login or welcome screen.

4.2.3.7 Postconditions

- On success, the user is logged out on the device, and a new login is required to access personalised features.
- Any screens that require authentication are no longer directly usable until the user logs in again.
- If an error occurs when contacting the server, the local session is still cleared and the user is taken to the login or welcome screen.

4.2.3.8 Functional Requirements

- FR-LOGOUT-01. The system shall provide a visible Logout option for logged-in users in a reasonable place such as a profile, settings, or menu screen.
- FR-LOGOUT-02. When the Logout option is confirmed, the system shall clear local session data for the current user on the device.
- FR-LOGOUT-03. After logout, the system shall navigate the user to the welcome or login screen.
- FR-LOGOUT-04. The system shall prevent direct access to screens that require login (for example, Favourites or History) after the user has logged out.
- FR-LOGOUT-05. If the system cannot contact the backend service during logout, it shall still clear the local session and show a simple, non-technical message if needed, without crashing.

4.2.4 Password Reset

4.2.4.1 Description and Priority

Description: The system shall allow a user who has forgotten their password to reset it using their registered email address. The user requests a reset, receives a reset link or code, and then sets a new password.

Priority: High

4.2.4.2 Actors

- Registered end-user.

4.2.4.3 Preconditions

- The user has a registered ParkMate account.
- User is currently logged out (or at least not able to log in).
- Network connectivity is available.

4.2.4.4 Triggers

- User taps the “Forgot Password” link on the Login screen.
- User opens the app from a password reset link or code received by email.

4.2.4.5 Stimulus/Response Sequence

1. System displays the Login screen with a “Forgot Password” link.
2. User taps the “Forgot Password” link.

3. System displays a Password Reset Request screen with an Email field and a Send Reset Link button.
4. User enters their email address and taps Send Reset Link.
5. System checks that the Email field is not empty and is in a valid email format.
6. If checks pass, system sends a password reset request to the backend service.
7. Backend service verifies that the email is registered and, if valid, creates a time-limited reset token and sends a reset email to the user.
8. System displays a message such as “If an account with this email exists, a reset link has been sent.” and returns the user to the Login screen.
9. User opens the reset email and taps the reset link or enters the provided code, which opens the Password Reset screen in the app (or browser).
10. System displays fields for New Password and Confirm New Password and a Reset Password button.
11. User enters a new password and confirmation, then taps Reset Password.
12. System checks that both password fields are filled, that the password meets the minimum length requirement, and that both entries match.
13. If checks pass and the reset token is valid and not expired, the backend updates the user’s stored password to the new value (securely stored as a hash).
14. System displays a success message such as “Your password has been reset.”
15. System directs the user back to the Login screen so they can log in with the new password.

4.2.4.6 Alternate/Exception Flows

- A1 - Email field empty → System displays “Please enter your email address.” and does not send a reset request.
- A2 - Invalid email format → System displays “Please enter a valid email address.” and does not send a reset request.
- A3 - Email not registered → System displays a generic message such as “If an account with this email exists, a reset link has been sent.” and does not reveal whether the email is registered.
- A4 - Service or network unavailable when sending reset request → System displays “Unable to send reset link. Please check your connection and try again later.” and does not proceed.

- A5 - Reset token invalid or expired → When user opens the reset link, system displays “This reset link is no longer valid. Please request a new password reset.” and returns the user to the Password Reset Request or Login screen.
- A6 - New password fields empty or too short → System displays a message such as “Password must be at least 8 characters.” and does not reset the password.
- A7 - New Password and Confirm New Password do not match → System displays “Passwords do not match.” and does not reset the password.
- A8 - Service or network unavailable when saving new password → System displays “Unable to reset password at this time. Please try again later.” and does not change the stored password.

4.2.4.7 Postconditions

- On success, the user’s password is updated to the new value, stored securely (not in plain text).
- The user can log in with the new password, and the old password no longer works.
- On failure, the existing password remains unchanged, and the user is informed of the reason where appropriate.

4.2.4.8 Functional Requirements

- FR-RESET-01. The system shall provide a “Forgot Password” link on the Login screen that opens a Password Reset Request screen.
- FR-RESET-02. The system shall allow the user to enter an email address and request a password reset link or code.
- FR-RESET-03. The system shall validate that the email field is not empty and is in a valid email format before sending a reset request.
- FR-RESET-04. The system shall send a password reset email containing a time-limited reset link or code if the email corresponds to a registered account, without revealing whether the email is registered in the user-facing message.
- FR-RESET-05. The system shall provide a Password Reset screen, accessible from the reset link or code, with fields for New Password and Confirm New Password.
- FR-RESET-06. The system shall validate that the new password meets the minimum password policy and that New Password and Confirm New Password are identical before resetting the password.

- FR-RESET-07. If the reset token is invalid or expired, the system shall not reset the password and shall show a simple message indicating that a new reset request is needed.
- FR-RESET-08. On successful password reset, the system shall update the stored password securely and show a success message, then direct the user to the Login screen.
- FR-RESET-09. If the service or network is unavailable during either the reset request or the password update, the system shall display a simple, non-technical error message and shall not create a partial or inconsistent password change.

4.2.5 Delete Account

4.2.5.1 Description and Priority

Description: The system shall allow a logged-in user to permanently delete their ParkMate account. Deleting an account removes access to personalised features and removes or anonymises user-specific data such as favourites and history.

Priority: Medium–High

4.2.5.2 Actors

- Logged-in end-user.

4.2.5.3 Preconditions

- User is currently logged in.
- Network connectivity is available.

4.2.5.4 Triggers

- User taps a Delete Account option from a profile or settings screen.

4.2.5.5 Stimulus/Response Sequence

1. System displays a profile or settings screen that includes a Delete Account option.
2. User taps the Delete Account option.
3. System shows a confirmation screen or dialog explaining that account deletion is permanent and that the user will lose access to their data.
4. System may ask the user to re-enter their password or confirm their intention (for example, by typing DELETE) to avoid accidental deletion.

5. User confirms the delete action by pressing a Delete button on the confirmation screen.
6. System sends a delete account request to the backend service for the current user account.
7. Backend service deletes or securely anonymises the user's account record and associated personalised data such as favourites and history, according to the data policy.
8. System receives a success response from the backend.
9. System clears any local session data and cached user-specific data from the device.
10. System displays a message such as "Your account has been deleted."
11. System navigates the user to the welcome or login screen as a non-logged-in user.

4.2.5.6 Alternate/Exception Flows

- A1 - User cancels at the confirmation step → System closes the confirmation screen and keeps the user account active and logged in.
- A2 - Password re-entry fails (if required) → System displays "Incorrect password. Your account has not been deleted." and keeps the user account active.
- A3 - Service or network unavailable → System displays "Unable to delete your account at this time. Please check your connection and try again later." and does not delete the account.
- A4 - Backend cannot complete deletion → System displays "We could not delete your account. Please try again later." and keeps the user account active.

4.2.5.7 Postconditions

- On success, the user's account is deleted or anonymised on the backend, and the user is logged out on the device.
- The user can no longer log in with the deleted account credentials.
- Personalised data such as favourites and history linked to the deleted account are removed or anonymised according to the system's data policy.
- On failure or cancel, the user's account remains active and the user stays logged in.

4.2.5.8 Functional Requirements

- FR-DEL-01. The system shall provide a Delete Account option for logged-in users on a profile or settings screen.
- FR-DEL-02. The system shall show a clear confirmation step before deleting an account, explaining that the action is permanent.
- FR-DEL-03. The system may require the user to re-enter their password or perform a similar confirmation step before deleting the account.
- FR-DEL-04. The system shall send a request to the backend service to delete or anonymise the current user's account and associated personalised data such as favourites and history.
- FR-DEL-05. On successful deletion, the system shall clear local session data, show a confirmation message, and navigate the user to the welcome or login screen.
- FR-DEL-06. If the delete operation cannot be completed due to service or network errors, the system shall show a simple, non-technical error message and shall not delete the account.

4.3 External Data & UX States

4.3.1 Carpark Data Fetching

4.3.1.1 Description and Priority

Description: The system shall fetch carpark information from an external carpark data service and prepare it for use in Nearby Carparks, Carpark Details, EV Features, and any other screens that show carpark information. The system may keep a copy of the latest successful response so that it can reuse it when needed.

Priority: High

4.3.1.2 Actors

- Authenticated end-user
- External carpark data service.

4.3.1.3 Preconditions

- User is logged in.
- Network connectivity is available.
- The system is configured with the endpoint and access details for the carpark data service.

4.3.1.4 Triggers

- Nearby Carparks needs carparks for an active destination.
- Carpark Details needs up-to-date data for a selected carpark.
- Live Availability Refresh requests new availability data.
- EV Features need EV-related attributes for carparks.

4.3.1.5 Stimulus/Response Sequence

1. A feature such as Nearby Carparks or Carpark Details requests carpark data for one or more locations.
2. System checks whether a recent carpark response is already available in memory or cache and whether it is still within an acceptable age.
3. If a valid cached response is available and allowed, the system returns the cached data to the requesting feature.
4. If no valid cached response is available or a fresh fetch is required, the system builds a request to the external carpark data service, including any needed parameters (for example, bounding area or carpark identifiers).
5. System sends the request to the carpark data service and waits for a response.
6. System receives the response and checks that it is in the expected format.
7. System extracts and maps relevant fields for each carpark, such as carpark identifier, name, address, coordinates, total lots, available lots, last updated time, tariffs, and any EV attributes.
8. System stores the processed carpark data in memory or cache with a timestamp for later use.
9. System returns the processed carpark data to the requesting feature (for example, Nearby Carparks or Carpark Details) so that it can be displayed to the user.

4.3.1.6 Alternate/Exception Flows

- A1 - Response format invalid → System discards the response, does not update cache, and treats this as a service failure for the calling feature.
- A2 - No carpark records in the response → System returns an empty list to the calling feature, which can show a “No carparks found” message.
- A3 - Service or network unavailable → System does not receive a valid response and treats this as a service failure for the calling feature.
- A4 - Cached data expired and new fetch fails → System reports a failure to the calling feature and does not show the expired data as if it were fresh.

4.3.1.7 Postconditions

- If the fetch is successful, processed carpark data is stored in memory or cache with a last updated timestamp.
- The requesting feature receives processed carpark data ready to use in its own display logic.
- If the fetch fails, the requesting feature is informed that no fresh data is available and may decide how to handle this.

4.3.1.8 Functional Requirements

- FR-FETCH-01. The system shall be able to send requests to an external carpark data service to retrieve carpark information.
- FR-FETCH-02. The system shall process responses from the carpark data service and extract relevant fields such as identifiers, names, addresses, coordinates, lot counts, last updated time, tariffs, and EV-related fields where available.
- FR-FETCH-03. The system shall provide processed carpark data to features that request it, such as Nearby Carparks, Carpark Details, and EV Features.
- FR-FETCH-04. The system shall store the latest successful carpark response in memory or cache with a timestamp so it can be reused when appropriate.
- FR-FETCH-05. The system shall be able to decide whether to use cached carpark data or request new data based on simple rules such as how old the cached data is.
- FR-FETCH-06. If the carpark data service or network is unavailable, or if the response is invalid, the system shall report a clear failure state back to the requesting feature and shall not crash.

4.3.2 API Failure & Stale Data Banner

4.3.2.1 Description and Priority

Description: The system shall show clear banners or messages when carpark data cannot be loaded or refreshed, and when the app is showing older cached data instead of fresh live data. This helps the user understand that live data may be unavailable or out of date, without using technical language.

Priority: High

4.3.2.2 Actors

- Authenticated end-user.

4.3.2.3 Preconditions

- User is logged in.
- A screen that depends on carpark data is active (for example, Nearby Carparks or Carpark Details).
- The system has attempted to fetch or refresh carpark data.

4.3.2.4 Triggers

- A carpark data fetch or refresh request fails or times out.
- The most recent cached carpark data is older than an acceptable freshness limit.
- Network connectivity is lost while the user is on a screen that shows carpark data.

4.3.2.5 Stimulus/Response Sequence

1. System attempts to fetch or refresh carpark data for a screen such as Nearby Carparks or Carpark Details.
2. System detects the result of the fetch, either success or failure.
3. If the fetch fails or times out, the system checks whether any cached carpark data is available.
4. If no cached data is available, the system shows a visible error banner or message such as “Unable to load carpark data. Please check your connection and try again later.” and shows an empty or placeholder state on the screen.
5. If cached data is available, the system shows the cached data and displays a banner or message such as “Live data is temporarily unavailable. Showing last updated at HH:MM.”
6. If the fetch succeeds but the data is older than the defined freshness limit, the system may still show the data and displays a banner or message such as “Information may be out of date. Last updated at HH:MM.”
7. System keeps the banner visible until the user dismisses it or until a later successful refresh that provides fresh data.
8. When a later fetch is successful and fresh data is loaded, the system updates the carpark data on screen, updates any “last updated” time, and removes or updates the banner to reflect that live data is available again.

4.3.2.6 Alternate/Exception Flows

- A1 - Initial load fails and no cached data exists → System shows an error banner, shows an empty or placeholder state, and may provide a simple Retry action.
- A2 - Fetch fails but cached data exists → System continues to show the cached data, displays a banner explaining that live data is unavailable, and does not treat cached data as fresh.
- A3 - Network connection lost while viewing carparks → System displays a banner such as "You are offline. Some information may be out of date." and continues to show any data already loaded.
- A4 - User dismisses the banner → System hides the banner but keeps the current data on screen; future errors or stale conditions may show a new banner.

4.3.2.7 Postconditions

- When a failure or stale condition occurs, the user sees a clear banner or message explaining that live data is unavailable or may be out of date.
- If cached data exists, it may still be shown, but it is not treated as fresh data.
- If no data exists, the user sees an empty or placeholder state instead of a broken or crashed screen.
- When fresh data is successfully loaded later, the banner is removed or updated and the screen shows the latest data.

4.3.2.8 Functional Requirements

- FR-STATUS-01. The system shall display a visible banner or message when a carpark data fetch or refresh fails.
- FR-STATUS-02. The system shall use simple, non-technical text in failure banners (for example, "Unable to load carpark data. Please check your connection and try again later.").
- FR-STATUS-03. If cached carpark data exists when a fetch fails, the system shall display the cached data and clearly indicate that it may be out of date (for example, by showing "Last updated at HH:MM").
- FR-STATUS-04. If no cached carpark data exists when a fetch fails, the system shall show an error message and an empty or placeholder state instead of leaving the screen blank or crashing.
- FR-STATUS-05. The system shall update or remove the error or stale-data banner once a later fetch succeeds and fresh data is loaded.

- FR-STATUS-06. If the device is offline while the user is viewing carparks, the system shall show a simple banner indicating that the user is offline and that some information may be out of date.

4.3.3 Location Permission Handling

4.3.3.1 Description and Priority

Description: The system shall handle device location permissions in a clear way whenever a feature needs the user's current location. If permission is denied or turned off, the system shall explain what happened and allow the user to continue by typing a destination instead.

Priority: High

4.3.3.2 Actors

- Authenticated end-user.

4.3.3.3 Preconditions

- User is logged in.
- The device supports location services.
- A feature that needs current location is being used.

4.3.3.4 Triggers

- User taps a "Use current location" option in the app.
- A screen that relies on current location is opened and tries to use location.

4.3.3.5 Stimulus/Response Sequence

1. System receives a request from a feature to use the user's current location.
2. System checks the current location permission state on the device (for example, not asked yet, allowed, denied, or blocked in settings).
3. If permission has not been asked before, the system shows the standard OS location permission prompt with a short explanation screen if needed.
4. If permission is granted, the system calls the device location service to read the current location.
5. If the location is successfully obtained, the system returns the coordinates to the requesting feature (for example, Search Destination) to use as the active destination.
6. If permission is denied or blocked, the system does not call the location service and instead shows a simple message explaining that location cannot be used and that the user can type a destination manually.

7. If location is turned off at the device level, the system shows a simple message explaining that location is off and may provide a link or hint to turn it on in device settings, while still letting the user continue with manual input.

4.3.3.6 Alternate/Exception Flows

- A1 - Permission not yet decided → System shows the OS permission prompt; if the user chooses Allow, processing continues as in step 4; if the user chooses Don't Allow, processing continues as in step 6.
- A2 - Permission already denied → System does not show the OS prompt again, shows a message such as "Location access is denied. Please turn it on in Settings or type your destination instead." and allows manual destination entry.
- A3 - Permission blocked in device settings → System shows a message such as "Location is blocked for this app. Please enable it in Settings if you want to use current location." and allows manual destination entry.
- A4 - Location services turned off → System shows a message such as "Location services are turned off on your device. Please turn them on or type your destination." and does not attempt to read location.
- A5 - Location request fails or times out → System shows "Unable to get your current location. Please try again or type your destination." and does not crash.

4.3.3.7 Postconditions

- If permission is granted and location is available, the requesting feature receives current location coordinates and can continue using them.
- If permission is denied, blocked, or location is unavailable, no location coordinates are returned, but the user is clearly informed and can continue with manual input.
- The app does not crash or get stuck when location permission is denied or location is turned off.

4.3.3.8 Functional Requirements

- FR-LOC-01. The system shall check the device's location permission state before trying to access the user's current location.
- FR-LOC-02. If location permission has not been asked before, the system shall trigger the standard OS location permission prompt when the user selects "Use current location".

- FR-LOC-03. If location permission is granted, the system shall request the current location from the device and return it to the feature that requested it.
- FR-LOC-04. If location permission is denied or blocked, the system shall not attempt to access location and shall show a simple message explaining that location cannot be used.
- FR-LOC-05. The system shall always allow the user to continue by typing a destination manually when location is denied, blocked, or unavailable.
- FR-LOC-06. If device location services are turned off, the system shall show a simple message and shall not repeatedly try to read the location.
- FR-LOC-07. If a location request fails or times out, the system shall show a non-technical error message and shall not crash.

4.3.4 Refresh Loading & Throttle

4.3.4.1 Description and Priority

Description: The system shall show clear loading indicators when a refresh is in progress and shall limit how often refresh actions can be triggered, so that the app does not spam the carpark data service or feel "stuck" to the user.
Priority: Medium–High

4.3.4.2 Actors

- Authenticated end-user.

4.3.4.3 Preconditions

- User is logged in.
- A screen that supports refresh is open (for example, Nearby Carparks, Carpark Details, or any screen that uses Live Availability Refresh).
- Network connectivity is available when a new refresh is attempted.

4.3.4.4 Triggers

- User performs a pull-to-refresh gesture on a supported screen.
- User taps a refresh button on a supported screen.
- An automatic refresh timer fires for a supported screen.

4.3.4.5 Stimulus/Response Sequence

1. System displays a screen that shows carpark data and, where applicable, the last updated time.
2. User triggers a refresh using pull-to-refresh, a refresh button, or an automatic refresh timer fires.

3. System checks whether a refresh is already in progress for that screen.
4. System checks how much time has passed since the last successful refresh and compares it to the minimum refresh interval.
5. If no refresh is in progress and the minimum interval has passed, the system starts a new refresh and shows a loading indicator (for example, a spinner or pull-to-refresh animation).
6. System temporarily disables further manual refresh actions for that screen while the refresh is in progress.
7. System sends a request to the carpark data service to retrieve updated information for the relevant carparks.
8. When a response is received, the system updates the carpark data on the screen and, if applicable, updates the last updated time.
9. System hides the loading indicator and re-enables the refresh control.

4.3.4.6 Alternate/Exception Flows

- A1 - Refresh triggered while another refresh is already in progress → System ignores the new refresh request or shows a short message such as "Still refreshing…" and keeps the current loading indicator.
- A2 - Refresh requested before minimum interval has passed → System does not call the service again and may show "Please wait a moment before refreshing again." while keeping the current data.
- A3 - Network or service unavailable during refresh → System stops the refresh, hides the loading indicator, displays "Unable to refresh. Please check your connection and try again later." and keeps the previous data on screen.
- A4 - Refresh request times out or returns an error → System stops the refresh, hides the loading indicator, displays a simple error message, and does not change the currently displayed data.

4.3.4.7 Postconditions

- If the refresh succeeds, the screen shows the latest available data and an updated last updated time.
- If the refresh fails, the previously displayed data and last updated time are preserved.
- The refresh control is left in a stable state (no stuck loading indicator), and the system is ready for a future refresh after the minimum interval.

4.3.4.8 Functional Requirements

- FR-THROT-01. The system shall show a visible loading indicator whenever a refresh is in progress on a supported screen.
- FR-THROT-02. The system shall prevent multiple refresh requests from being sent at the same time for the same screen.
- FR-THROT-03. The system shall enforce a minimum time interval between refreshes for the same screen and use this to decide whether to call the carpark data service again.
- FR-THROT-04. The system shall temporarily disable the refresh control while a refresh is in progress and re-enable it when the refresh finishes.
- FR-THROT-05. If a refresh request fails or times out, the system shall hide the loading indicator, keep the previous data, and show a simple, non-technical error message.
- FR-THROT-06. If a refresh is requested before the minimum interval has passed, the system shall not call the carpark data service again and shall not crash.

4.4 Open-In-Maps Handoff

4.4.1 Open in Maps

4.4.1.1 Description and Priority

Description: The system shall allow the user to open the selected carpark in the device's default maps application so that the user can start navigation to that carpark.

Priority: Medium

4.4.1.2 Actors

- Logged-in end-user

4.4.1.3 Preconditions

- User is logged in.
- A carpark has been selected and its details (including address and/or coordinates) are available.
- The device supports opening external apps via map links or intents.

4.4.1.4 Triggers

- User taps an Open in Maps or Navigate button on the Carpark Details screen.

- User taps a similar navigation action for a carpark from favourites or history, if provided.

4.4.1.5 Stimulus/Response Sequence

1. System displays the Carpark Details screen (or another screen) for a selected carpark, including an Open in Maps or Navigate button.
2. User taps the Open in Maps or Navigate button.
3. System prepares a map link or intent using the carpark's coordinates or full address.
4. System calls the device to open the preferred maps application with the prepared destination.
5. If the maps application opens successfully, the system hands control to that app and the user sees the carpark destination ready for routing.
6. If the maps application cannot be opened, the system does not crash and instead uses the copy-address fallback feature.

4.4.1.6 Alternate/Exception Flows

- A1 - No maps application available → System displays "No maps app is available on this device. You can copy the address instead." and calls the copy-address fallback.
- A2 - Invalid or missing address/coordinates → System displays "Navigation is not available for this carpark." and does not attempt to open the maps app.
- A3 - Device blocks external app launch → System displays "Unable to open your maps app. Please try again or copy the address." and offers the copy-address fallback.
- A4 - User cancels from the external maps app → System simply returns to ParkMate with the Carpark Details screen still visible; no extra action is needed.

4.4.1.7 Postconditions

- If successful, the device's maps application is opened with the carpark destination pre-filled, ready for the user to start navigation.
- If unsuccessful, the user is informed and can still access the carpark details and use the copy-address fallback.
- The ParkMate session remains active in the background while the user is in the maps application.

4.4.1.8 Functional Requirements

- FR-MAPS-01. The system shall provide an Open in Maps or Navigate control on the Carpark Details screen for each carpark.
- FR-MAPS-02. The system shall generate a map link or intent using the selected carpark's coordinates or full address when the user chooses to navigate.
- FR-MAPS-03. The system shall attempt to open the device's maps application with the generated destination link.
- FR-MAPS-04. If no suitable maps application is available or the app cannot be opened, the system shall show a simple error message and shall offer the copy-address fallback.
- FR-MAPS-05. If the carpark record does not contain valid address or coordinate information, the system shall not attempt to open the maps application and shall show a simple message that navigation is not available.
- FR-MAPS-06. The system shall not crash if the maps application fails to open or if the user returns from the maps application without starting navigation.

4.4.2 Copy Address Fallback

4.4.2.1 Description and Priority

Description: The system shall allow the user to copy a carpark's address to the device clipboard so that the user can paste it into another maps or navigation app if Open in Maps is not available or does not work.

Priority: Medium

4.4.2.2 Actors

- Logged-in end-user

4.4.2.3 Preconditions

- User is logged in.
- A carpark has been selected and its address is available.

4.4.2.4 Triggers

- User taps a Copy Address button or option on the Carpark Details screen.
- Open in Maps fails and the system falls back automatically to Copy Address.

4.4.2.5 Stimulus/Response Sequence

1. System displays the Carpark Details screen for a selected carpark, including the full address and a Copy Address option.
2. User taps the Copy Address option.

3. System copies the full carpark address text to the device clipboard.
4. System displays a short confirmation message such as “Address copied to clipboard.”
5. User can then switch to another app and paste the copied address as needed.

4.4.2.6 Alternate/Exceptions Flows

- A1 - Address missing or incomplete → System displays “Address is not available for this carpark.” and does not attempt to copy anything.
- A2 - Clipboard operation fails → System displays “Unable to copy address. Please try again.” and does not crash.
- A3 - Automatic fallback from Open in Maps → When Open in Maps fails, the system automatically copies the address to the clipboard where possible and shows “We could not open your maps app. The address has been copied so you can paste it into another app.”

4.4.2.7 Postconditions

- On success, the carpark’s address is stored in the device clipboard and ready to be pasted in another app.
- The user remains on the Carpark Details screen and can still use other actions.
- On failure, the clipboard contents are unchanged and the user is informed of the problem.

4.4.2.8 Functional Requirements

- FR-COPY-01. The system shall provide a Copy Address option on the Carpark Details screen for any carpark with a known address.
- FR-COPY-02. When the user selects Copy Address, the system shall copy the full address text of the selected carpark to the device clipboard.
- FR-COPY-03. After copying the address, the system shall show a short confirmation message indicating that the address has been copied.
- FR-COPY-04. If the carpark does not have a valid address, the system shall not attempt to copy and shall show a simple message that the address is not available.
- FR-COPY-05. If the clipboard operation fails, the system shall show a simple, non-technical error message and shall not crash.
- FR-COPY-06. When Open in Maps fails and a fallback is needed, the system shall attempt to use Copy Address and show a message explaining what happened.

4.5 Carpark Selection History

4.5.1 Capture Selection

4.5.1.1 Description and Priority

Description: The system shall automatically record a carpark selection in the user's history whenever the user opens a carpark's details or starts navigation to that carpark. This creates a list of recently used carparks for the user.

Priority: Medium

4.5.1.2 Actors

- Logged-in end-user

4.5.1.3 Preconditions

- User is logged in.
- A valid carpark has been selected in the app.

4.5.1.4 Triggers

- User taps a carpark in the Nearby Carparks list.
- User taps a carpark in the Favourites list.
- User taps a carpark in any other supported list and opens its details or navigation.

4.5.1.5 Stimulus/Response Sequence

1. System displays a screen that allows the user to select a carpark (for example, Nearby Carparks or Favourites).
2. User taps a carpark to view details or start navigation.
3. System identifies the selected carpark and reads its unique identifier and basic information (such as name and address).
4. System checks the user's existing history list for the most recent entry.
5. If the most recent entry in the history list is the same carpark, the system updates the timestamp of that entry instead of creating a new entry.
6. If the most recent entry is different, the system creates a new history entry for the current user containing the carpark identifier, basic display information, and the current timestamp.
7. System applies any history size rules (for example, keeping only the latest 20 entries) by removing the oldest entry if needed.

8. System then continues with the normal flow (for example, opens the Carpark Details screen or calls Open in Maps).

4.5.1.6 Alternate/Exception Flows

- A1 - User not logged in → System does not record the selection in a server-side per-user history and may keep only basic in-memory behaviour for the current session, if any.
- A2 - History store not available or write fails → System does not crash, continues with the normal action (for example, opens Carpark Details), and simply does not update the history for that selection.
- A3 - Carpark identifier missing or invalid → System skips creating a history entry for that selection and continues with the normal flow.

4.5.1.7 Postconditions

- If recording succeeds, the user's history list is updated with the selected carpark and the latest timestamp, with consecutive duplicates avoided.
- The history list respects the size rule.
- If recording fails or is skipped, the rest of the app behaviour (such as viewing details or navigation) still works as normal.

4.5.1.8 Functional Requirements

- FR-HIST-01. The system shall automatically attempt to create a history entry whenever a user selects a carpark to view details or start navigation.
- FR-HIST-02. Each history entry shall include the carpark identifier, basic display information (such as name and address), and a timestamp of when it was selected.
- FR-HIST-03. The system shall store history entries separately for each logged-in user.
- FR-HIST-04. If the most recent history entry is for the same carpark, the system shall update that entry's timestamp instead of adding a new entry.
- FR-HIST-05. The system shall keep only the most recent N entries (for example, 20) in the user's history by removing the oldest entries when the limit is exceeded.
- FR-HIST-06. If the history store is unavailable or a write fails, the system shall not crash and shall still let the user continue viewing details or using navigation.

4.5.2 View History

4.5.2.1 Description and Priority

Description: The system shall allow the user to view a list of their most recent carpark selections, ordered from most recent to oldest, and select any entry to reopen that carpark in the app.

Priority: Medium

4.5.2.2 Actors

- Logged-in end-user

4.5.2.3 Preconditions

- User is logged in.
- History capture is enabled and may already have stored entries for the current user.
- Network connectivity is available if the history is stored on a backend service.

4.5.2.4 Triggers

- User taps a History option from the app navigation, menu, or profile/settings screen.

4.5.2.5 Stimulus/Response Sequence

1. System displays a screen or menu that includes a History option.
2. User taps the History option.
3. System requests the history list for the current user (for example, the latest 20 entries) from local storage or a backend service.
4. System orders the history entries from most recent to oldest based on their timestamps.
5. If one or more entries exist, the system displays them in a list showing basic information such as carpark name, address or area, and the time or date of last selection.
6. User scrolls the history list and reviews past carpark selections.
7. User taps a carpark entry in the history list.
8. System uses the carpark identifier from the selected history entry to open the Carpark Details screen (or to follow the normal selection flow for that carpark).

4.5.2.6 Alternate/Exception Flows

- A1 - No history entries → System displays “You have no recent carpark history yet.” and may show a hint such as “Your recent carparks will appear here.”
- A2 - History retrieval fails (for example, backend error or network issue) → System displays “Unable to load your history. Please try again later.” and does not crash.
- A3 - History entry refers to a carpark that no longer exists or cannot be loaded → System displays “This carpark is no longer available.” and keeps the user on the History screen.

4.5.2.7 Postconditions

- If retrieval succeeds, the user can see their recent carpark selections in order from most recent to oldest.
- If the user selects an entry, that carpark becomes the current selection and the normal Carpark Details flow is started.
- If retrieval fails or there are no entries, the user is informed and the app remains stable.

4.5.2.8 Functional Requirements

- FR-HIST-01. The system shall provide a History option that shows the current user’s recent carpark selections when tapped.
- FR-HIST-02. The system shall retrieve up to the latest N entries (for example, 20) from the user’s history and display them in order from most recent to oldest.
- FR-HIST-03. Each history list item shall display basic information such as the carpark name and the time or date it was last selected.
- FR-HIST-04. When the user taps a history entry, the system shall use that entry’s carpark identifier to open the Carpark Details screen or equivalent selection flow.
- FR-HIST-05. If the history cannot be retrieved due to service or network issues, the system shall display a simple, non-technical error message and shall not crash.
- FR-HIST-06. If no history entries exist, the system shall show a clear “no history yet” message rather than an empty, unexplained screen.

4.5.3 Limit & Clear History

4.5.3.1 Description and Priority

Description: The system shall automatically limit the number of history entries stored per user and shall allow the user to clear all carpark history entries in one action.

Priority: Medium

4.5.3.2 Actors

- Logged-in end-user

4.5.3.3 Preconditions

- User is logged in.
- History capture is enabled.
- At least one history entry may already exist for the current user.

4.5.3.4 Triggers

- A new history entry is added for the user.
- User taps a Clear History option from the History screen.

4.5.3.5 Stimulus/Response Sequence

1. System adds or updates a history entry when the user selects a carpark.
2. System checks how many history entries exist for the current user after this update.
3. If the number of entries is greater than the maximum allowed, the system removes the oldest entries until only the latest allowed number remain.
4. System keeps the remaining entries in order from most recent to oldest.
5. System displays the History screen showing the current entries and a Clear History option.
6. User taps the Clear History option.
7. System shows a confirmation prompt such as “Clear all history? This cannot be undone.” with Confirm and Cancel options.
8. If the user confirms, the system deletes all history entries for the current user.
9. System updates the History screen to show that there are no entries and may display a message such as “You have no recent carpark history yet.”

4.5.3.6 Alternate/Exception Flows

- A1 - User cancels at confirmation → System closes the confirmation prompt and leaves the history list unchanged.
- A2 - No history entries to clear → System may disable the Clear History option or show “There is no history to clear.” and keep the screen as is.

- A3 - History deletion fails (for example, backend error or storage problem) → System displays “Unable to clear history. Please try again later.” and leaves the existing history entries in place.

4.5.3.7 Postconditions

- The user’s history list never exceeds the defined maximum number of entries; oldest entries are removed when the limit is exceeded.
- If the user confirms Clear History and the operation succeeds, the history list for that user becomes empty.
- If the user cancels or an error occurs, existing history entries remain unchanged.

4.5.3.8 Functional Requirements

- FR-HIST-01. The system shall enforce a maximum number of history entries per user (for example, 20) and shall remove the oldest entries when this limit is exceeded.
- FR-HIST-02. The system shall provide a Clear History option on the History screen.
- FR-HIST-03. The system shall show a confirmation prompt before clearing all history entries for a user.
- FR-HIST-04. When the user confirms the clear action, the system shall delete all history entries for that user and update the History screen to show that no entries remain.
- FR-HIST-05. If the clear action cannot be completed due to a service or storage error, the system shall show a simple, non-technical error message and shall not delete any entries.

4.6 EV Features

4.6.1 EV Mode Toggle

4.6.1.1 Description and Priority

Description: The system shall provide an EV Mode toggle that lets the user switch the app into an EV-focused view. When EV Mode is on, the app prioritises carparks with EV chargers so that EV drivers can find suitable carparks more easily.

Priority: Medium–High

4.6.1.2 Actors

- Logged-in end-user

4.6.1.3 Preconditions

- User is logged in.
- Nearby carparks for an active destination are available.
- Carpark data includes information about EV chargers for at least some carparks.

4.6.1.4 Triggers

- User taps an EV Mode toggle on the Nearby Carparks screen or a related settings area.

4.6.1.5 Stimulus/Response Sequence

1. System displays the Nearby Carparks screen and shows an EV Mode toggle control.
2. User taps the EV Mode toggle to turn EV Mode on.
3. System marks EV Mode as enabled for the current session (and may store this as a user preference if supported).
4. System updates the active filters and sorting rules to favour carparks with EV chargers (for example, by showing only EV-capable carparks or moving them to the top of the list).
5. System refreshes the nearby carpark list using the updated EV-focused filters and sorting and displays the updated results.
6. System shows a clear visual indication that EV Mode is on (for example, a highlighted toggle or EV icon).
7. At a later time, user taps the EV Mode toggle again to turn EV Mode off.
8. System marks EV Mode as disabled, resets filters and sorting to the normal non-EV-focused behaviour, and refreshes the nearby carpark list to show all matching carparks according to the user's other filters.

4.6.1.6 Alternate/Exception Flows

- A1 - EV data not available for this area → System displays "EV information is not available for this area.", does not change the current list, and may disable the EV Mode toggle.
- A2 - Service or network unavailable when applying EV Mode → System displays "Unable to update for EV Mode. Please check your connection and try again later." and keeps the previous list and mode state.

- A3 - User not logged in (if EV preference is normally stored per user) → System may still allow EV Mode for the current session but does not store it as a lasting preference, and may show "Log in to save EV Mode as your default."

4.6.1.7 Postconditions

- When EV Mode is on and data is available, the nearby carpark list is filtered or ordered in an EV-focused way.
- When EV Mode is off, the nearby carpark list is shown in the normal way, subject only to standard filters and sorting.
- The current EV Mode setting is known to the system for the duration of the session (and may be stored as a preference if supported).

4.6.1.8 Functional Requirements

- FR-EV-01. The system shall provide an EV Mode toggle on the Nearby Carparks screen or an equivalent main screen.
- FR-EV-02. When EV Mode is turned on, the system shall adjust the nearby carpark results to favour carparks with EV chargers (for example, by filtering out non-EV carparks or ranking EV-capable carparks higher).
- FR-EV-03. When EV Mode is turned off, the system shall remove any EV-specific constraints and show nearby carparks according to the user's standard filters and sorting.
- FR-EV-04. The system shall show a clear on-screen indication when EV Mode is enabled.
- FR-EV-05. The system shall maintain the EV Mode setting for at least the current app session and may store it as a user preference if supported.
- FR-EV-06. If EV-related data is not available, the system shall not attempt to apply EV Mode and shall show a simple, non-technical message instead of crashing.
- FR-EV-07. If the service or network is unavailable while updating EV Mode results, the system shall show a simple error message and keep the previous carpark list.

4.6.2 Filter by Connector & Speed

4.6.2.1 Description and Priority

Description: The system shall allow the user to filter carparks with EV chargers by connector type and charging speed so that EV drivers can find carparks that match their vehicle's charging requirements.

Priority: High

4.6.2.2 Actors

- Logged-in end-user.

4.6.2.3 Preconditions

- User is logged in.
- Carpark data includes EV charger information such as connector types and charging speeds.
- Nearby carparks for an active destination are available.

4.6.2.4 Triggers

- User opens the Filters panel from the Nearby Carparks screen while EV Mode is on, or while EV-related filters are supported.
- User taps a specific EV Filters or Connector/Speed Filters option if shown separately.

4.6.2.5 Stimulus/Response Sequence

1. System displays the Nearby Carparks screen and allows the user to open the Filters panel.
2. User opens the Filters panel and scrolls to the EV-related section.
3. System shows one or more connector type options (for example, Type 2, CCS, CHAdeMO) and charging speed groups (for example, slow, fast, rapid), based on what the data source supports.
4. User selects one or more connector types that match their vehicle.
5. User selects one or more charging speed groups, if desired.
6. User taps Apply or a similar button to confirm the chosen EV filters.
7. System applies the selected connector and speed filters to the carpark dataset, keeping any non-EV filters (for example, distance) that are already active.
8. System updates the Nearby Carparks list to show only carparks that have at least one EV charger matching the selected connector types and speeds.
9. System indicates that EV connector and speed filters are active (for example, by showing a small summary chip or badge).

4.6.2.6 Alternate/Exception Flows

- A1 - No EV data available for this area → System displays "EV charger information is not available for this area." and disables connector and speed filters for that view.
- A2 - No carparks match the selected connector types and speeds → System displays "No carparks match these EV filters. Please adjust your filters and try again." and allows the user to change or clear EV filters.
- A3 - User cancels EV filter changes → User closes the Filters panel or taps Cancel; system keeps the previous connector and speed filter settings unchanged.
- A4 - Service or network unavailable when reloading results → System displays "Unable to update EV results. Please check your connection and try again later." and keeps the last shown list and EV filters.

4.6.2.7 Postconditions

- If EV data is available, the user's selected connector and speed filters are stored for the current session and applied to the Nearby Carparks results.
- The Nearby Carparks list shows only carparks that satisfy the chosen connector and speed filters, together with any other active filters.
- If EV data is not available, no EV-specific filtering is applied, and the user is informed.

4.6.2.8 Functional Requirements

- FR-EV-08. The system shall provide connector type and charging speed filter options in the Filters panel when EV charger data is available.
- FR-EV-09. The system shall allow the user to select one or more connector types and one or more charging speed groups for filtering.
- FR-EV-10. When EV connector and speed filters are applied, the system shall show only carparks that have at least one EV charger matching the selected connector types and speeds.
- FR-EV-11. The system shall keep other active filters (for example, distance or availability) when applying EV connector and speed filters.
- FR-EV-12. If no carparks match the selected EV connector and speed filters, the system shall show a clear "no matches" message and shall not crash.
- FR-EV-13. If EV charger data is not available, the system shall not allow EV connector and speed filtering and shall show a simple message instead.

- FR-EV-14. If the service or network is unavailable when updating EV-filtered results, the system shall show a simple, non-technical error message and keep the previous carpark list.

4.6.3 EV Sorting

4.6.3.1 Description and Priority

Description: The system shall allow the user to sort EV-capable carparks using EV-related criteria, such as the number of available EV chargers and distance from the destination, so that EV drivers can quickly find the most suitable options.

Priority: Medium–High

4.6.3.2 Actors

- Logged-in end-user.

4.6.3.3 Preconditions

- User is logged in.
- Nearby carparks for an active destination are available.
- Carpark data includes EV charger information such as number of EV chargers and their availability.
- EV Mode and/or EV filters may already be active.

4.6.3.4 Triggers

- User taps a Sort control on the Nearby Carparks screen while EV data is available.

4.6.3.5 Stimulus/Response Sequence

1. System displays the Nearby Carparks list, which may already be filtered for EV use.
2. User taps the Sort control on the Nearby Carparks screen.
3. System shows sorting options that include EV-related choices such as “Most EV chargers first” and “EV nearest first”, alongside existing non-EV options if any.
4. User selects an EV sorting option, such as sorting by number of available EV chargers and then by distance.
5. System applies the chosen EV sorting rule to the current carpark list, keeping any active filters (including EV filters) in place.

6. System reorders the list to match the selected EV sorting rule and updates the display.
7. System indicates which sort option is currently active (for example, by highlighting it or showing a label above the list).

4.6.3.6 Alternate/Exception Flows

- A1 - EV data missing for some carparks → System still sorts using EV data where available and may place carparks with unknown EV data lower in the list or mark them as "EV info unknown".
- A2 - EV data not available for this area → System hides or disables EV-specific sort options and may show "EV sorting is not available for this area." while still allowing standard sorting.
- A3 - User cancels sort selection → User closes the sort dialog or panel without choosing a new option; system keeps the previous sort order.
- A4 - Service or network unavailable when reloading sorted results (if a refresh is triggered) → System displays "Unable to update EV sort. Please check your connection and try again later." and keeps the last shown list and previous sort order.

4.6.3.7 Postconditions

- The Nearby Carparks list is ordered according to the selected EV sorting rule, if EV data is available.
- Any active filters, including EV filters, remain in effect after sorting.
- If EV sorting cannot be applied, the previous sort order remains and the user is informed if needed.

4.6.3.8 Functional Requirements

- FR-EV-15. The system shall provide EV-specific sorting options on the Nearby Carparks screen when EV charger data is available.
- FR-EV-16. The system shall support at least one EV sort option that prioritises carparks by the number of available EV chargers, and may use distance as a secondary criterion.
- FR-EV-17. When an EV sorting option is selected, the system shall reorder the carpark list according to the chosen EV criteria while keeping any active filters applied.
- FR-EV-18. The system shall show which sort option is currently active so that the user understands how the list is ordered.

- FR-EV-19. If EV charger data is not available, the system shall not offer EV-specific sorting options and shall fall back to standard sorting only.
- FR-EV-20. If a sorting update depends on a data refresh and that refresh fails due to service or network issues, the system shall show a simple, non-technical error message and keep the previous list and sort order.

4.6.4 EV Details in Carpark View

4.6.4.1 Description and Priority

Description: The system shall show EV-specific information for a carpark that has EV chargers when the user opens the Carpark Details screen. This includes connector types, number of EV chargers, basic availability, and charging speed bands where available.

Priority: Medium

4.6.4.2 Actors

- Logged-in end-user.

4.6.4.3 Preconditions

- User is logged in.
- The user has opened the Carpark Details screen for a selected carpark.
- Carpark data for that carpark includes EV-related fields if it is EV-capable.

4.6.4.4 Triggers

- User taps a carpark that has EV chargers from the Nearby Carparks list.
- User opens a favourited or history carpark that has EV chargers.

4.6.4.5 Stimulus/Response Sequence

1. System receives the selected carpark identifier and opens the Carpark Details screen.
2. System loads general carpark information such as name, address, distance, and general availability.
3. System checks whether the carpark has EV-related data (for example, any EV connectors defined).
4. If EV data is present, the system retrieves EV-related fields such as connector types, number of EV charging points, basic EV availability, and charging speed categories.

5. System displays an EV section within the Carpark Details screen, showing connector types, charger count, and, where available, a simple view of EV availability and speed groups.
6. User reviews the EV details to decide whether the carpark is suitable for their vehicle.
7. User may use other actions (for example, favourite or Open in Maps) as normal from the same Carpark Details screen.

4.6.4.6 Alternate/Exception Flows

- A1 - Carpark has no EV chargers → System does not show an EV section and the Carpark Details screen only displays general carpark information.
- A2 - EV data is incomplete (for example, connector types known but speeds missing) → System shows the EV details that are available and may show simple text such as "Speed information not available" for missing parts.
- A3 - EV data cannot be loaded due to service or network issues → System hides or disables the EV section and displays a simple message such as "EV information is not available right now."

4.6.4.7 Postconditions

- If EV data exists and can be loaded, the Carpark Details screen shows an EV section with the available EV-specific details.
- If EV data does not exist or cannot be loaded, the user still sees the general carpark details and the app remains stable.

4.6.4.8 Functional Requirements

- FR-EV-21. The system shall check whether a selected carpark has EV-related data when opening the Carpark Details screen.
- FR-EV-22. For carparks with EV data, the system shall display an EV section showing available fields such as connector types and number of EV chargers.
- FR-EV-23. Where provided by the data source, the system shall show simple charging speed categories (for example, slow, fast, rapid) in the EV section.
- FR-EV-24. If a carpark has no EV data, the system shall not show an EV section and shall not label the carpark as EV-capable.
- FR-EV-25. If some EV fields are missing, the system shall show the available fields and use simple text to indicate that some information is not available.

- FR-EV-26. If EV data cannot be loaded due to a service or network error, the system shall show a simple, non-technical message and shall not crash; general carpark details remain visible.

5. Non-Functional Requirements

For ParkMate, a mobile app that helps drivers find suitable carparks (with live availability, filters, and favourites), we define the following non-functional requirements. These set expectations for quality, performance, and usability across typical use cases and peak periods. Here are some key non-functional requirements:

5.1 Efficiency Requirements

- REQ-EFF-1: First search results (map + list) appear within 3 seconds on a typical 4G connection and mid-range phone.
- REQ-EFF-2: Changes to filters or sort update visible results within 1 second.
- REQ-EFF-3: Panning/zooming the map feels smooth; new markers load without obvious stutter or long pauses.
- REQ-EFF-4: Auto-refresh of live data runs quietly in the background and does not interrupt user actions (typing, scrolling, tapping).

5.2 Robustness Requirements

- REQ-ROB-1: If live data cannot be fetched, ParkMate shows the last known values with a clear “live data unavailable” banner and a timestamp.
- REQ-ROB-2: Missing or odd data (e.g., no height limit) does not crash the app; the UI shows “info unavailable” where needed.
- REQ-ROB-3: Temporary failures (e.g., network hiccups) trigger automatic, limited retries; ParkMate returns to normal once data is back.
- REQ-ROB-4: If the user leaves and returns during a refresh, ParkMate resumes safely and shows a clear state (loading, updated, or failed).

5.3 Maintainability Requirements

- REQ-MAI-1: Features are grouped by area (Search, Carparks, Profile) so a change in one area doesn’t ripple through the app.
- REQ-MAI-2: Common rules (e.g., validation messages) are kept in one place so updates happen once and appear everywhere.
- REQ-MAI-3: The app produces clear, human-readable logs and error messages to help developers find issues quickly.
- REQ-MAI-4: A lightweight setup guide (how to run, where data comes from, key toggles) is kept up to date for new contributors.

5.4 Reliability Requirements

- REQ-REL-1: Core features (search, view carparks, view details) target 99% availability during normal hours.
- REQ-REL-2: ParkMate shows the “last updated” time on live values so users always know how fresh the data is.

- REQ-REL-3: The app remembers the current selection and filters after a brief app switch or restart.
- REQ-REL-4: ParkMate avoids duplicated actions (e.g., double-taps don't create duplicate favourites or history items).

5.5 Security Requirements

- REQ-SEC-1: All communication uses secure connections (HTTPS); passwords are stored securely (industry-standard hashing).
- REQ-SEC-2: Error messages about login or accounts are generic (no hints about which field is wrong) to avoid information leaks.
- REQ-SEC-3: Sensitive fields are never sent to the client (e.g., password hashes are stripped from responses).
- REQ-SEC-4: Accounts auto-log out after 30 minutes of inactivity on shared devices; users can log out manually at any time.

5.6 Software Quality Attributes

1. Scalability
    a. Handles growth in users and carpark data without slowing basic tasks (search, list, details).
    b. Data refresh strategies adapt so the app stays responsive during heavier usage.
2. Reliability
    a. Graceful fallback to last-known data with timestamps.
    b. Clear status and recovery behavior (loading, updated, failed).
3. Usability
    a. Simple, consistent patterns (search → view → details → favourite).
    b. Messages are plain English, guiding users on what happened and what to do next.
4. Compatibility
    a. Works across common Android/iOS devices and screen sizes; opens navigation via the user's default maps app.
5. Data Integrity
    a. Displays source attribution and last updated time for live data.
    b. Avoids partial or misleading values (labels unavailable fields clearly).
6. Maintainability
    a. Modular structure by feature area; shared rules in one place.
    b. Concise docs for setup, data sources, and feature flags.

5.7 Business Rules

- BR-1 (Access): Unauthenticated users may search and view carparks and details; login is required to favourite carparks and sync history.
- BR-2 (Data Use): Availability and prices are informational; ParkMate shows the data source and last-updated time and does not guarantee real-time accuracy.

- BR-3 (Sorting & Filters): When sorting produces ties, distance breaks the tie; if a filter yields no matches, ParkMate explains and shows how to clear filters.
- BR-4 (Region & Coverage): The app operates within the supported region(s) and only displays carparks from approved providers.
- BR-5 (External Navigation): “Open in Maps” hands off to the device’s maps app; ParkMate itself does not provide turn-by-turn navigation.
- BR-6 (Account Removal): Deleting an account removes favourites and profile preferences but does not affect provider-owned source data.

Appendix A : Data Dictionary

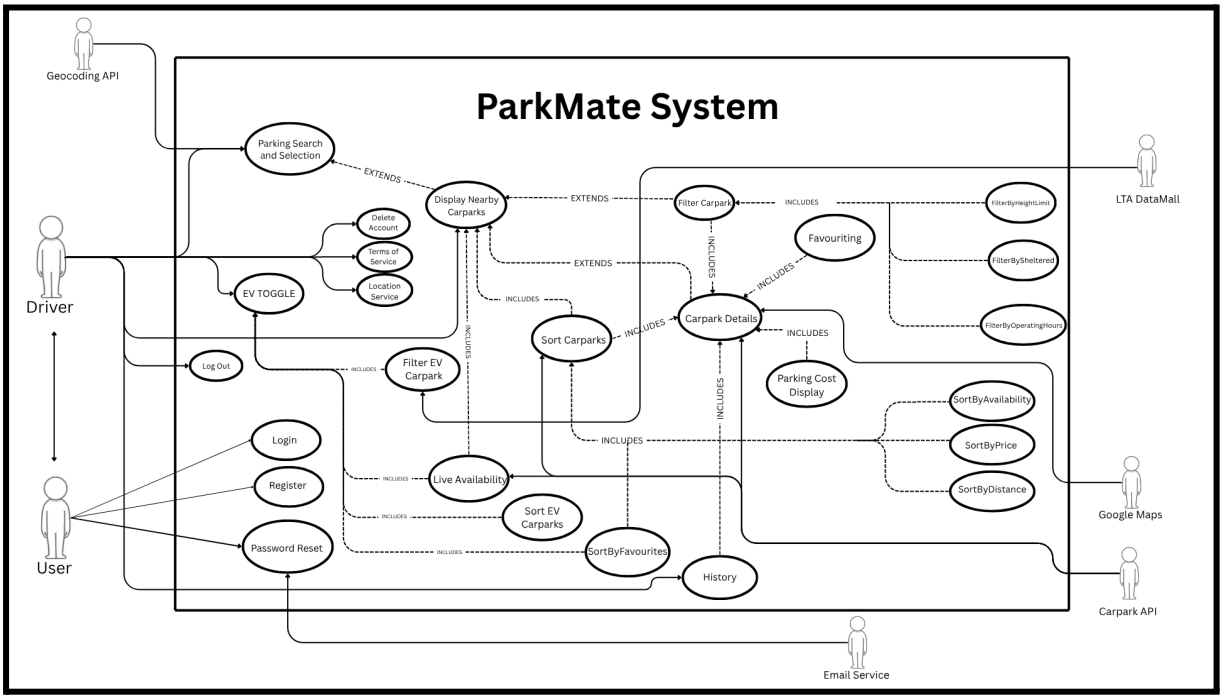
Term	Definition
Account	A registered user profile in the ParkMate system that stores authentication credentials and user preferences.
API (Application Programming Interface)	An external service that ParkMate integrates with to retrieve data such as carpark information, geocoding, and availability.
API Timeout	A failure condition that occurs when an API does not respond within the expected timeframe.
Availability	The current number of parking lots available at a specific carpark at a given time.
Auto-refresh	An automated system process that updates carpark availability data at regular intervals (every 60 seconds).
Cached Data	Previously retrieved information stored locally on the device to display.
Carpark	A parking facility where vehicles can be parked, with associated attributes like location, pricing, and availability.
Carpark API	An external service that provides real-time information about carpark details and availability.
Carpark Details	Comprehensive information about a specific carpark including name, distance, availability, price, type, and height limit.
Carpark Type	The classification of a carpark (e.g.,open, sheltered).
Charging Speed	The rate at which an EV charger can charge a vehicle.
Connector Type	The specific type of charging plug used at an EV charging station (e.g., Type 2, CCS2, CHAdeMO).
Credentials	The email and password combination used by a user to authenticate and access their account.
Current Location	The user's present geographic position, obtained through device location services.
Destination	A location selected by the user as the reference point for searching nearby carparks.

Distance	The measured proximity between a carpark and either the user's current location or selected destination.
EV (Electric Vehicle)	A vehicle powered by electricity rather than conventional fuel.
EV Charging Station	A facility within or near a carpark that provides charging infrastructure for electric vehicles.
EV Mode	A specialized interface theme and feature set designed specifically for electric vehicle drivers, with green UI color scheme.
Favourite	A carpark that a user has marked for quick access and reference in future sessions.
Filter	A user-applied criteria that limits displayed carparks to only those matching specific attributes.
Geocoding API	An external service that converts address text into geographic coordinates and provides location suggestions.
Height Limit	The maximum vehicle height (in meters) that can enter a carpark facility.
History	A chronological record of carparks that a user has previously selected or viewed.
Hourly Rate	The parking fee charged per hour at a carpark.
Live Availability	Real-time data showing the current number of available parking slots at a carpark.
Location Permission	Device settings that allow or deny ParkMate access to the user's current geographic position.
Login	The process of authenticating a user's credentials to grant access to the ParkMate system.
Logout	The process of terminating a user's active session and returning to the login screen.
Map View	A display format showing carparks as markers on an interactive map interface.
Operating Hours	The time periods during which a carpark is open and available for use.
Password	A secret authentication credential used with email to access a user account.
Password Hash	An encrypted version of a user's password stored securely in the

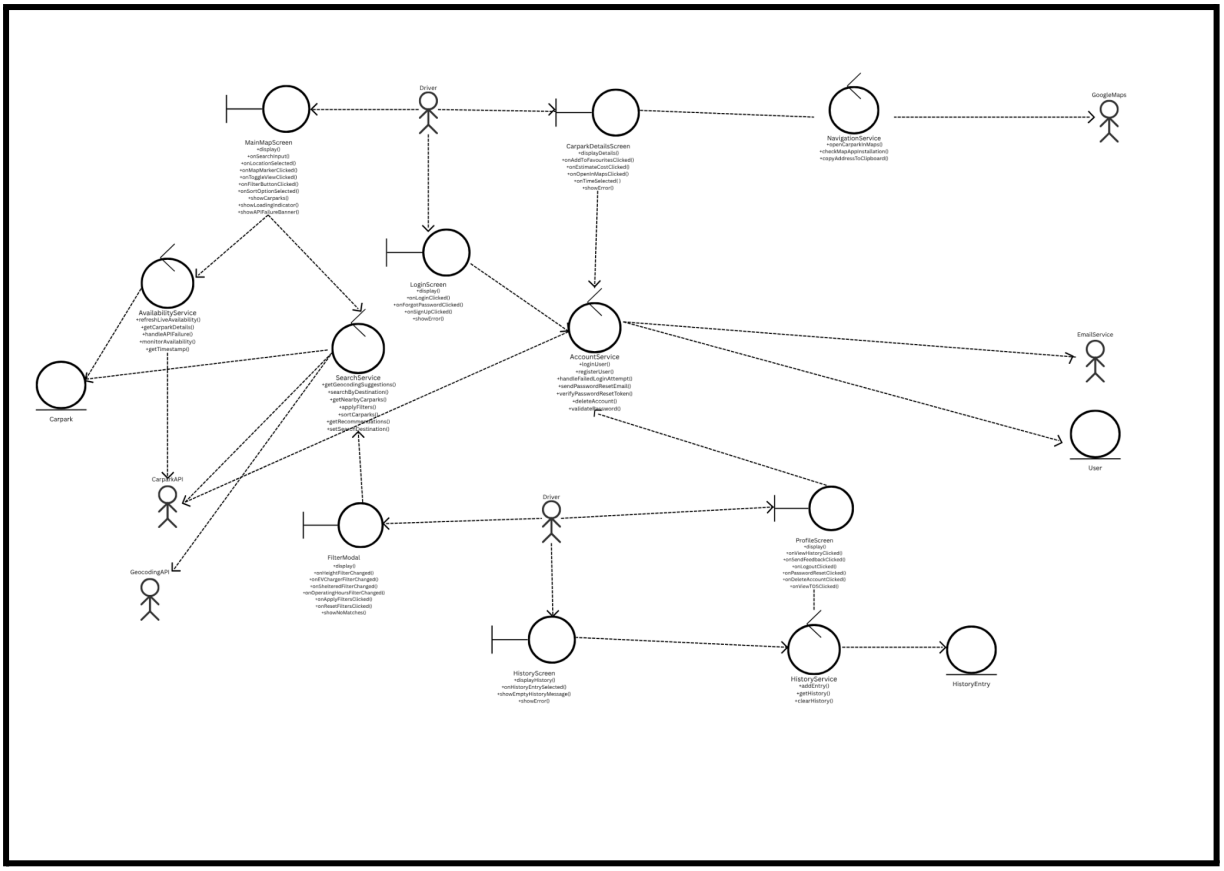
	database.
Price	The cost of parking at a carpark, which may vary based on duration and time of day.
Session	An authenticated period during which a user remains logged into the ParkMate system.
Sheltered Parking	Covered parking facilities that protect vehicles from weather elements.
Sort	A function that reorders displayed carparks based on selected criteria (Distance, Price, Availability, Favourites, Nearby).
Tariff	The pricing structure and rules for parking fees at a specific carpark.
Timestamp	A date and time marker indicating when an event occurred or data was last updated.
Toggle	A UI control that switches between two states (e.g., map view/list view, EV mode on/off).
User	A person who interacts with the ParkMate system, typically a driver seeking parking.

Appendix B: Analysis Models

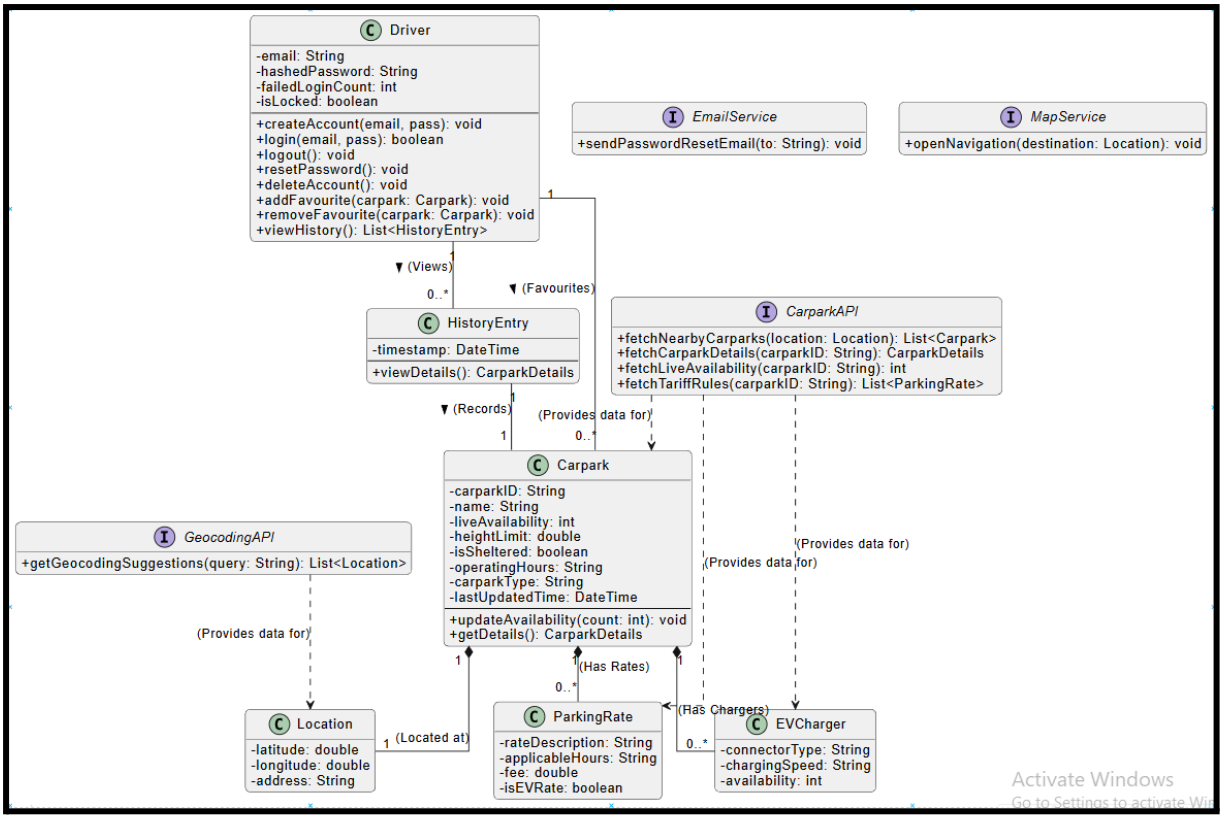
Please refer to our GitHub repository for a clearer view of the diagrams.



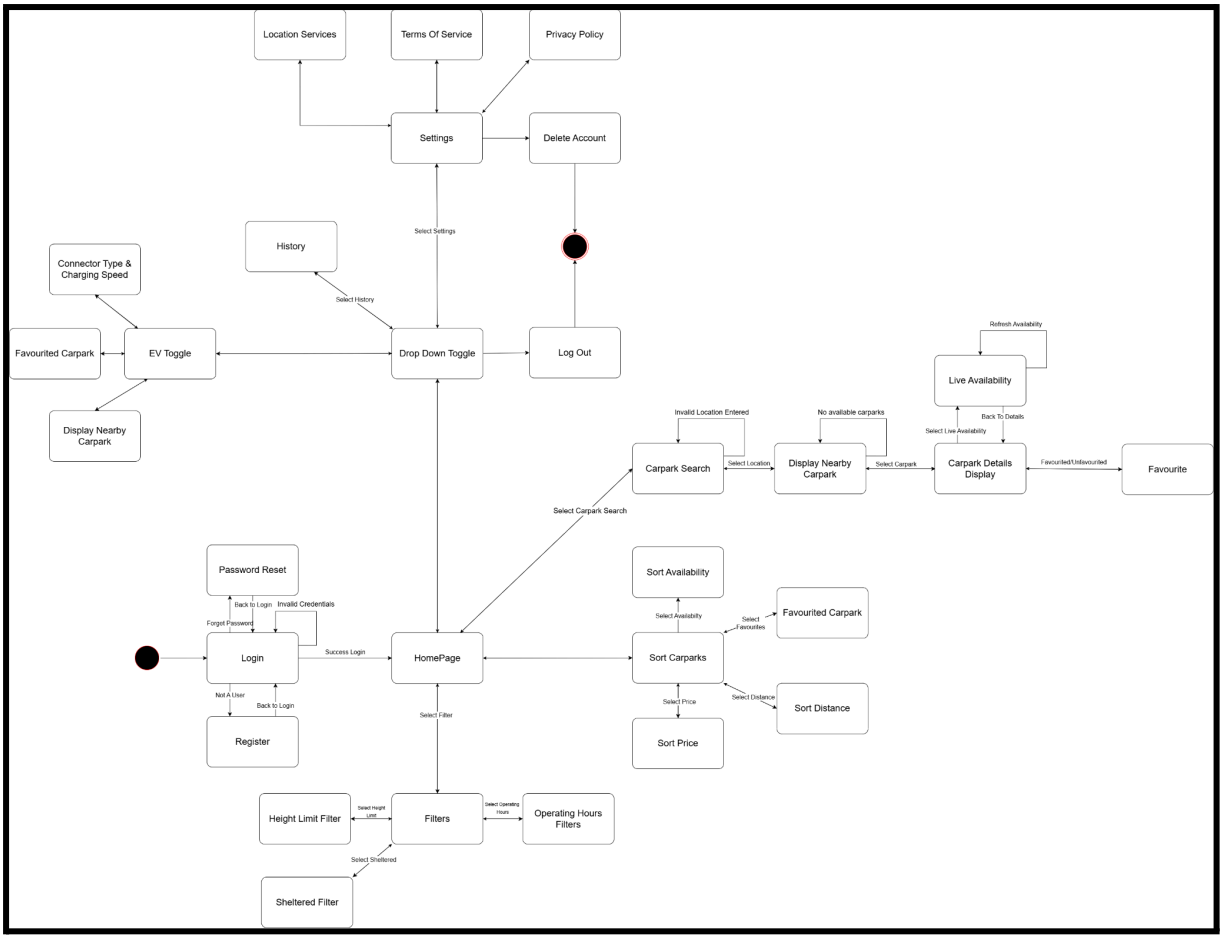
Use Case Diagram



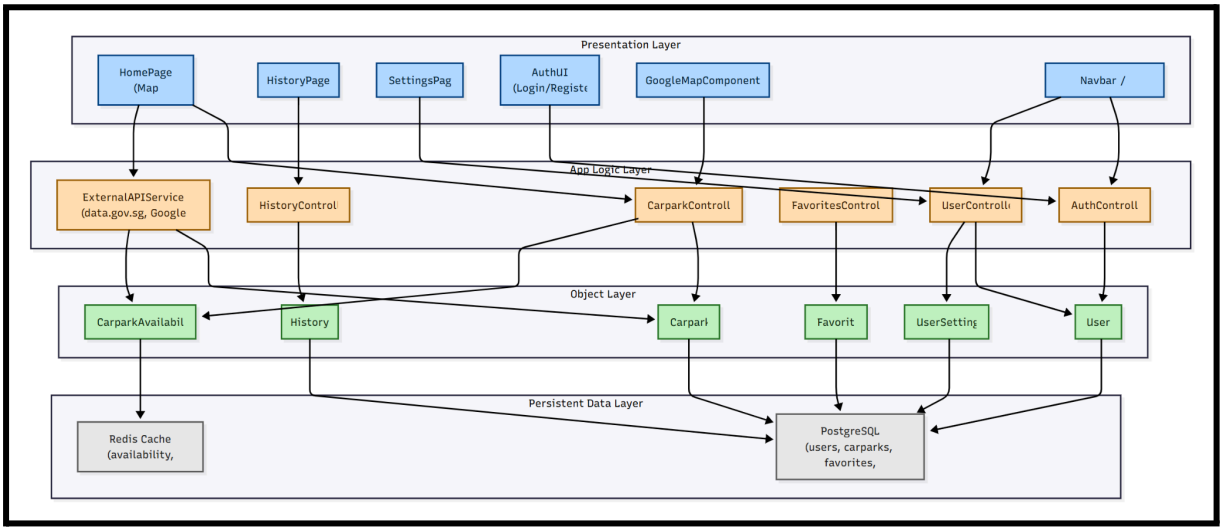
Controller Boundary Diagram



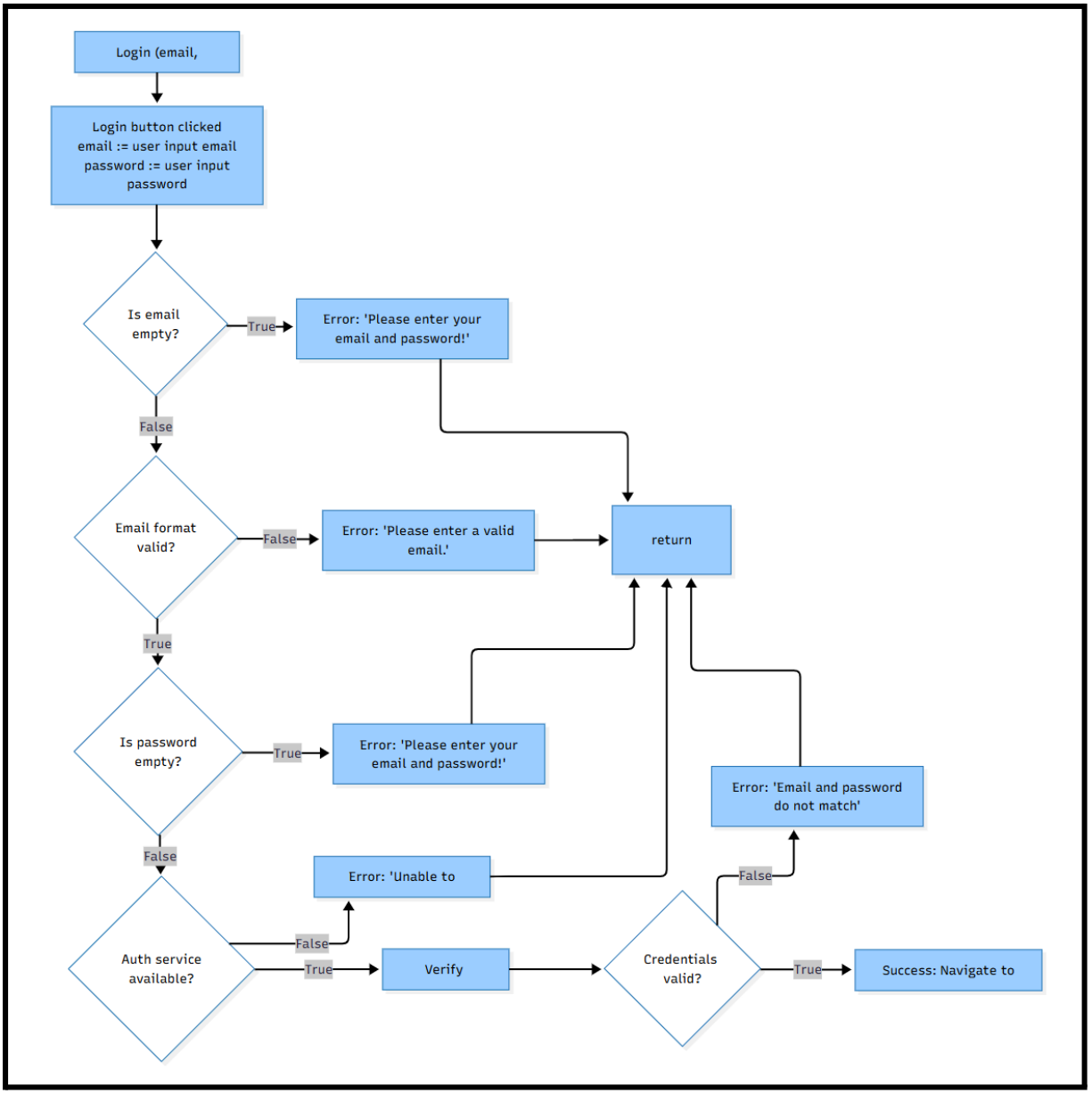
Entity Class Diagram



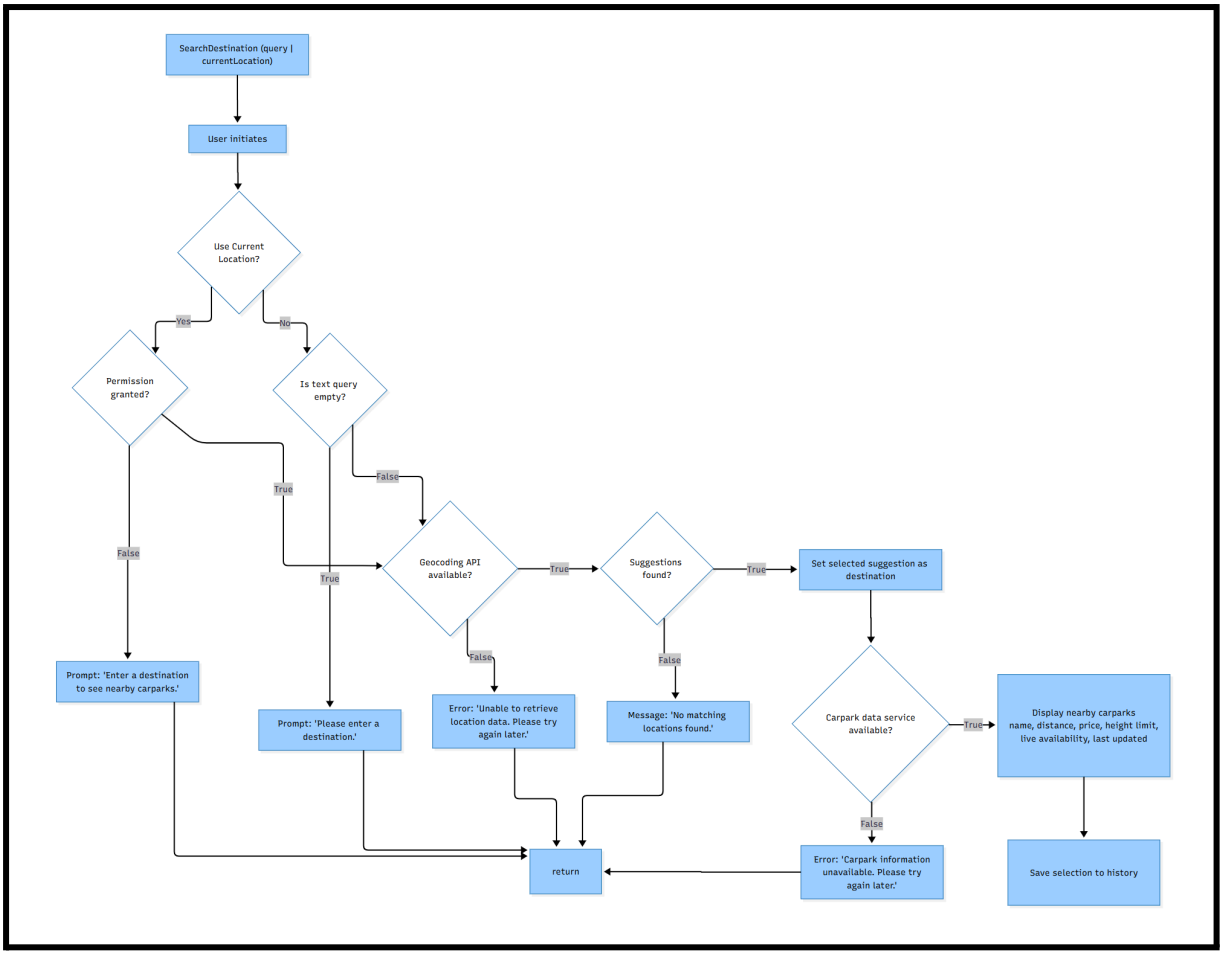
Dialog Map



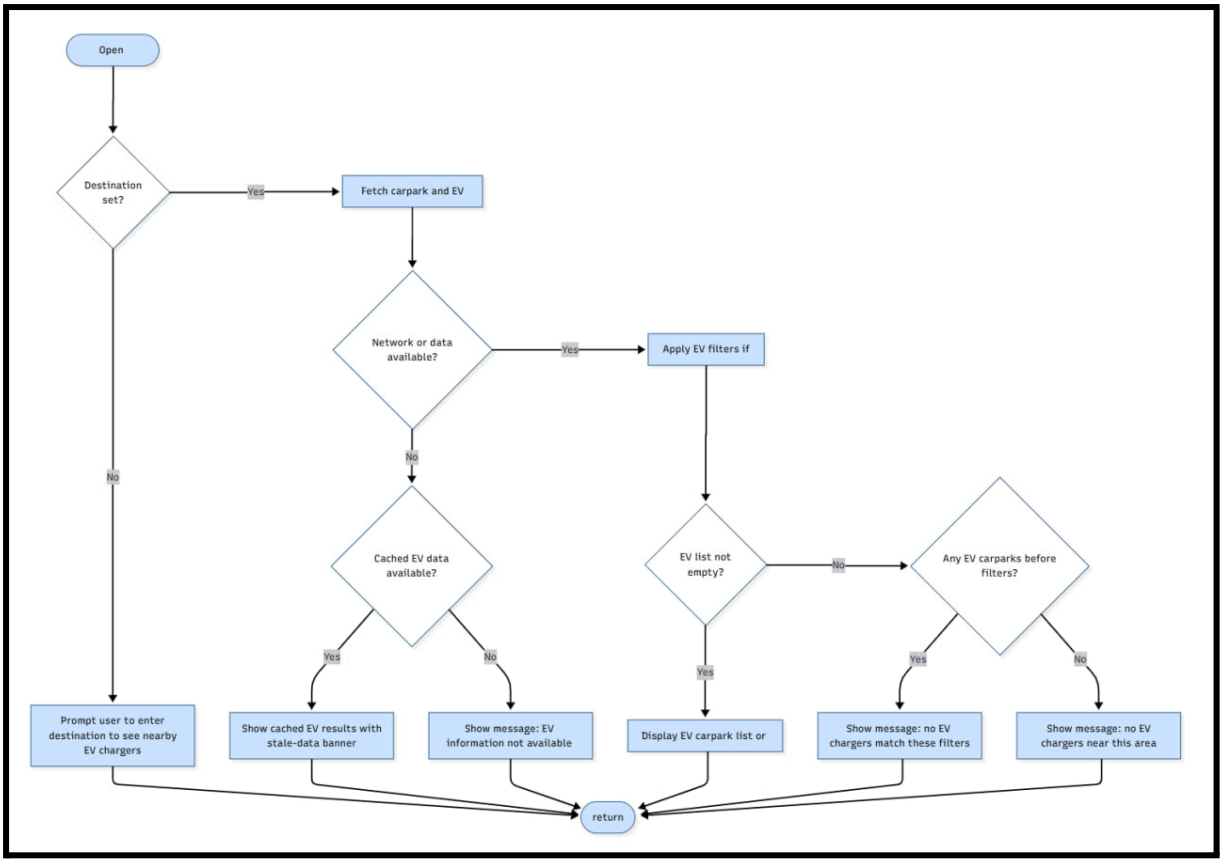
System Architecture Diagram



Basic Path Test for Login Testing



Basic Path Test for Destination Search & Nearby Carparks Testing



Basic Path Test for EV Carpark Results & Filters Testing