COMP3900: Computer Science Project

System and Software for Smart Vehicle Parking Management Project Report

Submitted by T16A-Cherry (AutoSpot) on 14 August 2025

Team Members:			
Name	zID	Email	Role
Jansen	z5373141	z5373141@ad.unsw.edu.au	Scrum Master, Frontend Development Team
Jianhui Li	z5397360	z5397360@ad.unsw.edu.au	Backend Development Team
Jonathan Lee	z5260139	z5260139@ad.unsw.edu.au	Database Development Team
Joycelin Natasha Jamin	z5423943	z5423943@ad.unsw.edu.au	Frontend Development Team
Lee Kai Li Kylie	z5443789	z5443789@ad.unsw.edu.au	Backend Development Team
Yuchao Wang	z5413536	z5413536@ad.unsw.edu.au	Product Owner, Database Development Team

Table of Contents

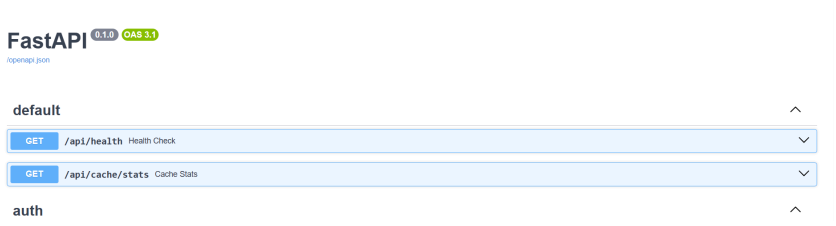
A. Installation Manual

1. **Running the Backend:**

Using Docker Compose

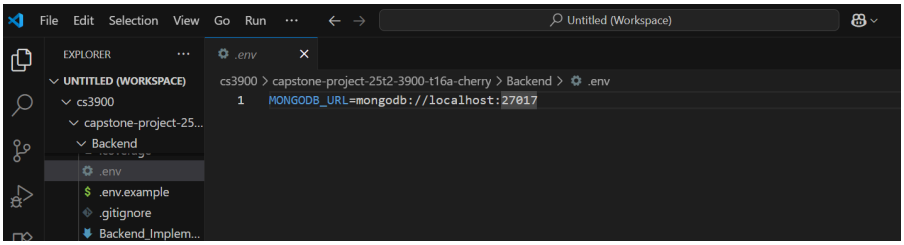
1. Install **Docker** and **Docker Compose**.
    a. Download from: https://docker.com/products/docker-desktop
2. From the backend project root, start all services (backend + MongoDB): docker-compose up --build
3. The backend will run at

- API base: http://localhost:8000
- API documentation: http://localhost:8000/docs



Running Locally with Python

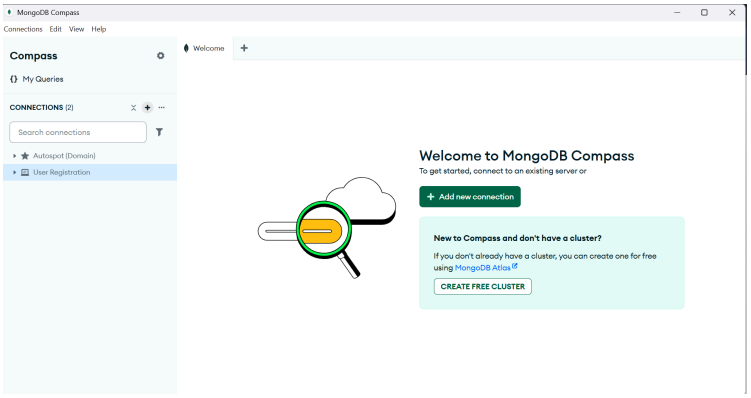
1. Install dependencies
   pip install -r requirements.txt
2. Create a ***.env*** file in the backend directory with the following content:
   MONGODB_URL=mongodb://localhost:27017



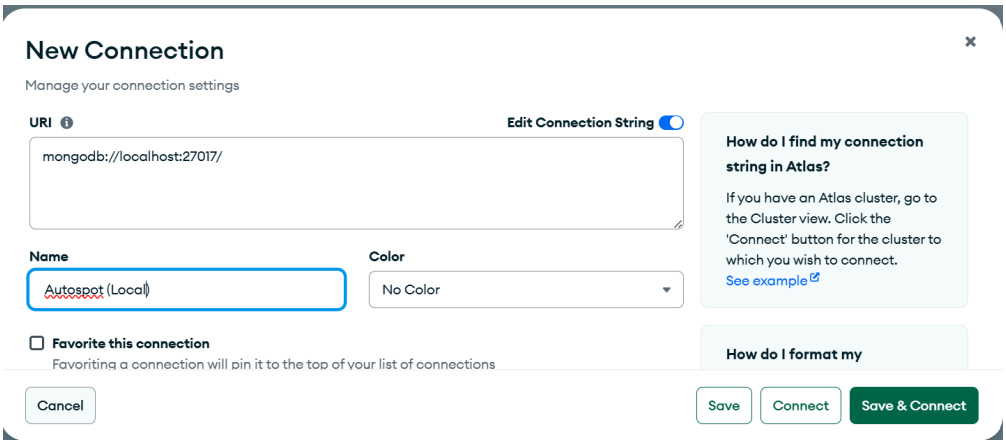
3. Start the server using the following command in terminal:
   uvicorn app.main:app --reload --port 8000
4. Access the API documentation at http://localhost:8000/docs
5. Load Sample Data (if needed):
   python app/examples/local_mongodb_storage.py

Downloading MongDB Compass:

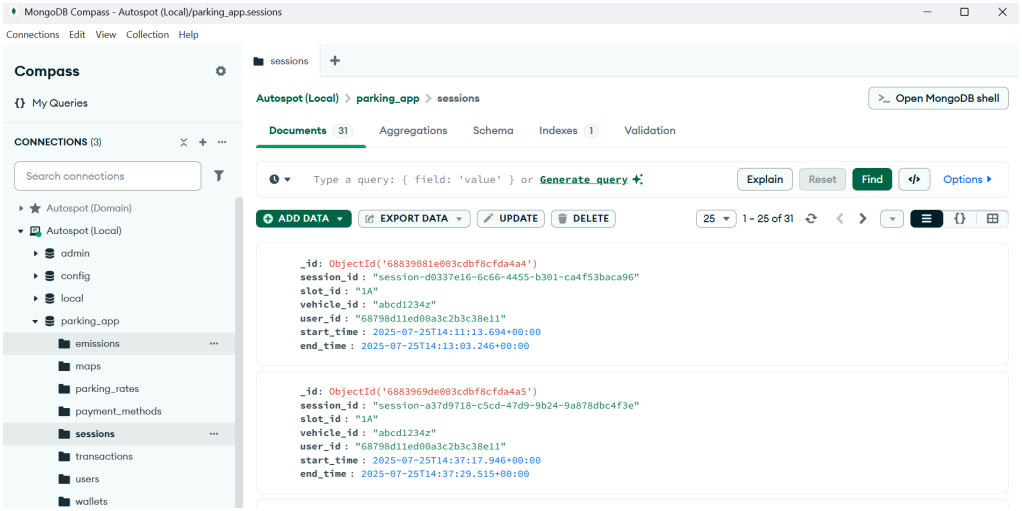
- (for better visualisation of how data is stored)
- Download MongoDB Compass here:
  https://www.mongodb.com/products/tools/compass
- Add a new connection

- 

- Fill in the following for the local database and then click ‘Save & Connect’



- Data can be viewed through Autospot (Local) -> parking_app



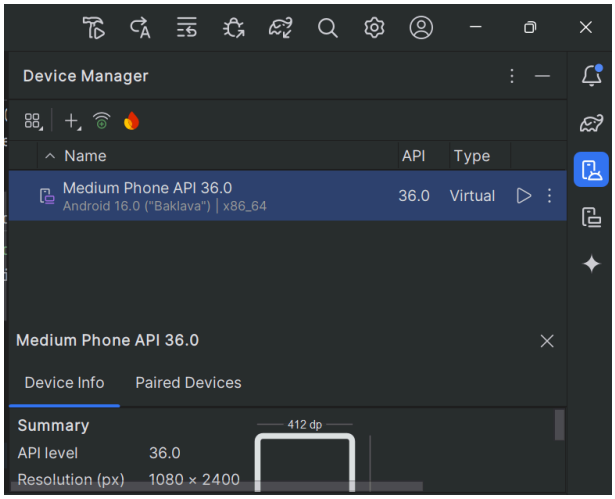
- Press ‘Find’ or the refresh button to get the latest updates saved to the database.

2. Running the Frontend:

<u>Setting up the Emulator (Android Studio):</u>

1. Install Android Studio https://developer.android.com/studio

2. Open Android Studio -> Tools -> SDK Manager:
   - Install the latest Android SDK
   - Install Android SDK Build-Tools and Android Emulator
3. Go to Device Manager -> Add a new device -> Create Virtual Device
   - Select ‘Medium Phone’ (Medium Phone API 36.0)



Installing Flutter:

1. Download Flutter from https://docs.flutter.dev/get-started/install and add it to your system PATH
2. Verify the installation by running:
   flutter doctor
3. In Android Studio -> File -> Settings -> Plugins:
   - Install Flutter and Dart plugins
   - Restart Android Studio
4. Open a terminal and navigate to:
   cd Frontend/autospot
5. Install dependencies and run the app:
   flutter pub get
   flutter run

3. Accessing the Domain sites:

A. FastAPI documentation
   - https://api.autospot.it.com/docs

B. Frontend interface
   - https://autospot.it.com/

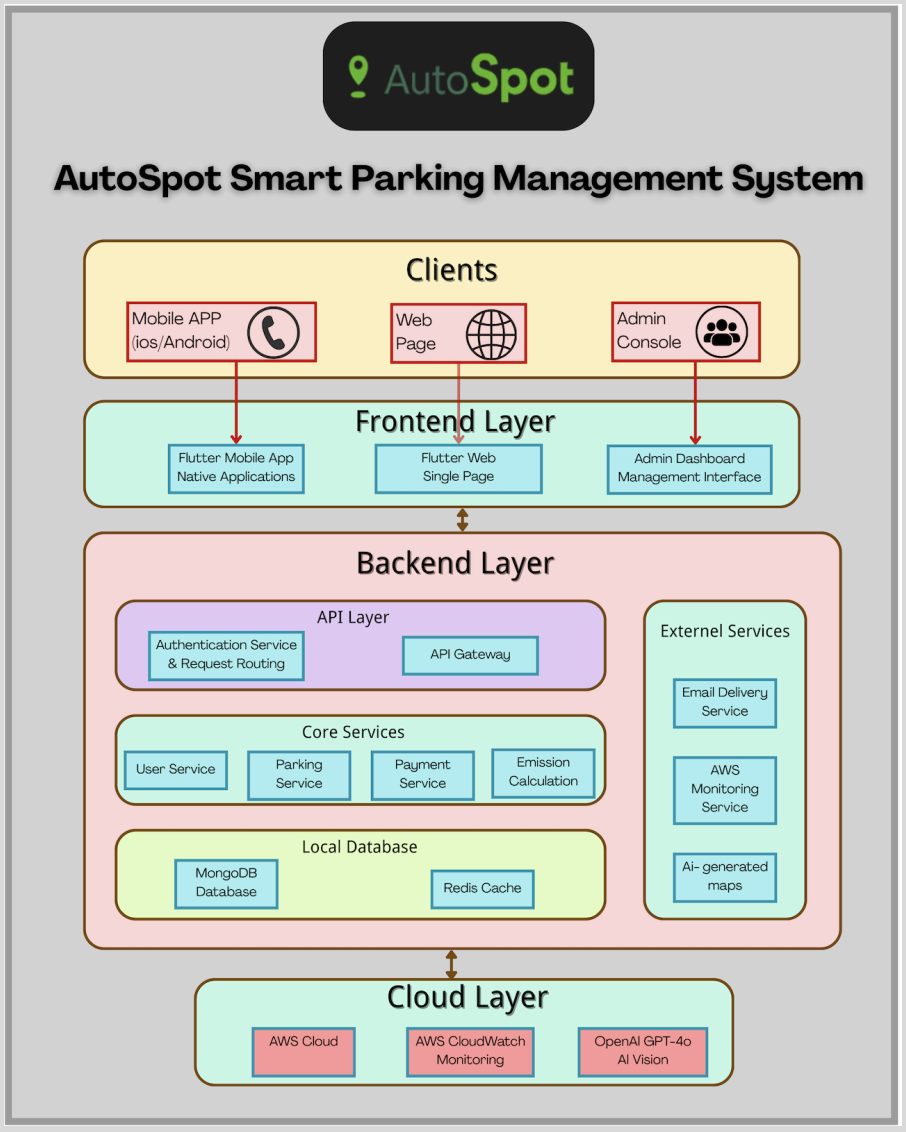
B. System Architecture Diagram

1. AutoSpot Smart Parking Management System – High-Level Architecture:

The High-Level Architecture diagram offers a top-level representation of the AutoSpot system, showcasing its key layers and interactions. It highlights the major components, technology stack, and primary data flows, enabling stakeholders to quickly grasp the system's overall structure.

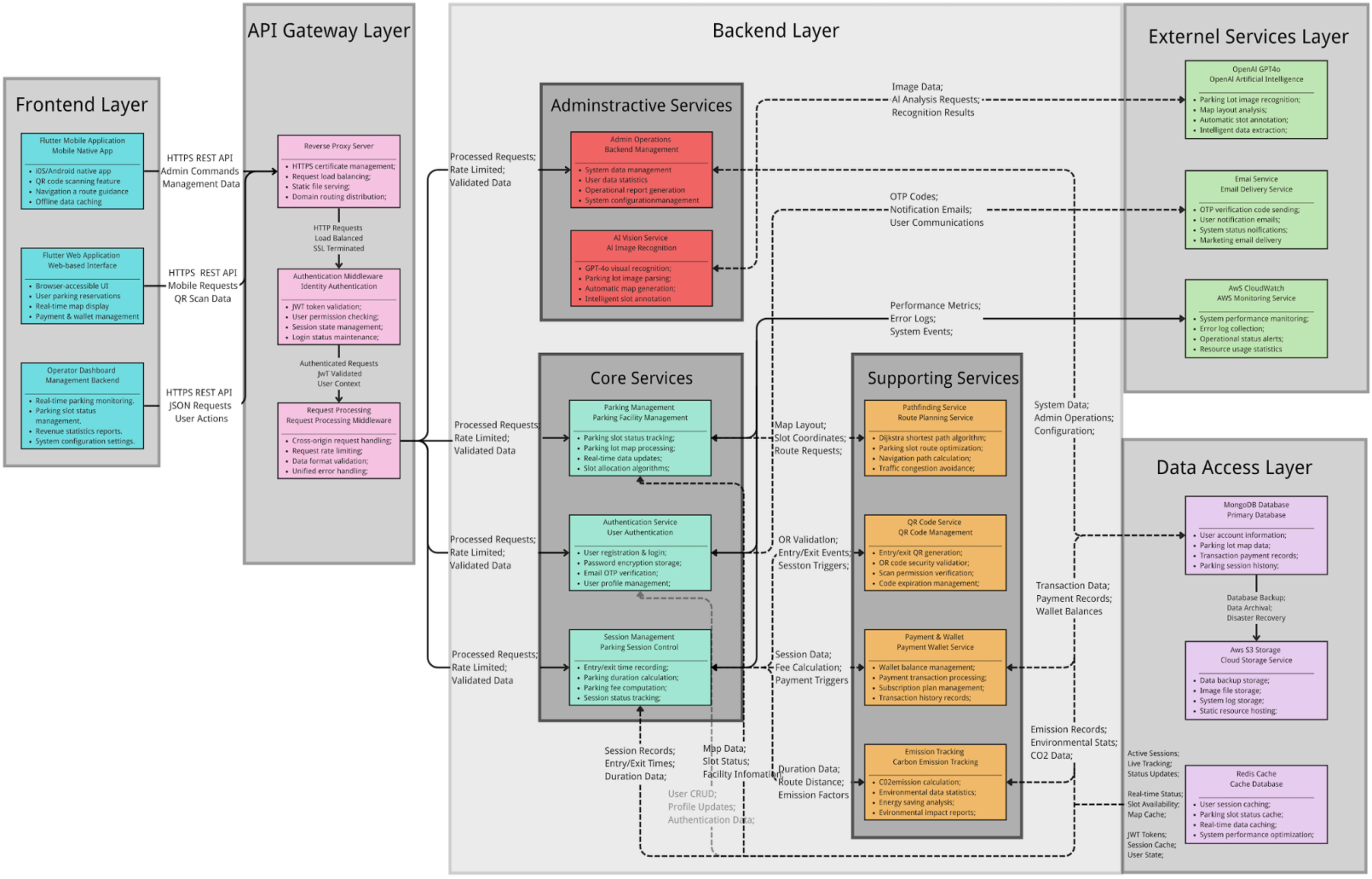
This architecture consists of four main layers:

- ***Clients*** – Mobile apps (iOS/Android), web applications, and admin consoles.
- ***Frontend Layer*** – User-facing applications built with Flutter for mobile and web, and an admin management interface.
- ***Backend Layer*** – API services, core parking/payment/emission functionalities, local databases, and integrations with external services.
- ***Cloud Layer*** – Cloud hosting, monitoring, and AI-powered image recognition for parking-related tasks.



2. AutoSpot Smart Parking Management System – Detailed Service Architecture:

The Detailed Service Architecture presents an in-depth view of the AutoSpot system's internal components and their interactions. This diagram focuses on specific service modules, API communications, database relationships, and technical implementation details, providing developers and architects with the necessary information for system development and maintenance.



For clarity, it is divided into five main sections, each addressing a specific functional area.

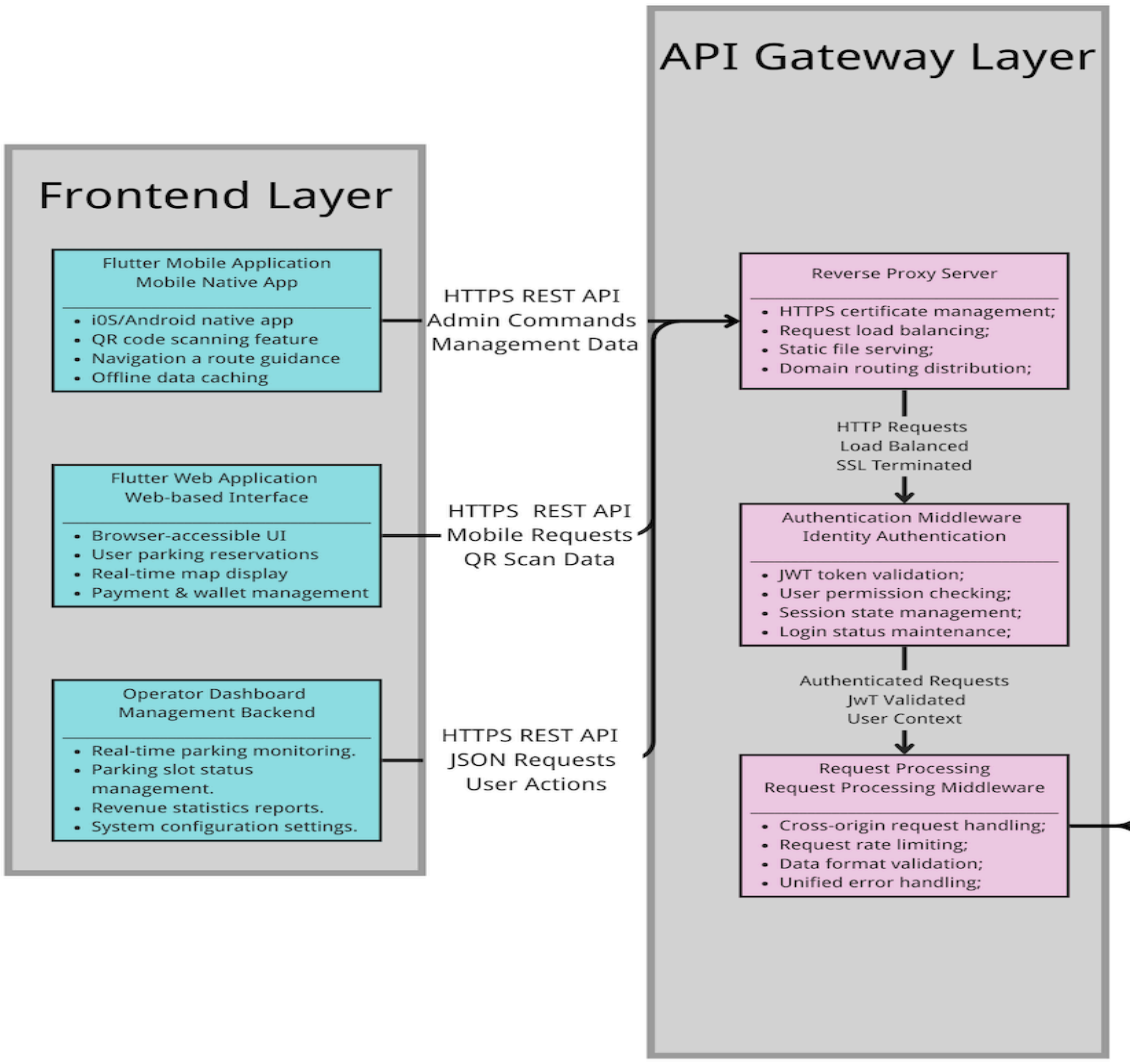
2.1 Frontend Layer & API Gateway Layer

Description:

This section covers the user-facing components — Mobile Application, Web Application, and Operator Dashboard — and their communication with the API Gateway.

The API Gateway handles:

- HTTPS REST API requests from clients.
- SSL termination and load balancing.
- Authentication (JWT validation, user permissions).
- Request processing and error handling.



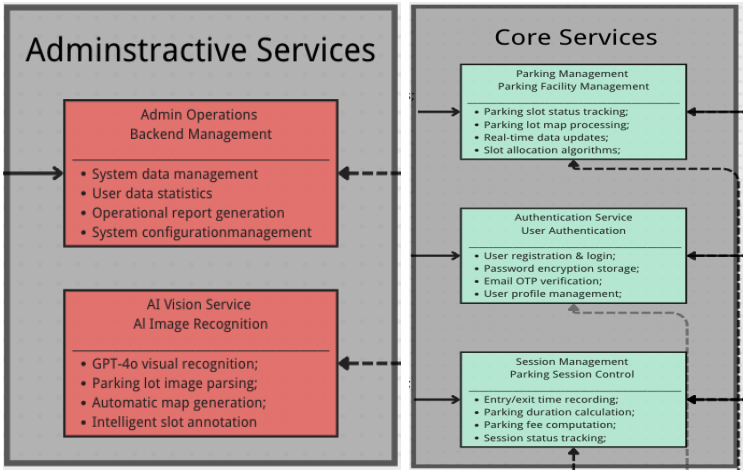
2.2 Backend Layer – Administrative & Core Services

Description:

This section represents the heart of the backend, split into:

- ***Administrative Services*** – system data management, AI-powered image recognition, parking map generation, configuration tools.
- ***Core Services*** – parking facility management, user authentication, and parking session control.

These modules are responsible for processing validated requests and executing business logic.

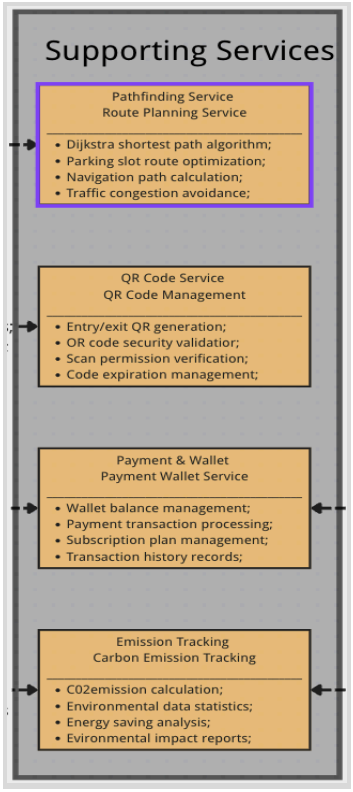


2.3 Backend Layer – Supporting Services

Description:

This section represents the heart of the backend, split into:

- ***Pathfinding Service*** – route planning and path optimization.
- ***QR Code Service*** – QR code generation and validation.
- ***Payment & Wallet Service*** – balance management, payment processing.
- ***Emission Tracking Service*** – CO_2 emission calculation and environmental impact statistics.



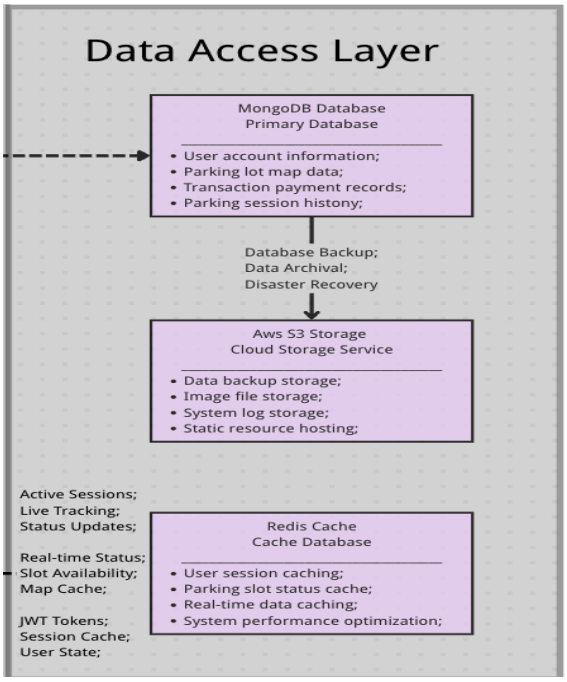
2.4 Data Access Layer

Description:

This section manages data storage, caching, and backup operations:

- ***MongoDB Database*** – primary data storage for users, parking data, and transactions.
- ***Redis Cache*** – session caching, real-time status tracking.
- ***AWS S3 Storage*** – backups, disaster recovery, and large file storage.

These modules are responsible for processing validated requests and executing business logic.

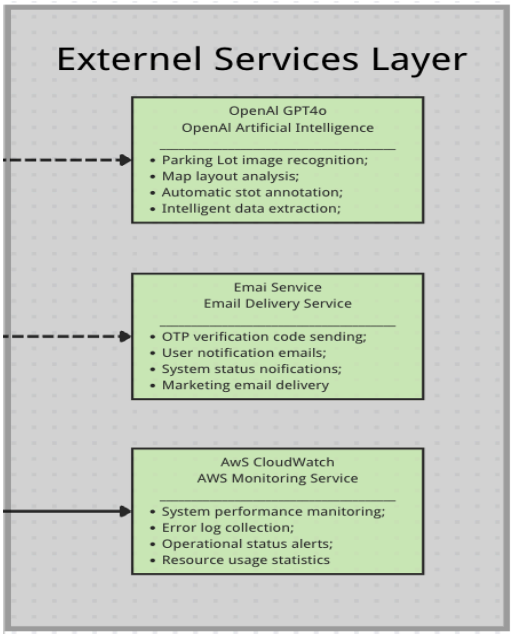


2.5 External Services

Description:

Third-party services integrated to enhance system performance and intelligence:

- ***OpenAI GPT-4o*** – AI-based image recognition, parking slot detection, and intelligent data extraction.
- ***Email Delivery Service*** – OTP codes, system notifications, and marketing emails.
- ***AWS CloudWatch*** – system monitoring, performance alerts, and resource usage statistics.



C. Design Justifications

Sprint 1 – Foundational User & Map Features

Initial Design:
Our original proposal planned for all services, including frontend web hosting, backend APIs, and the database, to run on AWS EC2 instances. The frontend was to be a mobile-only Flutter app, with all user interactions handled through Android/iOS. This includes the basic car space reservation system with user authentication, profile management, parking map display, and the ability to reserve a slot. The initial backend focus was on static map integration and simple distance-based slot recommendations.

What Changed:

- **Client Requirements:** Our client clarified the need for shortest path navigation to the parking slot, not just nearest-slot recommendation.
- **Team Realisation:** We found that hosting static web content on EC2 was unnecessarily expensive and less scalable. Additionally, we realised that the map format (JSON) was difficult for non-developers to maintain.
- **Technical Constraints:** We also realised that our initial slot assignment algorithm did not account for real-time parking slot availability, which was critical for user experience.

How We Responded:

- **Frontend:** We implemented S3 static hosting with automated deployment scripts. This reduced hosting costs by over 90% and provided global CDN distribution.
- **Backend:** We iterated on our slot assignment algorithm to include real-time slot status, improving assignment accuracy and user satisfaction. Added plan to use Dijkstra's algorithm for shortest path calculation. Furthermore, we began to explore AI-based methods to convert JPG/PDF maps into JSON format.
- **Infrastructure:** We kept backend APIs and MongoDB on EC2 for flexibility, but decoupled the web frontend to S3 for cost and scalability.

Justification:
The new architecture is more cost-effective, scalable, and user-friendly. S3 hosting for the web frontend is cheaper and more reliable, while EC2 remains ideal for backend processing and database needs. The improved slot assignment algorithm better meets user and client requirements.

Algorithm/Tool Implementation in Sprint 1:
We introduced ***Dijkstra's Algorithm*** to calculate the shortest path from a single entrance to any target parking slot. Implemented with a min-heap priority queue, it achieves ***O (E log V)*** time complexity, making it efficient for our relatively sparse 10×10 parking grid. This initial approach provided accurate pathfinding when only one entrance was relevant, but did not yet support scenarios with multiple entrances or exit comparisons.

Sprint 2 – Expanded Features, Sensor Simulation, and Infrastructure

Initial Design:
We expand the system to handle slot reservations, payment interface, improve map realism, QR code implementation, as well as adding sensor-based triggers to start the session.

What Changed:

- **Client Requirements:** Our client requested more engaging parking maps with obstacles (walls, ramps, mid-lot slots) instead of simple grid layouts. Furthermore, he asked for simulated sensor flow to more closely simulate the real world behaviour, with confirmation prompts before marking a slot as occupied. He also agreed that we could use AWS S3 to host static files, but decided to hold off on full deployment for now.
- **Team Realisation:** We realised that generating QR codes after allocating a slot limited flexibility and created extra backend complexity. Changing to scanning a building entrance QR code first allowed for a simpler flow and more accurate slot recommendations. We also noticed that while the core features were coming together, the UI/UX design still needed significant improvement. In addition, we also saw

opportunities to improve some algorithms, like the shortest path, and to enhance the system responsiveness when multiple users were active.

- **Technical Constraints:** Since real sensor integration was out of scope for this sprint, we went with a simulation instead. At the same time, the system was slowing down when multiple users were active, and our tests only covered the backend code, leaving most of the frontend code untested.

How We Responded:

- **Frontend:** Redesigned parking maps to 20×20 grid layouts with obstacles for realistic navigation challenges, and improved overall UI flow and transitions between screens. We started hosting our frontend on a s3 bucket, allowing us to save 90% of the cost from our previous ec2 instance.
- **Backend:** Improved the QR code implementation by switching from generating codes after slot allocation to scanning a code at each building entrance, making it easier to manage and enabling more accurate slot recommendations. In addition, we enhanced the QR payload structure and expiration handling to improve security and usability. We also updated the sensor simulation to include a 10 seconds weight detection period, a user confirmation step, and differentiated colors when a slot is occupied.
- **Infrastructure:** We set up AWS CloudWatch to monitor API performance, spot bottlenecks, and generate alerts for abnormal response times or traffic spikes. This proactive alerting helps us quickly address issues before they impact users. Our deployment structure now supports separate development, staging, and production environments, allowing us to test new features and fixes in isolation before releasing to users.

  To further improve backend performance, we implemented Redis caching for frequently accessed data such as parking slot availability and user session information. This reduced database load and improved API response times, especially under high user traffic. Docker containerization ensures consistent environments across all stages, and MongoDB remains our primary data store for its flexibility with semi-structured data.

Justification:

The changes in Sprint 2 made the system more realistic, scalable, and user friendly. The redesigned parking maps improved navigation accuracy and visual engagement, while the new QR code workflow simplified the reservation process, reduced backend complexity, and enabled real-time slot recommendations. Enhancements to the sensor simulation created a closer match to real-world parking behaviour, improving the accuracy of occupancy status. For the performance, AWS CloudWatch gave us better visibility on how the APIs were running so we could spot potential bottlenecks at an early stage. Additionally, hosting the frontend static files on AWS S3 kept the app available and loading quickly, but still keeping the backend in a local/simulated setup let us focus on building core features before worrying about scaling to production.

Algorithm/Tool Implementation in Sprint 2:
To support the new multi-entrance navigation requirement, we implemented the ***Floyd–Warshall Algorithm*** for all-pairs shortest path calculation, with a time complexity of ***O (V^3)***. This allowed the system to precompute distances from all entrances to all slots, enabling real-time comparison before the user enters the mall. While effective for our current map size, this approach became slower as the grid size and complexity increased.

Sprint 3 – Secure Payments, Real Time Updates, and Carbon Emissions Tracking

Initial Design:
Since we already had the payment interface in place, our plan for Sprint 3 was to fully implement the payment feature, enable instant updates to parking slot occupancy, and add a carbon emission tracking system to highlight sustainability benefits. Furthermore, due to time constraints, we decided to narrow our scope from developing both the user and admin interfaces to focusing only on the user interface.

What Changed:

- **Client Requirements:** The client required support for multiple payment card types with strong validation. They also wanted parking slot occupancy status to update instantly for all users without manual refresh, and requested a carbon emission feature to calculate and display CO_2 emissions saved for each parking session.
- **Team Realisation:** We found that simple card number input validation was insufficient, so we needed full compliance with card industry format checks. In addition, relying on manual page refresh for occupancy changes was inefficient and diminished the user experience. We also saw that carbon tracking could serve as a unique selling point, but its accuracy would depend on integrating distance and emissions factor data.
- **Technical Constraints:** Payment integration had to handle various BIN/IIN ranges, different card lengths (13–19 digits), and CVV rules, while also preventing invalid numbers from being processed. At the same time, instant slot updates required a real-time communication mechanism that wouldn't overload the server. Finally, the emissions calculation needed both baseline and actual journey distances, along with reliable emissions factor constants.

How We Responded:

- **Frontend:** Connected the carbon emission and wallet features to the backend. For the wallet, users can now top up using a valid card, with form validation ensuring all required details are completed. Users must also fill in their profile information, especially the vehicle plate number, before they can choose a parking slot. In addition, we updated the

"Send Code" button for account registration and password reset so it can only be pressed once per request to prevent multiple OTP requests before the first code is received.

- **Backend:** We developed a carbon emission calculator to compare baseline and actual journey distances, displaying CO_2 savings in both grams and percentage. In addition, we added instant slot status updates so all users see changes immediately without refreshing. We also implemented a comprehensive card validation system for all supported card types (Visa, Mastercard, American Express, Diners Club, Discover, UnionPay, Maestro), including:
    - BIN/IIN prefix checks.
    - Card length verification for each type.
    - CVV length validation (3–4 digits).
    - Expiry date validation to reject expired cards.
- **Infrastructure:** Configured and optimised the server to support real-time communication for instant slot status updates. Deployed updated backend services with the new payment validation and carbon emission calculation features.

Justification:
Our sprint 3 updates have significantly improved the app's security, usability, and overall user experience. The comprehensive card validation system ensured secure and reliable payments by checking BIN/IIN prefixes, card length, CVV, and expiry date. The real-time slot status updates meant users could instantly see changes without refreshing, improving responsiveness and reservation accuracy. Carbon emission tracking added a valuable sustainability feature that differentiated the app in the market. On the frontend, connecting the wallet and carbon emission features to the backend completed the payment and tracking workflows, while the OTP button update for registration and password reset prevented multiple requests and improved usability. Infrastructure changes to support real-time communication and deploy new features ensured that the system was ready for smooth, responsive operation.

Algorithm/Tool Implementation in Sprint 3:
To overcome Floyd–Warshall's performance bottleneck, we upgraded to the ***A* algorithm***, using Manhattan distance as the ***heuristic*** for grid-based maps. This preserved the optimality of Dijkstra's results while significantly reducing search space and improving computation time, especially in complex maps with obstacles.

D. Complex Algorithms & Tools

Slot Assignment Algorithm:
We researched and implemented a dynamic slot assignment algorithm that considers both real-time slot availability. This required integrating live data from our MongoDB backend and using Dijkstra's algorithm for shortest path calculation within the parking map graph.
Automated S3 Deployment:

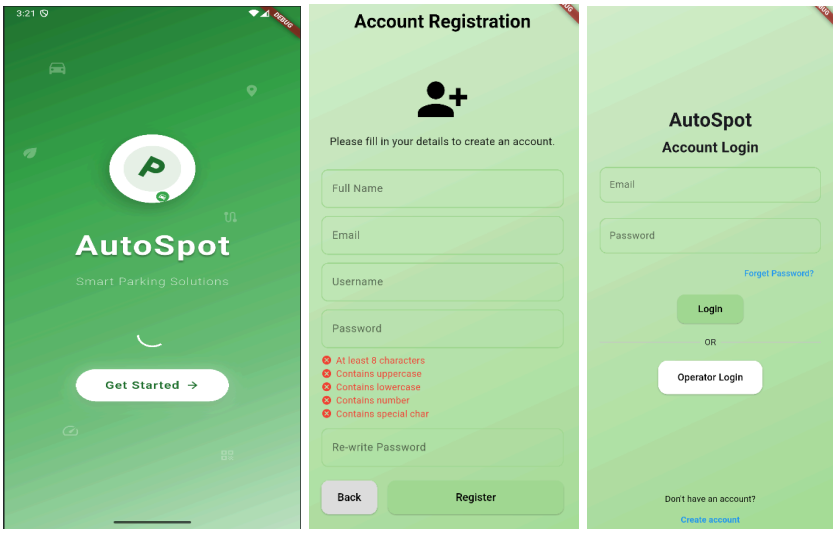
We developed scripts to automate the build and deployment of our Flutter web app to S3, ensuring correct MIME types and cache headers for optimal performance and browser compatibility. We were also about to cut our costs from $15 per month to around $1 per month, as the ec2 instance charges us 24/7 for a live server whilst the s3 bucket only charges us for storage and requests when someone visits the website.

E. User-Driven Evaluation of Solution

Main Features of the Solution (with criteria and evaluation of the effectiveness)

Our system incorporates key features designed to improve the parking experience for users:

- **Welcome page, Registration and Login**



Users can register by providing their full name, email, username, and password. The system then validates and securely stores the hashed password and credentials provided. To verify the validity of the email, an OTP is sent to the user's email address, which must then be entered to complete account verification. Upon successfully verifying their account, users can then log in by entering their email and password.

Evaluation criteria:

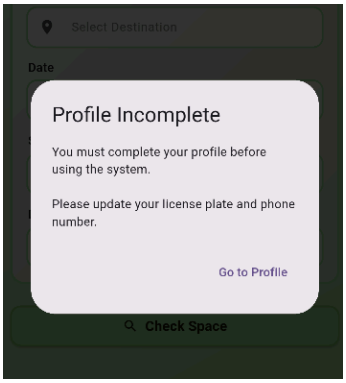
The effectiveness of this process is measured by whether the system is reliable in authenticating user credentials, securely handling password storage, and intuitive to use from the user perspective. Additionally, clear error messages (but without revealing too much about user credentials) is also taken into account.

Effectiveness Evaluation:

The solution meets the criteria as it securely validates user data through an OTP verification system, ensuring the email provided at registration is valid. Passwords

are also required to meet length and complexity requirements, avoiding common passwords, and retyped to reduce errors. They are then securely stored as hashed strings, making it harder for attackers to recover the password if the database is compromised. This thus offers a reliable authentication system. Additionally, an intuitive and easy to navigate interface with animated elements are used to capture user attention, and clear error messages are displayed if login credentials entered by users are incorrect. Perhaps future improvements could include integrating social login features (including linking to existing Google, Facebook, Instagram, Twitter accounts). This might further simplify the login process and improve user experience, thus driving registration rates.

- **Edit Profile Details**



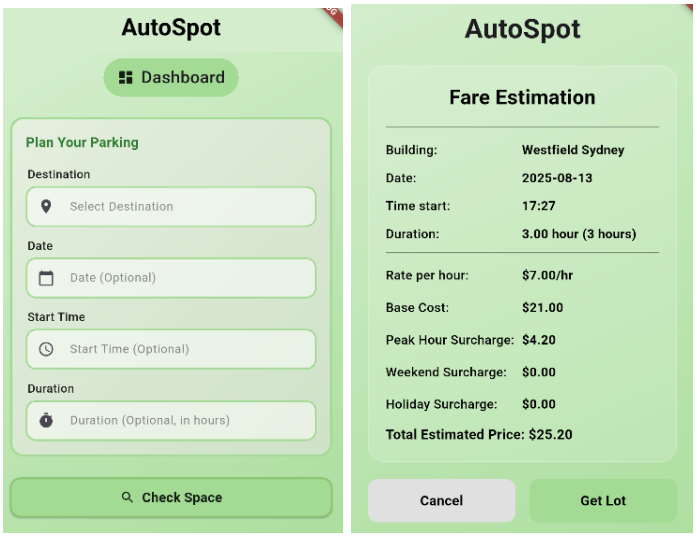
Apart from basic features allowing users to modify their account details (full name, username, email, phone number, address, license plate) and changing or resetting passwords, the latest update to this feature adds a prompt redirecting users to fill in their license plate and phone number before they can book a slot. This helps manage parking sessions by saving user details, preventing double booking and ensuring multiple users are not assigned the same 'nearest parking slot' simultaneously.

<u>Evaluation criteria:</u>
This feature should allow users to easily and securely update their details including their license plate and phone number. It should also reliably prevent double booking by enforcing the condition that only allows slot booking after the required details have been successfully updated. The effectiveness will be evaluated based on its accuracy in preventing booking conflicts, as well as whether updated details are able to be securely saved and retrieved.

<u>Effectiveness Evaluation:</u>
This feature effectively meets the objectives as it securely stores updated user details and would continue to prompt users to update any required fields that are missing. This thus ensures bookings cannot proceed with incomplete information,

preventing double bookings caused by system errors or missing data. However, perhaps one improvement could be to implement external license plate and phone number verification processes to prevent users from entering random or false details to bypass the system.

- **Planning Parking & Fee Estimation**



Before getting a slot, users can select their destination and optionally input their expected arrival date and time, as well as the duration of stay. Using these data, the system estimates the parking fare including peak-hour rates and other surcharges. This feature allows users to plan their cost ahead of time.

Evaluation criteria:

The fare estimation feature can be evaluated based on the accuracy of cost calculations (compared to actual charges), clarity and transparency of cost breakdown, as well as ease of access to these data.

Effectiveness Evaluation:

The implementation meets these objectives as fare estimates are close to the actual amount in most cases and clearly indicate the breakdown of peak-hour, holiday or weekend surcharges. This allows users to easily access and understand how parking fares are calculated and plan their budget in advance especially if they plan to stay for extended periods of time. Despite this, some discrepancies were noted when users stay beyond their indicated duration, resulting in higher final charges compared to the initially predicted amount. As such, perhaps one way to improve this feature in the future would be to add a real-time update, notifying users about their extended stay.

- **Calculation of Routes**

We compute routes with A* on a weighted directed grid graph where nodes are (level, x, y) and edges represent walkable corridors, ramps with level-transition penalties, and one-way links into parking slots. The heuristic is admissible and consistent: Manhattan for 4-directional grids or Octile/Euclidean for 8-directional movement, scaled by the minimum step cost, plus a fixed per-level penalty. This preserves Dijkstra-level optimality while expanding far fewer nodes and reducing latency.

For global distance fields or when an admissible heuristic is unavailable, we fall back to (multi-source) Dijkstra. For multiple entrances, we run A* per candidate entrance and pick the minimum, or use a precomputed distance field when global comparison is required. Real-time occupancy changes trigger fast re-planning.

Evaluation criteria:

A. Optimality and correctness: optimal paths under non-negative weights; correct handling of ramps, one-way slot edges, and blocked cells.
B. Robustness and scalability: stable under frequent occupancy changes and many simultaneous users; graceful fallback to Dijkstra if needed.

Effectiveness Evaluation:

On realistic multi-level 20×20 maps with obstacles and ramps, A* produced the same optimal routes as Dijkstra while reducing average node expansions by 50–80%. Multi-entrance selection via per-entrance A* or precomputed distance fields consistently improved perceived walking distance. Real-time updates were reflected with rapid re-planning and no visible lag.

- **Map (Core Design & Data Model + GPT-4o Optional Enhancement)**

Overview:

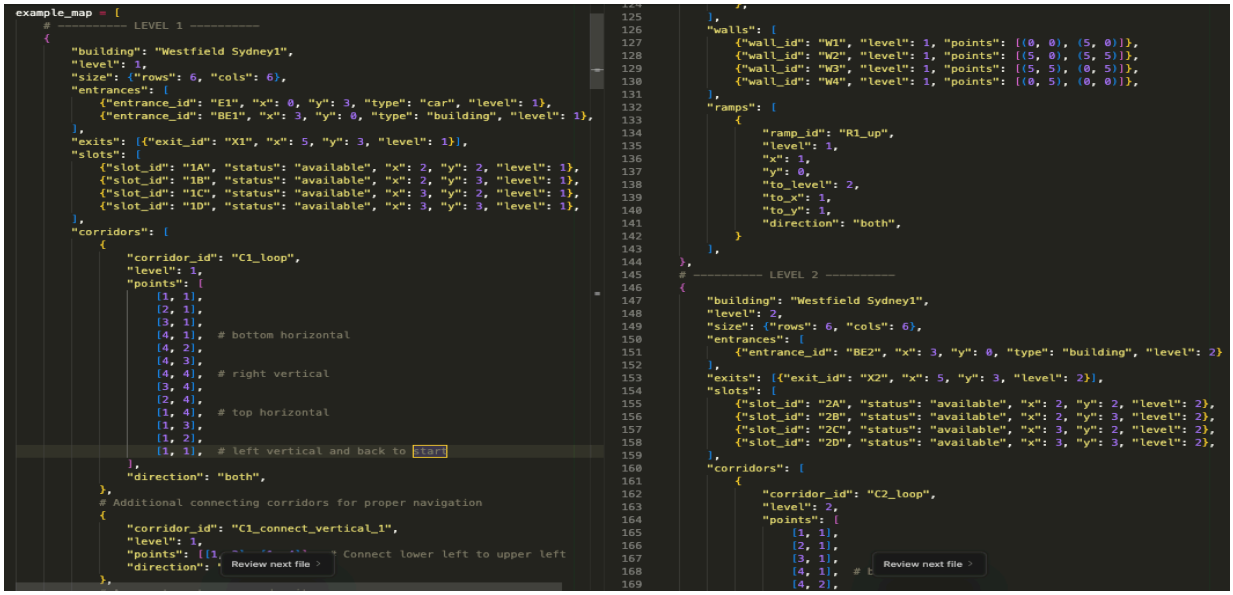
1. The system adopts a typed grid graph abstraction, using an editable JSON data model to precisely describe the parking lot structure.
2. Path planning and visualization both consume this model as the single source of truth.
3. GPT-4o serves as an optional enhancement to semi-automatically convert parking lot images/PDFs into an initial typed JSON draft, accelerating the modeling process.

Coordinates & Grid:

1. **Coordinate System:** (level, x, y); each level defines rows/cols, only orthogonally adjacent edges (no diagonals).

2. **Node Types & Attributes:**

- *Slot:* slot_id, status(available/allocated/occupied), x,y,level; optional attributes such as accessible/ev_charging/shade/size/weight.
- *Edge rule:* corridor → slot (one-way; slot is a destination, not a pass-through).
- *Corridor:* bi-directional connectivity between adjacent points; may have weights (e.g., narrow passage).
- *Wall:* non-passable, blocks connectivity.
- *Entrance/exit:* linked to nearest corridor point (entrance bi-directional, exit one-way corridor → exit).
- *Ramp:* (level,x,y) ↔ (to_level,to_x,to_y), can connect across levels/maps with explicit level_cost (floor change penalty).



GPT-4o Optional Enhancement (Image → JSON):

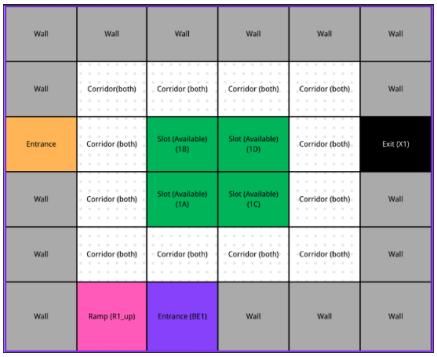
Purpose: Convert parking lot images/PDFs into a typed JSON draft (levels, corridors, ramps, walls, entrances/exits, slots & attributes) to speed up modeling.

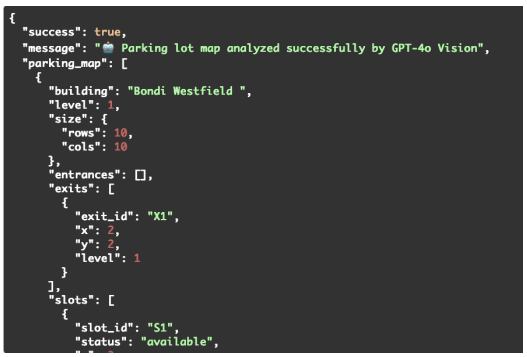
Accuracy Expectation: Output should resemble the source map (not 1:1); manual review required at /parking/maps/{id} and correction via /parking/maps/update.

Workflow:

1. Upload image;
2. File validation (type/size/duplicate);
3. GPT-4o parsing;
4. Return parking_map + validation;

5. Metadata + human confirmation/edit;
6. Commit to database;





API Endpoints (FastAPI):

- **Upload/Parse:** POST /parking/upload-map (image ≤10MB, typed JSON generation; PDF auto-converted to image).
- **Retrieve/Edit:** GET /parking/maps*, GET /parking/maps/{map_id}, PUT /parking/maps/update.
- **Frontend Data:** GET /parking/slots, /parking/entrances, /parking/exits, /parking/slots/summary.

Evaluation criteria:

A. **Accuracy:** strict corridor adjacency; slot one-way edges; ramps correctly link across levels/chains; walls are non-passable.
B. **Editability:** full typed structure retrievable & atomically updatable; changes instantly reflected in routing & visualization.
C. **Extensibility:** any node can have additional attributes (accessible, charging, shade, size, weight, etc.) without changing algorithms; supports multi-level/multi-map chaining via ramps.

User experience:

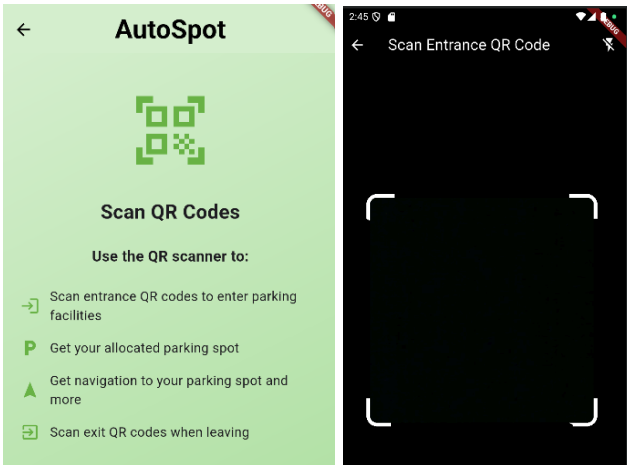
Clear, turn-efficient routes with accurate distance and travel time estimates, ensuring smooth and reliable navigation within the parking facility.

Effectiveness Evaluation:

Consistently converts parking maps into structured, typed JSON; edits (slot status, corridor adjustments, cross-level ramp chains) are reflected in routing and map UI instantly.

Slot one-way semantics effectively prevent “path-through-slot” issues; ramp chaining builds multi-level connectivity with correct routing, low latency, and maintainable structure.

- **QR Scanner for Arrival Detection**



Upon entering the building, users scan a QR code to confirm their arrival. This triggers the system to allocate the nearest available parking slot based on the user's preferences (e.g., preferred level) and current availability. This ensures that the allocation process is accurate and responsive to real-time arrivals.

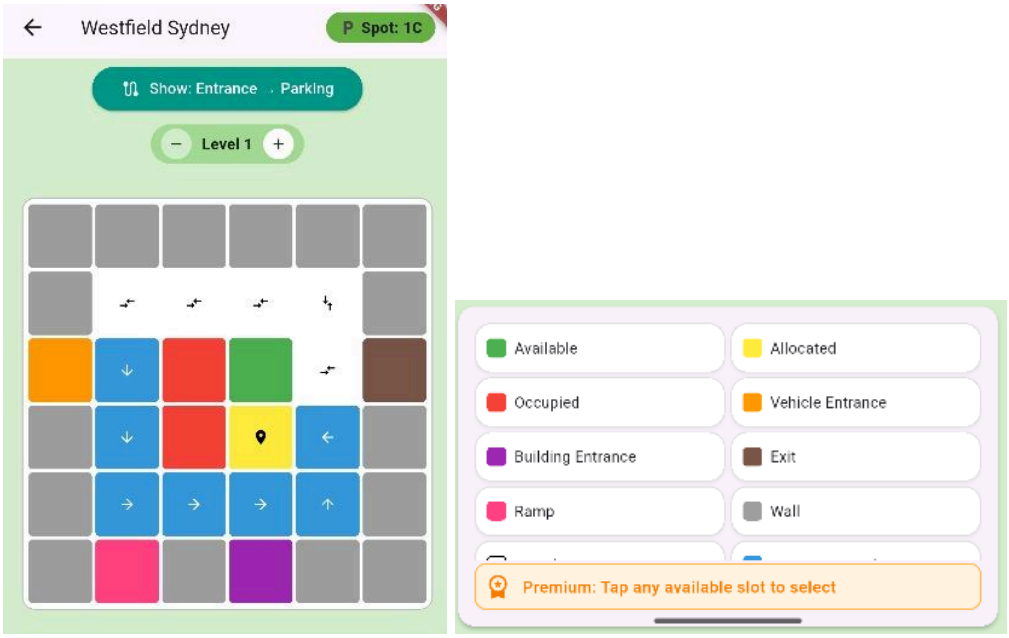
Evaluation criteria:

The QR scanner should accurately detect user arrivals and allocate the nearest available parking slot or the slot picked by the user (premium subscription). The scanning process should be quick, intuitive and compatible with common mobile devices. It should then reliably process scans, update slot status and allocate slots without delay.

Effectiveness Evaluation:

The QR scanner feature effectively detects user arrivals and allocates parking slots in real time which meets both the accuracy and reliability objectives. The operation is also smooth and quick under normal usage, updating users on availability of slots promptly. However, more testing would be needed for high-traffic conditions. Additionally, perhaps an improvement that can be made to this feature includes implementing QR expiry mechanisms (provided a screen is used to display the QR code at the entrance and exits) to prevent misuse.

- **Interactive and Easy-to-Navigate Parking Map**



The system provides an intuitive map interface that dynamically guides users to their allocated slot.

- Once a destination or building is selected, the system automatically displays the path from the car entrance to the assigned slot.
- A toggle feature allows users to seamlessly switch between two map views:
    - From car entrance to parking slot
    - From parking slot to destination or nearest building entrance
        - This dual-view approach ensures that users have clear navigation both when parking and when proceeding to their intended destination.

Evaluation criteria:

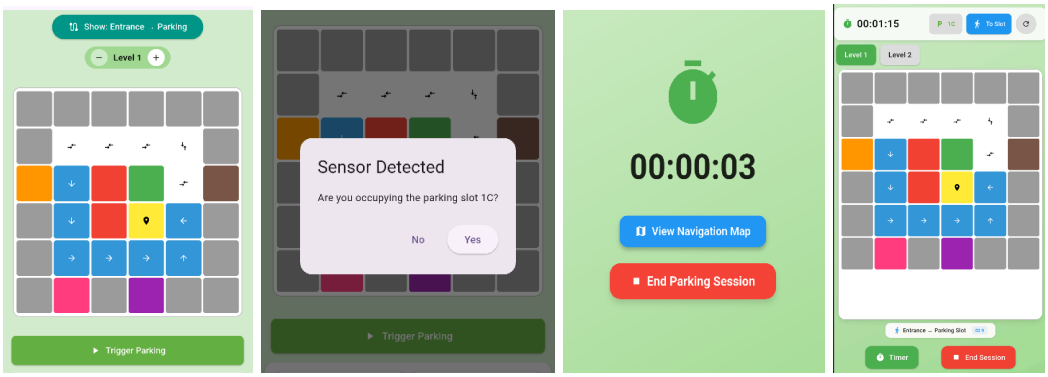
The effectiveness of the map interface and dual-view navigation is evaluated based on the accuracy of navigation routes, how usable and intuitive it is, allowing users to operate the system without requiring prior training; the clarity of marked routes and obstacles (e.g. walls, ramps, building entrances), as well as responsiveness of the page which provides smooth transitions between different interfaces.

Effectiveness Evaluation:

The system mostly meets these objectives, with high accuracy in navigation in most cases, and is easy to use without requiring prior instructions. Additionally, the routes are clearly marked with directional labels; obstacles, entrances and exits are distinctly marked in different colours. While performance of transitioning screens is generally smooth, more user feedback after deployment of

the app will have to be collected to determine the effectiveness and satisfaction of these features.

- **Sensors & Sessions Mechanism**




```
_id: ObjectId('689cbabfb743466a70c6a856')
session_id : "session-e3f68d84-4312-4073-80dc-2e9ac27e22c6"
slot_id : "1C"
vehicle_id : "abc124545z"
user_id : "68913435d5f5dba76f111651"
start_time : 2025-08-13T16:18:07.454+00:00
end_time : 2025-08-13T16:22:02.927+00:00
entrance_id : ""
exit_id : ""
```


The sensor features currently includes a Trigger Parking button, simulating a weight detection system in a parking slot. When pressed, it sends a prompt to users: “Sensor Detected - Are you occupying the parking slot?” and starts the session if the user selects “Yes”. User profile details (user_id and vehicle_id) are then saved to the parking session, allowing operators to identify the vehicle and user assigned to the slot. This also helps address cases where non-users of the app occupy a slot, as operators can manually mark it as occupied using the vehicle_id (vehicle number plate). Additionally, the saved phone number from the user profile would be useful for contacting the owner in situations where their vehicle is damaged or other accidents.

<u>Evaluation criteria:</u>

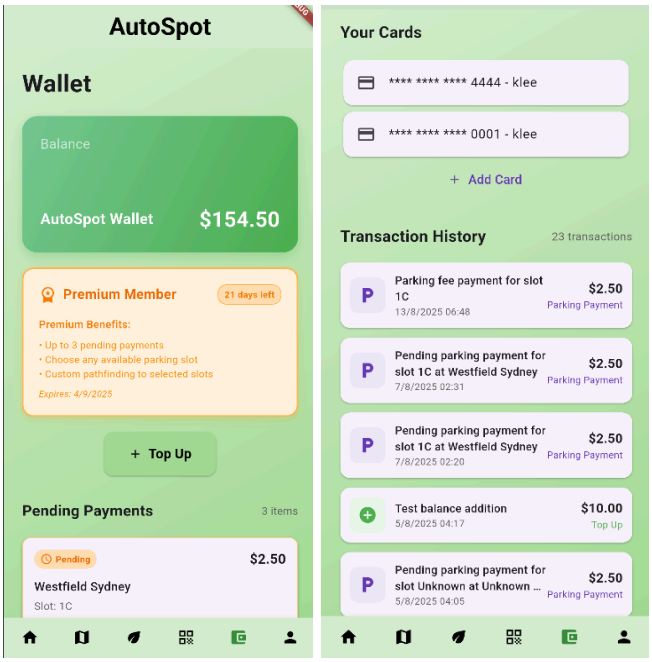
The sensor and session booking mechanism can be evaluated based on accuracy of detection, whereby the system reliably prompts users when a weight is detected at the slot. Additionally, ease of confirming occupancy, and accuracy in linking the correct user_id and vehicle_id to each parking session should also be considered.

<u>Effectiveness Evaluation:</u>

At the current stage, the sensor trigger response is simulated using a “Trigger Parking” button. This feature reliably allows users to confirm their occupancy through a simple and straightforward prompt. Once it has been confirmed, user_id

and vehicle_id details are correctly reflected under the session details. However, in real-world implementation, integration with actual weight sensors and detection systems would be needed to fully test and verify the reliability of this feature.

- **Wallet Features**



The wallet function allows users to save payment methods, top up wallet balances and display pending payments as well as transaction history.

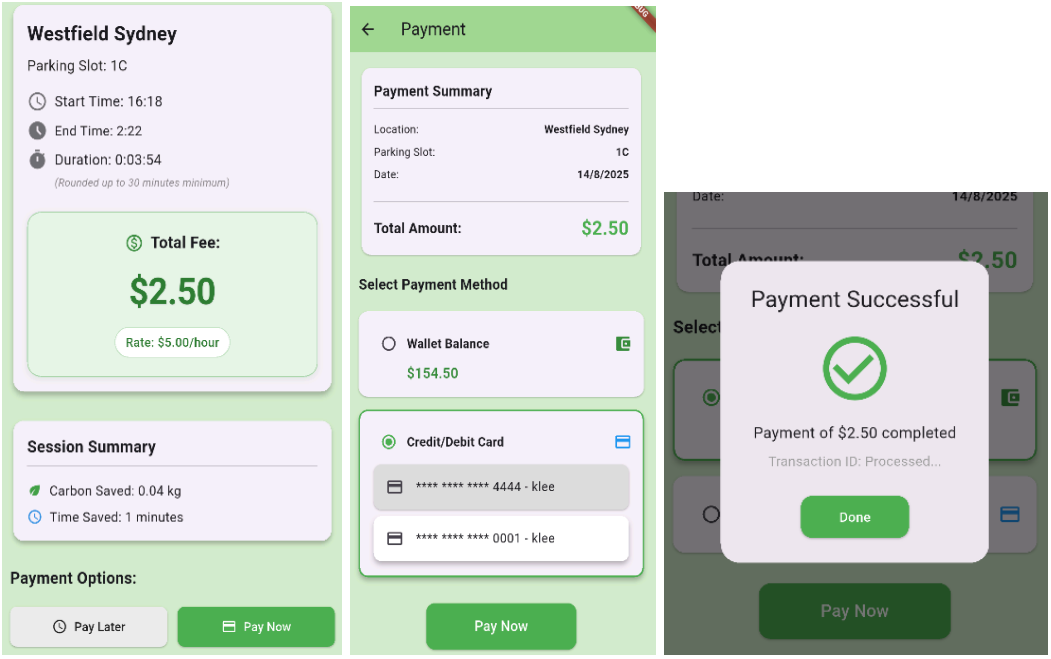
Evaluation criteria:

The card validation and payment management feature can be evaluated based on the accuracy of validating card details, security compliance where sensitive card data should be handled securely, records of wallet balance, transaction history and pending payment should be displayed clearly and accurately reflected.

Effectiveness Evaluation:

This feature effectively achieves these objectives, with card validation using length and pattern checks (through use of Luhn checksum library), preventing the use of invalid card numbers. Security is maintained by storing only a unique non-sensitive identifier (payment_method_id) and only the last four digits of the card. Additionally, transaction history clearly indicates a payment description, amount, date and time of transaction, while pending payments are updated consistently. In real-world deployment, integrating secure payment gateways such as Stripe, Paypal or other PCI-DSS compliant services would further enhance security, handling sensitive card data externally to ensure full card digits and CVV codes are not being stored in the system.

- **Pay Now Option**



Users can choose to pay immediately after a parking session ends. This can be done either by using their wallet balance or to pay using a saved credit/debit card.

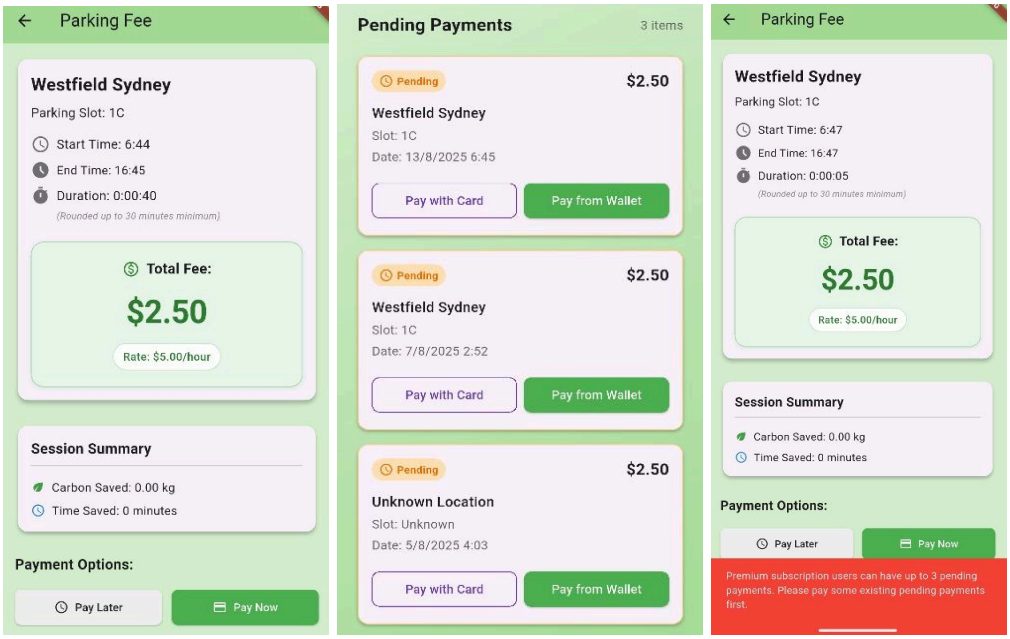
Evaluation criteria:

The immediate payment feature should be intuitive and allow for payment flexibility, allowing choice between multiple ways to pay. Payments should be processed quickly without delays, with a confirmation message indicating the correct amount and this receipt can also be retrieved in the future.

Effectiveness Evaluation:

The feature meets the criteria in terms of offering payment through both wallet balance as well as from securely stored cards, as outlined in the wallet feature. Upon successfully making a payment, the wallet balance is updated, and users would receive a payment success message. This can also be referred to under the transaction history in the wallet tab at a later time. While the current setup works well for testing purposes, integrating PCI-DSS compliant payment gateways would enhance security and management of payment methods in real-world use. Future improvements could also include having options to directly input card details instead of relying on saved methods.

- **Pay Later Option**



At the end of a parking session, users can choose to pay immediately or defer payment (based on their subscription tier).

- **Basic Tier**: Can hold one unpaid session at a time.
- **Premium Tier**: Can hold up to three unpaid sessions simultaneously. This feature offers flexibility to frequent users.

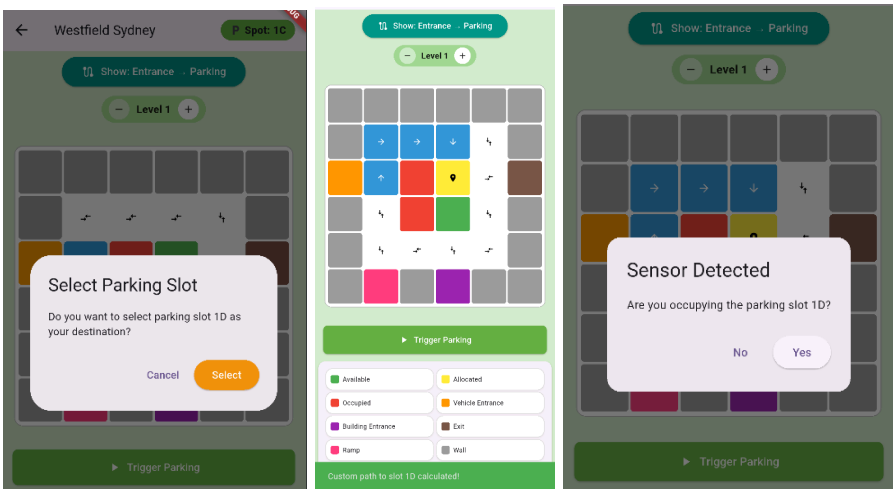
Evaluation criteria:

The Pay Later feature can be evaluated based on payment flexibility, allowing users the option to pay immediately or defer payment within the limits of their subscription plan, the accuracy of session tracking whereby unpaid sessions are kept up to date, and the simplicity and reliability of the payment process so that it runs smoothly without errors or delays that might cause disruptions to parking.

Effectiveness Evaluation:

The Pay Later option has been well-received for its convenience, particularly in premium users perspective. However, real-world testing with a live payment gateway is needed to assess performance under load.

- **Premium User Slot Selection**



Premium tier users have the option to manually select their preferred parking slot from the available spaces. This feature provides greater control and convenience for premium subscribers who may prioritize specific locations within the parking facility.

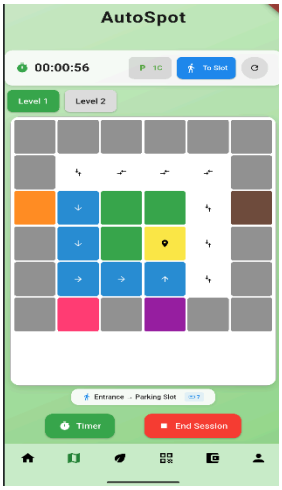
Evaluation criteria:

The effectiveness of the premium manual slot selection feature can be determined based on whether the system only displays genuinely vacant slots for selection, ease of browsing and selection of preferred slots for premium users, as well as real-time reflection of slot availability to prevent conflicts or accidental double booking.

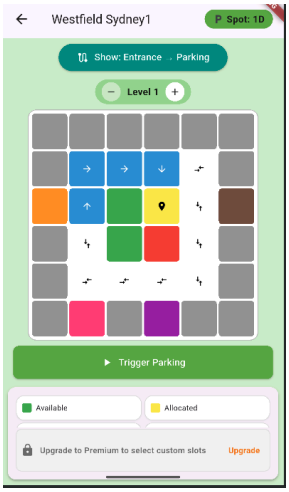
Effectiveness Evaluation:

The feature does meet its main objectives to a large extent, as real-time availability of parking slots are regularly updated and the selection process of slots is fairly straightforward and easy to navigate. While more testing and user feedback will be needed after deployment to ensure operations are working optimally especially in high-traffic conditions, perhaps future improvements could include having maps with live view or images to allow users to visualise slot locations and other attributes (e.g. size, shade, EV charging compatibility).

- **Multi-Threading Enhanced Slot Selection**



(User 1)



(User 2)

All users have access to an advanced manual parking slot selection system that utilizes multi-threading technology for enhanced performance, real-time updates, and superior user experience. This feature provides all users with greater control and convenience while ensuring optimal system responsiveness even under high-traffic conditions, through concurrent processing of slot status updates, conflict detection, and user validation.

Evaluation criteria:

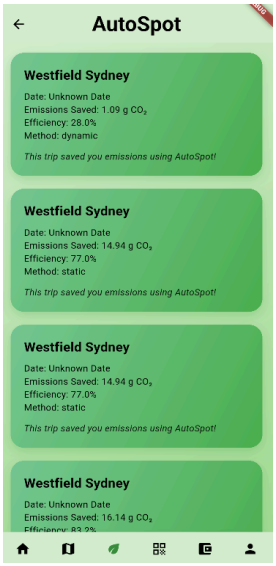
The effectiveness of the multi-threading enhanced manual slot selection feature can be determined based on whether the system only displays genuinely vacant slots for selection through concurrent validation threads, ease of browsing and selection of preferred slots for all users with real-time visual updates, as well as real-time reflection of slot availability to prevent conflicts or accidental double booking through multi-threaded conflict detection and resolution mechanisms.

Effectiveness Evaluation:

The feature does meet its main objectives to a large extent, as real-time availability of parking slots are continuously updated through dedicated monitoring threads and the selection process of slots is enhanced with concurrent processing for faster response times. The multi-threading architecture ensures that slot validation, conflict detection, and user authentication run in parallel, significantly reducing latency and improving user experience for all users regardless of subscription tier. While more testing and user feedback will be needed after deployment to ensure operations are working optimally especially in high-traffic conditions, the current implementation demonstrates superior performance through concurrent thread management accessible to all users. Future improvements could include having maps with live view or images to allow users to visualise slot locations and other attributes (e.g. size, shade, EV charging compatibility), enhanced with real-time 3D rendering capabilities and

AI-powered slot recommendation algorithms based on user preferences and historical data analysis, ensuring equal access to advanced features for all users.

- **Carbon emission calculations**



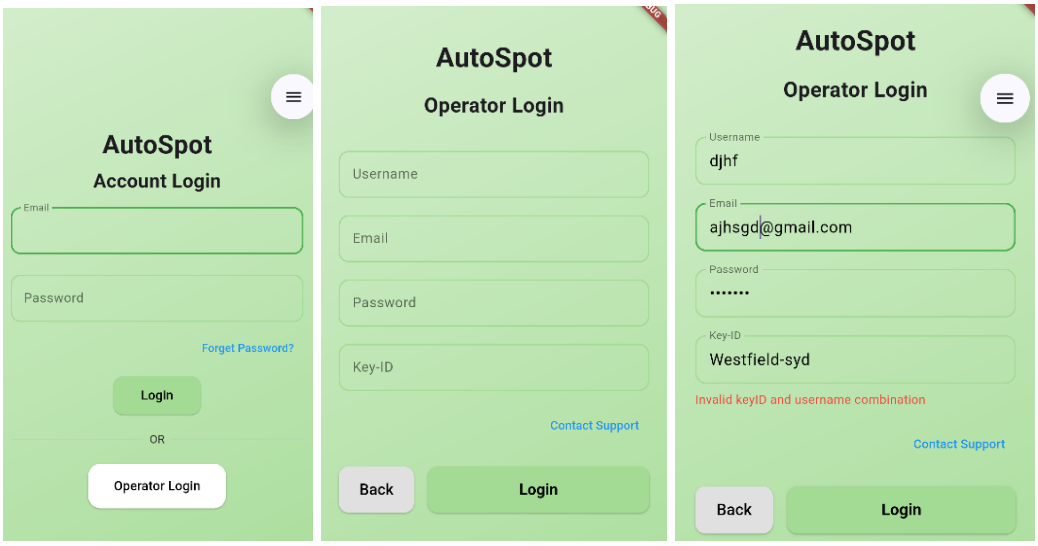
The emissions calculation system estimates the environmental impact of a full parking route (from an entrance, to the assigned slot and then to the exit) by comparing the actual distance travelled using the app as compared to a fixed baseline average distance for non-users. This feature determines the actual distance travelled through static calculations of the shortest path between a user's start and destination coordinates by Dijkstra's algorithm. While baseline distance is set to 100 meters based on data from typical cruising distances in carparks, an emission factor of 0.194g/m (based on 2024 Australian averages) is applied to both distances to calculate actual and baseline emissions with the difference representing emissions saved. This would then be returned as a percentage of total emissions saved for a full parking journey.

<u>Evaluation criteria:</u>
The effectiveness of emission calculations can be determined by accuracy of calculation, ensuring that distances and emissions are correctly calculated, values used for baseline distance and emission factor are up to date and calculation methods are transparently disclosed. Current and past emission data should also be easily retrieved without much fuss.

<u>Effectiveness Evaluation:</u>
The system meets these criteria as it provides clear and quantifiable comparisons between baseline and actual emissions and by storing results for future reference under the emissions tab. As calculations rely on a static baseline at present, future

improvements could include adopting dynamic methods and consider collecting additional user data including car models, fuel type, exact dimensions of carpark, actual travel or cruising time and distance to get a more precise estimate of emissions saved.

- **Operator Registration and Login**



This feature allows developers to create an admin (or operator) account given an employee email address and unique organisation keyID. Upon creation, the system returns a randomly generated username and password for the new account, which can then be used for login. This process is restricted to developers to prevent misuse, as regular users should not be able to create operator accounts on their own. Admins/operators would then have to use the provided username and password, as well as the organisation keyID and email address to log into their admin accounts.

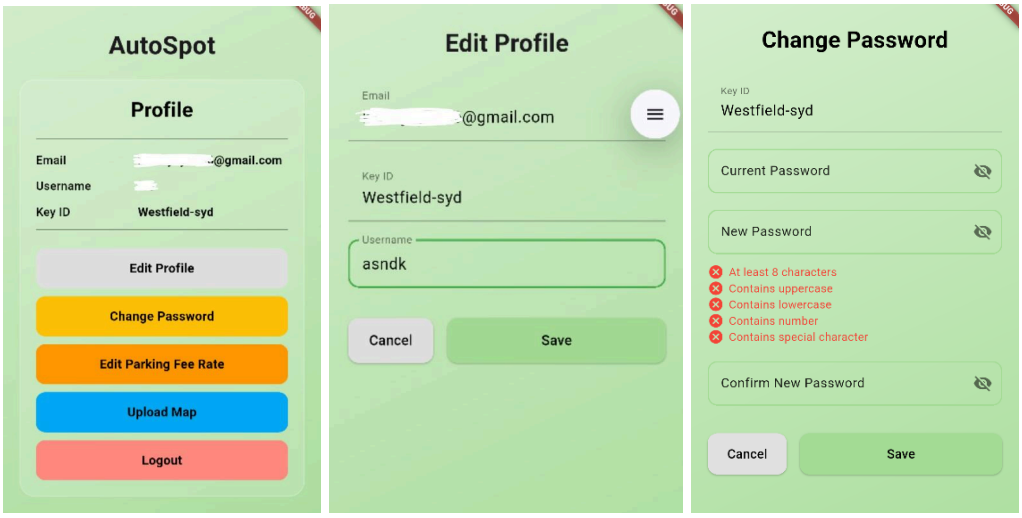
Evaluation criteria:

The effectiveness of the solution can be evaluated based on its security of creating an admin account, reliably generating unique randomised username and password without duplication, as well as usability in the admin login process.

Effectiveness Evaluation:

In this case, our implementation meets the security aspect as only developers have access to create such an account. On the frontend, the interface provides operator login options only, with no registration buttons. This prevents normal users from mistakenly gaining admin access or viewing other sensitive parking information. Username and passwords are also reliably generated to be random and unique, and the operator login page is intuitive, clearly indicating fields that are required to be filled in as well as indicating with error messages if the credentials are incorrect.

- **Operator Edit Profile & Change Password**



Similar to the user edit profile and change password functionalities, the operator edit profile and change password features allows operators to update their username, and password respectively. This provides them with the option to change their username and password to something easier to remember after receiving the initial randomly generated username and password strings.

Evaluation criteria:

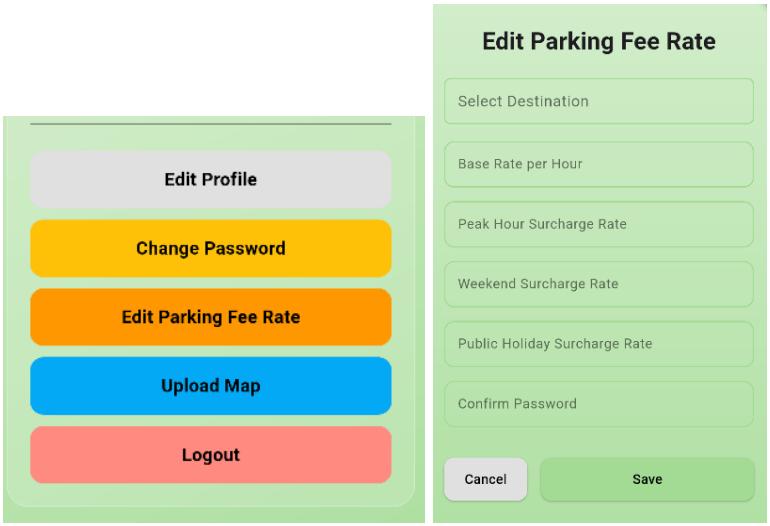
The effectiveness of this solution can be evaluated through its security, usability, and flexibility. Updated passwords should be tested for strength and stored securely. Additionally, these features should be straightforward and intuitive to use for the purpose of updating credentials. Upon successfully modifying their username or password, these changes should be reflected immediately. Finally, this feature should allow for operators to personalise their login details while keeping important account identifiers such as email and organisation keyID fixed.

Effectiveness Evaluation:

This implementation is effective in safeguarding important account identifiers while allowing for modifications to username and passwords (that were initially randomly generated) to be made. Passwords changes follow similar security and strength check mechanisms as from the user edit profile features, ensuring newly updated passwords are secure. They are also stored as hashed strings to prevent attackers from gaining easy access in cases where accounts are compromised. This thus meets the security and flexibility criteria. Furthermore, the interface is fairly straightforward with clear indications of the fields that can or cannot be edited and buttons to confirm the change. When any modifications are made, this change is immediately reflected and users would be able to log in with the new credentials. Perhaps future improvements could include password expiry which

requires operators to change their password after a set time to reduce risk of long term compromise and increase account security.

- **Operator Edit Parking Rates**



This feature allows operators to update rates for the respective carparks they are tending to.

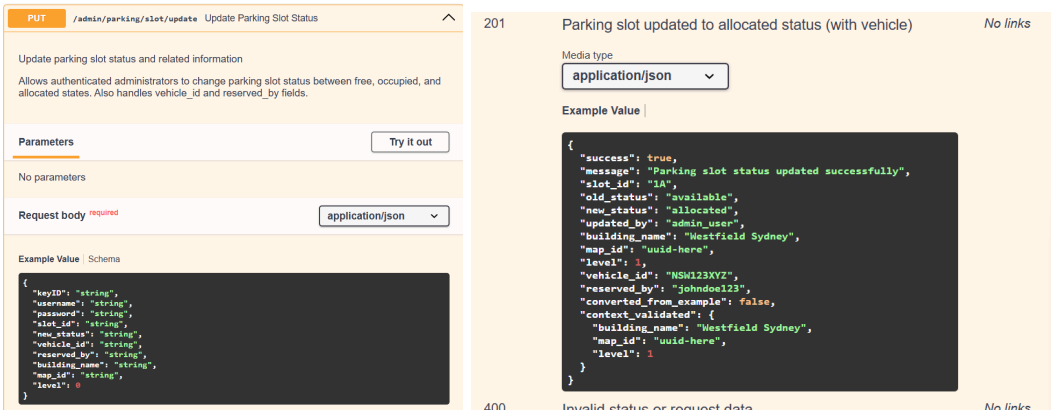
Evaluation criteria:

The Operator Edit Parking Rates feature should be intuitive, allowing operators to update rates without extensive training. It must support efficient changes, with all rate components (base rate and surcharges) editable in one place. Input validation helps prevent errors, and flexibility is provided by allowing selection of specific destinations. Accessibility, including clear labels and large buttons, further enhances usability.

Effectiveness Evaluation:

This feature delivers a straightforward and efficient process for updating parking fees. Operators can quickly locate and edit rates, with changes applied instantly in the system. In testing, navigation was smooth, and the form fields were easy to complete. Although an enhanced input validation could improve accountability and accuracy, the feature effectively meets its primary goal of providing a secure, easy-to-use tool for rate management.

- **Operator Update Parking Slot Details**



The slot management feature allows operators to modify parking slot details, including updating its status to either available, allocated or occupied. This is especially useful in special cases where:

- Case 1: A non-user parks in an assigned slot. This requires manual reallocation of the parking session.
- Case 2: A user mistakenly occupies another person's slot and has already started their session. This requires manual updates or swaps to the session details.

Hence, this feature helps ensure that parking session records are kept accurate and reduces operation disruptions.

Evaluation criteria:

The effectiveness of this feature can be evaluated by how accurately it reflects the real-time slot availability and status, whether changes made are reflected instantly, and the flexibility to manage unexpected parking situations effectively.

Effectiveness Evaluation:

The current implementation meets these criteria as it accurately updates slot status and parking session details, ensuring real-time status of slots and occupancy are stored within the system. Additionally, operators can quickly make changes to slots through a straightforward process, and these modifications would be reflected immediately within the system. Perhaps future improvements to this system could include integrating an automated anomaly detection system which identifies and flags vehicles that do not appear to match the assigned user credentials. This could potentially help reduce reliance on need for manual intervention.

F. Limitations and Future Work

1. Current Limitations

Sensor Implementation

As for now, the application operates using a ‘Trigger Parking’ button and is not yet implemented with real physical sensors. This limits the system’s ability to automatically detect vehicle occupancy in parking spaces.

Building Infrastructure Dependency

For real deployment, the system will need to integrate with the specific infrastructure of the building where it is going to be deployed. This will include wiring, network connectivity, and physical placement of sensors at each parking spot.

No Actual Data or Parking Map Testing

The parking maps used in our testing are simulated layouts designed to be visually clear and relatively simple. While these maps allow us to validate the basic functionality of our system, they do not fully replicate the complexity of larger or more intricate real-world parking layouts. This limits our ability to assess how the system performs under more challenging and realistic conditions.

No Preferred Destination Implementation

According to the initial project specifications, the system was intended to support a “preferred destination” feature, allowing users to select a specific location within a building as their target. However, implementing this proved challenging, as it would require detailed and accurate mapping of building interiors, which was beyond the scope of our current resources. As an alternative, we implemented a “preferred level” approach. In this model, users specify their desired building entrance (associated with a particular level), and the system allocates the nearest available parking slot on that level and as close as possible to the chosen entrance (if available).

Admin-Side Functionality

Due to time constraints, the development focus shifted heavily toward the user-facing features, leaving limited time for implementing the administrative side of the system. During development, we also identified that an admin interface would be more suitable in a web-based format rather than a mobile application, as originally envisioned. This change in approach will be more efficient for future scalability and ease of management.

Calculation of Carbon Emissions

Due to the limited user-provided data on vehicle types and the actual baseline travel distance (including the exact perimeter of specific carparks), the app currently estimates carbon emissions saved using a static baseline of 100m and a fixed emission factor of 0.194g/m, which is the average for registered vehicles in Australia as of Jan 2024. However, this might not reflect real-world variations as more information would be required.

Card and Payment Management

Currently, the system performs basic validation on card numbers using length checks depending on the first digit of the card, as well as the luhncheck library which uses a standard checksum method to verify that card numbers follow a valid format. However, to enable real transactions, more testing and integration of payment gateways such as Stripe and Paypal would be required to ensure that transactions are being authorised, securely saved and processed.

2. **Client Handover:**

Steps Taken:

- Throughout the term, we have engaged in consistent meetings with the client to provide updates on new features and project plans, involving them in the decision-making process and making improvements based on the suggestions provided.
- In our last meeting on **Week 10 Thursday**, we have provided a final walkthrough of all implemented features, including latest updates to subscription plans, multithreading and session management. During this time, we have also discussed the test coverage rate, how emission estimations are calculated, how real-time slot availability updates, as well as future plans to expand for use of larger-scale maps. Additionally, while deployment of the app is currently put on hold at this stage, the app can be tested locally using the emulator device.
- On **Week 12 Monday,** we are planning to have a final meeting to handover the project. During this session, we will also be demonstrating how to set up and run the app locally on both the Backend server and Frontend emulator.
- Items to prepare:
    1. Providing access to our latest version of code via the Client's Github repository
    2. README documentations for explaining how to set up both the frontend and backend interface/server locally
    3. A copy of the Project Report which contains documentation of key features and design justification
    4. A video of our final demo session in Week10
    5. A video of how to set up and run the server and emulator.

3. **Future Work**

- User Registration and Login:

  While our current registration and login process are intuitive and easy to use, perhaps one way to further improve user experience could be to develop social login integration features (linking to existing Google, Facebook, Instagram, Twitter etc. accounts). This might further simplify the login process and improve user experience when signing up or logging into the app.

- Edit Profile Details:
  While the current implementation does allow users to easily and reliably update their account details, perhaps one improvement that could be made is to integrate use of external license plate and phone number verification processes in the future to prevent users from entering fake details to bypass the system.

  Another improvement that can be made is the periodic password expiry feature for both user and operator accounts. This ensures accounts are kept secure and would potentially reduce risk of prolonged exposure of credentials.

- QR expiry mechanism:
  Although we had initially considered implementing this, this feature was put on hold due to the requirement for screens to generate and display updated QR codes. In the future, it could be introduced to enhance security and prevent the misuse of the current single QR code system for entrances and exits.

- Availability of visuals/images of parking slots in maps:
  Future improvements could include having maps with live view or images to allow users to visualise slot locations and other attributes (e.g. size, shade, EV charging compatibility).

- Parking time update/reminder
  Future improvements to calculating parking rate/sessions could include real-time updates notifying users about their extended stay.

- Option to provide more data for greater accuracy in emission estimations
  While current methods only use static modes of calculation, with fixed baseline distance and emission factor, future features catered towards more environmentally conscious users could offer more accurate dynamic calculations of estimated emissions through collection of relevant user data including car make, fuel type, exact dimensions of carpark, actual travel or cruising time and distance.

- Integration of an automated anomaly detection system for sensors
  While current methods for cases where sessions are started for the wrong user/vehicle would require human intervention to update the status and credentials saved to slots, future improvements could include integrating an automated anomaly detection system to identify and flag vehicles that might not match the credentials (e.g. license plate number) of the assigned user. This could help reduce reliance on manual intervention, thereby improving overall operational efficiency.

References:

1. Assemi, B., Baker, D., & Paz, A. (2020). Searching for on-street parking: An empirical investigation of the factors influencing cruise time. *Transport Policy*, *97*, 186–196. https://doi.org/10.1016/j.tranpol.2020.07.020

2. National Transport Commission. (2024, December 10). Light Vehicle Emissions Intensity in Australia: Trends Over Time. Retrieved from https://www.ntc.gov.au/light-vehicle-emissions-intensity-australia