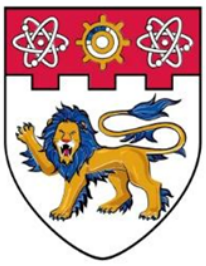


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006 - Software Engineering

Lab 4 Deliverables

Lab Group	SCED
Team	Glitch
Members	Harshil Gupta (U2421166J)
	Goh Jin Long Abdillah (U2321634L)
	Guan Yibin (U2423353E)
	Kumar Preetham (U2422986F)
	Goh Jun Xian Bryant (U2423462J)

1. Black Box Testing

I. AuthController

The AuthController class manages end-user authentication for ParkMate, specifically account creation (sign-up), login, password reset, and permanent account deletion. For login, the controller accepts an email and password, validates the input, and verifies the submitted password against the stored salted hash of the corresponding user record. If the credentials are valid and the account is not restricted, the controller issues a secure session or token and returns the user to the application's home context; on failure it returns a generic authentication error without revealing which field was incorrect. Observable behaviours include clear messages when an email is not registered, and graceful handling when the authentication service or backend is unreachable. Security-relevant aspects are part of the externally visible contract: secure session management is required, sensitive fields are never echoed in responses, and service-level errors are surfaced with user-safe wording.

For account creation, the controller validates essential profile fields and enforces email uniqueness before persisting the new user record with a hashed password. On success it creates a new user identifier and the account becomes available for subsequent login. The controller also supports user-initiated password resets by sending a reset link via email and enforcing token expiry; after a valid reset flow the user can set a new password and continue using the system. Finally, users may permanently delete their account; upon confirmation the controller removes the account and associated preferences, with the operation marked as irreversible. These behaviours are specified in ParkMate's use cases and constitute the observable interface that our black-box tests will exercise end-to-end.

II. SearchController

The SearchController class manages the discovery flow for carparks and EV charging locations in ParkMate. It accepts either a destination text query (with geocoding suggestions) or a location context ("Current Location"), validates inputs, and sets the selected place as the search destination. Once a destination is set, the controller retrieves and returns a list or map of nearby carparks around that destination. The response includes key details for each carpark such as name, distance, live lot availability, price, height limit, and the last-updated time; user interactions like selecting a marker or list item save the chosen carpark to history. When input is invalid (for example, gibberish addresses) the controller returns a clear validation message; when external services are unavailable (geocoding, carpark database) it surfaces a user-safe error and preserves app continuity where possible. These behaviours define the externally observable contract that will be verified via black-box tests.

For EV charging, the controller—working with the EV data services—fetches nearby stations and returns charger-level information including availability state, connector type, charging speed, and last-updated time. Results refresh automatically every one to two minutes or on manual refresh, with data-freshness communicated explicitly in the payload so the client can show a visible "last updated" indicator. If live feeds are temporarily unavailable, the controller returns the last known results with a clear stale-data notice. The EV experience is integrated with ParkMate's filter and sort capabilities so users can narrow by connector type and charging speed while still benefiting from deterministic, reproducible ordering in the returned list or map. These behaviours provide a stable specification for black-box validation without reliance on internal implementation details.

III. Equivalence Class and Boundary Value Testing

Equivalence Class Testing

A basic black-box test design technique in which test cases are designed to execute representatives from equivalence partitions. Equivalence classes (partitions) are portions of an input or output domain. The behavior of a component is assumed to be the same for every member of a partition class; therefore we choose at least one value from each valid class and at least one value from each invalid class to achieve good coverage with few tests. In ParkMate, we identify valid classes that the system should accept and process normally and invalid classes that the system should reject with clear messages. We apply the lecture rule of one valid EC and two invalid ECs for ranges, and one valid / one invalid EC for discrete sets, ensuring each test contains at most one invalid input so error handling can be isolated.

Boundary Value Testing

Boundary Value Testing extends equivalence partitioning when members of an input class are ordered. It focuses on edges or limits where faults are most likely to appear. For ParkMate, natural boundaries arise at empty vs non-empty text, token/expiry cut-offs, and data-freshness thresholds (e.g., EV updates every 1–2 minutes), so we design tests on, just below, and just above those limits.

1. Login Function (AuthController)

Valid Equivalence Class: Syntactically valid email paired with a password; user exists; credentials match; backend reachable → user is authenticated and navigated to Home
Invalid Equivalence Class: Malformed or unregistered email; wrong password; locked or non-existent account; backend unavailable → generic authentication error or “Unable to connect” without leaking which field failed
Boundary Values: Empty vs one-character password (empty treated as invalid input), and exactly one failed attempt before lockout policies are considered (functional behavior remains generic error as specified)

2. Sign-Up Function (Auth Controller)

Valid Equivalence Class: Name/email present in correct format; password meets policy (ideally ≥12 characters with a mix of upper/lowercase, numbers and symbols); confirm password matches; email not already registered → account created and ready for subsequent login
Invalid Equivalence Class: Malformed email; weak password that does not meet stated requirements; confirm password mismatch; duplicate email; transient network/server error → specific error messages such as “Email already in use,” “Password must meet the requirements,” or “Service unavailable.”
Boundary Values: Password strength right at the policy edge (e.g., exactly meeting vs just missing the composition guideline), and confirm-password exactly equal vs off by one character

3. Destination Search & Selection (SearchController)

Valid Equivalence Class: Non-empty, human-readable destination text or “Current Location”; geocoding API reachable; user selects one of the suggestions → destination is set and nearby carparks are shown
Invalid Equivalence Class: Gibberish/special-character input the geocoder cannot resolve; geocoding API unavailable; carpark database unreachable → “No matching locations found,” “Unable to retrieve location data. Please try again later,” or equivalent error
Boundary Values: Query length at the edge of acceptance (empty string representing “Current Location” vs one-character query), and suggestion list size at boundaries (no suggestions vs exactly one suggestion) with correct messaging and flow back to input

4. Display Nearby Carparks (SearchController)

Valid Equivalence Class: Destination selected; carpark data available → ParkMate displays nearby carparks in map or list view with name, distance, live availability, price, height limit and last-updated time; selecting an item writes to history
Invalid Equivalence Class: Carpark data service unavailable; map rendering failure → user sees “Carpark information unavailable. Please try again later” or system falls back to list view
Boundary Values: Exactly zero results vs one result displayed; last-updated timestamp at the freshness threshold still shows a valid “Last updated HH:MM” rather than an error

5. EV Charger Discovery & Live Availability (SearchController / EV features)

Valid Equivalence Class: Location or destination known; EV data source reachable; chargers exist in area → stations appear with , connector type and charging speed, and a visible last-updated time; results auto-refresh or update on manual refresh
Invalid Equivalence Class: User denies location permission request; Geolocation service fails for reasons other than permission denial;Map initialization fails to get user location;No Search Results / Invalid Address;The backend carpark database API is unreachable;The Google Maps script fails to load (e.g., no internet, API error). → "Location access denied. Please enable permissions.";"Could not get your location. Please try again.";"Could not get your location. Using default location.";"Invalid input. Please enter a valid address.";"Carpark database not reachable".;"Failed to load Google Maps. Please check your API key and internet connection"
Boundary Values: Data-freshness exactly at the 1–2 minute update window (just before and just after the threshold); manual refresh immediately after auto-refresh; zero available connectors transitioning to one available connector displays the correct state and ordering

6. EV Filters: Connector Type & Charging Speed

Valid Equivalence Class: Connector types drawn from the supported set (e.g., Type 2, CCS2, CHAdeMO) and a recognised charging-speed category; applying filters narrows results without excluding valid matches
Invalid Equivalence Class: Unsupported connector label; contradictory filters that yield no possible results → empty state with clear guidance to adjust filters
Boundary Values: Toggling a single connector type vs the full set; clearing all filters vs applying exactly one filter; last-updated carried through filtered lists at the freshness edge

IV. Test Cases and Results

a) Login

Input parameters: Email and Password

No.	Test Input	Expected Output	Actual Output	Pass?
1.	(Valid) Email: user1@parkmate.app (Valid) Password: valid hash match	Successful login; session/token issued; navigate to Home	Successful login; session/token issued; navigate to Home	Yes
2.	(Invalid) Email: “” (Valid) Password: ValidPass!1234	Login failed; system notifies “Please enter your email and password.”	Login failed; system notifies “Please enter your email and password.”	Yes
3.	(Invalid) Email: test123 (Valid) Password: ValidPass!1234	Login failed; system notifies “Please enter a valid email.”	Login failed; system notifies “Please enter a valid email.”	Yes
4.	(Valid) Email: user1@parkmate.app (Invalid) Password: “”	Login failed; system notifies “Please enter your email and password.”	Login failed; system notifies “Please enter your email and password.”	Yes
5.	(Valid) Email: user1@parkmate.app (Invalid) Password: wrong secret	Login failed, system notifies “Invalid email or password.” (generic, no field leakage)	Login failed, system notifies “Invalid email or password.” (generic, no field leakage)	Yes
6.	(Invalid) Email: ghost@parkmate.app (not registered) (Valid) Password: any	Login failed, system notifies “Invalid email or password.”	Login failed, system notifies “Invalid email or password.”	Yes
7.	(Valid) Email: user1@parkmate.app (Valid) Password: valid (Service Down)	Login failed, system notifies “Unable to connect. Please try again later”	Login failed, system notifies “Unable to connect. Please try again later”	Yes

b) Signup

Input parameters: Name, Email, Password, Confirm Password

No.	Test Input	Expected Output	Actual Output	Pass?
1.	(All Valid Inputs) Name: “Zerius Heng” Email: zerius0139@gmail.com Password: ParkMate!2025 Confirm Password: ParkMate!2025	Successful signup; useID created, account can log in thereafter	Successful signup; useID created, account can log in thereafter	Yes
2.	(All valid except Name) Name: “”	System notifies “Name is required.”	System notifies “Name is required.”	Yes
3.	(All Valid except Email) Email: “”	System notifies “Please enter a valid email.”	System notifies “Please enter a valid email.”	Yes
4.	(All Valid except Password) Password: “”	System notifies “Please fill in all required fields.”	System notifies “Please fill in all required fields.”	Yes
5.	(Weak Password) Password: password Confirm: password	System notifies “Password does not meet the requirements.”	System notifies “Password does not meet the requirements.”	Yes
6.	(Mismatch Password) Password: ParkMate!2025 Confirm: ParkMate!2024	System notifies “Passwords do not match.”	System notifies “Passwords do not match.”	Yes
7.	(Duplicate Email) Email already registered	System notifies “Email already in use.”	System notifies “Email already in use.”	Yes
8.	(Valid Inputs) with transient DB error	System notifies “Service unavailable. Please try again later.”	System notifies “Service unavailable. Please try again later.”	Yes

c) Destination Search (Set Destination)

Input parameters: Text query or Current location

No.	Test Input	Expected Output	Actual Output	Pass?
1.	(Valid) Query: “NTU North Spine”	Geocoding suggestions shown; user selects a suggestion; destination set	Geocoding suggestions shown; user selects a suggestion; destination set	Yes
2.	(Valid) Use “Current Location” (permission granted)	Destination set to current coordinates	Destination set to current coordinates	Yes
3.	(Invalid) Query: “#@$$%”	System notifies “No matching locations found.”	System notifies “No matching locations found.”	Yes
4.	(Service Down) Geocoding unavailable	System notifies “Unable to retrieve location data. Please try again later.”	System notifies “Unable to retrieve location data. Please try again later.”	Yes
5.	(Permission Denied) Current Location denied; empty query	System prompts to enter a destination text	System prompts to enter a destination text	Yes

d) Display Nearby Carparks (List/Map)

Precondition: Destination set successfully

No.	Test Input	Expected Output	Actual Output	Pass?
1.	(Valid) Destination set to “NTU North Spine”	Nearby carparks displayed with name, distance, price, height limit, live availability, last updated.	Nearby carparks displayed with name, distance, price, height limit, live availability, last updated.	Yes
2.	(Zero Results) Area with no carparks	System shows “No nearby carparks found” with the option to adjust the destination	System shows “No nearby carparks found” with the option to adjust the destination	Yes
3.	(Data Stale) Provider returns cached data	Results shown with “Last updated…” freshness indicator; no crash	Results shown with “Last updated…” freshness indicator; no crash	Yes
4.	(Select Item) Tap marker/list item	Carpark details open and saved to history	Carpark details open and saved to history	Yes
5.	(Service Down) Carpark data API unavailable	System notifies “Carpark information unavailable. Please try again later.”	System notifies “Carpark information unavailable. Please try again later.”	Yes

e) EV Charger Discovery

Input parameters: Destination/location context (as above)

No.	Test Input	Expected Output	Actual Output	Pass?
1.	(Valid) Destination with chargers	Show stations, connector type, charging speed, and last updated	Show stations, connector type, charging speed, and last updated	Yes
2.	(Auto Refresh) Wait 1 minute	Data refreshes; last updated time changes accordingly	Data refreshes; last updated time changes accordingly	Yes
3.	(No Chargers) Area without chargers	The system shows “No EV chargers found nearby” guidance	The system shows “No EV chargers found nearby” guidance	Yes
4.	(Feed Down) EV data source offline	Show last known results with a clear notice “Live data temporarily unavailable.”	Show last known results with a clear notice “Live data temporarily unavailable.”	Yes
5.	(Manual Refresh) Pull to refresh immediately after auto refresh	No errors; consistent latest timestamp	No errors; consistent latest timestamp	Yes

f) EV Filters: Connector Type & Charging Speed

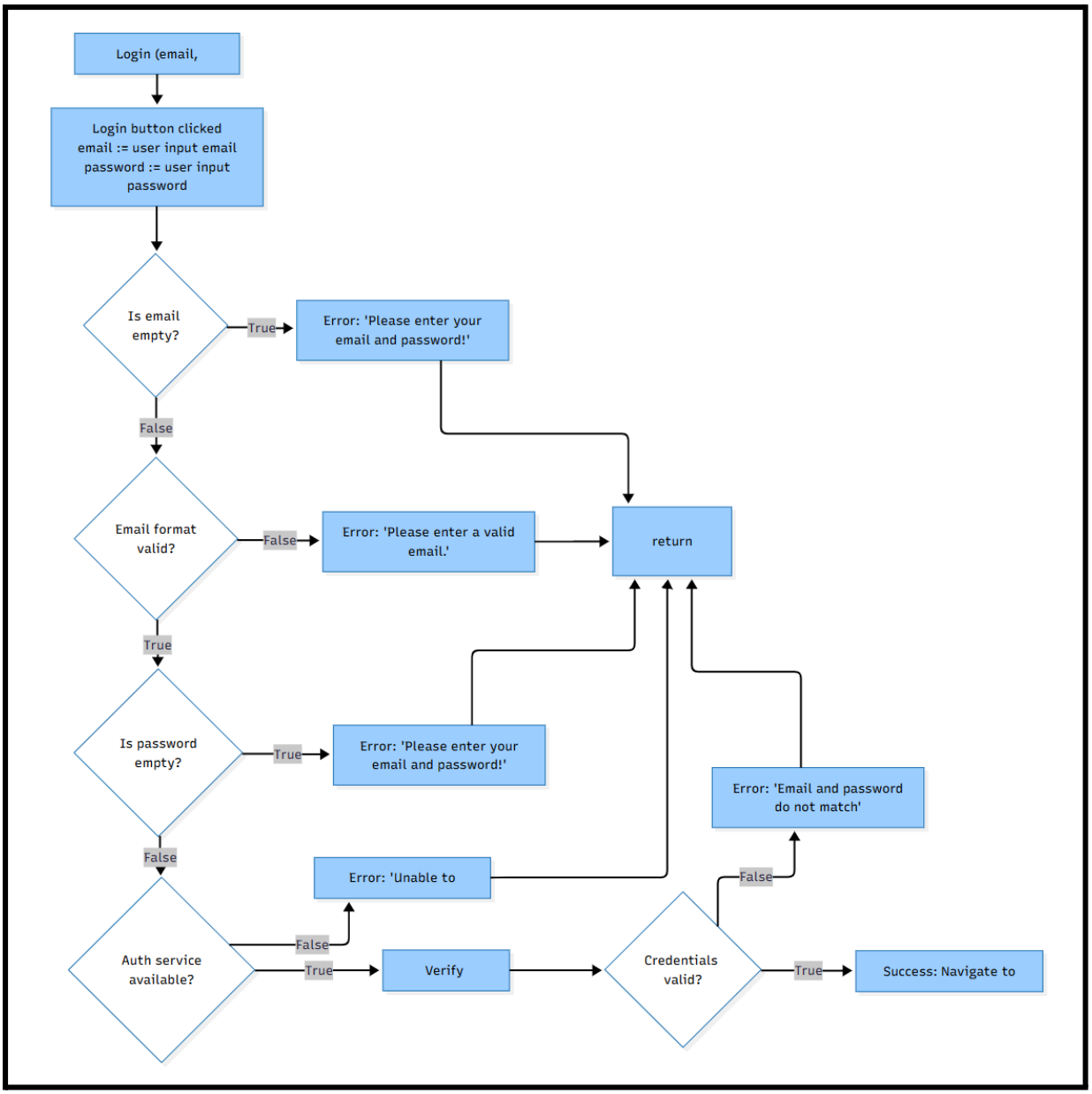
Input parameters: connectorTypes, chargingSpeed, requireAvailable

No.	Test Input	Expected Output	Actual Output	Pass?
1.	(Valid) Connector Types: {Type 2, CCS2} Charging Speed: “Fast”	List/map shows only matching stations; last update retained	List/map shows only matching stations; last update retained	Yes
2.	(Valid) Require Available: true	Results show only stations with at least one available connector	Results show only stations with at least one available connector	Yes
3.	(Contradictory) Filters yield empty set	Empty state with guidance to clear or adjust filters	Empty state with guidance to clear or adjust filters	Yes
4.	(Preference Memory) Apply filter, navigate away, return	Prior EV filter preferences are remembered and applied	Prior EV filter preferences are remembered and applied	Yes

2. White Box Testing

I. Login

a. Control Flow Graph



b. Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = 5 + 1 = 6

Basis Paths:

1. Baseline path: 1 → 2 → 3(F) → 5(T) → 7(F) → 9(T) → 11 → 12(T) → 15
2. Basis path 2: 1 → 2 → 3(T) → 4 → 14
3. Basis path 3: 1 → 2 → 3(F) → 5(F) → 6 → 14
4. Basis path 4: 1 → 2 → 3(F) → 5(T) → 7(T) → 8 → 14
5. Basis path 5: 1 → 2 → 3(F) → 5(T) → 7(F) → 9(F) → 10 → 14
6. Basis path 6: 1 → 2 → 3(F) → 5(T) → 7(F) → 9(T) → 11 → 12(F) → 13 → 14

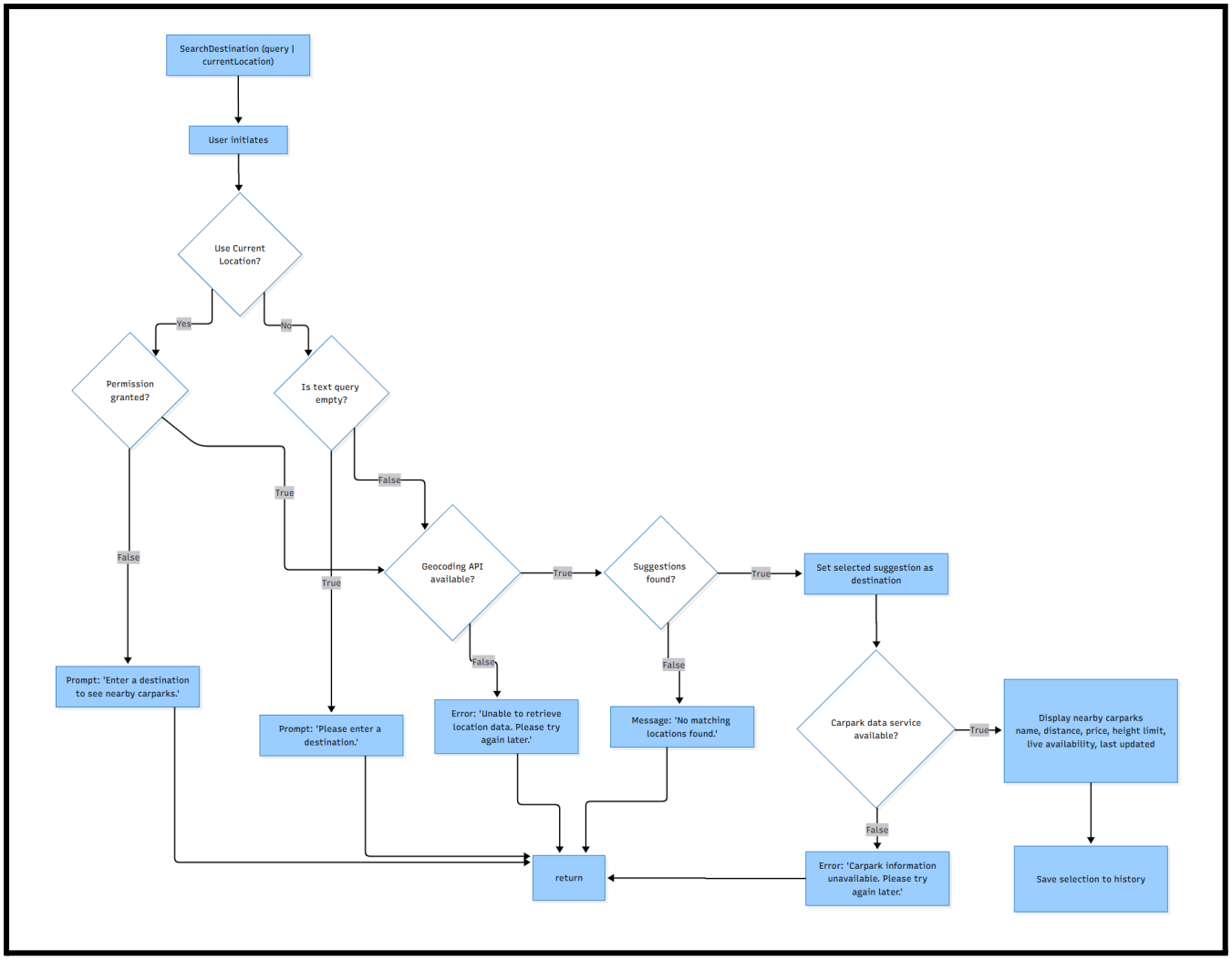
c. Test Cases and Results

Login(email, password)

No.	Test Input	Expected Output	Actual Output	Pass?
1.	email = user1@parkmate.app password = Correct#2025	Log in successfully → navigate to Home	Log in successfully → navigate to Home	Y
2.	email = “” password = Correct#2025	“Please enter your email and password!”	“Please enter your email and password!”	Y
3.	email = admin1 password = Correct#2025	“Please enter a valid email.”	“Please enter a valid email.”	Y
4.	email = admin1@ password = Correct#2025	“Please enter a valid email.”	“Please enter a valid email.”	Y
5.	email = user1@parkmate.app password = “”	“Please enter your email and password!”	“Please enter your email and password!”	Y
6.	email = ghost@parkmate.app password = Any#1234	“Email and password do not match”	“Email and password do not match”	Y
7.	email = user1@parkmate.app password = Correct#2025 **Auth service down**	“Unable to connect”	“Unable to connect”	Y

II. Destination Search & Nearby Carparks

a. Control Flow Graph



b. Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = 6 + 1 = 7
(Decisions at D3, D3a, D4, D5, D6, D8)

Basis Paths:

1. Baseline path: 1 → 2 → 3(No) → 4(F) → 5(T) → 6(T) → 7 → 8(T) → 9 → 10
2. Basis path 2: 1 → 2 → 3(No) → 4(T) → 4e → 14
3. Basis path 3: 1 → 2 → 3(No) → 4(F) → 5(F) → 5e → 14
4. Basis path 4: 1 → 2 → 3(No) → 4(F) → 5(T) → 6(F) → 6e → 14
5. Basis path 5: 1 → 2 → 3(Yes) → 3a(F) → 3e → 14
6. Basis path 6: 1 → 2 → 3(Yes) → 3a(T) → 5(F) → 5e → 14
7. Basis path 7: 1 → 2 → 3(Yes) → 3a(T) → 5(T) → 6(T) → 7 → 8(F) → 8e → 14

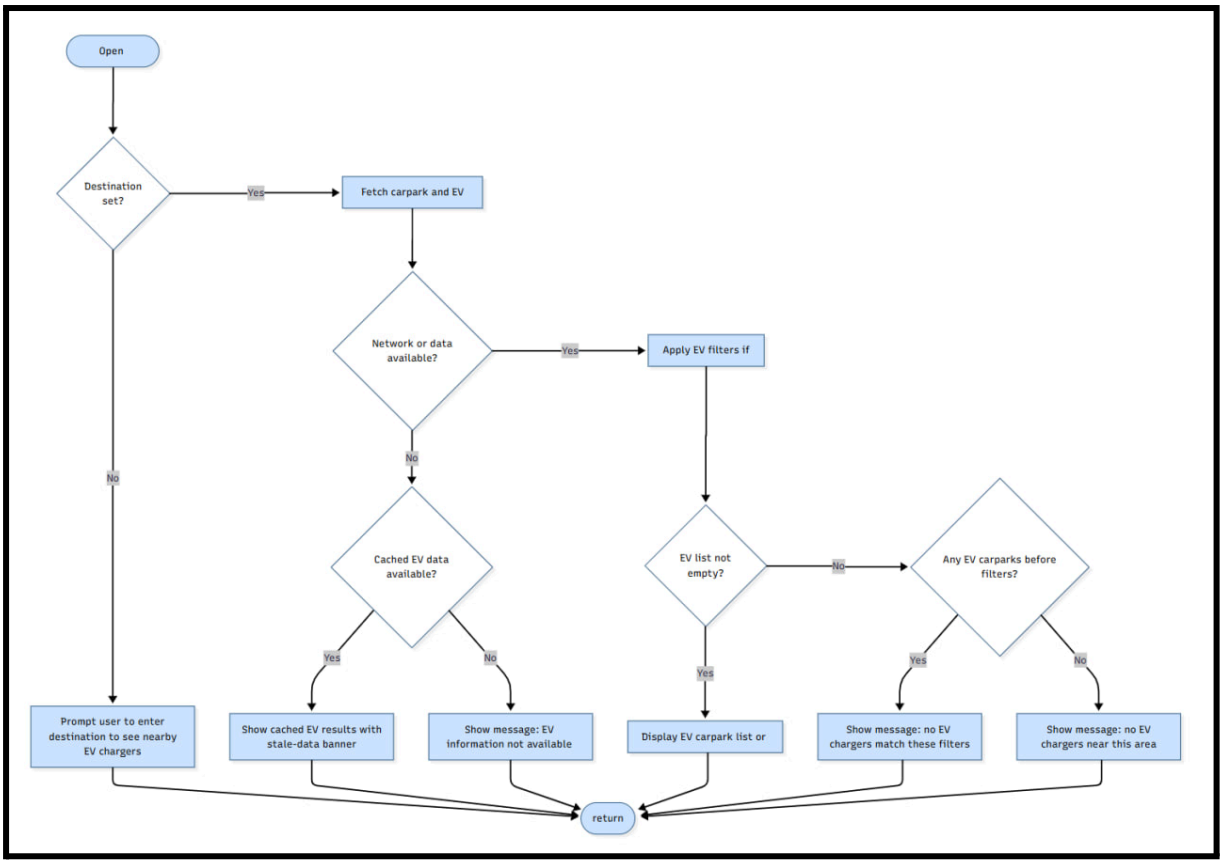
c. Test Cases and Results

SearchDestination(query | currentLocation)=

No.	Test Input	Expected Output	Actual Output	Pass?
1.	Query mode: “NTU North Spine”	Suggestions shown → user selects → destination set → nearby carparks displayed with name, distance, price, height limit, live availability, last updated; selection saved to history	Suggestions shown → user selects → destination set → nearby carparks displayed with name, distance, price, height limit, live availability, last updated; selection saved to history	Y
2.	Query mode: “”	Prompt “Please enter a destination”	Prompt “Please enter a destination”	Y
3.	Query mode: “@@@###”	“No matching locations found”	“No matching locations found”	Y
4.	Query mode: “NTU North Spine”, Geocoding down	“Unable to retrieve location data. Please try again later.”	“Unable to retrieve location data. Please try again later.”	Y
5.	Current Location mode (permission granted)	Destination set to device location → nearby carparks displayed with full details	Destination set to device location → nearby carparks displayed with full details	Y
6.	Current Location mode (permission denied), no query	Prompt “Enter a destination to see nearby carparks.”	Prompt “Enter a destination to see nearby carparks.”	Y
7.	Destination set, Carpark data service unavailable	“Carpark information unavailable. Please try again later.”	“Carpark information unavailable. Please try again later.”	Y

III. EV Carpark Results and Filters

a. Control Flow Graph



b. Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = 5 + 1 = 6
(Decisions at 2, 5, 6, 10, and 12)

Basis Paths:

1. Baseline path: 1 → 2(T) → 4 → 5(T) → 9 → 10(T) → 11 → 15
2. Basis path 2: 1 → 2(F) → 3 → 15
3. Basis path 3: 1 → 2(T) → 4 → 5(F) → 6(T) → 7 → 15
4. Basis path 4: 1 → 2(T) → 4 → 5(F) → 6(F) → 8 → 15
5. Basis path 5: 1 → 2(T) → 4 → 5(T) → 9 → 10(F) → 12(F) → 14 → 15
6. Basis path 6: 1 → 2(T) → 4 → 5(T) → 9 → 10(F) → 12(T) → 13 → 15

c. Test Cases and Results

No.	Test Input	Expected Output	Actual Output	Pass?
1.	Valid destination; network/data source	Carparks with EV chargers are rendered	Carparks with EV chargers are	Y

	available; area has EV-capable carparks	with connector type, charging speed category, and any available EV details.	rendered with connector type, charging speed category, and any available EV details.	
2.	No destination and location permission denied	Prompt “Enter a destination to see nearby EV chargers”	Prompt “Enter a destination to see nearby EV chargers”	Y
3.	Valid destination; EV-related data source down; cached data exists	Show cached results (if any) with banner “EV data temporarily unavailable — showing last update at HH:MM.”	Show cached results (if any) with banner “EV data temporarily unavailable — showing last update at HH:MM.”	Y
4.	Valid Destination; no chargers in area	Message “No EV chargers found near this area.” with guidance to expand radius, pan map, or clear filters	Message “No EV chargers found near this area.” with guidance to expand radius, pan map, or clear filters	Y
5.	Active EV filters (e.g., Type 2 + DC Fast) applied; data is reloaded (manual refresh or user revisits screen)	Results remain filtered after data reload; only matching EV-capable carparks appear; any “last updated” label reflects the new fetch time if available.	Results remain filtered after data reload; only matching EV-capable carparks appear; any “last updated” label reflects the new fetch time if available.	Y

3. Demo Script

Yibin

Slide 1:

Ok, we are the team behind ParkMate — an all-in-one solution for EV drivers in Singapore. We'll introduce how ParkMate helps drivers to save time more efficiently.

Slide 2:

Our flow of presentation would like as such. (Point)

Slide 4:

Finding a place to park or charge your EV often feels like a game of luck — pots may be overpriced, full, or even incompatible with your vehicle.With the number of EVs increasing each year, this problem is only getting worse. Existing parking apps treat charging as an afterthought, leaving users with fragmented, unreliable information and no real-time visibility.
SO why our app ? Because park mate is able to resolve the issues, it gives live updates on parking availability, nearby charging ports, rates, and connector types.
This helps drivers make faster, smarter parking decisions — reducing stress, traffic congestion, and range anxiety — all within one seamless interface.

Slide 5:

We created a unified platform where EV users can rely on up to date information on both parking and charging needs. By integrating data sources and live feeds. Parkmate enhances the user convenience and supports Singapore's EV-friendly ecosystem.

Slide 7:

Carpark Search:
Live updates on parking spots near the user, including cost and availability.
Assisted Navigation:
Once a spot is chosen, tapping Navigate opens an external map for guided directions.
Filtering & Preferences:
Users can customize searches by charger type, sheltered parking, or price range.
Although, these features ensure ParkMate truly satisfies the needs of EV drivers.

Slide 8:

Heres the use-case diagram showing how users interact with the app. It highlights the relationship between the driver, the system backend, and external APIs that provide live data.

Slide 9:

Our data comes from reliable government sources:
For HDB carpark availability and infomation it comes from data.gov.sg
And for EV charging point dataset, it comes from the LTA datamall. And we use google maps API for navigation.

Slide 10:

And this is our tech stack, the front end is built with react + vite + typescript, the backend uses Node.js + Express + TypeScript, and the database layer runs on PostgreSQL with REDIS for caching. This architecture enables scalability, speed, and smooth user experience.

Harshil (11-18)

Slide 12:
We designed ParkMate around four principles.
SRP: each feature—auth, carparks, EV—has its own controller, service, and adapter.

OCP: new providers or filter rules plug in without changing core code.
ISP: clients call small, purpose-built interfaces only.
DIP: controllers depend on abstractions, so providers and loggers are swappable. Result: faster changes, safer releases.

Slide 13:
On the frontend we use a feature-based structure—auth, carparks, EV—each self-contained. Shared UI and API logic live in reusable *services* and *components*. Pages handle routing only; feature logic stays inside modules. This makes new features plug-in simple and keeps builds fast with Vite and hot-reload for quick dev feedback.

Slide 14:
On the backend we use a modular structure—each feature (auth, carpark, favorites) owns its routes, controller, and service. External data is isolated under 'integrations' for LTA DataMall, EMA EVSE, and OneMap. Postgres handles persistence, Redis caches live EV/carpark data. Shared middleware gives request validation, auth, and logging. This makes new modules plug-in cleanly and scale.

Slide 15:
Here's our layered architecture: the Presentation layer (MainUI, MapView, EVDashboard) talks to feature Controllers, which orchestrate domain objects like User, Carpark, and EVCharger. All persistence goes through a single database, while an ExternalAPIController isolates LTA/EMA/OneMap calls. This keeps UI clean, logic modular, and data access consistent.

Slide 16:
On efficiency, we cap render load by deterministically down-sampling map markers. This line computes a `step = ceil(filteredEvChargers.length / maxMarkers)` and only renders indices where `index % step === 0`, so 3,000 points become ~500. That keeps panning smooth and frame rates stable across zoom levels.

Slide 17:
Here's our robustness guarantee: every write runs inside a transaction—`BEGIN`, app logic, then `COMMIT`. If anything throws, we hit `ROLLBACK` immediately to avoid partial writes and keep data consistent. Finally, `client.release()` always returns the connection to the pool, preventing leaks under failure.

Slide 18:
Our validation is centralized and reusable. Each Joi schema—like `registerSchema`, `loginSchema`, `forgotPasswordSchema`—is exported once and consumed by routes and tests, so rule changes happen in a single file (DRY). This separation from business logic keeps code cleaner, reduces bugs, and makes future fields easy to add.

Slide 19:
Reliability here is enforced by a **single submission gate**. The form gathers all field errors, then only returns `true` when `Object.keys(errors).length === 0`. That means we

never send partial or dirty data to the server—users fix everything first, reducing retries and preventing inconsistent state.

Slide 20:
Security is enforced end-to-end: we verify credentials with **bcrypt** (salted hashing), return **generic errors** so attackers can't enumerate emails, and issue **access + refresh tokens** on success. Most importantly, we **strip `passwordHash` server-side** before serialization, so sensitive fields never leave the backend."

Bryant (21-26)
Now ill be showing the traceability of our functional requirements.

We will be tracing our functional requirement number 8. Which is the filtering of the EV connector type and charging speed. This functional requirement basically covers the function of filtering the chargers and parking spots based on the charger types and charging speed.

And it is dependent on FR 1-1 and 1-2 where the user's current location/search is of a valid location/destination. As well as the fact that the EV charger data is available.

This is our use case diagram

And zooming into our functional requirement, we first have to go through two use cases:

parking search and selection and display nearby carparks..

As you can see, the live ev charger availability also comes with EV filters, charging cost estimation and smart charging recommendations,

And here is our entity class diagrams, with the entities that are involved in the use case, mainly the EV charger availability, ev charger station and carpark.

Now moving on to our sequence diagram,

From here we can see that the user selects the EV mode to enter the EV map screen.

From there, they press a method to search for EV carparks will be called, which will then activate the EV controller.

Which will then get nearby carparks and retrieve the locations of the charging ports.If there is an error regarding the EV charger information fetching, an error will be displayed

Here is our dialog map, where you can see the process of getting to the 'Connector type and charging speed'

ill now be passing the time over to preetham to cover the testing.

Preetham (26-38)

Alright next up we'll look at our testing procedure. For this we used the 2 main testing paradigms black box and white box. So for this we split it into 5 main stages.

And we came up as many possible test cases.

1.Checking credentials

2. Validity of Destinations

3. Validity of Carpark Details

4. Testing for EV Display

5. Testing for EV Filters

Awesome, hope you liked the live demo. Now for the way ahead. Here are 3 improvements we can make.

Improvement 1: Integrate the navigation into the app itself for the user to have a more seamless experience.

Improvement 2: Incorporate AI into helping the users compare the options around them, listing out the pros and cons of each charger

Improvement 3:Further expansion of our app to be used in carplay for convenience, and increasing safety since the users will not have to use their phones

Our ParkMate testing strategy begins with comprehensive black box testing, where we evaluate the system from a user's perspective without examining the internal code.

We've organized our black box tests into five key stages: User Authentication, Destination Search, Carpark Display, EV Display, and EV Charger functionality. Each stage ensures our features work correctly from the user's viewpoint.

For User Authentication, we tested everything from valid signups to edge cases like weak passwords and duplicate emails. Our Destination Testing verified geocoding accuracy and error handling for invalid queries. The Carpark Display tests confirmed that users see accurate information including availability, pricing, and real-time updates. For EV features, we validated charging station displays, filter functionality, and live availability data—ensuring EV drivers get reliable, current information

Next, we performed white box testing, examining the internal logic and code paths to ensure robust implementation.

Our Login flow diagram shows every decision point and validation step, from empty field checks to authentication service failures. The Destination Search flowchart maps permission

handling, geocoding logic, and API error recovery. Finally, our EV Live Availability diagram demonstrates how we handle data source failures, implement caching strategies, and gracefully manage service outages, ensuring users always receive helpful feedback even when external services are down.

Together, these testing approaches ensure ParkMate delivers a reliable, user-friendly parking experience.

Abdi (live demo)

“Good afternoon, I will be doing the live demo of ParkMate. Parkmate combines real-time data, smart filters, and Google Maps integration for a complete parking experience.”

1. Authentication (1.5 min)

“Let’s start with user authentication.”

Action: Click Register

“Our registration flow has:

Frontend + backend validation

Password hashing with bcrypt

JWT access + refresh tokens”

Fill form:
demo@parkmate.com / SecurePass123 → Register

“Once done, the user is saved in PostgreSQL, tokens are issued, and Redux stores your session.”

Action: Logout → Login → enter same credentials → Login

“The token auto-refreshes using Axios interceptors, so users never get logged out mid-session. And we’ve got a secure password reset flow via email tokens too.”

Action: Implements forgot password flow

We use the node mailer library which enables smtp protocols between gmail and the recipient's email server, triggering the reset password link in the recipient's email.

2. Interactive Map & Geolocation (2 min)

“After login, we’re brought to our main dashboard powered by Google Maps API.”

Action: Show map and move around the map hovering the variety colours of marker

“It visualizes 1,138 carparks with color-coded markers:
🟢 High availability, 🟡 Medium, 🔴 Low, ⚪ No data.”

Action: Click geolocation icon

“Clicking this button uses the browser’s Geolocation API — centers the map on you, adds a blue marker, and sorts carparks by distance.”

Action: Click marker

“Each marker opens an info window showing availability, total lots, type of carpark, height limit and refreshes every 60 seconds.
We also use marker clustering for performance and good UI practice.”

3. Smart Search & Autocomplete (1.5 min)

“Next — the search bar. I’ll type ‘Bedok’.”

Action: Type “Bedok Central”

“Our smart search system helps users find carparks quickly by name or address.
As I type, the system searches our database of 1138 carparks in real time, return matches by address using PostgreSQL ILIKE queries.
For example, typing Bedok will show carparks with "Bedok" in their address
After that, the map will center itself to the marker and it will load the current availability of the carpark and other information.

4. Advanced Filters (2 min)

“Now let’s navigate the filters displayed as the navigation tabs at the bottom of the app. We have nearest, price, availability, and favourites filters.”

Action: Click on all the filters at the bottom nav.

As you can see, the results as pressed will be returned and it is unique to the user and stored in the cache, if the user logs back in, it will still be there.

5. Favourites (1.5 min)

“Users can save their go-to carparks.”

Action: Click marker → star icon

“This sends a POST request, saves to PostgreSQL, and updates Redux instantly. It’s an optimistic update — so it feels instant.”

Action: Open Favourites tab

“Here’s the saved list — real-time data, quick access buttons, and syncs across devices.”

Action: Unheart one favourite

“Removing works the same way — DELETE request, immediate UI update.”

6. EV Charger Mode (1.5 min)

“Now for EV owners — toggle this.”

Action: Turn on EV Mode

“The theme switches to teal and lime, and the map now shows only carparks with EV chargers.”

Action: Click an EV marker

“We display number of chargers, type, speed, and price.
Plus, EV-specific filters — by plug type, speed, and network.”

“All theme changes are handled with MUI ThemeProvider and Redux state, updating the entire UI in under 100ms.”

Action: Toggle back off

7. Google Maps Directions (1 min)

“Finally — navigation.”

Action: Click marker → Get Directions

“This opens Google Maps in a new tab using a generated URL with coordinates — giving real-time traffic, route options, and ETA.”

“Users can switch to walking, cycling, or public transport modes too.”