PARK IN PEACE

Class Diagrams

Sequence Diagrams

Dialog Maps

Revision History

rev	changes	author
20230924001	*Add Dynamic and Conceptual Models*	Isaac Chun
20230922001	*Identify classes and update use cases*	Isaac Chun
20230914001	*Initial version for Lab 2*	Software Squad Team

Table of Contents

Functional Requirements

F1 The user must be able to search for car parks near a location.

- **F1.1** The system must support searching by a valid Singapore postal code.
- **F1.2** The system must support searching by building/street name.
- **F1.3** The system must support searching by full Singapore mailing address.
- **F1.4** The system must support searching by current user location.
- **F1.5** The system must support searching through the use of saved locations such as
  - **F1.5.1** Frequented locations
  - **F1.5.2** Custom locations, such as “Home” and “Work”
- **F1.6** The system must reject any address/location not within mainland Singapore, and allow the user to pick again.

F2 When searching, the system must present the closest car parks to the destination searched within a preset search radius.

- **F2.1** If there are no car parks within the search radius, the search must instead return up to 3 nearest car parks closest to the location regardless of the price.
- **F2.2** For each subsequent result beyond the first, if the next best car park by distance is further than 1km beyond the distance of the first car park to the desired location, they must be omitted.

F3 The system must visually identify the location indicated in **(F1)** via UI elements such as drop pins.

F4 The system must visually identify the locations of the nearest car parks on the map via UI elements such as drop pins.

- **F4.1** The drop pin must be greyed out in the case that there are no vacant lots available at the indicated car park.
- **F4.2** The drop pin must display full information about the car park, as elaborated in **(F5)**, when tapped on.

F5 When returning the list of search results, the system must include the following information about each car park.

- **F5.1** The name of the car park.
- **F5.2** The estimated distance of the car park from the desired location entered, in metres.

F5.3 The price of parking at the location, in dollars per hour.

F5.4 The number of vacant lots in the car park, in real time.

F5.4.1 The lots displayed must correspond to the user's current vehicle type, if such information had been shared prior by the user.

F5.5 The estimated travelling time to reach the car park by car, assuming ideal traffic conditions.

F5.6 The projected number of vacant lots in the car park at the expected time of arrival, assuming **(F5.5)** and that the user departs immediately; with the use of historical data.

F5.6.1 The lots displayed must correspond to the user's current vehicle, if such information had been shared prior by the user.

F5.7 The special features of the car park, if any, including but not limited to:

F5.7.1 Electric vehicle charging lots

F5.7.2 Vehicle washing bays

F6 The user must be able to set their global/long-term user preferences.

F6.1 The user must be able to specify their vehicle type, such as

F6.1.1 Car

F6.1.2 Motorcycle

F6.1.3 Heavy vehicle

F6.1.4 (Electric Vehicle)

F6.2 The user must be allowed to save custom locations.

F6.2.1 Each location must be a valid location within Singapore

F6.2.2 Each location must have a custom name given by the user, such as "Home" and "Work"

F7 The user must be able to add filters to the search.

F7.1 The following filters must be implemented

F7.1.1 Pricing

F7.1.2 Preferred distance from destination (preset search radius)

F7.1.3 Mode of payment

F7.1.4 Vehicle Type

F7.1.5 Minimum vacant lots

F7.2 Results that do not match the filter criteria must be hidden.

F7.3 The user must be able to combine multiple filters.

F7.4 The user must be allowed to clear a filter.

F7.5 The user must be allowed to clear all filters at once.

F8 The user must be allowed to define a sorting criteria.

F8.1 The following sorting criteria must be made available.

F8.1.1 Sorting by price – the system must sort by price followed by distance.

F8.1.2 Sorting by distance – the system must sort by distance followed by price.

F8.1.3 If unspecified, the system should default to **(F8.1.1).**

F9 The system must redirect the user to Google Maps for directions.

F9.1 The system should support choosing between opening Google Maps on a web browser or via the installed app.

F10 The system must be able to store the previous search history of the user.

F11 The system must be able to store the user's frequented car parks.

F12 While travelling, the system must automatically monitor the availability of the chosen carpark.

F12.1 If the car park becomes full, the system must automatically select the next best car park based on the preferences selected by the user.

F12.1.1 The system must notify the user through a notification bubble that must include

F12.1.1.1 The information that the car park being full.

F12.1.1.2 The location of the next best car park.

F12.1.1.3 A button to get directions to the new car park on Google Maps.

Non Functional Requirements

Usability

- **N1** Minimal size for buttons and icons should be 48px by 48px.
- **N2** The contrast ratio between the buttons, icons, text and background must be at least 4.5:1.
- **N3** The map should be full-screen.

Reliability

- **N4** The system must achieve a minimum SLA of 0.001% uptime.

Performance

- **N5** The system must return the search results within 1 second.
- **N6** The system must display nearby car parks on the map within 1 second.

Supportability

- **N7** Documentation
  - **N7.1** Comprehensive documentation of the parking app's architecture, components, and APIs should be available
  - **N7.2** How-to guides and FAQs must be provided for users and be easily accessible for users' reference.
  - **N7.3** All documentation must be regularly reviewed and updated every 6 months to reflect any changes in the application.
- **N8** Monitoring and Logging
  - **N8.1** Robust monitoring and logging systems must be implemented to track the app's performance and usage.
  - **N8.2** Important events, errors, and users actions must be logged for troubleshooting and auditing purposes.
- **N9** User Communication:
  - **N9.1** Develop a strategy for proactively communicating with users about maintenance windows, updates, and any disruptions in service.
  - **N9.2** Implement a user-friendly mechanism for users to report issues and provide feedback. This can include:
    - **N9.2.1** In-app feedback forms or buttons.
    - **N9.2.2** A dedicated support email address.
    - **N9.2.3** Social media channels for support and feedback.

Scalability

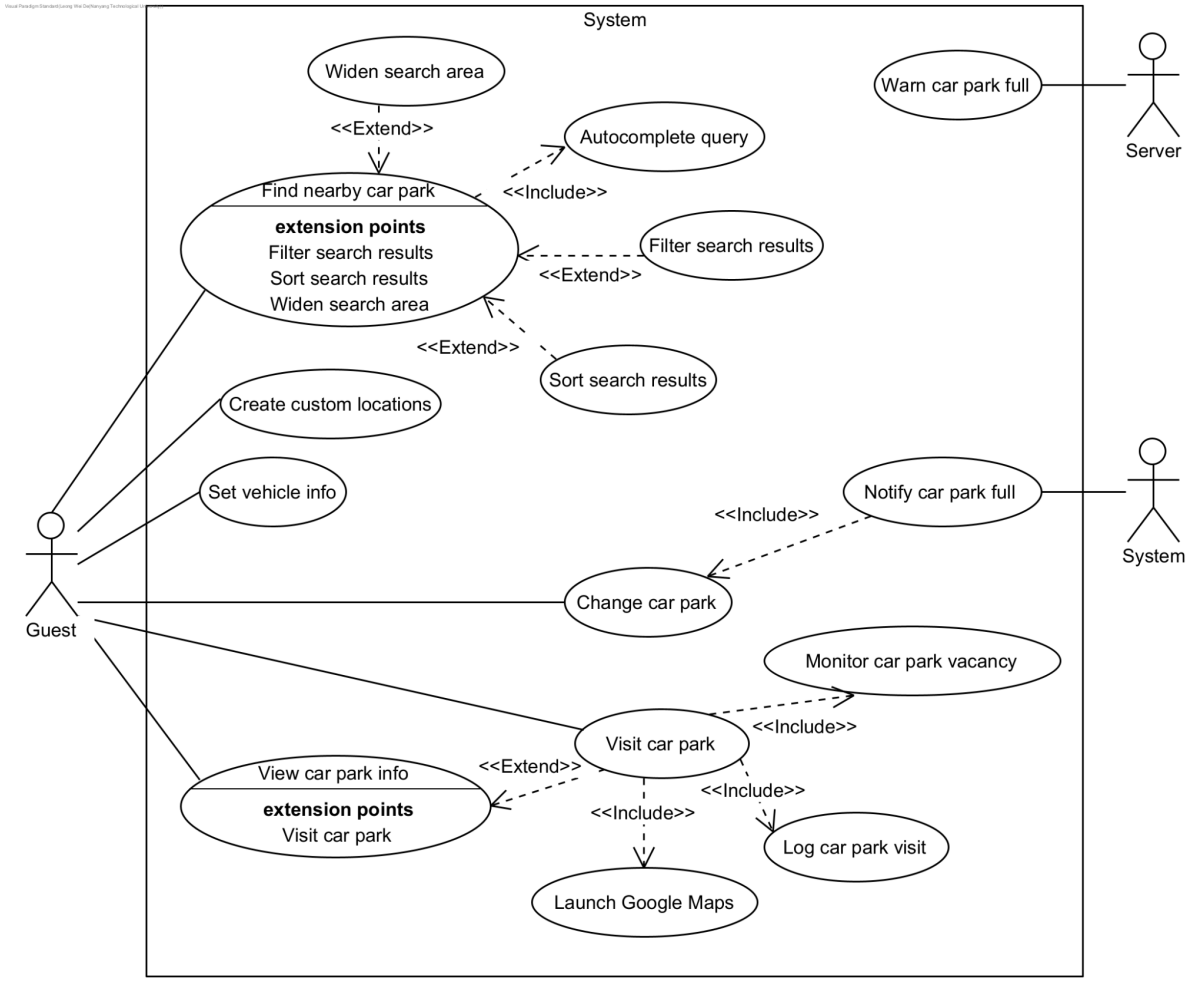
- **N10** The system should automatically scale resources up or down based on demand to optimise performance.
  - **N10.1** Use load testing tools like Apache JMeter, Gatling, or locust.io to simulate a large number of virtual users accessing the car park application concurrently.
  - **N10.2** Gradually increase the load on the system to mimic real-world scenarios where user traffic fluctuates.
- **N11** Regularly review and update capacity planning based on usage trends and growth projections
  - **N11.1** Periodically conduct load testing to validate the capacity planning assumptions and projections.
    - **N11.1.1** Measure the response times for critical user actions and compare them against the defined requirement (100ms in this case).
- **N12** Ensure that the database system can scale to accommodate growing data volumes and User base.
  - **N12.1** Consider configuring auto-scaling for your database infrastructure. Cloud platforms like AWS, Azure, and Google Cloud offer auto-scaling capabilities based on usage metrics.
  - **N12.2** Consider implementing caching mechanisms (e.g., Redis or Memcached) to reduce the load on the database for frequently accessed data.

Data Dictionary

TERM	DEFINITION
availability/ vacancy	Indicates whether the car park is currently operating or closed and number of available slots left per vehicle type.
car park	A designated area or facility where motor vehicles, such as cars and motorcycles can be temporarily parked or stored. Car parks often charge a fee and have limited vacancy for vehicles. Should a user arrive when a car park is full, they have no choice but to find a different location to park. For the purposes of our system, only designated car parks recognised by URA will be in scope.
car park features	Additional features present in the car park, such as electric vehicle charging lots and vehicle washing bays.
destination	The intended location the user wishes to go to.
destination car park	The intended car park the user wishes to go to.
drop pin	The virtual marker that is placed on a map to denote a specific location of interest. By tapping or clicking on the map at the desired Destination, the pin is dropped, serving as a reference point for the app and user and providing relevant information and directions to that specific destination.
fuel type	The fuel that powers the vehicle, such as gasoline or electric hybrid.
guest	An unregistered user of the application. Unless otherwise stated, a guest may access all features of the application.
map	A graphical representation of geographic data or information within the application. The map provides a visual interface for users to view and interact with.
preset search radius	A zone around the intended destination of the user extending up to 500m. The system will consider car parks within this zone as “nearby” unless otherwise specified. The preset search radius may be extended in certain cases.
price	The cost or fee that a person is required to pay for parking their vehicle in a car park. Car park fares vary depending on

	factors such as the location, duration of parking, and the policies of the specific car park or parking facility. Usually measured in dollars per hour.
SLA	Service Level Agreement - a contract between an end-user and company that outlines minimum expected service requirements, including quality, availability, and timeliness.
user	A person that is presently accessing the system.
vehicle type	The class of the vehicle such as car, motorcycle or heavy vehicle.

Use case diagram



Use Case Descriptions

U1

FIND NEARBY CAR PARK

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F1^**, **F2**, **F3**, **F4^**, **N5**, **N6**
description	*As a driver, I want to explore and compare possible places to park my vehicle, before I set off to my destination.* This use case takes in a user-supplied location and returns the nearest car parks that the user can choose to park at.
preconditions	N/A
postconditions	N/A
priority	1
frequency of use	Daily
flow of events	1. **User** types in a location in the search bar 2. **LocationManager** retrieves a list of closest matching locations to the given query as described in **(U5 Autocomplete query)** 3. **User** picks one of the suggestions returned by **LocationManager** 4. **NavigationManager** narrows down list of car parks close to the provided location 5. If there are nearby car parks found in the carpark list, the **MapUI** plots the results on the map

alternative flows	AF2. The location supplied does not exist or is not within mainland Singapore 1. The system informs the user that no results were found AF4. No car parks are within preset search radius of the location 1. The system will proceed into the **(U2 Widen search area)** use case.
exceptions	EX1. The user is not connected to the internet 1. The system informs the user that he/she is not connected to the internet EX2. The app does not have the “Location” permission 1. The system directs the user to grant location permission to continue using the app
includes	1. **(U5 Autocomplete query)**
special requirements	1. The system must return the search results within 1 second. 2. The system must display nearby car parks on the map within 1 second
assumptions	N/A
notes and issues	N/A

U2

WIDEN SEARCH AREA

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F2.1, F2.2**, **F4^, N5, N6**
description	*As a driver, when there are no car parks decently near the destination, I want to know about the next best possible car park(s), before I set off to my destination.* This use case follows up on **(U1 Find nearby car park)** and activates when there is no nearby car park within the preset search radius. As there are no nearby car parks, the system will then return the closest car park to the destination. The system will additionally return up to 2 more car parks provided that they are no more than 1 km away from the first car park found.
preconditions	1. Can only be entered from **(U1 Find nearby car park)** 2. Location is valid
postconditions	N/A
priority	4
frequency of use	Rarely

flow of events	1. No car parks are within preset search radius of the location 2. **NavigationManager** finds the closest car park by distance to the destination. The distance is labelled d. 3. **NavigationManager** expands the search range to $d+1\ km$. 4. **NavigationManager** retrieves two other car parks within the search range, if any 5. **MapUI** plots the results on the map
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	1. The system must return the search results within 1 second. 2. The system must display nearby car parks on the map within 1 second
assumptions	N/A
notes and issues	N/A

U3

FILTER SEARCH RESULTS

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F7^, N5, N6**
description	*As a driver, I would like to hide car parks that are overpriced or have no vacancy.* This use case follows up on **(U1 Find nearby car park)** and allows the user to add filters to hide car parks that do not meet the criteria.
preconditions	1. Can only be entered from **(U1 Find nearby car park)**
postconditions	1. Items that do not match the criteria are also hidden from the map
priority	3
frequency of use	Monthly
flow of events	1. User opens the filters menu. 2. User adds one or more filters. 3. **SearchManager** hides results that do not satisfy the filter criteria 4. **MapUI** plots the results on the map

alternative flows	AF1. User wants to remove a filter 1. User removes a filter 2. System shows all results that matches existing filters AF2. User wants to remove all filters 1. User removes all filters 2. System shows all results AF3. No results after filtering 1. User adds a filter and no results satisfies the existing criteria 2. System displays a helpful message indicating that no results matches the user's criteria
exceptions	N/A
includes	N/A
special requirements	1. The system must return the search results within 1 second. 2. The system must display nearby car parks on the map within 1 second
assumptions	N/A
notes and issues	N/A

U4

SORT SEARCH RESULTS

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F8^, N5**
description	*As a driver, I would like to sort by price, distance or availability when I am in a rush to get to my destination.* This use case follows up on **(U1 Find nearby car park)** and allows the user to sort the results by a predefined sorting criteria.
preconditions	1. Can only be entered from **(U1 Find nearby car park)**
postconditions	N/A
priority	3
frequency of use	Monthly
flow of events	1. **User** opens **SearchResultUI**. 2. **User** selects a sorting criteria in **SearchResultUI**. 3. **SearchManager** arranges the results based on sorting criteria
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	1. The system must return the search results within 1 second.

assumptions	N/A
notes and issues	For full list of sorting criteria, refer to **(F8)**

U5

AUTOCOMPLETE QUERY

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F1.5^, F10**
description	*As a driver, I would like to have previous destinations saved so that I do not need to recall/go through the hassle of reentering the address of my destination.* This use case is an extension of **(U1 Find nearby car park)** and allows the user to pick one location from a list of previously searched locations as the intended destination. Additionally, it also suggests custom locations and frequent locations.
preconditions	N/A
postconditions	1. Successful new searches are saved
priority	5
frequency of use	Daily

flow of events	1. **User** types a location into the search bar 2. **SearchManager** repeatedly calls into **LocationManager** which presents a. all relevant past searches for the user to pick from b. Custom locations c. Closest locations to the query 3. **LocationManager** continually refines suggestions as user continues typing 4. **User** picks his intended destination 5. **UserDB** saves the query if the user did not choose the presented suggestions
alternative flows	N/A
exceptions	EX1. User cancels the search 1. System discards and does not save the query EX2. Invalid location entered 1. System discards and does not save the query
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

U6

SET VEHICLE INFO

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F6.1**
description	*As a driver, I would like the application to remember the type of vehicle I drive so that it would not suggest vehicle lots that I cannot park in.* This use case acts when the guest wants to amend the information of their vehicle.
preconditions	N/A
postconditions	The information of the vehicle the guest drives is stored in the application.
priority	7
frequency of use	Rarely
flow of events	1. **User** opens the SettingsUI 2. **SettingsUI** presents the user with options of vehicle types to choose from 3. **User** chooses their vehicle type 4. The data will be saved in the **UserDB**
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	N/A

assumptions	N/A
notes and issues	See Data Dictionary entry for “Vehicle Type” for full list of vehicle types supported.

U7

VIEW CAR PARK INFO

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F5**
description	*As a driver, I would like to know the information of the car park I am travelling to, so that I can make an informed decision on what to choose.* This use case acts when the guest clicks to view the additional information of any car park they would like to travel to. These destinations could be either in search results or in the custom locations created.
preconditions	N/A
postconditions	N/A
priority	1
frequency of use	Daily
flow of events	1. **User** selects a car park. 2. **CarparkInfoUI** fetches carpark information from **NavigationManager** 3. **CarparkInfoUI** displays the information about the car park such as distance from searched location(if applicable), price of parking and availability of car park.
alternative flows	N/A

exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	It is not possible to show distance from intended location if user enters this use case via other means other than **(U1 Find nearest car park)**

U8

CREATE CUSTOM LOCATIONS

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F6.2**
description	*As a driver, I would like to tag certain locations with a more recognisable name for convenience.* This use case allows the user to save specific locations with a special name, such as “Home” or “Work”.
preconditions	N/A
postconditions	The custom locations created are saved into the system.
priority	7
frequency of use	Rarely
flow of events	1. **User** clicks on create custom location in **SettingsUI** 2. **User** types the name the location should be saved under 3. **User** clicks save 4. **UserDB** saves the **user’s** custom location

alternative flows	AF1. The guest wants to rename a custom location 1. The guest selects the custom location 2. The guest selects to rename the location 3. The guest renames the location 4. The guest presses to save the name, and **UserDB** saves it AF2. The guest wants to delete a custom location 1. The guest selects the custom location to be deleted 2. The guest selects to delete the custom location 3. **UserDB** deletes the saved custom location and displays that the location has been deleted
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

U9

VISIT CAR PARK

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F12**
description	*As a driver, I would like to get driving directions to my intended car park. Additionally, I would like to be notified if my intended car park is no longer vacant so that I may pick an alternative car park ahead of time instead of getting disappointed and delaying my travel time when I inevitably have to find another car park.* This use case acts as a parent use case for the following use cases: 1. **(U12 Launch Google Maps)** 2. **(U10 Monitor car park vacancy)** 3. **(U11 Log car park visit)**
preconditions	1. Car park exists 2. At most 1 instance of this use case can be active per device. In other words, starting another visit will cancel any previous trip.
postconditions	N/A
priority	2
frequency of use	Daily

flow of events	1. User taps on the “Google Maps” button 2. **NavigationManager** launches **(U10 Monitor car park vacancy)** 3. **NavigationManager** launches **(U12 Launch Google Maps)** for the user to get directions 4. **User** drives to the car park following the directions in (3) 5. **User** successfully finds a vacant parking lot 6. **User** marks trip as complete 7. **NavigationManager** launches **(U11 Log car park visit)** 8. Use case terminates successfully
alternative flows	AF1. **User** cancels the visit mid-trip 1. Child use cases are cancelled 2. **NavigationManager** informs the user that the visit is cancelled 3. **User** is redirected to the previous screen
exceptions	EX1. **User** was unable to find a parking lot 1. **User** cancels the visit 2. **User** enters **(U1 Find nearby car park)** EX2. **User** starts another visit while one is currently active 1. Previous visit is automatically cancelled 2. **NavigationManager** proceeds as normal
includes	1. **(U12 Launch Google Maps)** 2. **(U10 Monitor car park vacancy)** 3. **(U11 Log car park visit)**
special requirements	Destination car park may change several times throughout the lifetime of this use case. The system must not require the user to re-input any details again.

assumptions	N/A
notes and issues	The real-time API has a lag time of 3-5 minutes

U10

MONITOR CAR PARK VACANCY

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F12**
description	*As a driver, I would like to know in real time whether the car park I am travelling to is going to still be available* This use case is utilised when the guest has decided on their preferred car park and will travel to it. The system will run to check the availability of the car park in real time and return it to the guest.
preconditions	The user has decided on a car park and is travelling to it
postconditions	1. **NavigationService** maintains the subscription while the user has not reached the destination. 2. **NavigationService** drops the subscription when the user reaches the destination or cancels the trip or changes destination.
priority	6
frequency of use	Daily

flow of events	1. **User** selects their destination car park to travel to. 2. **NavigationManager** informs **server** that it wants to be kept updated about destination car park 3. **User** starts driving to the destination car park 4. **Server** periodically refreshes the number of vacant lots in the car park 5. **User** successfully parks at destination car park 6. Subscription is cancelled
alternative flows	N/A
exceptions	EX1. The carpark the guest is travelling to becomes full 1. **Server** executes warn car park full (**U13 Warn Carpark Full**) and the following use cases
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

U11

LOG CAR PARK VISIT

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F1.5.1, F11**
description	*As a driver, I would like to have previous destinations saved so that I do not need to recall/go through the hassle of reentering the address of my destination.* This use case is a child use case of **(U7 Visit car park)**. It is responsible for storing car parks successfully visited by the user.
preconditions	N/A
postconditions	1. Visit must be logged into database
priority	5
frequency of use	Daily
flow of events	1. **User** reaches intended car park 2. **User** marks trip as complete 3. **NavigationManager** logs visit into **UserDB**
alternative flows	N/A
exceptions	N/A
includes	N/A

special requirements	Destination car park may change several times throughout the lifetime of this use case. The system must ensure that only the final destination is recorded and not the unsuccessful ones.
assumptions	N/A
notes and issues	N/A

U12

LAUNCH GOOGLE MAPS

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F9, F12.1.1.3**
description	*As a driver, I would like to receive directions to get to my destination car park.* This use case is a child use case of **(U9 Visit car park)**. It will redirect the guest to Google Maps with the intended destination car park, to facilitate faster navigation.
preconditions	1. Can only be entered from **(U9 Visit car park)**
postconditions	The guest is redirected to Google Maps with the preferred location in the destination box
priority	3
frequency of use	Daily
flow of events	1. **User** searches and decides on the car park they would like to go to 2. **User** clicks on the button to redirect to Google Maps for the navigation 3. **User** gets redirected to Google Maps with the location in their destination box 4. **User** chooses their start location and begins navigation to the car park.
alternative flows	N/A

exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

U13

WARN CAR PARK FULL

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Server
described by	**F12, N4**
description	*As a server, I would like to inform the system when the car park is full* This use case is called by the server when it detects that the destination car park is no longer vacant.
preconditions	1. The app has an active subscription to monitor the vacancy of the car park server side.
postconditions	1. Message must be sent to the correct device at the end of this use case.
priority	6
frequency of use	Yearly

flow of events	1. **NavigationManager** informs **server** that it wants to be kept updated about destination car park 2. **User** starts driving to the destination car park 3. **Server** periodically refreshes the number of vacant lots in the car park 4. **Server** notifies the **NavigationManager** if car park is full before the user reaches 5. **NavigationManager** launches **(U15 Notify car park full)** 6. Subscription cancelled if the user changes destination car park
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	Destination car park may change several times throughout the lifetime of this use case. The system must ensure that it is subscribed to the latest destination car park. Server must achieve an SLA of 0.001% uptime.
assumptions	N/A
notes and issues	The real-time API has a lag time of 3-5 minutes. Multiple subscriptions are allowed but should be cleared within 2-3 hours to reduce load on the server.

U14

CHANGE CAR PARK

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	System, Guest
described by	**F12.1**
description	*As a driver, when I want to change the car park I am travelling to, I would want the system to change the destination of the car park.* This use case is a child use case of **(U15 Notify Carpark Full)**, and is called to change the destination of the car park.
preconditions	The new car park has been decided
postconditions	The new car park becomes the destination
priority	7
frequency of use	Often
flow of events	1. **(Entry - U15 Notify Car Park Full)** 2. **NavigationManager** dispatches notification to inform **User** that destination car park is full 3. **User** accepts the suggested car park 4. **NavigationManager** switches the destination to the suggested car park.

alternative flows	AF1. **(Entry - U7 View Car Park Info)** 1. **User** previews a different car park from the destination car park 2. **User** clicks “Google Maps” button for the new car park **(U9 Visit Car Park)** 3. **NavigationManager** switches the destination to the suggested car park.
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	The next car park is decided beforehand.
notes and issues	This use case affects the child use cases of **(U9 Visit car park)**

U15

NOTIFY CAR PARK FULL

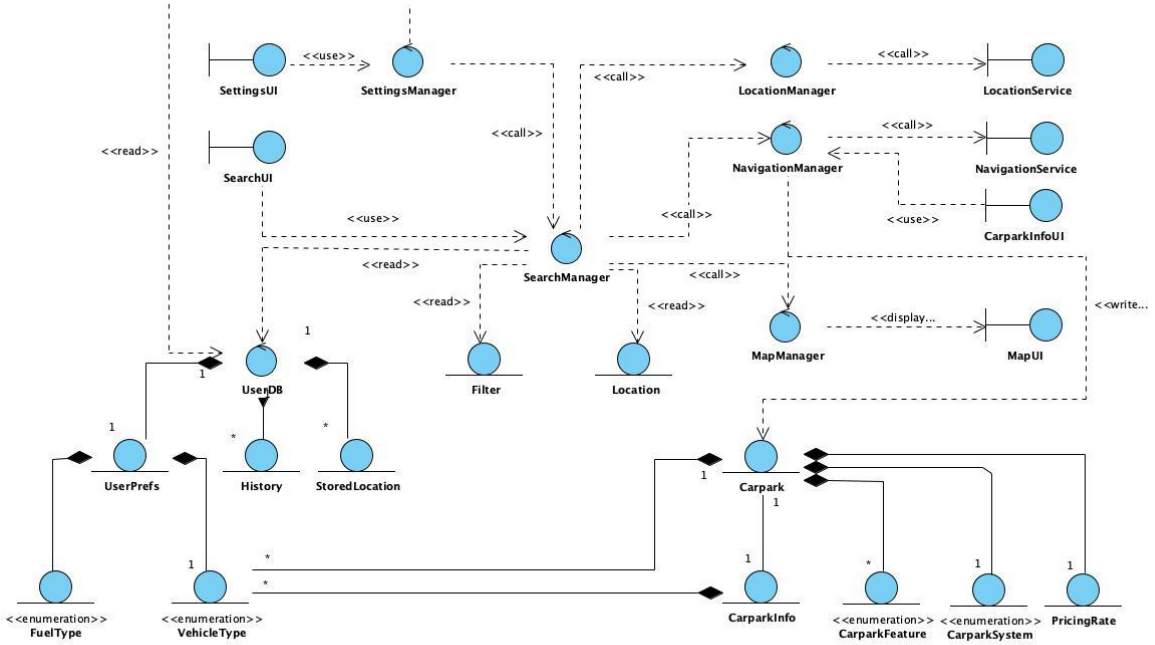
2023-09-03 **date of creation**
G R Devansh Rao **first author**

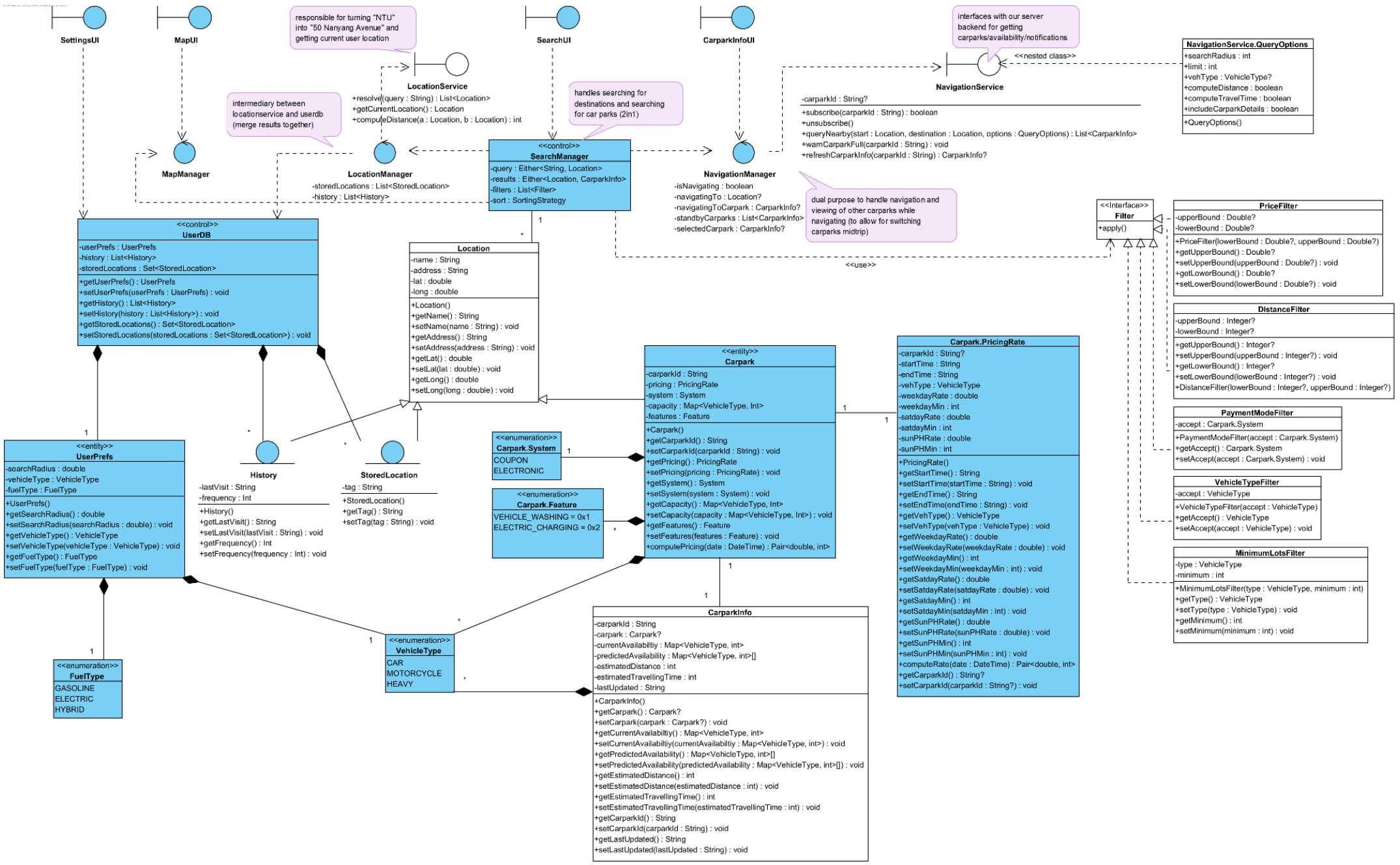
2023-09-22 **date of revision**
Isaac Chun **last author**

actor	System
described by	**F12.1.1.2**
description	*As a driver that has been informed that the car park I am travelling to is not available, I would like to know the next best alternative and make a decision to change my car park destination.* This use case is done by the system to prepare to inform the driver that the car park is full.
preconditions	The server has notified the system that the car park they are travelling to is full.
postconditions	Notification sent to the guest
priority	7
frequency of use	Often
flow of events	1. **NavigationManager** finds the next most relevant car park 2. **NavigationManager** sends a notification to the **user** informing them that the car park they are travelling to is full 3. The notification will also have the next most relevant car park shown and gives the **user** a decision whether they would like to change their car park.

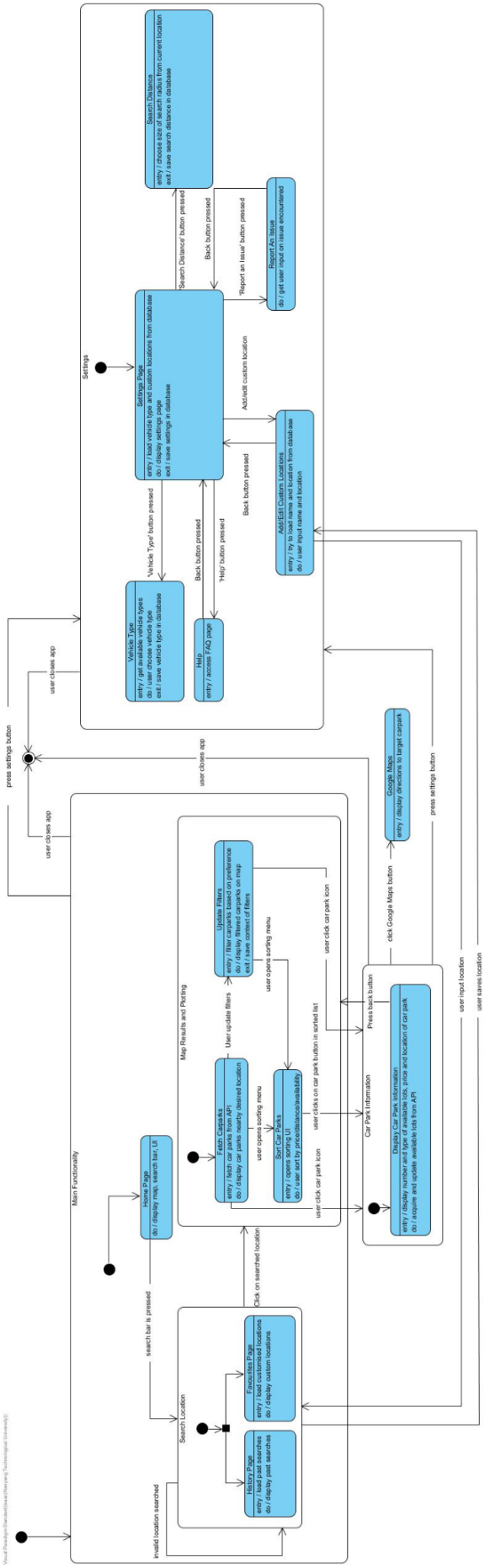
alternative flows	AF1. **User** wishes to swap destination 1. **NavigationManager** carries out **(U14 Change car park)** AF2. **User** wishes to keep driving 1. **NavgiationManager** does not change the destination
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	The system uses the stored car park locations from the initial search to the general location as viable car parks. From these, the system will check their availability and return the most relevant car park.

Class Diagrams




SettingsUI
MapUI
responsible for turning "NTU" into "50 Nanyang Avenue" and getting current user location
LocationService
+resolve(query : String) : List<Location>
+getCurrentLocation() : Location
+computeDistance(a : Location, b : Location) : int
SearchUI
CarparkInfoUI
interfaces with our server backend for getting carparks/availability/notifications
<<nested class>>
NavigationService
-carparkId : String?
+subscribe(carparkId : String) : boolean
+unsubscribe()
+queryNearby(start : Location, destination : Location, options : QueryOptions) : List<CarparkInfo>
+wamCarparkFull(carparkId : String) : void
+refreshCarparkInfo(carparkId : String) : CarparkInfo?
NavigationService.QueryOptions
+searchRadius : int
+limit : int
+vehType : VehicleType?
+computeDistance : boolean
+computeTravelTime : boolean
+includeCarparkDetails : boolean
+QueryOptions()
intermediary between locationservice and userdb (merge results together)
handles searching for destinations and searching for car parks (2in1)
MapManager
LocationManager
-storedLocations : List<StoredLocation>
-history : List<History>
<<control>>
SearchManager
-query : Either<String, Location>
-results : Either<Location, CarparkInfo>
-filters : List<Filter>
-sort : SortingStrategy
NavigationManager
-isNavigating : boolean
-navigatingTo : Location?
-navigatingToCarpark : CarparkInfo?
-standbyCarparks : List<CarparkInfo>
-selectedCarpark : CarparkInfo?
dual purpose to handle navigation and viewing of other carparks while navigating (to allow for switching carparks midtrip)
<<use>>
<<Interface>>
Filter
+apply()
PriceFilter
-upperBound : Double?
-lowerBound : Double?
+PriceFilter(lowerBound : Double?, upperBound : Double?)
+getUpperBound() : Double?
+setUpperBound(upperBound : Double?) : void
+getLowerBound() : Double?
+setLowerBound(lowerBound : Double?) : void
DistanceFilter
-upperBound : Integer?
-lowerBound : Integer?
+getUpperBound() : Integer?
+setUpperBound(upperBound : Integer?) : void
+getLowerBound() : Integer?
+setLowerBound(lowerBound : Integer?) : void
+DistanceFilter(lowerBound : Integer?, upperBound : Integer?)
PaymentModeFilter
-accept : Carpark.System
+PaymentModeFilter(accept : Carpark.System)
+getAccept() : Carpark.System
+setAccept(accept : Carpark.System) : void
VehicleTypeFilter
-accept : VehicleType
+VehicleTypeFilter(accept : VehicleType)
+getAccept() : VehicleType
+setAccept(accept : VehicleType) : void
MinimumLotsFilter
-type : VehicleType
-minimum : int
+MinimumLotsFilter(type : VehicleType, minimum : int)
+getType() : VehicleType
+setType(type : VehicleType) : void
+getMinimum() : int
+setMinimum(minimum : int) : void
<<control>>
UserDB
-userPrefs : UserPrefs
-history : List<History>
-storedLocations : Set<StoredLocation>
+getUserPrefs() : UserPrefs
+setUserPrefs(userPrefs : UserPrefs) : void
+getHistory() : List<History>
+setHistory(history : List<History>) : void
+getStoredLocations() : Set<StoredLocation>
+setStoredLocations(storedLocations : Set<StoredLocation>) : void
Location
-name : String
-address : String
-lat : double
-long : double
+Location()
+getName() : String
+setName(name : String) : void
+getAddress() : String
+setAddress(address : String) : void
+getLat() : double
+setLat(lat : double) : void
+getLong() : double
+setLong(long : double) : void
<<entity>>
Carpark
-carparkId : String
-pricing : PricingRate
-system : System
-capacity : Map<VehicleType, Int>
-features : Feature
+Carpark()
+getCarparkId() : String
+setCarparkId(carparkId : String) : void
+getPricing() : PricingRate
+setPricing(pricing : PricingRate) : void
+getSystem() : System
+setSystem(system : System) : void
+getCapacity() : Map<VehicleType, Int>
+setCapacity(capacity : Map<VehicleType, Int>) : void
+getFeatures() : Feature
+setFeatures(features : Feature) : void
+computePricing(date : DateTime) : Pair<double, int>
Carpark.PricingRate
-carparkId : String?
-startTime : String
-endTime : String
-vehType : VehicleType
-weekdayRate : double
-weekdayMin : int
-satdayRate : double
-satdayMin : int
-sunPHRate : double
-sunPHMin : int
+PricingRate()
+getStartTime() : String
+setStartTime(startTime : String) : void
+getEndTime() : String
+setEndTime(endTime : String) : void
+getVehType() : VehicleType
+setVehType(vehType : VehicleType) : void
+getWeekdayRate() : double
+setWeekdayRate(weekdayRate : double) : void
+getWeekdayMin() : int
+setWeekdayMin(weekdayMin : int) : void
+getSatdayRate() : double
+setSatdayRate(satdayRate : double) : void
+getSatdayMin() : int
+setSatdayMin(satdayMin : int) : void
+getSunPHRate() : double
+setSunPHRate(sunPHRate : double) : void
+getSunPHMin() : int
+setSunPHMin(sunPHMin : int) : void
+computeRate(date : DateTime) : Pair<double, int>
+getCarparkId() : String?
+setCarparkId(carparkId : String?) : void
<<entity>>
UserPrefs
-searchRadius : double
-vehicleType : VehicleType
-fuelType : FuelType
+UserPrefs()
+getSearchRadius() : double
+setSearchRadius(searchRadius : double) : void
+getVehicleType() : VehicleType
+setVehicleType(vehicleType : VehicleType) : void
+getFuelType() : FuelType
+setFuelType(fuelType : FuelType) : void
History
-lastVisit : String
-frequency : Int
+History()
+getLastVisit() : String
+setLastVisit(lastVisit : String) : void
+getFrequency() : Int
+setFrequency(frequency : Int) : void
StoredLocation
-tag : String
+StoredLocation()
+getTag() : String
+setTag(tag : String) : void
<<enumeration>>
Carpark.System
COUPON
ELECTRONIC
<<enumeration>>
Carpark.Feature
VEHICLE_WASHING = 0x1
ELECTRIC_CHARGING = 0x2
<<enumeration>>
FuelType
GASOLINE
ELECTRIC
HYBRID
<<enumeration>>
VehicleType
CAR
MOTORCYCLE
HEAVY
CarparkInfo
-carparkId : String
-carpark : Carpark?
-currentAvailabiltiy : Map<VehicleType, int>
-predictedAvailability : Map<VehicleType, int>[]
-estimatedDistance : int
-estimatedTravellingTime : int
-lastUpdated : String
+CarparkInfo()
+getCarpark() : Carpark?
+setCarpark(carpark : Carpark?) : void
+getCurrentAvailabiltiy() : Map<VehicleType, int>
+setCurrentAvailabiltiy(currentAvailabiltiy : Map<VehicleType, int>) : void
+getPredictedAvailability() : Map<VehicleType, int>[]
+setPredictedAvailability(predictedAvailability : Map<VehicleType, int>[]) : void
+getEstimatedDistance() : int
+setEstimatedDistance(estimatedDistance : int) : void
+getEstimatedTravellingTime() : int
+setEstimatedTravellingTime(estimatedTravellingTime : int) : void
+getCarparkId() : String
+setCarparkId(carparkId : String) : void
+getLastUpdated() : String
+setLastUpdated(lastUpdated : String) : void

Dialog Maps


Settings
Settings Page
entry / load vehicle type and custom locations from database
do / display settings page
exit / save settings in database
Search Distance
entry / choose size of search radius from current location
exit / save search distance in database
Report An Issue
do / get user input on issue encountered
Add/Edit Custom Locations
entry / try to load name and location from database
do / user input name and location
Vehicle Type
entry / get available vehicle types
do / user choose vehicle type
exit / save vehicle type in database
Help
entry / access FAQ page
'Search Distance' button pressed
Back button pressed
'Report an Issue' button pressed
Add/edit custom location
'Vehicle Type' button pressed
Back button pressed
'Help' button pressed
Back button pressed
user closes app
press settings button
user closes app
user closes app
Main Functionality
Home Page
do / display map, search bar, UI
search bar is pressed
Search Location
Favourites Page
entry / load saved locations
do / display custom locations
History Page
entry / load searches
do / display past searches
invalid location searched
Click on searched location
Map Results and Pitching
Fetch Carparks
entry / fetch car parks from API
do / display car parks nearby claimed location
Update Filters
entry / filter carparks based on preference
do / display filtered carparks on map
exit / save current of filters
User update filters
user opens sorting menu
Sort Car Parks
entry / opens sorting UI
do / user sort by price/distance/availability
user opens sorting menu
user clicks on car park button in sorted list
user click car park icon
user click car park icon
Car Park Information
Display Car Park Information
entry / display number and type of available lots, price and location of car park
do / acquire and update available lots from API
Press back button
click Google Maps button
Google Maps
entry / display directions to target carpark
press settings button
user input location
user saves location

Appendix

<Use case ID>

<Use case Name>

2023-09-03 **date of creation**
Robert Huang **first author**

2023-09-03 **date of revision**
Robert Huang **last author**

actor	*An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks. Different actors often correspond to different user classes, or roles, identified from the customer community that will use the product. Name the actor(s) that will be performing this use case.*
described by	*The functional requirements and non-functional requirements handled by this use case*
description	*Provide a brief description of the reason for and outcome of this use case, or a high-level description of the sequence of actions and the outcome of executing the use case.*
preconditions	*List any activities that must take place, or any conditions that must be true, before the use case can be started. Number each precondition.* *Examples:* 1. *User's identity has been authenticated.* 2. *User's computer has sufficient free memory available to launch task.*

postconditions	*Describe the state of the system at the conclusion of the use case execution. Number each postcondition.* *Examples:* 1. *Document contains only valid SGML tags.* 2. *Price of item in database has been updated with new value.*
priority	*Indicate the relative priority of implementing the functionality required to allow this use case to be executed. The priority scheme used must be the same as that used in the software requirements specification.*
frequency of use	*Estimate the number of times this use case will be performed by the actors per some appropriate unit of time.*
flow of events	*Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions. This dialog sequence will ultimately lead to accomplishing the goal stated in the use case name and description. This description may be written as an answer to the hypothetical question, "How do I <accomplish the task stated in the use case name>?" This is best done as a numbered list of actions performed by the actor, alternating with responses provided by the system.*
alternative flows	*Document other, legitimate usage scenarios that can take place within this use case separately in this section. State the alternative course, and describe any differences in the sequence of steps that take place. Prefix alternative flows with AF.*
exceptions	*Describe any anticipated error conditions that could occur during execution of the use case, and define how the system is to respond to those conditions. Also, describe how the system is to respond if the use case execution fails for some unanticipated reason. Prefix exceptions with EX.*

includes	*List any other use cases that are included ("called") by this use case. Common functionality that appears in multiple use cases can be split out into a separate use case that is included by the ones that need that common functionality*.
special requirements	*Identify any additional requirements, such as nonfunctional requirements, for the use case that may need to be addressed during design or implementation. These may include performance requirements or other quality attributes.*
assumptions	*List any assumptions that were made in the analysis that led to accepting this use case into the product description and writing the use case description.*
notes and issues	*List any additional comments about this use case or any remaining open issues or TBDs (To Be Determineds) that must be resolved. Identify who will resolve each issue, the due date, and what the resolution ultimately is.*

This use case template was derived from Karl E. Wiegers' prior work.