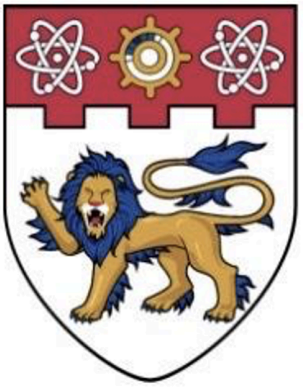
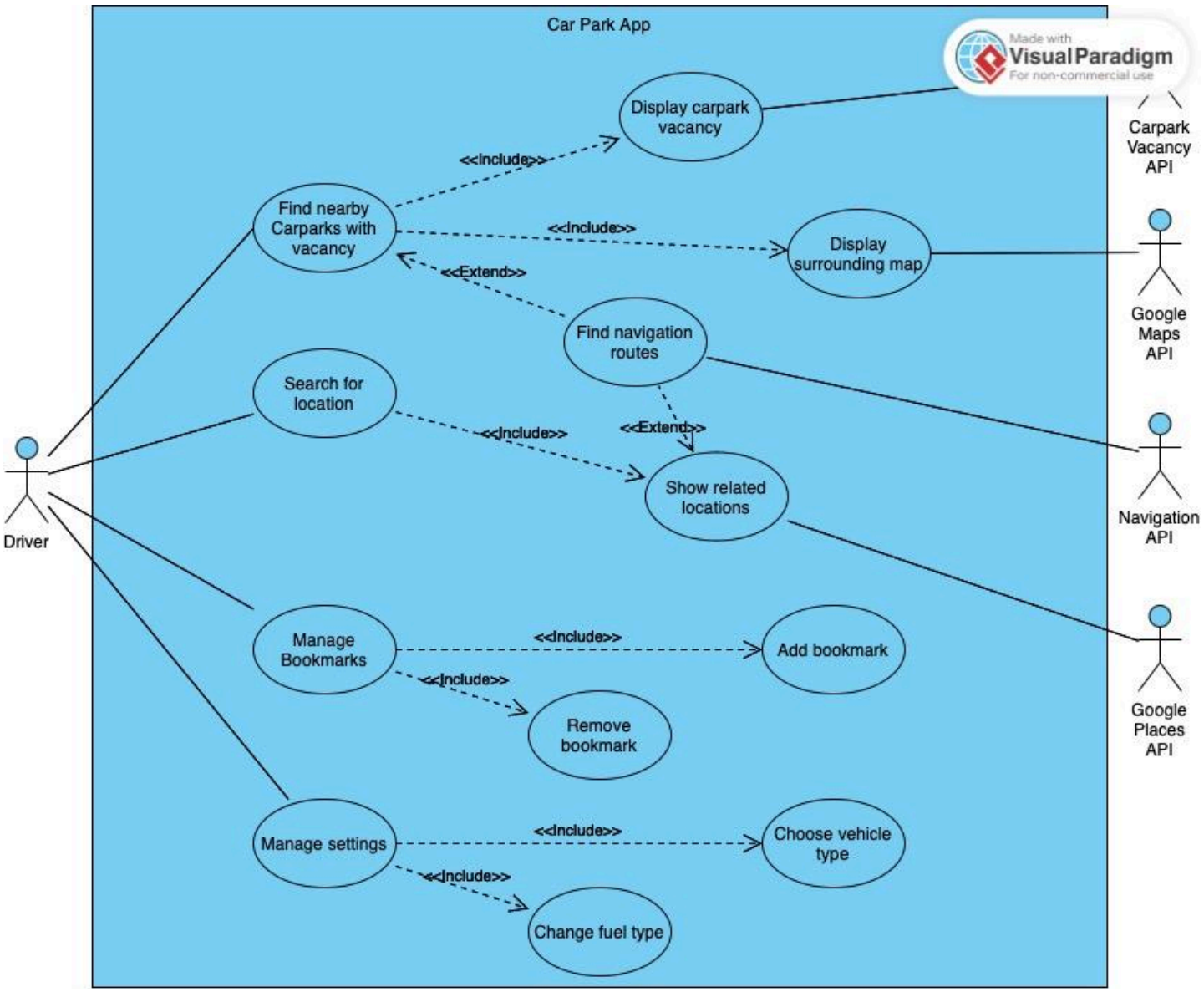


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

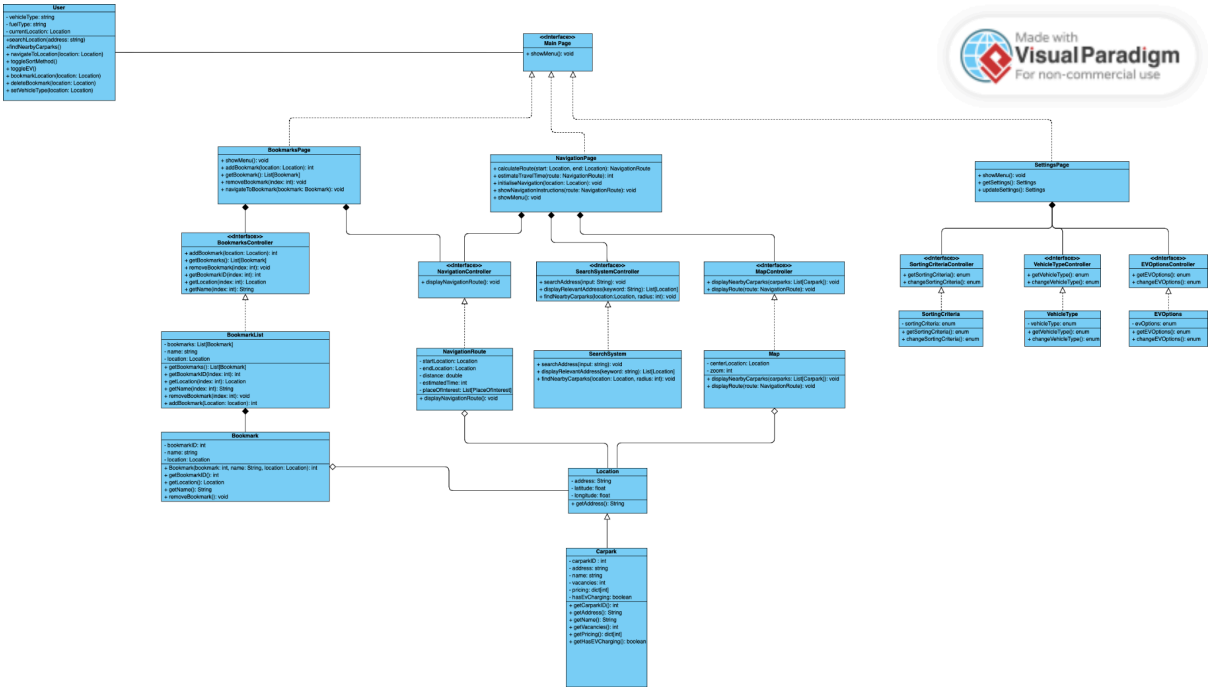
SC2006-24S1 Software Engineering
#Lab 3 Deliverables

Group Member	Matric Number
Quek Jun Siong	U2322145G
Sun Sitong	U2322401J
Jiang Zong Zhe	U2322460F
Tan Yu Xiu	U2322532B
Tan Chong Yao	U2321552F
Solis Aaron Mari Santos	U2322252G

Complete Use Case model



Class Diagram



Description

We integrated multiple design patterns into our system to ensure robustness and maintainability.

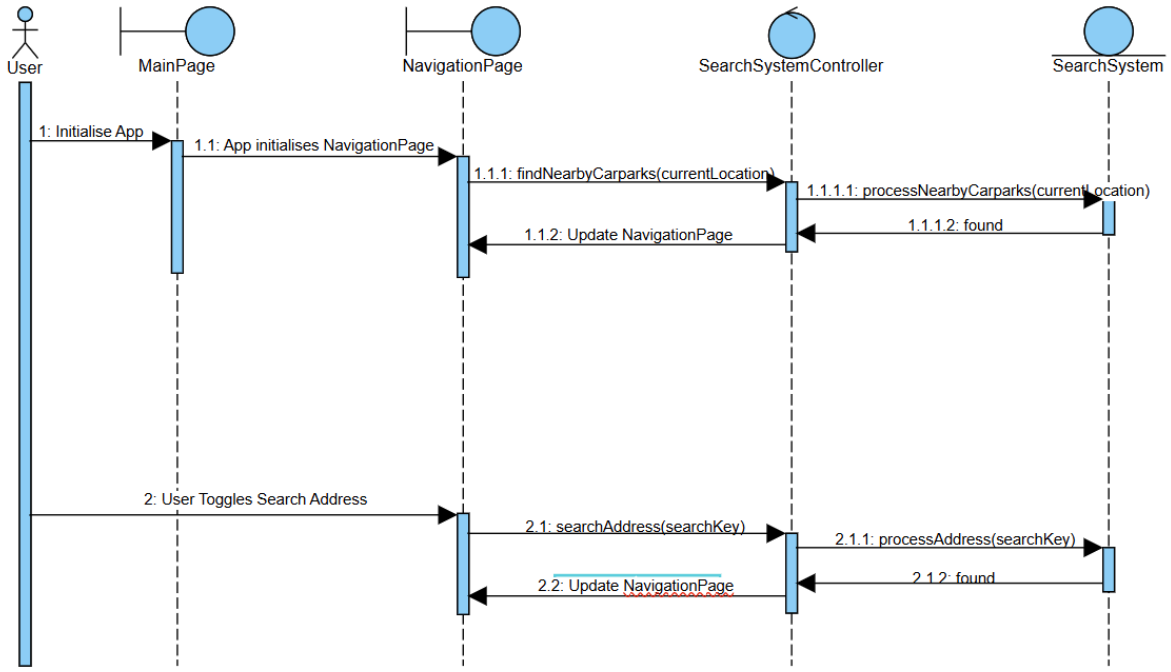
The Strategy Pattern manages menu navigation, allowing seamless switching between interfaces like searching, carpark lists, bookmarks, and settings through a common interface that each menu type implements, enabling runtime flexibility and clean separation of menu-related code.

The Observer Pattern handles real-time updates of carpark information, where the carpark data manager acts as the subject and UI components as observers, ensuring automatic notifications when data changes without constant polling and maintaining loose coupling between data sources and UI components.

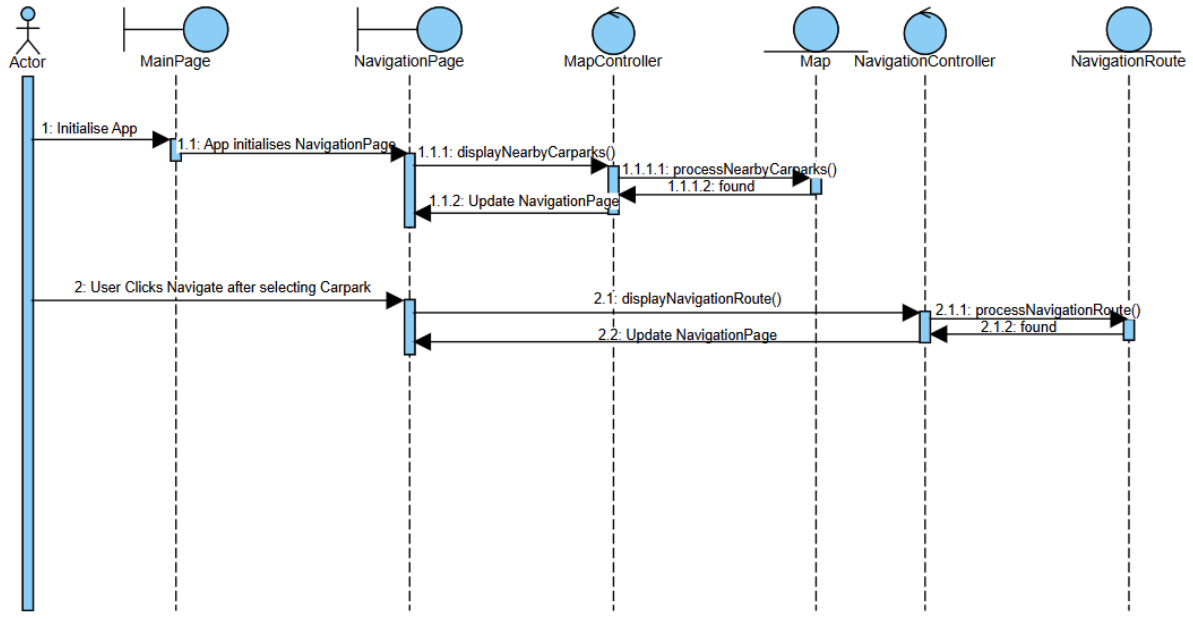
The Facade Pattern simplifies the settings subsystem by providing a unified interface that manages user preferences, hiding the complexity of coordinating multiple settings components and reducing dependencies.

Sequence diagrams

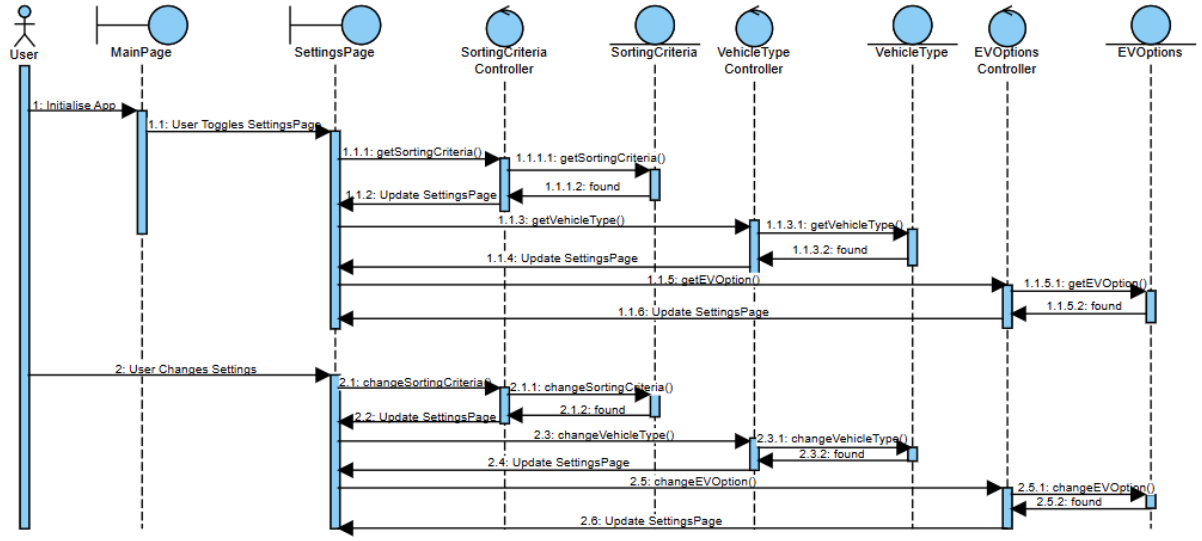
Sequence #1: Searching for Carparks



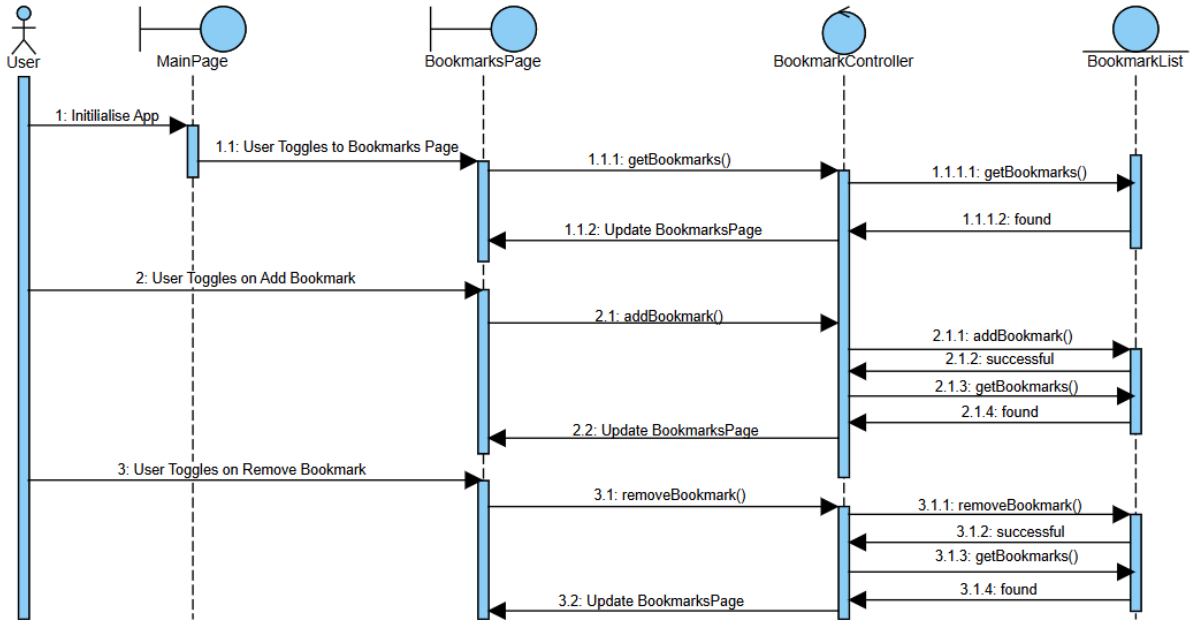
Sequence #2: Showing Nearby Carparks & Navigation Route on Map



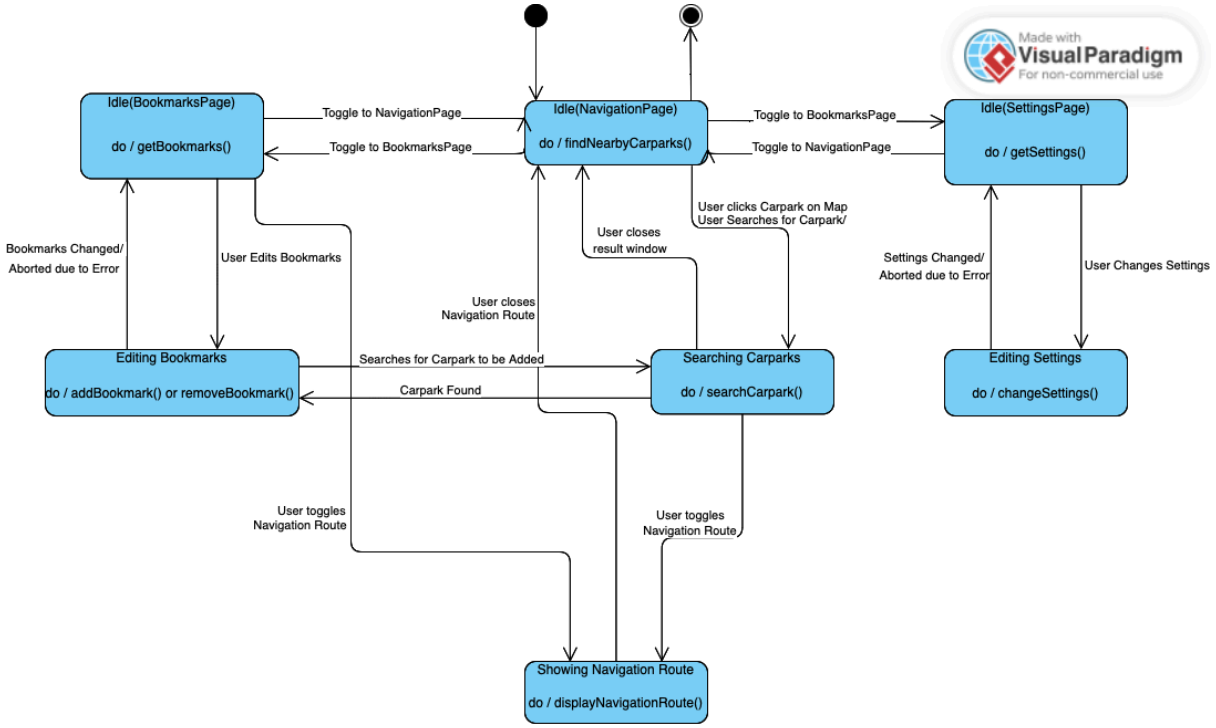
Sequence #3: Accessing and Editing Settings



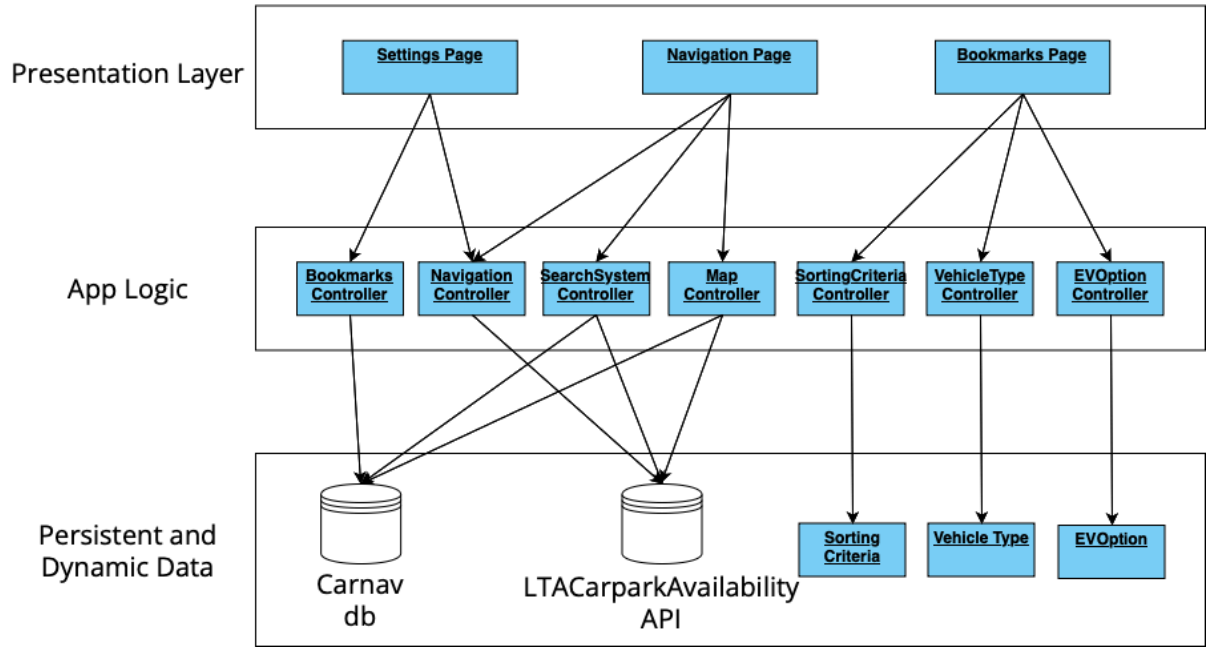
Sequence #4: Adding and Editing Bookmarks



Dialog map


Idle(BookmarksPage)
do / getBookmarks()
Toggle to NavigationPage
Toggle to BookmarksPage
Idle(NavigationPage)
do / findNearbyCarparks()
Toggle to BookmarksPage
Toggle to NavigationPage
Idle(SettingsPage)
do / getSettings()
Bookmarks Changed/
Aborted due to Error
User Edits Bookmarks
User closes
result window
User clicks Carpark on Map
User Searches for Carpark/
Settings Changed/
Aborted due to Error
User Changes Settings
User closes
Navigation Route
Editing Bookmarks
do / addBookmark() or removeBookmark()
Searches for Carpark to be Added
Carpark Found
Searching Carparks
do / searchCarpark()
Editing Settings
do / changeSettings()
User toggles
Navigation Route
User toggles
Navigation Route
Showing Navigation Route
do / displayNavigationRoute()
Made with
VisualParadigm
For non-commercial use

System architecture



Architectural Style

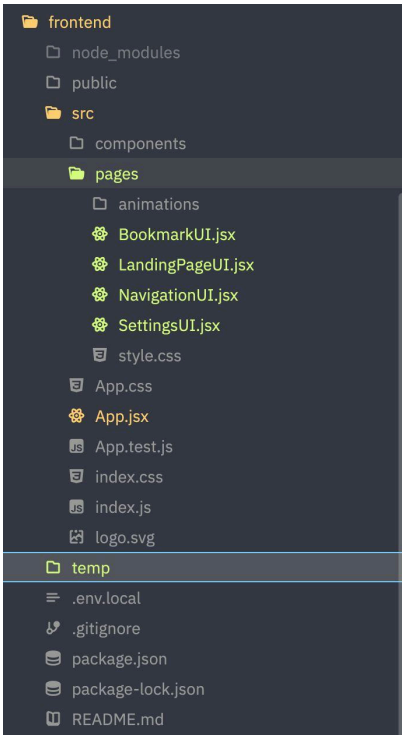
We will mainly be adopting a closed-layered architecture with these 3 layers:

- Presentation layer: Consists of the user interface. The main interface will be the search interface and a menu allowing users to navigate to the bookmarks or setting interface.
- Logic layer: Implements the logic which processes carpark data, location data, bookmarks and settings.
- Database layer: Manages all data storage.

We chose a closed-layered architecture because of its security benefits, as users cannot access the database from the presentation layer. From a maintainability perspective, the closed architecture establishes clear dependencies between layers, ensuring that modifications in one layer don't impact non-adjacent layers, thus simplifying debugging processes. Testing becomes more straightforward as each layer can be tested in isolation, with mock interfaces available for adjacent layers and clear testing boundaries

Application Skeleton

#1: Frontend

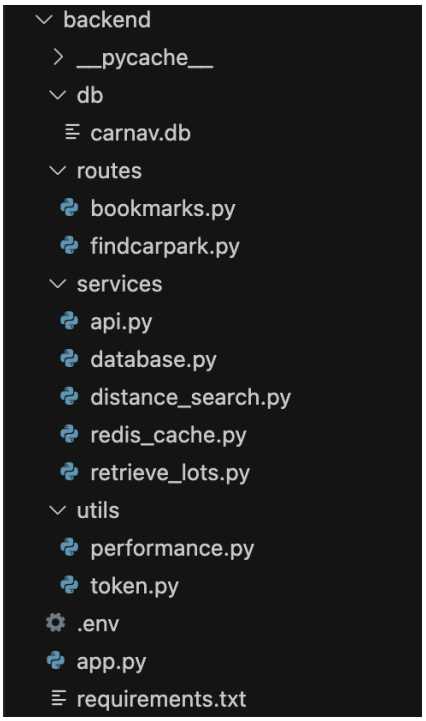


Pages: Contains the different interfaces that the user will be interacting with, including landing page, bookmarks, navigation panel and settings

Components: Contains components like Search Box that will be displayed on the navigation and landing page

App.jsx: Entry point for our application

#2: Backend



Db: Database to store all carpark information, including bookmark, carpark name, coordinates and rates. The db will be renamed to [ProjectName].db eventually after deciding on our application name

Routes: Contains bookmark and find carpark API endpoints for fetching and updating of data between frontend and backend

Services: Contains the methods to access or modify fetched data

Utils: Contains helper methods for external API access and dev-stage performance tracking (not in final production)