System Verification and Validation Plan for Park'd

Team #29, caPstOneGroup
Albert Zhou
Almen Ng
David Yao
Gary Gong
Jonathan Yapeter
Kabishan Suvendran

April 5, 2023

1 Revision History

Date	Version	Notes
Nov. 2, 2022	1.0	Revision 0
Apr 2, 2023	1.1	Revision 1

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

Symbol/Abbreviation/Acronym	Description
T	Test
BE	Business Event
SRS	Software Requirements Specification
NFR	Non-Functional Requirement
FR	Functional Requirement
HTTP Request	A message sent by a client to initiate an action on the server through Hypertext Transfer Protocol

This document outlines caPstOneGroup's plan for verification and validation of the correct and intended specification, implementation, and behaviour of our software system, Park'd. It specifies plans to verify these elements, as well as tests for the requirements in our Software Requirements Specification.

3 General Information

This section provides a brief overview of the software and its general functions, gives the objective of the Verification and Validation plan, and outlines relevant documentation to support the Verification and Validation plan.

3.1 Summary

We are testing the software named Park'd, our parking assistant application. Park'd aims to help drivers find parking spaces by using machine learning algorithms to locate empty spaces from overhead cameras. Our application then directs drivers to those spaces, taking into account restrictions like reserved or accessible spaces. It will maintain a database of spaces.

3.2 Objectives

Our primary objective, through this Verification and Validation plan, is to ensure that our product delivers on the functionality outlined in the functional and non-functional requirements outlined in the SRS without failures. Another objective of enacting this plan is to build confidence in our software correctness and software consistency. By passing the tests outlined in this report and thus, providing the intended outputs, we intend to demonstrate correctness in our software's behaviour. It would also give a measure of assurance in the consistency of our software, as the system should ideally return the same output for a given input. We also wish to use this plan to demonstrate a satisfactory level of usability, as passing the outlined tests will give us the confidence that end users will not encounter unintended bugs or glitches with the system.

3.3 Relevant Documentation

This Verification and Validation Plan document describes tests for functional and non-functional requirements laid out in the Software Requirements Specification document. Details of the components and modules being tested can be found in the Module Guide and Module Interface Specification documents.

4 Plan

This section outlines plans to verify the SRS, the design, the Verification and Validation plan itself, the implementation, and the automated testing. Furthermore, it also outlines the plan to validate the software.

4.1 Verification and Validation Team

Name	Roles	Documentation
Albert	Implementation Verification	Implementation
Almen	Automated Testing and Verification Tools	Automation
David	SRS Verification	SRS
Gary	Software Validation	Validation
Jonathan	Verification and Validation Verification	VnV
Kabishan	Design Verification	Design

Table 1: Team Members

Each individual is in charge of the plan whose role they are assigned. They are not solely responsible for the implementation of the relevant plan. Any team member who has finished their testing can support others, particularly those whose testing is the most time consuming or who has the least % of testing completed.

4.2 SRS Verification Plan

The Software Requirements Specification verification plan ensures that our requirements efficiently cover the breadth of the system, and that they are feasible to implement. Verification will be conducted with group members, as well as classmates. It will be in the form of a questionnaire to be applied to some or all requirements, as well as ad hoc feedback where necessary. Given the large number of requirements in our project, an individual should not necessarily review every requirement.

Our SRS comprises project drivers which include the purpose and the stakeholders, project constraints, functional and non-functional requirements, and potential project issues. Each component should be covered by the questionnaire, and reviewed thoroughly by reviewers.

The questionnaire should cover the following points:

- What stakeholders are missing? Are there any listed that are irrelevant?
- Are there any missed constraints?
- Are there any erroneous facts or assumptions?
- Do requirements specify "what", and not "how"?
- Are requirements unambiguous, verifiable, and abstract?
- What requirements are missing? Are there any listed that are unnecessary?
- Are there any potential issues that are not listed?

The SRS verification plan aims to cover the entire document, and is considered complete when there are no outstanding issues.

4.3 Design Verification Plan

The design verification plan will build confidence that our design infrastructure will take the necessary inputs and deliver the intended outputs with completeness and correctness.

Our system is multi-tiered, in that it is comprised of a back-end and front-end system. The back-end system consists of a machine learning model that will identify the parking spaces and their availability, and path finding algorithms that direct the driver to their desired parking space. The front-end system allows the user to interact with this information. Therefore, the design verification plan will be twofold. The design verification for the back-end system will be performed by our internal team members, as we possess intimate knowledge of the frameworks, and thus, would be able to identify the subtleties as to where our design infrastructure could fail.

For the back-end system, we will be verifying the following:

- Machine learning model accepts the camera and sensor footage
- Machine learning model identifies normal, accessibility, and reserved parking spaces from the camera and sensor footage
- Machine learning model updates the availability of a parking space based on the presence of a vehicle and the driver's geographical information
- Machine learning model outputs the aforementioned information to the front-end system
- Path finding algorithm accepts the driver's geographical information and their desired parking space
- Path finding algorithm calculates instructions to navigate the driver to their desired parking space
- Path finding algorithm delivers these instructions to the front-end system

The front-end system consists of an interface, authored in a framework of our choice that will communicate with the back-end to retrieve parking availability information and driving instructions. It will also send requests to the back-end system to update information, such as when a driver selects a parking space, at which point, the spot becomes unavailable. For the outward facing interface, our design verification approach involves both internal team members and external members, such as fellow classmates. This approach will help us eliminate any biases that we, the developers, may have against testing the system and this allows others to try to break the system.

For the front-end system, we will be verifying the following:

- Interface retrieves normal, reserved, and accessibility parking space information from back-end system
- Interface displays the aforementioned information to the user

- Interface accepts input from the driver to change an available parking space to unavailable
- Upon selecting a parking space, the interface provides the back-end with the driver's geographical information
- Interface will accept the driving instructions from the back-end system
- Interface will display the driving instructions to the driver

4.4 Verification and Validation Plan Verification Plan

This Verification and Validation Plan also needs to be verified and validated. The plan to accomplish this will include reviews and checklists to ensure that every aspect of the Verification and Validation plan accurately and sufficiently describe the development team's plans to verify and validate the system. A review will be completed by the the members of the development team as well as our peers. This review will cover the following points:

- Plan to verify and validate the Software Requirements Specification of the system is described and explained in a clear manner
- Plan to verify and validate the design of the system is described and explained in a clear manner
- Plan to verify and validate the implementation of the system is described and explained in a clear manner
- Tools that will be used for automated testing and verification are described and explained in a clear manner
- Plan to verify and validate the software of the system is described and explained in a clear manner
- Tests for all business events are explained thoroughly are deemed valid tests for its requirement

After reviewing our peers' feedback, if all members of the development team agree that the Verification and Validation Plan satisfies the above criteria, the plan is considered complete.

4.5 Implementation Verification Plan

The implementation verification plan will verify that our implementation has the correct behaviour specified in our SRS. The plan will be split in two: functional and nonfunctional requirements.

For functional requirements, we will split them up according to the back end and front end system. For the back end, testing the machine learning algorithm will be done by dynamic, manual, and automatic testing. At first, we will test manually with a small set of data. Then we will test automatically with a large amount of data to develop the AI further.

For the requirements that use the machine learning algorithm, we will use manual dynamic tests and verify the correct output. For the front end, we will use automatic dynamic tests for requirements such as logging in. The remaining will be done manually.

For non-functional requirements, we will split them up based on system and user requirements. System requirements will be verified with dynamic testing, similar to functional requirements. Instead of checking the output we will verify the system's properties. For user requirements, we will create a usability survey and evaluate based on user feedback.

4.6 Automated Testing and Verification Tools

The automation testing and verification tools is outlined and discussed in the Development Plan.

With regards to code coverage metrics, as mentioned in the Development Plan, the plan is to use Coverage.py and Istanbul to measure the code coverage for our Python and Cypress test cases respectively. The goal is to achieve a minimum of 80% code coverage, putting emphasis on creating high quality test cases that test the most critical components of the system.

4.7 Software Validation Plan

To address the problem of developers including features based on their own understanding and changing requirements without effectively communicating with the client, a software validation plan is needed to determine whether the software satisfies specified requirements. We intend to hold a review session with the stakeholders to verify whether we have built the right product. Different review sessions should be held with different types of stakeholders. For example, functional requirements 1 to 25 are verified in collaboration with a driver who uses the software, whereas the rest is verified with the parking lot owner/manager.

The plan is as follows:

- Each requirement is tested by conducting an acceptance test with the stakeholders.
- The acceptance test scenarios and test cases are identified from the business events from the SRS document. The test cases should cover all the business events.
- Execute test cases and report bugs if any. Test results should be recorded accordingly and re-test bugs once fixed.
- Confirm with the stakeholders whether the business objective has been met. The team must resolve any issue discovered in the previous steps before this.

5 System Test Description

5.1 Tests for Functional Requirements

Every functional requirement in the SRS is assigned a corresponding test in this section. Non-functional requirements are tested in the next section.

5.1.1 BE1

1. BE1-FR1-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, device location is disabled

   Input: N/A

   Output: System prompts the driver to turn on location.

   Test Case Derivation: Location needs to be enabled to allow the application to direct the user to a space.

   How test will be performed: Tester will launch the application while their device's location is disabled.

2. BE1-FR2-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, device location is enabled

   Input: N/A

   Output: System displays to the driver a map of their surroundings.

   Test Case Derivation: The driver needs to see where they are located relative to nearby parking lots so they can navigate to the desired lot.

   How test will be performed: Tester will launch the application with their device's location enabled.

3. BE1-FR3-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, device location is disabled

   Input: N/A

   Output: System displays to the driver a navigation disabled message.

   Test Case Derivation: The driver needs to know that the system won't show their location or nearby parking lots.

   How test will be performed: Tester will launch the application with their device's location disabled.

5.1.2 BE2

1. BE2-FR4-T1

   Control: Structural, Dynamic, Automatic

   Initial State: The backend machine vision system is running

   Input: Video feed

Output: The backend system recognizes the video feed and processes the image.

Test Case Derivation: A video feed will need to be fed to the system for it to do its work. The function of recognizing parking spaces is meant to be fulfilled using this feed.

How test will be performed: The backend system will be run with a given video.

2. BE2-FR5-T1

   Control: Structural, Dynamic, Automatic

   Initial State: The backend system is running with a video feed displaying a disabled space.

   Input: Test space that is known to be marked as disabled

   Output: The backend system recognizes the space and marks it in the database as a disabled space.

   Test Case Derivation: Disabled spaces are to be marked accordingly so they are not suggested to drivers who cannot use them.

   How test will be performed: The system will run with a known disabled space to ascertain correct functioning of its space detection.

3. BE2-FR6-T1

   Control: Structural, Dynamic, Automatic

   Initial State: The backend system is running with a video feed displaying a valid space

   Input: Test space with known location

   Output: The test space and the correct location input to the database.

   Test Case Derivation: The location given by the system should be the same as that determined by us beforehand with an accuracy of at least $DISTANCE\_TOLERANCE$ meters..

   How test will be performed: The system will run with a space with known location to ascertain correct functioning of its space detection.

4. BE2-FR7-T1

   Control: Functional, Dynamic, Manual

   Initial State: System running with driver location enabled

   Input: Driver selects the option to navigate to a parking lot

   Output: System displays a prompt for the driver to select their desired parking area.

   Test Case Derivation: The system should make it obvious that the driver should select a destination.

   How test will be performed: Tester will select the option to navigate to a parking lot with location enabled.

5. BE2-FR8-T1

   Control: Functional, Dynamic, Manual

   Initial State: System running with driver location enabled, parking lot selected

   Input: N/A

   Output: System displays the number of spaces available at the selected destination.

   Test Case Derivation: The system should inform the driver of conditions at their selected destination before they arrive.

   How test will be performed: Tester will select the option to navigate to a parking lot with location enabled and observe the output.

5.1.3 BE3

1. BE3-FR9-T1

   Control: Functional, Dynamic, Manual

   Initial State: System running with driver location enabled, parking lot selected

   Input: Driver selects the option to filter the visible spaces

   Output: Depending on the input, only normal, accessible, and reserved spaces are displayed.

   Test Case Derivation: Some drivers need to be able to find specially designated spaces.

   How test will be performed: Tester will select the option to filter the visible spaces at a selected parking lot.

2. BE3-FR10-T1

   Control: Functional, Dynamic, Manual

   Initial State: System running with driver location enabled, parking lot selected

   Input: N/A

   Output: Reserved and disabled spaces are not displayed.

   Test Case Derivation: These spaces are not to be displayed to drivers unless they are qualified to use them.

   How test will be performed: Tester will observe the output of the system with a selected parking lot and no further inputs.

5.1.4 BE4

1. BE4-FR11-T1

   Control: Functional, Dynamic, Manual

   Initial State: System running with driver location enabled, parking lot selected

   Input: Driver selects a parking space at the selected parking lot

Output: System displays the selected parking space.

Test Case Derivation: The system should give feedback that the driver has selected a space, and that it will work with that space.

How test will be performed: Tester will select a parking space.

2. BE4-FR12-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spaces available, but no parking space selected

   Input: Selecting the parking space

   Output: Driver is shown directions from their location to the desired parking space.

   Test Case Derivation: The directions should begin at the driver's location and end at the parking space's location, otherwise, the driver will be provided wrong directions.

   How test will be performed: Tester will pick a parking space and validate that the directions take them from their current position to the parking space.

3. BE4-FR13-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spaces available, but no parking space selected

   Input: Driver selects default recommendation for parking space

   Output: Driver is given directions to reach the parking space provided as a recommendation.

   Test Case Derivation: To limit the driver's interaction with the interface whilst driving, the recommendation should be available for parking and should meet the needs of the user (e.g. default accessibility parking for those with disabilities)

   How test will be performed: Tester will toggle between normal and accessibility parking modes and check that the system provides a default parking space that is available.

4. BE4-FR14-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spaces available, but no parking space selected

   Input: N/A

   Output: The system will display navigation directions to the default recommended parking space after $DEFAULT\_DELAY$ seconds.

   Test Case Derivation: After $DEFAULT\_DELAY$ seconds, the driver must receive instructions, since they should minimize their interaction with the interface while driving.

   How test will be performed: Tester will toggle between normal and accessibility parking modes and wait for $DEFAULT\_DELAY$ seconds to check that the system provides directions to a default parking space that is available.

5.1.5 BE5

1. BE5-FR15-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started

   Input: Driver selects a parking space that is unavailable.

   Output: System informs the driver that they cannot park in a spot that is already occupied by another vehicle.

   Test Case Derivation: Driver should not be allowed to park in a spot that houses another vehicle.

   How test will be performed: Tester will attempt to select an unavailable parking space and validate that they receive a notification from the system.

2. BE5-FR16-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, reserved or accessibility parking spaces available, driver has not filtered out reserved or accessibility spaces

   Input: Driver selects a reserved or accessibility parking space.

   Output: The system will inform the driver that they have tried to select a parking space that does not meet their needs.

   Test Case Derivation: Drivers should not be allowed to park in spaces that they are not entitled to.

   How test will be performed: Tester will not filter reserved and accessibility parking spaces. Upon selecting one of the aforementioned spaces, the tester will be notified by the system.

5.1.6 BE6

1. BE6-FR17-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking space selected

   Input: Driver's geographical location

   Output: As the driver commutes, the directions should update such that the parking space remains stationary and the user's location is dynamic

   Test Case Derivation: If there are obstacles on the road, the driver's change in location should lead the system to recalculate the path to the parking space

   How test will be performed: Tester will follow the instructions and ask another tester to stand in front of their vehicle. The directions should change at this moment.

2. BE6-FR18-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking space selected, other parking spaces available

   Input: N/A

   Output: Once the initial parking space is no longer available, the system shows a recommendation to the driver.

   Test Case Derivation: Driver should not be led to a parking space that is taken, as this will result in frustration and traffic congestion.

   How test will be performed: Tester will drive to the initial parking space. Another tester will park at the initial parking space. The first tester will receive a recommendation to another parking space.

3. BE6-FR19-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking space selected, other parking spaces available

   Input: N/A

   Output: Once the initial parking space is no longer available, the system lets the user pick another space from those that are available.

   Test Case Derivation: User should be allowed to pick another parking space if their initial parking space is taken.

   How test will be performed: Tester will drive to the initial parking space. Another tester will park at the initial parking space. The first tester will be given the ability to pick another parking space.

4. BE6-FR20-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking space selected, parking space taken

   Input: N/A

   Output: System terminates the route and informs the driver to select and new spot.

   Test Case Derivation: The driver needs to know if the spot they're going to is taken by someone else.

   How test will be performed: Tester will drive to the initial parking space. Another tester will park at the initial parking space. The first tester will be informed of the change.

5. BE6-FR21-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking space selected

   Input: N/A

   Output: System terminates the route after a time estimate plus $TIMEOUT\_TOLERANCE$ seconds and informs the driver to select and new spot.

   Test Case Derivation: The driver needs to know if they've taken too long to drive to the spot

   How test will be performed: Tester will take a long drive to the parking space. The tester will be informed of the timeout

6. BE6-FR22-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started

   Input: N/A

   Output: System displays current location within $DISTANCE\_TOLERANCE$ meters

   Test Case Derivation: The driver should have an accurate estimation of their location on the map.

   How test will be performed: Tester will start the system and note the location on the map compared to their known location.

7. BE6-FR23-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spot selected

   Input: N/A

   Output: System displays a time estimate to the destination

   Test Case Derivation: The driver should have an accurate estimation of the driving time within $DRIVING\_ESTIMATE\_TOLERANCE$ seconds.

   How test will be performed: Tester will drive to a selected parking spot and compare the total driving time with the initial estimate.

5.1.7 BE7

1. BE7-FR24-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spot selected

   Input: Driver's geographical location

   Output: The system shall inform the driver when they have reached their parking spot.

   Test Case Derivation: If the driver is not informed, they might not be aware that they have reached the parking spot of their selection.

   How test will be performed: Tester will drive to their selected parking space and validate that they receive a notification.

2. BE7-FR25-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started

   Input: Driver's geographical location

   Output: The parking space changes from available to unavailable.

   Test Case Derivation: Other drivers are aware that the parking space is unavailable, such that they will not pick the location when parking in the future.

   How test will be performed: Tester confirms that while driving, the selected parking space is available. When they park, the selected parking space should become unavailable.

3. BE7-FR26-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spot reached

   Input: Driver's feedback

   Output: The feedback is saved.

   Test Case Derivation: Storing the driver's feedback will allow the team to detect issues and dissatisfaction with the system.

   How test will be performed: Tester will reach a parking spot, verify that they receive a feedback prompt, and confirm the feedback is saved.

5.1.8 BE8

1. ~~BE8-FR27-T1~~

   ~~Control: Functional, Dynamic, Manual~~

   ~~Initial State: System started~~

   ~~Input: New settings~~

~~Output: The system settings change to the ones given~~

~~Test Case Derivation: Not changing the settings would make being able to enter new ones useless.~~

~~How test will be performed: Tester will check the current settings, apply new ones, and verify the change to the system.~~

2. ~~BE8-FR28-T1~~

   ~~Control: Functional, Dynamic, Manual~~

   ~~Initial State: System started~~

   ~~Input: N/A~~

   ~~Output: System settings including volume and sound adjustment, unit identification, vehicle details, and notification preferences~~

   ~~Test Case Derivation: These settings are specified under the functional requirement.~~

   ~~How test will be performed: Tester will open the settings and verify these settings are present.~~

5.1.9 BE9

1. BE9-FR29-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spot selected, parking spot path found

   Input: Cancel signal

   Output: Parking spot selection and path are discarded

   Test Case Derivation: The driver is no longer going to this spot so they should no longer be given the path.

   How test will be performed: Tester will choose a parking slot to go to, cancel the path, and verify that the selection and path are gone.

2. BE9-FR30-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking spot selected, parking spot path found

   Input: Cancel signal

   Output: The driver's location

   Test Case Derivation: Once the path is terminated, the driver will may to choose another parking spot.

   How test will be performed: Tester will choose a parking slot to go to, cancel the path, and verify that they return to spot selection.

5.1.10 BE10

1. BE10-FR31-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started

   Input: N/A

   Output: Admin console login

   Test Case Derivation: The admin console is a protected and is intended only for parking lot owners/managers.

   How test will be performed: Tester will verify the admin console requires admin credentials.

2. BE10-FR32-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started

   Input: Admin credentials

   Output: Admin console

   Test Case Derivation: The admin console is a protected console for parking lot owners/managers.

   How test will be performed: Tester will verify the admin console is accessible only with the admin credentials.

3. BE10-FR33-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, accessing admin console

   Input: N/A

   Output: Occupancy map

   Test Case Derivation: The occupancy map will show the availability of all parking spaces in the parking lot as specified under the functional requirement.

   How test will be performed: Tester will access the admin console and verify the occupancy map is accurate to the parking lot.

5.1.11 BE11

1. BE11-FR34-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, accessing admin console

   Input: N/A

   Output: Parking analytics

Test Case Derivation: The parking analytics will show live data of the availability of all parking spaces in the parking lot as specified under the functional requirement.

How test will be performed: Tester will access the admin console and verify the live parking analytics are given.

2. BE11-FR35-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, parking lot selected

   Input: N/A

   Output: Graph with occupancy levels

   Test Case Derivation: Twenty-four hours worth of historical information is required to allow the user/admin to make an informed decision about choosing a particular spot, or conducting maintenance work.

   How test will be performed: Tester will see if live data for a particular hour is within 20% of the historical data for that hour.

5.1.12 BE12

1. BE12-FR36-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, accessing admin console

   Input: Parking lot layout change

   Output: The parking lot layout changes to the one given

   Test Case Derivation: Parking lot owners/managers will want to modify the layout in case it changes.

   How test will be performed: Tester will access the admin console, make a change to the parking lot layout, and verify the new layout is kept.

2. BE12-FR37-T1

   Control: Functional, Dynamic, Manual

   Initial State: System started, accessing admin console

   Input: Parking lot spot change

   Output: The parking lot spot changes to the one given

   Test Case Derivation: Parking lot owners/managers will want to redefine a parking spot in case of adding/removing/moving reserved spots.

   How test will be performed: Tester will access the admin console, redefine a parking spot, and verify the spot has changed.

5.2 Tests for Nonfunctional Requirements

Non functional requirements can be found in the SRS. They are organized by type and subtype.

5.2.1 Look and Feel Requirements

Appearance Requirements

1. NFR-LF1-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Go through all pages and uses all interface functions and features

   Output/Result: N/A

   How test will be performed: Each tester will go through the whole app and try to use all available functions and features. If an interface feature, such as a button, is unanimously deemed unnecessary, it will be removed.

Style Requirements

1. NFR-LF2-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Go through all pages and uses all interface functions and features

   Output/Result: N/A

   How test will be performed: Navigation app users will be surveyed. Users will be given $EXPLORATION\_TIME$ to explore the app, after which they will answer whether or not the app is visually/stylistically similar to existing popular navigation apps such as Google Maps, Apple Maps, and Waze. At least 5 users will be surveyed. At least 80% of users surveyed should indicate that there is a similarity.

5.2.2 Usability and Humanity Requirements

Ease of Use Requirements

1. NFR-UH1-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Go through all pages and uses all interface functions and features

   Output/Result: N/A

   How test will be performed: Testers will test all functions considered atomic and make sure that none take more than MAX_TAPS, defined in the SRS document, to complete.

2. NFR-UH2-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Go through all pages and uses all interface functions and features

   Output/Result: N/A

   How test will be performed: Navigation app users will be surveyed. Users will be given *EXPLORATION_TIME* to explore the app, after which they will answer whether or not the app is as easy to use as to existing popular navigation apps such as Google Maps, Apple Maps, and Waze. At least 5 users will be surveyed. At least 80% of users surveyed should indicate that it is as easy to use.

Learning Requirements

1. NFR-UH3-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Go through steps to find an available parking spot

   Output/Result: N/A

   How test will be performed: Random variety of users will be surveyed. Users will be given *EXPLORATION_TIME* to figure out how to find and navigate to an available parking spot without any prior instructions. Testers will record how many users successfully complete the task. At least 5 users will be surveyed. At least 80% of users should successfully complete the task.

Understandability and Politeness Requirements

1. NFR-UH4-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Valid address of a parking lot

   Output/Result: Symbols identifying special parking spaces (i.e. accessibility, electric vehicle, or reserved) are displayed.

   How test will be performed: Testers will check that all graphic files of symbols used in the application conform to any regulations set out by the Ontario Ministry of Transportation.

5.2.3 Performance Requirements

Speed and Latency Requirements

1. NFR-PE1-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched

   Input/Condition: Use all the functions and features of the system

   Output/Result: The system must maintain a minimum of $MIN\_FRAMERATE$ at all times

   How test will be performed: This test would be performed on multiple devices (i.e. PC, mobile device). The tester would open up the system on a device and monitor the framerate while performing operations on the application.

Safety-Critical Requirements

1. NFR-PE2-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched in a vehicle

   Input/Condition: Address of a parking lot is entered and an empty parking spot located near a sidewalk or at the corner of a parking lot is selected.

   Output/Result: The system successfully provides directions without going on a sidewalk or off the road.

   How test will be performed: The tester will launch the system on their device and select an available parking spot located beside a sidewalk. The tester will then follow the directions provided by the system until the tester reaches the designated parking spot without going off the road or on a sidewalk.

2. NFR-PE3-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is launched in a vehicle

   Input/Condition: Address of a parking lot is entered and an empty parking spot located near a restricted area like a fire lane is selected.

   Output/Result: System directs the user to park only in a designated space.

   How test will be performed: Select different available parking spot as a destination for the navigation especially those edge cases spots near fire lanes and drive ways. Make sure the suggested spot actually exists, and is not in a space that only resembles a parking space.

Robustness or Fault-Tolerance Requirements

1. NFR-PE4-T1

   Type: Structural, Dynamic, Manual

   Initial State: The back-end service is disabled.

   Input/Condition: System is launched.

   Output/Result: The most recent parking spot information before back-end service is disabled should be displayed.

   How test will be performed: The developers will disable the API in the back-end service that is responsible for sending spot information back to the system. Then the system is launched which sends a request to acquire parking spot information. Since the remote cache should store the information of the most recent parking spot information, the system should still receive the most recent information from the cache.

Capacity Requirements

1. NFR-PE5-T1

   Type: Structural, Dynamic, Manual

   Initial State: The system is operating normally.

   Input/Condition: Sending $CONCURRENT\_USERS$ concurrent requests.

   Output/Result: The system must be able to handle the $CONCURRENT\_USERS$ different request with a valid response

   How test will be performed: The developers will create $CONCURRENT\_USERS$ different API requests with different parameters that requests different parking lot information. Then relevant API testing frameworks would be used to send $CONCURRENT\_USERS$ different web requests at the same time. The developers should observe the system critical metrics such as CPU utilization rate and RAM usage and relevant API response time on the dashboard.

Scalability or Extensibility Requirements

1. NFR-PE6-T1

   Type: Functional, Dynamic, Manual

   Initial State: System has at least $PARKING\_LOTS$ parking lot layout in $TIMEFRAME$ and is launched in a vehicle.

   Input/Condition: Valid addresses of $PARKING\_LOTS$ parking lots and an empty parking space for each.

   Output/Result: Correct directions and information on the parking lot is displayed.

   How test will be performed: The tester should enter a valid address of a parking lot, select an empty parking space, and follow the directions provided by the application. Upon successful completion, the tester should enter a different valid address of a

parking lot, select an empty parking space, and follow the directions provided by the application. The success of the test is determined if the tester is able to successfully navigate to each empty parking space located in $PARKING\_LOTS$ different parking lots.

5.2.4 Operational and Environmental Requirements

Expected Physical Environment

1. NFR-OE1-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is opened on a mobile device in a vehicle

   Input/Condition: Input an address, select a parking space, and follow the directions given by the application to the parking space.

   Output/Result: The tester is able to complete all of the tasks in a vehicle and is able to park successfully.

   How test will be performed: A tester will be in a vehicle, opens the system on their device and inputs a parking lot address and a space to navigate to. The tester will then start the vehicle and navigate to the parking space with the directions given by the application. The test will complete when the tester is parked in the parking space.

Release Requirements

1. NFR-OE2-T1

   Type: Structural, Dynamic, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   How test will be performed: Every month, the developers will check the GitHub commit log for a new update.

5.2.5 Maintainability and Support Requirements

Maintenance Requirements

1. NFR-MA1-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is running in its normal state

   Input/Condition: Either the software or hardware component of the system will go under maintenance.

   Output/Result: Component that is not under maintenance does not throw an unexpected error.

How test will be performed: The developers will take the software component down for maintenance and check that the hardware component does not throw an unexpected error or fail, and vice versa with taking down the hardware component down and checking the software component for unexpected errors.

Supportability Requirements

1. NFR-MA2-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is running on the user's device

   Input/Condition: N/A

   Output/Result: N/A

   How test will be performed: Random users will be surveyed. Users will be give 5 minutes to look through instructions provided in the interface and answer the survey if they are satisfied with the instructions and contacts provided. At least *SUPPORTABILITY_SATISFACTION* of surveyed users should be satisfied with the instructions and contacts provided. At least 5 users will be surveyed. At least 80% of users surveyed should be satisfied with the instructions given.

5.2.6 Security Requirements

Access Requirements

1. NFR-SR1-T1

   Type: Functional, Dynamic, Automatic

   Initial State: Administrative website is launched and a login screen is presented.

   Input/Condition: Valid username and password for the parking lot owners/team.

   Output/Result: Redirection to the administrative console with proper parking lot data of their parking lot.

   How test will be performed: A Cypress automation script will be created that will input a valid username and password and check to ensure that the login is successful and that the proper parking lot data of their parking lot is displayed.

2. NFR-SR1-T2

   Type: Functional, Dynamic, Automatic

   Initial State: Administrative website is launched and a login screen is presented.

   Input/Condition: Invalid username and password for parking lot owners/team.

   Output/Result: Prompt indicating that the login was unsuccessful.

   How test will be performed: A Cypress automation script will be created that will input an invalid username and password and check to ensure that the login was unsuccessful, no redirection occurs, and that a prompt is shown that the login was unsuccessful.

3. NFR-SR1-T3

   Type: Functional, Dynamic, Automatic

   Initial State: Driver application is launched.

   Input/Condition: Address of a parking lot.

   Output/Result: Unable to see analytics and parking lot data.

   How test will be performed: A Cypress automation script will be created that will input a valid address of a parking lot and ensure that analytics and parking lot data are not displayed.

4. NFR-SR2-T1

   Type: Functional, Dynamic, Automatic

   Initial State: Logged in to the administrative console

   Input/Condition: Edit function is enabled and is selected for the designated parking lot.

   Output/Result: Edit view is displayed allowing for editing of the designated parking lot.

   How test will be performed: A Cypress automation script will be created that will select the edit option and ensure that all the editing options are available for the designated parking lot.

5. NFR-SR2-T2

   Type: Functional, Dynamic, Automatic

   Initial State: Driver application is launched.

   Input/Condition: Address of a parking lot.

   Output/Result: Edit function is not shown.

   How test will be performed: A Cypress automation script will be created that will input a valid address of a parking lot and ensure that no editing functionality is enabled.

6. NFR-SR3-T1

   Type: Functional, Dynamic, Automatic

   Initial State: Logged in to the administrative console

   Input/Condition: Search for a parking lot that is not the designated parking lot.

   Output/Result: Edit functionality disabled and analytics are not displayed.

   How test will be performed: A Cypress automation script will be created that will input a valid address of a parking lot that is not the designated parking lot and ensure that no editing functionality is enabled and analytics are not displayed for the parking lot.

Integrity Requirements

1. NFR-SR4-T1

   Type: Structural, Dynamic, Manual

   Initial State: Logged into the database on the server

   Input/Condition: Change the status of an empty parking lot entry in the database to full and refresh the database

   Output/Result: The database entry would revert the status of the parking lot entry to the correct one, saying it is empty.

   How test will be performed: The developers would enter the database and change an empty parking lot entry in the database to full, refresh the database, and ensure that the entry was reverted back to the correct status, which is empty.

2. NFR-SR5-T1

   Type: Functional, Dynamic, Manual

   Initial State: Logged in to the Administrative console

   Input/Condition: Edit and save the parking lot layout and disconnect the system from the internet.

   Output/Result: Parking lot layout is cached and a prompt is shown indicating that the unsaved layout is cached and the location.

   How test will be performed: The tester will manually login to the administrative console with proper credentials, edit the designated parking lot, and disconnect from the internet. The tester will then see a prompt with saying the layout was successfully cached locally and the link to the location.

3. NFR-SR6-T1

   Type: Functional, Dynamic, Manual

   Initial State: Logged in to the Administrative console

   Input/Condition: Edit and save the parking lot layout and disconnect the system from the internet.

   Output/Result: System attempts to upload a cached parking lot layout every $ATTEMPT\_UPLOAD\_TIME$ seconds and provides a message that includes the countdown till the next attempt to upload.

   How test will be performed: The tester will manually login to the administrative console with proper credentials, edit and save the designated parking lot, and disconnect from the internet. The tester will then check the developer console for HTTP requests every $ATTEMPT\_UPLOAD\_TIME$ seconds to upload to the server as well as a message that provides the countdown till the next upload.

4. NFR-SR7-T1

   Type: Structural, Dynamic, Automatic

   Initial State: Logged into the database on the server

   Input/Condition: Check the last time the parking lot layouts on the server have been backed up.

   Output/Result: Each entry must have been backed up at 11:59PM daily.

   How test will be performed: An automated pipeline would be created and ran daily to facilitate the checking of each of the parking lot layouts.

5. NFR-SR8-T1

   Type: Structural, Dynamic, Automatic

   Initial State: Logged into the database on the server

   Input/Condition: Attempt to add an entry with a different format in the database from other parking spaces.

   Output/Result: The database would produce an error and deny the addition of the inconsistent entry.

   How test will be performed: An automated test script will be created to test the data consistency of the database through attempting to add an invalid entry and ensuring that an error is produced and the addition is denied.

6. NFR-SR9-T1

   Type: Structural, Dynamic, Automatic

   Initial State: Logged into the database on the server

   Input/Condition: Attempt to add an entry with more than *MAXIMUM_SPECIAL_PROPERTY* special property.

   Output/Result: The database would produce an error and deny the addition of the inconsistent entry.

   How test will be performed: An automated test script will be created to test the data consistency of the database through attempting to add an entry with more than *MAXIMUM_SPECIAL_PROPERTY* special property and ensuring that an error is produced and the addition is denied.

7. NFR-SR10-T1

   Type: Structural, Dynamic, Manual

   Initial State: Database is offline and logged in to the Administrative console

   Input/Condition: Attempt to edit and save the parking lot layout.

   Output/Result: Parking lot layout is cached and a prompt is shown indicating that the changes are unable to be uploaded to the database and that it is cached locally.

How test will be performed: The developers would have to manually change the database to offline, log into the administrative console, edit a parking lot layout, and observe to see if a prompt is shown indicating that the changes are unable to be uploaded to the database and that it is cached locally.

8. NFR-SR11-T1

   Type: Functional, Dynamic, Automatic

   Initial State: Logged in to the Administrative console

   Input/Condition: Edit a parking lot layout and prompt the upload of the layout to the database server.

   Output/Result: Initiates the upload of the parking lot layout, prompts the user that an upload is underway, and a prompt will appear upon success indicating that the layout is successfully uploaded.

   How test will be performed: A Cypress automation script will be created that will edit a parking lot layout and prompt the upload of the layout to the database server. It will verify that the upload was initiated and completed successfully along with the prompts that the upload is underway and that the layout is successfully uploaded respectively.

9. NFR-SR12-T1

   Type: Functional, Dynamic

   Initial State: Logged into the database on the server, download the parking lot layout data represented in terms of a weighted undirected graph

   Input/Condition: Find the shortest path between a given starting node to a designated end node

   Output/Result: The return value should be the set of edges from the starting node to the end node

   How test will be performed: The parking lot layout data is retrieved from the database and reconstructed as a weighted undirected graph. Then the developers dynamically test the connection and path between a starting node and designated node by running a path finding algorithm such as Breath-first search.

10. NFR-SR13-T1

    Type: Functional, Dynamic, Manual

    Initial State: System is operating, but the navigation request is set to be rejected by the back-end service

    Input/Condition: User requests navigation

    Output/Result: The system prompts a message indicating an error has occured.

    How test will be performed: The navigation request is set to be rejected by the back-end services by the developer. The developer then can test whether a error message prompt appears when it cannot get the response from the server.

Privacy Requirements

1. NFR-SR14-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is running on the user's device

   Input/Condition: N/A

   Output/Result: N/A

   How test will be performed: Testers will check that a form pops up on the first time launching the system to ask for permission to use the user's location.

5.2.7 Legal Requirements

Compliance Requirements

1. NFR-LR1-T1

   Type: Functional, Dynamic, Manual

   Initial State: The system is operating

   Input/Condition: The user is in navigation mode and while moving, the user attempt to interact with the website

   Output/Result: A notification window should pop out, informing the driver to follow the regulations

   How test will be performed: Developer will sit in the passenger seats with navigation on, while the other developer will drive the vehicle. The developer on the passenger seat will attempt to use the website but is prompted to follow the regulations.

5.2.8 Health and Safety Requirements

Compliance Requirements

1. NFR-HS1-T1

   Type: Functional, Dynamic, Manual

   Initial State: System is operating.

   Input/Condition: User navigates between web pages from the system

   Output/Result: The browser renders different web pages from the system

   How test will be performed: Developer will manually test and navigate between different web pages from the system and make sure no flashing graphics or disturbing imagery exists

5.3 Traceability Between Test Cases and Requirements

Included in this section are traceability matrices that maps test cases to functional requirements and non-functional requirements outlined in the SRS.

Table 2: Traceability Matrix for Test Cases and Functional Requirements - Part 1

		Functional Requirements												
		FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13
Test Cases	BE1-FR1-T1	X												
	BE1-FR2-T1		X											
	BE1-FR3-T1			X										
	BE2-FR4-T1				X									
	BE2-FR5-T1					X								
	BE2-FR6-T1						X							
	BE2-FR7-T1							X						
	BE2-FR8-T1								X					
	BE3-FR9-T1									X				
	BE3-FR10-T1										X			
	BE4-FR11-T1											X		
	BE4-FR12-T1												X	
	BE4-FR13-T1													X

Table 3: Traceability Matrix for Test Cases and Functional Requirements - Part 2

		Functional Requirements											
		FR14	FR15	FR16	FR17	FR18	FR19	FR20	FR21	FR22	FR23	FR24	FR25
Test Cases	BE4-FR14-T1	X											
	BE5-FR15-T1		X										
	BE5-FR16-T1			X									
	BE6-FR17-T1				X								
	BE6-FR18-T1					X							
	BE6-FR19-T1						X						
	BE6-FR20-T1							X					
	BE6-FR21-T1								X				
	BE6-FR22-T1									X			
	BE6-FR23-T1										X		
	BE7-FR24-T1											X	
	BE7-FR25-T1												X

Table 4: Traceability Matrix for Test Cases and Functional Requirements - Part 3

		Functional Requirements											
		FR26	FR27	FR28	FR29	FR30	FR31	FR32	FR33	FR34	FR35	FR36	FR37
Test Cases	BE7-FR26-T1	X											
	BE8-FR27-T1		X										
	BE8-FR28-T1			X									
	BE9-FR29-T1				X								
	BE9-FR30-T1					X							
	BE10-FR31-T1						X						
	BE10-FR32-T1							X					
	BE10-FR33-T1								X				
	BE11-FR34-T1									X			
	BE11-FR35-T1										X		
	BE12-FR36-T1											X	
	BE12-FR37-T1												X

Table 5: Traceability Matrix for Test Cases and Non-Functional Requirements - Look & Feel, Usability & Humanity, and Performance

		Non-Functional Requirements											
		LF1	LF2	UH1	UH2	UH3	UH4	PE1	PE2	PE3	PE4	PE5	PE6
Test Cases	NFR-LF1-T1	X											
	NFR-LF2-T1		X										
	NFR-UH1-T1			X									
	NFR-UH2-T1				X								
	NFR-UH3-T1					X							
	NFR-UH4-T1						X						
	NFR-PE1-T1							X					
	NFR-PE2-T1								X				
	NFR-PE3-T1									X			
	NFR-PE4-T1										X		
	NFR-PE5-T1											X	
	NFR-PE6-T1												X

Table 6: Traceability Matrix for Test Cases and Non-Functional Requirements - Operational & Environmental and Maintainability & Support

		Non-Functional Requirements			
		OE1	OE2	MA1	MA2
Test Cases	NFR-OE1-T1	X			
	NFR-OE2-T1		X		
	NFR-MA1-T1			X	
	NFR-MA2-T1				X

Table 7: Traceability Matrix for Test Cases and Non-Functional Requirements - Security

		Non-Functional Requirements													
		SR1	SR2	SR3	SR4	SR5	SR6	SR7	SR8	SR9	SR10	SR11	SR12	SR13	SR14
Test Cases	NFR-SR1-T1	X													
	NFR-SR1-T2	X													
	NFR-SR1-T3	X													
	NFR-SR2-T1		X												
	NFR-SR2-T2		X												
	NFR-SR3-T1			X											
	NFR-SR4-T1				X										
	NFR-SR5-T1					X									
	NFR-SR6-T1						X								
	NFR-SR7-T1							X							
	NFR-SR8-T1								X						
	NFR-SR9-T1									X					
	NFR-SR10-T1										X				
	NFR-SR11-T1											X			
	NFR-SR12-T1												X		
	NFR-SR13-T1													X	
	NFR-SR14-T1														X

Table 8: Traceability Matrix for Test Cases and Non-Functional Requirements - Legal and Health & Safety

		Non-Functional Requirements	
		LR1	HS1
Test Cases	NFR-LR1-T1	X	
	NFR-HS1-T1		X

6 Unit Test Description

For non-functional requirements, we will utilize a unit test library to perform unit testing for this toolbox. To create unit tests for the backend portion of the program, we will generate a corresponding test file for two key modules, which will include unit tests for every function within the module. These tests will consist of a range of inputs, including those that are essential for the success of updating the parking spot availability in real time. For the frontend user interface portion, we created a couple of test files with Cypress, a UI testing framework, one for each main page of the application (i.e. Login Screen, User Mode, Admin View). These tests consists of input validation logic, ensuring that the CSS of the UI elements are correctly defined, and that certain elements are displayed on the screen dependent on their user mode (if they are an administrator or a user).

To evaluate the thoroughness of our testing, we will employ coverage metrics. Specifically, we will utilize the coverage.py Python library for the backend, which enables swift analysis of code coverage for all project modules. Our target is to attain 80% code coverage per module, which will guarantee that all functions are tested comprehensively. For the frontend, we be using Istanbul, which plays very well with the UI automation framework we are using and allows us to get a full analysis of the code coverage. We wish to similarly achieve at least 80% line coverage for the frontend.

6.1 Unit Testing Scope

Database module as well as parking stats module will be verified for correct functionality (correct sample inputs output correct sample outputs). For the Authentication Module, it would be verified for successful logins and error handling. The machine learning module will not be verified through unit testing as the model is obtained from DarkNet with pre-trained layers.

In terms of the importance of the modules for verification, below is a table that ranks each module in the Module Guide and Module Interface Specification based on the priority for verification. We rank using High, Medium, and Low.

Module	Priority
Video capture module	High
View module	High
Machine learning model module	High
Database module	High
Parking Stats module	Medium
Admin console module	Medium
Admin module	Medium
Parking lot layout module	Medium
Parking layout element module	Medium
Parking spot module	Medium
Authentication module	Medium
Navigation module	Medium
User action handler module	Medium
User module	Low
Vehicle module	Low

Table 9: Module Verification Priority

6.2 Tests for Functional Requirements

These tests refer to functional requirements outlined in the Software Requirements Specification.

6.2.1 Database Module Test Cases

Test 6.2.1.1:	Saving admin annotation coordinates from the user interface into the database.
Test file:	src/test/testDB.py
Requirements:	FR5, FR6, FR7, FR36, FR37
Description:	Tests if the annotation coordinates for ML model is properly saved in the database with a identifier that is unique to a parking lot
Type:	Unit test (static, automated)
Initial State:	The user has enter the annotation page.
Input:	Json object includes the annotation coordinate under the "CamCoords" key.
Output:	Response body object
Pass:	The response body object contains the previously saved CamCoords data.

Test 6.2.1.2:	Saving admin annotation coordinates from the user interface into the database.
Test file:	src/test/testDB.py
Requirements:	FR5, FR6, FR7, FR8, FR36, FR37
Description:	Tests if the annotation coordinates for user view is properly saved in the database with a identifier that is unique to a parking lot
Type:	Unit test (static, automated)
Initial State:	The user has enter the annotation page.
Input:	Json object includes the annotation coordinate under the "MapCoords" key.
Output:	Response body object
Pass:	The response body object contains the previously saved MapCoords data.

Test 6.2.1.3:	Saving YouTube stream link
Test file:	src/test/testDB.py
Requirements:	FR5, FR6, FR7, FR8
Description:	Tests if the YouTube stream link with a parking lot is saved in the database with a identifier that is unique to a parking lot
Type:	Unit test (static, automated)
Initial State:	
Input:	The user has enter the YouTube link.
Output:	Response body object
Pass:	The response body object contains the previously saved YouTube link data.

Test 6.2.1.4:	Updating model input source
Test file:	src/test/testML.py
Requirements:	FR5, FR6, FR7, FR8, FR9, FR11
Description:	Tests if the model input video can be correctly set
Type:	Unit test (static, automated)
Initial State:	Model is running as usual
Input:	The user enter the parking lot owner name and id
Output:	The current_input key on the database
Pass:	The string URL under the current_input key should be the same as the YouTube URL for the corresponding parking lot

Test 6.2.1.5:	Updating model input annotation coordinates
Test file:	src/test/testML.py
Requirements:	FR5, FR6, FR7, FR8, FR9, FR11
Description:	Tests if the model can correctly retrieve the annotation coordinates
Type:	Unit test (static, automated)
Initial State:	Model is running as usual
Input:	The user enter the parking lot owner name and id
Output:	The current_coordinate key on the database
Pass:	The JSON object under the current_coordinate key should be the same as the coordinates for the corresponding parking lot annotations

6.2.2 Parking Stats Module Test Cases

Test 6.2.2.1:	Analytics data updating
Test file:	src/test/testAnaly.py
Requirements:	FR34, FR35
Description:	Tests if hourly analytics data is successfully appended to the database
Type:	Unit test (static, automated)
Initial State:	Model is running
Input:	User append the analytic 1 hour data to the database
Output:	The up to date analytic data with a testing parking lot
Pass:	The returned analytic data should include the recently appended data of the parking lot

6.2.3 View Module

Test 6.2.3.1:	Successful Login
Test file:	cypress/e2e/Login_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Tests the successful login of an administrator
Type:	Unit test (static, automated)
Initial State:	Login screen is presented.
Input:	Valid email and password for the parking lot owners/team
Output:	Administrative console
Pass:	Redirected to their respective parking lot administrative console

Test 6.2.3.2:	Unsuccessful Login - Invalid Email
Test file:	cypress/e2e/Login_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Tests the unsuccessful login attempt of an administrator with an invalid email
Type:	Unit test (static, automated)
Initial State:	Login screen is presented.
Input:	Invalid email for the parking lot owners/team
Output:	Error message
Pass:	An error message is shown indicating an invalid email was given and prompts to try again.

Test 6.2.3.3:	Unsuccessful Login - No real user
Test file:	cypress/e2e/Login_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Tests the unsuccessful login attempt of an administrator with an email that does not exist
Type:	Unit test (static, automated)
Initial State:	Login screen is presented.
Input:	Non-existent email and password for the parking lot owners/team
Output:	Error message
Pass:	An error message is shown indicating the email does not exist within the database and prompts to try again.

Test 6.2.3.4:	Unsuccessful Login - No input
Test file:	cypress/e2e/Login_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Tests the unsuccessful login attempt of an administrator with no input
Type:	Unit test (static, automated)
Initial State:	Login screen is presented.
Input:	N/A
Output:	Error message
Pass:	An error message is shown indicating no input is given and prompts to try again.

Test 6.2.3.5:	Unsuccessful Login - Wrong Password
Test file:	cypress/e2e/Login_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Tests the unsuccessful login attempt of an administrator with a valid email but incorrect password
Type:	Unit test (static, automated)
Initial State:	Login screen is presented.
Input:	Valid email and incorrect password for the parking lot owners/team
Output:	Error message
Pass:	An error message is shown indicating an incorrect password is given and prompts to try again.

Test 6.2.3.6:	Redirection to User Mode
Test file:	cypress/e2e/Login_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Tests the proper redirection to User Mode
Type:	Unit test (static, automated)
Initial State:	Login screen is presented.
Input:	Click button to navigate to User Mode
Output:	User Screen
Pass:	Redirected to the user screen

Test 6.2.3.7:	Admin privileges
Test file:	cypress/e2e/Admin_Test_Suite.cy.js
Requirements:	FR32, FR33, FR34, FR36, FR37
Description:	Test if an admin has all the proper privileges
Type:	Unit test (static, automated)
Initial State:	Admin is logged in
Input:	Open sidebar
Output:	Sidebar opens
Pass:	Checks if all the correct features are included in the sidebar

Test 6.2.3.8:	Admin Logout
Test file:	cypress/e2e/Admin_Test_Suite.cy.js
Requirements:	FR31, FR32
Description:	Test if an admin can logout
Type:	Unit test (static, automated)
Initial State:	Admin is logged in
Input:	Click on logout button
Output:	Login Page
Pass:	Redirected to login page

6.3 Tests for Nonfunctional Requirements

Unit testing for nonfunctional requirements is not applicable for this project.

6.4 Traceability Between Test Cases and Modules

Test Number	Modules
6.2.1.1	Database Module
6.2.1.2	Database Module
6.2.1.3	Database Module
6.2.1.4	Database Module
6.2.1.5	Database Module
6.2.2.1	Parking Stats Module
6.2.3.1	View Module
6.2.3.2	View Module
6.2.3.3	View Module
6.2.3.4	View Module
6.2.3.5	View Module
6.2.3.6	View Module
6.2.3.7	View Module
6.2.3.8	View Module

Table 10: Trace Between Unit Test Cases and Modules

7 Appendix

Additional information that supports this documentation is provided in the following sections.

7.1 Symbolic Parameters

$MAX\_TAPS = 2$
$MIN\_FRAMERATE = 15$
$CONCURRENT\_USERS = 10$
$PARKING\_LOTS = 2$
$DEFAULT\_DELAY = 10$
$ATTEMPT\_UPLOAD\_TIME = 30$
$MAXIMUM\_SPECIAL\_PROPERTY = 1$
$SUPPORTABILITY\_SATISFACTION = 80\%$
$EXPLORATION\_TIME = 5$ minutes%
$TIMEFRAME = 6$ months%
$TIMEOUT\_TOLERANCE = 60$
$DISTANCE\_TOLERANCE = 5$
$DRIVING\_ESTIMATE\_TOLERANCE = 60$

7.2 Usability Survey Questions?

1. How responsive was the system in providing directions to your desired parking space? [1: Lowest, 5 Highest]

2. How responsive was the system in providing new directions when your desired parking space became unavailable? [1: Lowest, 5 Highest]

3. Were you able to reach your desired parking space using the given directions? [Yes/No]

4. Is the system visually/stylistically similar to Google Maps, Apple Maps, or Waze? [Yes/No]

5. Is the system as easy to use as Google Maps, Apple Maps, or Waze? [Yes/No]

6. Were you able to use the system to find an available parking spot? [Yes/No]

7. Were you satisfied with the instructions and contacts provided in the interface? [Yes/No]

8. How often would you use this system when looking for parking spaces? [1: Very Infrequently, 2: Somewhat Infrequently, 3: Occasionally, 4: Somewhat Frequently, 5: Frequently]

7.3 Reflection

1.

2.

7.3.1 Manual Testing

Despite the advent of automated testing tools, which are indeed useful, not all features of our system can be tested this way. For example, to validate one of our requirements, we need to test that a new set of directions are provided to the user if the current set of directions are infeasible to follow. As such, we need to test this feature by asking one of the testers to stand in front of the vehicle and check that the driver receives a new set of instructions. This verification cannot simply be done using an automated approach.

Learning Approaches

1. Observation Skills: It is inadequate to simply fail a test on the principle that the output did not match the expected result. In order to hone your manual testing skills, you must pay close attention to all of your actions and the corresponding system response, such that the failure can be reproduced in the future.

2. Analytical Skills: To improve your manual testing skills, you must think critically about the reason why the test scenario failed because the system may have been in a state that it should have not been in. Furthermore, by being more analytical, you can think of more ways to break the system and thus, reduce the number of bugs and glitches.

- Almen: To improve my manual testing skills, I believe working on my analytical skills should be the priority. With such a complex system, having analytical skills would help with breaking it up into smaller components to gain better understanding of the components of the system and how they interact with each other and collecting data needed to make decisions and suggestions.

- Kabishan: I believe that I can improve my manual testing skills by mastering my analytical skills. In doing so, not only will I test our system to a higher depth, but I will also improve my ability to test my software solutions at work. I will improve my analytical skills by consulting my notes from 3S03, the Software Testing course, and attempting manual testing interview questions online.

- Jonathan: To further improve my manual testing skills, I believe that focusing on my observation skills would be useful. A failure to meet a test case is not always as clear-cut as it may seem. The context of the failure and use case may give further insight to the underlying issue. Improving my observation skills would allow me to better identify these details when manually testing our system.

- David: I will focus on observation skills so I can verify that an error has been resolved by reproducing the steps exactly. This way I can more reliably know when the program

is behaving as expected. It will also help me look closely at the code to predict why an error may have arisen.

- Albert: I will look to improve my observation skills as there are too many times that a test fails because I wasn't paying enough attention and put the wrong initial conditions.

- Gary: I will try to improve my analytical skills because as the complexity of our software increases, we have to separate the system into different individual parts for testing and then integrate it later.

7.3.2 Learning a New Automation Framework

Learning how to use new automation frameworks, specifically Cypress, could be challenging for group members who do not have experience with it.

Learning Approaches

1. Dedicate a time during our meetings for Almen, who has professional experience using Cypress, to explain how to use Cypress and ensure that people have the general knowledge to automate UI test cases.

2. Learn individually through online documentation, videos, and tutorials on Cypress's website.

- Almen: Already proficient in developing UI test cases, I would gravitate towards the later option to strengthen my understanding on the matter and aid my team whenever it arises.

- Jonathan: As I do not have any experience with automated testing frameworks, I would prefer to dedicate meetings with Almen. This would allow me to more efficiently learn how the framework works and specifically how it integrates with our specific system and use cases.

- Kabishan: My automated testing experience lies mostly in iOS and Android testing through Appium. While the foundations of creating test cases are the same, irrespective of platform, scheduling some time with Almen will allow me to expedite the exploration of any nuances and subtleties of the Cypress framework.

- David: I will aim to self-learn the Cypress framework, and hopefully Almen can provide advice for any specific questions I may have.

- Albert: As someone with experience, it would be best to follow Almen as she already knows the scope of the framework for our purposes.

- Gary: I would prefer to have meeting with Almen and learning about this framework as this is more efficient. Learning from documentation can be time-consuming and not all the details are needed for the purpose of testing this project.

7.3.3 Validating Plans

It is just as important to know how to verify and validate our test plans as it is to create the test plans. It can be difficult to set the criteria that effectively and sufficiently validate the plans.

Learning Approaches

1. Learn individually from books or online resources about how professionals in the industry verify and validate their plans effectively.
2. Collaborate with professors and peers to see how their thought process when laying out their plans and how they verify and validate them.

- Jonathan: I believe that this is an aspect that I have a lot of room to improve in. Because of that, I think a mix of both learning approaches would suit me best. In my opinion, collaborating with professors and peers would be easier to learn from. However, I think that it is also important to learn how industry professionals go about verifying and validating their plans.
- David: I will collaborate with peers and classmates to see how they are verifying their plans. A variety of perspectives is beneficial in learning the best way to do something, and I will be able to combine the best practices from each person I ask.
- Kabishan: In order to better verify and validate our test plan, I will learn from books and online resources, and I will also ask professionals in the industry about how they perform this task. As enticing as the second option sounds, it would be very difficult for me to arrange a time with my professors and my peers to master this skill.
- Albert: I wish to collaborate with others as it is a more hands on experience and can provide more help than what was asked for.
- Gary: I personally believe it is always better to learn from professional in the industry about how they verify their products and plans effectively because they have done it countless number of times and they have very mature methodology.
- Almen: I would like to approach the latter option of collaborating with professors and peers as this provides first hand insight on what is needed to verify and validate plans. The former option might have a lot of unnecessary content that may not be relevant to us.

7.3.4 Structuring Peer Review

A well-structured peer review allows us to derive deeper insight on topics that are important, rather than surface-level feedback on the topic of the reviewer's choice. It's important to know how to write good questionnaires, for example, while directing our reviewers' attention to the most impactful topics in our documentation and design.

Learning Approaches

1. Learn from the surveying principles taught in SFWRENG 4HC3, including questionnaire and interview formatting

2. Study peer review formats from previous courses and online resources

- David: I will apply the practices we learn in SFWRENG 4HC3 in structuring peer review. We have practice with these from assessments in that course, and it will be interesting to apply my learning to other courses.

- Kabishan: In order to ask better questions, and therefore derive more effective feedback from our users, I will learn from the surveying principles taught in SFWRENG 4HC3. For example, instead of asking open-ended questions, which are difficult to compare for multiple participants, asking with a range of answers would be better.

- Jonathan: I would prefer to apply the principles and knowledge we have learned about reviewing from SFWRENG 4HC3. I have found many of the principles we have covered in that course interesting and applicable to our project.

- Albert: There's a lot of research done for understanding people and asking the correct questions. Using these resources may be overwhelming, but can provide more information than from one course.

- Gary: I am happy to apply the knowledge I learned from 4HC3 class and I will look through the resources provided from that course to understand these principles better.

- Almen: I wish to apply my knowledge taught in SFWRENG 4HC3 as the course has provided a lot of valuable information in constructing proper questionnaires that has been previously assessed in the course.

7.3.5 Static Testing

Static testing can be very difficult and time consuming for those not used to it. It requires a full understanding of the program and is not only about getting the right output.

Learning Approaches

1. Practice with small programs on our own and compare against static testing tools

2. Work together on static testing a part of the system and let the main contributors lead the testing.

- Albert: I have never used static testing tools before, so I think comparing my abilities to them will provide a good indication of my abilities.

- Kabishan: During SFWRENG 3S03, I worked on several practice problems that required me to use static testing to identify faulty code that may never be uncovered at run-time. In addition to this knowledge, I will practice with small programs on my own and compare my results against static testing tools, such as SonarQube, to ensure that our project does not crash during execution.

- David: I will practice on small programs of my own, such as previous projects. This will be a good way to see where my shortcomings are, and it will be good practice because I won't have worked on these projects for a significant time.
- Jonathan: I think that practicing on smaller programs would be a more effective way to learn about static testing and improve my skills in that area. I would then be able to apply the skills I have learned to our project when putting our modules through static testing.
- Gary: I have learned some static testing tools before but I haven't used it for a while, so I think the best way for me to refresh the memory is to practice with a small dummy program and refresh the memory.
- Almen: I personally would choose to practice on a smaller program, such as ones that can be found online, and do a code review on the program. I would then compare it with static testing tools to validate what I have found. That why, I would be able to improve my skills in static testing and be able to contribute to static testing of the project.

7.3.6 API Testing

Having a back-end service that is intended to handle user web request also needs to be tested through an API testing tool, such as Postman. API testing can speed up the testing process compared with UI testing by simply changing the input parameters of each web request sent to the server and provides a form of early validation for the data and response of the back-end system.

Learning Approaches

1. Download Postman software and follow their user guidelines and check their documentation on the website.
2. Ask a group member who has relevant experience with it.

- Jonathan: As I have no experience with the framework, I think that downloading the Postman software and going through their guidelines and documentation would be the best way for me to learn about the API testing framework. I would probably supplement this with online videos or tutorials to further understand how the framework works.
- Kabishan: Even though I have previously tested endpoints using another software, ReadyAPI, I believe that I can still benefit from following the user guide provided by Postman, as there may be industry proven techniques that identify certain oversights when designing the back-end API.
- David: I plan to download the software and follow their guidelines and documentation myself. Postman's website hosts tutorials which I hope I will find useful. Any questions I have can be directed to a group member with experience to speed up the learning process.

- Albert: Learning a new tool can be an interesting experience but can also take up a lot of time. Getting tests wrong because you were using the tool wrong can lead to even more time loss. Relying on someone else who is experienced with it would be best.
- Almen: I believe that having organizing a knowledge transfer session with a group member with experience in this should be done as they would provide a lot more relevant information that will be useful to the project. Checking the documentation can be overwhelming and not all the information will be used.
- Gary: I have prior experience with API testing and Postman therefore I can check the documentation when questions comes up.