Department of Software Engineering
Lakehead University

ESOF 4969 – DEGREE PROJECT 2022-23

Non-Invasive Automatic Parking Spot Finder:
A Mobile Application

Design Document

Group 3

Contributing Members:
Nicholas Imperius 0645031
Jimmy Tsang 1098204
Mason Tommasini 1110432

Date Submitted: 04/16/23

Approval

This document has been read and approved by the following team members responsible for its implementation:

PRINT NAME: Nicholas Imperius
Signature:

Nicholas Imperius

PRINT NAME: Jimmy Tsang
Signature:

Jimmy Tsang

PRINT NAME: Mason Tommasini
Signature:

Mason Tommasini

COMMENTS:
Individual Contributions are as follows:
Nicholas Imperius: Introduction, Literature Survey, System Specification, System Design (all subsections), System Implementation (all subsections), Testing Methodology, QA Standards, Project Timeline, Conclusion, and Discussion
Jimmy Tsang: Introduction, Functional and Non-Functional Requirements, Design Impacts (all subsections), Project Timeline, Member Contribution, Conclusion, Limitations & Future Work
Mason Tommasini: Abstract, Software Process Methodology, System Implementation (all subsections) Use Cases, Test Cases and System Faults, Conclusion, and Discussion

Non-Invasive Automatic Parking Spot Finder: Design Document

Abstract

This paper describes and represents the development completion for the smart parking application project. This paper details the requirements, specifications, design, implementation, and testing of how the application was constructed. The application uses different algorithms to determine whether the user has parked their car or not, all whilst recording this information in a cloud database for up-to-date information to be available for all users. The requirements and specifications of the application are clearly defined and detailed on. The applications components and interface are laid out in a manner that is understandable and thorough. The system implementation deals with how the hardware and software interact with each other, their capabilities, and the system's modularity as well as quality. The parking algorithm designed in this mobile application uses GPS sensors within the user's smartphone to determine if a parking space should be filled or emptied. Testing is also important to show the different use cases and system faults to ensure quality standards have been met. Design impacts to deal with possible risks that come with the creation of the system as well as the societal and economic impacts as well as the costs and profits that come with creating the system. Upon completion of the developed application, a cost calculation was performed and compared to the estimated results from October 2022. Our findings determined that while the development cost was only slightly higher than the estimated cost, we had developed a larger program than initially intended but felt that the difficulty of development had a reduced impact compared to our initial estimations. Societal and economic impacts are discussed in detail, along with methods to generate revenue from our developed solution.

Acknowledgement

We would like to express gratitude towards the Department of Software Engineering at Lakehead University for this opportunity and Dr. T. Akilan, Chair of the Software Engineering Department, for being our project supervisor and providing us with continuous support, feedback, and advice.

Contents

Chapter 1

Introduction

It can be attested that Lakehead University faces the issue of inadequate parking spaces for students and staff, causing unnecessary stress and wasted time. According to Lakehead University [1], there are approximately 7000 full-time students. By conducting an analysis of the amount of G parking lot parking spaces, it was found that there were approximately 1200 spots in total. This is a substantial difference and at peak times of the day, there could be increased stress and time spent locating a vacant parking space. To address this problem, we have developed a cutting-edge mobile application that offers a seamless, fast, and effective solution to the existing parking problem; the name of the application is *Link & Park*. The software was designed to provide a hands-free solution for users to find an open parking spot upon arriving at the university, eliminating the need to loop around the parking lot until a satisfactory spot is found. The application would provide real-time information regarding user location, the differing types of parking spaces, and their availability, whether they are occupied or available. The application would provide additional benefits for the user, such as increased efficiency, reduced stress, and lower environmental impact of a vehicle by reducing carbon emissions and unnecessary idling. Our team has conducted thorough training and testing of the software, with a focus on the R9 sub-lot of the G campus parking lot at Lakehead University; however, the application has the potential to be expanded to public parking structures such as shopping centre lots or parking structures.

From a technical standpoint, this application will be quite distinct from other pre-existing smart parking applications. When entering the keywords "parking application" into the Google Play Store and the Apple App Store, we are greeted with various different applications to help users with their parking needs, although no application had the same algorithm and idea as ours. Most of the applications were found to help users find different parking locations throughout cities while showing parking fees, time, and more. The difference is our application shows the vacancy of a parking lot to show the user how busy the lot is. By viewing an application called "Parkopedia Parking" on March 25$^{\text{th}}$, 2023, we can see that the application has features such as parking lot prices, availability, and distance to the user. This application has more than 1 million downloads and a rating of 4.6 stars. When looking at the majority of applications on the Google Play Store, we found that most applications are free and have no additional in-app purchases for more features. This is a key indicator, and our application will most likely be free to first establish our initial user base, before considering more features that could be unlocked via payment methods.

Instead of relying on external hardware sensors in each parking space or internet protocol cameras to locate and detect empty parking locations, this application will use the GPS location, speed, and network information from a user's personal mobile device to determine whether you have parked within a defined parking spot and notify other users in real-time through a cloud database that the

parking space has been filled. Additionally, when you have left your parking spot, our application is able to detect that and mark the parking space as free accurately. The application determines your action based on your speed and location before and after a full stop; it is able to recognize when the user is within the bounds of a parking space and the differences in speed between walking and driving, which allows it to determine whether the user has parked their car.

This paper details the overall design of the smart parking application that was designed. The design document is outlined as follows: Chapter 1 provides an initial introduction; Chapter 2 provides a summary of related literature in this domain with similar solutions to this problem; Chapter 3 details the requirements, specifications, and software process methodology; Chapter 4 uses diagrams to outline the construction of the design of the system and how components integrate together; Chapter 5 discusses the various hardware and software used within the application as well as its modularity and quality; Chapter 6 gives an understanding into the testing methodology, different use cases for validation aspects, and quality assurance standards; Chapter 7 provides an overview into the various decisions and impacts that result from the design choices of application; Chapter 8 outlines the individual contributions of each group member throughout this design project; finally, Chapter 9 provides a conclusion of the overall findings presented in this report.

Chapter 2

Literature Survey

With the uptrend in the number of vehicles on the road, finding available parking locations has been a frustrating procedure for many commuters. Reducing the time it takes to find a parking spot, which, in turn, increases productivity and creates more time for more important tasks. In this section, we explain current research in this domain, the methods being used, and the success rate of these methods.

Athira *et al.* [2] studied how different solutions can be used to solve the problem of parking lot congestion and control. They noted that using sensors, whether complex or simple, can have a high cost and be quite complex in terms of installation and setup. Image processing techniques were used, since the accuracy has increased over time, to visually determine if parking spaces are available within the parking lot. They used optical character recognition to coordinate with specific labels and numbering that each parking space would have to determine which particular parking space was available. Their solution to use image identification was found to be more cost-effective when compared to the commonly found sensors placed at each parking space.

Ma *et al.* [3] aimed to prevent parking lot-related accidents compared to the efficiency and congestion-related problems. Their solution used image identification techniques in conjunction with machine learning and pattern recognition methods in order to plan routes for vehicles to take within a parking lot to navigate to a parking space, as well as when leaving the parking lot. The idea was to add a level of vehicle automation so that drivers could have reduced stress and increased ease too.

Aftab *et al.* [4] studied the impact of a smart parking application reducing the time it takes to search for a parking space along with the environmental impacts associated with it. They designed an Android application called ParkUs and users would be able to select their destination, get directions, and confirm when they have parked. Their approach to determining if a user has parked revolved around looking at the timestamps from the cruising data they had, determining where stops were made, and then choosing the closest parking location. They used this approach because the majority of the time, a user would choose the closest parking space to their location. They trained three different models, a decision tree classifier, k-nearest neighbours, and a support vector machine. It was concluded that the support vector machine performed the best in their trial. They noted that there was a decrease in carbon emissions due to the lesser amount of driving that was being conducted as well.

How *et al.* [5] discusses the current problems surrounding parking infrastructure in cities and their parking structures or areas. They presented a solution that used QR codes in conjunction with a mobile application. Their solution could use license plate recognition to detect cars coming to park that have already booked the parking spot. In addition, if the weather conditions were not

good, a QR code was present that could be scanned to confirm the parking location. This allows parking authorities to have a better understanding of which cars are allowed to be parked in the parking spots. Their goal was to lower the time spent searching for a parking spot.

Torres *et al.* [6] presented an e-recommended parking mobile application called P.E.T.E.R Parking. Their approach used rapid application development to obtain a business model canvas for a start-up project. Their application also incorporated user input to help improved recommendations too. The model that they created also aimed to improve revenue for the owners of the parking spaces and in their findings was able to achieve a rating of *highly usable* from the users and the parking lot owners had a positive response too.

Anand *et al.* [7] presented a real-time system that allowed drivers to locate unoccupied through a web or android application. Their proposed solution incorporated hardware and software components. The drivers would be issued an RFID card that would be used when entering the parking premises, and the information would be sent to their Raspberry Pi database. From here, a parking token would be granted to the user and directions to the parking spot would be provided through their mobile application. In the end, their solution reduced the time spent searching for parking spaces and helped reduced traffic congestion at peak times while being user-friendly enough for anyone to operate.

Mahendra *et al.* [8] described a solution to car parking issues by utilizing Internet of Things (IoT) devices for interconnectivity of devices, which would allow devices to gain access to surrounding objects. Their model was an IoT sensor that would be used in conjunction with a mobile application to allow users to pre-reserve parking spaces. They used IR sensors and a Raspberry Pi to compute if a parking space was taken; however, their system was designed for three parking spaces, with each having its own IR sensor. Thus, a limitation of this would be the scale and cost associated with large-scale configurations.

Ng *et al.* [9] presented an outdoor parking space detection model that would be used through a mobile application. Their model used a CNN and images of the parking spaces through a security camera to determine if the parking spaces were empty or full. They used a university parking lot as a testing facility for real-time vacancy detection. Their mobile application provided a visual for their users of the empty parking spaces and users could provide feedback on the model's correctness which would help it learn for future reference. It was found that most users found the app useful in determining the best location to park, which they treated as a success.

Tahmidul Kabir *et al.* [10] discussed a solution to recognize vacant parking spaces for areas where free parking can be an issue. Their solution used both hardware and software; a YOLOv3-tiny model was trained to detect vehicles based on the images from their hardware. On the software side, they used a web application for the client side and a SQL database for the server side. Users were able to utilize the model through a mobile application that would display a map with pins showing available parking or parking that can be pre-booked too. Their system was fully automated for online function and, while performing duties offline, required very minimal intervention from a human. Finally, their system had a low cost associated with it, which helps with large-scale implementation concerns.

Lewis *et al.* [11] presented a mobile application for saving parking locations of your vehicle. Their solution was a mobile application that would be able to provide other features, such as a compass or directions to their parking location. The model had to correctly identify when their vehicle was parked as well as when it departed the parking location as well. To track this, they observed the user's speed throughout time and stored information whenever it thinks the user has parked. When trying to provide directions to the parking spot, the model would then look up the collected data and give the user the best results to find their parking spot. The model had correctness measured

at 94%, which is practical enough for usage in the real world.

Mangiaracina *et al.* [12] developed a simulated model of urban parking in Milan, Italy, before and after their smart parking solution was implemented. For their Intelligent Transportation System model, they implemented several technologies, such as GPS, WSNs, and Gateways, in order to accommodate the traffic of both truck and car drivers. They placed 80,000 sensors around parking spots in the city and developed a mobile application that would point users to the nearest possible parking spot available. It was found from their results that this implementation could save the average person around 77 hours per year and cut fuel costs by €86.5. Most importantly, CO_2 emissions would be reduced by 44,470 tons per year in the city of Milan.

Chapter 3

System Requirements & Specification

Before any system development occurs, we must define the system requirements and specifications. Requirements aid in development by defining and dictating how the system will be developed. There are functional and non-functional requirements that are outlined as follows. Furthermore, the design methodology for this software product is detailed as well.

3.1 Functional Requirements

As detailed in the software requirements document, the functional requirements dictate the features that this system must be able to perform successfully for the application function as designed. The following key functional requirements will need to be implemented:

- Display a birds-eye-view of the G parking lot on the campus of Lakehead University.
- Allow the user to zoom in or out on the map view to locate available parking locations.
- Users will be shown an indicator on each parking space in the parking lot to see if it is available or not.
- Users should be able to view the current parking location of their car.
- When a user has stopped driving, parked their vehicle and started walking away from the vehicle, the parking algorithm should be able to determine this and mark the location as not available.
- When a user has gotten in their vehicle and driven away, the machine learning model should be able to recognize this and mark the parking location as available.
- The mobile application will recognize when the user has arrived at campus and have precise tracking turned on.

3.2 Non-Functional Requirements

Non-functional requirements are those that do not directly influence how the program must be constructed in terms of functionality, unlike functional requirements. These are requirements that focus more on the operation of the application, such as performance capabilities or limitations in

place that can improve or hinder the program's efficiency. Such non-functional requirements needed include:

- Speed: The application must update the user's position at a speed that mimics the real-time movement and run with minimal latency.
- Security: The application must be secure to not allow intruders to access any sensitive information.
- Usability: The application must be easy to navigate and require little effort from users to learn how the application operates.
- Reliability: The application must be reliable, and able to detect the user's position as accurately as possible.
- Capacity: The application must be able to handle multiple users at any given time, this could require a large database as roughly 97% of post-secondary students own and use a smartphone [13].
- Compatibility: The application must be able to function on the widest range of mobile devices possible.

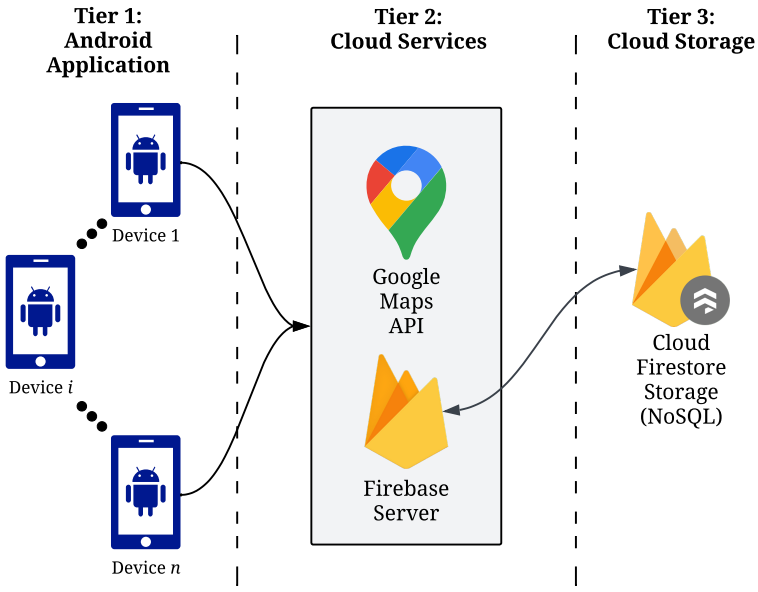


Figure 3.1: A layout of the 3-tier architecture that the application has been designed with where the number of active devices is from $1, 2, ..., n$, where i is a number between 1 and n.

3.3 System Specification

A 3-tier architecture, shown in Fig. 3.1 is the designated architectural style for this parking application. The application relies on two cloud services, namely the Google Maps application program

interface (API) and Firebase Servers. These cloud services offer management and other services, such as retrieving database information or gathering an up-to-date Google Maps layout. Additionally, a database that holds user information in addition to parking-related information is crucial too; therefore, we have our cloud storage handled through Cloud Firestore using the querying language NoSQL.

The visualization of the architecture and design of each function or progress within the parking application allows for a greater understanding of how each component within the system operates. Furthermore, comprehending how components communicate with each other within the system allows for more optimized methods of handling function calls, relaying information between APIs and activities. The majority of our application relies on information that is collected through the user's mobile application in addition to various APIs in order to meet the requirements from the initial requirements document.

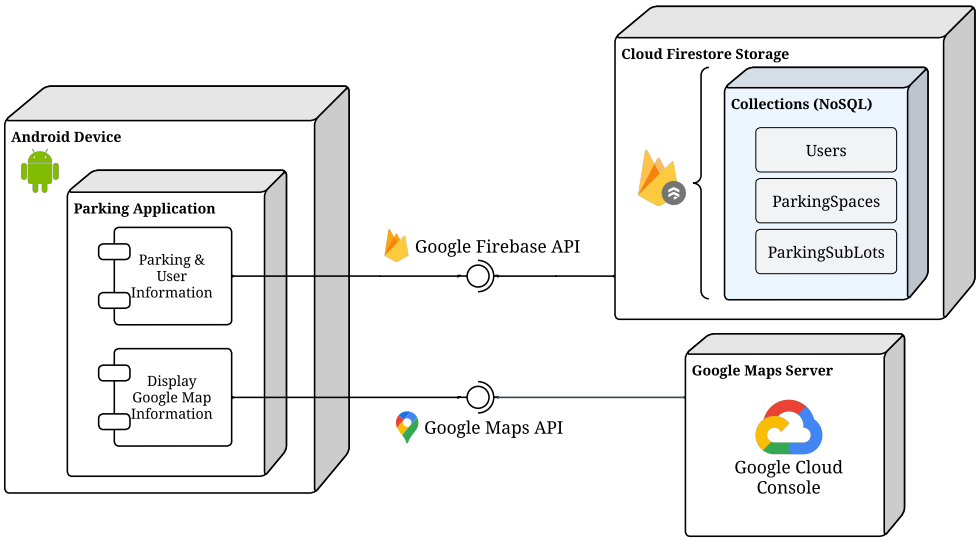


Figure 3.2: Deployment diagram that details how the Android application will communicate with external systems.

The deployment diagram shown in Fig. 3.2 details the communications between the application on an Android phone to the Cloud Firestore database as well as the Google Maps server on Google Cloud. The component that handles the parking and user information within the application uses the Google Firebase API to access the collections of documents stored within the Cloud Firestore NoSQL database. Likewise, the Google Map that is displayed within the app gathers the information from the Google Cloud by connecting to the application through the Google Maps API.

3.4 Software Process Methodology

For this project, the waterfall model was used, shown in Fig. 3.3, as it is a linear-sequential life cycle used for a relatively quick development period. The waterfall model is good because it makes the project predictable with its different milestones, making it easy to measure progress. This allows us to work at a suitable pace and ensure that everything is completed before moving on to the next phase, which makes the waterfall method a suitable choice for this project. Choosing a more complex model may overcomplicate the methodology of the project due to the scope of the project

and time constraints. At any stage in the life cycle, if we feel that something is missing or needs to be remodelled, we can proceed to a previous phase of the cycle, rather than continuing forward.

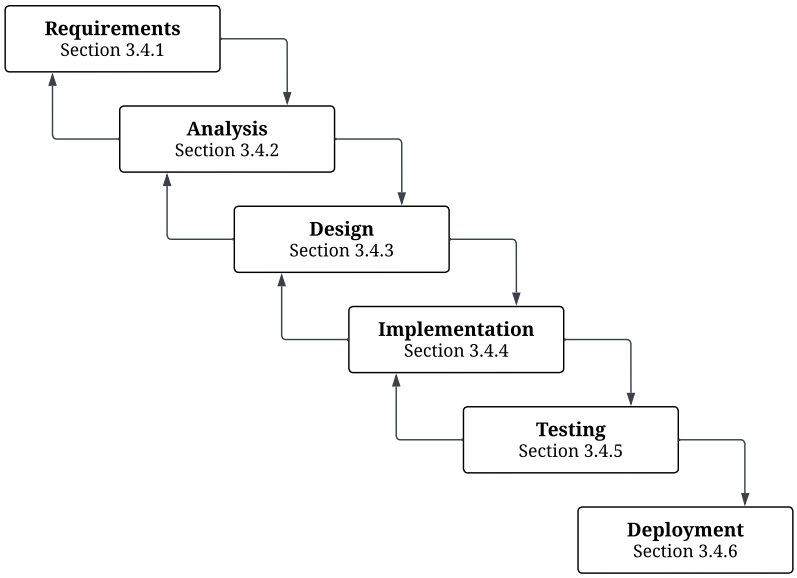


Figure 3.3: The waterfall software development life cycle used throughout the development of this application.

3.4.1 Requirements Phase

The requirements phase involves identifying project requirements, deadlines, and any restrictions. This phase also includes understanding the project domain, both functional and non-functional requirements, and identifying necessary software or hardware. A clear project outline and task delegation are necessary for the subsequent phases.

3.4.2 Analysis Phase

The analysis phase involves analyzing project specifications to generate models and determine the required functionality for each component. The phase also involves breaking down each component and its subcomponents to achieve the requirements from the previous phase. A process for completing the subcomponents is created for smooth execution.

3.4.3 Design Phase

The design phase involves creating a full specification document for the product design and technical components, including required programming languages and hardware. The phase also involves designing solutions for each subcomponent, including system architectures, interfaces, and libraries. In this project, the design phase includes creating detailed wireframe figures for mobile user interfaces and outlining the database design.

3.4.4 Implementation Phase

The implementation phase involves developing and integrating all components outlined in the design document to create a working prototype of the product. The components will be created for each aspect of the application and converted into code, then properly implemented into modules. The goal is to have an alpha version of the final mobile application ready for testing for integration issues and bugs by the end of this phase.

3.4.5 Testing Phase

The testing phase involves testing the components and the system as a whole to ensure all functionality works as described and to find and fix coding errors. If errors are found or updates are needed, the implementation phase is revisited to fix them, and testing is repeated. The analysis phase would have outlined testing procedures to ensure the functionality meets requirements, and all requirements must be met before proceeding to the final phase.

3.4.6 Deployment Phase

The deployment phase involves implementing the final working prototype of the product as if it were ready for publishing to app stores. Maintenance would be required periodically to ensure the software remains bug-free and stable for users. Additionally, more features and functionality could be developed and pushed as a software update to the application if needed.

Chapter 4

System Design

For a more detailed look as well as a demonstration video, Appendix A has the necessary resources.

4.1 Database Design

The database was designed with NoSQL using services provided through Google Firebase, specifically their Cloud Firestore Storage. An entity relationship diagram is shown in Fig. 4.1 which details how the different collections are connected. Each collection is connected through an identification number. The *spaceID* in the Users collection is related to the *spaceID* in the ParkingSpaces collection. Likewise, *subLotID* is related to the *subLotID* in the ParkingSubLots collection. Furthermore, each parking space can only be assigned to one and only one parking sub lot, but each parking sub lot can have at least one parking space. Each user can be in zero or one parking space, and each parking space can have zero or one parked user.

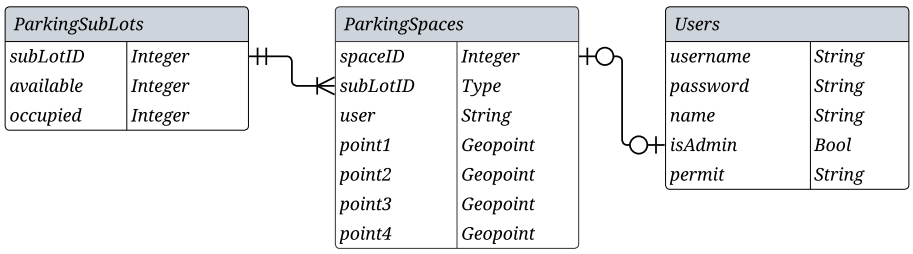


Figure 4.1: Database entity relationship diagram of the NoSQL database. The three collections each have document entries with the specified fields of information.

Constant changes and updates are made in real-time to the cloud database. When users create or update their account information through the mobile application, these changes are updated in real-time on the cloud to their respective objects in the database to reflect the changes.

4.2 Application Interaction Design

The activity diagram in Fig. 4.2 details the general flow of the parking application with its current set of features. When the user opens the application on their phone or tablet, they are prompted

to log in or register if they are a new user. If they choose to register, then the application will bring the user to a new screen that will prompt them for information about themselves and their parking permit. If the registration is successful, then the system will perform a few functionalities in the background; this includes updating the database with the new user information and then granting access to the application, all while the application gathers the Google Maps information. Initially, if the user went to log in instead of register, they would have been brought straight to the main screen, maps activity. From here, they would be able to see all the available and unavailable parking locations within the parking lot. In this activity, the user could log out or close the application when they are done, or they could go to the settings screen, info activity, that allows them to change any details about themselves. When the user decides to reopen the application, they will be brought to the map activity as the application remembers the user's information which is stored locally on the device and therefore bypasses the login and register screen.

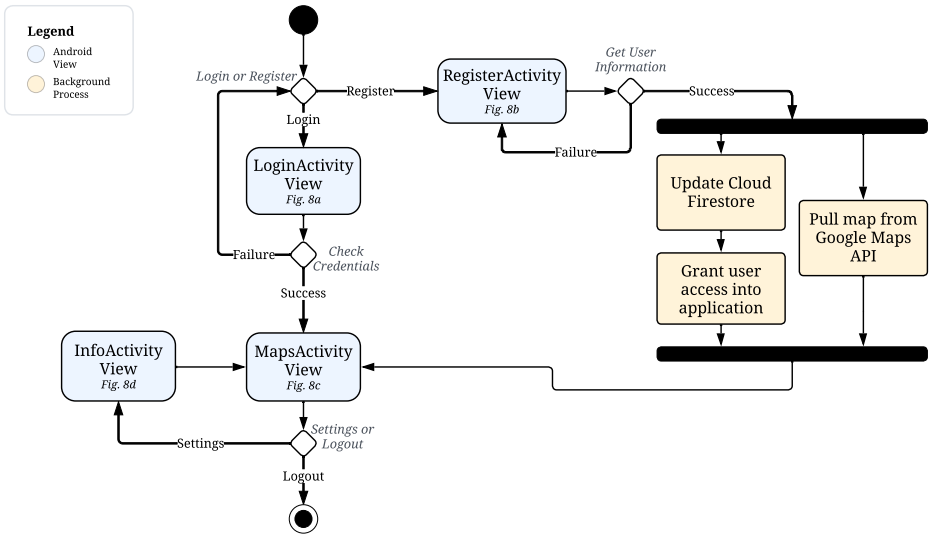


Figure 4.2: An activity diagram that shows the general flow and directions a user can take when using the application.

4.3 Algorithm Design

Algorithmically, this project is quite straightforward. An algorithm was designed, simplified in Fig. 4.3, using background services within the application, similar to a creating a thread. Once a user has entered the campus geofence, the background service will begin, where it waits for the user to make stops before executing a series of code. At every stop, the algorithm will check to see if they have parked within a parking space, or even if they are in between multiple parking spaces. The application will go through the list of parking spaces that the user is currently parked in and filter them based on if they can be parked in or not. These candidate parking spaces are then sent to a function that will determine the distance to the centre of the parking space. The user would be assigned to the parking space they are closest to the centre of. Once this has all finished, the user's actions after the stop are monitored; if they are walking after this stop, then the application assumes they have parked since they were just driving. If instead they were driving after making

a stop, we can assume they have left the parking space if they were walking before the stop. The algorithm mainly relies on the GPS location of the user, during our testing we found that once a user of the application has stopped, the GPS triangulates and increases in accuracy which is beneficial for determining the correct parking location.

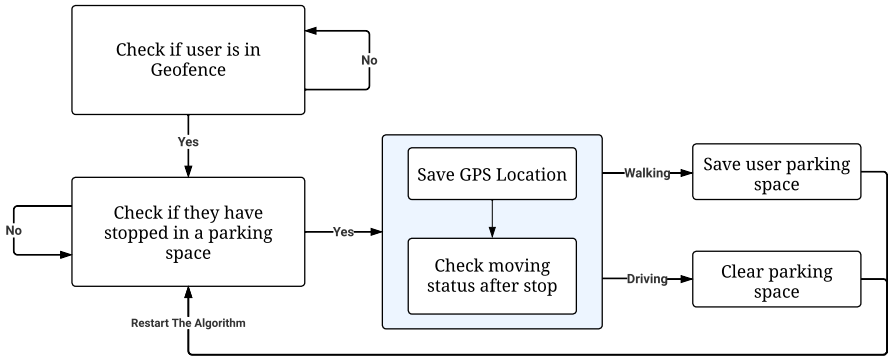


Figure 4.3: A block diagram describing the steps that the application conducts in order to determine if a parking space has been parked in or left.

Big O notation is a method for determining the complexity of an algorithm depending on how intricate the size of the input is. With our current implementation, we have determined the complexity to be of level $O(n)$. There are threads that run concurrent to other activities, but there exist no nested loops in the executing code. The code structure is not complex in nature, which is crucial to maintaining a low response time for in-app computations and updates.

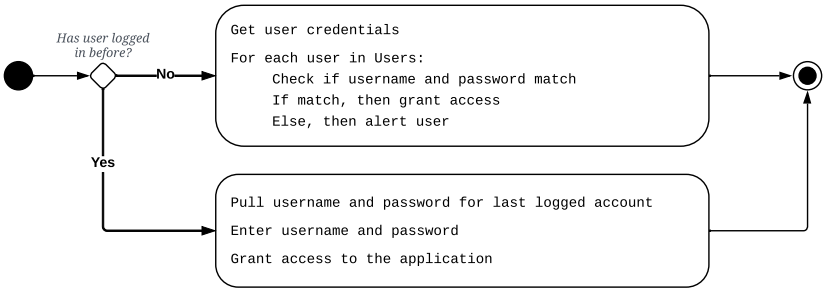


Figure 4.4: The process of a user opening the application on their Android device. This determines if they have logged in before and if they should be auto-logged in, or if a username and password need to be collected from the user.

Fig. 4.4 details, with pseudocode, the process of a user when they open the application. Initially, the application determines if they have logged in to the application before through the use of the *SharedPreferences* functionality. *SharedPreferences* is a method of storing information locally on the device within the application storage, rather than storing information online. If they have logged in before, the visually described steps show how the system handles this algorithmically even if the user has never logged in before too.

The algorithm that is used to compute the course of action when the update information button is selected inside the settings screen is shown in Fig. 4.5. The system checks to see if the text fields have information within them and update the fields accordingly. The information that the user sees

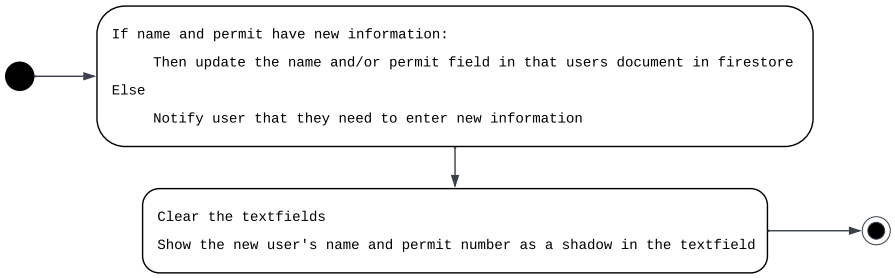


Figure 4.5: The internal process when the update button is clicked on our settings screen.

is then updated as well. The text fields are cleared, and then the shadow or hint text within them is updated to the new information that the user provided.

4.4 Key Design Principles

Throughout the design process of the application, the following ten design principles were kept as a higher priority in the decision-making process for different design choices.

1. Innovation
    - Based on our research, there has yet to be a competing product that performs similarly to this.
    - It's a smart, cost-effective solution to assisting with parking on a day-to-day basis.
2. Usefulness
    - The large majority of individuals carry a smartphone on them, and since it is a mobile application, it is quick and easy to take out your smartphone and open the application.
    - Being able to locate your parked car after a long day is very convenient and often can save someone from wasted time wandering the parking lot.
3. Aesthetics
    - The design philosophy within the mobile application was to take a modern approach, but not overcrowd and take attention away from the core purpose with a distracting design.
    - Aesthetically, the colours are similar to Lakehead University's colours in some aspects to retain the cohesion between the application and the intended target audience to make it feel more natural for the users.
4. Least Possible
    - Most users that use the application will want the information they seek as quickly as possible. They could be in a vehicle on campus or looking to approach their vehicle after class.

- This quick-and-ready idea was met with a design that reduced the amount of user feedback required to complete actions. That is, the application automatically determines if you are parking or leaving a parking space, limited screens to navigate, and tracking via the Google Maps view.
- Parking spaces are shown immediately, as well as a legend to quickly identify the availability or type of parking space it is.

5. Understandable

   - In order to make the application easier to understand for the average user, subtle design choices were made around the application. For instance, the settings button on the main maps screen contains the word *settings* rather than a gear symbol traditionally found on other applications, or the *find my car* button explicitly says its intended purpose compared to a graphic or icon that may not depict its function.

6. Honest

   - Having reliable and trustworthy information in this type of application is crucial to its success.
   - A user should be able to trust the information that they are seeing on the application is correct and the most up-to-date.

7. Thorough

   - The application is straightforward and simple for a first-time user.
   - There does not exist a lengthy user manual for someone to gain an understanding of how to use the application, it is well laid out and defined clearly for even experienced parkers.

8. Environmentally Friendly

   - One of the application's benefits was reducing an individual's carbon footprint by reducing any unnecessary, extra driving throughout the parking lot.
   - The application was also designed to be more efficient in terms of battery consumption on a mobile phone to help reduce e-waste over time. Battery degradation can play a huge role in why someone upgrades their phone to something new, and reducing this impact would help our user's devices last longer.

9. Economic Perspective

   - Reduced driving time when searching for a parking space results in less fuel consumed by the vehicle. This decreases the carbon footprint but also reduces the amount of money that needs to be spent on average to fill up for gas.
   - From a marketing and commercial perspective, campus security could be partnered with to ensure the validity of user parking permits on campus, and we could even take a percentage of the parking permit fees in exchange for operation and control with our application.

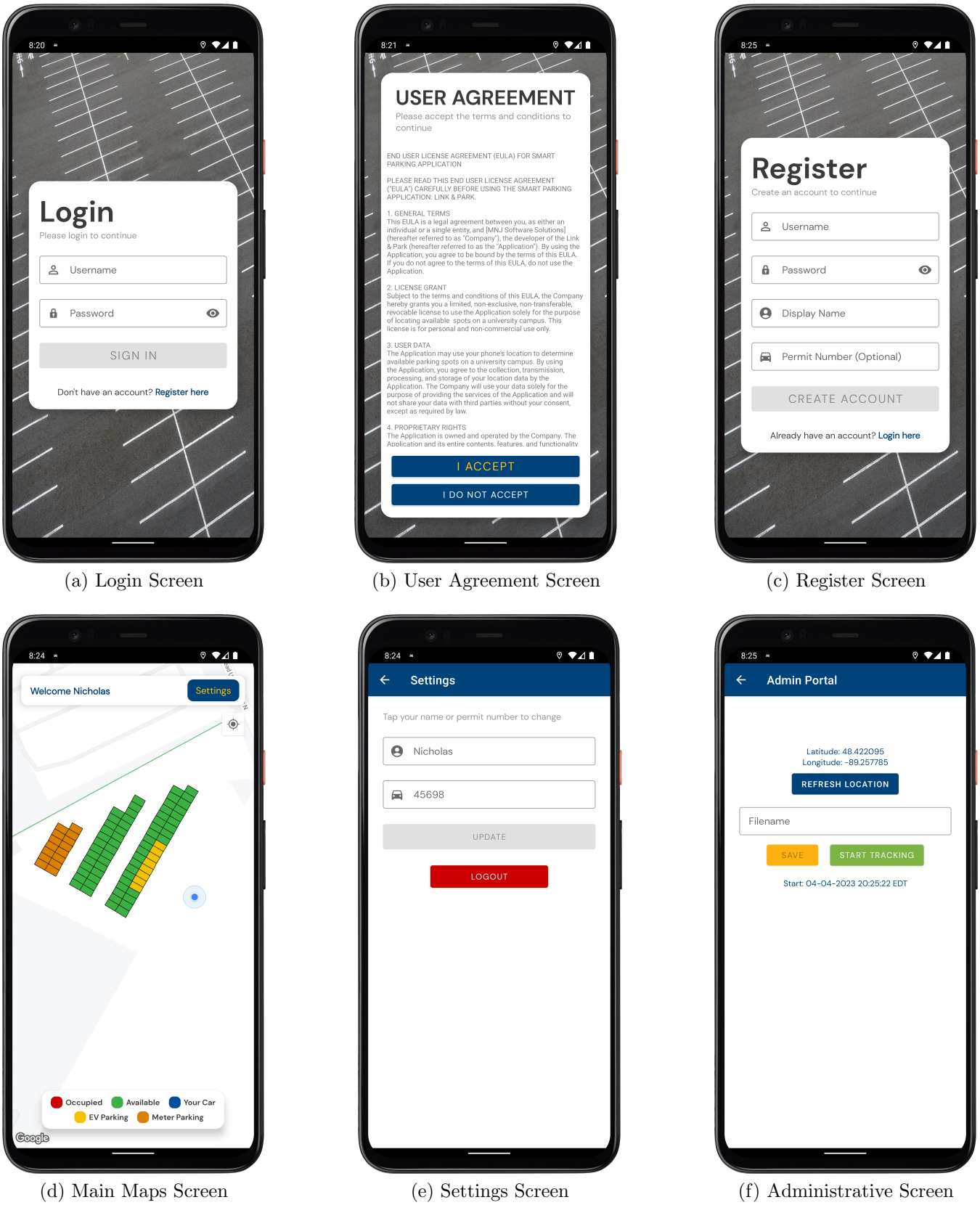
10. Equity, Diversion, and Inclusion

    - Every user of the application will retain the same experience as another.

- Every member of the development team has and will have equal opportunities and standards.
- The development team consists of a diverse group of individuals.
- Ethnicity, gender, age, or sexuality are not discriminated against when using the application.

4.5 User Interface Design

The user interface design for the Android application is aesthetically pleasing while also being thorough and understandable. Every user will be presented with the login screen, shown in Fig. 4.6a, where they can log in if they have valid credentials, or they can proceed to the register screen, shown in Fig. 4.6c, if they are a first time user. In order to get to the register screen, there is a helpful text message below the *sign-in* button that they can tap to get to the next screen. This design had the most positive feedback when showcasing different options to other peers. In order to register an account with our application, one must accept the end user license agreement, which ensures their acceptance of the specific tracking features and more that are application requires. Once a user has successfully logged in or registered their account, they get presented with the screen in Fig. 4.6d, where a Google Map overview of the parking lot is displayed along with the availability of each parking space. If a user would like to adjust their account name or their linked permit number, the settings screen in Fig. 4.6e is where they would be able to do such. This can be accessed via the settings button located on the main maps screen. The administrative screen, shown in Fig. 4.6f, is where administrators are able to log tracking information; this is where a lot of our initial testing for our parking algorithm had information collected and actioned.



(a) Login Screen (b) User Agreement Screen (c) Register Screen

(d) Main Maps Screen (e) Settings Screen (f) Administrative Screen

Figure 4.6: The various wireframe screenshots of the (a) login screen, (b) end user agreement screen, (c) register account screen, (d) main maps screen, (e) settings screen, and (f) the administrative screen.

Chapter 5

System Implementation & Simulation

Definitions on how the system was implemented in terms of hardware, software, modularity, and quality are presented here. It is important that the system works on a wide range of hardware and software while still being modular to emphasize good coding guidelines.

5.1 Hardware

As this project is being developed using Android Studio, the current devices able to use the application are limited to Android devices running Android version 7.1 or higher. This application is roughly 12 MB in size, so almost, if not all, Android devices should have enough space to store the application. Since this application is designed for mobile applications, the user does not need any additional hardware aside from the smartphone they are using the application on. This smartphone will require a GPS sensor and wireless capabilities. Hardware that the user may not be directly interacting with would be the cloud storage, hosted through Google Firebase, and the server that runs the Google Maps API. All of these are third-party controlled and specifications on the equipment are up to the discretion of said party.

5.2 Software

Android Studio: Android Studio is a powerful development environment that was designed for Android application development. It provides all the necessary tools that a mobile developer would need to complete their application. Two languages that are supported within Android Studio are Kotlin and Java, with the latter being the selected programming language for this mobile application. GitHub is also easily integrated within Android Studio, and since this was our way of merging different members' components and managing versions, this proved to be very useful. This collaborative coding environment is perfect for modular-based application development, as well as other styles too.

GitHub: This internet hosting service tool is used for integrating developed software between users and software version control. GitHub provides a simplistic space to store the developed program in addition to any supporting documentation, where authorized users can upload files and see the revision history. All source code, and other types of files, can be located in a GitHub repository. There are ways to track insights of contributors, as well as manage versions and track bugs. Programmers are also able to create branches off of the master file if they want to implement and test their own modifications without affecting the main source

code, and later when complete they would be able to merge the changes to the main source code.

Zoom/Email/SMS: Zoom is a video call communication platform used for interacting with different team members of a project, work, or school environment. Zoom was our main way of hosting meetings with each other to discuss, share or implement new changes to the project. Sometimes there were in-person meetings, but for the most part, Zoom was used as it was much easier than meeting in person and allowed each member to share their screen. For text communications, we used emails to communicate with Dr. Akilan to ask any questions or schedule any in-person meetings. The three team members communicate using SMS (short messaging service) as it is a quick and effective way to communicate either directly with one team member or the entire group.

LucidChart: A business diagram creation program where charts, graphs, and complex figures can be created for projects such as application development in our case. LucidChart was the main software used for creating diagrams, as it allowed all members to contribute at the same time and allowed for in-depth customization and integration.

Overleaf: A collaborative cloud-based editing program which is based on the LaTeX language. It allows users to write, edit, and publish professional reports with advanced features that place a high emphasis on details. Overleaf can be used by multiple people at the same time, allowing each member to contribute to portions of the document at the same time while saving in real time through cloud storage. Overleaf allowed the group to create our final report in a professional and collaborative manner.

5.3 System Modularity

System Modularity is a design principle that encompasses breaking down a complex system into simpler parts or units that allow for easier testing and development. Since this application was designed using Android Studio, many of its features were intuitive and allowed us to code in a modular format that separated different pieces of code into different sections, or classes.

The application uses background services and broadcast receivers in order to monitor and pull from APIs while the application is closed or open. The services are standalone Java classes that implement specific methods in order to achieve their required functionality. Each activity within the Android application has a respective Java and XML file. The XML file dictates the layout of the screen, namely, what the user will interact with and see, while the Java file implements functionality to components laid out in the corresponding XML file. Java functions are used throughout the entirety of the application to reduce redundant, repeating code to decrease the lines of code. Furthermore, constants are used, which represent set parameters, and can alter the program by adjusting their value. For instance, the time to check if a user is stopped, walking, or driving can be increased or decreased via one single line.

Furthermore, the APIs used within the application provide robustness since we can implement a large amount of functionality with a few lines of code. For instance, to check if a user is in the database is a simple Firebase API call to an endpoint that returns a user, if successful, or nothing if it fails. Instead of creating a function and query to go through the database and check each field in the database, this API does everything for us.

Chapter 6

Testing & Evaluation

In order to ascertain that the software system is fully functional and free from potential bugs, it is essential to subject it to thorough testing and evaluation. Due to the nature of the application, various testing methodologies were used to ensure as much coverage from errors as possible. By building out a use case diagram for the different cases a user could perform in the application, implementing these testing methodologies was easier. Some test cases for output that was able to be procedurally done are shown here, along with the standards of quality assurance that were maintained in this project.

6.1 Testing Methodology

Since this is a mobile application, there are many testing approaches that can be taken to ensure the application's robustness and validity. The following list of testing methodologies was followed throughout the development of this application.

1. **Usability Testing:** This is where the usability and intractability of the user-interface portion of the application were tested, as well as its responsiveness. The application on a smartphone was provided to other individuals outside the development team, and they were asked to perform a task to ensure that it was intuitive enough and easy to understand. This is very similar to quality assurance testing.

2. **Functional Testing:** Ensures that the application is functioning as expected during execution. The flow between activities was tested to ensure that everything was correct, ensure that new users are displayed with the correct permission prompts and that nothing crashes the first time through the application, and more. Errors were also noted and acted on to ensure a smooth experience for the user.

3. **Security Testing:** Since we have a login mechanism within the application, even though there is only a small amount of sensitive information being stored, the prevention of data leaks and the security of the data have to be tested. Attempts to log in and authenticate as a verified user using false login credentials were tested. As well as verified users being able to access other user information, or even update and delete database entries were tested as well.

4. **Compatibility Testing:** Some non-functional requirements are tested here. Ensuring the success of the application on varying versions of Android operating systems that our application is supposed to be compatible with and making sure that everything operates the same

throughout both in terms of user-accessibility and functionality. Devices of different screen sizes were tested to validate that all information on the screen was still understandable. If a beta version of the application is published, this can be seen as compatibility testing too, as we are testing a version of the application itself.

5. **Performance & Load Testing:** Measuring the performance of the application while under particular loads or in certain situations is done here. The application was tested in the target environment with different phones that have different levels of cellular plans, i.e., 5G vs 4G vs LTE networks. The response time for an action interacting with one of the APIs or functions built into the application was tested and compared with one another to verify it is as low as possible. A lot of the features tested here are to increase user satisfaction.

6. **Manual Testing:** This is where specific, edge test cases were conducted that involve specific steps to test for an error or application crash and the application's ability to respond to or recover from such scenarios. This type of testing was also performed for ensuring the success of the parking algorithm. Simulations of different scenarios someone could take when parking or leaving a parking space were played out, in reality, to make sure the application performs as expected.

6.2 Use Cases

Fig. 6.1 displays a UML use case diagram of the different use cases within this application. There are three actors in the system, namely, the user, database, and application, with each having its own interactions in the system. A description of each use case is as follows:

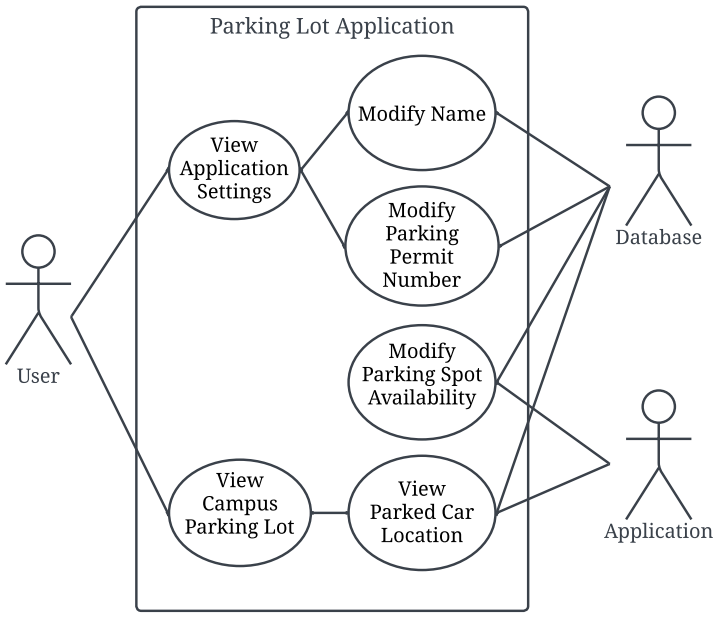


Figure 6.1: The use case diagram for our mobile application that displays the interactions between the user, database, and application.

View Application Settings

Enables the user to adjust their name and parking permit number directly inside the application:

(i) The user will click on the *gear* icon at the top right of the welcome screen.

(ii) The user will click on the name text field.

(iii) The **Modify Name** use case is used.

Modify Name

Enables the user to have their name updated in the database and have these changes reflected in the application:

(i) The user will enter their name into the text field.

(ii) The application will send this information to the database.

(iii) The database will update with the new changes.

Modify Parking Permit

Enables the user to have their parking permit number updated in the database and have these changes reflected in the application:

(i) The user will enter their parking permit number into the text field.

(ii) The application will have the database updated with the new information.

Modify Parking Spot Availability

The application will update availability information once it has recognized that a user has parked or left their parking spot:

(i) The application will assess the user's speed before and after the stop.

(ii) If the model determines that the parking availability has changed, then update the database; otherwise, do nothing.

View Campus Parking

Enables the user to view an overview of the entire campus G parking lot that the application is supported for:

(i) Show a satellite view of the campus G parking lot along with the user's current location.

(ii) The user will be able to zoom in and out on the map.

(iii) The user will be able to visually see which spots are available, taken, and their parking location.

View Parked Car Location

Enables the user to easily view where they parked their car on the satellite map:

(i) A visual indicator will describe the location of the vehicle to the user.

(ii) The user's current location will be shown to relate to where they are on the map.

6.3 Test Cases & System Faults

All the test cases for the system can be found in Table 6.1. This table represents the many possible interactions that can occur with the created system. For each of these interactions, it will have a designated type as well as its expected result and if it has passed or failed based on the expected result. Test cases, allow for developers and users to give different possible inputs into the system and receive an expected output. Based on this, changes can be made to improve the overall system and user experience as a whole before delivering an end product to the user or customer. Some aspects of testing can be performed through the emulator, such as account creation, modification, and access privileges, however, to test the functionality of the designed parking algorithm, we needed to physically attempt parking our vehicles within the R9 sub-lot.

Table 6.1: Various Test Cases Used In Testing Methods

Type	Description	Expected Result	Status
Usability	User creates an account	Account created and stored in DB	Pass
Usability	User changes username	Update username in DB	Pass
Usability	User changes permit number	Update permit number in DB	Pass
Usability	User logs out of account	User logged out	Pass
Functionality	User reopens app after creating account	Autofill login info	Pass
Functionality	User declines user agreement	Returned to home screen	Pass
Security	User creates an account	Password stored in DB as encrypted	Pass
Security	Unauthorized user changes DB	Denied permission	Pass

6.4 Quality Assurance Standards

The International Organization of Standards (ISO) developed the ISO 9000 standard, which is a set of quality management standards that outlines a series of principles to be followed in order to ensure that software meets all quality management measures [14]. ISO 9000 consists of the following, seven quality management principles [15]:

1. Customer Focus

2. Leadership
3. Engagement of People
4. Process Approach
5. Improvement
6. Evidence-Based Decision-Making
7. Relationship Management

Throughout the development of this application, ISO 9000 standards were kept in mind and the seven principles were followed as closely as possible. Customer focus was followed by doing a thorough analysis into the domain, as well, we are a part of the target audience, which means we have an understanding of what would be beneficial and what would not too. Each member of this team took their turn being the sole leader throughout the project. They took on leadership duties, such as meeting minutes, organizing and delegating tasks, and more. This level of trust and vision was able to let us succeed to the point we are at today. Each of us has a strong suit in one of the applications of our project; therefore, each person is actively engaged in creating as well as explaining components to one another. Open discussions of ideas and adjustments were always welcomed too. Our process approach consisted of grouping a set of tasks into a process or activity which was managed in terms of scope and time so that there was a streamlined approach to high-quality task completion. Improvements were made during the development process to enhance the quality of the product. Throughout the testing procedures of the application, different evidence was observed and collected that required decisions to be made on how components were constructed. Going back and adjusting was able to lead to a higher quality for the application. Since different members have more experience in different aspects of the application, exploiting the relationships between each other to maximize the quality of the application was crucial.

Chapter 7

Design Impacts

7.1 Risk Analysis

7.1.1 Privacy Concerns

With all applications that store user data and location, privacy concerns must be addressed to ensure that user data stays confidential even in a data breach or any form of attack. Over-the-shoulder attacks occur with mobile applications when a perpetrator watches the user enters crucial information using the on-screen keyboard, to prevent this and ensure the user has a secure password, we have hidden the characters being typed into the password field, whilst also implementing certain restrictions on the password itself. When a user creates a password, we require a combination of both capital and lower-case characters, in addition to a numerical value and a special character. Without these requirements, the user's password would not be accepted, and they will be prompted to create a new password. Additionally, once the password has been accepted and the account is generated, the password information is encrypted to provide another layer of security in the cloud database. Attackers would not be able to decrypt the password from the cloud database, as the decryption key is stored locally on the user's device.

Regarding user location, some may feel uncomfortable with the application tracking their precise location. Addressing these concerns, we have implemented a feature to prevent precise tracking outside the geofenced location. Outside the geofence, the application will only use approximated location tracking, this can be done as we have set the geofence around Lakehead University at such a size that the application will have enough time to switch to precise tracking once it detects it is in the geofence.

7.1.2 Safety Concerns

Aside from protecting user privacy, we must ensure user safety when operating the application. We do not promote using this application in a hands-on approach while the user is operating a vehicle. As such, we have decided to design this application with the intent of using it in a hands-free manner. This is done by redesigning the built-in recenter button to accurately follow your location when clicked. With this, users can rest their devices in a hands-free apparatus, such as a car mount, and simply drive to their intended location, the application would update the map in real-time, so the user would stay in the centre of the map.

Security is also a concern when looking at the safety of users. Fortunately, this system does not allow a way for another user to get the location of them. The specific GPS coordinates are

only used within the application and the Google Maps API (which has its own security measures). Furthermore, since we are using a database that uses NoSQL, potential attacks, such as an SQL injection attack, could be a possibility. A NoSQL injection attack involves an attacker including infected input/code into a query that is delivered by the user; the infected portion will execute unwanted commands on the database [16]. To avoid this, we ensure that the inputs being sent to the database have all been sanitized and cleaned of any unwanted, malicious code. Additionally, we prevent unauthorized users from accessing any information within the database; this includes reading, writing, updating, deleting, and more.

7.1.3 Public Acceptance

The major concern with launching this application is the public perception or acceptance of such a design. For this application to function as desired at Lakehead University, we would require a majority of those who currently hold parking passes to register for and regularly use the application, as only spots parked in by those using the application are considered taken on the app. If users were to not use the app and park in an open spot, the database would be unable to detect the change and update it on the database. The largest consideration with a lack of public acceptance relates to privacy concerns regarding the tracking of user location. To mitigate this, we must ensure that the app will switch to approximate tracking when outside the geofence, and while the application is not in use or running in the background, it will not track user location. By alleviating these concerns, we feel that many would feel inclined to use the app, as it would benefit them greatly. More research would need to be conducted to determine public sentiment regarding such an implementation, this can be done through one-on-one communication, online polls or surveys, and at public events. When registering an account, there is a user agreement that each individual must clearly read, as it details important information regarding the application, our rights as the developers, the rights of the user, and what information is being held while the application is in use. The full license agreement can be found in Appendix B, where it details all requirements the user must understand before using the application.

7.2 Societal & Economic Impact

7.2.1 Societal Benefits

If the application is used as intended, it can lead to many societal benefits. With an assistive tool to visualize available parking spaces, it can lead to better time management, as less time is spent on average searching for an available parking space. Additionally, with less traffic circling the parking lots, there is a decreased risk of vehicle collisions. As of 2021, in Thunder Bay alone, 30% of vehicle accidents occur within parking lots [17]. Reducing the amount of time spent driving in a parking lot, will thus lead to a decrease in traffic accidents at Lakehead University, which provides a safer experience for both drivers and pedestrians. Lastly, as a consequence of having to drive around endlessly to look for an available parking spot, more fuel is consumed and more carbon is emitted into the environment. On average, a typical user of our application can expect to reduce their CO_2 consumption by up to 1 lb/week, based on the findings by Surpris *et al.* [18].

7.2.2 Economic Benefits

The use of this application provides economic benefits not only to the user of the application but towards the client that implements this smart-parking application in addition. By reducing the distance driven from searching for an available parking space, users will naturally burn through less fuel. According to Fybr [19], the average person could save 4 km per week in driving with the use of our application. It can be said that 1 km of driving in a vehicle that averages 10 L per 100 km would cost \$0.16 worth of fuel, assuming a fuel cost of \$1.60 per L. Based on Equation (7.1), an average savings of \$8.32 per semester may not seem a fair amount; however, when taken into account that each person of the over 7000 individuals that attend Lakehead University in some capacity saves such an amount, it begins to accumulate.

$$\$0.16/\text{km} \times 4\ \text{km/week} \times 13\ \text{weeks/semester} = \mathbf{\$8.32}\ \text{/semester} \tag{7.1}$$

From the perspective of the client, Lakehead University, this application can provide economic benefits, through its verification of users holding a valid parking pass. Currently, there is no existing method to verify that a parking space has been occupied by a user holding a valid parking pass aside from physically examining the spot and validating the parking pass. With this application, it has the ability to require that users obtain a valid parking pass before using the application, and while parking enforcement patrols, any taken spot that is seen as open within the application itself signifies that the individual does not currently hold a valid parking pass. This application provides a validation methodology to ensure that all individuals of Lakehead University have paid for a legitimate parking pass.

7.3 Project Cost & Profitability Perspective

7.3.1 Project Cost

Upon completion of the development of the smart parking application, we can now calculate the actual cost and duration of development, and compare it to our initial estimations. This is done by using the Intermediate Constructive Cost Model (COCOMO) method, first developed by Barry Boehm [20], for use with the waterfall method of development. Similar to the estimated value previously calculated, we must determine three key values. The number of lines of code specified in kilo-lines of code (KLOC), the intrinsic difficulty or development mode, and the nominal effort multiplier which is calculated as the product of 15 different cost drivers.

Like previously, we must first collectively determine the development mode or the intrinsic difficulty of development. Each development mode has its own set of multipliers, found in Table 7.1. Previously, we had estimated this project to be a semi-detached developed system given the three possible choices: organic, semi-detached, and embedded, ranging from least to most difficult, respectively. Upon completion, we feel that this is an accurate level of difficulty estimated, given our experience and expertise regarding the tasks at hand in addition to the project deadline. The team each had different experiences with certain aspects of the system, with some being experienced in some areas of design, while some areas were completely foreign at the time of development. Advancing forward, the next step is to determine the KLOC value of our application. It should be noted that white space and comments do not count towards this value, as it focuses on source instruction lines of code. KLOC can be counted manually or using a programming tool called Statistic, which analyzes the total number of lines of code, in addition to the source instruction

Table 7.1: Software Development Mode and Constants [20]

Mode	α	β
Organic	3.2	1.0
Semi-detached	**3.0**	**1.12**
Embedded	2.8	1.20

lines of code. Using this plugin, we have determined that collectively, we have written a total of 2927 lines of code, or 2.927 KLOC, compared to the initial estimate of 2.050 KLOC.

Once we have obtained these values, we can then determine the nominal effort to develop the system, represented as person-months, the number of months required for one person to complete development. The calculation of nominal effort in person-months for this project is found by applying the development mode multipliers to the KLOC, as seen in Equation (7.2).

$$\begin{aligned} \textit{Nominal Effort} &= \alpha \times KLOC^{\beta} \\ \textit{Nominal Effort} &= 3.0 \times 2.927^{1.12} \\ \textit{Nominal Effort} &= 9.99 \text{ person-months} \end{aligned} \tag{7.2}$$

Given that certain aspects of the application or the development team may affect development time in either a positive or negative manner, those factors need to be accounted for when determining the overall effort of development. As such, there are 15 different software development cost drivers that are considered when determining the overall development effort, each with ranging values for different levels of difficulties. These cost drivers can be separated into three main categories, product factors, platform factors, and personnel factors. By considering and applying these cost drivers as an additional multiplier, we can come to a more accurate calculation of the total development effort in addition to the overall project cost. The total list of cost drivers, along with our selection for the value of a given cost driver, is found in Table 7.2. Additionally, a brief explanation detailing what the cost driver relates to along with our justification for that value is found below.

Required Software Reliability – High = 1.15: This system must be reliable, as it must handle conflicts and system crashes from overload. It is the main interface for viewing available parking spaces and as such, must be active for the majority of the time. In addition, if the system crashed, the functionality of the application would disappear, rendering it useless; therefore, it needs to be developed reliably.

Database Size – Low = 0.94: The end goal of this database is to keep track of every G lot parking space at Lakehead University, with a capacity of approximately 1200 parking spaces, the database must individually keep track of the availability of all the parking spots. Given that the database should not have more than 10,000 values, we believe the database size is small in comparison to more mainstream databases.

Product Complexity – Very High = 1.30: We feel as the project complexity is quite high, the application not only tracks user location in real-time, it must account for their location relative to the geofence and all parking spaces. Additionally, we must measure the user's speed in addition to determining their action, whether walking to class or driving away from the spot and update that in real-time.

Execution Time Constraint – Nominal = 1.00: Very minimal in current technology compared to the technology of 1984 when Intermediate COCOMO was created. With technological

Table 7.2: Software Development Cost Drivers [20]

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Required Software Reliability	0.75	0.88	1.00	1.15	1.40	
Database Size		0.94	1.00	1.08	1.16	
Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Execution Time Constraint			1.00	1.11	1.30	1.66
Main Storage Constraint			1.00	1.06	1.21	1.56
Virtual Machine Volatility		0.87	1.00	1.15	1.30	
Computer Turnaround Time		0.87	1.00	1.07	1.15	
Analyst Capabilities	1.46	1.19	1.00	0.86	0.71	
Applications Experience	1.29	1.13	1.00	0.91	0.82	
Programmer Capability	1.42	1.17	1.00	0.86	0.70	
Virtual Machine Experience	1.21	1.10	1.00	0.90		
Programming Language Experience	1.14	1.07	1.00	0.95		
Use of Modern Programming Practices	1.24	1.10	1.00	0.91	0.82	
Use of Software Tools	1.24	1.10	1.00	0.91	0.82	
Required Development Schedule	1.23	1.08	1.00	1.04	1.10	

advancements since then, we can ignore the effect of this cost driver. Thus, we have chosen the smallest effort multiplier.

Main Storage Constraint – Nominal = 1.00: Storage is a very small item to worry about with how inexpensive large storage is today; therefore, we can ignore the effect of this cost driver and have selected the smallest effort multiplier.

Virtual Machine Volatility – Low = 0.87: The underlying volatility of the virtual machine for this software product can be considered to be quite low. We have selected the smallest effort multiplier.

Turnaround Time – Low = 0.87: Since this project has a small team that is working closely and our scope is well-defined, we can consider the turnaround time to be quite small since we are not dealing with separate teams within an organization. Thus, we have selected the smallest effort multiplier.

Analyst Capabilities – Nominal = 1.00: We had limited experience in this space aside from a few course projects regarding some areas of development. However, we felt capable in our abilities to produce a working application. Additionally, we have learned most of the required information before, but have simply not yet applied it to an entire system like this before.

Applications Experience – Nominal = 1.00: Application experience is nominal as we have some experience in some areas needed for the development of this prototype, however, we still lack knowledge in some areas due to limited experience.

Programmer Capability – Nominal = 1.00: Since some group members have been programming for 6–10 years, we are confident in our abilities to quickly learn, adjust and present a working prototype.

Virtual Machine Experience – Low = 1.10: With only about 1 year of experience in the fields required for this project, we feel that we also lack experience here.

Programming Language Experience – Nominal = 1.00: Some members of the group have experience with NoSQL (∼1 year), Python (∼6-8 years), Cloud Servers (<1 year), and mobile application development (<1 year) either from school courses or freelance learning.

Use of Modern Programming Practices – Low = 1.10: Due to a lack of experience, most of what we know about modern programming practices has been from in-class lectures rather than on-site practices.

Use of Software Tools – Nominal = 1.00: We have a strong foundational understanding of software tools as learned from school courses, however we still lacked experience with on-site or real-life applications.

Required Development Schedule – High = 1.04: The development schedule was carried out throughout the school year while also working on other course projects. Given the hard deadline for the completion of the project, there were some concerns regarding the development schedule.

From the collectively determined value for each of the cost drivers, we can then find the cost multiplier as the product of all the cost drivers multiplied together, as seen in Equation (7.3).

$$\textit{Cost Multiplier} = 1.339 \tag{7.3}$$

Using the new multiplier, we can calculate the adjusted nominal effort, which we call the development effort, by multiplying the values together in Equation (7.4).

$$\begin{aligned} \textit{Development Effort} &= \textit{Nominal Effort} \times \textit{Cost Multiplier} \\ \textit{Development Effort} &= 9.99 \text{ person-months} \times 1.339 \\ \textit{Development Effort} &= 13.38 \text{ person-months} \end{aligned} \tag{7.4}$$

Accounting for the development mode and the cost drivers, we have calculated a development effort of 13.38 person-months for the project development period. This can be broken down into 4.46 months per person. Here, we can then calculate the total cost of production. As we will be entering the workforce, we estimate we would earn around the average rate as an intern starting out. The average software engineering intern earns about $23 /hour [21]. Assuming a 40-hour work week, the monthly salary would be around $3,680, or a yearly salary of around $44,000. Now that we have a base rate, we can estimate the cost of development, as seen in Equation (7.5).

$$\begin{aligned} \textit{Total Cost} &= 13.38 \text{ person-months} \times \$3680/\text{month} \\ \textit{Total Cost} &= \$49,238.40 \end{aligned} \tag{7.5}$$

Calculating the cost of development using the Intermediate COCOMO, it was calculated that the total development effort was 13.38 person-months and will have a production cost of $49,238.40 as of April 2023. Now that development of the application is complete, we can compare it to the estimated values determined before development began in October 2022. The results can be

summarized in Table 7.3. As seen in the table, while the KLOC value increased by a large amount, the difference between total cost was minimal, this is due to the large change in cost multipliers. Upon completing the application, we felt that the cost drivers were inaccurate to what we developed, as such, we reviewed the drivers as a team and decided on new values for each driver. The decrease in difficulty was attributed to removing the machine learning portion of the application, as we felt it was unnecessary at this stage. Nevertheless, our estimations resulted in a similar value for both project cost and development effort, which shows the effectiveness of estimating project cost and development effort before any programming is performed.

Table 7.3: Summary of Cost Estimations: Estimated vs Actual

	Estimated	**Actual**	***Difference***
Total Cost	$47,766.40	$49,238.40	+$1472.00
Cost Multiplier	1.936	1.339	-0.597
Development Effort	12.98 person-months	13.38 person-months	-0.4 person-months
KLOC	2.050	2.927	+0.877

7.3.2 Profitability Perspective

Our approach to this problem is much more lightweight than existing solutions that require hardware such as proximity sensors or IP cameras. As such, it is a much cheaper implementation that requires less setup and can be used in a wider capacity at a much lower cost, which can be attractive to clientele. Various methods exist for marketing this software, whether it be software as a product (SaaP) or software as a service (SaaS). Marketing this using SaaP would entail a high upfront cost to obtain a product license, along with additional maintenance fees over time to ensure the continuous operation of the application. According to Software Equity [22], a good benchmark for a SaaP development firm is a gross margin of over 75%. Based on our calculated cost of project development using Intermediate COCOMO of $49,238.40, we would need to market the product at $86,167.20 in order to generate a 75% gross margin or profit of $36,928.80. These profit margins; however, are calculated assuming that the firm is well-known and established. For a SaaP startup firm, a profit margin of around 30% should be acceptable to establish credibility; as such, recalculating using the net profit margin of 30%, the product would need to be marketed at around $64,009.92, generating a net profit of around $14,771.52. This product is mainly reusable, requiring little configuration for reuse, and can be applied and marketed to many organizations with large parking areas. Given enough interest in the product, a framework could be designed based on our application to create a product designed with different features for different organizations, maintaining the same backend to increase revenue through licensing and distribution agreements. Another method of revenue generation would be to market this as a SaaS. This can be done through marketing this application with a yearly subscription, rather than a flat or large upfront cost, that would contain the rights to use the application, along with any additional maintenance required throughout the contracted period. This can be more enticing to organizations as there are no high upfront costs and may expand market opportunities, as only larger organizations can comfortably afford the full product cost upfront. Marketing this as a SaaS, any size organization would be able to negotiate their terms and comfortably afford the deployment of our application. In the case of organizations or businesses that charge for parking, such as Lakehead University, revenue

can be generated by receiving a small percentage of the earnings from parking passes purchased. This would be beneficial to both parties, as while the application is in use, it can be adjusted to require all users to have purchased and linked a viable parking pass to their accounts within the application. Given the unpredictability regarding the evolution of software, it is nigh impossible to predict the maintenance costs that would be incurred over time; however, it would still generate revenue overall.

Chapter 8

Project Timeline & Member Contribution

8.1 Project Timeline

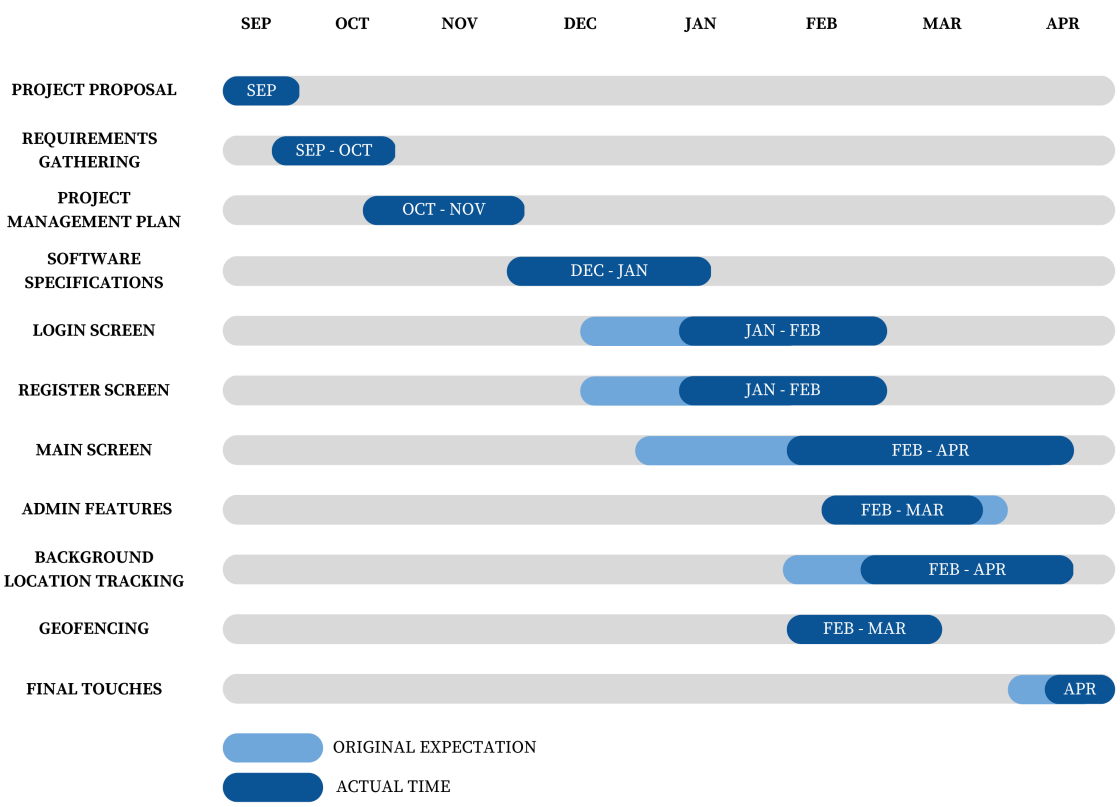


Figure 8.1: Gantt chart with the initial estimated timeline and our actual project progress.

Fig. 8.1 displays a Gantt chart that highlights the progress of the individual components throughout the project. The light blue lines were our original expectation for what and when items would have ideally been planned to be executed, and the dark blue lines are the actual

progress that was completed. Given the busy nature of the fall semester exams, initial progress on the development of the application was delayed until January, when progress started to begin. There were periods where development stagnated, these usually coincided with upcoming course project submissions, midterms, or other important events occurring. Compared to the original estimations, we did begin development later than expected, which may have resulted in not having additional time to implement extra features. Nonetheless, given these setbacks and conflicts, the project was still completed on time and on budget.

8.2 Member Contributions

Throughout development, the role of group leader was rotated. The duties of the group leader were to organize meetings in order to determine each member's progress, record the minutes of each meeting, and delegate tasks based on the importance of a given component towards the application. The list of leaders and their respective months are:

- Nicholas Imperius
  - September
  - March
  - April
- Jimmy Tsang
  - October
  - November
- Mason Tommasini
  - December
  - January
  - February

The breakdown of the minutes of each meeting can be found in Appendix C, here, the date, time, and key points regarding what was discussed in the meeting are listed, along with the group leader at the time.

In Table 8.1, a summary of each member's role, contribution percentage, number of lines of code written, and the estimated hours worked is provided. It is important to note that while team members were assigned roles, they were to assist other members in their full capacity when needed. Additionally, regarding the lines of code, this does not reflect the deleted contributed code or the contribution made to the Firestore Database, in which members populated the database with parking spaces. Furthermore, this number was computed from the GitHub repository by subtracting the total lines added from the total lines deleted. It can be considered a rough estimation of overall contribution, but won't reflect the true contributions of each member, as there were tasks that may not have had a lot of code attached to them. It is important to mention that the total number of lines of code will differ from the estimated KLOC used in Section 7.3, this is because the KLOC calculated does not account for whitespace or commented lines of code. There were additional contributions that cannot be accounted for in the code, these involved activities such

as updating the documentation, creating test cases, testing the application, and organizing the collective thoughts and ideas of the group. The contribution rate towards the overall development of the application, encompassing each aspect, is represented as the contribution ratio found in the table. This ratio was decided upon collectively as a team, accounting for every component faced during this development process. It can be said that estimated hours worked are not the best metric for measuring contribution, if a component designed by a group member did not work, countless hours could be spent debugging and testing the issue until the solution was found. Overall, through teamwork and perseverance, we were able to achieve the main objective of this project, to develop a functional and assistive smart parking application.

Table 8.1: Member Contribution Breakdown

Member	**Role**	**Contribution Ratio**	**Lines of Code**	**Hours Worked**
Nicholas	• Parking Detection Algorithm • Code Refactoring • User Interface	40%	2954	150
Jimmy	• Geofence Implementation • Location Tracking • User Interface	30%	1421	135
Mason	• Firestore Database Initialization • Database Integration • System Security and Encryption	30%	660	125

Chapter 9

Conclusion & Discussion

9.1 Conclusion

Throughout the design and development of *Link & Park*, our main objective was to provide an intuitive and useful application to aid students, teaching staff, and visitors with parking on Lakehead University's campus parking lot. By exploring new development methodologies and learning how to develop applications for a new domain, this objective was completed. *Link & Park* is able to successfully register your parking location on campus, locate your parked car quickly and easily, and even shorten the parking space search that we all face on a day-to-day basis. Based on our initial findings and testing, we consider this application a large success, and we expect those who choose to use *Link & Park* would agree. Android Studio is an extremely powerful IDE and made our development much easier than initially expected.

Throughout this project, we have gained a wealth of knowledge and skills that will be invaluable to us as software engineers. We have learned how to write clean, efficient, and well-structured code in Java, one of the most widely used programming languages in the world. We have also become familiar with XML, which is an essential markup language used for creating attractive and functional user interfaces in Android Studio. One of the most important lessons we have learned is the importance of clear and concise documentation. We have seen firsthand how vital it is to outline all the important aspects of a software development life-cycle, including planning, designing, testing, and deployment. Proper documentation ensures that all team members have a shared understanding of the project's goals, timeline, and progress, which helps to ensure that the project is completed on time and within budget. Finally, working with Android Studio, we have learned how to use its built-in tools and templates to create high-quality user interfaces, test our applications on virtual devices, and debug any issues that arise. Overall, this project has given us a strong foundation in software development that will serve us well in our future endeavours.

The design of an application is an intricate and meticulous process. Understanding the requirements and needs of the target domain of users was crucial to refining the key design components of the product. This process involved many steps, namely, clear tasks, developing an aesthetically pleasing and user-friendly application, including useful and thorough features, and more. Such is the case for the development of our application, *Link & Park*, and the intricate steps followed from the beginning to end of development.

The start of the process began with requirements gathering and deciding on the specification of the system. These were pivotal steps that are the backbone of the following steps, as they are all built on the information found here. System requirements were separated into two main categories, functional and non-functional requirements, that define how the system should operate

and the capabilities of the system, respectively. The next step of the process was defining the system architecture, or the process methodology, in how development is commenced. Given certain constraints regarding time and manpower, it was decided to implement the linear waterfall model, as with its clearly defined milestones, we were able to distinctly measure progress. The system design was the next phase of the process. In this stage, we asked many questions, such as: how will different components operate? How are they going to be designed? What should they look like? These were answered here, giving more clarity into the problem at hand.

Specifics regarding the implementation of the system and how it was simulated and experimented with were touched upon as well. The limitations behind the testing of the application were present here, as the problem being solved could not be simulated via an emulator; therefore, having to physically venture to the parking lot to test the application proved to be a big hurdle and consumed additional time which we did not have. Moreover, testing the application was crucial due to the number of different scenarios a user could be in when travelling through the parking lot. Exhaustive test cases and different testing methodologies were used to bring a high-quality product to market that can meet standards concerning functionality, usability, security, and more.

By following these best practices and paying attention to the latest trends in mobile application design, innovative and engaging apps can be created that meet the evolving needs of users and achieve a high standard of quality, meeting global certifications. Ultimately, a well-designed mobile application can offer significant benefits to both users and businesses, enhancing user experiences and driving growth and profitability. Designing such a smart-parking application can provide many societal and economic benefits to users. Additionally, it provides environmental benefits, in a society already struggling to maintain the quality of the environment. With a calculated cost of development of $49,238.40 using Intermediate COCOMO, it provides us with the freedom to market this software in different ways, either as a product or as a service. Both options have their benefits and drawbacks, but ultimately, it should be left to the product owners to come to an agreement with the clients over how it will be licensed.

9.2 Limitations & Future Work

Given the limited time frame to complete development, there are some additional components we wished to have completed given more time. Additionally, once development had concluded, there were aspects we felt could be expanded on and improved. Regarding the application's functionality, many adjustments will need to be made regarding the reliability, security, and performance to handle the large user base compared to the prototype developed. Differing algorithms may be introduced to replace the current implementation, given they provide more performance, such as replacing our current parking algorithm design with Google's activity recognition API which utilizes low-power signals from various sensors, i.e., accelerometer or gyroscope, within the smartphone to determine the type of activity that is being conducted. The API can determine if an individual is walking, standing still, driving, on foot or in a vehicle, or running. The API is unique in that a large collection of sensory data was collected, and a machine learning model was trained on this to provide a concrete result through the API. This could be used to complement our current implementation as it would increase the accuracy of determining when a user is driving, walking, or stopping by using GPS and sensors built within the phone itself.

As of now, we have implemented parking spaces for the entire R9 sublot at Lakehead University, with plans for an upcoming beta test launch, all the spaces found at Lakehead University's G lot will need to be added to the database for full functionalities. To accommodate users with Apple devices running iOS to allow them to use our app, the framework will need to be redeveloped for iOS

devices, as development principles differ between operating systems. To provide more hands-free operations to users, the application could be extended to work with Android Auto or Apple CarPlay, so users with newer generation vehicles can have the application displayed on their vehicle's larger, built-in, centre console, rather than on a smaller cellular device. This would be beneficial, as more users may be inclined to use it, given it removes more of the safety concerns associated with using a phone while driving. Lastly, for full functionality of the smart parking application, we hope to partner with Lakehead University Campus Security to integrate parking permits into the system, as this would allow the university to monitor and track who holds a valid permit and where the vehicle associated with the permit is parking at any given time.

Bibliography

[1] LakeheadU, "Overview of lakehead university," 2017. https://www.lakeheadu.ca/about/overview. [Accessed: 2023].

[2] A. Athira, S. Lekshmi, P. Vijayan, and B. Kurian, "Smart parking system based on optical character recognition," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1184–1188, 2019.

[3] S. Ma, H. Jiang, M. Han, J. Xie, and C. Li, "Research on automatic parking systems based on parking scene recognition," *IEEE Access*, vol. 5, pp. 21901–21917, 2017.

[4] K. Aftab, P. Kulkarni, I. Shergold, M. Jones, M. Dogramadzi, P. Carnelli, and M. Sooriyabandara, "Reducing parking space search time and environmental impacts: A technology driven smart parking case study," *IEEE Technology and Society Magazine*, vol. 39, no. 3, p. 62–75, 2020.

[5] C. C. How, I. Farhana Kamsin, N. K. Zainal, H. Aysa Abdul Halim Sithiq, and N. A. Abd Rahman, "Smart parking reservation mobile application," *2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, 2022.

[6] C. L. Torres, V. R. Torres, M. C. Estira, J. D. Osias, and J. M. De Los Reyes, "P.e.t.e.r. parking: Park easier through e-recommended parking mobile application," *2021 1st International Conference in Information and Computing Research (iCORE)*, 2021.

[7] A. Anand, A. Kumar, A. N. Rao, A. Ankesh, and A. Raj, "Smart parking system (s-park) – a novel application to provide real-time parking solution," *2020 Third International Conference on Multimedia Processing, Communication & Information Technology (MPCIT)*, 2020.

[8] B. M. Mahendra, S. Sonoli, N. Bhat, Raju, and T. Raghu, "Iot based sensor enabled smart car parking for advanced driver assistance system," *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2017.

[9] C.-K. Ng, S.-N. Cheong, E. Hajimohammadhosseinmemar, and W.-J. Yap, "Mobile outdoor parking space detection application," *2017 IEEE 8th Control and System Graduate Research Colloquium (ICSGRC)*, 2017.

[10] A. Z. Tahmidul Kabir, A. M. Mizan, P. K. Saha, M. S. Hasan, M. Pramanik, A. J. Ta-Sin, F. T. Johura, and A. M. Hossain, "An iot based intelligent parking system for the unutilized parking area with real-time monitoring using mobile and web application," *2021 International Conference on Intelligent Technologies (CONIT)*, 2021.

[11] J. Lewis, R. Abbas, S. Reisenfeld, and S. Maric, "Little locator: A smartphone parking-location application," *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, 2018.

[12] R. Mangiaracina, A. Tumino, G. Miragliotta, G. Salvadori, and A. Perego, "Smart parking management in a smart city: Costs and benefits," *2017 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, 2017.

[13] S. Rosenberg, "Smartphone ownership is growing rapidly around the world, but not always equally," Aug 2020. https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally/. [Accessed: 2023].

[14] ASQ, "What is the iso 9000 standards series?," 2023. https://asq.org/quality-resources/iso-9000#:~:text=ISO%209000%20is%20defined%20as,maintain%20an%20efficient%20quality%20system.. [Accessed: 2023].

[15] ISO, "Iso 9000:2015," Sep 2015. https://www.iso.org/standard/45481.html. [Accessed: 2023].

[16] Imperva, "What is nosql injection?: Mongodb attack examples: Imperva," 2020. https://www.imperva.com/learn/application-security/nosqlinjection/#:~:text=NoSQL%20injection%20occurs%20when%20a,unwanted%20command%20on%20the%20database. [Accessed: 2023].

[17] A. Anderson, "Road collision statistics in canada: A recent review," Feb 2020. https://www.netnewsledger.com/2020/02/13/road-collision-statistics-in-canada-a-recent-review/. [Accessed: 2023].

[18] G. Surpris, D. Liu, and D. Vincenzi, "How much can a smart parking system save you?," *Ergonomics in Design: The Quarterly of Human Factors Applications*, vol. 22, no. 4, p. 15–20, 2014.

[19] Fybr, "How far do we really drive while looking for parking?," 2015. https://www.fybr.com/how-far-do-we-really-drive-while-looking-for-parking/. [Accessed: 2023].

[20] S. R. Schach, *Chapter 9 - Planning and Estimation*, p. 270–282. McGraw-Hill, 8 ed., 2011.

[21] PayScale, "Average software engineering intern hourly pay in canada," 2023. https://www.payscale.com/research/CA/Job=Software_Engineering_Intern/Hourly_Rate. [Accessed: 2023].

[22] M. Soltesz, "How gross margins impact saas: Software equity group," Apr 2023. https://softwareequity.com/blog/gross-margin-saas/#:~:text=Based%20on%20our%20experience%2C%20a,dive%20into%20several%20other%20metrics. [Accessed: 2023].

Appendix A

Technical Documentation

For a more technical analysis, the source code and related documents can be viewed at the following GitHub repository: https://github.com/nickimps/Parking_Application

Furthermore, a video demonstration can be found at: https://youtu.be/cdRph8VEZDU

Appendix B

End User License Agreement

END USER LICENSE AGREEMENT (EULA) FOR SMART PARKING APPLICATION

PLEASE READ THIS END USER LICENSE AGREEMENT ("EULA") CAREFULLY BEFORE USING THE SMART PARKING APPLICATION: LINK & PARK.

1. GENERAL TERMS

This EULA is a legal agreement between you, as either an individual or a single entity, and [MNJ Software Solutions] (hereafter referred to as "Company"), the developer of the Link & Park (hereafter referred to as the "Application"). By using the Application, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not use the Application.

2. LICENSE GRANT

Subject to the terms and conditions of this EULA, the Company hereby grants you a limited, non-exclusive, non-transferable, revocable license to use the Application solely for the purpose of locating available spots on a university campus. This license is for personal and non-commercial use only.

3. USER DATA

The Application may use your phone's location to determine available parking spots on a university campus. By using the Application, you agree to the collection, transmission, processing, and storage of your location data by the Application. The Company will use your data solely for the purpose of providing the services of the Application and will not share your data with third parties without your consent, except as required by law.

4. PROPRIETARY RIGHTS

The Application is owned and operated by the Company. The Application and its entire contents, features, and functionality (including but not limited to all information, software, text, displays, images, video, and audio) are owned by the Company and are protected by Canadian and international copyright, trademark, patent, trade secret, and other intellectual property or proprietary rights laws. You acknowledge and agree that the Application and its contents are the property of the Company, and that you have no right to use the Application or its contents in any manner other than as expressly permitted under this EULA.

5. RESTRICTIONS

You may not copy, modify, distribute, sell, or transfer any part of the Application without the Company's prior written consent. You may not reverse engineer, decompile, or disassemble the Application, or attempt to derive the source code of the Application, except to the extent that such activity is expressly permitted by applicable law. You may not use the Application in any way that violates any applicable laws or regulations.

6. DISCLAIMER OF WARRANTIES

THE APPLICATION IS PROVIDED "AS IS" AND "AS AVAILABLE" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. COMPANY DOES NOT WARRANT THAT THE APPLICATION WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE APPLICATION WILL BE UNINTERRUPTED OR ERROR-FREE. YOU ASSUME ALL RESPONSIBILITY FOR THE SELECTION OF THE APPLICATION TO ACHIEVE YOUR INTENDED RESULTS, AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM THE APPLICATION.

7. LIMITATION OF LIABILITY

IN NO EVENT SHALL THE COMPANY BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES ARISING OUT OF OR RELATED TO THE USE OF THE APPLICATION, WHETHER BASED ON CONTRACT, TORT, NEGLIGENCE, STRICT LIABILITY, OR OTHERWISE, EVEN IF THE COMPANY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE COMPANY'S LIABILITY SHALL NOT EXCEED THE AMOUNT YOU PAID FOR THE APPLICATION, IF ANY.

8. TERMINATION

This EULA will remain effective until terminated by either party. You may terminate this EULA at any time by uninstalling the Application and destroying all copies of the Application in your possession. Company may also terminate this EULA at any time without notice to you if you breach any of its terms or conditions. Upon termination, you must immediately cease all use of the Application and destroy all copies of the Application in your possession.

9. GOVERNING LAW

This EULA shall be governed by and construed in accordance with the laws of the Province of Ontario, Canada, without regard to its conflicts of law principles. Any dispute arising out of or related to this EULA shall be subject to the exclusive jurisdiction of the courts of the Province of Ontario, Canada.

10. ENTIRE AGREEMENT

This EULA constitutes the entire agreement between you and Company regarding the use of the Application and supersedes all prior agreements and understandings, whether written or oral, regarding the subject of this EULA. Any waiver, modification, or amendment of any provision of this EULA will be effective only if in writing and signed by Company.

11. SEVERABILITY

If any provision of this EULA is found to be invalid or unenforceable, the remaining provisions shall remain in full force and effect. Any such invalid or unenforceable provision shall be replaced with a valid and enforceable provision that most closely reflects the original intent of the parties.

12. AMENDMENTS

Company reserves the right, at its sole discretion, to modify or replace this EULA at any time. If a revision is material, we will provide at least 30 days' notice prior to any new terms taking effect. What constitutes a material change will be determined at our sole discretion.

13. ASSIGNMENT

You may not assign or transfer this EULA, by operation of law or otherwise, without Company's prior written consent. Any attempt by you to assign or transfer this EULA, without such consent, will be null and of no effect. Company may assign or transfer this EULA, in whole or in part, to any third party without restriction.

14. SURVIVAL

The provisions of Sections 4, 5, 6, 7, 8, 9, 10, 11, 13, and 14 shall survive the termination of this

EULA.

15. CONTACT INFORMATION

If you have any questions about this EULA, please contact us at jtsang@lakeheadu.ca

By using the Smart Parking Application, you acknowledge that you have read this EULA, understand it, and agree to be bound by its terms and conditions. If you do not agree to the terms and conditions of this EULA, do not use the Application.

Appendix C

Meeting Minutes

Current Leader: Nicholas Imperius

- September 16, 2022: 8:54pm - 10:23pm
  - Begin Project Proposal
- September 20, 2022: 7:35pm - 9:54pm
  - Complete Project Proposal
- September 22, 2022: 7:38pm - 8:12pm
  - Practice Proposal
  - Start Software Requirements Document

Current Leader: Jimmy Tsang

- October 1, 2022: 8:52pm - 10:09pm
  - Work on Software Requirements Document
- October 7, 2022: 9:39pm - 10:45pm
  - Work on Software Requirements Document
- October 12, 2022: 8:14pm - 9:25pm
  - Complete first draft of Software Requirements Document
- October 26, 2022: 8:14pm - 10:01pm
  - Complete second draft of Software Requirements Document
  - Begin Software Project Management Plan
- October 31, 2022: 8:22pm - 10:05pm
  - Complete Software Project Management Plan
  - Apply any changes as directed by Dr. Akilan

- November 3, 2022: 8:12pm - 9:27pm
  - Adjust Software Project Management Plan
- November 8, 2022: 8:04pm - 10:13pm
  - Start Project Presentation
- November 10, 2022: 9:35pm - 10:09pm
  - Complete and Practice Project Presentation
- November 18, 2022: 8:11pm - 10:21pm
  - Touch up Software Project Management Plan
  - Perform research into the development

Current Leader: Mason Tommasini

- December 22, 2022: 8:30pm - 9:30pm
  - Discussion of project works over the Christmas break
- January 3, 2023: 8:30pm - 9:30pm
  - Creation of the software specifications document
  - Updates on the GUI and database
- January 9, 2023: 8:04pm - 10:10pm
  - Continue working on software specifications document
  - Try to combine database with GUI
- January 11, 2023: 8:00pm - 10:00pm
  - Fine tuning of GUI and database components
  - Create different diagrams and images for specifications document
- January 13, 2023: 9:00pm - 11:00pm
  - Optimizing code of GUI/database
  - Implementation of settings page into GUI
- January 14, 2023: 6:00pm - 8:00pm
  - Adding final touches to GUI/database before submission of specifications
  - Adding final touches/review of the specifications document
- January 16, 2023: 8:30pm - 9:00pm
  - Get together to review specifications document before submitting
- January 17, 2023: 9:40pm - 10:10pm

  - Small talk about what we need to add next

- January 22, 2023: 7:30pm - 9:40pm
  - Figure out the implementation of the geofencing
  - Talk about what we else we would use the geofence for

- February 10, 2023: 6:00pm - 7:00pm
  - Sign up for Rita Nicholas poster board presentation

- February 10, 2023: 6:00pm - 7:30pm
  - Going to Lakehead university to see how accurate the tracking of a person is to see if they line up with a parking spot

- February 19, 2023: 4:20pm - 7:00pm
  - Finish Rita Nicholas poster board presentation

- February 20, 2023: 9:00pm - 10:00pm
  - Implement changes to Rita Nicholas poster board presentation from Dr. Akilan

- February 21, 2023: 11:00am - 1:00pm
  - Attend and present Rita at the Nicholas poster board presentation

Current Leader: Nicholas Imperius

- March 14, 2023: 7:15pm - 8:45pm
  - Looked into adding some additional parking spaces to our existing small subsection

- March 21, 2023: 8:30pm - 10:15pm
  - Discussed how we would go about gathering testing data
  - Created a service in the admin activity to provide a tracking interface so that we can log information

- March 27, 2023: 9:45pm - 11:00pm
  - Started implementation of security checks
  - Discussed what permissions we should be looking out for

- April 4, 2023: 7:00pm - 11:10pm
  - Started the oral presentation
  - Worked on the draft slideshow

- April 5, 2023: 9:11pm - 11:30pm
  - Finished the slideshows based on comments received

- April 10, 2023: 8:30pm - 10:00pm
  - Started progress on the design document
  - Talked about what sections we need to include etc.
- April 11, 2023: 8:00pm - 9:15pm
  - Worked on the design document
- April 15, 2023: 8:00pm - 12:00am
  - Worked on final edits for the draft submission
  - Coordinate anything that needs to be done for the code submission