Department of Software Engineering
Lakehead University

ESOF 4969 – DEGREE PROJECT
2022-23

Progress Report 3 - Software Specifications (Interim)

Group 3

Contributing Members:
Nicholas Imperius
Jimmy Tsang
Mason Tommasini

Date Submitted: 01/16/23

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, Functional Req., System Architecture (created all figures), Technical Documentation, Conclusion
Jimmy Tsang: Non-Functional Req., Database Design, Algorithm Design, User Interface Design, Interim Results, Conclusion
Mason Tommasini: Abstract, Index Terms, Algorithm Design, Pseudocode, Conclusion

Smart Parking Application: Software Specifications (Interim)

Nicholas Imperius Jimmy Tsang Mason Tommasini

Abstract

This paper describes the design of a software application to help users find a parking spot in a city. The application will use machine learning techniques to determine whether you have parked your car or not, and then use GPS location information to notify other users that the parking space has been filled. This application would provide further benefits, such as saving the user time, reducing unnecessary stress, and helping with the environmental impact of vehicles. A brief overview of the project's three components machine learning, graphical user interface (GUI), and database functionalities will be displayed to give a guideline for further development of the project. The parking application will be broken apart into various types of diagrams to further understand how each component works with one another. The time and storage complexity along with algorithm design are further detailed and expanded on. We will discuss the current progress of the project as well as detail the current state of our project progress in relation to the described schedule from previous documents.

Index Terms— Parking application, machine learning, mobile android application, database model

Contents

1 Introduction

Many students and staff can attest to problematic parking at Lakehead University. To provide a simple, fast and effective solution to this problem, we are creating a software application that makes finding a parking spot much easier. The software is implemented as a mobile application for smartphones that displays information such as free parking spaces, busy times, and the location of your parked car. When arriving at the university, it can be difficult to find an open parking spot and frustrating when you are running late. This application would provide a solution to this issue and provide additional benefits, such as saving the user time, reducing unnecessary stress, and helping to reduce the environmental impact of carbon emissions produced by motorized vehicles. Our initial testing and training of this project will be focused on the G campus parking lot at Lakehead University, but could be further expanded to other fields, such as public parking structures and mall parking lots.

This application will be technologically quite different from other parking applications that exist currently. Instead of relying on sensory data or IP cameras to locate and detect empty parking locations, this application will use machine learning techniques to determine whether you have parked your car or not and then use GPS location and network information to notify other users that the parking space has been filled. Additionally, when you have left your parking spot, our model will be able to accurately detect that and mark the parking space as free. Different machine learning classification methods will need to be tested to ensure that we have chosen the most appropriate model. The machine learning model will predict based on your speed before and after a full stop; it should be able to recognize the difference between walking and driving, which allows it to determine the stopping location.

A 3-tier architecture, shown in Fig. 1 is an appropriate style for this parking application. The application relies on two cloud services, namely the Google Maps API and Firebase Cloud 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 can hold user information as well as parking related information is crucial too; therefore, we have our cloud storage handled through Cloud Firestore using the querying language NoSQL.

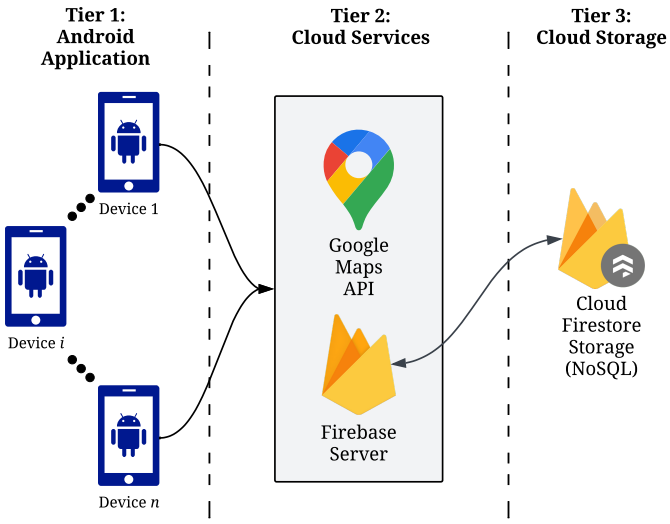


Figure 1: A layout of the 3-tier architecture that the application has been designed with where the number of active devices are from $1, 2, ..., n$, where i is a number between 1 and n.

The architecture and design of each functionality within the parking application allows for a greater understanding of how each component within operates and communicates with each other. Furthermore, from understanding how components communicate within a system, this allows for more optimized methods of handling function calls and relaying information from APIs and more. The majority of our application relies on information that is collected through various APIs in order to meet the requirements from the requirements document.

This paper aims to detail the specifications of the parking application that is being built. It is outlined as follows: Section 1 provides an initial introduction, Section 2 details the current functional and non-functional requirements of the parking application, Section 3 describes the system in terms of understandable diagrams between system components and more, Section 4 provides an option for a further and deeper look into the source code of the parking application, and Section 5 summarizes a conclusion to the document and our interim results.

2 Functional and Non-Functional Requirements

2.1 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, and started walking, the machine learning model should be able to determine this and mark the location as not available.
- When a user has gotten in their vehicle and drove 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.

2.2 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 efficiency of the program. Such non-functional requirements needed include:

- Speed: The application must update the user's position at a speed that mimics 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, being 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.
- Compatibility: The application must be able to function on the widest range of mobile devices as possible.

3 System Architecture

3.1 Class and Interaction Diagrams

The deployment diagram shown in Fig. 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.

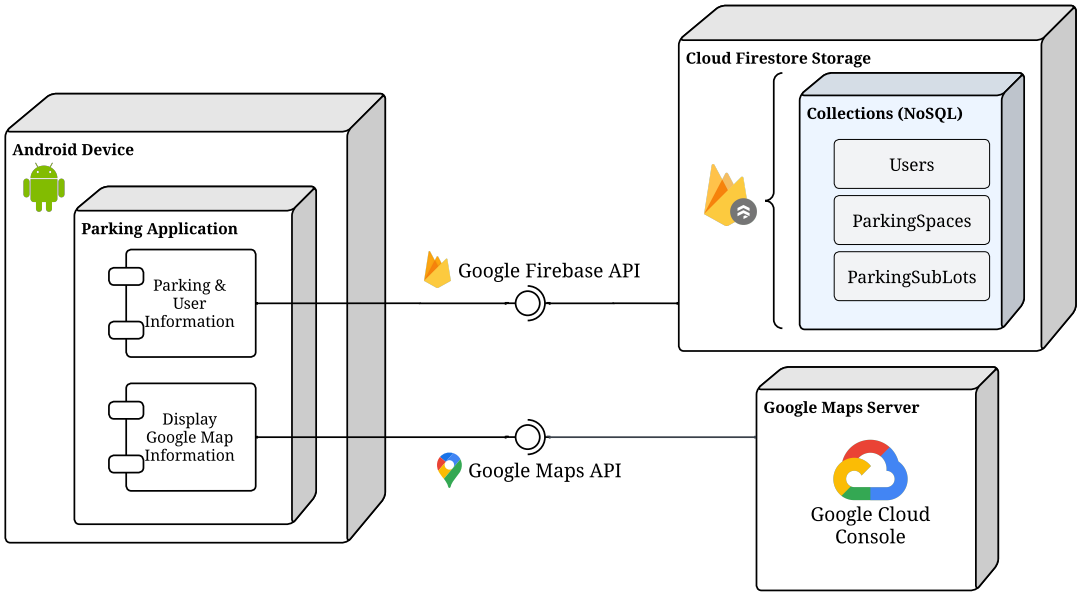


Figure 2: Deployment diagram that details how the android application will communicate with external systems.

The activity diagram in Fig. 3 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 gathering 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, which 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 simply 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 users information which is stored locally on the device and therefore bypasses the login and register screen.

The class structure for this parking application is simple in design. The login activity has associations with the register and maps activity, while the info activity is associated with the map's activity, as shown in Fig. 4.

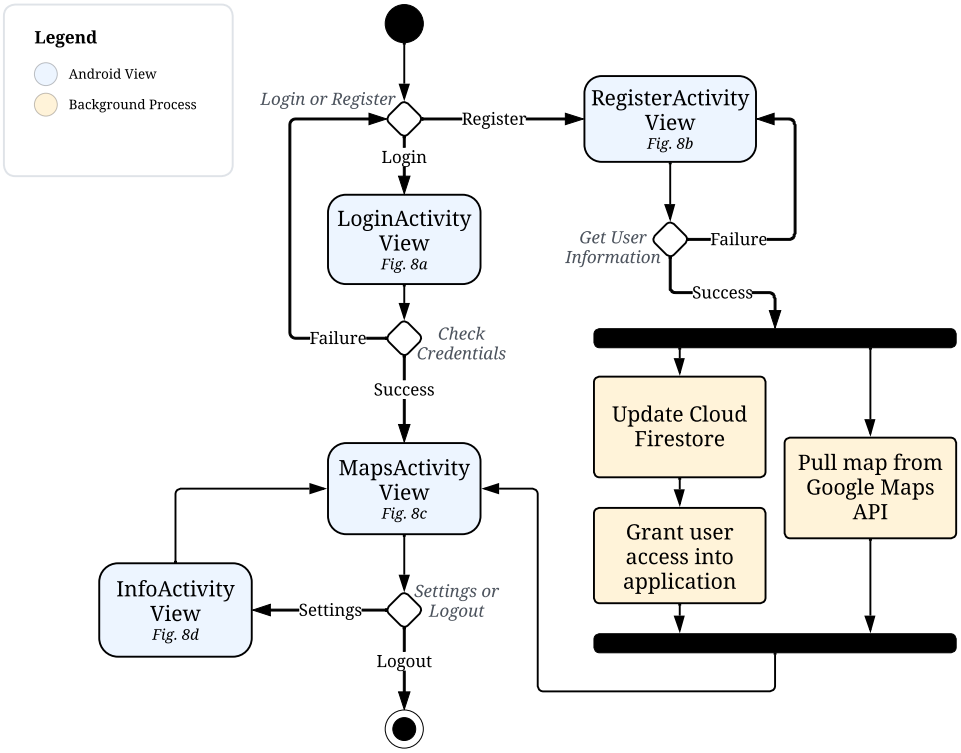


Figure 3: An activity diagram that shows the general flow and directions a user can take when using the application.

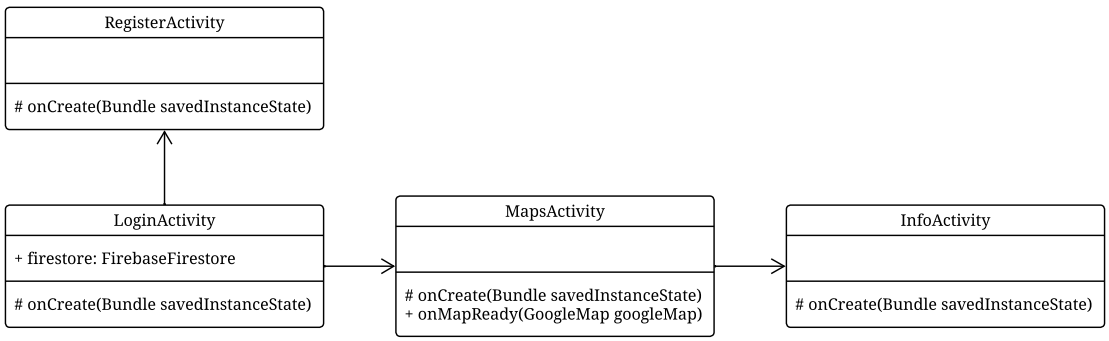


Figure 4: UML Class Diagram for the overall functionality and associations within the parking application.

3.2 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. 5 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.

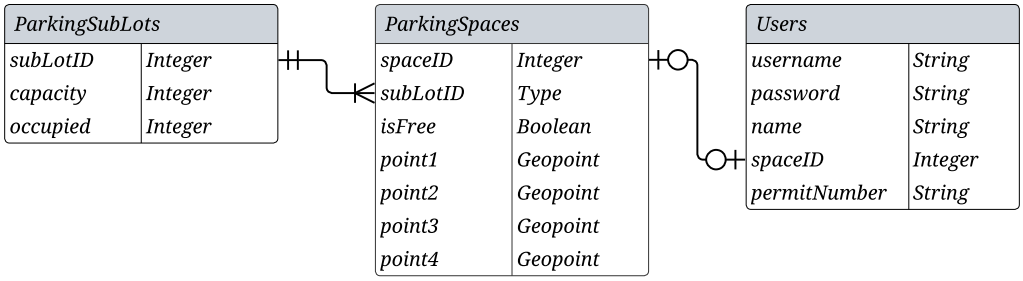


Figure 5: Database entity relationship diagram of the NoSQL database. The three collections each have document entries with the specified fields of information.

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.

3.3 Algorithm Design

Algorithmically, this project is quite straightforward. A machine learning model will perform the core calculation of determining if the user has parked their vehicle or not within the Lakehead University campus parking lot. Data will be collected in such a way that the speed and location of the user is monitored to ensure that the model can accurately detect when a user has parked inside a valid parking spot on Lakehead University.

Big O notation is a method for determining the complexity of an algorithm depending on how intricate or the size of the input. With our current implementation, we have determined the complexity to be of level $O(n)$, however, the location tracking has not been fully implemented. As development progresses, we expect the complexity to increase as there are more calculations needed to be performed in real time in the final implementation of the designed application.

Fig. 6 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. 7. The system checks to see if the text fields have information within them and update

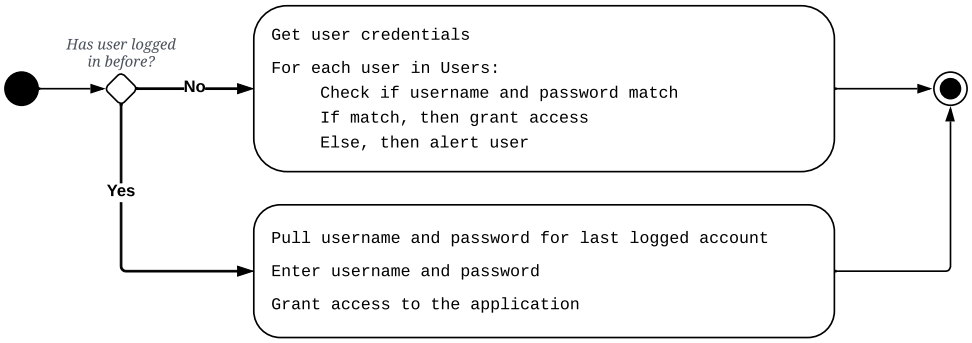


Figure 6: The process of a user opening the application on their android device. This determines if they have logged in before and if they should be auto logged in, or if a username and password need to be collected from the user.

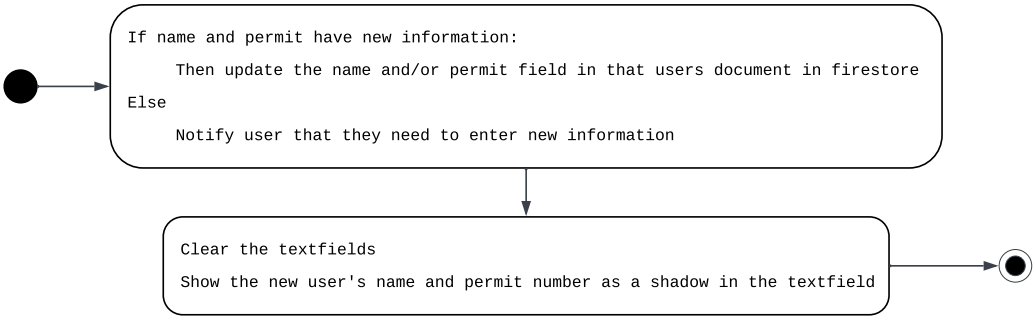


Figure 7: The internal process when the update button is clicked on our settings screen.

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

3.4 User Interface Design

For the graphical user interface, we construct a mobile application that allows users to sign in to the application and view available parking spots located on Lakehead University. This mobile application is directly connected to a Google Firebase NoSQL Database, which will communicate with each other in real time. In order for the mobile application to function as required, users must allow the application to access information pertaining to the device location and have a stable internet connection, whether through a Wi-Fi connection or cellular data. This allows the application to be able to accurately track their location and update information to the database in real time.

The application will have a simplistic view for ease of use, consisting of four main pages. The first page will consist of a login screen, here, users will sign in with their credentials, consisting of a username and a password, as depicted in Fig. 8a. If the user does not yet have an account, they can navigate to the registration page shown in Fig. 8b. to create an account. On the registration page, the user will need to provide a username, password, and first name to generate an account, with the option of including a permit number.

Once the user successfully signs in to their account, they are then navigated to the view of the parking lot, which would depict the parking spaces on the Lakehead university campus. Fig. 8c. represents this screen in which users will be able to see the status of each parking spot, with green spaces representing open locations, red spaces representing unavailable locations, and a blue space representing the location of the user's vehicle, once parked. Users can navigate the map freely by zooming in or out on the map, along with updating their account information through the click of a button, which would bring users to a new page. When clicking the button to update account information, users will be led to a different page which contains information regarding their name and permit number, shown in Fig. 8d. New information can be entered into the text fields and updated through the click of the *update* button, which would update the information, at which point users can navigate to the previous page with the *back* button. Lastly, users can log out of their account through the *logout* button, taking them to the login screen.

3.5 Interim Results and Pseudocode

3.5.1 Interim Results

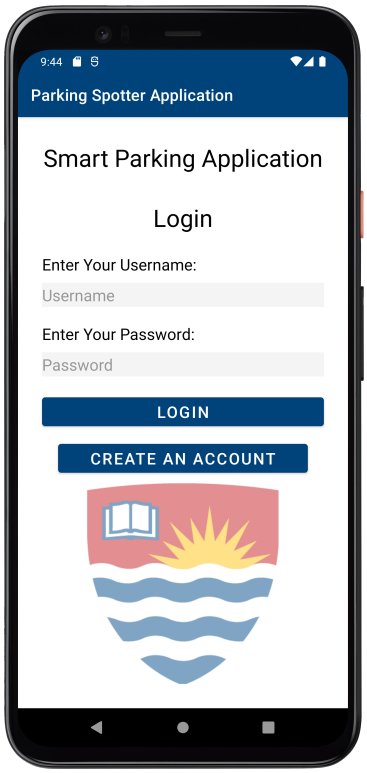
As of January 16, 2023, the login, account creation, and update information functionalities are fully integrated. However, at this time, the location tracking and parking detection features have been incorporated but not fully integrated.

In the application, users are able to successfully log into their accounts provided they entered the correct information. If a user does not yet have an account, they can easily create one. Once they have signed in to their account, their information is stored locally within the app on the device, that means, if a user exits the application without signing out, they will be automatically signed in the next time the application is open. Users may also update their information once signed in, the username and parking permit can be updated at any time, with full real time communication through a Firebase Cloud Database using NoSQL queries. Additionally, if the user enters any incorrect information or are missing required fields, there are pop up notifications that notify the user what they are missing.

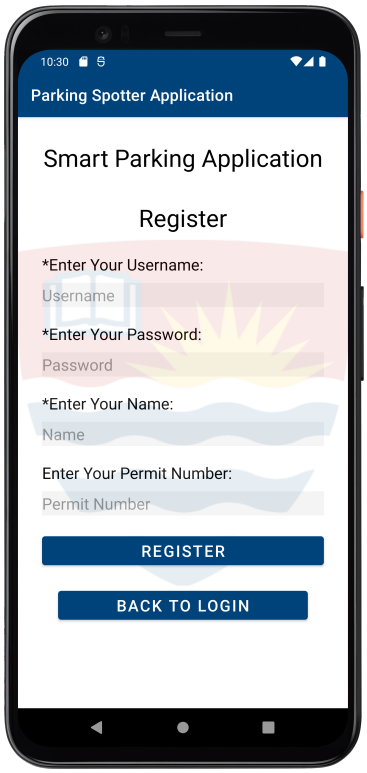
Integration of the live location tracking as well as monitoring of speed components will be integrated at a later time to determine whether a user has parked within a parking spot on campus.

3.5.2 Pseudocode

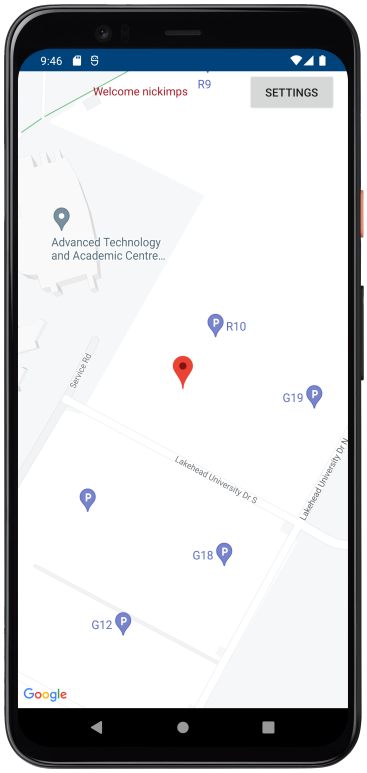
The pseudocode shown below is a representation of how the three main components being the GUI, database and machine learning will interact with each other. Based on Fig. 3, we can see a more intuitive display of how the three components interact. The pseudocode, shown below, can be seen as a scenario of all the possible occurrences and outcomes that come with interacting with the application via GUI. The pseudocode also goes more in depth, showing the back-end processes of what goes on behind each interaction of the application. The four main functions in the pseudocode are



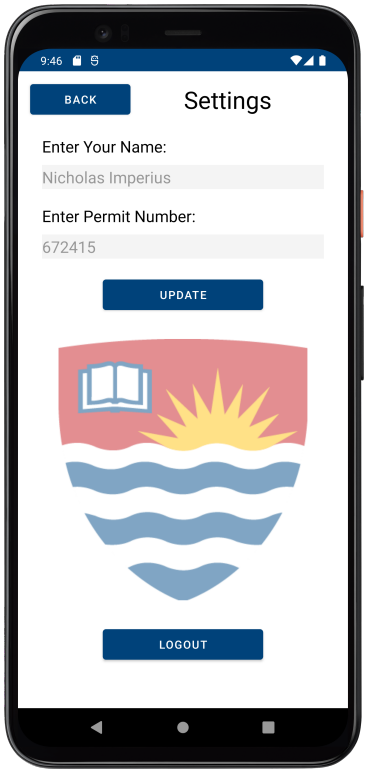
(a) LoginActivity



(b) RegisterActivity



(c) MapsActivity



(d) InfoActivity

Figure 8: User Interface Layout. (a) depicts the login screen in which users can sign in using their existing credentials, or navigate to (b), where users can create an account, which will be stored in the database. Once users are successfully logged in, they are navigated to (c), in which the map which contains the parking spots found on Lakehead University campus. Users can then go to (d) to update account information, or navigate back to the map view.

the *applicationStartUp*, *RegisterActivity*, *LoginActivity*, *MapsActivity*, and *InfoActivity* as these activities are the main components of the GUI that interact with the backend server/database and machine learning algorithm. Although the functionality of the machine learning algorithm is not yet implemented, the pseudocode can still represent the placement and functionality it provides.

```
applicationStartUp(){
    if(no previous user login)
        RegisterActivity() or LoginActivity()
    else
        goto(MapsActivity)
}

RegisterActivity(register button press){
    if(all fields filled & username is unique)
        Update cloud firestore
        Grant user access into application
        Pull map from Google Maps API
        goto(MapsActivity)
    else
        Display("register failed")
}

LoginActivity(login button pressed){
    if(all fields filled & username and password match)
        Update current login details
        goto(MapsActivity)
    else
        Display("login failed")
}

MapsActivity(){
    view(parking spots) or goto(InfoActivity)
}

InfoActivity(settings button press){
    if(logout button press)
        Logout user
        Update current login credentials;
    else if(update button press)
        if(name or permit field filled)
            Update database
    else if(back button press)
        goto(MapsActivity)
}
```

4 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

5 Conclusion

Changes and improvements will be made to the current implementation of the GUI and database functionality, as well as starting the development and integration of the machine learning model component that will detect whether a user is occupying a parking or not, which is the main functional component of the project. Overall, the current state of the project is following the expected timeline, workload, and estimated effort needed to be completed; therefore, we expect to finish the application and complete final documentation on schedule. Upon completion, the cost and duration estimates will be recalculated and compared to our initial findings

Our prototyped user interface was detailed with the key activities, or views, to show the current state of the application. As time goes on, the indicators for parking spaces as well as the visibility of the occupied locations will be implemented soon. As of now, the supporting functionality exists and navigating within the application has flawless execution. That is, auto-login of users, database updates and queries are successful, saving of local user account information is functional, and more.

This stage of the project provides a rough prototype of the current parking application's capabilities. This document aims to outline how the individual processes connect and communicate with one another based on key functional and non-functional requirements. These requirements are important in providing a guideline as to what is required to be designed and engineered into the system. Through the visual understanding of specific diagrams relating to class structure, database design, and more, the parking application can be designed more rigorously and operate more efficiently than without.

Acknowledgement

We would like to express gratitude towards the department of Software Engineering at Lakehead University for this opportunity and Dr. Thangarajah Akilan, Chair of the Software Engineering Department, for being our project supervisor.