CENG 319: Software Project – **Final Report**

**WATECH PARK**

**Name:** Vikas Sharma

**Student ID:** N01160135

**Course Code:** CENG 319

**Instructor:** Austin Tian

**Section:** 0ND

**Date:** December 13, 2019

**Abstract:**

In retrospect, this report will outline the development and final testing of the software portion of the semester which entails the final Capstone Project in Winter 2020. Watech Park is a parking lot management APP that allows consumers/businesses to be able to park, pay and reserve parking spaces and retrieve real-time data through use of an online server. Along with, making use of real hardware sensors to address increasing capacity levels, and reducing competition between drivers/parking lot owners in finding the best spot to park. This report will address the key fundamentals of the project, including the main purpose, introduction to the project, background, the software design/specification and the connection between the main data structures. The topics covered in this document will address how the software component, being the mobile APP interacts with the hardware components developed from CENG 317. The project covers the general foundation of an IoT (Internet of Things) format, where software and hardware interaction would be vital to address an industry related issue and help solve real-world problems. I will be using different use-case scenarios to touch on these details, and provide a guideline of the authentication/database design taken. This includes, the Android resources used to support and successfully complete the software portion of the project, in preparation for the capstone project integration. A table of contents has been provided for your convenience, please use if required.

**Table of Contents**

Abstract……………………………………………………………………………... iii

Introduction to Project…………………………………………………………………………….1

Background ……………………………………………………………………………………….1

Schedule…………………………………………………………………………………………...2

Software Design/Specification………………………………………………….............................2 Proposal……………………………………………………………………………………2

Software Requirements Specification……………………………………………………. 6

Mockups ………………………………………………………..…..................................27

UI Design………………………………………………………………………………………...30

UML Class Design ……………………………………………………........................................39

Data Structure Design……………………………………………………………………………40

Part 1: Authentication……………………………………………………………………40

Part 2: Data………………………………………………………………………………41

Hardware Implementation……………………………………………………………………….43

Testing Cases/Trouble-Shooting…………………………………………………………………43

Android Components Used………………………………………………………………………44

Individual/Team Contribution…………………………………………………………………...44

Concluding Remarks……………………………………………………………………………..44

GitHub Project Address………………………………………………………………………….44

References………………………………………………………………………………………..44

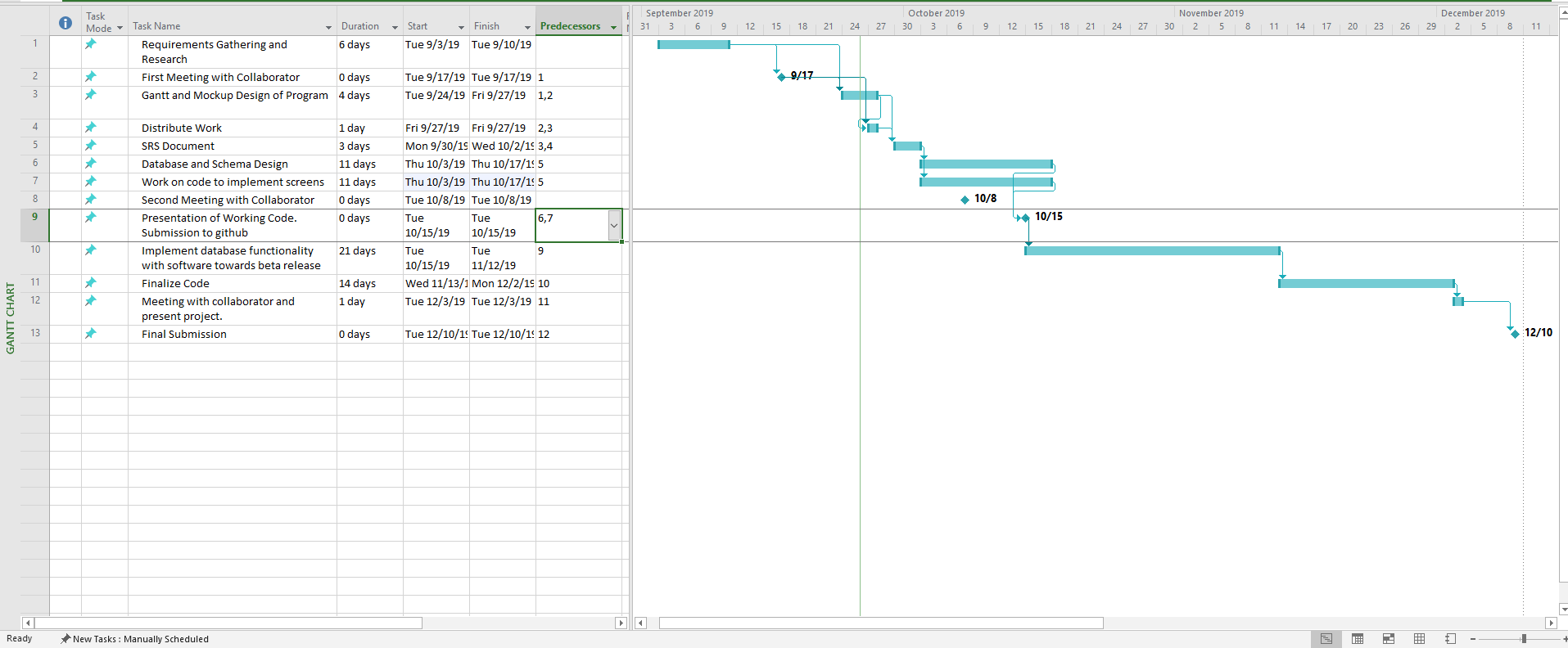
**Introduction to Project:**

Watech Park is a SMART parking lot management system centered around the idea of providing users with a simpler, intuitive method to access parking locations. This is done by addressing industrial issues such as major congestion in traffic cities, and reduce time spent for manual searching. The goal, to manage capacity levels, and in-return increase productivity. My plan to reach this guideline, was to provide options of consumer + business use. The consumer side was the main focus for this semester, with the business component planned to be executed in the next following semester. The assigned task was to build a mobile application in an IoT fashion. Taking this approach, the idea was to server consumers with a more efficient, and reliable method to retrieve data in real-time. Therefore, using this mobile APP will serve as a more convenient method. The main purpose being to connect consumers to parking lot owners, and provide real-time statistics based on the capacity levels of a parking lot. This includes, providing real-time proximity measurements, lot detection methods with use of the hardware elements. Such as, the entry/exits of vehicles, detecting parking space movement, gate control. Our focus was to develop a mobile APP that would be the gateway to support consumers with finding the best parking space during any time, any place or anywhere in the world.

**Background:**

In the industry today, there have many occurrences where parking in general has become a hassle for city residents and parking lot owners. This includes, not possessing the right tools to manage capacity when a parking lot is full, where drivers are struggling to find the best spot to park their vehicles. This can lead to dis-satisfying scenarios, where drivers are unaware of their surroundings, before even entering into the space. Due to this reason, it can lead to congestion in major traffic centric cities, with drivers competing to find a spot. This can be time-consuming, inefficient where productivity is lost for consumers and businesses. This project is focused on helping reduce the impact of this cause, by developing a system that will address payment for parking by taking an advanced and modern approach, capacity management, and real-time information gathering to keep consumers up to date with their daily occurrences.

**Schedule:**

****

The initial development phase of the software portion of the project, consisted of a timeframe of 14 weeks. This included completing the requirements gathering, preparing for each milestone, prototyping, coding and testing the mobile APP to ensure successful connection to the chosen database. For this reason, I followed an Agile methodology for programming, and coding. Taking this approach, development was separated into phases of smaller and distinct sprints where part of the design was integrated each sprint, in blocks of development periods. For the most part, development was active and changes were applied as the semester went by.

**Software Design/Specification:**

**Proposal:**

**Iot Capstone Software Development Proposal**

Project Name: Watech Park

Team Members: Vikas Sharma, George Alexandris (CENG 317 Only)

Project Collaborator: Mike Wrona, Installation Manager.

Contact(email): mike@parkingboxx.com

**Overview:**

This document will outline the software portion of the project in CENG 319 that will be coupled with hardware in CENG 317 for the final integration in CENG 355. Our project is going to be on a smart parking lot system.

**Background:**

Many busy parking lots are often plagued with congestion, drivers competing to find a spot by cruising around and visually finding spots. This is inefficient, time consuming where productivity is lost for consumers and businesses. The system we will be developing will address payment for parking, capacity management and location finding.

Examples of some existing platforms are Indigo and BestParking. Both of these platforms have reservation capabilities, on the go parking with mobile or web application. What we are going to do differently from these companies is to integrate sensors and internal parking maps to help users navigate to a parking spot.

**Knowledge and Skills (Software Component):**

* Java application development from CENG 212 from Programming Techniques in Java.
* Agile programming methodologies and project management from CENG 216 Intro to Software Engineering.
* SQL and database modelling from CENG 254 Database with Java.
* Building a server-side application to connect a client with a data source from CENG 256 Internet Scripting.

**Development Environment:**

Android Studio – Java

Database – Firebase Cloud Datastore

**Hardware:**

* IR Beam Sensor for gate control to detect if car is present.
* 1 Camera at gate to get license plate. (LSR Camera Sensor)
* VNCL4010 Proximity Sensor.
* Nth amount of Tri Color Led’s for parking indicators.
* 2 stepper motor for the gates.

**Rough App Development:**

* Login & registration screen -> login as Business or Consumer.
* Account Management
* Parking Passes
* Order History
* Add A Car
* Manage Cars
* Database schemas for different functions.

**End User:**

* Parking lot reservation.
* Monitor and manage parking passes
* View Details (sensor data – real-time proximity/status of lot)
* Payment
* Settings
* Help documentation.
* About

**Business:**

* Parking lot overview of parked users.
* Statistics of parked users.
* Ticket and dispute management.

**Test Cases:**

- Test Register/Login is successful with a database. (Authentication)

- Test parking status, does it provide real time data matched in database?

- Test payment -> Deduct amount -> update parking pass screen.

- Test proximity data rea-time updates

- Test Add A Car option/ View Added Cars (in Manage Cars)

- Test Order History, does it display the correct Order(transaction) details

- Test Firebase pushing/retrieving of data

- Test Settings, localization integration(English/French)

- Test account management screen, can registration information be retrieved and displayed

**Time Estimation:**

14 weeks for developing the software portion of the project. This includes requirements gathering, prototyping, development and testing. The project will be done using an agile methodology for programming. This will be done in the development phase.

If development stalls or there are significant delays, the team will review immediately. The team take appropriate action whether to change development plan to accommodate or remove the problem causing that delay.

**Conclusion:**

This proposal presents an initial outline for the software portion of the IoT solution for our project Park Smart. This project will utilize the skills learned from previous course in a team setting and will demonstrate knowledge in a practical manner. We request approval for initiating the project.

**References:**

Park Indigo Canada Inc. (2019). Indigo. Retrieved from https://ca.parkindigo.com/en

ParkWhiz. (2019). Find and Book Parking Anywhere. Retrieved from https://www.bestparking.com/

**SRS (Software Requirements Specification):**

Software Requirements Specification

for

Watech Park

Smart Parking Lot System

Version 1.0 approved

Prepared by

Vikas Sharma

Humber College

Institute of Advanced Learning

December 6, 2019

Table of Contents

[1. Introduction 1](#_Toc21120875)

[1.1 Purpose 1](#_Toc21120876)

[1.2 Product Scope 1](#_Toc21120877)

[1.3 Intended Audience and Reading Suggestions 2](#_Toc21120878)

[1.4 References 3](#_Toc21120879)

[2. Overall Description 3](#_Toc21120880)

[2.1 Product Perspective 3](#_Toc21120881)

[2.2 Product Functions 3](#_Toc21120882)

[2.3 User Classes and Characteristics 3](#_Toc21120883)

[2.4 Operating Environment 4](#_Toc21120884)

[2.5 Design and Implementation Constraints 4](#_Toc21120885)

[2.6 User Documentation 4](#_Toc21120886)

[2.7 Assumptions and Dependencies 4](#_Toc21120887)

[3. External Interface Requirements 5](#_Toc21120888)

[3.1 User Interfaces 5](#_Toc21120889)

[3.2 Hardware Interfaces 5](#_Toc21120890)

[3.3 Communications Interfaces 6](#_Toc21120891)

[4. System Features 6](#_Toc21120892)

[4.1 Use Case Representation 6](#_Toc21120893)

[4.1.1 Use Case View 6](#_Toc21120894)

[4.2 Parking Lot Proximity Detection – “TBD” 8](#_Toc21120895)

[4.3 Payment/Transaction 8](#_Toc21120896)

[4.4 Registration 9](#_Toc21120897)

[4.5 Login 10](#_Toc21120898)

[4.6 Update Orders 11](#_Toc21120899)

[4.7 Manage Cars 11](#_Toc21120900)

[4.8 Sign - Out 12](#_Toc21120901)

[4.9 Settings 12](#_Toc21120902)

[5. Other Nonfunctional Requirements 12](#_Toc21120903)

[5.1 Performance Requirements 12](#_Toc21120904)

[5.2 Safety Requirements 14](#_Toc21120905)

[5.3 Security Requirements 14](#_Toc21120906)

[5.4 Software Quality Attributes 15](#_Toc21120907)

[5.5 Business Rules 16](#_Toc21120908)

[6. Other Requirements 16](#_Toc21120909)

[6.1 Internationalization/Localization Requirements 16](#_Toc21120910)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Initial Draft | 10/4/2019 |  | 0.1a |
| Final Draft | 12/6/2019 |  | 1.0 |

# Introduction

## Purpose

To develop a mobile application platform for a smart parking lot system. The purpose of this system is to automate the processes of parking such as: payment, entry, exit, passes, and parking lot management in an internet of things fashion.

The first version of the document will be describing both hardware and software functionality of the platform, related processes as drafted in earlier documents.

Content of this document is subject to change as development progresses. Version numbers of this document will change as follows:

**0.# a (Alpha)**

Alpha: Product specifications based on proposal, initial mockups and incomplete code.

**1.# b (Beta)**

Beta: Procut specifications developed and implemented in working code. Processes, features, interfaces and environment updated to reflect current build.

**2.# rc (Release Candidate)**

Release Candidate: Product specifications based on complete integration of all components including hardware.

**3.#**

Final: Release version of working platform.

## Product Scope

Watech park is a parking lot management platform that connects consumers to parking lot owners. It consists of a mobile application for both the consumer and businesses. The consumer can pay for parking beforehand with their car and license plate information. When the consumer drives up to the parking lot, there is no physical contact or buttons needed. The cameras installed at the lot will allow the consumer in and track when the consumer drives out.

The on-site hardware will relay and manage the parking data and monitor how long each consumer has been in a lot. It will provide statistics on how many cars are parked in the lot, if there are any consumers over a time limit and revenue.

The benefits to businesses would be reducing the need to have staff do checks for valid consumers in the parking lot. This would reduce staffing costs and increase efficiency for handling ticketing, removing vehicles and having more paid utilization of parking space.

## Intended Audience and Reading Suggestions

This document is intended for a wide variety of readers including users, businesses, developers, and marketing staff marketing.

Section 1 Introduction

* This section describes the purpose of the document, the product and recommendations on what to read.

Suggested Readers: All

Section 2 Overall Description

* This section describes the Watech Park in a high-level overview that summarizes how the platform works in less technical detail. The content of the section would be a good starting point for marketing staff to get the key features to advertise the platform. All users will benefit from reading this section to understand how the platform works.

Suggested Readers: All

Section 3 External Interface Requirements

* This section provides details on what types of interfaces are required for the platform from a software and hardware perspective. The content of this section would be good for developers, and businesses to understand some of the hardware requirements, what is involved with the platform.

Suggested Readers: Businesses, Developers

Section 4 System Features

* This section provides more low-level detail on required functions, data for a working platform. The content of this section is best suited for users with a technical background and less for general business cases.

Suggested Readers: Developers, Businesses (Technical Background).

Section 5 Other Nonfunctional Requirements

* This section provides low level detail on functional requirements but important for development and good performance. The content of this section is good for developers to derive development tasks that must be met.

Suggested Readers: Developers.

Section 6 Other requirements

* An extension of section 5.

Suggested Readers: Developers

## References

C. Treude, M. P. Robillard and B. Dagenais, "Extracting Development Tasks to Navigate Software Documentation," in IEEE Transactions on Software Engineering, vol. 41, no. 6, pp. 565-581, 1 June 2015.

doi: 10.1109/TSE.2014.2387172

# Overall Description

## Product Perspective

The Watech Park platform is intended as a supplement for an existing system or a replacement depending on the hardware a business has.

## Product Functions

Consumer Functions:

-Find a parking lot and pay for parking

-Manage parking passes.

-View order history

-Add a car/Manage Cars

-Pay for parking tickets.

- View Order History

- Access Settings, Help, About

Business Functions

-View parking spot utilization.

-View parking revenue.

-View and manage tickets.

-View current parked users

-Access to camera data.

System Functions

-Let users in and out of parking lot based on license plate and pass.

-Issue tickets if user is over allotted time.

-Manages sensor data

## User Classes and Characteristics

The user classes for our apps are consumers, businesses and administrators.

Consumers need only to know how to use a smartphone. No technical expertise needed as it is simple to use.

Businesses will not need a lot of technical expertise. However different departments may handle different data such as finance for revenue and a point of contact for ticket disputes.

Administrators will need backend access and technical expertise to setup new platforms for each business.

## Operating Environment

-Front End: Android devices and operating systems. Android Version 5.0(Lollipop) and above.

-Backend: Google firebase database for storage.

A server to delegate data.

-On site devices:

A computational/server platform that can process images and sensor data.

## Design and Implementation Constraints

-End user interface is limited to a mobile application.

-Software must have bilingual capabilities for Canadas main languages (English and French) for country wide accessibility.

-Hardware has a max of 32gb of storage and 4gb of ram. Images stored must be compressed to save space. Images should not be too compressed such that retrieval exceeds 500ms.

-All data that passes through the internet between the application, onsite servers and backend must be encrypted.

## User Documentation

User documentation for consumers and businesses will be provided within the app. The documentation will be developed based on an action the user will perform. Methodology to be used is relating development tasks to action words and turning that into a help topic.

## Assumptions and Dependencies

* Internet connection is needed to use the application.
* To get full application functionality for business app functions, hardware will have to be installed at on site.
* Embedded Computer will always need a connection to the server for the purposes of authenticating users and getting data.

# External Interface Requirements

## User Interfaces

The user interface for our mobile application will be using the material design guidelines for color palate, simplicity and usability. The core design principle is less screens for more functionality. Users should be able to get what they want in very few actions. Our application will mainly use 2 colors with some highlights as is the design standard from most applications.

The application will be using a navigation drawer / side menu for navigation. There will also be buttons on the main screen to access common functions quicker such as account management. The main screen of the app is where the main actions will occur such as selecting a parking lot or dashboard options for business users.

## Hardware Interfaces

The Watech Park platform consists of three major hardware interfaces. The smartphone for the front end, a server to process information, and a microcontroller that also connects to a server for parking lot data.

The mobile app will have different functionality and access to different data depending on the user. The consumer will only have access to data from the server for the purposes of obtaining parking. A business user will have access to more data provided by the microcomputer to manage the parking lot such as parking availability, images and timestamps of cars that are in and out.

Each business on our platform will have a separate embedded computer to manage each parking lot or group of parking lots.

The server will handle data from both mobile app and embedded computer.

## Communications Interfaces

Protocols:

HTTPS / SSL Encryption end to end communication.

TCP / UDP Connection Protocols.

Wifi/Cellar

ServerSide:

Email: SMTP

Data:

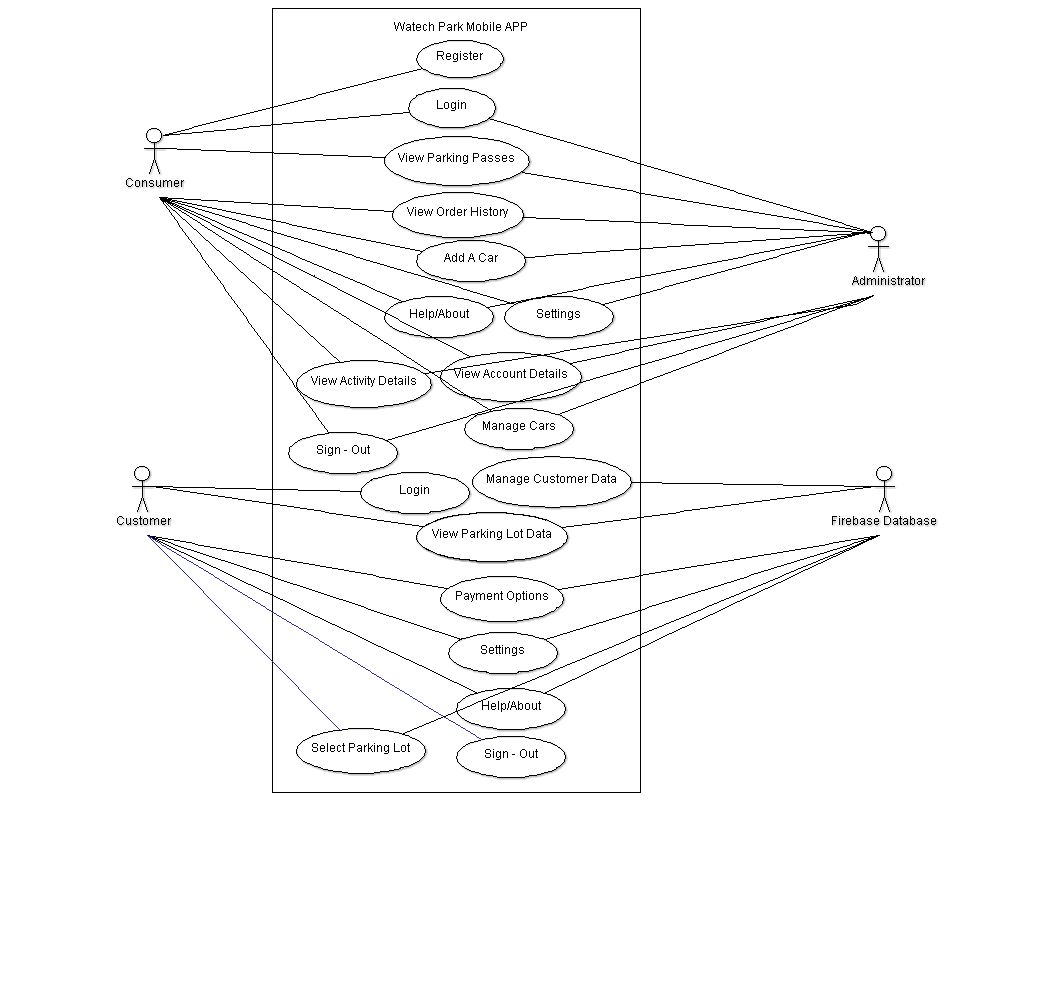
Data transfer rates must be capped to not use a lot of data. Should have some compression or limitations depending on connectivity through wifi or cellular.

# System Features

The following functional modules are the key requirements that will be implemented by the system. This section will be organized into multiple use case scenarios, where a feature would be described, and explained further with brief details with retrospect to each subheading below. Some of these use case scenarios may be different and branch off into other important aspects of the system. In this case we decided to show how the system would work with a use case diagram, as well as a user class representation to show the types of users that will be using the APP and the specific functions for each user type. We will be presenting a consumer/business model for the mobile app system.

## Use Case Representation

### Use Case View

This use case diagram showcases the different users and the interactions/relationships they have with other use cases. This includes, what features the users are able to access/or not access. In this case under a user class representation there are 3 types of users for the mobile app system which are:

* **Consumer**
* **Customer**
* **Administrator**

## Parking Lot Proximity Detection

4.2.1 Description and Priority

This feature will allow the system to react to different scenarios when a certain object is detected from a short/far distance and provide the user with real-time information in regards to a specific location. This also includes, updating details on parking space availability/status. The feature is of High priority on our features list, as it is the main objective of our application. This is because, the idea of having digital means to access a location is a much more simple and intuitive way to access information. In terms of the risks associated with the feature addition, from a scale of 1 to 9 this would stand at a relatively High (9) chance.

4.2.2 Stimulus/Response Sequences

- User selects a location, and is presented with options in relation to the available parking lots

- Data is displayed through a screen, showing the different statuses of the parking spaces in the parking lot. (open/closed or occupied)

- Sensor data is displayed(showing different proximity readings from Real-Time Firebase database)

- User is directed to proceed to park in that particular location/space outlined on the screen.

4.2.3 Functional Requirements

REQ-1: The data must be able to correctly use the proximity sensors provided with the hardware to make judgments and update the information to the database.

REQ-2: The system must be able to handle any internal/external errors that may pop up as a result of coding, such as system crashes, and report on these instances immediately.

REQ-3: The system must show detailed representation of the parking lots through the TBD data display screen which will be implemented further down the line.

## Payment/Transaction

4.3.1 Description and Priority

The Transaction System controls the business side of things, which includes the payment options present for the user at the time of ticket purchase. The payment process will be done through a QR code system with automatic gnerttaion based on total amount calculated with tax.. This feature is of High priority as it would allow user users to book tickets in a quicker format, to result in less hassle for the benefit of the consumer’s interest. The priority of benefits is also relatively high, as it would allow the ease of management, and more simpler methods for safe, secure payment methods.

4.3.2 Stimulus/Response Sequences

- Details on parking lot location is displayed to the screen, including the name.

- Floating action button present to allow users to select from a list of registered vehicles, or add a new car.

- Parking Options displayed with 3 buttons to indicate the “Hourly”, “Daily” or “Monthly” rates.

- Total required amount is calculated and the final payment information is displayed to the screen in an organized listed layout.

4.3.3 Functional Requirements

REQ-1: The system shall allow payment through only after generating a valid QR code

REQ-2: The system shall update the chosen parking options and provide the final accumulated value, including all costs, taxes.

REQ-3: The system shall not allow the transaction process to go through if the user has not generated a QR code before order confirmation.

## Registration

4.4.1 Description and Priority

This feature would require the system to require registration if not already signed up through the mobile application. The user would register to access the main features of the APP, such as payment options, vehicle management, parking lot data. This feature is of High priority as it is required to create an account through the database to use the features of the application, without being limited to a set amount.

4.4.2 Stimulus/Response Sequences

- Input fields displayed asking for “Full Name”, “Phone Number”, “Email”, “Username” and “Password”.

- Displayed checkbox to indicate user accepts the “privacy policy” – TBD (not yet implemented), checkmark is needed to move on and confirm registration.

- Entered data is sent to the database, and user is sent to the startup login/register screen to Login to their new account.

4.4.3 Functional Requirements

REQ-1: The system shall only proceed to the next activity if the input is valid from the forms.

REQ-2: The system shall display an error message if invalid input is read, or the account is already registered.

REQ-3: The system shall display a \*(asterisk) to indicate a required field from the fields missing/or incorrectly entered ones.

REQ-4: The system shall not allow successful registration if the “privacy policy” has not been read and accepted.

## Login

4.5.1 Description and Priority

The consumer signs in to the system using registered username/password identification. This feature is of High priority as access to the system would require the need of user identity authentication to proceed to other elements of the application.

4.5.2 Stimulus/Response Sequences

- User enters in “Username” and “Password” into 2 input fields.

- Optional checkbox displayed to “Stay Signed In” to the account for the duration of the session.

- Displayed link for “Forgot Your Password” for a new password creation. (if needed)

- Input fields for “Email” and “Phone Number” displayed

- User chooses which is applicable based on registered information entered into the database.

- Submit the request, confirmation code is sent to the chosen type of service.

- Verify the new password, enter the “Verification Code” into the input field and press the checkmark button to move on.

- Link to “Resend” the command is available to send the code again if not successful the first time.

- Button displayed to “Sign in With Google” to use a G-Mail account to login. (TBD – not yet implemented) but will most likely use implicit intents to have the button be clicked and access a different application though the system. In this case, it would be the G-Mail website.

- User logs in to the system if input is valid and is sent to a different screen (dashboard- main menu)

4.5.3 Functional Requirements

REQ-1: The system shall only proceed to the next screen if the input is valid and matches the registered fields from the registration screen.

REQ-2: The system shall display an error message if invalid input is read, and re-direct to repeat the step until input is valid

REQ-3: The system shall not allow to Login if the username/password does not exist in the database.

## Order History

4.6.1 Description and Priority

This feature shows the previous orders through the database, which is updated in real-time fashion. This includes, the history of the purchase, location, parking lot type, date/time and the total amount in dollars. This feature is of Medium priority, as it allows the consumer to keep a real-time log based on his/her ticket purchases for reference down the line. The benefit of this is of High priority due to this reason.

4.6.2 Stimulus/Response Sequences

- Displayed order information, including type of parking pass, purchase date, location, and the vehicle type, and total purchased amount.

4.6.3 Functional Requirements

REQ-1: The system must present the data though the database, and the information must be accurate at all times.

REQ-2: The system must keep a log of all order data through the database and update this information accordingly after order confirmation.

## Manage Cars

4.7.1 Description and Priority

This feature allows to register a vehicle into the system depending on personal preference, with a focus on vehicle management. It takes in to account license plate information to identify the recommended parking space, depending on lot availability. This is a High priority feature for us, as it is the central point to allow the user to interact with the system, not having to resort to manual ways of registering a vehicle in order to book a spot on the location. The benefit of this is also High priority, due to this reasoning.

4.7.2 Stimulus/Response Sequences

- User is prompted to “Add A Car” to the system, including entering the make, model, color, and license plate number into the input fields.

- Option to “Manage Cars” is presented, with a floating button to “Add A Car” if the user requires the need to do so.

- Car is now added into the system, and information is sent to the database.

4.7.3 Functional Requirements

REQ-1: The system shall only allow registration if all fields are entered and satisfied by the requirements

REQ-2: The system shall use the registered vehicles to allow management to occur.

REQ-3: The system shall not allow to “Manage Cars” if the vehicle is not yet registered into the system.

## Sign - Out

4.8.1 Description and Priority

After the user has finished his/her session, exploring each feature of the system he/she would need to sign out of the application. This feature would allow a multi- sign out feature for both the casual consumer and business customer.

4.8.2 Stimulus/Response Sequences

- Bars(settings) icon button is displayed on the top left corner of the main menu.

- Displays a “Sign-Out” feature to sign out from the specific account under either consumer application or business application (TBD).

- Prompted to “Confirm Sign-Out” to allow user to accept/decline the decision.

- Sign out account from either user experience

4.8.3 Functional Requirements

REQ-1: The system must be logged in to allow to sign-out from the menu.

REQ-2: The system must sign out the specific user from their account after the “Confirm Sign-Out” button pop-up is clicked. (TBD)

## 4.9 Settings

4.9.1 Description and Priority

This feature would allow the user to make changes/update the system options based on personal preferences/device needs to meet the best performance standard.

4.9.2 Stimulus/Response Sequences

- Option to “Select Language Preference” which would be a localization feature under a checkbox layout.

4.9.3 Functional Requirements

REQ-1: The system must react instantly to changes in the APP settings.

REQ-2: The system must save the settings once changed, and automatically once switched back to the previous activity.

# Other Nonfunctional Requirements

## Performance Requirements

**System Feature 1 - Parking Lot Proximity Detection:**

**The response time to show the number of available/closed/occupied parking spaces must be less than 10 seconds.** This means that when the user chooses a location then he/she should be able to view the data in real-time from the database under limited delays. It is important for the system to act as quickly as possible during this time, and not cause any unwanted delays so to get the data from the database and update it to the mobile app, data needs to be read in a fast way.

**The system must update changes on the parking lot data screen within a response time of 3 seconds.** So, data fetched from the online database must be immediately updated through the connection to the mobile app and the changes should be visible visually to the user.

**System Feature 2 - Payment/Transaction:**

**The system must take only 5-8 seconds for the transaction process to complete, and present a successful or unsuccessful message to the screen.** Therefore, sensitive information must be processed within a set amount of time, without having the user wait for specified amount of time which can cause the user to grow frustrated with the experience. We want fast, secure ways to store this information into the data base so at this rate the system can experience less tolerance for activity running in the background.

**System Feature 5 -Order History:**

**The “Order History” screen must update order information to the database and the mobile app by 15 seconds of the initial order confirmation.** This will ensure fast, reliable service to the user and store inputted information to the database for the best connection of sending/receiving data.

**System Feature 7 - Sign-Out**:

**The system must log out of the account the consumer or customer is using within 2-3 seconds of the committed action.** This requirement states how logging out of the system should be instantly done and not require the user to wait for some time. This is because, the user would perhaps want to explore the consumer and business side of things, so logging out should be done quickly from the system and the database should also keep a log of information in regards to previous users who have exited the system.

**System Feature 8 - Settings:**

**Response time to save changes must be less than 2 seconds, and for special cases such as toggling features on/off, until the next restart of the app/ next session.** The intent of this is to have the developer functions change automatically within a switch/toggle of a button. This ensures this change is taken quickly within shorter/longer durations of using the system.

**General System Performance Requirements:**

**The system must be centered on high interaction, with a simple and intuitive experience of reduced delays.** The intent of this is to have no immediate delays on the system. Such as, opening/closing dialog boxes or side bars from the main menu to access a set of features. Also, including pop up error messages, submitting/declining user requests, saving system settings or connecting to the database and retrieving information.

**The system must support a hardware system running Android version API 21 and above, any version less than the required version will not be applicable/run the app.** This is to make sure the user is under the correct operating system environment for full access to the app and its features. It is important to state the required OS to run the system to avoid any hardware issues using lower versions which may not work.

**The system must support at least 2 concurrent users to login into the system at the same time.** The intent of this requirement is to allow more than 1 user to use the system, without experiencing any lengthy traffic within the app. We decided to allow 2 users at this stage, which may be consumer or (business)customer depending on the type of account using the system.

**Response time of the password verification code email/phone service must be only 1-3 seconds**

**after the form has been submitted into the system.** For the choice of design of the password verification screen, it is necessary for the system to perform the tasks quickly and provide the service stated without any interruptions. The chosen service is indicated by the user and data is sent to the server, from there the service must provide the verification code automatically after the email/phone number information is submitted.

**The system shall take roughly 8-10 seconds to load the app fully from the splash screen to its transition into the startup login/register screen.** This is to ensure all data is ready, all assets and required resources are downloaded from the database to be used and the connection to the database is made successfully before starting to use the app. In some cases, the app may experience longer loading times depending on the size of the app, and if the connection to the database is made.

## Safety Requirements

For secure operation of the system, if there are any internal issues that occur as a result of a system crash, or a bug that is discovered through the app, we will take the most efficient safety measures to ensure there is no loss of data or harm to the system. Working code should be transferred to an external source actively to prevent permanent loss of data, and regularly checked using an agile methodology. Modifying the code of this application may result in unexpected actions to occur. Information transmission to the database and back should also be done securely without any external issues effecting the sending/receiving of data. To prevent such occurrences, it is required the consumer follow the guidelines of our terms of service/ privacy policy and not abuse any part of the system to be unusable. We will act accordingly to any issues that may occur internally as part of our coding process and aim to fix/secure the system to run smoothly on all mobile devices (from the beta release – to the final version release).

## Security Requirements

The system must allow privileges for the user to utilize the camera function of the mobile device, to scan in the license plate number information and check for valid/invalid circumstances. The system may also need to access the devices storage in order to check if there is enough space available on the device for the application to run at high efficiency, and a quicker rate with the download size. (which is TBD at the moment).

To protect the data being used or the code created by the APP itself, we will use our privacy policy. In this document, will be set of rules/regulations the user must obey to use the app, and to be identified as a safe user (without any negative intentions). The system must not allow any access to the features if the privacy policy/terms of service is not accepted beforehand. We will also have the user go through multiple steps to securely recognize an individual before gaining access to a particular account. The system must recognize any unwanted attempts by an unknown source and prevent access if there is some sort of fraud detection, potential of data being stolen, etc.

We will be applying a security policy on our payment methods as well, ensuring the generated QR code is not abused for multiple purchases in the event of a technical issue. This will be implemented further down the line, TBD. The system must only keep sensitive details in the system for a maximum of 24 hours in the database, and then it shall be deleted.

## Software Quality Attributes

**Reliability:**

The system must be reliable in serving the transaction processes, and managing sensitive information sent by the user to the database.

**Usability:**

The system needs to be usable at an extent of reduced delays and a simpler, and effective user experience. The system needs to be able to change between states for the user to experience all features of the APP.

**Compatibility:**

The preference for this quality attribute is centered on the hardware limitations the system needs to run the app successfully. The system will be compatible with most Android devices (newer models in this case) running API 21 and above. The app must also be compatible with ranges of devices/sizes with mobile devices/tablets being the main focus.

**Adaptability:**

The system needs to adapt to new environments. Such as night/day themes and change states as needed. Also, using hardware features may drain out the battery source so it is important for the system to adapt and conserve energy.

**Maintainability:**

Any bugs found, or system crashes need to be identified and with this quality attribute it would provide the ease of maintaining the system. This can be from defects within the system, this quality would help in correcting those issues. Also, prevent unexpected working conditions, in order to make sure efficiency, reliability and safety is key priority. This would also make future updates to the system easier to get done and to cope with changes in the environment. Such as, changing settings.

**Availability:**

If there are any interruptions discovered by the sending/receiving of data from the mobile app system and the database, information can be re-sent and verified again.

## Business Rules

In terms of the users who can perform specific actions: there will be 2 types of end users and the administrative role that must be followed to avoid any business interruption.

**User Class Representation:**

**Consumer:**

* The user must be able to View Parking Passes, Order History, Add/Manage Cars
* The user must be able to access system settings, help/about screens and view activity details in regards to the account profile (on the consumer dashboard)
* View Lot Details/Make Reservations

**Business:**

* The customer must be able to view the parking lot data present at the parking lot location, including space availability/status.
* The customer must only be allowed to access the payment options to make an order, and book at ticket through the transaction system.
* The customer shall be able to access system settings and help/about features.
* The customer shall not be able to access any other features from the consumer side during this business login session

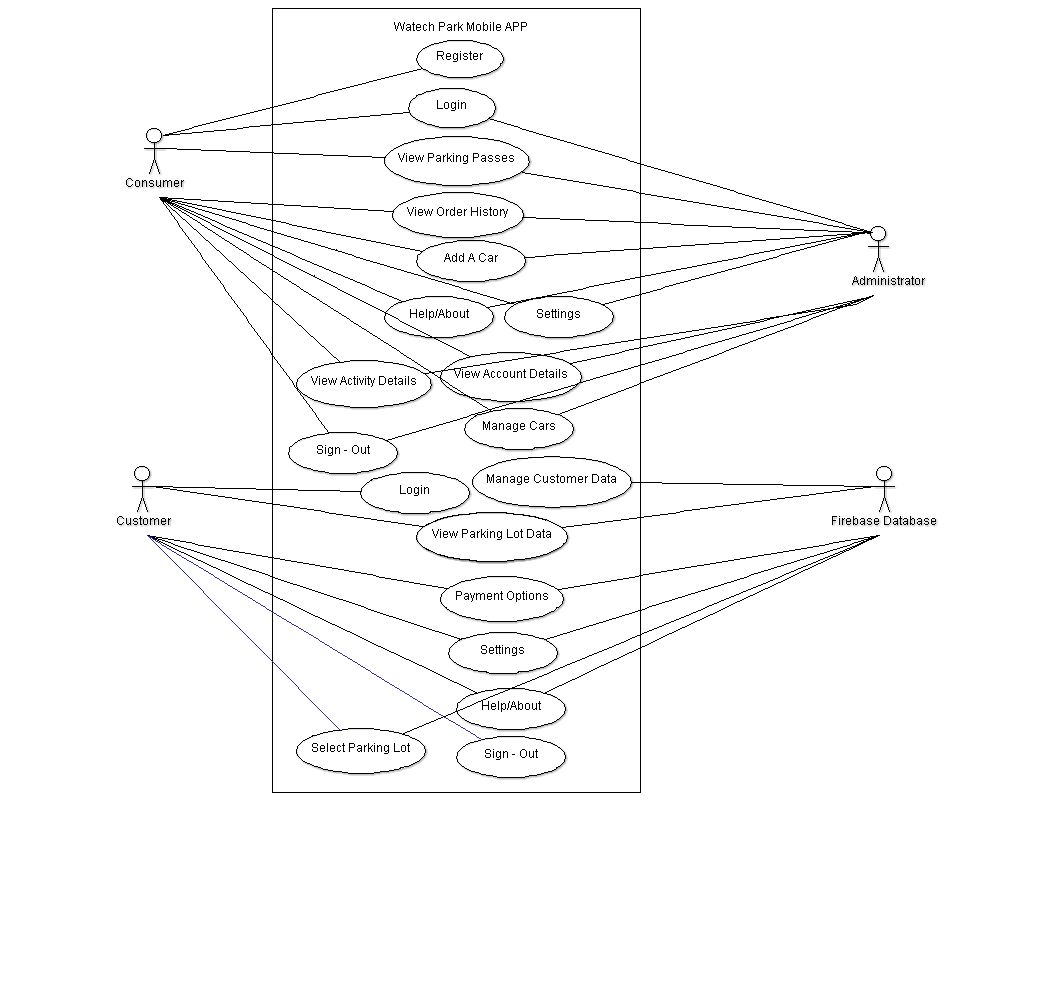
**Administrator:**

* The administrator must be able to perform all indicated actions from both the consumer and (business)customer side.
* The user must be able to control all operations including data sent/received or updating data.

# Other Requirements

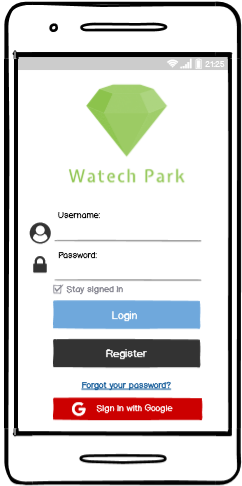
## Internationalization/Localization Requirements

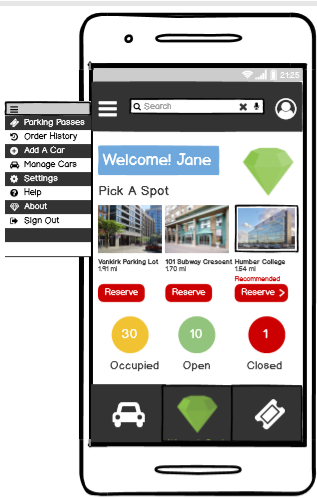
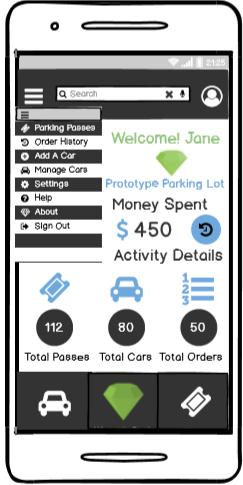
* The system must be able to handle text, number, currency and graphics files under many circumstances in different regions
* The software application must support at least 2 distinct languages: English/French



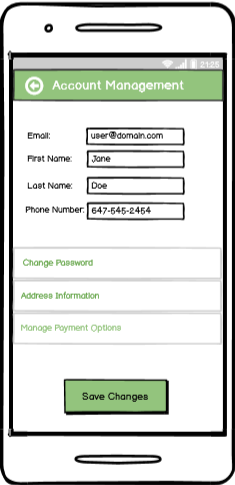
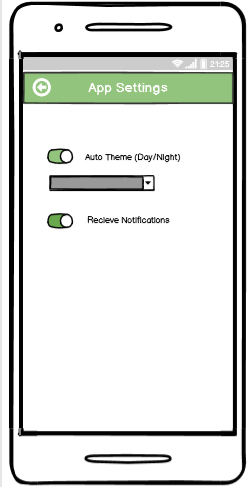
**Mockups:**

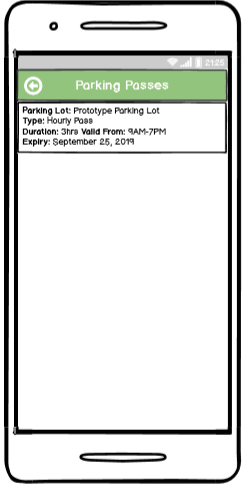
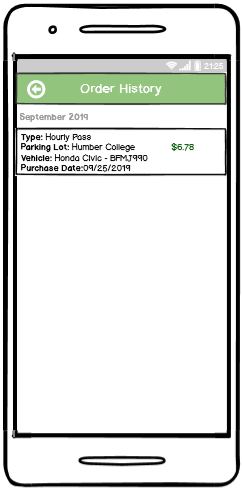
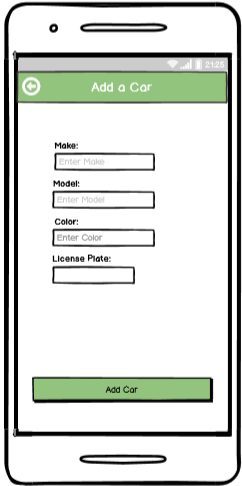
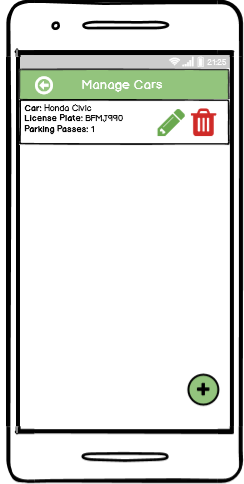
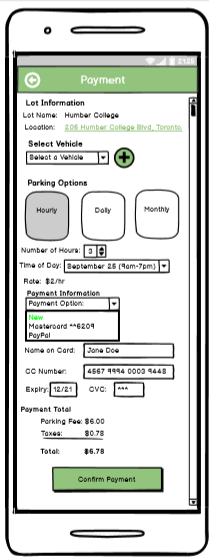
**Splash Screen Login Register**

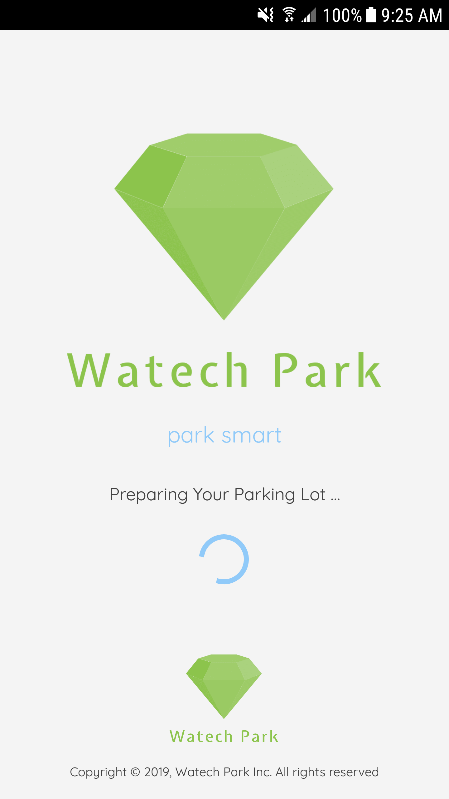
**Home (Consumer) Home(Business) Forgot Your Password**

**UI Design:**

**Splash Screen:**

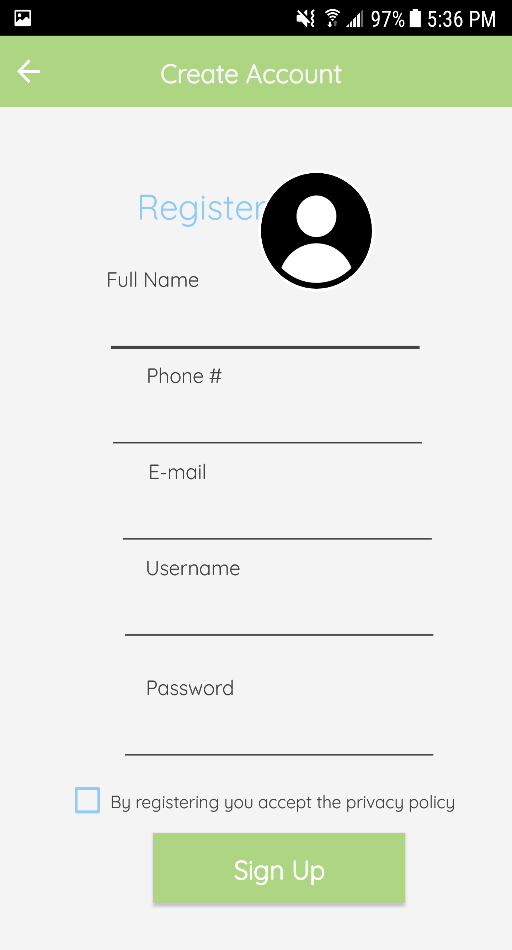
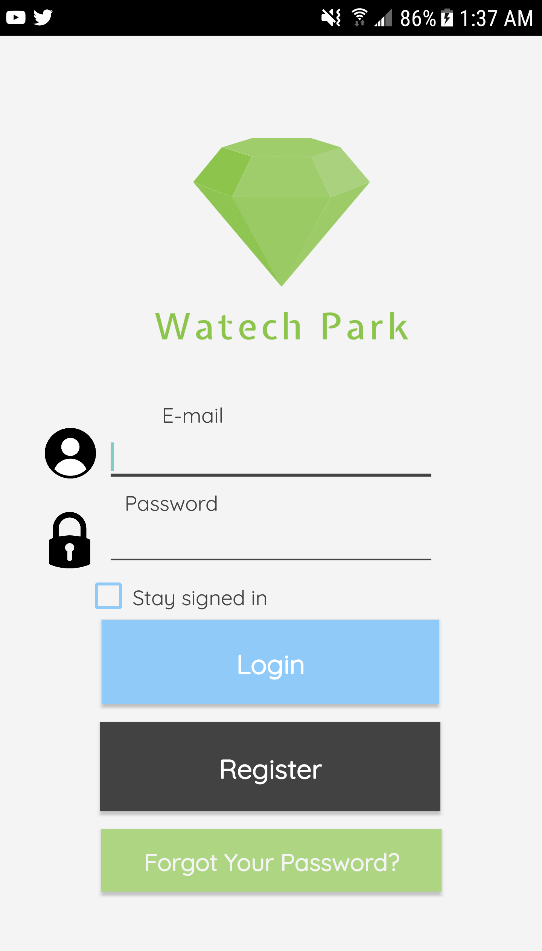


At the start of launching the app, is the splash screen which uses a ProgressBar to load all the components of the app within 3 seconds. This screen is the initial screen that acts as the bridge between the login screen. The design followed a simple layout showing an ImageView of the APP logo, followed by TextView’s to show the APP is loading into the content. The time to get into the app and access the Login screen was kept as 3 seconds for a delay, without having to keep the user waiting for too long. The Splash Screen should serve its purpose, by showing a 2-3 second delay and then launch into the app, which was the focus for designing this screen.

**Authentication:**

The user interface design followed the material design guidelines for a three-tone overall palette, industry standard. (light green, white, dark grey). Based on the project outline, the requirements were followed to ensure the design meets the standard with every screen. From this section I will be starting off with the authentication design and from there will cover the parking lot system design below and the relating user interfaces of the APP.

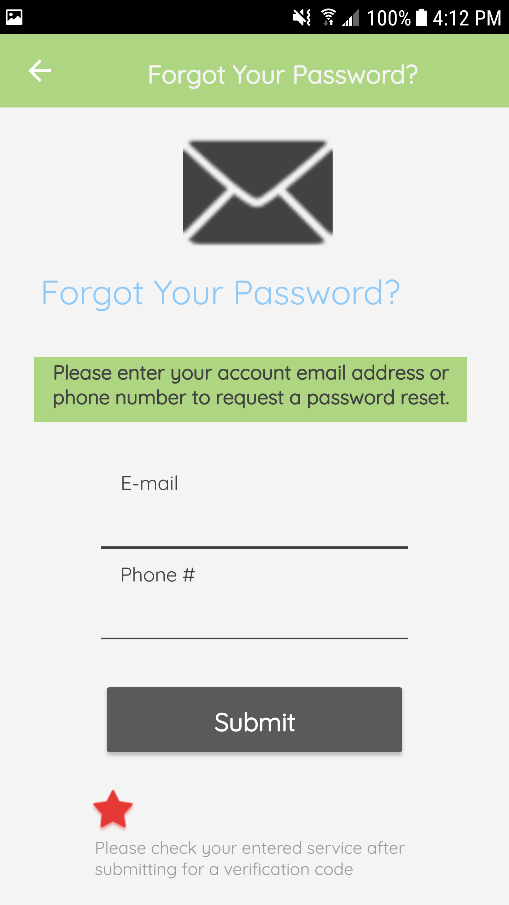
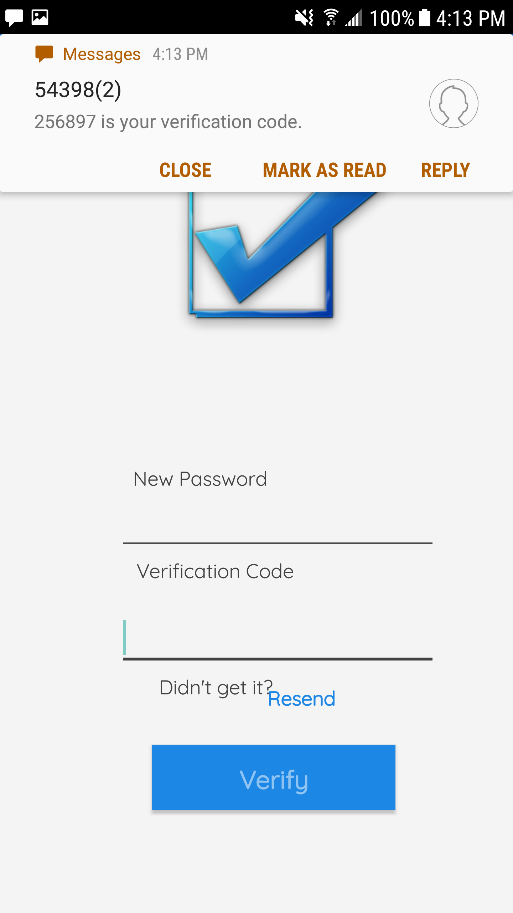
**Register Login**



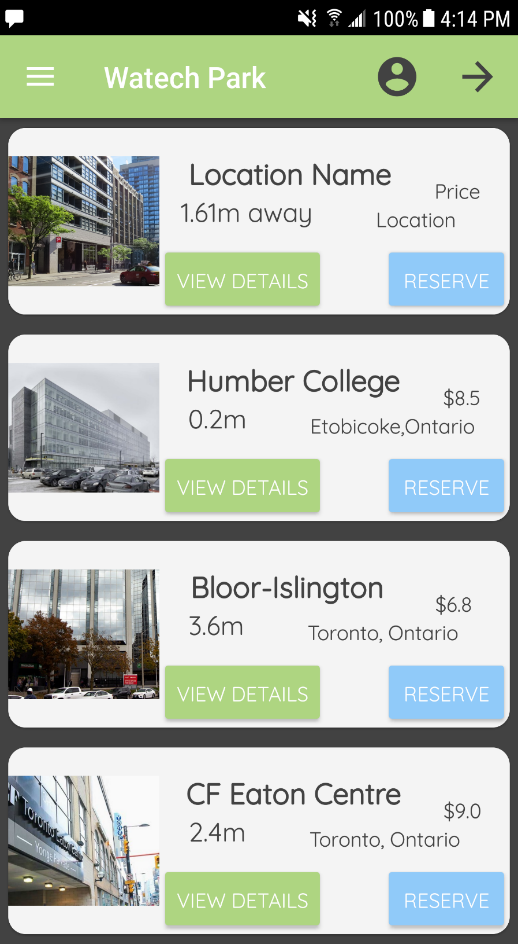
The authentication part was designed with having a login/register screen separate from each other. The Register screen prompts user input in terms of personal information specific to a user. There is also the option for the user to choose a profile image from a real device through Firebase external storage permission. This account information is submitted onto the Firebase Real-Time database under a data structure, which holds the data. This information incudes, the full name, phone #, e-mail, username, image uploaded. So, once a user registers into the system the data is sent to the “TestUsers” data structure in Firebase.

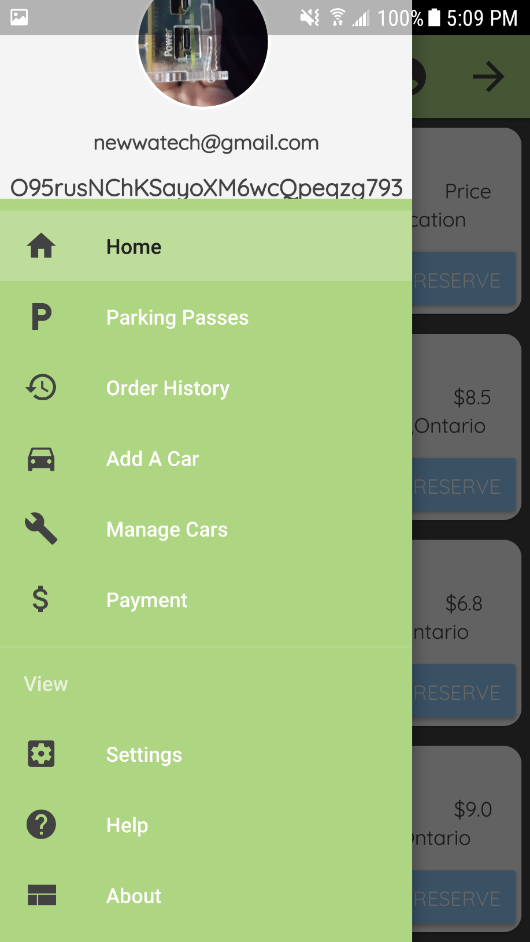
After, registering the next step is to login. To login, I used the Firebase Authentication system for e-mail and password. This is a built-in feature of Firebase, that is used to authenticate a user that exists in the database. So, in the Login screen, the user would sign-in with the “registered” email and password. Firebase checks for valid/invalid credentials based on the information stored in the TestUsers table under that UID. User selections are stored with a “Stay signed in” option using SharedPreferences. This means, account information is visible after user leaves the session, or returns to resume the activity. At this time in Firebase, the logged in user would appear in the Authentication section with the corresponding email/password information.

**Forgot Your Password Verify Your Password**



The authentication process continues with the “Forgot Your Password” and “Verify Your Password” screen. The Forgot Your Password screen is used to allow the user to reset their account password. There is the option to use e-mail or phone # authentication. This authentication is done through the Firebase database, where once a consumer selects a service, the other unattended service is not allowed to be accessed. The user enters an email address and through valid checking Firebase will then send a verification email to the corresponding email address. Phone # authentication requires the consumer to enter in a valid phone # using registration data from Firebase. Once the phone # has been validated, and follows the required system format (+1) a verification code is sent randomly through the phone service provider.

**Home - Main Dashboard:**



Once the user has logged in, the main home screen appears. This is where the parking location data is displayed, and the details regarding each location with name, distance, and the total price of the parking pass. For this screen, I redesigned the main-menu which finally had a much sleeker interface, with use of a RecyclerView to hold the list of items and wrapped into a CardView to display each parking lot location ,and its information to the screen. So, as shown above this screen offers two options to the consumers. This being “View Details and “Reserve”. View Details is where an expandable view pops up of the lot with an image, cost and the real-time proximity level of the lot at that particular time. Once the spot has been reserved, the corresponding data is sent automatically to the Parking Passes screen. At this time, the parking location that has been reserved on-the go, and its details as represented in the design are sent to Firebase. This data is stored under the “ParkingLocations” data structure in the database holding each lot that has been reserved.

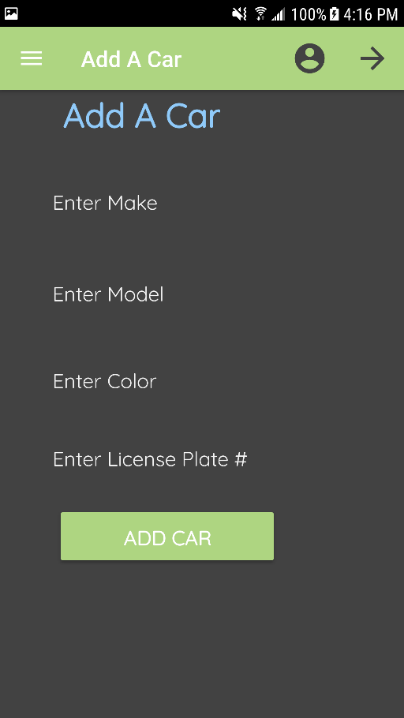
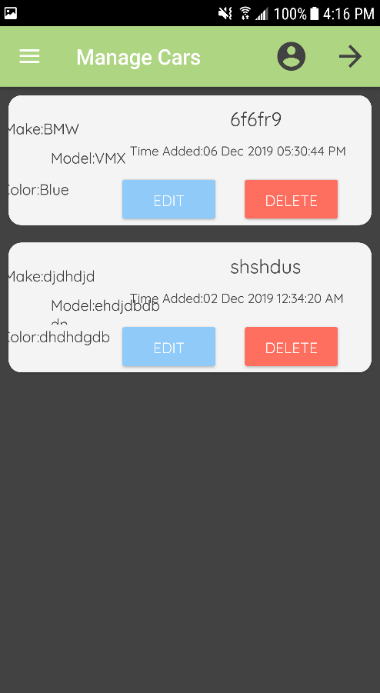
If the parking lot has been reserved successfully, a notification pops up on the device presenting the reserved lot + to view the parking passes for the next step. By swiping to the right of the screen, brings up the Side NavigationDrawerLayout that is used to hold the main/other fragments/features of the app.

**Account Management:**



This is where general account information is displayed specific to the user, in this case it would display different details based on the active UID. The screen followed a general layout, a little different from having to fit everything into a fragment view, I decided to use a new separate screen. This screen displays account information from the Register screen. The data is populated and displayed. Such as, the profile image stored from Firebase Storage, phone #, e-mail address, the name and a timestamp. For displaying the image, I used a CircleImageView for design purposes to follow the side navigation header layout in the NavigationDrawer.

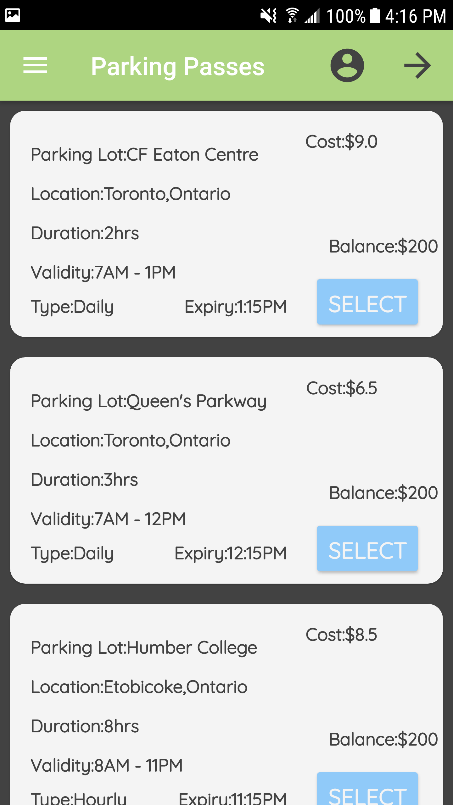
**Add A Car Manage Cars**



In the “Add A Car” screen, the user enters in their vehicle details to register a car to the account. There are 4 EditText fields, prompting for the Make, Model, Color, and License Plate #. There is a button to ADD CAR, which registers the car to the Firebase database under the “Cars” data structure. Once the car is added into the system, and the data is sent to Firebase the consumer can access these details and the registered vehicles in the “Manage Cars” screen.

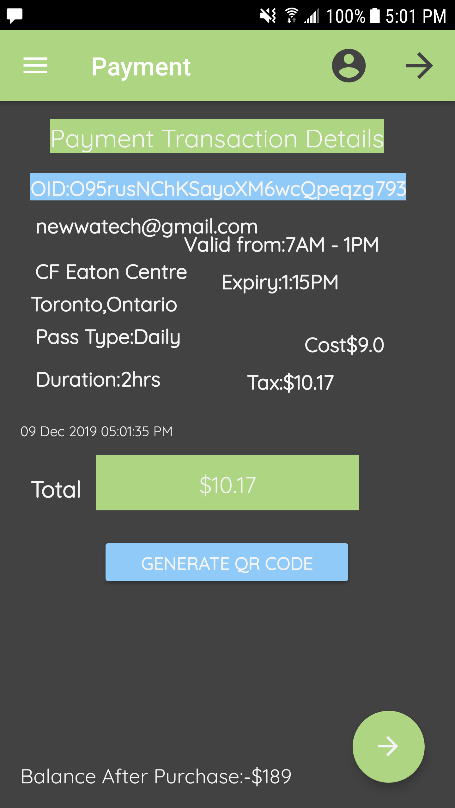
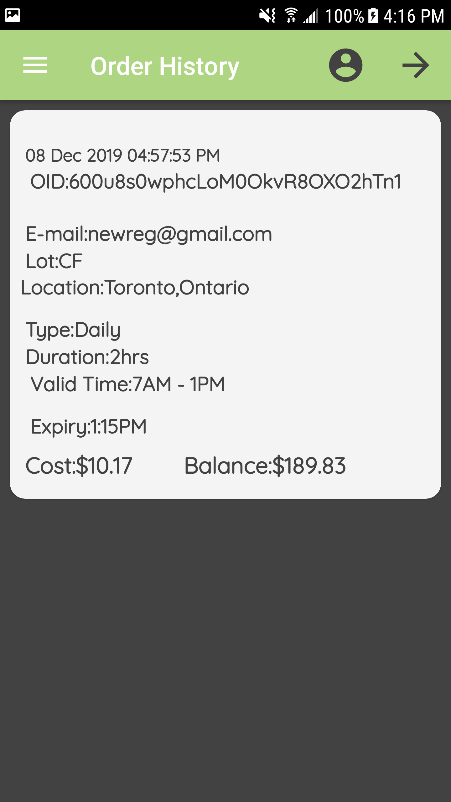
Manage Cars is where the data is fetched from Firebase database and the information for each “Car” is displayed following a similar format of the main menu of a RecyclerView plus a CardView. For this screen as well, I used the RecyclerView as it is much faster to load the elements compared to the ListView, and is more commonly used. There are 2 options to choose from here: Edit/Delete. Edit allows the user to basically make a change and update the information to the database structure in Firebase. Once the user selects Edit, an inflated view pops up of the fragment prompting to enter in the new information. The user then would tap on the “Apply Changes” button to apply the changes automatically. The changes are visible in real - time through Firebase, once they are set. The Delete option asks the user if they are sure they want to delete the car. (using a AlertDialog box). If the user approves, the car is deleted from the real-time database under the “Cars” structure and is removed automatically after the next time you access the Manage Cars section. If the latter is chosen, the action is dropped and cancelled to continue the session.

**Parking Passes:**



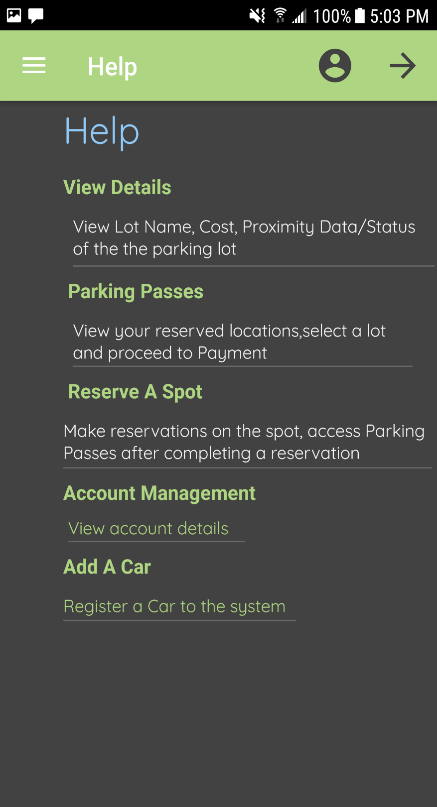
Parking Passes holds all of the available parking passes for each lot. Also, following a RecyclerView + CardView layout the data is displayed for each lot. This includes, the name, location, duration (in hours), validity (time the pass is valid for), type, expiry time, cost, and the account balance before the purchase. There is a button to “SELECT” a parking lot. Once selected the data for that lot is sent to Firebase under the “ParkingLocation” data structure and stored under the UID of the user. This data is also then sent to the Payment screen, which would be the next step to finalize the reservation through the transaction process.

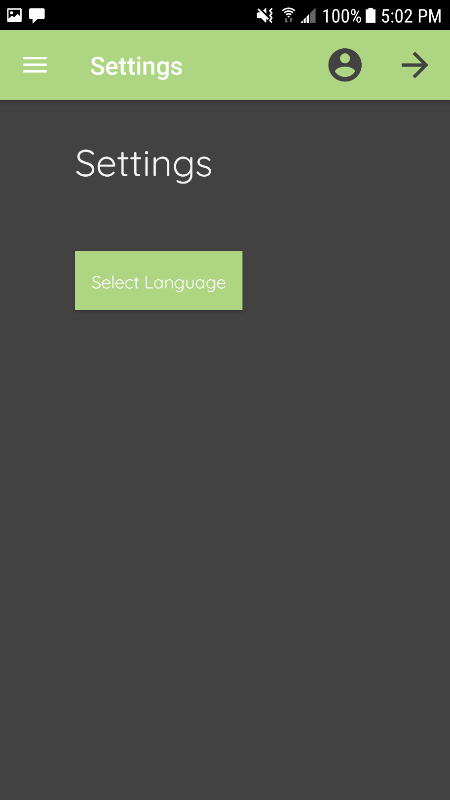
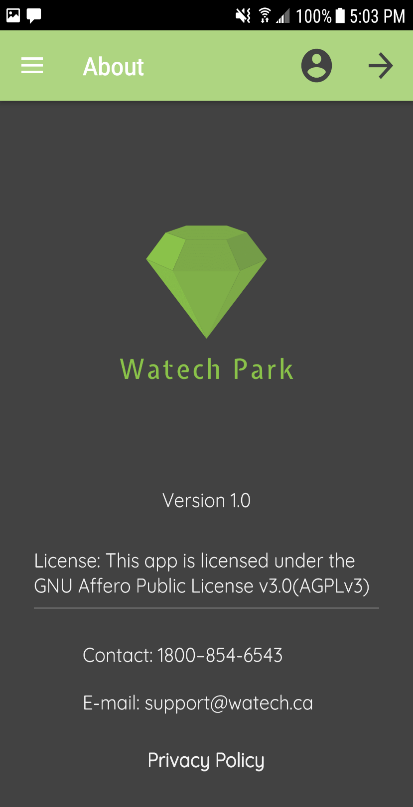
**Payment Order History**



On the Payment screen, the selected parking pass is now visible with all the data related to the particular parking lot. This data is retrieved from the “ParkingLocation” data structure from when the pass was selected. On this screen, similar details are displayed, with the addition of an OID (order ID), and e-mail corresponding to the account that is processing the order. Also, the total is calculated for the parking pass with tax and displayed in an EditText field which is only readable. The balance after the purchase is calculated on the spot and displayed according to the total accumulation. Also, instead of following a credit card method which can be risky in terms of storing sensitive data. The user would have to enter in more fields of information which would be a more inconvenient method. Instead I decided to implement a QR Code generator. The total would be calculated and based on this set value, the “Generate QR Code” button generates a random QR Code using this value. The user would then tap on the FAB (floating action button) which asks to confirm the purchase. If the order is confirmed then it is successfully been processed. A toast message appears saying “Order has been successfully placed! Please View Order History for more details”.

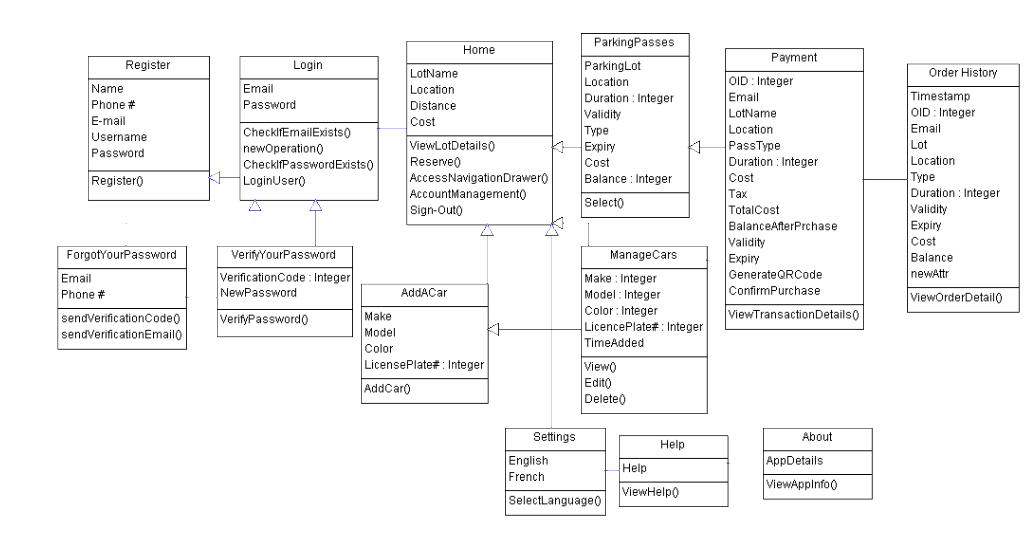
Order History displays the order’s that have been placed using a unique OID(UID) to refer to a specific account. The data is retrieved from the “Orders” data structure in Firebase displaying the processed information and a timestamp for when the order confirmation took place.

 **Settings Help About**



The Setting screen provides a localization feature for (English/French integration). The design basically followed a simple UI, with a button to “Select Language”. The button is clicked and an AlertDialog box appears with 2 options, the user checks which one to perform and the languages change state accordingly automatically without the need of re-entering the app. The Help screen, displays general help documentation for ways to navigate to the different screens and use the functionality. The About screen displays project/APP details.

**UML Class Design:**

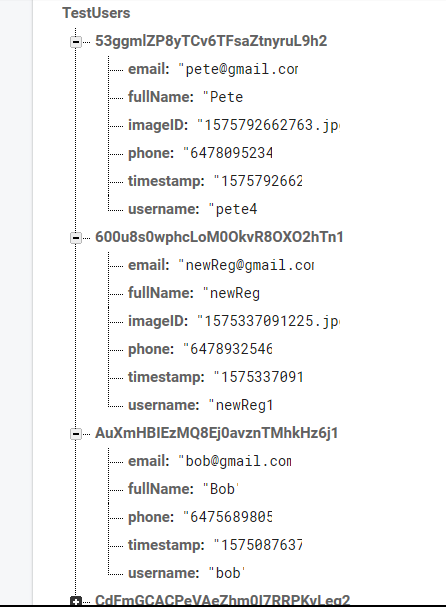
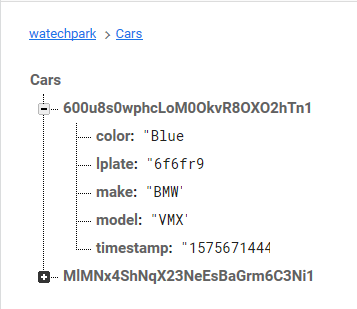
****

The process begins with the Register class. From here, the user inputs their Full Name, phone #, E-mail, username and password. After checking for empty fields/invalid input, and after a profile image is selected from external storage the user is registered into the app. As well as the data is pushed into the “TestUsers” data structure in Firebase. This is where all the registered users go. After registering, the user logs in to their account using valid email/password Firebase authentication checks. Firebase checks if the email/password exists or not, if they don’t exist in the table, entry is dis-allowed. If it exists then the user can log-in to the system to access the main menu/rest of the features of the app. There is an association between the Forgot Your Password, Verify Your Password screen. The only way to access this screen is to first have a registered account. From there, you would choose email/phone authentication service. Then you would receive a verification e-mail from Firebase to reset password authentication or a verification code. If the chosen method is phone authentication, then a new password must be typed in on the Verify Your Password screen, and the correct verification code. This way, the class design shows that Login is dependent on Register and Forgot Your Password/Verify Your Password depends on the Login screen (successful entry).

In the Home dashboard is the option to view available parking lots/locations. Also, make reservations for a spot, access the side navigation drawer, Account Management, or sign-out option. From there, once a user reserves a spot data is sent to the “ParkingLocations” data structure in Firebase. If View Details is chosen, then proximity data/status of the lot is displayed through the “ProximityData” structure in the real-time database showing different data for each lot. Other features such as Add A Car/Manage Cars depend on the main menu, with Parking Passes being the most important followed by Payment. As mentioned before, once a location is reserved, the data is sent to the Parking Passes screen. Then once a parking pass is selected the specific parking pass details is sent to the “ParkingLocations” data structure and Payment displays this information. This shows how Payment depends on Parking Passes to go through the selected process first, and then you can only access the Payment feature if a pass is selected. Of course, once a payment is made, the order details is sent to the “Orders” table and Order History views the details. Basically, showing each screen is chained to the previous and so on.

**Data Structure Design:**

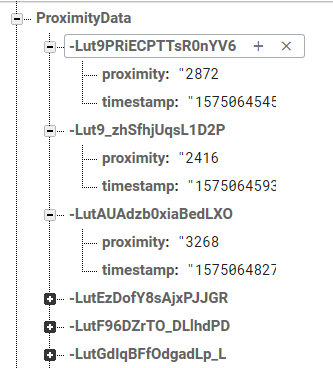
**Part 1: Authentication**

****

The database design that was used to organize each data structure included an **Authentication** and **Data** section. There are 5 main data structures to hold the data. The “TestUsers” structure is where the data is retrieved from the app in the Register screen. This information includes, the email, fullName, phone #, timestamp and username. For security purposes, the password was not sent to Firebase while transferring the data. This is because, the password that is visible online can be visible anywhere from the world. It can be used for abusing principles, such as hacking, modifying or retrieving personal data. Using an encrypted password would have also served no purpose as that may easily be traced or found by decoding.

The “Cars” data structure ties in with the “TestUsers” table because after the user registers and logs in they are authenticated to the system. Now, to reserve a parking spot the account must be registered to a vehicle through adding a car. Under the current UID is the data that is added to the structure after adding a car. This includes, the make, model, color, license plate #, and timestamp.

**Part 2: Data**



The “ProximityData” data structure holds all of the raw real-time proximity readings from the VCNL4010 Proximity sensor. In this data structure, the proximity value and the timestamp are recorded. The values randomly change state depending on the state of the parking lot.(Such as, if a car is approaching, the lot is full, or parking space is available) This proximity value is then displayed on the app, when the consumer selects a parking lot from the main menu and chooses “View Details” option.

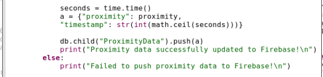
****

The reserved locations are sent to the “ParkingLocations” data structure. Including the details of the cost, lotDistance, lotImage, lotLocation, and lotName. These values are specific under each UID (or account).

The “ParkingLocation” data structure shows details of the parking pass as it is selected, and the details are sent to Firebase. This includes, the balance, duration, expiry time, cost, lot location, lot name, pass type, what time/day it is valid from and to. Under the same branch is a nested data structure called “Orders”. This data structure holds all of the processed orders/transactions and is identified by the orderID(or UID) of a specific account. In this case, it is taking the order of the same UID that has selected the pass in the first place. The data includes, the remaining balance after the purchase, cost plus tax, expiry time, location, name, pass type, valid time, duration, email, and the time purchased(timestamp).

**Hardware Implementation:**

The hardware components that are necessary for this project include 2 stepper motors, 1 camera sensor, to control the gate and allow entry/exit using valid license plate recognition (not yet implemented). VCNL4010 Proximity sensor, to detect the status of a parking lot at a given time, IR Beam sensor to detect parking lot gate opening/closing.



The strategy to implement the design/code started with initializing the sensor, and its library. Then I had to convert the data to grab the raw proximity value of the sensor from address 0x13.Once that was done to implement the proximity data to Firebase data structure “ProximityData”, I had to initialize the Firebase configuration, and the API key for the APP(WatechPark). The next step was to assign a variable with the value of the proximity, timestamp and push this value holding the two parameters to the data structure.

**Testing Cases/Trouble-Shooting**

- Test Register/Login is successful with a database. (Authentication)

- Test parking status, does it provide real time data matched in database?

- Test payment -> Deduct amount -> update parking pass screen.

- Test proximity data real-time updates, does it display the correct data?

- Test Add A Car option/ View Added Cars (in Manage Cars)

- Test Order History, does it display the correct Order(transaction) details

- Test Firebase pushing/retrieving of data

- Test Settings, localization integration (English/French)

- Test Account Management screen, can registration information be retrieved and displayed

- Test Sensor Connection to Firebase, can we read/write data to server and display on app

- Test Reserve Capability, does the parking data get sent to Firebase (with notification sent to device)

- Test Parking Passes, does it display the correct data for each lot

- Test SMS capability

- Test Phone Authentication

- Test Portrait/Landscape mode

- Test Firebase Storage, does the uploaded image appear when you log in to the app

To troubleshoot, it would be best to step through the code setting breakpoints and run the code in debug mode. For the most part, I used the debug mode to run the app on my device, and check for crashes. The most notable crashes occurred when reading from Firebase, where sometimes I would encounter the NullPointerException. I also used the logcat terminal to pinpoint exactly what is causing the issue and troubleshoot from there.

**Android Components Used:**

The components used for the project included the following:

* TextView
* ImageView
* Button
* FAB(FloatingActionButton)
* RecylerView/CardView
* CircleImageView
* NavigationDrawer
* EditText
* Checkbox
* AlertDialog

**Individual/Team Contribution:**

Development-wise I handled the majority of the coding, and connecting the APP UI’s to the Firebase database to be able to read/write data, due to my partner leaving it was up to me to finish the APP with the proposed functionalities. For the most part, I was able to get almost each feature in to the final release, with some removals, changes mid-point. I spent most of my time working with designing the APP UI’s to work with the database, making sure each screen functions as required.

**Concluding Remarks:**

Overall, the next steps are to work on each sensor and make sure the hardware functions with the database and back to the APP. Another key step that needs to be integrated is the business component of the project. This includes being able to track user activity, ticketing, orders, adding the camera sensor for valid license plate recognition. A lot of work was put into this course itself, developing the app, managing time wisely, figuring out issues/resolving those issues and moving forward to complete each milestone. For the capstone project this experience and level of work will be vital to integrate the final product with the hardware and software essentials for Winter 2020.

**GitHub Project Address:**

**GitHub Project Address:** [**https://github.com/InfiniteMaximum/WatechPark**](https://github.com/InfiniteMaximum/WatechPark)

**References:**

<https://developer.android.com/>