Final Project Document

Easy Park

USF Software Engineering

CEN4020

Team 007

**Name 1: Ahsan Wahab**

**U69744584**

**Name 2: Davin Hill**

**U03202325**

**Name 3: Kaleb Hoff**

**U66344603**

**Table of Contents**

[**1.** **Introduction** 3](#_Toc26730437)

[**1.1** **Purpose** 3](#_Toc26730438)

[**1.2** **Scope** 3](#_Toc26730439)

[**1.3** **Definitions** 4](#_Toc26730440)

[**1.4** **External References** 5](#_Toc26730441)

[**2.** **Detailed User Requirements** 5](#_Toc26730442)

[**2.1** **Assumptions, Dependencies and Constraints** 5](#_Toc26730443)

[**2.2** **User Interfaces** 6](#_Toc26730444)

[**2.3** **Hardware Interfaces** 7](#_Toc26730445)

[**2.4** **Software Interfaces** 7](#_Toc26730446)

[**2.5** **Communication Interfaces** 7](#_Toc26730447)

[**2.6** **Functional Requirements** 7](#_Toc26730448)

[**2.7** **Performance Requirements** 10](#_Toc26730449)

[**3.** **Detailed System Specifications** 11](#_Toc26730450)

[**3.1** **System Specifications** 11](#_Toc26730451)

[**3.2** **Middleware** 12](#_Toc26730458)

[**5.** **Detailed Design** 13](#_Toc26730465)

[**5.1** **Architectural Strategies** 13](#_Toc26730466)

[**5.2** **Model Graphics and User Interface Details** 13](#_Toc26730467)

[**5.3** **Acceptance Criteria** 18](#_Toc26730478)

[**6.** **Software Engineering Model** 19](#_Toc26730479)

[**6.1** **Waterfall Engineering Model** 19](#_Toc26730480)

[**6.1.1** **Requirements and Analysis** 20](#_Toc26730481)

[**6.1.2** **Design** 20](#_Toc26730482)

[**6.1.3** **Implementation** 20](#_Toc26730483)

[**6.1.4** **Testing** 20](#_Toc26730484)

[**6.1.5** **Deployment and Maintenance** 20](#_Toc26730485)

[**6.1.6** **Conclusion Using Waterfall** 20](#_Toc26730486)

[**7.** **Decomposition Model** 20](#_Toc26730487)

[**7.1** **Functional Decomposition Diagram** 20](#_Toc26730488)

[**7.1.1** **Login Decomposition Diagram** 21](#_Toc26730489)

[**7.1.2** **Logged-in Decomposition Diagram** 22](#_Toc26730490)

[**8.** **Maintenance** 24](#_Toc26730491)

[**9.** **Testing Plan** 24](#_Toc26730492)

[**10.** **Future Implementations** 28](#_Toc26730493)

[**11.** **Estimated Time and Cost** 28](#_Toc26730494)

[**12.** **Change Log** 28](#_Toc26730495)

# **Introduction**

# **Purpose**

This document provides a comprehensive architectural overview of the system, using several different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

# **Scope**

The “Easy Park” software is a mobile application that allows people to pay for a parking spot through their phone. When the user enters a parking lot, the application shows the available parking spaces in the lot. Then the user can go to a free space, park their car, and scan the QR code for that space to pay for it. The application should be free to download from GitHub link as an APK file that can directly be installed on the phone.

# **Definitions**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| User | Someone who interacts with the mobile phone application |
| Admin/Administrator | System administrator who is given specific permission for managing and controlling the system |
| Android | Android is a mobile operating system developed by Google. |
| Firebase | It is a back-end service that is based on the Google cloud platform. It provides developers with tools including a no-SQL database that can be integrated with the application. |
| QR-code | It is a matrix like barcode. |
| API | Application programming interface |
| Application Store | An installed application on mobile phone which helps user to find new compatible applications with mobile phone platform and download them from Internet. |
| Stakeholder | Any person who has interaction with the system who is not a developer. |
| Java | It is a general-purpose programming language. |
| PayPal | An online financial service that allows you to pay for items using a secure internet account. |
| Cloud Firestore | Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. |
| DESC | Description |
| RAT | Rational |
| DEP | Dependency |
| FR | Functional requirement |
| TC | Test case |

# **External References**

1. Google Firebase, accessed 22 September 2019, <https://firebase.google.com/docs>
2. Android Developer, accessed 22 September 2019, <https://developer.android.com/studio/intro>
3. Google Developers Bar Code, accessed 22 September 2019, <https://developers.google.com/vision/android/barcodes-overview>
4. \*QR Code Generator API, accessed 22 September 2019 <http://goqr.me/api/>
5. PayPal Developer Documentation, accessed 22 September 2019, <https://developer.paypal.com/docs/api/overview/>
6. Academia, Software Requirements for Name of Mobile App, accessed 22 September 2019, <https://www.academia.edu/11325107/software_requirements_specification_for_name_of_mobile_app>
7. Stephen Watts, BMC blog, accessed 22 September 2019, <https://www.bmc.com/blogs/software-requirements-specification-how-to-write-srs-with-examples/>
8. Android Developer Training, accessed 22 September 2019 <https://developer.android.com/training/basics/firstapp>
9. Pfleeger, S. & Atlee, J. (2010). Software engineering: theory and practice. Upper Saddle River N.J: Prentice Hall

# **Detailed User Requirements**

# **Assumptions, Dependencies and Constraints**

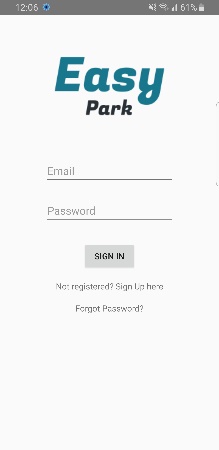
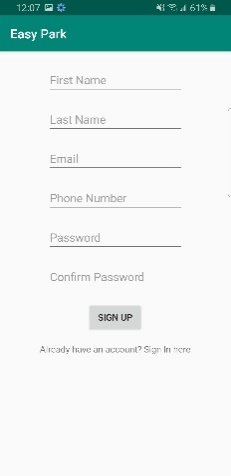
One assumption about the product is that it will always be used on mobile phones that have enough performance. If the phone does not have enough hardware resources available for the application, for example the users might have allocated them with other applications, there may be scenarios where the application does not work as intended or even at all.

Another assumption is that we are assuming that the camera quality of the mobile device is good enough the scan the QR code. If it generates poor quality image, the application will not be able to recognize the QR code.

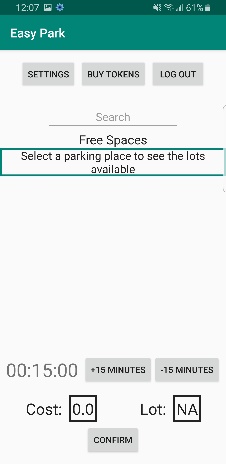
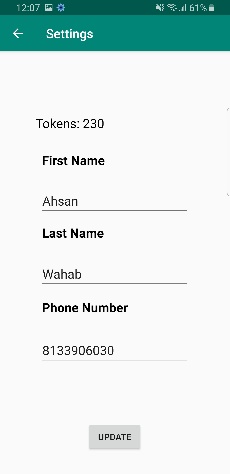
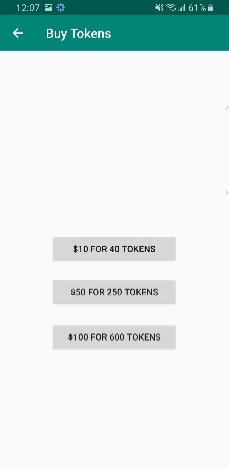
The last assumption for this application is that Google Firebase will always work. As all the back-end components of the application are taken from Firebase, a failure in Firebase will stop the application from working for all the users and might even result in user data being lost or deleted.

# **User Interfaces**

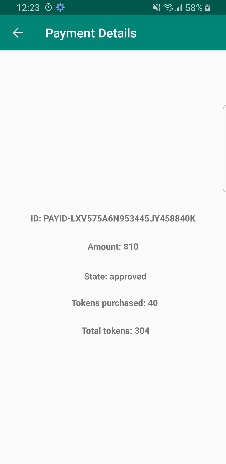
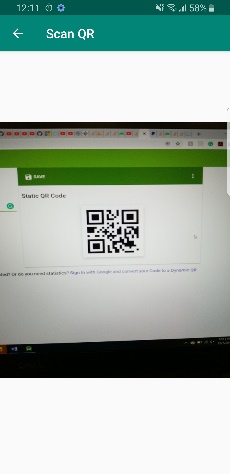
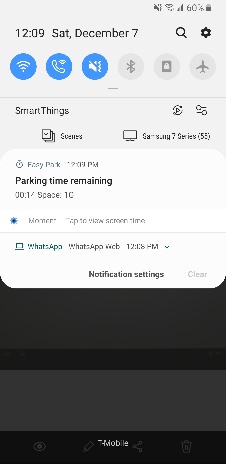
A first-time user of the mobile application should see the log-in page when they open the application, see Figure 1.2.1. If the user has not registered, they should be able to do that by going to the Sign-up page, see Figure 1.2.2, by clicking “Don’t have an account? Sign up here”. If the user is not a first-time user, they should be able to see the home page directly when the application is opened, see Figure 1.2.4. The home page will allow the user to look up a parking lot and see the number of available parking spots and their ids. Moreover, the user can also navigate to the packages page, see Figure 1.2.6, go to

**Figure 1.2.1 - Login page Figure 1.2.2 – Sign-up page Figure 1.2.3 – Reset password**

**Figure 1.2.4 - Home page Figure 1.2.5 - Update page Figure 1.2.6 – Package page**

   **Figure 1.2.7 – Payment page Figure 1.2.8 – QR Scanner page Figure 1.2.9 – Notification**

update page see Figure 1.2.5, or logout. The user can enter the time he wants to park the car for and confirm it. This shows the user the total cost for his specified time. The same text box can also be used to update time after the user has purchased a spot. The page also provides user the option to add a car. Once the user is inside the parking lot, he/she can confirm the location and go to the QR code scanner screen, see Figure 1.2.8.

# **Hardware Interfaces**

The mobile application needs to communicate with the camera of the phone it is running on. This will allow the application to scan the QR codes for available parking spaces. The hardware connection to the database server is managed by the underlying operating system on the mobile phone.

# **Software Interfaces**

The mobile application communicates with the PayPal API to allow the user to buy more packages, QR code API to check if the scanned code is valid and with Firebase in order to get the information about the user and the parking lot. The communication between Firebase and the mobile application consists of both reading and writing the data.

# **Communication Interfaces**

The communication between the different parts of the system is important since they depend on each other. However, in what way the communication is achieved is not important for the system and is therefore handled by the underlying operating systems for both the mobile application and the web portal.

# **Functional Requirements**

This section includes the requirements that specify all the fundamental actions of the software system. The requirements are prioritized in the order listed.

**ID: FR1**

TITLE: Download mobile application

DESC: A user should be able to download the mobile application through either an application store or similar service on the mobile phone. The application should be free to download. RAT: For a user to download the mobile application. DEP: None

**ID:FR2**

TITLE: User registration - Mobile application

DESC: Given that a user has downloaded the mobile application, then the user should be able to register through the mobile application. The user must provide first name, last name, phone number, email address, password, and confirm password. The Phone number should be 10 digits only. RAT: For a user to register on the mobile application. DEP: FR1

**ID: FR3**

TITLE: Mobile application - User log-in

DESC: Given that a user has registered, then the user should be able to log in to the mobile application. The log-in information will be stored on the database and in the future the user should be logged in automatically when connected to the internet. RAT: For a user to login the mobile application. DEP: FR1, FR2

**ID: FR4**

TITLE: Mobile application - Update page

DESC: The user should be able to update his information by navigating to the Update page. This information includes the first name, last name, phone number (10 digits only). The user should also be able to view the number of tokens left on the same page.

RAT: This enables the user to update his information in case it changes. DEP: FR3

**ID: FR5**

TITLE: Mobile application - Search

DESC: Given that a user is logged in to the mobile application, then the first page that is shown should have a dropdown with search bar on the top. The user should be able type the name of the parking lot and search for it. RAT: For a user to search for a parking lot. DEP: FR3

**ID: FR6**

TITLE: Mobile application - Search result

DESC: The parking lot that the user selects from dropdown search bar is displayed on the home screen. All the free spots themselves should be listed below the search bar. RAT: Defining how the result of the search will be displayed. DEP: FR5

**ID: FR7**

TITLE: Mobile application – Setting a parking time

DESC: The user should be able to set a parking time of his choice. The time should be no less than 15 minutes and no greater than 2 hours. Time can be updated in increments of 15 minutes. As soon as he enters the time, he should be able to view the tokens it will cost him to park for that time. RAT: To define your parking time. DEP: FR6

**ID: FR8**

TITLE: Mobile application – Buying token packages

DESC: The user can buy token packages of their choice. The packages must be paid for by PayPal. RAT: Defining the payment system. DEP: FR3

**ID: FR9**

TITLE: Mobile application – Payment details

DESC: The user should be able to view the details of the transaction they made to buy the packages. RAT: Enabling the user to see how much money they spent and if the transaction was successful. DEP: FR8

**ID: FR10**

TITLE: Mobile application – Switching to the QR code scanner

DESC: A user should be able to switch to the QR code scanner if they have adjusted the times according to their choice. RAT: For a user to scan the parking spot QR code. DEP: FR7

**ID: FR11**

TITLE: Mobile application – Scanning the QR code

DESC: A user should be able to successfully scan the QR code for the parking spot through his/her phone if the spot is free and they have enough tokens available to buy the spot for their specified spot. RAT: To book the spot. DEP: FR10

**ID: FR12**

TITLE: Mobile application – Timer notification

DESC: The application should generate a notification with a timer for the booked time. This notification should also have the id for the current booked spot. RAT: Enabling the user to see the remaining time and his spot. DEP: FR11

**ID: FR13**

TITLE: Mobile application – Clicking notification

DESC: The user should be able to click the notification which will take them to the home page. RAT: Opening the application with current data. DEP: FR12

**ID: FR14**

TITLE: Mobile application – Adding more time

DESC: The user can add more time when he opens the application through the notification and his/her current time has ended. RAT: If the user needs to park for more time. DEP: FR13, FR12

**ID: FR15**

TITLE: Mobile application – Reset Password

DESC: The user should be able to reset his/her password by going to a reset page from the log in page. There they will be able to provide their email address which will prompt Firebase to send a password reset email.

RAT: This enables the user to update their password in case they forget it. DEP: FR3

**ID: FR16**

TITLE: Mobile application – Logout

DESC: The user can successfully logout of the application if they have no parking lot booked now. RAT: Enabling the user to log out of this account and use another account. DEP: FR3

# **Performance Requirements**

The requirements in this section provide a detailed specification of the user interaction with the software and measurements placed on the system performance.

**ID: PR1**

TITLE: Load time

DESC: The load time for the user interface screens shall take no longer than 10 seconds. RAT: Fast loading of screen. DEP: none

**ID: PR2**

TITLE: Login time

DESC: The application shall verify the user login information and allow the user to login in more than 10 seconds. RAT: Fast login time. DEP: PR1

**ID: PR3**

TITLE: Search time

DESC: The search feature in the application shall take no more than 10 seconds to display the results. RAT: Fast search time. DEP: none

**ID: PR4**

TITLE: Buying packages

DESC: The application shall take no longer than 10 seconds to initiate a payment transfer from the user’s PayPal account. RAT: Fast payments. DEP: PayPal API

**ID: PR5**

TITLE: Cost calculation

DESC: The application shall take no longer than 3 seconds to correctly calculate and display the cost of the parking according to the time that the user inputs. RAT: Fast calculations. DEP: none

**ID: PR6**

TITLE: QR code scanner

DESC: The application shall take no longer than 15 seconds to scan and confirm a valid QR code. RAT: Fast QR code scan. DEP: QR code scanner API

**ID: PR7**

TITLE: Reset password

DESC: The application shall take no longer than 30 minutes to send a reset password email. RAT: Fast reset. DEP: Firebase

**ID: PR8**

TITLE: Notification click

DESC: Clicking the notification shall take no longer than 15 seconds to load the application. RAT: Adding time fast. DEP: PR1

# **Detailed System Specifications**

System specifications are how our requirements are implemented in our application. Each one maps directly to one of our requirements and shows how that requirement will be fulfilled.

# **System Specifications**

# **Create Account – Satisfied by Requirements: FR2**

When application is opened, on the first screen that opens up (Login Screen) there is a text button to click to register for the app. When clicked, the user is taken to a screen where they can enter their credentials into text boxes. The credentials they enter are tested to make sure that they are adequate to create an account, and if they are, then an account for them is created and added to the Firebase Database.

# **Login – Satisfied by Requirements: FR3**

When the application is opened, on the first screen that opens up the user can enter their account email and password and then click a sign in button that allows them access to the application if the information entered is shown to have an account linked to it in the database.

# **Reset Password – Satisfied by Requirements: FR15**

On the Login screen of the application, the user has the option to reset their password. If clicked, then the user is taken to a screen to enter the email that is linked to their account. If this email is connected to an account in the database, then an email will be sent to the email address entered that allows the user to update their password for their account.

# **Purchase Tokens – Satisfied by Requirements: FR8**

Once the user is logged in, they will be sent to the home screen and there will be a button to buy tokens. If clicked, then the user will be sent to a buy tokens screen where they can choose between three different token packages: 40 tokens, 250 tokens, or 600 tokens. When one of these is clicked, the user will be sent to a PayPal screen where they can either pay with a PayPal account or pay with a card. Once finished, then the user will be sent to a payment confirmation screen that shows payment information and the tokens will be added to their account in the database.

# **Purchase Parking – Satisfied by Requirements: FR5, FR6, FR7, FR9, FR10, FR11, and FR14**

On the home screen, the user should be able to add or subtract time in 15 minute intervals by using buttons that represent plus fifteen or minus fifteen minutes. This lets them choose the amount of time they want to buy a parking spot for, and only between fifteen minutes and two hours is allowed. Once a time is chosen, then the user can press a confirm button that takes them to a screen that shows their camera. The user will then scan a QR-code that will be in front of a parking spot and once that is done, then a timer will start on their phone, as a foreground service, that represents how much time is left on their spot. The database will update the spot to false, meaning it is occupied, till the timer runs out. When the timer runs out, then the spot will be updated to true, meaning it is open, unless the user clicks on the notification and adds more time.

# **Update User Information – Satisfied by Requirements: FR4**

When the user is logged in and on the home screen, there will be a settings button that, when clicked, sends the user to a page that shows their account information. There will be text boxes that show the user’s first name, last name, and phone number. If the user desires to update these, they can change the text in one or multiple text boxes and then press the update button and then the database will be updated with the new information.

# **Middleware**

Middleware is the extra software used to help facilitate and simplify the process of the main function of the application. The middleware software of APIs we will be using are PayPal API, QR Code Scanner, and Google Firebase. Each of them is described in detail below and how they assist our applications functions.



# **PayPal API**

Electronic E-commerce that simplifies the online payment method. Users sensitive data is stored there and not on our servers. We will use PayPal to process payments for us and then return if the payment went through so the user can purchase tokens.

# **QR Code Scanner**

Scans QR codes quickly and identifies what the codes information contains. For our purposes it will be used to identify parking spots and for the user to scan it when they park and the data for that parking lot will automatically be populated and the user just must confirm the purchase

# **Google Firebase**

Google service that provides data space to manage and manipulate it as required. For our purposes we store user’s data and their tokens as well as other information.

# **Detailed Design**

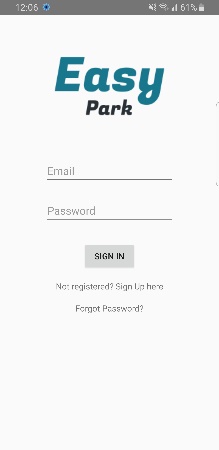
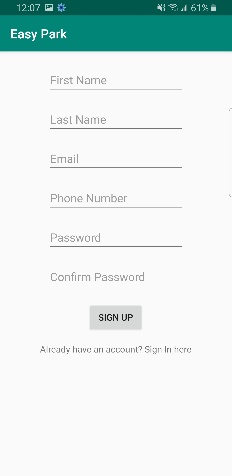
# **Architectural Strategies**

The strategy that our group is used for the development of the Easy Park application is pipes and filters. Since we are doing functional decomposition, it works well by breaking things down by modules and having each module send specific data and receive specific data. Our application can be broken down into modules and each part has its own function and sends data via pipes and filters to the next module or function to be processed.

# **Model Graphics and User Interface Details**

Our application users interface takes the users input through the hardware of the device and then gets processed based off users’ actions. The user interface is designed to help the user quickly understand and simplify the process. Below we have our prototypes of our user interface.

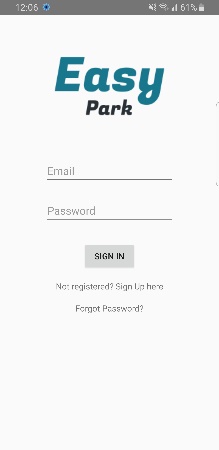
# **Create Account Activity – Requirements Satisfied: F2**

# **The activity should have:**

* + - A button to go to registration page
    - Text box for users email address input
    - Text box for users first name
    - Text box for users last name
    - Text box for user’s phone number
    - Test box for user’s password
    - Text box to confirm user’s password
    - A button to submit their information
    - A text-based button to go to the sign in screen

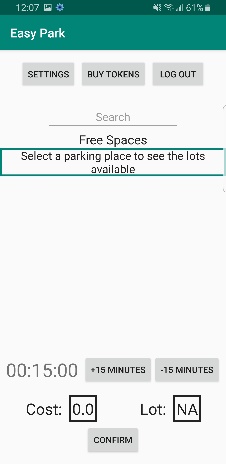
# **Login Activity – Requirements Satisfied: F3**



# **The activity should have:**

* + - Text box for user email
    - Text box for user password
    - A button to submit user information
    - User receive error if information entered incorrect
    - A text-based button to go to the sign-up screen
    - A text-based button to go to the reset password screen

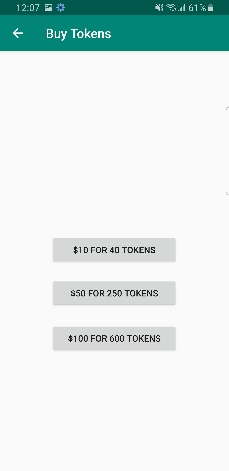
# **User Home Page Activity – Requirements Satisfied: FR4, FR5, FR6, FR7, FR8, FR10**



# **The activity should have:**

* + - Text search bar for parking lots
    - A button to go to purchase packages screen
    - A button to take the user to update their information
    - A button to Logout
    - Text box of available parking spots a user can select
    - A confirm button
    - A button to add 15 minutes to the time
    - A button to subtract 15 minutes from the time

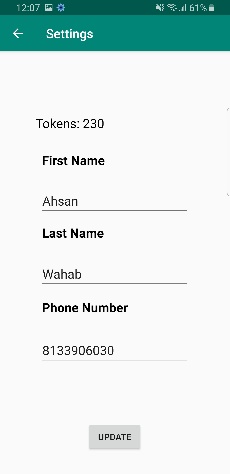
# **Purchase Time Activity – Requirements Satisfied: FR10**



# **The activity should have:**

* + - Three different buttons to select to purchase tokens
    - Button to go back to home page
    - If user selects a package, they will be taken to PayPal

# **Update User Information Activity – Requirements Satisfied: FR4**



# **The activity should have:**

* + - A button to go back home
    - Text box for user’s first name
    - Text box for user’s last name
    - Text box for user’s phone number
    - A button to update
    - A text box that shows the current amount of tokens

# **Acceptance Criteria**

Below are the acceptance criteria for each activity that we have for our application. These criteria must be meet for the application logic to flow correctly and process information.

* **New User Acceptance Criteria**: **FR Satisfied:** **FR2**
  + First name should be text, less than 50 characters
  + Last name should be text, less than 50 characters
  + Email should be unique (Not already exist in database)
  + Phone number should be text, exactly 10 characters. (3 for area code, 7 for number)
  + Password should be text, with any characters possible
  + Second password entry must match first
* **User Login Criteria**: **FR Satisfied: FR3**
  + User email that is in the database
  + User password that matched user email in database
* **User Home Page – Parking Lot Search Criteria**: **FR Satisfied: FR5, FR6**
  + User enters characters or word that match a known parking lot
* **User Home Page – Purchase Parking Criteria**: **FR Satisfied: FR11, FR12**
  + User changes the amount of time with 15-minute intervals
  + Clicks confirm
  + Scans QR bar code with camera
* **Package Purchase Page Criteria**: **FR Satisfied: FR10**
  + User selects desired tokens package by clicking one
  + PayPal confirms purchase
* **Update User Information Criteria**: **FR Satisfied: FR4**
  + First name should be text, less than 50 character
  + Last name should be text, less than 50 characters
  + Phone number should be text, exactly 10 characters. (3 for area code, 7 for number)
  + Password should be text with every character allowed
  + Second password entry must match first

# **Software Engineering Model**

# **Waterfall Engineering Model**

We used the Waterfall Engineering Model which is a software development model where development is completed in phases: Requirements and Analysis, Design, Implementation, Testing, and Deployment/Maintenance. In this model, phases are completed in a sequential manner where once one phase is complete, changes cannot be made to that phase. This is where we ran into issues with the model, as we had not made many Android projects before, so we ran into issues during the implementation phase and therefore had to make changes to our design and requirements. This normally is not allowed in Waterfall, but was necessary for us, and shows Waterfalls biggest downside, which is that it does not allow the change that is necessary in software development. So, while the easy to follow structure was nice, we did not completely follow Waterfall as it is supposed to be followed.

# **Requirements and Analysis**

In this phase we came up with requirements and specifications which is basically what we wanted our application to accomplish. Then we began thinking about what we would employ (such as APIs) to make these requirements into a working application.

# **Design**

In the design phase we created a low-level design of how we wanted the application to work, with things such as a decomposition model and UI examples. We decided on the main activity screens of the app and how they would work together.

# **Implementation**

This is the phase where the coding of the application occurred. We had to put the design of the application that we had from the previous phase into code. This is where we ran into issues as we had to make changes to the design to get our application to be functional as we wanted it.

# **Testing**

After building the application, we went into the testing phase. This involved just normal real-world usage and test cases from our testing plan. Some of our time was also spend trying to “break” the app by doing things that the user should not do but might (such as force closing the app while the timer is running).

# **Deployment and Maintenance**

This is the phase we are in now. As we do not have the time to upload Easy Park to a major app store such as Google Play, instead we are for now just uploading to GitHub. Later in the future, if we were to keep pursuing this project, an upload to Google Play would be necessary to allow for higher visibility to the general public. In this phase if any problems are found with the app we can update it and then release the new work as an update.

# **Conclusion Using Waterfall**

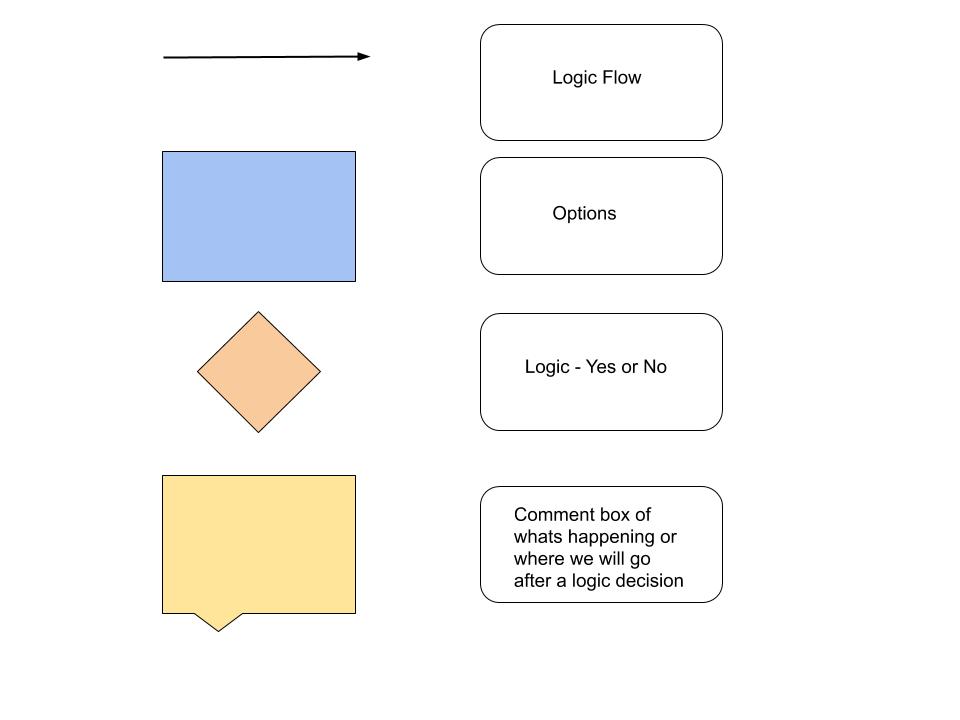
Our group did our best to keep to the engineering method Waterfall but were forced to go back phases and redo part of our requirements and analysis. Originally, we wanted to use a GPS and detect where the user was, so they didn’t have to do the searching, but we also wanted them to know where available parking spots were. So, we ended up going back to the requirements and analysis because we didn’t have a clear enough idea of what we needed to do.

Our group learned a lot using Waterfall and learned the importance of being able to re-evaluate things. Sometimes it takes you close to the end of the project to see that a type of desired implementation or a requirement may not be needed or necessary.

# **Decomposition Model**

# **Functional Decomposition Diagram**

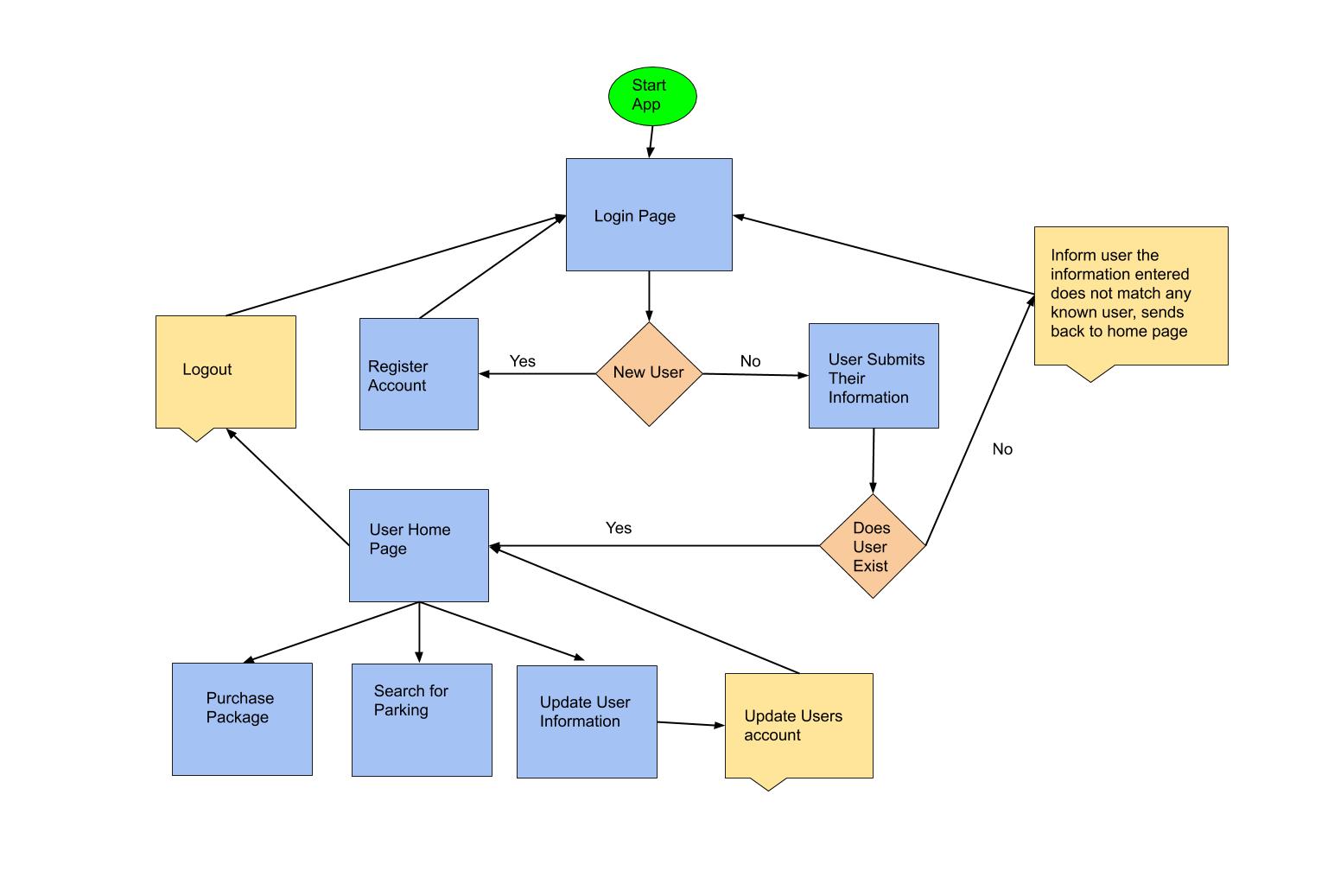
The decomposition process for the application Easy park was functional. Since we were using waterfall, it’s easy to break things down into functions or modules. For example, the function of the sign in screen will be to take input data and send it to the database. So, our sign in page function is to sign in a user or not, that is its function. We created our code in blocks to simplify how parts of the application interact.

******

***Figure 5.1 - Legend for Graphs***

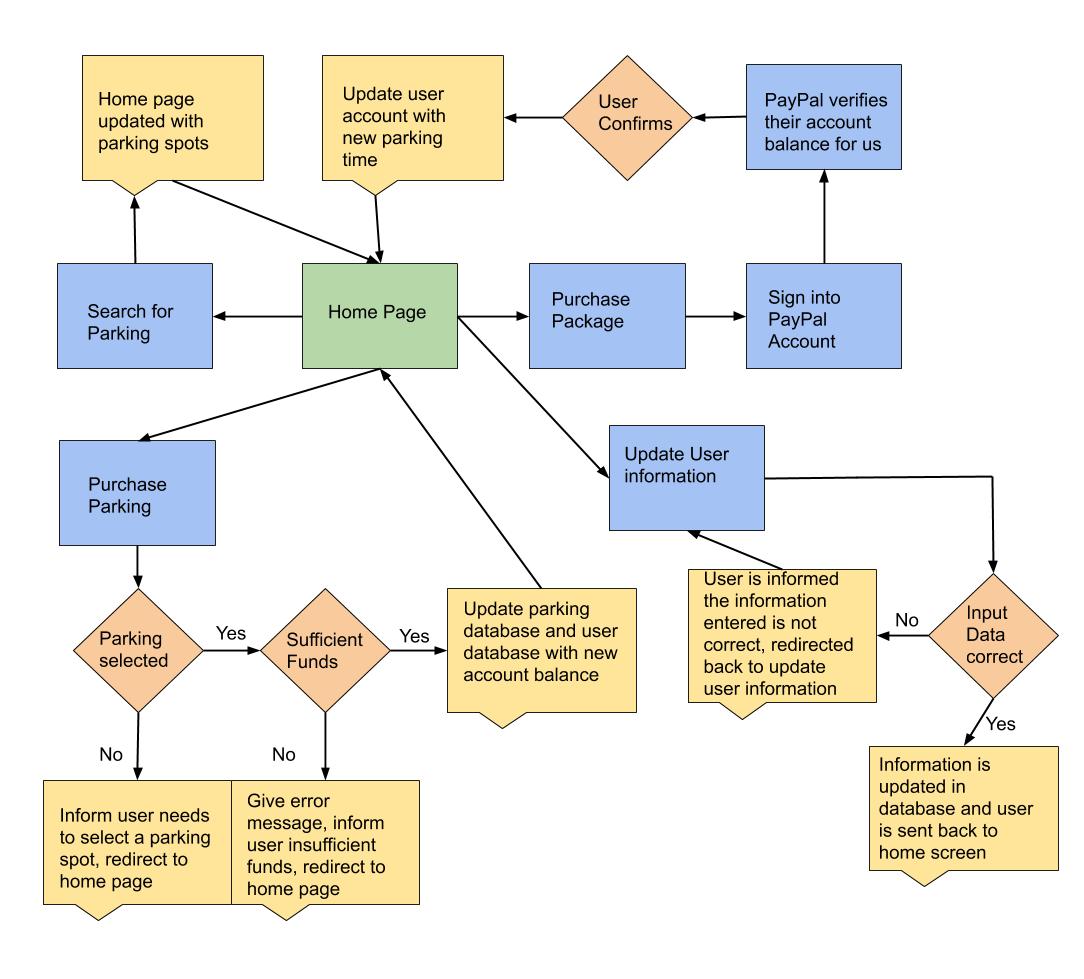
# **Login Decomposition Diagram**

Below we have broken up the diagram into two different diagrams for simplicity. The first diagram shows either a user signing in for the first time or a regular use signing in. There are four different types of boxes, the green is the start, blue is a page, orange is logic simulating our code processing user input, and yellow to comment on what is happening and then sending the user. This first diagram shows up to the user page and shows the options the user can then do. For simplicity, the add tag was put in this diagram because it was such a simple task. It is simply just put into the database.

***Figure 5.1.1 – Login Decomposition Functionality*** 

# **Logged-in Decomposition Diagram**

This second diagram shows what a user can do after they are signed in and the logic flow of those options. The color and shape of the boxes keep the same rules as before. The diagram shows the logic flow of purchasing a parking spot and checking if the users account has the correct number of tokens and checking if a parking spot is selected.

***Figure 5.1.2 – Logged in*** ***Decomposition Functionality***

# **Maintenance**

While Easy Park is being used, new issues may be found by either us or normal users. If any problem arises, we will fix the problem in the code and then release the new version as an update. This version of Easy Park that we just released is version 1.0, so likely there might be issues that we do not know about that could be found by extensive usage of the app.

In addition to finding issues, we also will update the app to have more parking lots and features in future updates for the app. We would like Easy Park to be accessible to as many people as possible and also as fully featured as possible.

# **Testing Plan**

Below we have designed a testing plan that builds after each test case. Test case one must be completed before moving on to further test cases. This is to prevent repeating steps for each case. Each test case will describe which test cases must have been completed before moving on to a next test case.

Before any testing can be fully completed the devices, camera needs to be authorized to be used by Easy Park.

**TC1**

**FR Tested:** FR1

**Required Test Case:** None

**Description:** User should be able to download the application from the google app store.

Steps:

1. Go to https://github.com/ahsan-wahab/Easy-Park
2. Select the .Apk file and download to your phone
3. Open the file and install
4. Follow the read me instructions to enable the permission for the camera

**TC2**

**FR Tested:** FR2, PR1

**Required Test Case:** TC1

**Description:** Once the user has the application on their device, they should be able to open the application and the first screen they go to should be the login page. The first time a user accesses the application they need to register an account. They need to go to register account button. Which takes them to a sign-up page. The user needs to put their information in and hit submit.

Steps:

1. Open Easy Park application on device
2. Click create account button
3. Enter all required boxes of information
4. Make sure that the phone number is 10 digits and the email address is correct
5. Click Sign up button

**TC3**

**FR Tested:** FR3, PR2

**Required Test Case:** TC2

**Description:** After a user has an account, they need to be able to sign in. The user is successfully able to login to their account in under 10 seconds and see the home screen if the email is correct and they have already signed up on the sign-up page. They get an error in case their email or password is incorrect. This action takes place in under 10 seconds.

Steps:

1. Open Easy Park application on device
2. Enter incorrect user name or password
3. Click login
4. Confirm no sign in
5. Now enter correct user name and password
6. Click login
7. Confirm login at home screen

**TC4**

**FR Tested:** FR3, PR1

**Required Test Case:** TC3

**Description:** The user should stay logged in after the application is closed or the user reopens the application.

Steps:

1. Start from already logged in
2. Go to device home screen
3. Find Easy park again and reopen
4. Confirm user is still signed in (No log in process required)

**TC5**

**FR Tested:** FR15, PR7

**Required Test Case:** TC3

**Description:** The user goes to the reset password page and enters his/her email address. Then, if the email address is correct the user will receive a reset password email within 30 minutes and will be able to reset the password. Do nothing if the email is incorrect.

Steps:

1. If logged in, log out
2. Go to Forgot password on login screen
3. Input your users email address
4. Click reset button
5. A email should be sent that email address

**TC6**

**FR Tested:** FR4, PR1

**Required Test Case:** TC3

**Description:** At the user’s home screen they should be able to go to settings and change their information such as phone, first and last name.

Steps:

1. Start from the user’s home page
2. Click settings button
3. Try to update nothing
4. User should be prompted saying no changes made
5. Make a small change
6. User should go back to home screen
7. Then go back to settings and ensure setting was changed

**TC7**

**FR Tested:** FR5, FR6, PR3

**Required Test Case:** TC3

**Description:** At the user’s home screen they should be able to go to type in a parking lot and see an array of available parking lots.

Steps:

1. Start from the user’s home page
2. Click in search bar up near top
3. Type “JpLot” as soon as you start typing it should appear underneath
4. Select it
5. An array of available parking spots should be right below the search bar

**TC8**

**FR Tested:** FR7, FR14, PR5

**Required Test Case:** TC7

**Description:** The user that is already signed in and tries to add and subtract more time. This test should not let the user go below 15 minutes, this is after test case 7, a parking spot and lot should be in the search bar.

Steps:

1. From user’s home page
2. Have a parking lot and spot selected in search bar
3. Click minus 15 minutes
4. A message should tell user not allowed to go under 15 minutes
5. Select plus 15 minutes the time should change to 30
6. Select minus the time should go back to 15 minutes

**TC9**

**FR Tested:** FR8, FR9, PR4, PR1

**Required Test Case:** TC7

**Description:** The user attempts to buy tokens all should take under 10 seconds for each screen to move to the next.

Steps:

1. From user’s home page
2. Click purchase tokens
3. Select a package
4. User should be taken to PayPal
5. After the user Confirms to pay, PayPal will tell our servers if it went through or not
6. A message should tell the user their purchase information if they have the money for the purchase

**TC10**

**FR Tested:** FR10, FR11, FR12, PR1, PR6

**Required Test Case:** TC7

**Description:** When a user wants to purchase a parking spot they hit confirm and the QR scanner comes up. The user then points the camera at the QR code. After the scanner successfully scans the code a confirmation button will appear, and the user confirms. This test confirms if the user has enough tokens.

Steps:

1. Start at user’s home page
2. User types in the time desired at least 15 minutes
3. Click Confirm button
4. QR code scanner should pop up (The device camera)
5. Points at QR code
6. Successful scan will ask the user to confirm
7. The user should be taken to home screen and time should be there for their parking

**TC11**

**FR Tested:** FR10, FR11, PR1, PR6

**Required Test Case:** TC10

**Description:** The user attempts to buy a parking space without enough tokens. The user should get a notification and not purchase parking spot.

Steps:

1. From user’s home page
2. Have a parking lot and spot selected in search bar
3. Click confirm
4. Scan QR code
5. A message should tell user they do not have enough tokens

**TC12**

**FR Tested:** FR12, FR13, PR8

**Required Test Case:** TC10

Description: After the time has rand out and can select the notification to be taken to home screen and choose to add time.

Steps:

1. Start at user home page, purchase parking for 15 minutes at any available parking spots
2. A permanent notification should be created counting down time
3. Wait for 15 minutes for the timer to end.
4. Click the notification
5. User should be taken to home screen
6. Add more time

**TC13**

**FR Tested:** FR13, FR14, PR8, PR5

**Required Test Case:** TC11

Description: If user’s current time has ended then more time is added, and the extra number of tokens is cut from the user’s account else a prompt generates stopping the user.

Steps:

1. Start at users come page
2. User types in the time desired greater than 15 minutes
3. Click Confirm button
4. QR code scanner should pop up (The device camera)
5. Points at QR code
6. The user should be prompted that they do not have enough tokens for the parking space.

**TC14**

**FR Tested:** FR16

**Required Test Case:** 10

Description: If the user has no booked spot now the application logs out successfully. Otherwise it tries to remain open until the user tries again and force closes it.

Steps:

1. Start at user home page, purchase parking for greater than 10 minutes (These steps are in TC11)
2. The user opens the app, should be taken to home screen
3. Add more time greater than 10 minutes
4. Click confirm
5. User should be prompted they do not have enough funds.
6. Taken back to home screen with time still below 5 minutes.

# **Future Implementations**

Future implementations of the application Easy park could be a better user interface and some added functionality. Additional functionality that would be desired to add would be to be able to purchase multiple parking spots at a time. Also, being able to add time before the timer runs out. These are just a few examples, but we think the base of the application is a good stepping stone to get a well-developed application that could compete with current parking lot apps.

# **Estimated Time and Cost**

Our estimations of about 8 hours a week for 12 weeks, each of us working on this project of our group of three people equals about 288 hours of working to develop the easy park application. If we use a conservative pay due to a lot of the time learning. So, at $25.00 an hour the total of this project would cost about $7,200 US dollars. Our original estimate to charge companies that wanted to use this application was $5,000 US dollars, then an additional $500 US dollars a month after that for maintenance. We are all happy about our original estimate being close to the after estimated cost. Our team was new to a lot of the software we used and middleware.

# **Change Log**

Here is own current change log that records our application changes from what we originally had. We will continuously keep this change log updated as changes arise and keep past changes as the process continues.

|  |  |  |
| --- | --- | --- |
| Original | Changes | Change Date |
| Use Google Maps API to determine if user is in certain parking lot | Removed the use of Google Maps API | 10/15/2019 |
| Test Case 10 in SRS | Removed due to not being part of the process anymore | 10/20/2019 |
| Test Case 14 in SRS | Changed due to typo. There was no Test Case 13 | 10/20/2019 |
| Test Case 13 in SRS | Removed because Test Case 12 already covers it | 10/20/2019 |
| None | Added Test Case 15, needed more to test application logic | 10/20/2019 |
| None | Added Test Case 16, needed more to test application logic | 10/20/2019 |
| None | Added a text box to show the time remaining in the home screen. | 10/12/2019 |
| The home screen had the location confirmation feature. | The location confirmation feature has been removed due to the complexities we faced while implementing it. | 10/15/2019 |
| The home screen had a confirm button to calculate the cost of parking | The confirm button has been removed and the cost calculation is in real time. | 10/15/2019 |
| User must pay for at least 10 minutes | Changed to 15, with a maximum of 120 minutes | 10/25/2019 |
| Test case 7 in design document: user tries to pay for less than 10 minutes | Changed to 15 | 10/25/2019 |
| Add license plate requirement | Removed, not saving license plate data anymore | 11/01/2019 |
| Test case 1 in design document: user must be able to download from Google Play Store | Changed to GitHub | 11/30/2019 |
| Functional Requirement 13 in design document: notification will be shown 5 minutes before time runs out | Changed to be a persistent notification showing time left | 10/30/2019 |
| Test Case 7 in design document: notification will show with 5 minutes left and user can add at least 10 minutes after finished | Changed to be a persistent notification and when time is finished, user must add at least 15 minutes | 10/30/2019 |