# Where2P4Free

Bachelor of Science (Honours) Software Development

**Name:** Patrick Browne

**ID:** C00190601

**Year:** 4th Year

**Supervisor:** Paul Barry

**Due date:** 12-04-2019

**Design Manual**

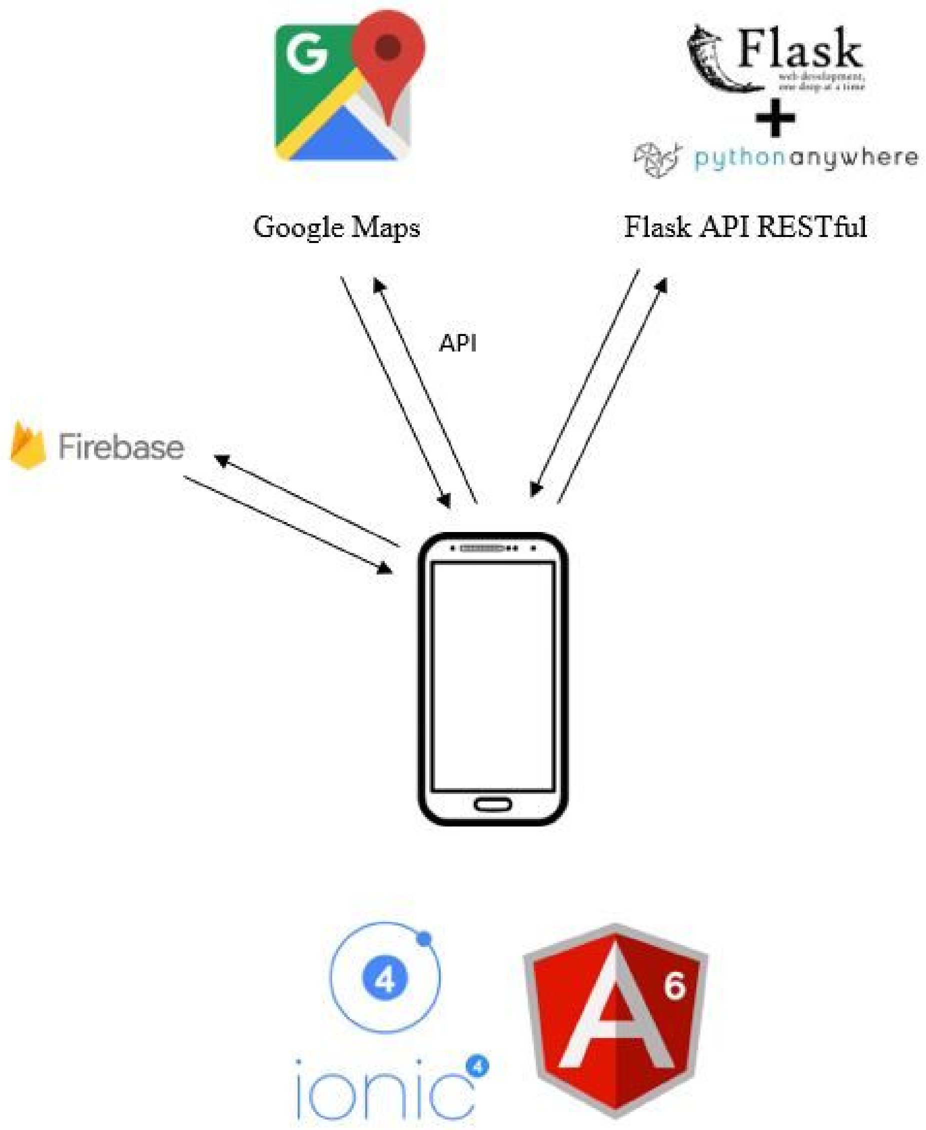
# 1. Table of Contents

# 1. Introduction

The purpose of this document is to outline the technologies which will be chosen for the project. The decisions made with regards to choosing said technologies will be based on research conducted on the development of a single codebase for a mobile application. Design of the database to store the data will be decided upon, and the flow of the application from a user and UI perspective. Details use cases will also be given for the different main events that will take place upon user interaction with the application.

The overall architecture of the system is detailed in-depth under the following: Python API, Firebase and Ionic 3 with Angular 4 for the Mobile Application. Database entity relationships and schemas along with description and examples of the API GET and POST requests. The proposed UI layout will be shown with navigation flow. Application use case diagrams for the Mobile Application and Database Admin Client including detailed use cases will be included.

## 2. System Architecture



As shown in the above diagram Mobile Application will connect to a Python database server-side through a RESTful API to process data through the cloud and display it onscreen according to the user's interaction.  Data sent from the application such as a user's review of a toilet or adding a toilet to the Mobile Application can also be sent via the API back to the server. Google Maps API will provide GPS map location and Firebase will handle registration/login requests.

# 3. Database Design

## 3.2 Tables

### 3.2.1 Table structure for ‘toilet’ table

The ‘toilet’ table is used to store data on nearby toilets.  The data being stored will be under the following columns: toiletID, location, description, type.

- toiletID - This column stores the unique ID used to identify each toilet.  The column also joins the toilet table with the review table.  The toiletID is the foreign key in the review table.
- location - This column stores the location of a toilet.
- type - Type column will store the type of accessibilities of the toilet (Male, Female, Disabled access, etc).

### 3.2.2 Table structure for ‘user’ table

The ‘user’ table is used to store data on users interacting with the system.  The data being stored will be under the following columns: userID, username, password.

- userID - This column stores the unique ID used to identify each toilet.  It also joins the user’s table with the review table.  It is the primary key in the table whilst being the foreign key in the reviews table.
- username - This column stores the username of the user.
- password - This column stores the password of the user.

### 3.2.3 Table structure for ‘review’ table

The ‘review’ table is used to store data on reviews made by users to toilets they have visited. The data being stored will be under the following columns: toiletID, userID, rating, comment.

- toiletID - This column joins the review and user's tables.  The toiletID is the foreign key in the review table and is the primary key in the toilet table
- userID - This column joins the review table with the user table.  The userID is the foreign key in the table.
- comment - The comment columns stores a comment left by the user about a particular toilet.

# 4. API Endpoints

## 4.1 findLocation

```
@app.route('/api/findLocation', methods=['GET'])
def retreive_location():
    cnx = mysql.connector.connect(user="where2p4free", password="restroom",
host="where2p4free.mysql.pythonanywhere-services.com",
database="where2p4free$where2p4free")
    cursor = cnx.cursor(buffered=True)

    cursor.execute("SELECT toiletID, location, description, type, latitude, longitude FROM toilet")
    cnx.commit()

    data = cursor.fetchall()

    d = []

    for location in data:
        d.append({'toiletID': location[0],
            'location': location[1],
            'description': location[2],
            'type': location[3],
            'latitude': location[4],
            'longitude': location[5]})
    cursor.close()
    cnx.close()

    return jsonify(d)
```

## 4.2 addLocation

```
@app.route('/api/addLocation', methods=['POST'])
def insert_location():
    cnx = mysql.connector.connect(user="where2p4free", password="restroom",
host="where2p4free.mysql.pythonanywhere-services.com",
database="where2p4free$where2p4free")
    cursor = cnx.cursor()

    data = request.get_json()
    # print(data)

    sql = "INSERT INTO toilet (location, description, type, latitude, longitude) VALUES (%s, %s,
%s, %s, %s)"
    cursor.execute(sql, (data['location'], data['description'], data['type'], data['lat'], data['lng']))
    cnx.commit()

    cursor.close()
    cnx.close()

    return 'Location added!'
```

## 4.3 updateLocation

```
@app.route('/api/updateLocation', methods=['POST'])
def update_location():
    cnx = mysql.connector.connect(user="where2p4free", password="restroom",
host="where2p4free.mysql.pythonanywhere-services.com",
database="where2p4free$where2p4free")
    cursor = cnx.cursor(buffered=True)

    data = request.get_json()

    sql = "SELECT toiletID, updateThreshold FROM toilet WHERE toiletID = %s"
    cursor.execute(sql % data['toiletID'])
    cnx.commit()

    values = cursor.fetchall()
    threshold = values[0][1]
    if threshold < updateThreshold:
        sql = "UPDATE toilet SET updateThreshold = updateThreshold + 1 WHERE toiletID = %s"
        cursor.execute(sql % data['toiletID'])
        cnx.commit()
        s = 'Location cannot yet be update!'
    else:
        sql = "UPDATE toilet SET location = %s, description = %s, type = %s, latitude = %s,
longitude = %s, updateThreshold = 0 WHERE toiletID = %s"
        cursor.execute(sql, (data['location'], data['description'], data['type'], data['latitude'],
data['longitude'], data['toiletID']))
        cnx.commit()
        s = 'Location updated!'

    cursor.close()
    cnx.close()

    return s
```

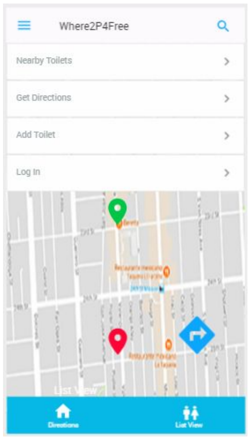
## 4.4 removeLocation

```
@app.route('/api/removeLocation', methods=['POST'])
def remove_location():
    cnx = mysql.connector.connect(user="where2p4free", password="restroom",
host="where2p4free.mysql.pythonanywhere-services.com",
database="where2p4free$where2p4free")
    cursor = cnx.cursor(buffered=True)

    data = request.get_json()

    sql = "SELECT toiletID, deleteThreshold FROM toilet WHERE toiletID = %s"
    cursor.execute(sql % data['toiletID'])
    cnx.commit()

    values = cursor.fetchall()
    threshold = values[0][1]

    if threshold < removeThreshold:
        sql = "UPDATE toilet SET deleteThreshold = deleteThreshold + 1 WHERE toiletID = %s"
        cursor.execute(sql % data['toiletID'])
        cnx.commit()
        s = 'Location cannot yet be removed!'
    else:
```

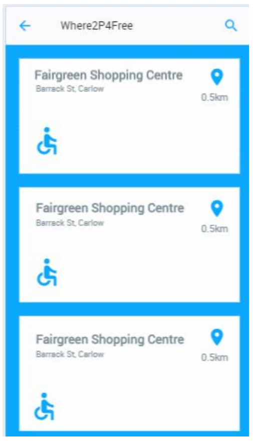
```
    sql = "DELETE FROM toilet WHERE toiletID = %s"
    cursor.execute(sql % data['toiletID'])
    cnx.commit()
    s = 'Location removed!'

cursor.close()
cnx.close()

return s
```

## 5. UI Layout

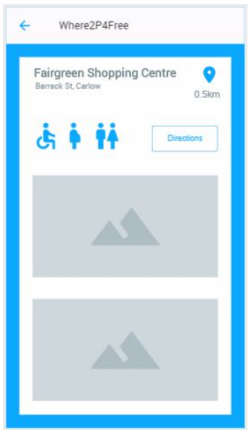
Slide in Menu



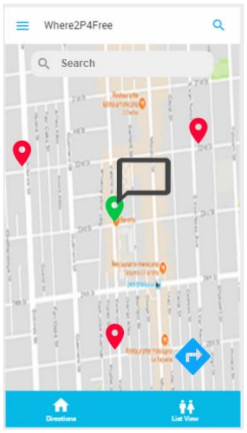
List of Toilets



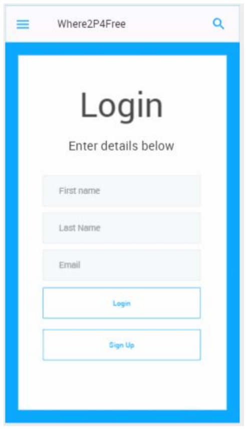
Toilet Details



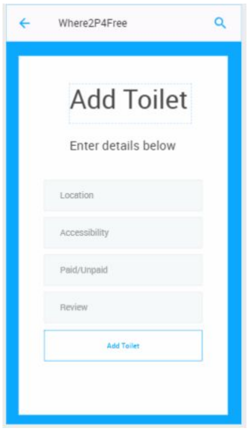
Home Screen



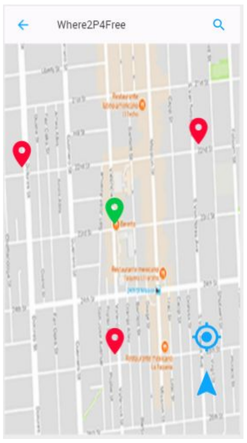
Login



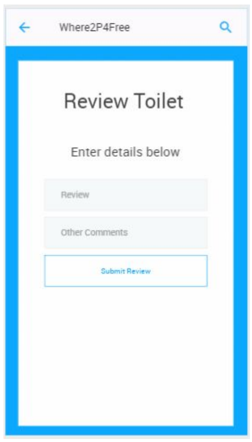
Add Toilet



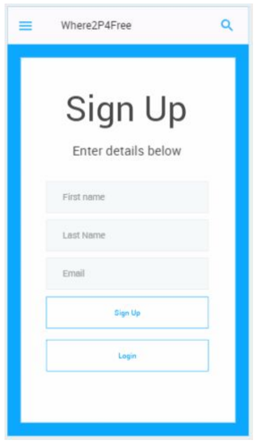
Directions



Review Toilet

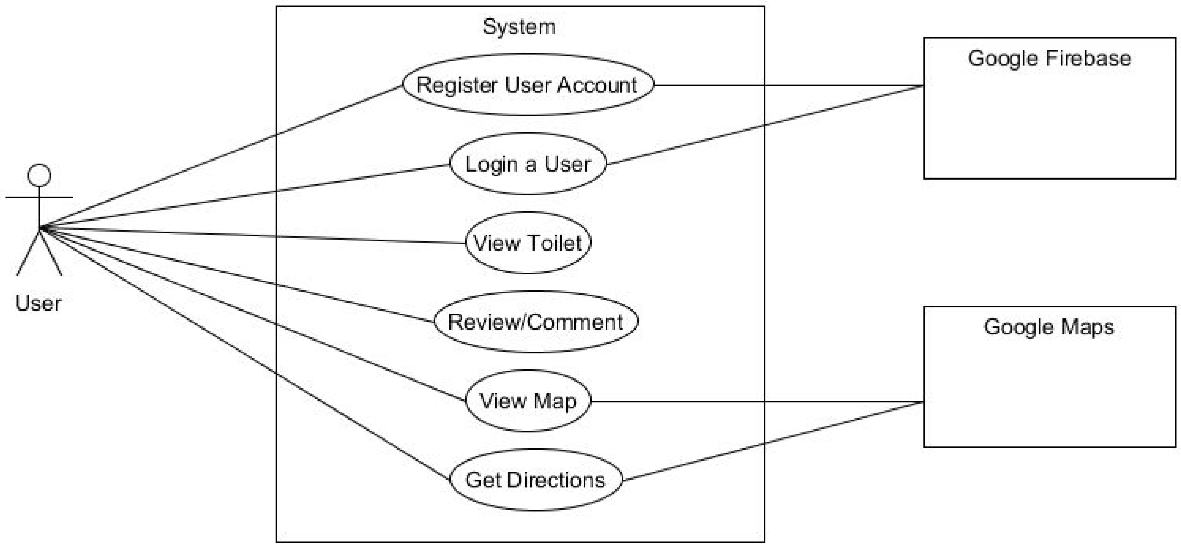


Sign Up

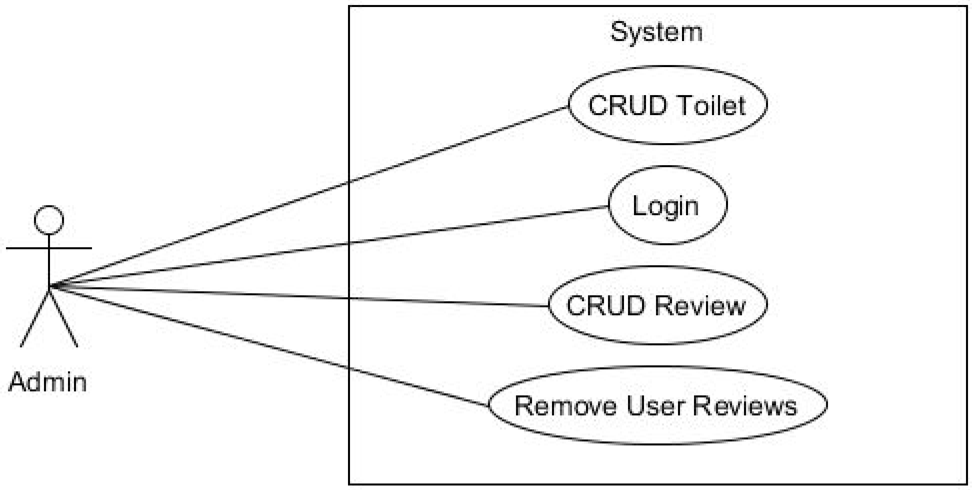


## 6. Use Case Diagrams

Mobile Application

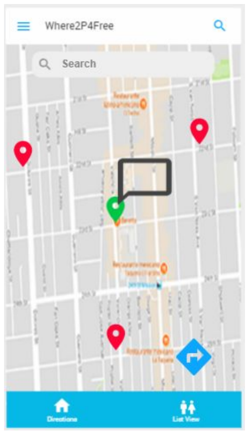


Web Admin



# 7. Detailed Use Cases

## 7.1 View Map



**Actors:**
User, Mobile Application, Database, Google Map, GPS, Toilet

**Preconditions:**
The user has access to the internet and is currently viewing a toilet.

**Description:**
This use case beings when the User wishes to view the location of a Toilet on a map. The User selects the view location icon displayed on-screen. The Mobile Application gets the GPS coordinates of the Toilet from the Database before displaying a Google Map with the pinpoint location of the Toilet on-screen.

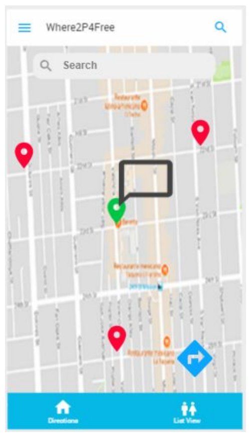
**Main Success Scenario:**
1. The User clicks the direction button displayed on-screen.
2. The Mobile Application requests the GPS coordinates from the Database.
3. The Database receives the request and returns the GPS coordinates.
4. The Mobile Application displays a Google Map with the location.

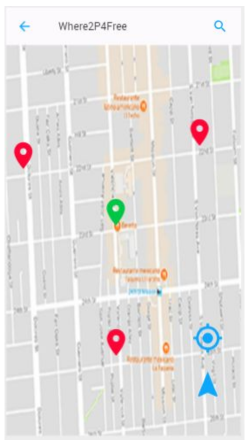
**Alternatives:**

2. The Mobile Application fails to send the request to the Database.
    a. The User is informed via an error message.
    b. Repeat step 1.

3. The Database fails to return the GPS coordinates to the Mobile Application.
    a. An error message is displayed on-screen for the User.
    b. The User is instructed to attempt the action again.
    c. Repeat step 1.

## 7.2 Get Directions





**Actors:**
User, Mobile Application, Google Maps, Toilet

**Preconditions:**
The user has access to the internet and is viewing a Toilet page.

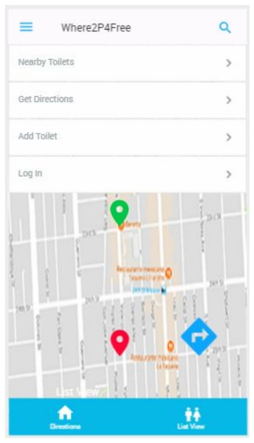
**Description:**
This use case begins when the user wishes to get directions to a Toilet that they are currently viewing. The User selects the get direction icon displayed on-screen. The Mobile Application obtains the User's current GPS and Toilet's GPS coordinates before passing it to Google Maps. Google Maps displays the directions to the Toilet.

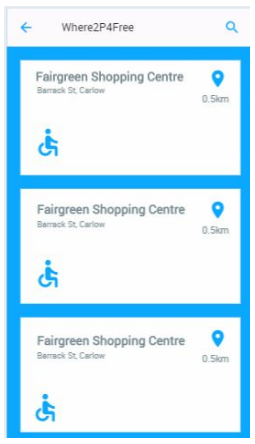
**Main Success Scenario:**
1. The user presses the get direction icon.
2. The Mobile Application records the GPS location of the User.
3. The GPS location of the Toilet is also recorded.
4. Google Maps receives the GPS coordinates.
5. The directions to the Toilet are displayed on-screen.

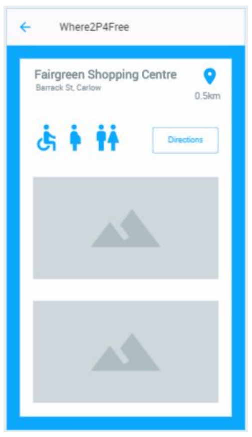
**Alternatives:**

1. The Mobile Application fails to obtain the GPS coordinates.
    a. The User is informed that they require internet access.
    b. Repeat step 1.
2. Google Maps encounters an issue parsing the coordinates.
    a. An error is displayed on-screen by the Mobile Application.
    b. The User is instructed to attempt the action again.
    c. Repeat step 1.

## 7.3 Viewing a Toilet Page







**Actors:**
User, Mobile Application, Database, Toilet

**Preconditions:**
The User has navigated to the Toilet page from the main menu.

**Description:**
This use case begins when the User wishes to view information about a Toilet. The User selects the Toilet from a list displayed on-screen. The Mobile Application requests the information regarding that Toilet from the Database; photo, information text, location, and other user comments. The Database sends the relevant information to the Mobile Application where it is then displayed on-screen for the User.

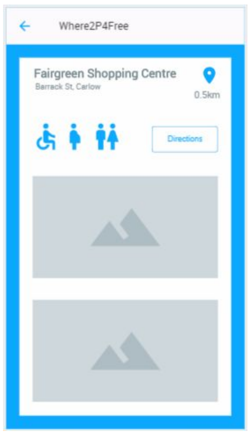
**Main Success Scenario:**
1. The user successfully navigates to the Toilet page.
2. The Mobile Application requests the information from the Database.
3. The Database receives the request and returns the information.

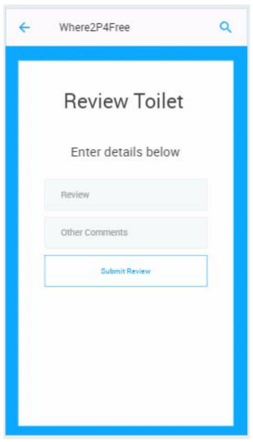
4. The Mobile Application displays the information on-screen.

**Alternatives:**

2. The Mobile Application fails to send the request to the Database.
    a. The User is informed via an error message.
    b. Inform the User to reload the page.
    c. Repeat step 1.
3. The Database fails to interpret the request.
    a. An error message is displayed to the User.
    b. The User is instructed to reload the page.
    c. Repeat step 1.

## 7.4 Review/Comment





**Actors:**
User, Mobile Application, Database, Toilet

**Preconditions:**
The User has access to the internet and has logged into the Mobile Application.

**Description:**
This use case begins when the User wishes to review/comment on a toilet. The User selects the rate/review button on the screen and the Mobile Application displays a form on-screen. The User enters their comment and then submits the form. The Mobile Application sends the User comment to the database where it is then stored.

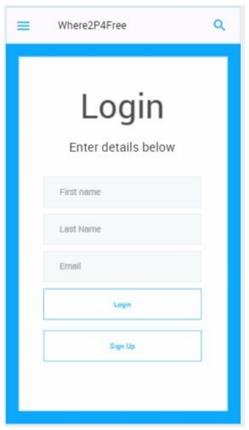
**Main Success Scenarios:**

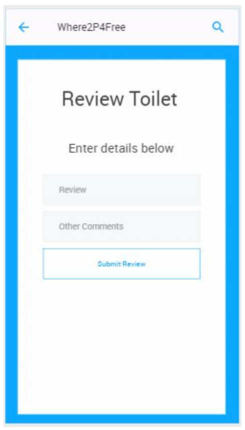
1. The User successfully navigates to the correct screen.
2. The Mobile Application receives the request from the user and displays the form.
3. The User then enters relevant information and submits the form.
4. The Mobile Application receives the information and sends it to the Database.
5. The Database accepts the request and stores the information into a table.

**Alternatives:**

2. The Mobile Application fails to connect to the Database.
    a. The Mobile Application attempts to re-establish a connection to the Database.
    b. Display an error message on-screen for the User.
    c. Go back to step 3.
3. The User incorrectly enters the information.
    a. The Mobile Application displays an error message on-screen.
    b. The User is instructed to re-enter the information correctly.
    c. Repeat step 3.
5. The Database encounters an issue storing the information.
    a. The Mobile Application displays an error message on-screen.
    b. The User is instructed to try again.
    c. Repeat step 3.

## 7.5 Login a User





**Actors:**
User, Mobile Application, Database

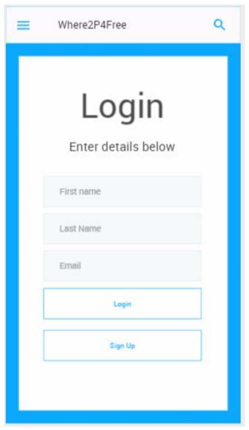
**Preconditions:**
The user has access to the internet and is currently viewing the login form.

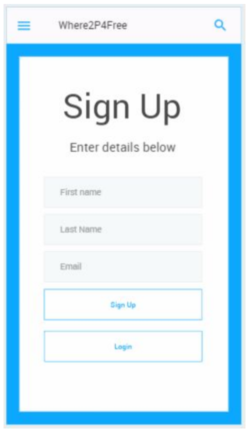
**Description:**
This use case begins when the User wishes to log in to their account. The User enters their details; email address, password, before pressing the login button. The Mobile Application sends the User's details to the Database where it is compared with the existing details. If the details match what is stored in the Database, the User is allowed access. Otherwise, the Mobile Application displays an error message on-screen for the User.

**Main Success Scenario:**
1. The User correctly enters all their details.
2. The Mobile Application sends the information to the Database.
3. The Database receives the request containing the information.
4. The information is compared to what is stored in the Database.
5. The User is given access to the Mobile Application.
6. The Mobile Application tells the User to re-enter their credentials.

**Alternatives:**
1. The User incorrectly enters their details.
    a. The Mobile Application displays an error message on-screen.
    b. The User is told to re-enter their credentials.
    c. Repeat step 1.
2. The Mobile Application fails to send the information to the Database.
    a. The Mobile Application informs the User via an error message.
    b. The User is told to try and re-submit the information.
    c. Repeat step 2.
4. The Database unsuccessfully compares the information.
    a. The User is informed by an error message.
    b. The Mobile Application tells the User to try again later.
    c. Go back to step 1.

## 7.6 Register a User Account





**Actors:**
User, Mobile Application, Database

**Preconditions:**
The user has access to the internet and is currently viewing the login form and does not wish to use a Google or Facebook account.

**Description:**
This use case begins when the User wishes to sign up for a user account. The User selects the registration option displayed on-screen. The Mobile Application displays a user account registration form. The User enters their details; email address, password, before pressing the submit button. The Mobile Application sends the User's details to the Database where it is then stored.

**Main Sucess Scenario:**
1. The User successfully navigates to the signup form.
2. The Mobile Application displays a form on-screen.
3. The User enters all of their details correctly.
4. The Mobile Application sends the information to the Database.
5. The Database receives the request and stores the information in a table.

**Alternatives:**
3. The User enters their details incorrectly.
    a. The Mobile Application shows an error message to the User.
    b. Repeat step 3.
4. The Mobile Application fails to send the information to the Database.

a. An error message is shown to the User.
b. The User is instructed re-submit their details.
c. Repeat step 4.

## 8. Conclusion

Having performed research in previous documentation, the framework of choice will be Ionic 4. This is the current version of the framework. The remote host for the API will be PythonAnywhere. Python is the language of choice for developing the API in. The platform that the app will target will be Android, as it is the largest mobile operating system currently.