COMP 3504 - Fall 2018

Assignment #3
Campus Navigation App

Team #1

Connor Beauchamp

Marc Francois

Maxwell Tyson

December 3, 2018

Table of Contents

Deliverable #1

(1-1) Software Process Model

We have decided to choose Scrum as our process model. There are many reasons why we thought Scrum would be the best suited methodology for us. To start, our clients are not entirely clear on what their requirements are. We chose to focus more on regular meetings with our clients and also creating prototypes to help ensure that any requirement changes can be dealt with efficiently. Implementing the shorter sprints involved with Scrum will also ensure our clients end up with a final product that provides value to them. We believe that having our stakeholders visibly see a working product as we progress through these sprints will dissolve all discrepancies between what they actually want and the initial requirements provided. Since our team is small, we have the ability to communicate effectively with daily stand-ups and throughout the planning phases of each sprint. To summarize, communicating frequently and building working prototypes in 1-4-we intervals will give us the best opportunity to deliver a successful product come December.

Another reason why we chose an agile methodology, instead of Spiral or Waterfall, is because we are all new to developing applications and this type of methodology allows for a high degree of flexibility. The short sprints will give us an idea of what is possible for us to accomplish on an ongoing basis, rather than running into critical issues during the single implementation phase when using a more linear design approach. Our team is fortunate to have on-site customers, who will not only provide an effective communication bridge between client and customer, but also help further define the currently vague requirements as we progress.

(1-2) Requirements

Functional Requirements

Must display the user's current location

- GPS services will be used to find the location of the user's phone
- The location will be displayed on the map

Student must be able to lookup a location and display it on the map

- Students can input a classroom ID (ex. EB202) that can be queried by the system
- Points of interest and parking lots can also be searched

Must store the classroom ID's entered by the user, in the user history section.

- Searches must be stored in the history section of a user's account so the user can re-select a previous location ID

Must be able to store selected locations as Favorites

- After being searched, the user can add a location to their favorites for quick access

User must be able to search for a location via the Explore tab

- Instead of typing a location, the user can tap the Explore tab to display a list of locations
- It will display three sections: POIs, Hallways, and Parking Lots

Non-Functional Requirements

Must handle large user loads

- Application can handle thousands of students at a time. Eventually this will need to go up with adoption to other schools.

Must have high availability

- Application must be up 99% of the time.

User-Map Interactivity

- Users must be able to interact with the campus maps with functions such as zoomability through pinching, and centering of current location.

Simplistic & User friendly

- Users can quickly open the app and find/be directed to a classroom without any previous knowledge of the application.

High memorability

- Users can come back to the application after a lengthy time period and remember how to use it.

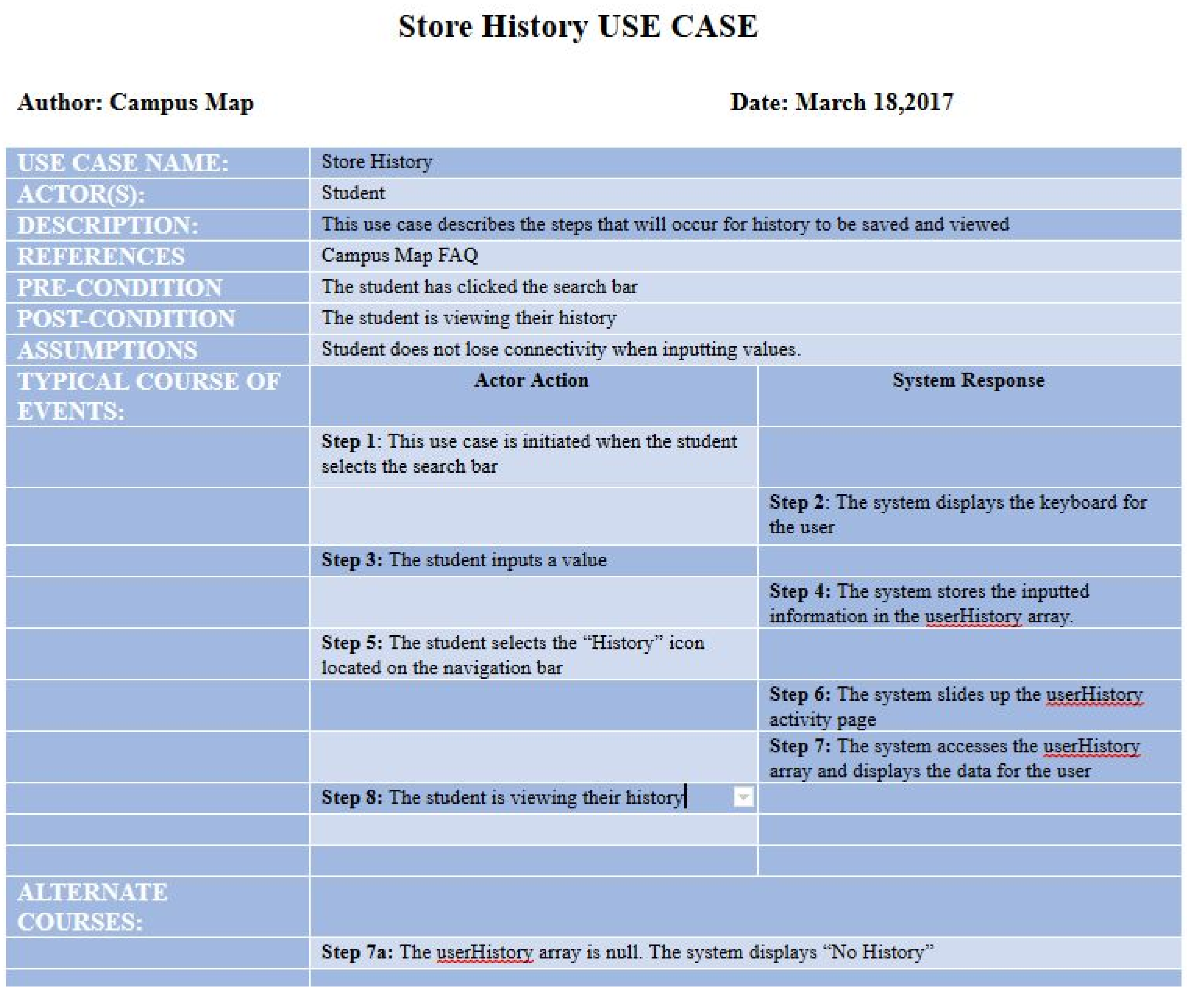
Deliverable #2

(2-1) Justification of Choosing Use Cases

We decided to choose Use Cases, and a Use Case Diagram for multiple reasons. A main reason is that our team has experience with representing system requirements through this format from both this course and courses taken in previous years. We have learned that use cases are a more descriptive format, and because of this we feel that the documenting of important details through use case narratives will better aide us in the development process. Since user stories tend focus more on what the user is trying to do, and use cases have a clearer documentation format, we feel this gives us even more reason to choose this method.

Use cases will force us to list the set of interactions between the users and the system in a clear and concise manner and communicating these with the team throughout the planning phases can spark important discussions on how the application should be functioning, and what is feasible in the time frame given. These user interactions, created from the business rules gathered, will help guide not only our logic, but also our workflows when it comes time to design and implementation phases. Use cases ensure that our documentation is thorough and consistent throughout each sprint and we feel this will help keep us organized and make sure high priority tasks are reaching completion.

(2-2) Three Use Cases



Store History USE CASE

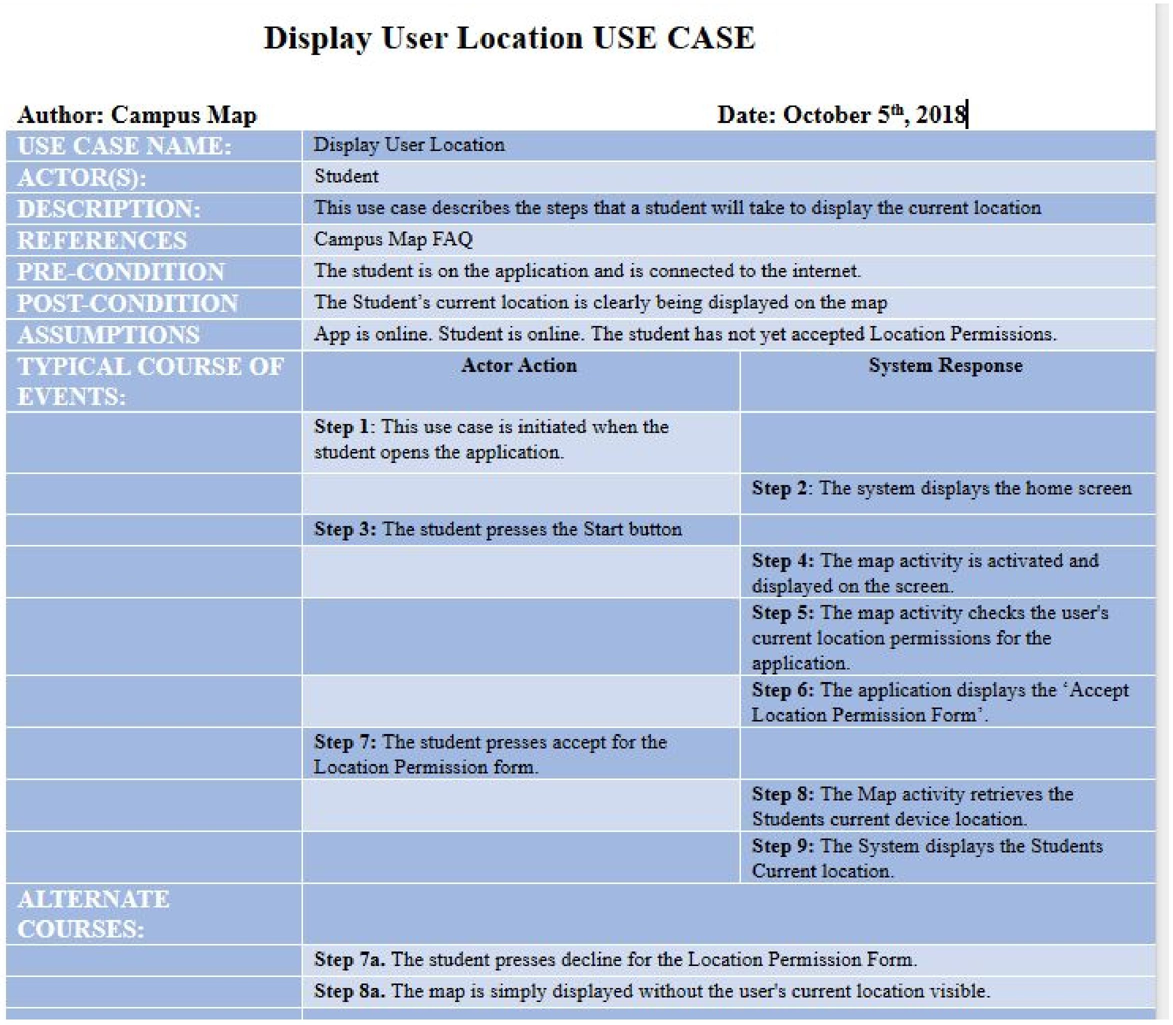
Author: Campus Map **Date: March 18,2017**

USE CASE NAME:	Store History	
ACTOR(S):	Student	
DESCRIPTION:	This use case describes the steps that will occur for history to be saved and viewed	
REFERENCES	Campus Map FAQ	
PRE-CONDITION	The student has clicked the search bar	
POST-CONDITION	The student is viewing their history	
ASSUMPTIONS	Student does not lose connectivity when inputting values.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: This use case is initiated when the student selects the search bar	
		Step 2: The system displays the keyboard for the user
	Step 3: The student inputs a value	
		Step 4: The system stores the inputted information in the userHistory array.
	Step 5: The student selects the "History" icon located on the navigation bar	
		Step 6: The system slides up the userHistory activity page
		Step 7: The system accesses the userHistory array and displays the data for the user
	Step 8: The student is viewing their history	
ALTERNATE COURSES:		
	Step 7a: The userHistory array is null. The system displays "No History"	

Look Up Hallway USE CASE

Author: Campus Map **Date: October 5th, 2018**

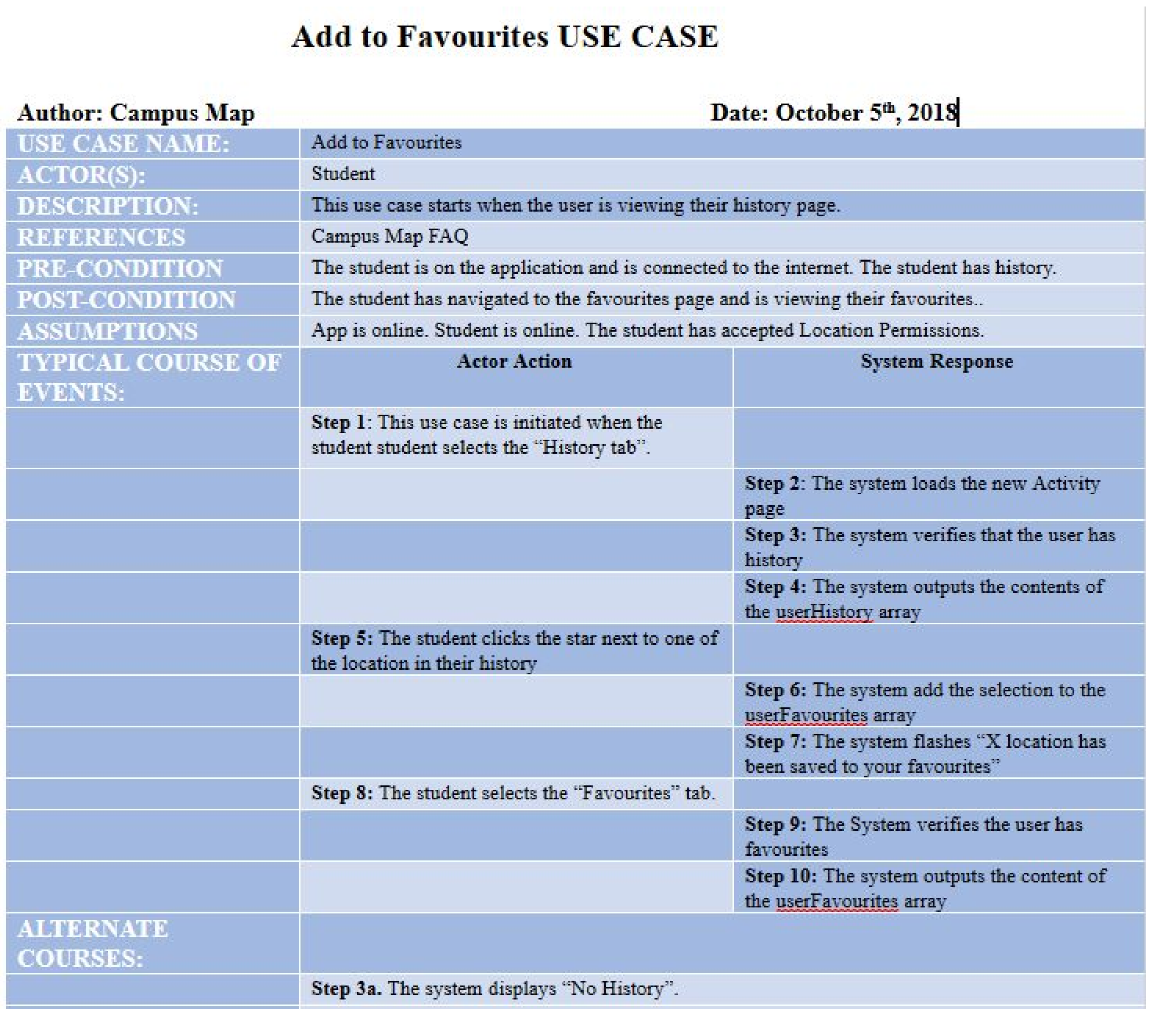
USE CASE NAME:	Search Hallway	
ACTOR(S):	Student	
DESCRIPTION:	This use case describes the steps that a student takes search a hallway	
REFERENCES	Campus Map FAQ	
PRE-CONDITION	The student is on the application and is connected to the internet.	
POST-CONDITION	The location is shown on the map	
ASSUMPTIONS	App is online. Student is online.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: This use case is initiated when the student selects the search bar located at the top of the application	
		Step 2: The system displays the keyboard
	Step 3: The student inputs a hallway and hits enter	
		Step 4: The system displays the location on the map marked with a red pin
ALTERNATE COURSES:		
	Step 3a: The student inputs a POI **Step 3b:** The student enters a parking lot	
	Step 4a: The system does not find a result and the search bar is underlined with red	



Display User Location USE CASE

Author: Campus Map **Date: October 5th, 2018**

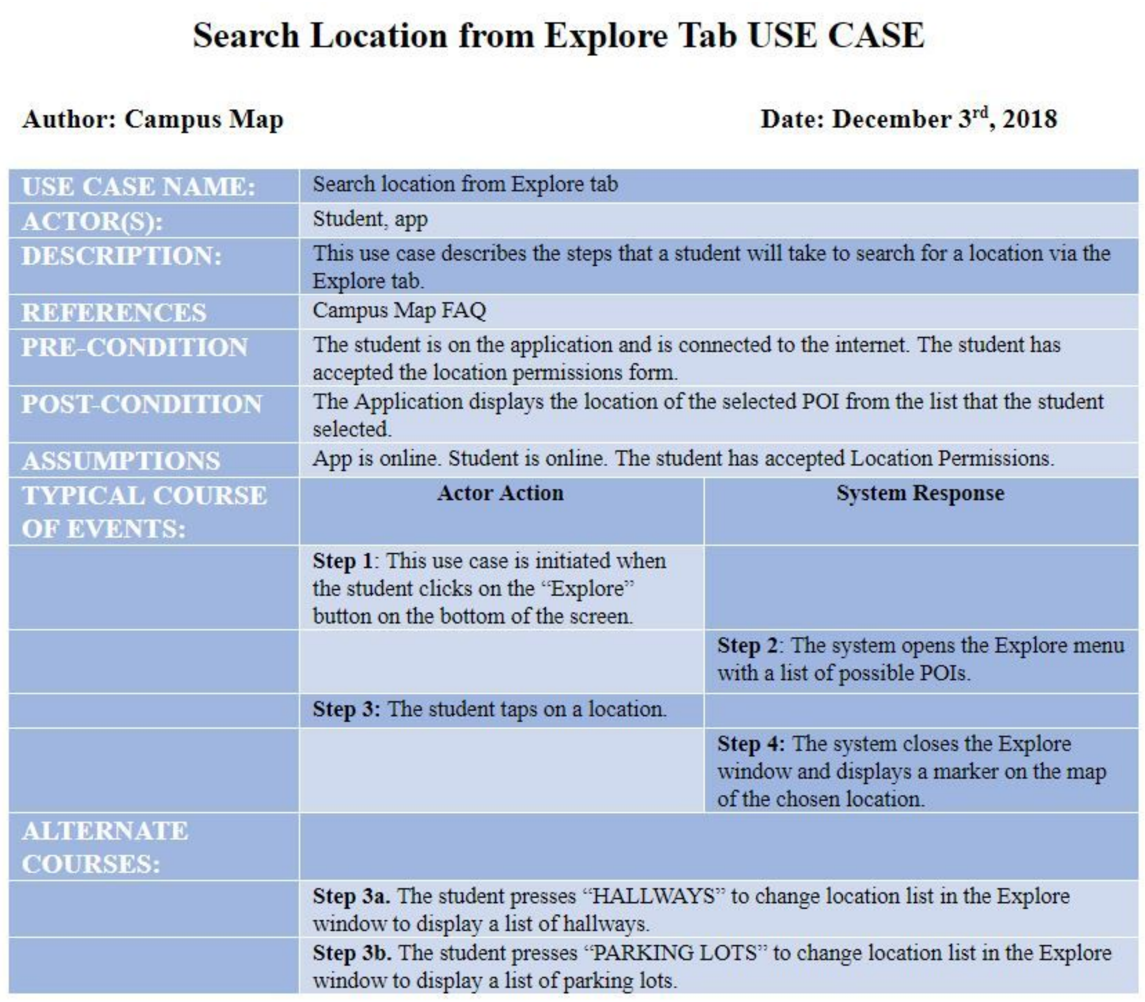
USE CASE NAME:	Display User Location	
ACTOR(S):	Student	
DESCRIPTION:	This use case describes the steps that a student will take to display the current location	
REFERENCES	Campus Map FAQ	
PRE-CONDITION	The student is on the application and is connected to the internet.	
POST-CONDITION	The Student's current location is clearly being displayed on the map	
ASSUMPTIONS	App is online. Student is online. The student has not yet accepted Location Permissions.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: This use case is initiated when the student opens the application.	
		Step 2: The system displays the home screen
	Step 3: The student presses the Start button	
		Step 4: The map activity is activated and displayed on the screen.
		Step 5: The map activity checks the user's current location permissions for the application.
		Step 6: The application displays the 'Accept Location Permission Form'.
	Step 7: The student presses accept for the Location Permission form.	
		Step 8: The Map activity retrieves the Students current device location.
		Step 9: The System displays the Students Current location.
ALTERNATE COURSES:		
	Step 7a. The student presses decline for the Location Permission Form.	
	Step 8a. The map is simply displayed without the user's current location visible.	



Add to Favourites USE CASE

Author: Campus Map **Date: October 5th, 2018**

USE CASE NAME:	Add to Favourites	
ACTOR(S):	Student	
DESCRIPTION:	This use case starts when the user is viewing their history page.	
REFERENCES	Campus Map FAQ	
PRE-CONDITION	The student is on the application and is connected to the internet. The student has history.	
POST-CONDITION	The student has navigated to the favourites page and is viewing their favourites..	
ASSUMPTIONS	App is online. Student is online. The student has accepted Location Permissions.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: This use case is initiated when the student student selects the "History tab".	
		Step 2: The system loads the new Activity page
		Step 3: The system verifies that the user has history
		Step 4: The system outputs the contents of the userHistory array
	Step 5: The student clicks the star next to one of the location in their history	
		Step 6: The system add the selection to the userFavourites array
		Step 7: The system flashes "X location has been saved to your favourites"
	Step 8: The student selects the "Favourites" tab.	
		Step 9: The System verifies the user has favourites
		Step 10: The system outputs the content of the userFavourites array
ALTERNATE COURSES:		
	Step 3a. The system displays "No History".	

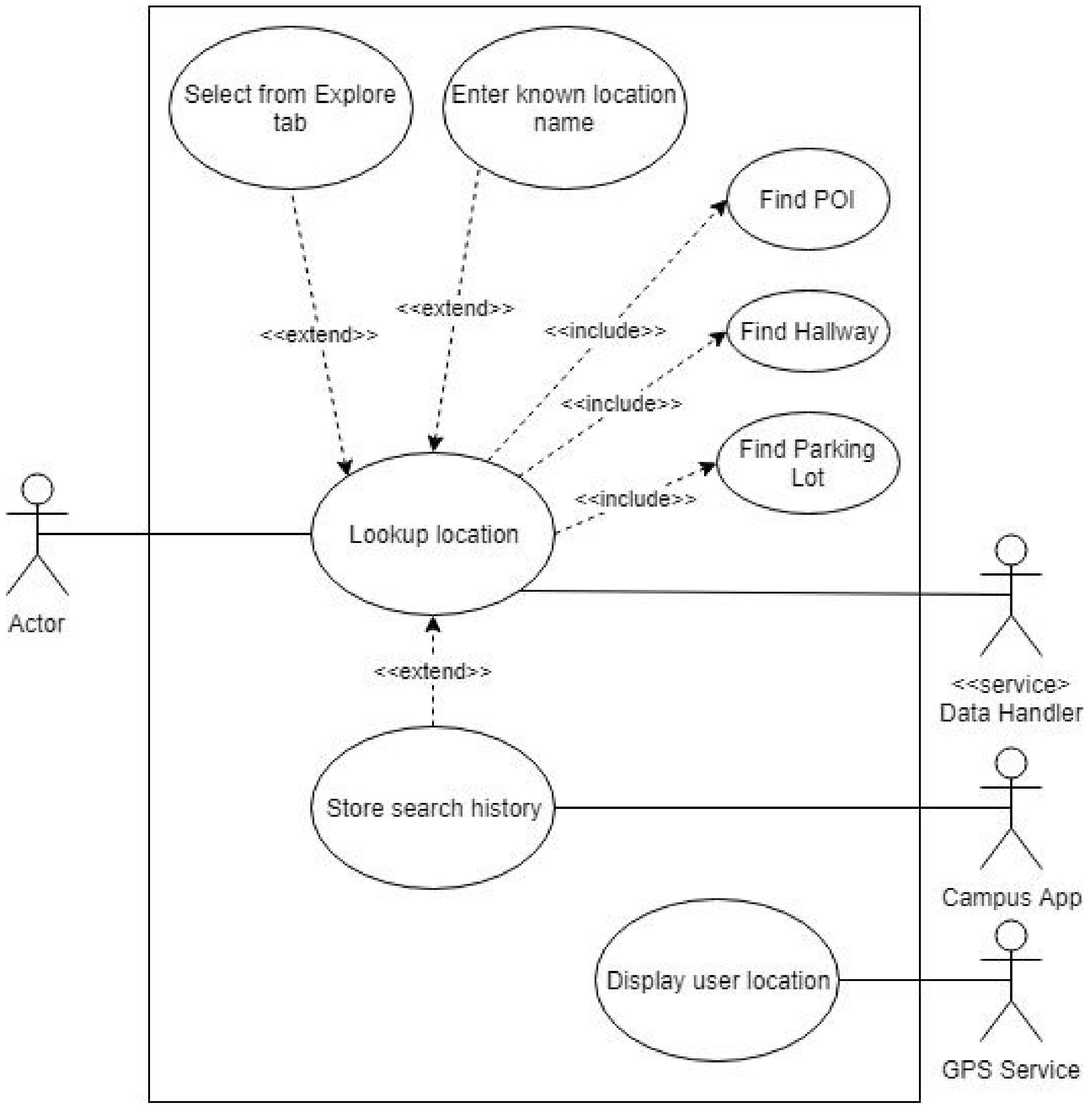


Search Location from Explore Tab USE CASE

Author: Campus Map **Date: December 3rd, 2018**

USE CASE NAME:	Search location from Explore tab	
ACTOR(S):	Student, app	
DESCRIPTION:	This use case describes the steps that a student will take to search for a location via the Explore tab.	
REFERENCES	Campus Map FAQ	
PRE-CONDITION	The student is on the application and is connected to the internet. The student has accepted the location permissions form.	
POST-CONDITION	The Application displays the location of the selected POI from the list that the student selected.	
ASSUMPTIONS	App is online. Student is online. The student has accepted Location Permissions.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: This use case is initiated when the student clicks on the "Explore" button on the bottom of the screen.	
		Step 2: The system opens the Explore menu with a list of possible POIs.
	Step 3: The student taps on a location.	
		Step 4: The system closes the Explore window and displays a marker on the map of the chosen location.
ALTERNATE COURSES:		
	Step 3a. The student presses "HALLWAYS" to change location list in the Explore window to display a list of hallways.	
	Step 3b. The student presses "PARKING LOTS" to change location list in the Explore window to display a list of parking lots.	

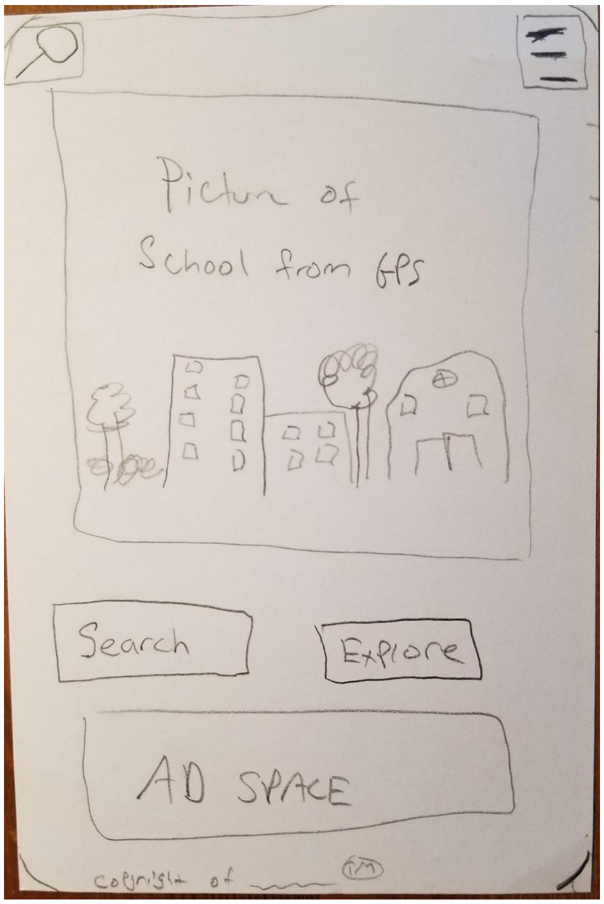
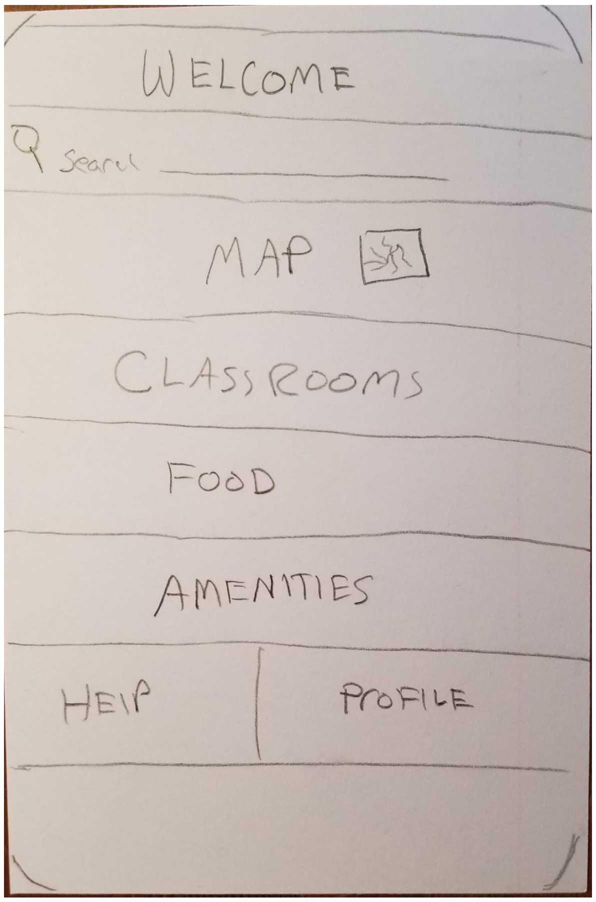
(2-3) Use Case Diagram



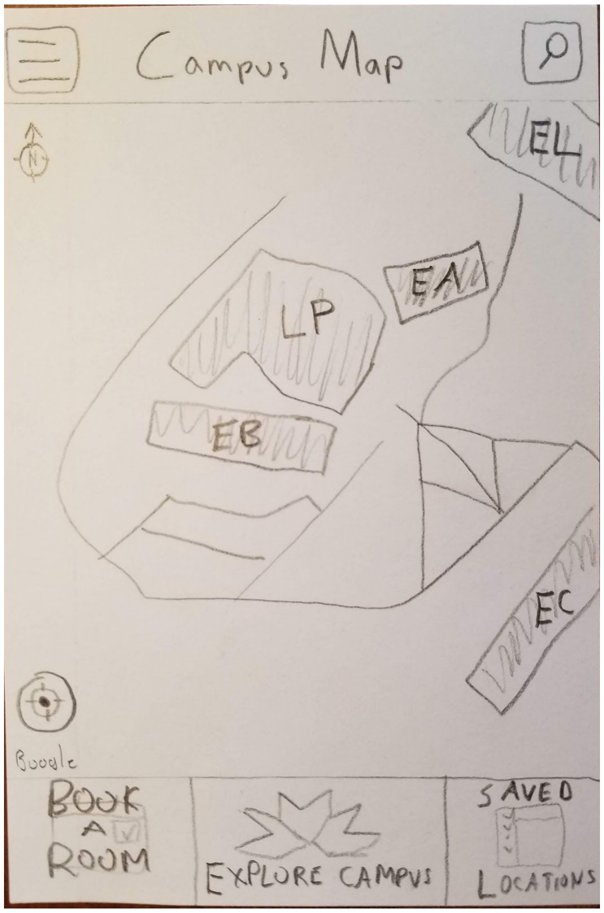
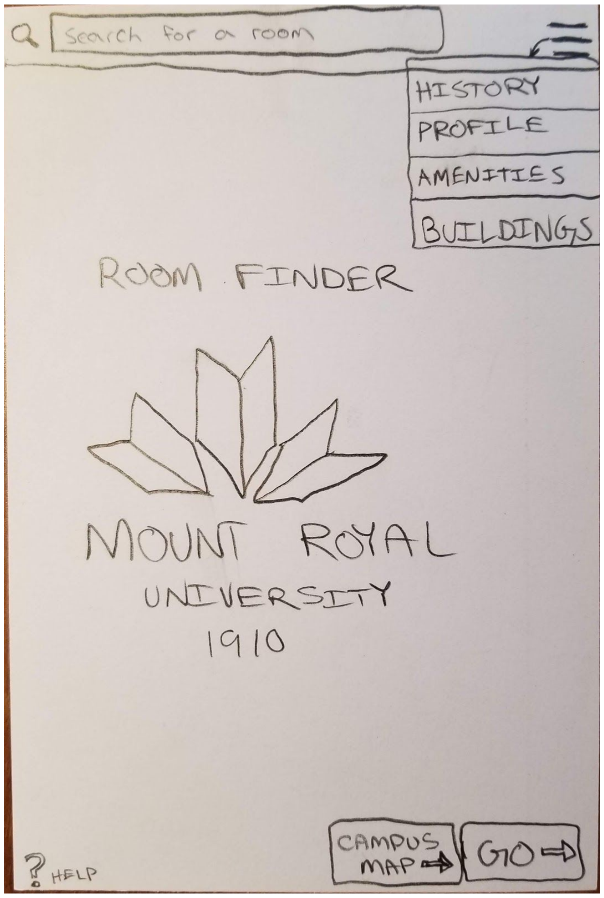
Deliverable #3

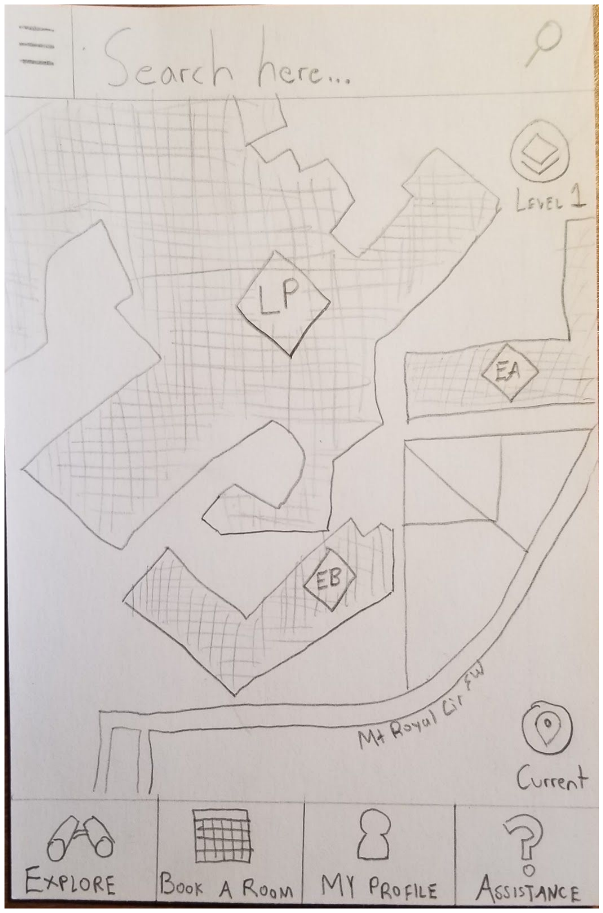
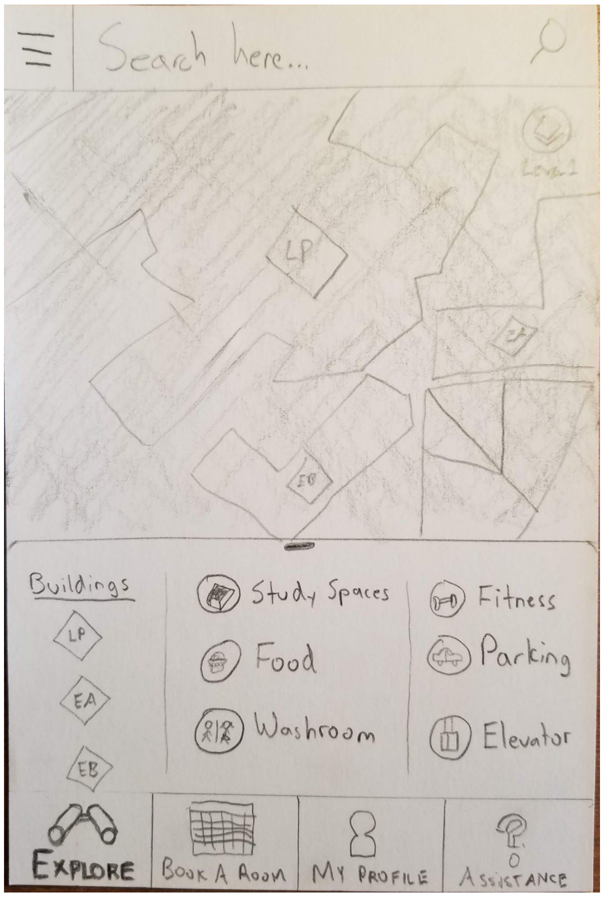
(3-1) Sketches

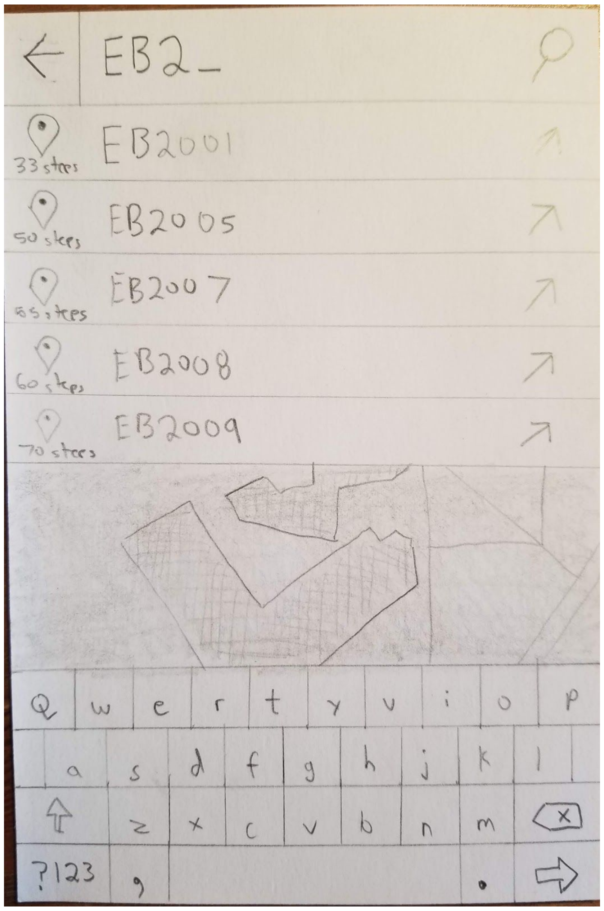
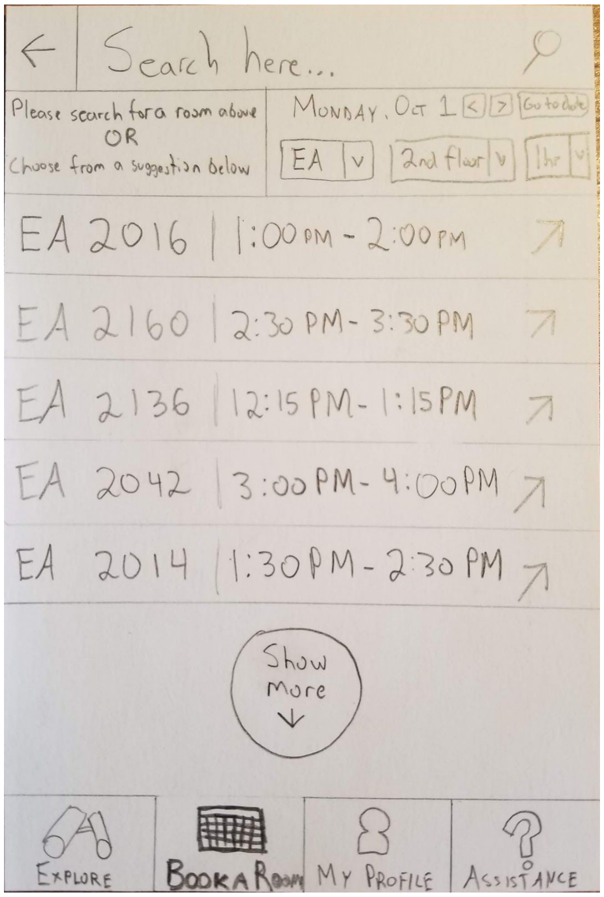
The first five sketches are ideas for the main starting point of the app. Sketches 6-10 are elaborations on sketch 5. We chose sketch 5 to use because the interface feels very familiar to many other apps out there of this kind. The layout is clean and organized and there are minimal elements on the screen to distract from the main functionality. It was easy to expand upon because the interface has many different options to flow to other screens relevant to the functionality of the app.

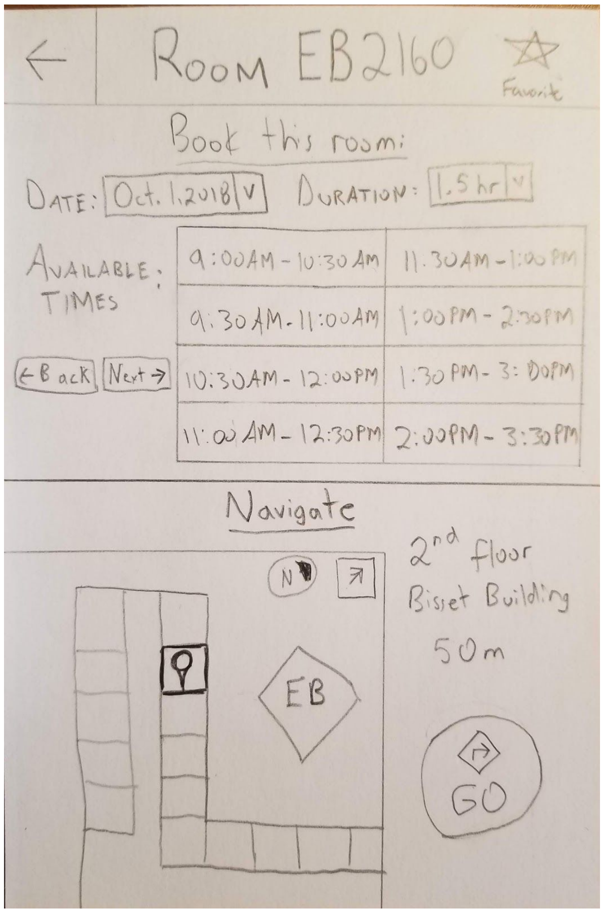
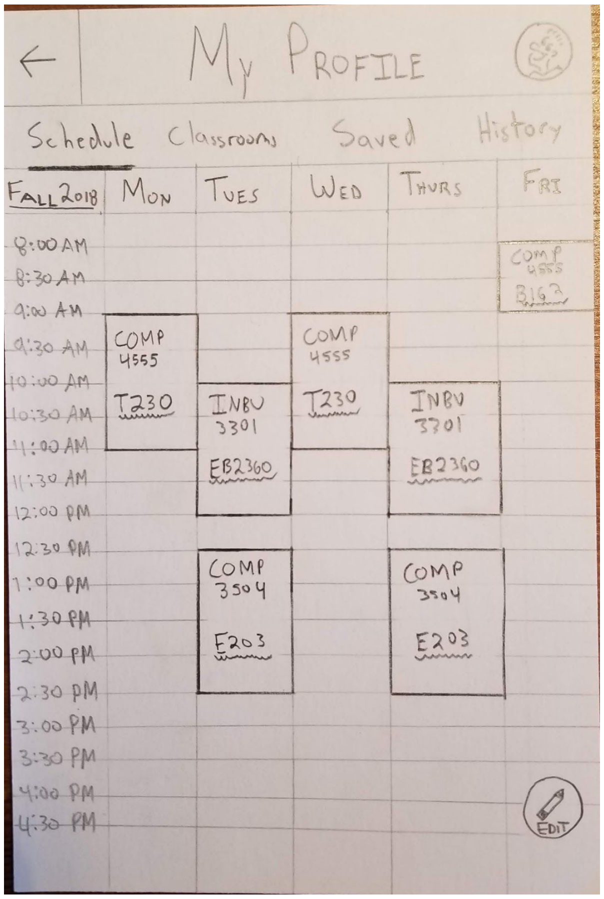
Sketch 1	**Sketch 2**
 Picture of School from GPS Search Explore AD SPACE copyright of TM	 WELCOME Search MAP CLASSROOMS FOOD AMENITIES HELP PROFILE

This interface was an option for the main splash page. A small map is in the center of the screen, a search button along with a menu button at the top, and other buttons at the bottom. Screen space was not fully utilized and therefore, we chose to not use this sketch.	Instead of showing the map from the beginning, the idea for this sketch was to let the user choose what they wanted to see. The interface is extremely self explanatory, but felt too plain and blocky and was therefore not used.

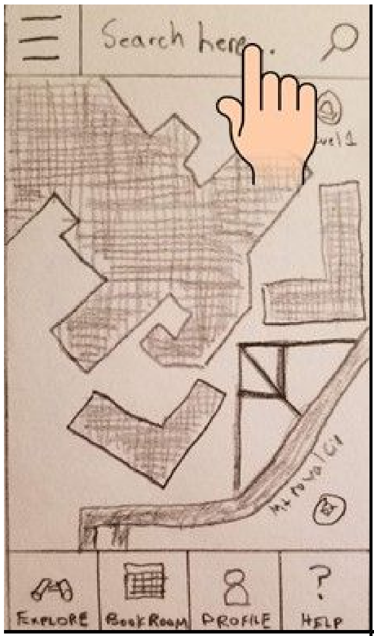
Sketch 3	**Sketch 4**
 Campus Map EL EA LP EB EC Google BOOK A ROOM EXPLORE CAMPUS SAVED LOCATIONS	 Search for a room HISTORY PROFILE AMENITIES BUILDINGS ROOM FINDER MOUNT ROYAL UNIVERSITY 1910 HELP CAMPUS MAP GO
This sketch attempted to use as much screen space without missing functionality. A large map and few options make it easy to understand and use.	A very minimalist approach to the starting screen. Most of the buttons are located in the menu button instead of in a more accessible area. It is very clean and organized.

Sketch 5	**Sketch 6**
 Search here... Level 1 LP EA EB Mt Royal Cir SW Current Explore Book A Room My Profile Assistance	 Search here... Level 1 LP EB Buildings LP EA EB Study Spaces Food Washroom Fitness Parking Elevator Explore Book A Room My Profile Assistance
We chose this sketch to expand upon because of its clean interface, accessible buttons, and good use of space. A large search bar at the top paired with non-intrusive buttons at the bottom give it a modern look and feel.	This sketch shows what happens when the "EXPLORE" button is pressed at the bottom. It shows places of interest and buildings for quick access in case a student is looking for a specific amenity. The map above gets greyed out and the popup screen can be pulled up like a drawer.

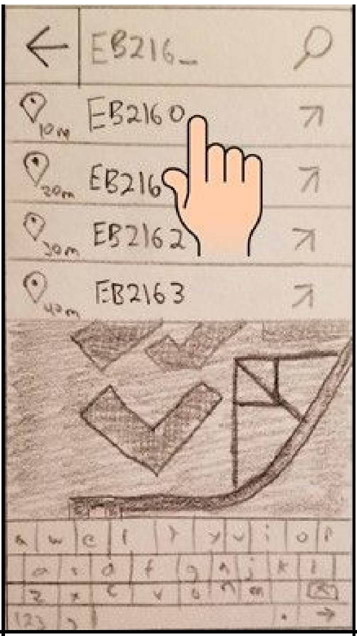
Sketch 7	**Sketch 8**
 EB2_ EB2001 33 steps EB2005 50 steps EB2007 65 steps EB2008 60 steps EB2009 70 steps q w e r t y u i o p a s d f g h j k l z x c v b n m ?123	 Search here... Please search for a room above OR Choose from a suggestion below MONDAY, OCT 1 Go to date EA 2nd floor 1hr EA 2016 | 1:00 PM - 2:00 PM EA 2160 | 2:30 PM - 3:30 PM EA 2136 | 12:15 PM - 1:15 PM EA 2042 | 3:00 PM - 4:00 PM EA 2014 | 1:30 PM - 2:30 PM Show more EXPLORE BOOK A ROOM MY PROFILE ASSISTANCE
When the search bar is used at the top, an autocomplete will appear over the map (which gets greyed out) as well as the system keyboard. It shows roughly how far away each of the suggestions is and they are all clickable to find more information about the room.	When the “BOOK A ROOM” button is selected at the bottom of the main page, a list of suggested open rooms with times is displayed. The user can also search for a specific room by narrowing down the building, floor, duration, and date, or just by searching the exact room at the top.

Sketch 9	**Sketch 10**
 Room EB2160 Favorite Book this room: DATE: Oct. 1, 2018 DURATION: 1.5 hr AVAILABLE TIMES 9:00AM - 10:30AM 11:30AM - 1:00PM 9:30AM - 11:00AM 1:00PM - 2:30PM 10:30AM - 12:00PM 1:30PM - 3:00PM 11:00AM - 12:30PM 2:00PM - 3:30PM Back Next Navigate N 2nd floor Bisset Building 50m EB GO	 My PROFILE Schedule Classrooms Saved History FALL 2018 MON TUES WED THURS FRI 8:00 AM 8:30 AM 9:00 AM 9:30 AM 10:00 AM 10:30 AM 11:00 AM 11:30 AM 12:00 PM 12:30 PM 1:00 PM 1:30 PM 2:00 PM 2:30 PM 3:00 PM 3:30 PM 4:00 PM 4:30 PM COMP 4555 T230 INBU 3301 EB2360 COMP 3504 E203 COMP 4555 B163 EDIT
When a room is selected from a search or other means, an information page is displayed. It has options for booking it and also displays a mini-map of where it is located. The user can navigate to the room by tapping the "GO" button.	Tapping the "MY PROFILE" button at the bottom of the starting page will display a schedule of classes and classrooms that the user has input. They can view this page either as a schedule, just the classrooms, saved/favorited rooms, or view their navigation or search history. This is currently a "nice to have".

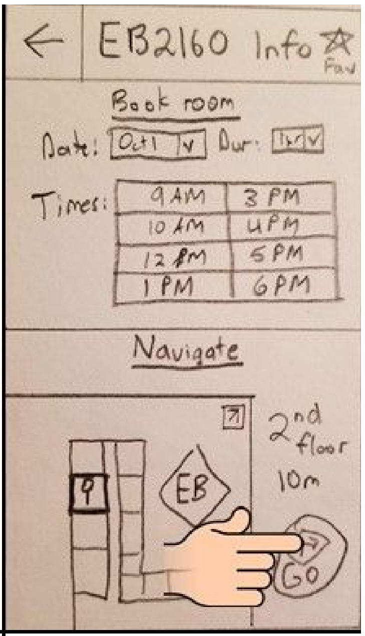
(3-2) Storyboard



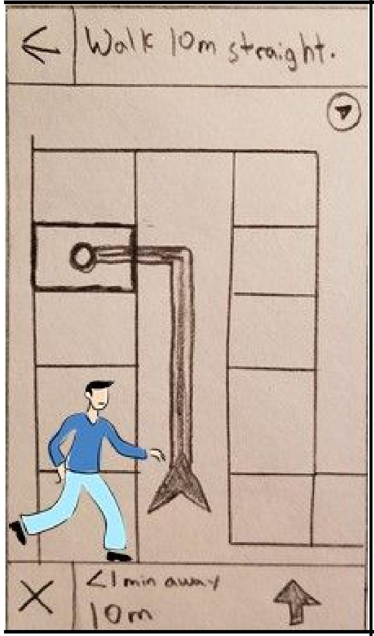
1. Jim is in his first year looking for his first class. He opens the app to see a map of campus, as well as a few convenient buttons at the top and bottom. He taps "Search here...".



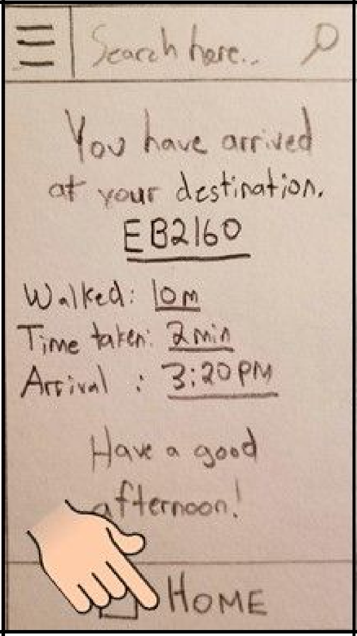
2. He then begins typing his classroom. The suggestions appear, and he sees his classroom at the top. Jim taps the first suggestion.



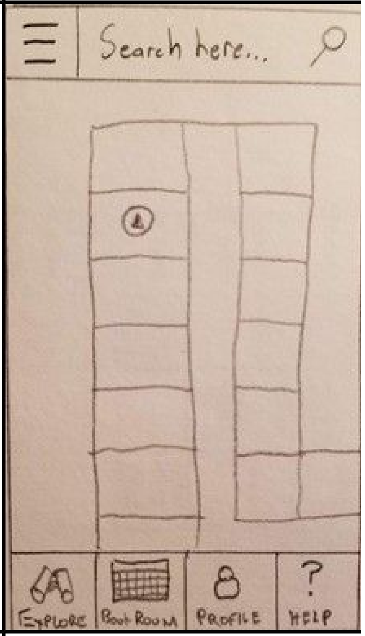
3. An information page appears giving him the option to book it or to navigate to it. It displays a map with the location and some short info about it. He taps "GO".



4. A simple and familiar navigation screen appears, directing Jim to his class which happens to be 10 meters ahead of him. He walks to the class.



5. When Jim enters the class, a pleasant screen appears showing stats about his trip including distance and time. He then taps "HOME" to go back to the home page.



6. The map goes back to the home page and the map shows his current position. He is now ready to search for his next class.

(3-3) Wizard of Oz Testing

Link to video: https://youtu.be/8x4icYsw0Xw

Client Rating Card

Task	Rating	Client Comments
Open the app and search for a room.	10/10	
Favorite the searched room. Navigate to that room.	10/10	
View the navigation history.	9/10	Initially click the top left menu button instead of profile button.

The client did not provide hand drawn sketches because they were satisfied with the interface that they were shown. They were pleased with the system with only one minor problem which had an easy solution.

Deliverable #4

(4-1) Poker Effort Estimation

Chosen functional requirement: **Must be able to lookup a classroom**

Functional Requirement: Be able to lookup a classroom

Tasks	Marc	Max	Connor
1. Must have classroom names stored in an array or other method			
2. Must have location data stored for classrooms			
3. Must be input to search bar in proper format			
4. Must only accept appropriate hall codes for MRU			
5. Must be in valid range of classrooms			

We chose this requirement because it is one of the most essential features of a map application. For our poker effort estimation exercise, we split the deck into three piles, one for each team member. We each had an Ace, 2, 3, 5, 8, and a King. The Ace to 8 cards were to symbolize how difficult we found the tasks, 8 being the most difficult. The King card was present in each hand just in case a member found that the task was completely impossible to do given our constraints in the course. The King could only be used once per person, but the card was never played during our session. The session began with five tasks:

1. Must have the classroom names/numbers stored in an array or other storage method.
2. Must have location data store for classrooms.
3. The user should input the classroom into the search bar in the proper format (i.e. Letter then number).
4. The user must input their search with classroom codes that exist at MRU.
5. The classroom must be within a valid range where the number exists

During the poker session we realized that tasks 3-5 were very similar and could be grouped into one task: validate user input. This was taken into account for the next estimation exercise. Two more general tasks were added to the poker exercise:

6. The location of the class should appear on the map when searched.
7. The actual creation of the search bar interface.

Here are the results from the exercise:

Task	Marc	Max	Connor
1. Must have the classroom names/numbers stored in an array or other storage method.	5	3	5
2. Must have location data store for classrooms.	3	2	2
3. The user should input the classroom into the search bar in the proper format (i.e. Letter then number).	2	A	2
4. The user must input their search with classroom codes that exist at MRU.	3	A	2
5. The classroom must be within a valid range where the number exists	2	A	A
6. The location of the class should appear on the map when searched.	3	2	3
7. The actual creation of the search bar interface.	A	A	A

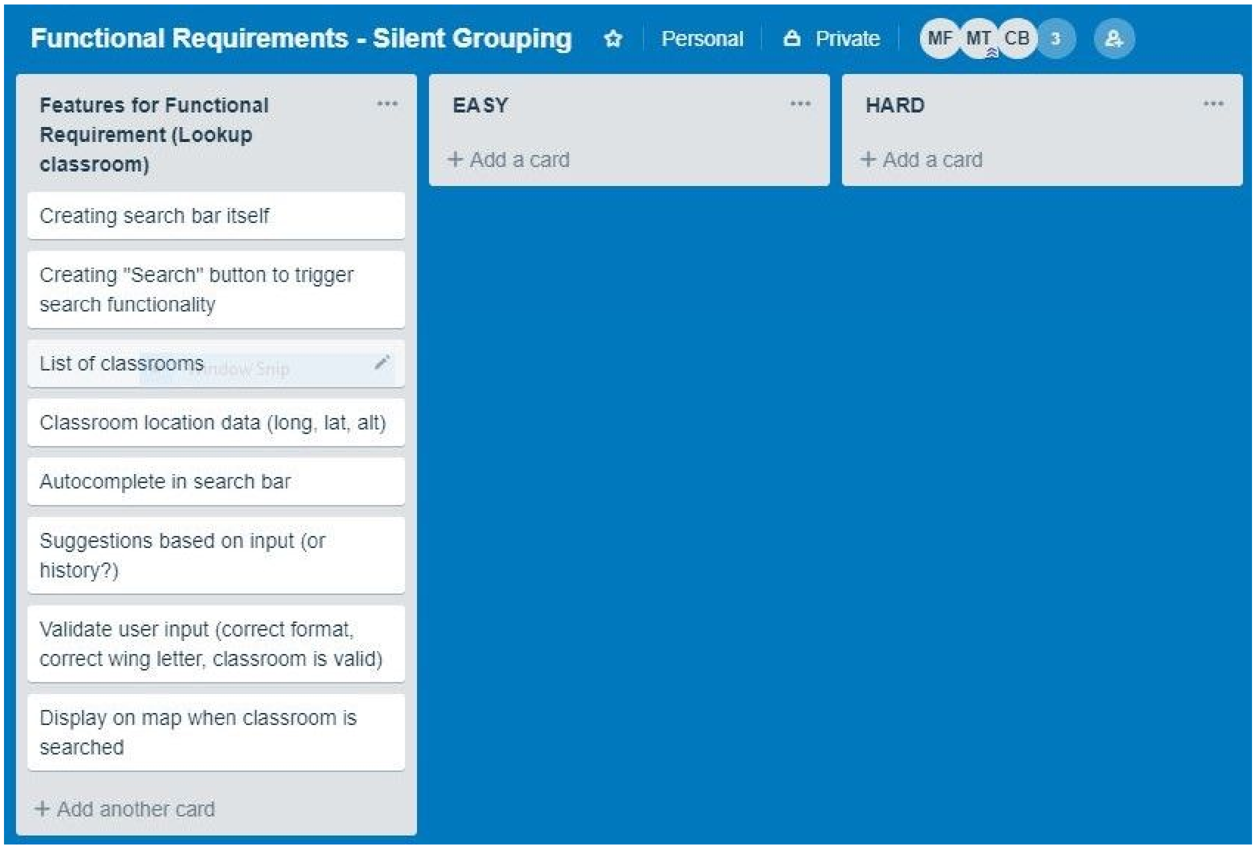
The task that had the highest rating of difficulty was the first task, storing the information for the classrooms. Tasks one and two were so similar that we decided to merge them together since that information should be best to be stored together. The reason we found this task to be more challenging is because there are hundreds of classrooms and it would be difficult to log every single one with its name and location data (consisting of latitude and longitude). This sparked a new idea that we should track our locations based on which building or wing they are located in. By using this method, we can get estimates fairly easily from Google Maps by placing custom pins on certain points of the building. Other concerns such as needing a custom Mount Royal overlay, getting altitude data for floor calculation, and choosing a method to store all this data is why the scores were higher than other tasks.

For tasks 3-5, we merged them together because they all are very similar to each other. Having had some experience in previous classes (Web 2, Programming 2 & 3) in validating user input, the scores for these tasks were low.

The final task (creating the search bar) was estimated to be the easiest task out of all seven. Two of us had experience creating UI elements in the Human Computer Interaction course and felt comfortable with this task. One of us has had a little experience with Android Studio and informed the other members that the creation of the bar itself is not as challenging as implementing the functionality.

(4-2) Silent Grouping Effort Estimation

After the poker estimation session, our tasks and features became more refined for the functional requirement we chose. We were unable to meet in person to complete this exercise, so we used a Trello board online to collaborate together. We started with a column of features that are required for the functional requirement and created the EASY and HARD columns next to it. One by one, we moved the cards from the features column under the difficulty rating we thought was appropriate. It only took a few rounds of moving the features around before we came to a consensus.



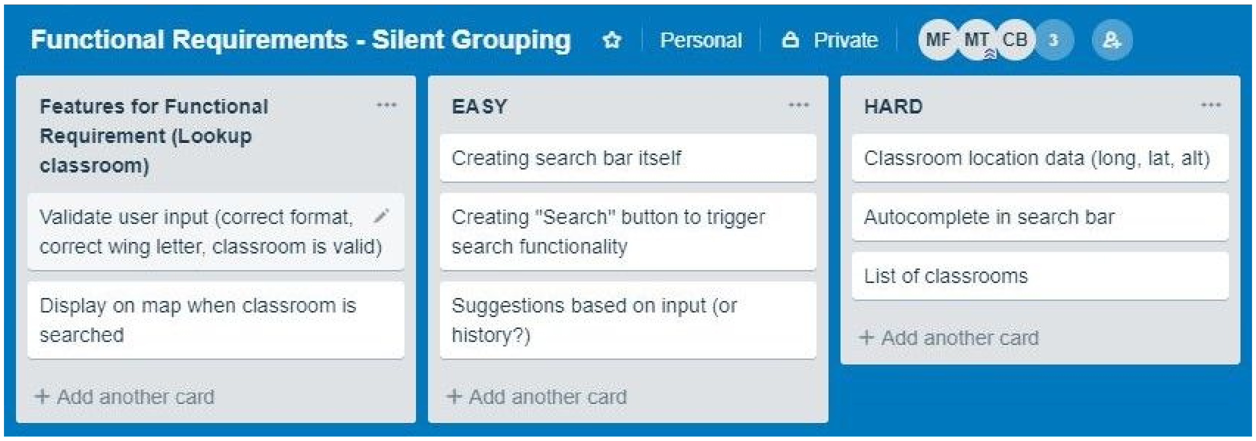
Functional Requirements - Silent Grouping

Features for Functional Requirement (Lookup classroom)
- Creating search bar itself
- Creating "Search" button to trigger search functionality
- List of classrooms
- Classroom location data (long, lat, alt)
- Autocomplete in search bar
- Suggestions based on input (or history?)
- Validate user input (correct format, correct wing letter, classroom is valid)
- Display on map when classroom is searched

EASY

HARD

1. Before grouping began.



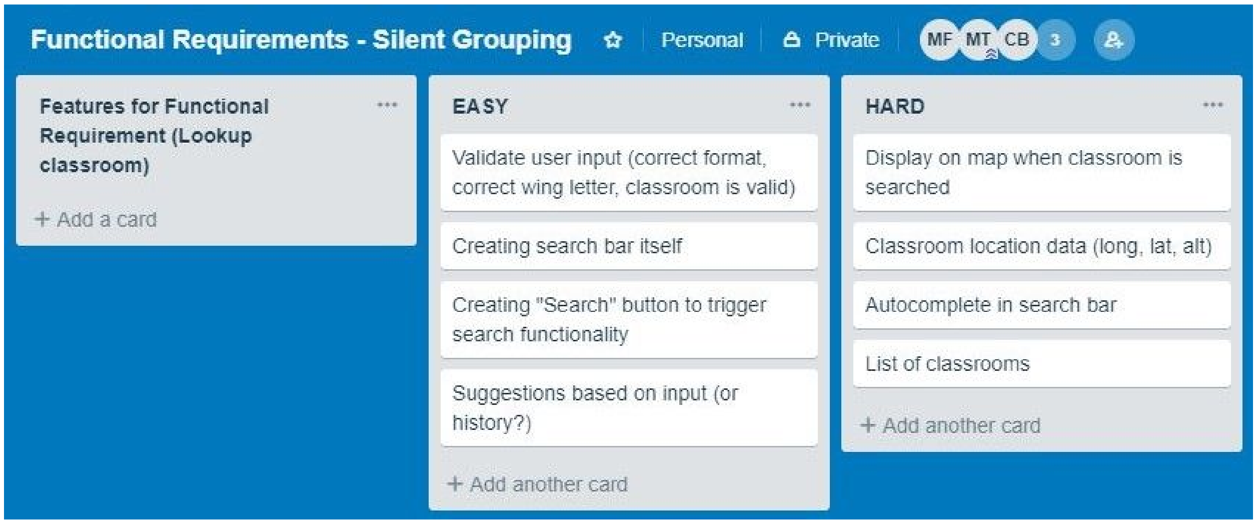
Functional Requirements - Silent Grouping

Features for Functional Requirement (Lookup classroom)
- Validate user input (correct format, correct wing letter, classroom is valid)
- Display on map when classroom is searched

EASY
- Creating search bar itself
- Creating "Search" button to trigger search functionality
- Suggestions based on input (or history?)

HARD
- Classroom location data (long, lat, alt)
- Autocomplete in search bar
- List of classrooms

2. Grouping in progress.


Functional Requirements - Silent Grouping

Features for Functional Requirement (Lookup classroom)	EASY	HARD
	Validate user input (correct format, correct wing letter, classroom is valid)	Display on map when classroom is searched
	Creating search bar itself	Classroom location data (long, lat, alt)
	Creating "Search" button to trigger search functionality	Autocomplete in search bar
	Suggestions based on input (or history?)	List of classrooms

3. Final grouping consensus.

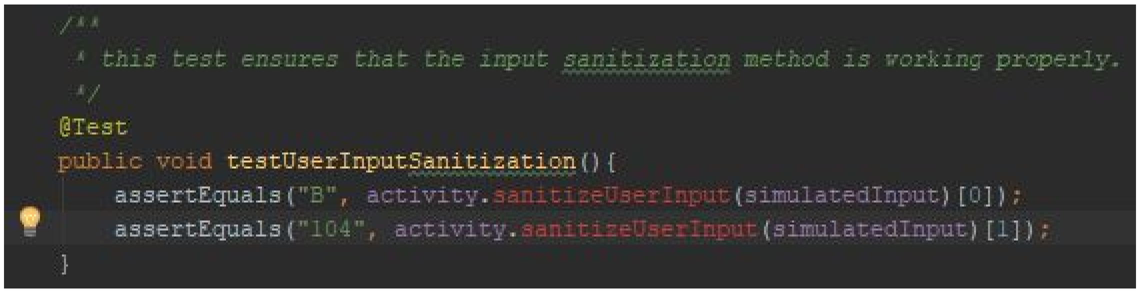
The grouping session was very straightforward and we were able to agree on which tasks might require more effort than others. We judged the difficulty based on the knowledge that we had of implementing such features, guesses as to how much time it might take, and estimations as to how complex a feature might be to implement. We all have had some experience with Java, some more than others, but we were all fairly new to Android Studio and app creation. We were able to understand the concepts behind validating user input or suggestions based on input, so that is why they are in the EASY column. Creating a search bar and button felt as though they may be simple tasks. The HARD cards are mostly related to the functionality of the search bar including: where the data will be stored, how much data to parse through, and how to link the search bar to the Google Maps API. We felt fairly confident in our abilities to be able to work with these features, but as an Agile development team we realize that things are subject to change on a weekly, sometimes daily notice.

This exercise allowed us to think and plan out how we were going to begin creating our classes and methods. Since we did the effort estimation and planning for the “lookup classroom” functional requirement, we decided to choose it as the requirement to implement using the TDD approach. We see how using these planning methods can help development teams gauge how much work might need to be done on specific tasks and how it helps set deadlines and priorities for projects.

Deliverable #5

(5-1) Initial Tests

1. `testUserInputSanitization`

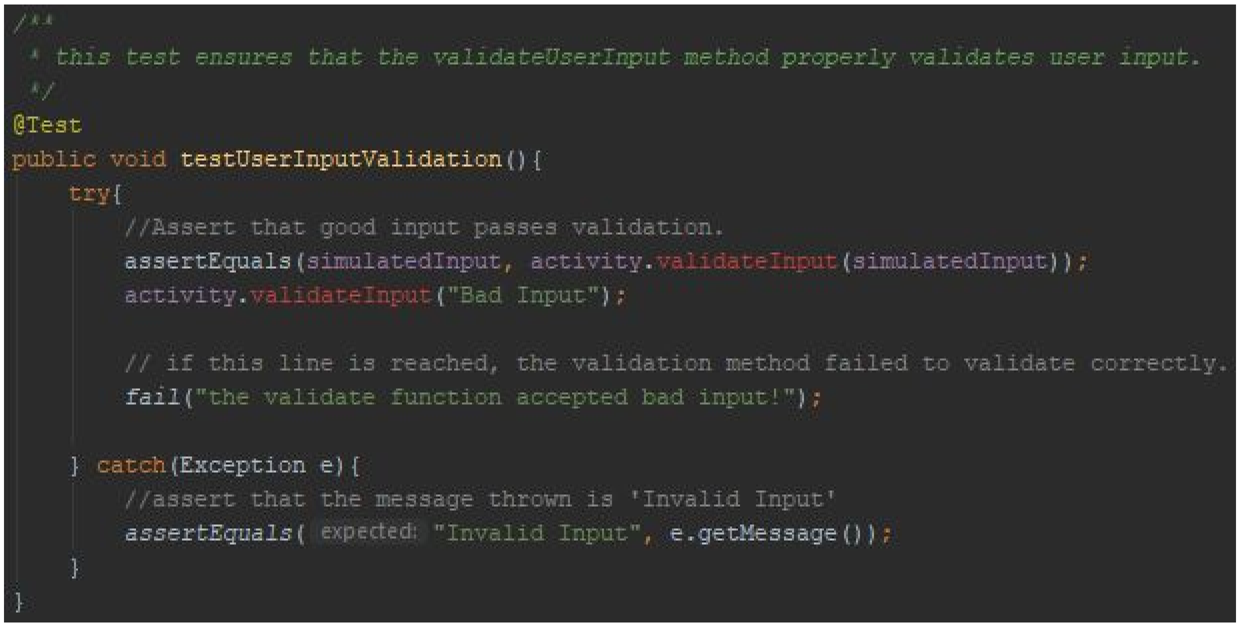


```java
/**
 * this test ensures that the input sanitization method is working properly.
 */
@Test
public void testUserInputSanitization(){
    assertEquals("B", activity.sanitizeUserInput(simulatedInput)[0]);
    assertEquals("104", activity.sanitizeUserInput(simulatedInput)[1]);
}
```

This will test that the sanitization method works properly.

- The method should take in an input such as B104 and return an array with the hallway code in the first position and the number in the second position.

2. `testUserInputValidation`

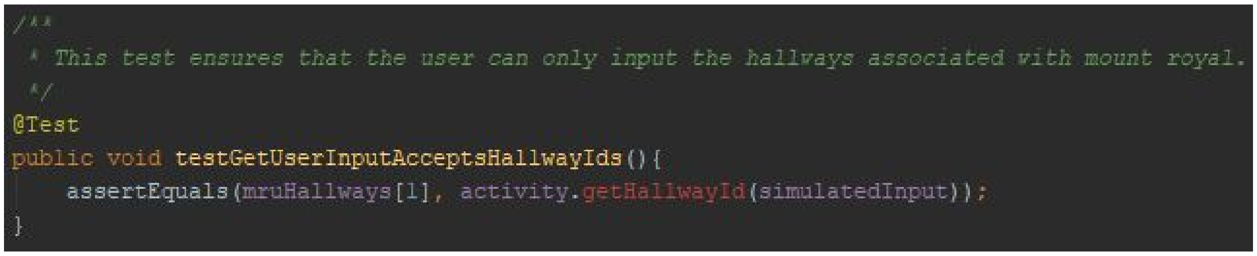


```java
/**
 * this test ensures that the validateUserInput method properly validates user input.
 */
@Test
public void testUserInputValidation(){
    try{
        //Assert that good input passes validation.
        assertEquals(simulatedInput, activity.validateInput(simulatedInput));
        activity.validateInput("Bad Input");

        // if this line is reached, the validation method failed to validate correctly.
        fail("the validate function accepted bad input!");

    } catch(Exception e){
        //assert that the message thrown is 'Invalid Input'
        assertEquals( expected: "Invalid Input", e.getMessage());
    }
}
```

This will ensure that the validation is working properly, and only hallway codes that the school has are accepted, and ones that do not exist are rejected

- The first assert will test a simulated valid input.
- By calling `activity.validateInput()` with an improper input, it should cause an error to be thrown
    - Assert that the message returned in the error equals ‘Invalid Input’

3. `testGetUserInputAcceptsHallwayIds`

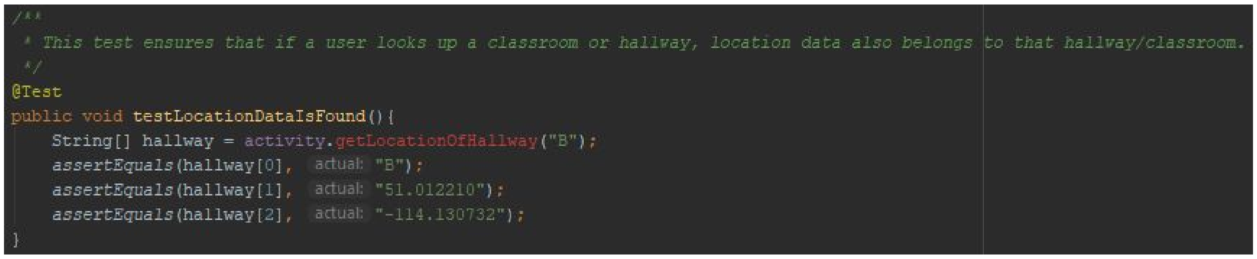


```java
/**
 * This test ensures that the user can only input the hallways associated with mount royal.
 */
@Test
public void testGetUserInputAcceptsHallwayIds(){
    assertEquals(mruHallways[1], activity.getHallwayId(simulatedInput));
}
```

This method will test that all of the hallway codes are accepted by the system.

- Possible hallway codes: `{"A","B","C","D","E","F","G","H","I","J","K","M","N","O","Q","R","S","T","U","V","W","X","Y","Z","EA","EB","EC","ED","EL"}`

4. `testLocationDataIsFound`

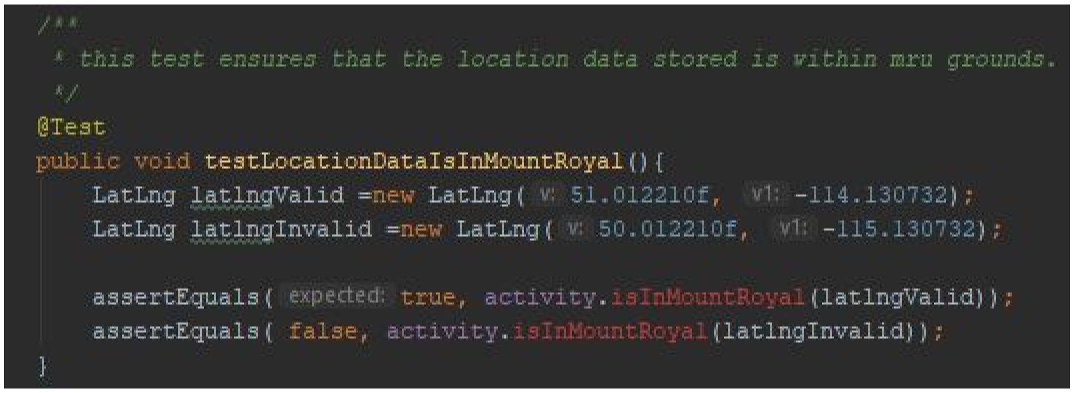


```java
/**
 * This test ensures that if a user looks up a classroom or hallway, location data also belongs to that hallway/classroom.
 */
@Test
public void testLocationDataIsFound(){
    String[] hallway = activity.getLocationOfHallway("B");
    assertEquals(hallway[0],  actual: "B");
    assertEquals(hallway[1],  actual: "51.012210");
    assertEquals(hallway[2],  actual: "-114.130732");
}
```

This method will test that there is location data associated with a hallway, and that it is returned in an array from the method get Location Hallway which takes in a hallway code.

- The first item in the array should be the hallway code.
- The second item in the array should be the latitude
- The third item in the array should be the longitude.

5. `testLocationDataIsInMountRoyal`

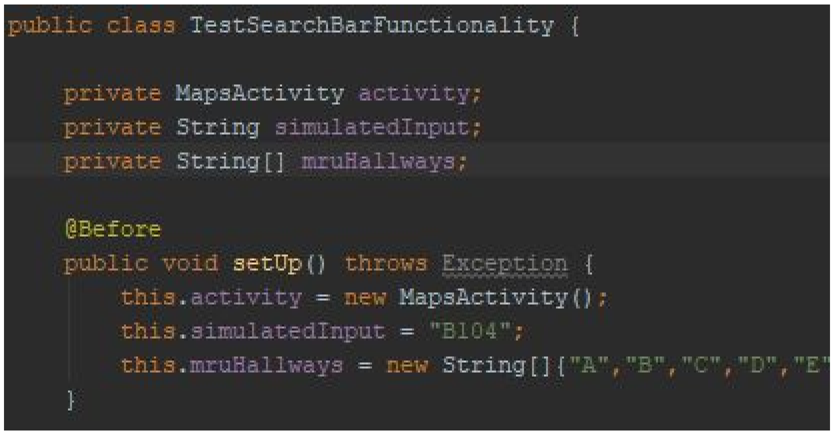


```java
/**
 * this test ensures that the location data stored is within mru grounds.
 */
@Test
public void testLocationDataIsInMountRoyal(){
    LatLng latlngValid =new LatLng( v: 51.012210f,  v1: -114.130732);
    LatLng latlngInvalid =new LatLng( v: 50.012210f,  v1: -115.130732);

    assertEquals( expected: true, activity.isInMountRoyal(latlngValid));
    assertEquals( false, activity.isInMountRoyal(latlngInvalid));
}
```

This test will ensure that each of the stored locations are actually within the bounds of MRU.

- Test a valid location and an invalid location returns false in the isInMountRoyal method.

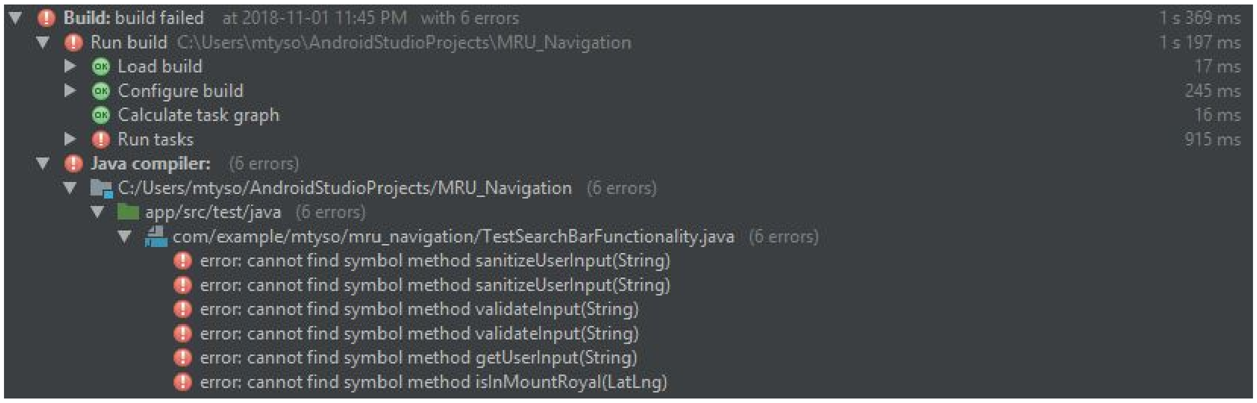
Tests Setup



```java
public class TestSearchBarFunctionality {

    private MapsActivity activity;
    private String simulatedInput;
    private String[] mruHallways;

    @Before
    public void setUp() throws Exception {
        this.activity = new MapsActivity();
        this.simulatedInput = "B104";
        this.mruHallways = new String[]{"A","B","C","D","E"
    }
```

- We use this method to build the objects required for the tests that are shared across many tests.

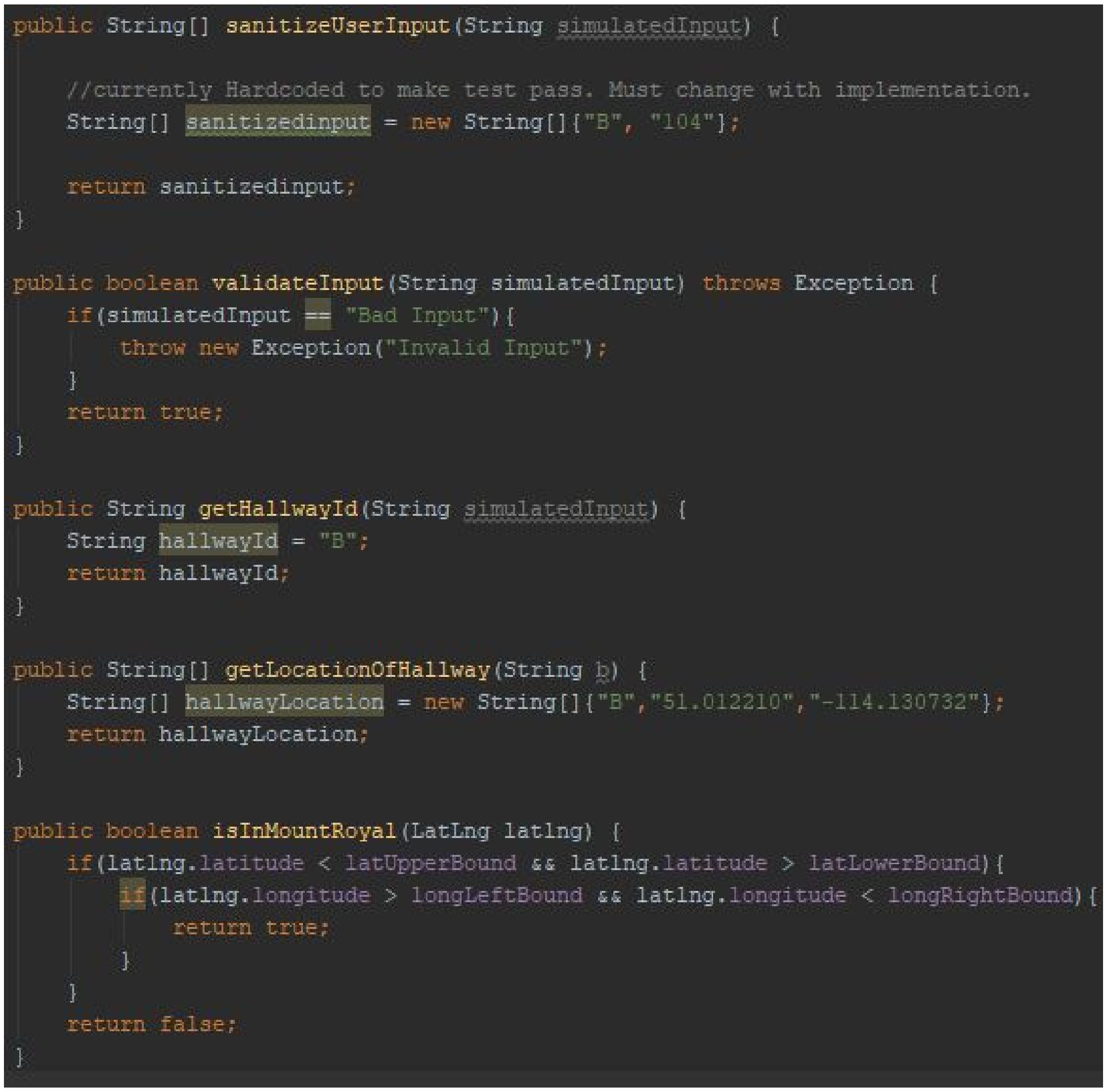
First Run



```
▼ Build: build failed   at 2018-11-01 11:45 PM   with 6 errors                                1 s 369 ms
  ▼ Run build  C:\Users\mtyso\AndroidStudioProjects\MRU_Navigation                              1 s 197 ms
    ▶ Load build                                                                                     17 ms
    ▶ Configure build                                                                               245 ms
      Calculate task graph                                                                           16 ms
    ▶ Run tasks                                                                                     915 ms
  ▼ Java compiler:   (6 errors)
    ▼ C:/Users/mtyso/AndroidStudioProjects/MRU_Navigation   (6 errors)
      ▼ app/src/test/java   (6 errors)
        ▼ com/example/mtyso/mru_navigation/TestSearchBarFunctionality.java   (6 errors)
            error: cannot find symbol method sanitizeUserInput(String)
            error: cannot find symbol method sanitizeUserInput(String)
            error: cannot find symbol method validateInput(String)
            error: cannot find symbol method validateInput(String)
            error: cannot find symbol method getUserInput(String)
            error: cannot find symbol method isInMountRoyal(LatLng)
```

- The first run of tests failed as these methods need to be implemented within the `MapsActivity` Class.

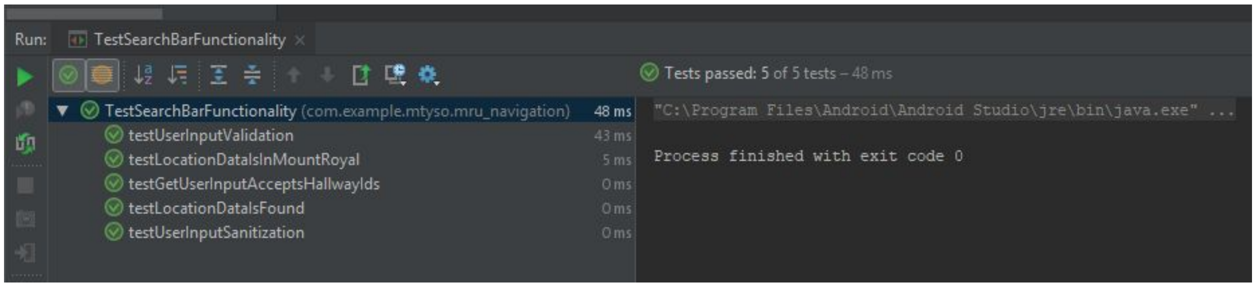
(5-2) Tests with Partial Implementation



```java
public String[] sanitizeUserInput(String simulatedInput) {

    //currently Hardcoded to make test pass. Must change with implementation.
    String[] sanitizedinput = new String[]{"B", "104"};

    return sanitizedinput;
}

public boolean validateInput(String simulatedInput) throws Exception {
    if(simulatedInput == "Bad Input"){
        throw new Exception("Invalid Input");
    }
    return true;
}

public String getHallwayId(String simulatedInput) {
    String hallwayId = "B";
    return hallwayId;
}

public String[] getLocationOfHallway(String b) {
    String[] hallwayLocation = new String[]{"B","51.012210","-114.130732"};
    return hallwayLocation;
}

public boolean isInMountRoyal(LatLng latlng) {
    if(latlng.latitude < latUpperBound && latlng.latitude > latLowerBound){
        if(latlng.longitude > longLeftBound && latlng.longitude < longRightBound){
            return true;
        }
    }
    return false;
}
```

Implemented Missing methods

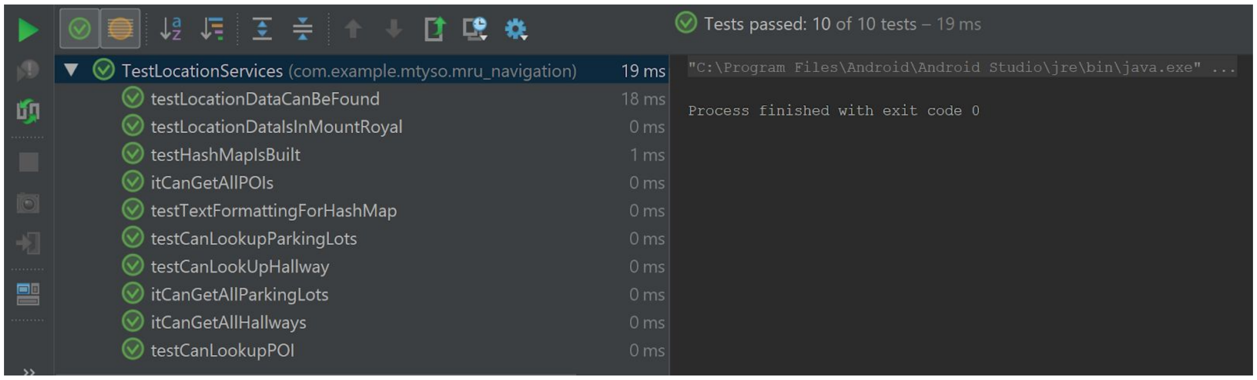
- Only returned minimum of what is required/what the test is expecting to pass all tests.
- As tests expand, these methods will require further implementation.
- The tests have not been changed at all from the first implementation.
- The image below shows that all tests are passing with this implementation.



```
Tests passed: 5 of 5 tests – 48 ms

▼ TestSearchBarFunctionality (com.example.mtyso.mru_navigation)   48 ms
    testUserInputValidation                                        43 ms
    testLocationDataIsInMountRoyal                                  5 ms
    testGetUserInputAcceptsHallwayIds                               0 ms
    testLocationDataIsFound                                         0 ms
    testUserInputSanitization                                       0 ms

"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...

Process finished with exit code 0
```

This test class has changed a lot due to the implementation of a hash map and location data access layer to meet other functional requirements. The dataHandler and the data access layer will handle the user input validation and sanitization before querying the map. This functionality still exists within our system but has been relocated and modified to allow our other functional requirements to be implemented properly.

Final Implementation of changed tests all passing with implementation:



```
Tests passed: 10 of 10 tests – 19 ms

TestLocationServices (com.example.mtyso.mru_navigation)  19 ms
    testLocationDataCanBeFound                           18 ms
    testLocationDataIsInMountRoyal                        0 ms
    testHashMapIsBuilt                                    1 ms
    itCanGetAllPOIs                                       0 ms
    testTextFormattingForHashMap                          0 ms
    testCanLookupParkingLots                              0 ms
    testCanLookUpHallway                                  0 ms
    itCanGetAllParkingLots                                0 ms
    itCanGetAllHallways                                   0 ms
    testCanLookupPOI                                      0 ms

"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...

Process finished with exit code 0
```

Link to Hallway Lookup final implementation:
https://www.youtube.com/watch?v=HIw2y3W1Q_o

Link to Show Users Location and store users search history implementation:
https://www.youtube.com/watch?v=qK266e4xSHA

Link to GitHub Repository: https://github.com/MXZTY/MRU_Navigation

DataHandler:
https://github.com/MXZTY/MRU_Navigation/blob/master/app/src/main/java/com/example/mtyso/mru_navigation/DataHandler1.java

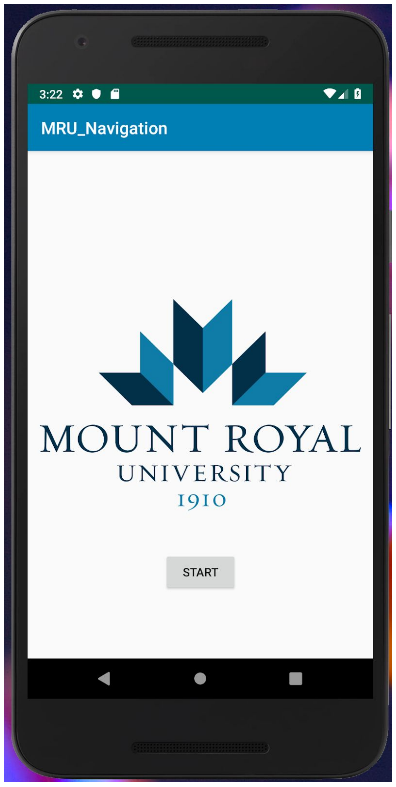
LocationAccessLayer:
https://github.com/MXZTY/MRU_Navigation/blob/master/app/src/main/java/com/example/mtyso/mru_navigation/LocationAccessLayer.java

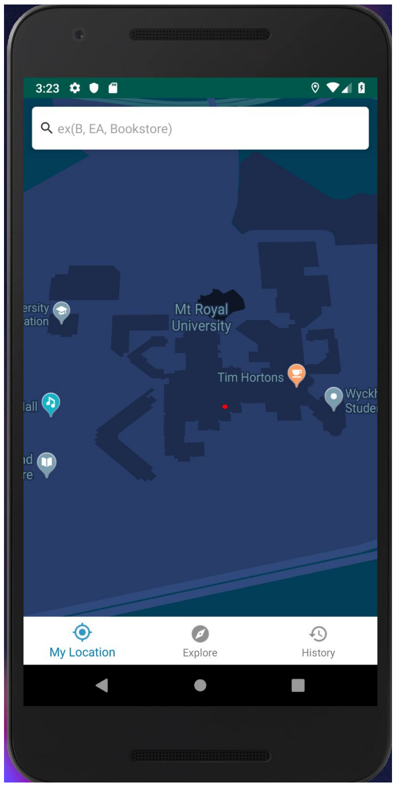
Tests:
https://github.com/MXZTY/MRU_Navigation/blob/master/app/src/test/java/com/example/mtyso/mru_navigation/TestLocationServices.java

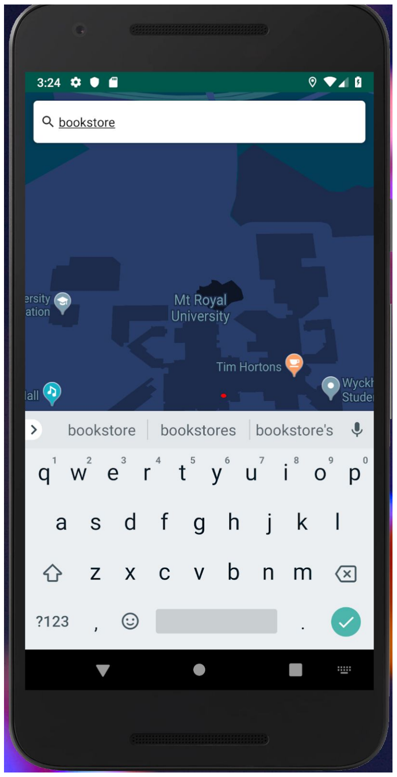
(5-3) Tests with Full Implementation

This video demonstrates how the application will respond to a user inputting a hallway ID. The user enters a hallway, such as B, into the search bar, and the application will process the request and place a pin on the Google Map. The pin represents where the hallways is located. If the user enters an incorrect hallway, no pinpoint will be displayed on the map. In the next iteration, we will visualize when a user input error occurs by outlining the textbox with red and adding a jitter to the search box.

Link to Video (https://youtu.be/HIw2y3W1Q_o)



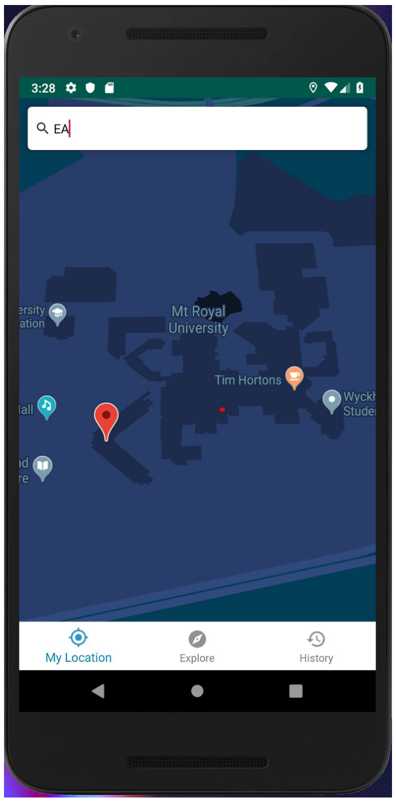


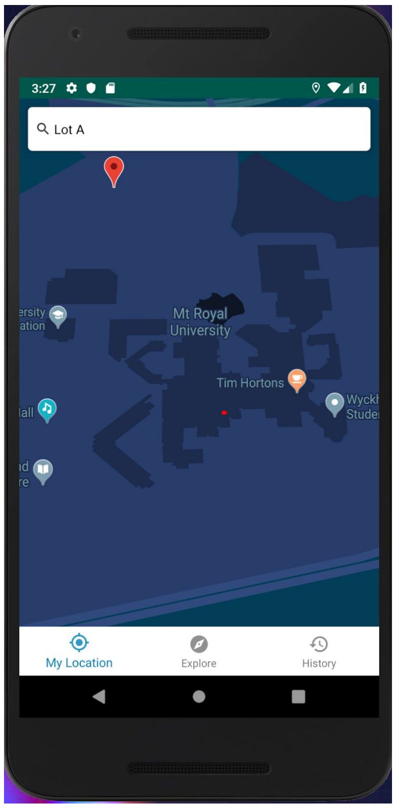


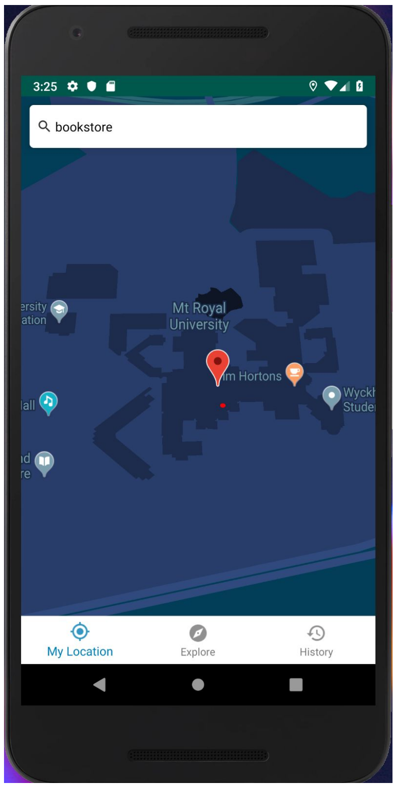
This is the welcome page that is displayed when the user launches the app. In the background, the location arrays are stored in a hashmap.

The Google Map is built and the users location is displayed by the red dot on the map. The application is waiting for the user to enter a location in the search bar.

On the click of the search bar, the users input is received. When the user hits enter, the string is formatted and then searched for in the hashmap. That string is also stored in a userHistory array.







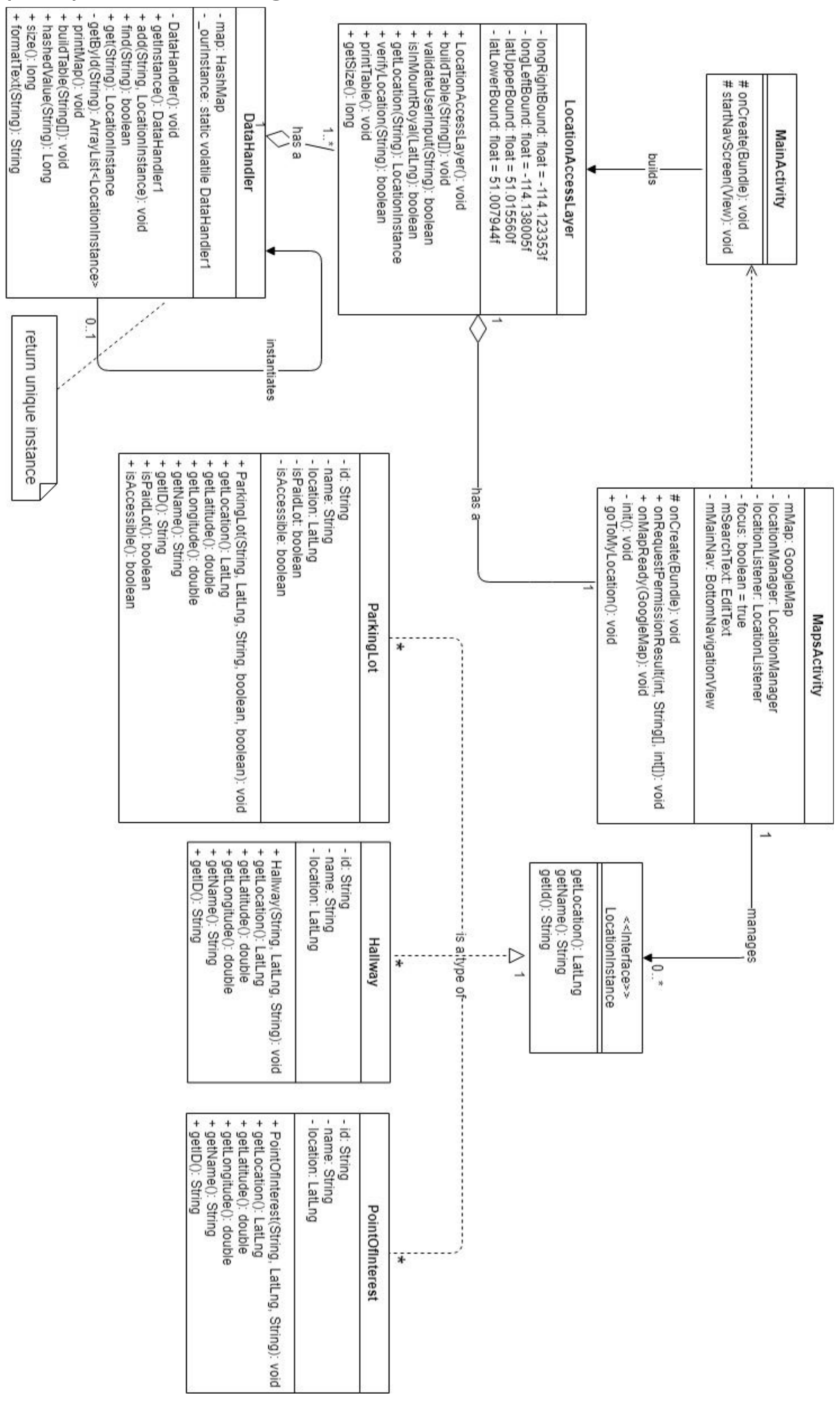
In this case, the user has entered “EA”. The EA building is displayed on the map with a red pin. This shows that users can search Mount Royal University hallways and buildings.

In this case, the user has entered “lot A”. A comparison is ran between the users entry and the hash map. Once found, the location is displayed on the map with a red pin.

In this case, the user has entered “bookstore”, which is one of the Points of Interest at Mount Royal University. The bookstore is displayed on the map with a red pin. We have added 39 POI’s for the user to search for.

Deliverable #6

(6-1) Class Diagram



(6-2) Class Diagram Explanation

The class diagram for this app is fairly simple. The MainActivity class serves to startup essential features of the app and allows for a simple start screen. Once the “Start” button is pressed, the MainActivity class creates the LocationAccessLayer, an abstraction for the DataHandler singleton that it creates and populates. The LocationAccessLayer class also validates user input to make sure that the inputted search string actually exists within the DataHandler singleton. The basic premise of the LocationAccessLayer is to verify location data inputted by the user and to access the DataHandler singleton hashmap.

The DataHandler class holds the hashmap of all the location instances including hallways, points of interest, and parking lots. Since it is a singleton, the object is always instantiated and can be accessed from across the app and other classes. It’s methods create the Location instances with the data that is held in the location arrays XML file. Once the hashmap has been populated, the MapsActivity screen is created. The MapsActivity screen holds a GoogleMap object, an EditText search bar, and a BottomNavigationView bar. This class manages the map itself and the interactions between the user inputs and the Location instances in the LocationAccessLayer

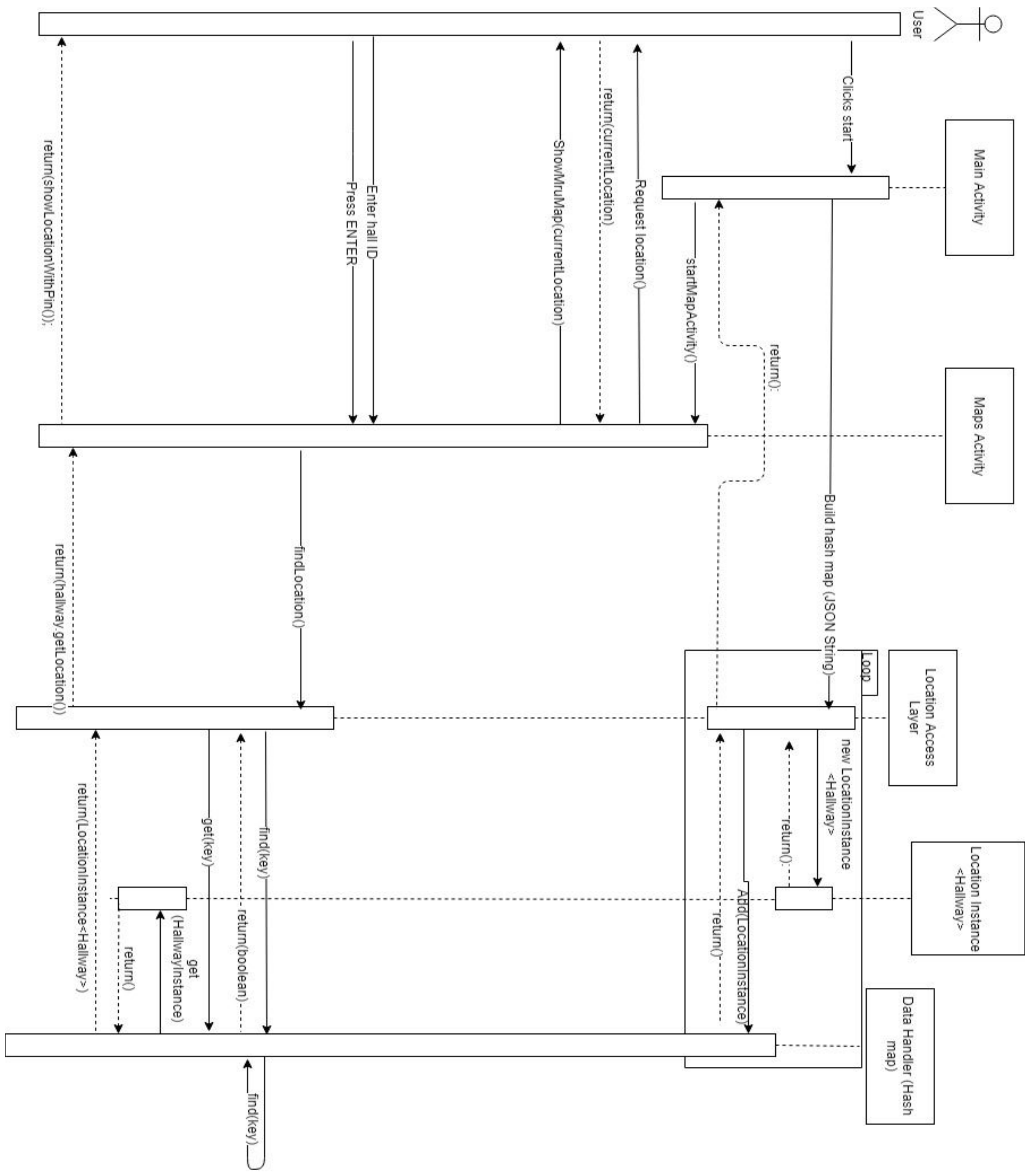
The LocationInstance interface requires that a Location has an ID, a name, and LatLng location data. Hallway and PointOfInterest classes implement the methods in the interface and has getter methods for each attribute. The ParkingLot class does the same, but has additional attributes for isPaidLot and isAccessible, each with their own getter method.

The relationships between the classes are very simple because of the structure of the app. The MainActivity class builds one LocationAccessLayer object which has one DataHandler singleton. The singleton can belong to one or many LocationAccessLayer classes, but in this implementation, there is only one LocationAccessLayer. The MapsActivity class has the one

LocationAccessLayer and the LocationAccessLayer only belongs to the one MapsActivity class. MapsActivity can manage zero or multiple LocationInstance objects. Each ParkingLot, Hallway, and PointOfInterest class is a type of one LocationInstance interface, and the LocationInstance interface is used by multiple types. The DataHandler singleton can make one or no instantiations of itself. If another DataHandler object is attempted to be created, the object will always reference the one singleton.

Deliverable #7

(7-1) Sequence Diagram



(7-2) Sequence Diagram Explanation

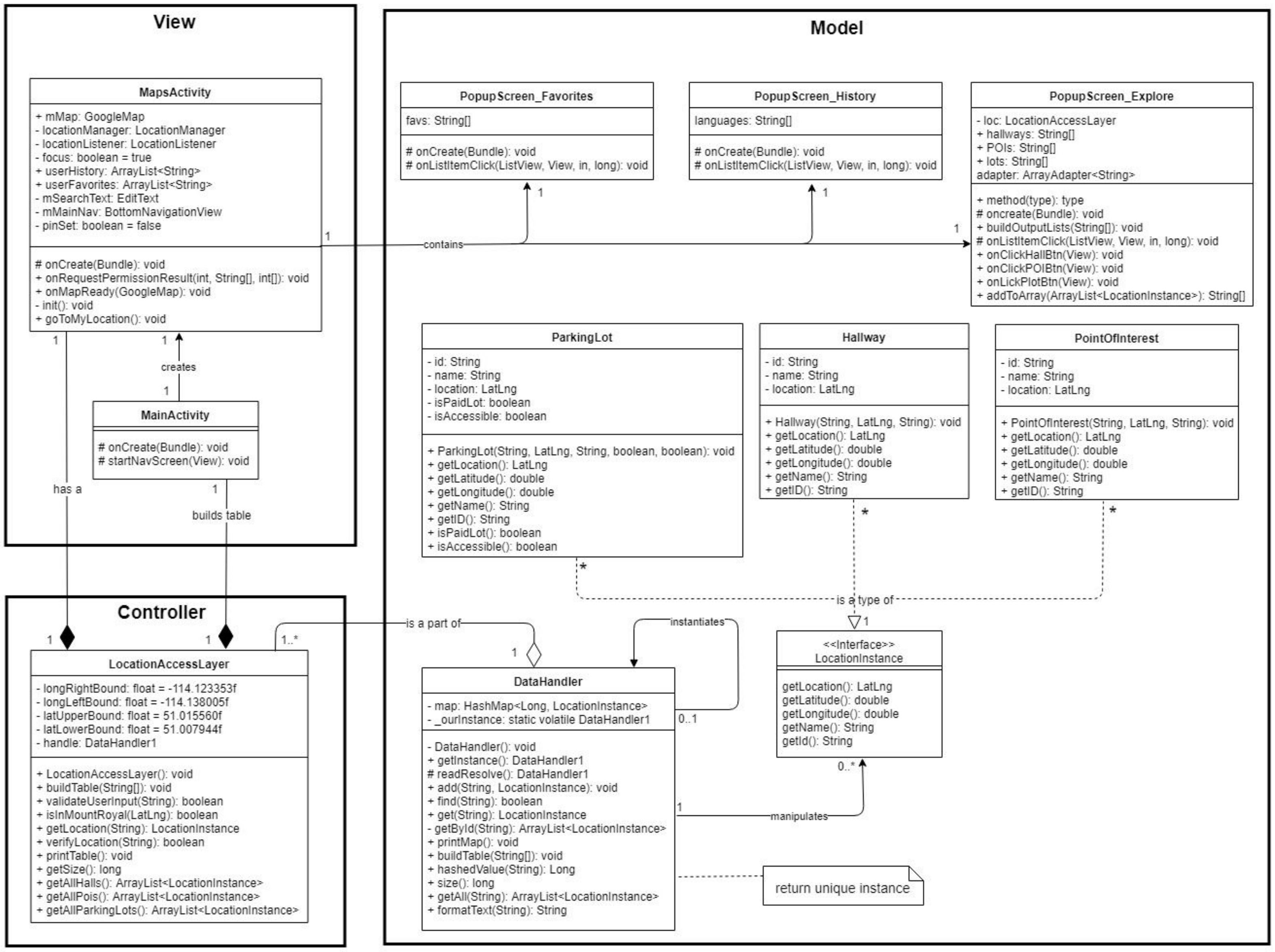
This diagram shows two of our use cases: shows users current location, and user can search for points of interest. The app begins when the user opens it and clicks “Start” on the MainActivity start page. This will build the LocationAccessLayer, which is an abstraction for the DataHandler singleton. The hash map of locations is created by making new Location instances. A location instance can be a Hallway, a Point of Interest (POI), or a Parking Lot (pLot). Once created, the Location instance is added to the DataHandler hashmap singleton. This process loops until all data from the XML data files is imported.

MainActivity now starts the MapsActivity page. When the MapsActivity page is loaded, it requests the user's current location from the device and is displayed on the screen, thus completing the functional requirement.

The MapsActivity screen now waits for user input. The user can enter a hall ID, a point of interest, or a parking lot ID into the search bar. They press enter and the MapsActivity access the LocationAccessLayer abstraction. It sends the key (user input) to the singleton hashmap to find the location and see if it exists. If it does, it returns true to the LocationAccessLayer and it sends the key again to get the location information from the singleton. It will return the instance with the location data to the LocationAccessLayer which again returns the location information to MapsActivity. With the LatLng information, the MapsActivity screen now displays a pin of the location that the user searched, completing the second functional requirement.

Deliverable #8

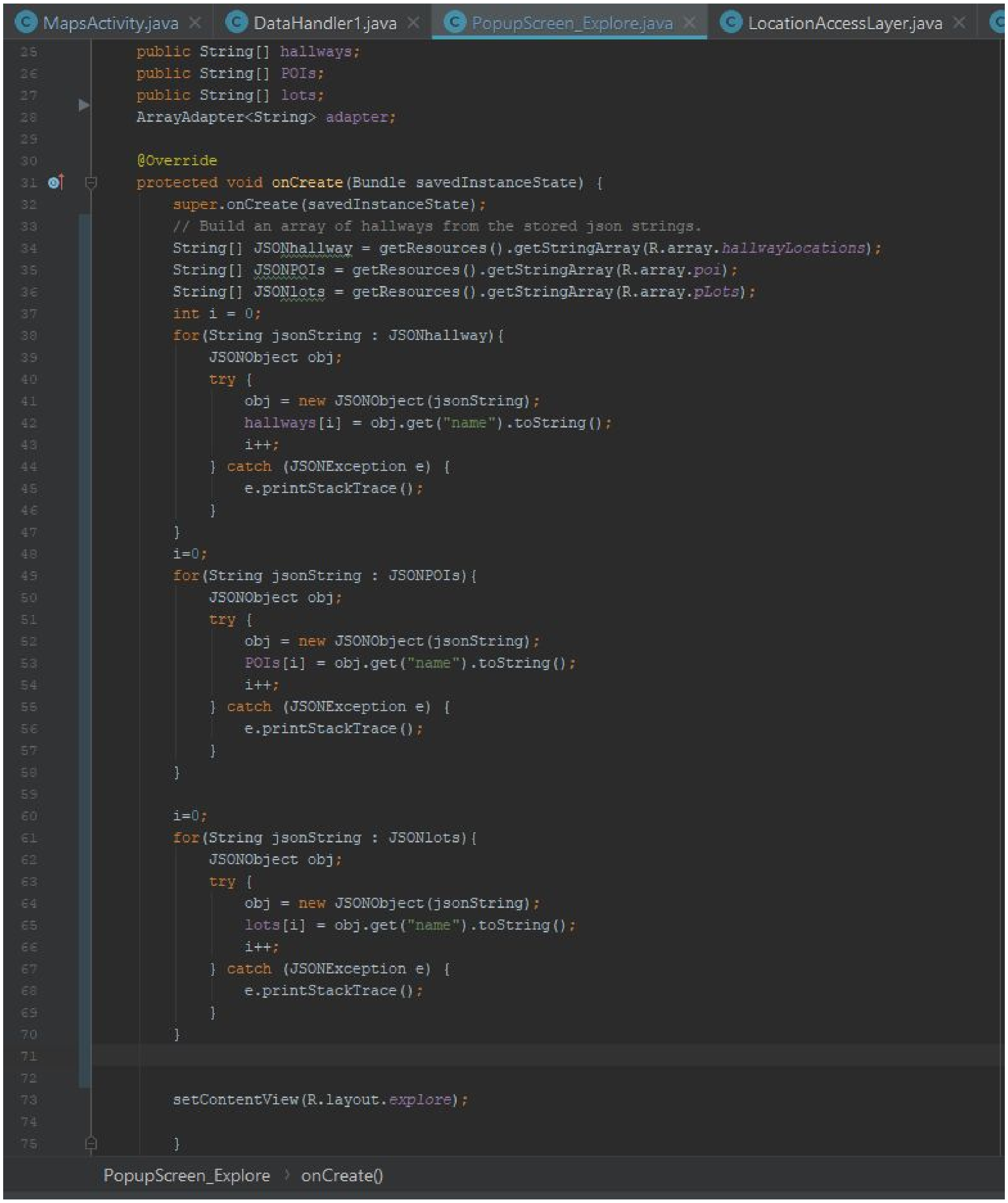
(8-1) Design Pattern


View
MapsActivity
+ mMap: GoogleMap
- locationManager: LocationManager
- locationListener: LocationListener
- focus: boolean = true
+ userHistory: ArrayList<String>
+ userFavorites: ArrayList<String>
- mSearchText: EditText
- mMainNav: BottomNavigationView
- pinSet: boolean = false
# onCreate(Bundle): void
+ onRequestPermissionResult(int, String[], int[]): void
+ onMapReady(GoogleMap): void
- init(): void
+ goToMyLocation(): void
creates
MainActivity
# onCreate(Bundle): void
# startNavScreen(View): void
has a
builds table
Controller
LocationAccessLayer
- longRightBound: float = -114.123353f
- longLeftBound: float = -114.138005f
- latUpperBound: float = 51.015560f
- latLowerBound: float = 51.007944f
- handle: DataHandler1
+ LocationAccessLayer(): void
+ buildTable(String[]): void
+ validateUserInput(String): boolean
+ isInMountRoyal(LatLng): boolean
+ getLocation(String): LocationInstance
+ verifyLocation(String): boolean
+ printTable(): void
+ getSize(): long
+ getAllHalls(): ArrayList<LocationInstance>
+ getAllPois(): ArrayList<LocationInstance>
+ getAllParkingLots(): ArrayList<LocationInstance>
Model
PopupScreen_Favorites
favs: String[]
# onCreate(Bundle): void
# onListItemClick(ListView, View, in, long): void
PopupScreen_History
languages: String[]
# onCreate(Bundle): void
# onListItemClick(ListView, View, in, long): void
PopupScreen_Explore
- loc: LocationAccessLayer
+ hallways: String[]
+ POIs: String[]
+ lots: String[]
adapter: ArrayAdapter<String>
+ method(type): type
# oncreate(Bundle): void
+ buildOutputLists(String[]): void
# onListItemClick(ListView, View, in, long): void
+ onClickHallBtn(View): void
+ onClickPOIBtn(View): void
+ onLickPlotBtn(View): void
+ addToArray(ArrayList<LocationInstance>): String[]
contains
ParkingLot
- id: String
- name: String
- location: LatLng
- isPaidLot: boolean
- isAccessible: boolean
+ ParkingLot(String, LatLng, String, boolean, boolean): void
+ getLocation(): LatLng
+ getLatitude(): double
+ getLongitude(): double
+ getName(): String
+ getID(): String
+ isPaidLot(): boolean
+ isAccessible(): boolean
Hallway
- id: String
- name: String
- location: LatLng
+ Hallway(String, LatLng, String): void
+ getLocation(): LatLng
+ getLatitude(): double
+ getLongitude(): double
+ getName(): String
+ getID(): String
PointOfInterest
- id: String
- name: String
- location: LatLng
+ PointOfInterest(String, LatLng, String): void
+ getLocation(): LatLng
+ getLatitude(): double
+ getLongitude(): double
+ getName(): String
+ getID(): String
is a type of
is a part of
instantiates
<<Interface>>
LocationInstance
getLocation(): LatLng
getLatitude(): double
getLongitude(): double
getName(): String
getId(): String
DataHandler
- map: HashMap<Long, LocationInstance>
- _ourInstance: static volatile DataHandler1
- DataHandler(): void
+ getInstance(): DataHandler1
# readResolve(): DataHandler1
+ add(String, LocationInstance): void
+ find(String): boolean
+ get(String): LocationInstance
- getById(String): ArrayList<LocationInstance>
+ printMap(): void
+ buildTable(String[]): void
+ hashedValue(String): Long
+ size(): long
+ getAll(String): ArrayList<LocationInstance>
+ formatText(String): String
manipulates
return unique instance
1
1..*
0..1
0..*
*

Although model-view-controller (MVC) is more of an architectural pattern, we utilized it as a design pattern for accessing our DataHandler Singleton that was previously implemented in assignment 2. The view is the MapsActivity class as it is what the user interacts with. The controller is the LocationAccessLayer which takes in input from the maps activity class and makes the proper call to the DataHandler which relays the response back. The model takes up a large amount of the classes. It contains the Datahandler which manipulates and deals with the LocationInstances within it. Having these classes in the same part of the MVC model increased the cohesion of the app by keeping related classes together. An MVC pattern is very easy to maintain as testability is increased by being able to see what changes to the components do to the view. This pattern also greatly increases usability, readability, and reusability of the different components. The communication channels between the classes is as minimal as possible, however some refactoring may be able to reduce it even further.

*Source code was submitted in the .zip file

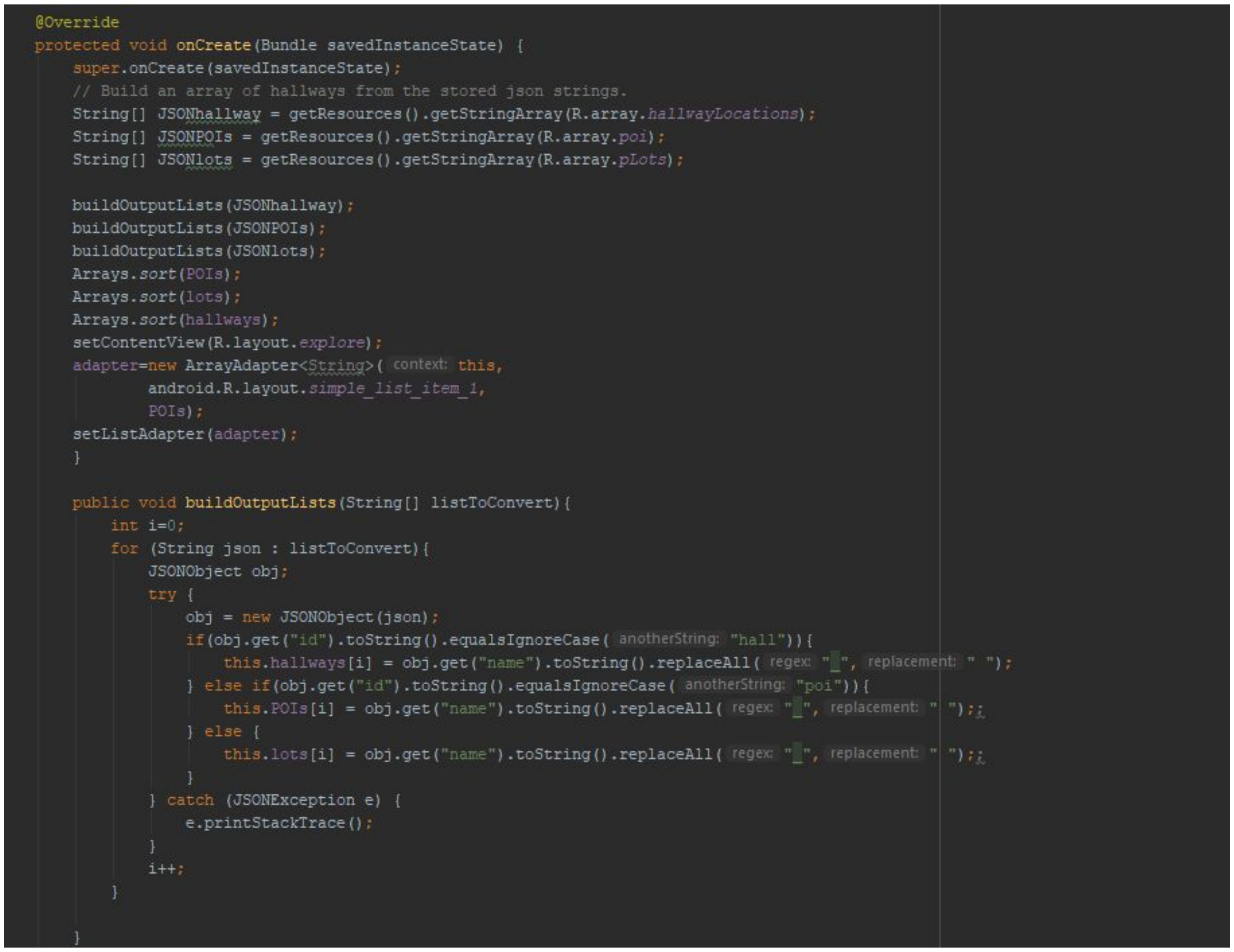
(8-2) Code Smell Discussion



```java
    public String[] hallways;
    public String[] POIs;
    public String[] lots;
    ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Build an array of hallways from the stored json strings.
        String[] JSONhallway = getResources().getStringArray(R.array.hallwayLocations);
        String[] JSONPOIs = getResources().getStringArray(R.array.poi);
        String[] JSONlots = getResources().getStringArray(R.array.pLots);
        int i = 0;
        for(String jsonString : JSONhallway){
            JSONObject obj;
            try {
                obj = new JSONObject(jsonString);
                hallways[i] = obj.get("name").toString();
                i++;
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        i=0;
        for(String jsonString : JSONPOIs){
            JSONObject obj;
            try {
                obj = new JSONObject(jsonString);
                POIs[i] = obj.get("name").toString();
                i++;
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }

        i=0;
        for(String jsonString : JSONlots){
            JSONObject obj;
            try {
                obj = new JSONObject(jsonString);
                lots[i] = obj.get("name").toString();
                i++;
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }


        setContentView(R.layout.explore);

        }
```

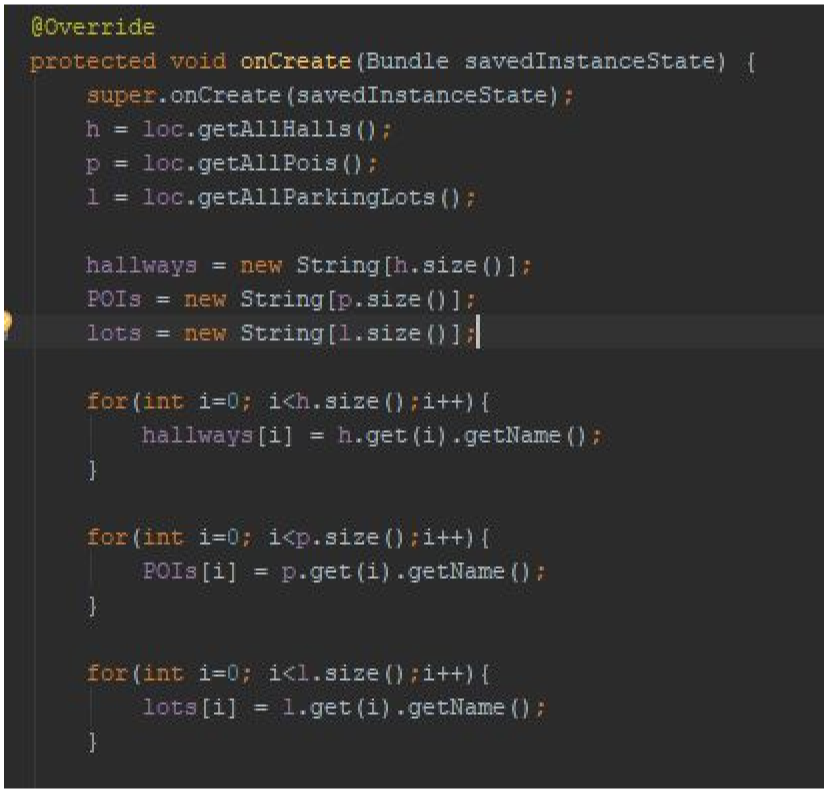
Smelly due to duplicate code.

Using extraction to remove similar functionality into a seperate method which can be called with the 3 different arrays.



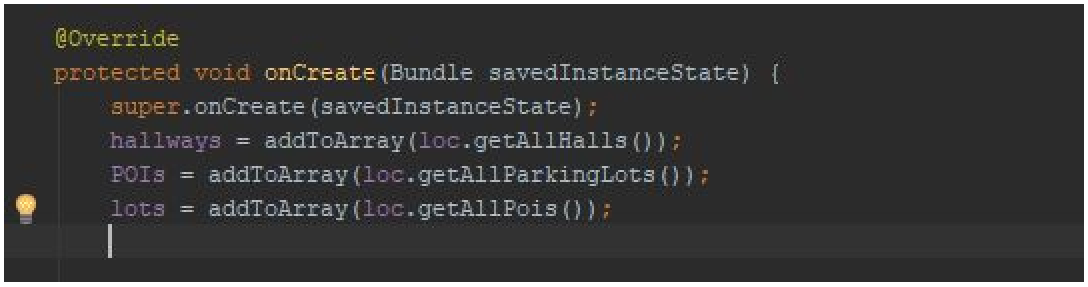
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Build an array of hallways from the stored json strings.
    String[] JSONhallway = getResources().getStringArray(R.array.hallwayLocations);
    String[] JSONPOIs = getResources().getStringArray(R.array.poi);
    String[] JSONlots = getResources().getStringArray(R.array.pLots);

    buildOutputLists(JSONhallway);
    buildOutputLists(JSONPOIs);
    buildOutputLists(JSONlots);
    Arrays.sort(POIs);
    Arrays.sort(lots);
    Arrays.sort(hallways);
    setContentView(R.layout.explore);
    adapter=new ArrayAdapter<String>( context: this,
            android.R.layout.simple_list_item_1,
            POIs);
    setListAdapter(adapter);
    }

    public void buildOutputLists(String[] listToConvert){
        int i=0;
        for (String json : listToConvert){
            JSONObject obj;
            try {
                obj = new JSONObject(json);
                if(obj.get("id").toString().equalsIgnoreCase( anotherString: "hall")){
                    this.hallways[i] = obj.get("name").toString().replaceAll( regex: "_", replacement: " ");
                } else if(obj.get("id").toString().equalsIgnoreCase( anotherString: "poi")){
                    this.POIs[i] = obj.get("name").toString().replaceAll( regex: "_", replacement: " ");;
                } else {
                    this.lots[i] = obj.get("name").toString().replaceAll( regex: "_", replacement: " ");;
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
            i++;
        }

    }
```

New implementation with calls to a single method (Extraction of functionality)

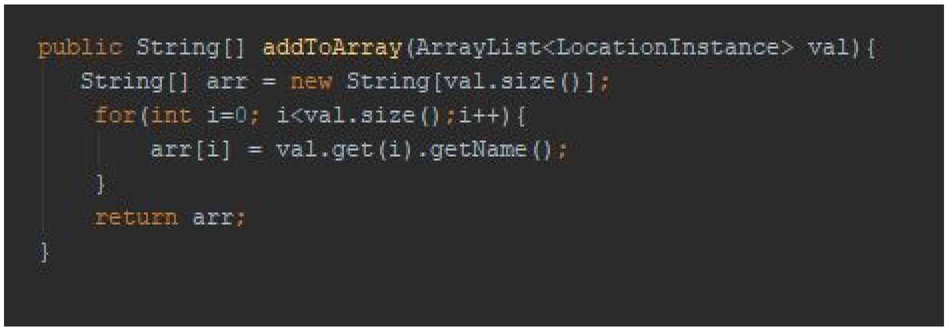


```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    h = loc.getAllHalls();
    p = loc.getAllPois();
    l = loc.getAllParkingLots();

    hallways = new String[h.size()];
    POIs = new String[p.size()];
    lots = new String[l.size()];

    for(int i=0; i<h.size();i++){
        hallways[i] = h.get(i).getName();
    }

    for(int i=0; i<p.size();i++){
        POIs[i] = p.get(i).getName();
    }

    for(int i=0; i<l.size();i++){
        lots[i] = l.get(i).getName();
    }
```

Before refactoring



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    hallways = addToArray(loc.getAllHalls());
    POIs = addToArray(loc.getAllParkingLots());
    lots = addToArray(loc.getAllPois());
```



```
public String[] addToArray(ArrayList<LocationInstance> val){
   String[] arr = new String[val.size()];
    for(int i=0; i<val.size();i++){
        arr[i] = val.get(i).getName();
    }
    return arr;
}
```

After refactoring

Through many refactoring sessions, we were able to find many instances of duplicated code and fix the code smell. We used the extract method and the pull up method several times to alter the code and reduce the duplication. The above code was the main code we tried to address. There are still a few instances of this, but this is due to the size of the app and the timeframe.

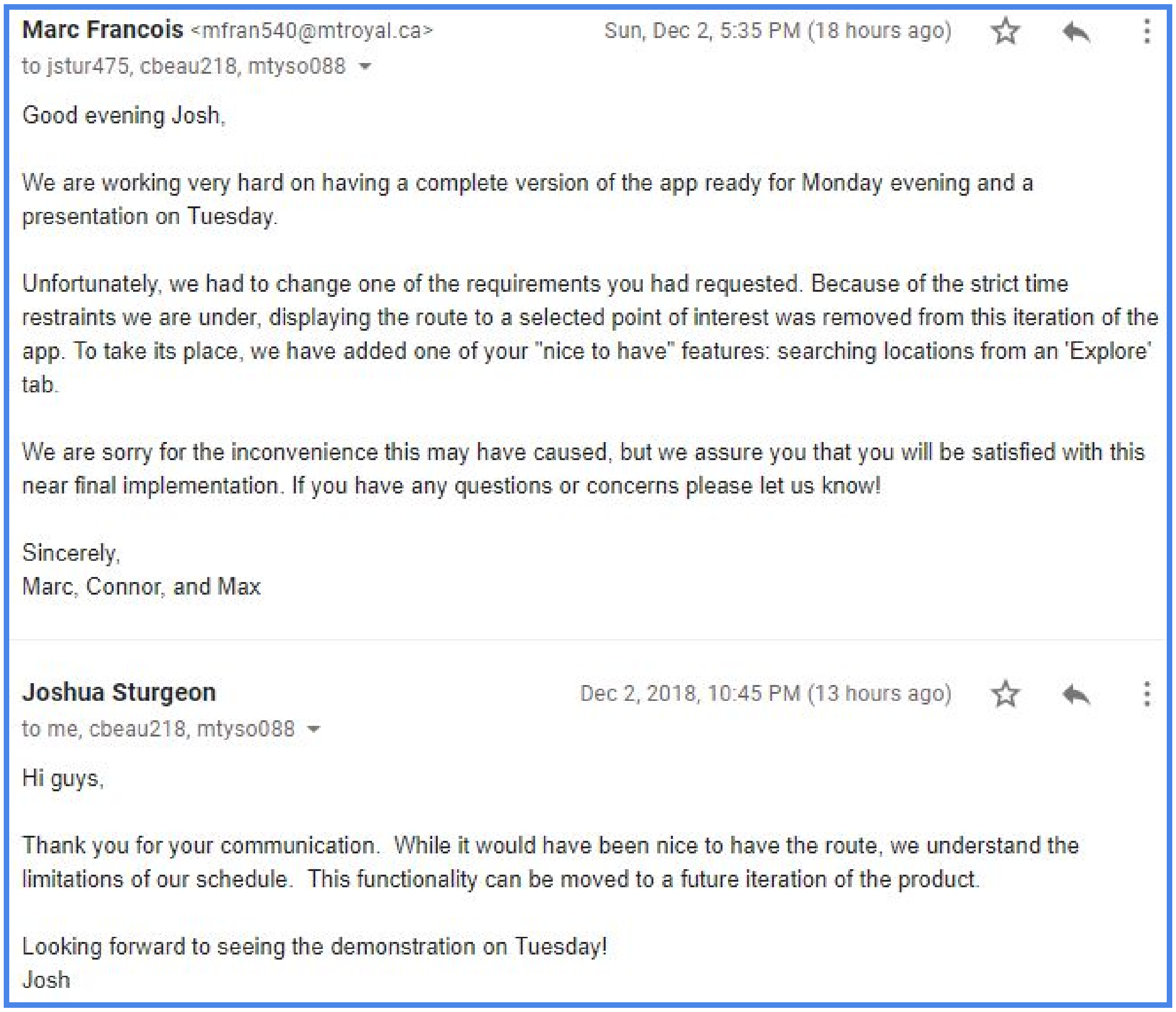
Deliverable #9

(9-1) Video of Completed App

Link: https://youtu.be/oyJrHWxUQGc

Deliverable #10

(10-1) Client Collaboration



Marc Francois <mfran540@mtroyal.ca> Sun, Dec 2, 5:35 PM (18 hours ago)
to jstur475, cbeau218, mtyso088

Good evening Josh,

We are working very hard on having a complete version of the app ready for Monday evening and a presentation on Tuesday.

Unfortunately, we had to change one of the requirements you had requested. Because of the strict time restraints we are under, displaying the route to a selected point of interest was removed from this iteration of the app. To take its place, we have added one of your "nice to have" features: searching locations from an 'Explore' tab.

We are sorry for the inconvenience this may have caused, but we assure you that you will be satisfied with this near final implementation. If you have any questions or concerns please let us know!

Sincerely,
Marc, Connor, and Max

Joshua Sturgeon Dec 2, 2018, 10:45 PM (13 hours ago)
to me, cbeau218, mtyso088

Hi guys,

Thank you for your communication. While it would have been nice to have the route, we understand the limitations of our schedule. This functionality can be moved to a future iteration of the product.

Looking forward to seeing the demonstration on Tuesday!
Josh

Over the past couple weeks, the project has gone through some ups and downs. The requirements had to be changed yet again due to time constraints and some functionality had to be removed and changed. Fortunately, our client was very understanding of the changes to their app and were willing to allow us to finish the functionality in another iteration of the app. During onsite discussions with our client, Joshua Sturgeon, we updated him on the progress of the app and confirmed with him that we were on the right track. Having consistent communication and transparency with the client was very important during the development of the app. The meetings were very short (<5 mins) because we and the client had many things to work on as the semester is coming to an end, but this time allowed us to communicate everything that we needed to. Involved in each meeting was Connor B., Marc F., Maxwell T., and Joshua the client.

Fortunately, our client was extremely understanding of the changes that were made to his app and we faced no challenges when communicating with him. If he were to not come to class or not respond to our emails in a timely manner, this would have impeded our ability to accurately extract requirements and ensure that our development was appropriate for the client’s needs.