Try This Place

ASSIGNMENT4

Group 7: Viet Nguyen, Ronan McCormack, Séan Doyle

Process Documentation

Before undertaking the development of the project we needed to decide on a methodology that would work best for us. As we are all new to software development we believed a methodology that would allow us to break the project into small, more manageable pieces would be necessary. We also felt we needed the ability to respond quickly to unpredicted issues that would arise from lack of experience in software planning. When taking these concerns into consideration we decided that the Agile methodology would work best for us.

Our feedback from Assignment 1 recommended we implement the functionality of login, viewing of friends' and personal recommendation when the phone is online. In order to meet this functionality we firstly split the project into a number of epics and then user stories. We each focused on a separate epic to begin which allowed us to pick off user stories and iterate our own sections, allowing the development of numerous functionality simultaneously, until they were ready for integration. The development was done in Android Studio and GitHub was used for version control.

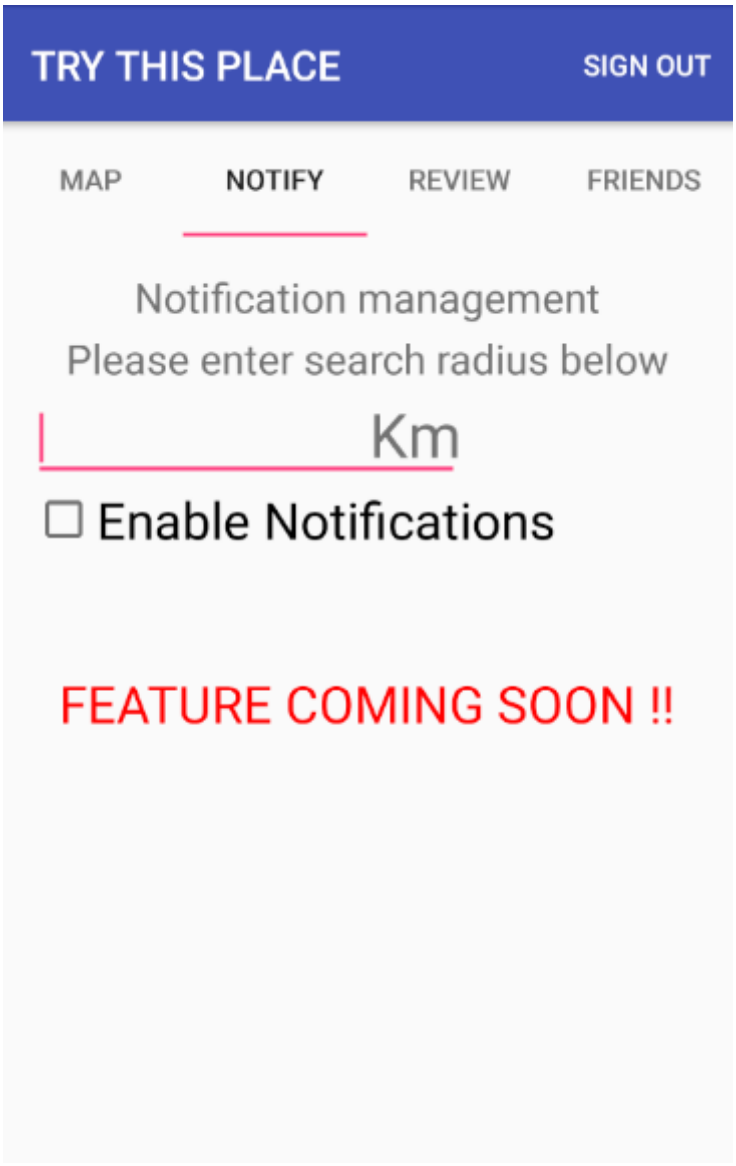
Sean – User Interface, Viet – login, friend's management and data base, Ronan – Google Place Picker and Maps.

Stand up meetings where held every day that there was not a clash with exams and this allowed us to update each other on progress by demonstrating working code, and also discuss potential problems encountered. As we are all in the same project room every day we found this practice worked very well and everyone was always informed of what each other were doing.

Following the completion of the individual components we began integrating them with each other. In order to do this we used pair programming. As we had originally all implemented different functionality we found that pair programming was very useful. It allowed integration of features with each other as the responsibility was split between the driver and navigator to keep a steady development flow, avoiding bugs, better structure and while doing that share more knowledge across the team. We found we were constantly talking and keeping everyone involved. We believe that although there was more than one of us working on the same thing, that this potentially saved us time overall. We discussed solutions and problems that we may not have come up with ourselves and as we were all learning as we went, pair programming was very useful in sharing tips we learned.

Notify Tab

We have plans to add the notification so that he user would get a notification anytime they were near pins and would be able to choose to enable or disable this feature as they wanted to. The reason we decided to leave the view in was because it gave a better look to the feel and functionality that the Application will eventually have , we also planned to try and get this in but unfortunately we ran out of time so decide the best thing to do would be to show how it would look on the screen and explain the rough idea of how we planned on implementing it so the idea is the user would have a radius set on them specified by the edit text box below when a reviewed location enters that radius a notification would be pushed to the screen that the user can choose to look at or ignore their choice. Below is the layout that will be used in the future



Remove Pins

We wanted to add a functionality that would allow you to edit and remove your own pins on the map if you didn't want them anymore or changed your opinion about the place specified by that pin. The reason we wanted to implement this function was so that the user can remove what they don't want or need any more this allows them to manage their pins and free up space in the database. This would be the next item to implement after the notifications functionality has been implemented. But unfortunately time did not allow us to implement this functionality.

Filtering

We wanted to be able to filter the tags to do this we would need to add a list of acceptable tags that the user could choose from when making a review. The review would then have a type assigned to it. We would most likely use a drop down list in review create containing all the allowed tags that the user could choose from where they must select at least one. They would then be able to display pins and reviews of certain types on the map depending on what they are looking for which would allow them to narrow down there search. This would be one of the final implementations needed for this design. After this we would maybe think about changing some other feature and refactoring the code to improve its maintainability and improving its overall speed and efficiency. In the future we will most likely think of new applications this app can be used in as well as new functionality that can be implemented. For this we would make a new version of the app and update it.

Testing decisions

We used two important API's in the development of this project one was Place picker and the other was Firebase. This was external code in which we interacted with to manipulate functionality they used and to allow our app to implement new functionality. WE were not able to Unit test this functionality as it was rather complex. We think the type of testing we need for this is Integrate testing. Which occurs after unit testing. This type of testing is adding multiple modules together and testing them. Due to this reason we were not able to unit test the APIs we communicated with. The Stage after integrate testing would be validate testing, where we would ensure everything is working as expected together and validate it against a known state.

Due to the fact that we couldn't do a unit test we did a lot of UI testing on the code to see what happens at various screen changes and how the data updates and is shown. By doing this it allowed us to manipulate and tweak the code to get it to display as we wanted. This allowed us to see when the next step wasn't clear to the user and to keep our app as simple as possible.

We ran a lot of test on signing in and using the sign out and sign in buttons on the screen. To ensure we were getting the correct account data and that the user could log out and potentially log in as someone different if they wanted to.

We also added the pins directly from the app with the content needed. We then checked the map to ensure review and pin where implemented correctly. We used the sync button to refresh the map when an update from friends or user has been added.

We tested adding a friend and pulling in there pins and reviews, by using the friends tab on the app adding the friend going back to the map hitting sync to see if the friends pins were loaded correctly . We then went back to the friends tab and removed the friend by clicking on the icon. After this we returned to the Map tab and clicked again and seen that the friends reviews and pins have been removed.

We used toast to display information when waiting for screen to lad in the case of logging in or searching a location using place picker.

We added limits to review lengths and tested them on this App by entering reviews above that limit to ensure that it wasn't submitting them and that it was showing a toast saying review too long ! . This is to inform the user they need to reduce their review length.

We can also check on the app if the email entered isn't in the database or if it is an invalid email to some extent which is explained later on. If an email isn't in the database it means they do not have an account stored in the database by doing this it can be clearly seen if the user is in the database or not. If an invalid email is enter a toast will pop up saying invalid email to the user so that they know they have made a mistake as the edit text will clear upon clicking add.

Test cases

Latitude and longitude

Latitude can be between $-90°$ and $90°$, Longitude can be between $-180°$ and $180°$. We need to capture the latitude and longitude of a location so that we can place it on the map correctly. Latitude and longitude will only allow values inside there specified ranges, for example if I give the following latitude and longitudes it will produce the following values

	Latitude	Longitude		Latitude	Longitude
First input	$89°$	$120°$	Result of input	$89°$	$120°$
Second input	$91°$	$150°$	Result of input	$89°$	$150°$
Third input	$-45°$	$-185°$	Result of input	$-45°$	$-175°$
Fourth input	$-93°$	$182°$	Result of input	$-87°$	$178°$

To ensure that we are confident in the ability of inbuilt functions to handle the data to the best of its ability. We need to check the pinData.Coord field in the Pindata class of the project. We need to do this to ensure that the data being passed is behaving as we expect and will behave as other cord and latlng variables do.

To do this test we needed to check a few tasks such as below

Test number	**Latitude**	**Longitude**	**Check performed**
1	$60°$	$3°$	Checking normal value check
2	$-90°$	$-180°$	Checking negative boundary
3	$90°$	$-180°$	Checking another boundary condition
4	$-90°$	$180°$	Checking another boundary condition
5	$90°$	$180°$	Checking positive Boundary condition
6	$-91°$	$-181°$	Checking values beside the boundary
7	$91°$	$181°$	checking values beside the boundary
8	$89°$	$179°$	checking values beside the boundary
9	$-89°$	$-179°$	checking values beside the boundary
10	$0°$	$0°$	checking values at zero
11	$120°$	$240°$	checking values outside the positive boundary
12	$-120°$	$-239°$	checking values outside the negative boundary
13	$-145°$	$270°$	checking values outside both boundaries
14	$145°$	$-270°$	checking values outside both boundaries

The code used will test all of the above 14 cases and get the expected values. To do this we simple built a test in android studio set up the variables and repeated the test for all of the required cases as above to ensure that the data was being received as we anticipated it to.

Below I will include the code used to set up the initial values and two of the ran tests there are more in this test case but they are very similar so I will not add more than these two tests

```
private PinData pinData;
private LatLng ExpectedLatLng;
private LatLng SetLatLng;
private latlngCoord newlat;

@Before
public void setUp() throws Exception {
    newlat= new latlngCoord(60,3);
    SetLatLng = new LatLng(60,3);
    pinData = new PinData();
    pinData.setReview("Great fun ,had a good time check out the cathedral");
    pinData.setCoord(newlat);
    pinData.setType("bar");
    pinData.setAddress("Germany Koln");
    pinData.setTitle("Germany trip");

}


@Test
public void createMarkerLatLngTestNormalValue() throws Exception {
    getPinHeaderCallback call = new getPinHeaderCallback();
    ExpectedLatLng = call.createMarkerLatLng(pinData);
    assertEquals("latidue and longtitude inside boundaries", ExpectedLatLng, SetLatLng);
}

@Test
public void createMarkerLatLngTestNegativeBoundary() throws Exception {
    newlat= new latlngCoord(-90,-180);
    SetLatLng = new LatLng(-90,-180);
    pinData.setCoord(newlat);
    getPinHeaderCallback call = new getPinHeaderCallback();
    ExpectedLatLng = call.createMarkerLatLng(pinData);
    assertEquals("latidue and longtitude at negative boundary", ExpectedLatLng, SetLatLng);
}
```

Review length test

In this test we are checking if the review has been made too big to submit or not. When we are displaying the review on the pins we want it to just be a quick line saying if it was good or bad. This is the basic idea of the review we want to keep them short and sweet, we don't want the users to be writing to much. We set the review to be 40 characters long and decided that this would be a good value as it gives a good sentence or so about the place. For this we needed to test the string length Once again we ran a number of tests. To see if the String was acceptable or not the tests are as below

Test number	String length checked	Check performed
1	String Length > 40	Checked to see if the string length was less than 40
2	String Length = 40	Checked the string length against the max characters value
3	String Length = 39	Checked the string length when just below
4	String Length = 41	Checked the string length when just above 40
5	String Length >>> 40	Checked the string Length when much greater than 40

We do not need to check when the review length is zero as this can never happen, because the code will not allow the user to submit a review until they have written something in the review box. Below I have included the code to setup and run the first two tests. Once again since the code is repetitive I will not include it all.

```
private String review;
private int maxSize =40; // the maximum amount of characters a review can have
private boolean check;
private boolean set;

@Before
public void setUp() throws Exception {
    review ="the cat went meow ";
    set = true;
}

@Test
public void checkReviewUnderLimitTestReviewunderCount() throws Exception {
    FragmentReviewManager test =new FragmentReviewManager();
    System.out.println(review.length());
    check = test.checkReviewUnderLimit(review , maxSize);
    assertEquals("the review is under 40 characters this is acceptable", check, set);
}

@Test
public void checkReviewUnderLimitTestREviewOverCharacterCount() throws Exception {
    review= " the dog chased the cat up a tree  and barked  a lot.....................";
    System.out.println(review.length());
    set = false;
    FragmentReviewManager test =new FragmentReviewManager();
    check = test.checkReviewUnderLimit(review , maxSize);
    assertEquals("charcter count is above 40 not allowed", check, set);
}
```

Valid Email Test

When the user's needs to add a friend the database initially checked if the friend was there regardless of the string it got , this is fine when the database is small but if the database becomes too large this will lead to really long delays. We implemented a method to check common mistakes in an email if the last character is a dot (' . ') then we know that the code has some error in it as all emails end with their top-level domain name such as .com , .uk , .ie , etc . Example below

UCD.student@ucdconnect.ie

RandomPerson@hotmail.com

MadeUpEmail@gmail.uk

If an email has a dot at the end of their top-level domain name it incorrect. We also added a check that if there is no @ symbol in an email address entered then it is an invalid email as all emails use an email address such as above . We do not need to check if the user tried to add a null string and add it as this cannot happen below is the test cased we checked for

Test number	Email entered	Check performed
1	UCD.student@ucdconnect.ie	Checked against a valid email
2	UCD.student@ucdconnect	Checked email missing top level domain name
3	UCD.studentucdconnect.ie	Checked email missing @ symbol
4	UCD.student@ucdconnect.ie.	Checked email added . character at end of email
5	UCD.studentucdconnect.ie.	Checked email missing @ symbol and added . at the end

We do not need to check if the email is null as an email can never be entered as a null variable as the add button will not accept it. Below we have included the code with the first two tests, we will not need to

```
private String email;
private boolean emailValid;
private boolean valid;


@Test
public void isEmailValidTestValidEmail() throws Exception {
    FragmentFriendManager emailCheck = new FragmentFriendManager();
    email="UCD.student@ucdconnect.ie";
    emailValid = emailCheck.isEmailValid(email);
    valid = true;
    assertEquals("Email is valid genuine email entered", emailValid, valid);
}

@Test
public void isEmailValidTestEmailMissingTopLevelDomainName() throws Exception {
    FragmentFriendManager emailCheck = new FragmentFriendManager();
    email="UCD.student@ucdconnect";
    emailValid = emailCheck.isEmailValid(email);
    valid = true;
    assertEquals("Email is valid but email is missing the top-level domain name", emailValid, valid);
}
```

Testing Firebase Integration

We faced major difficulties in carrying out unit testing related to the integration of the Firebase real-time database with logic implemented in the FirebaseConnection class, including the following methods. Although this lies outside of the scope of unit testing as database interactions is more at the system level, it is code that we have written which couldn't been testing in the simple Junit – Assertion framework due to high coupling with a real Firebase database that couldn't be mocked, i.e. using Mockito framework.

The following methods used to interact with Firebase are:

```
public void createPin(PinData pindata) {
    DatabaseReference keyRef = pushRef.child("Pins").push();
    keyRef.setValue(pindata);

}
```

The method createPin(), which takes a PinData object and stores this in the Firebase Database at the node child specified by reference root/User UID/Pins in the form of a json data structure. The only way to test if this is stored correctly is either by inspecting the database, or to implement another method to get this PinData from the database by specifying the same reference root/User UID/Pins/Pin Push ID, however the way Firebase sends back data is it does not return the data, rather it attaches a listener to the node, which triggers an asynchronous onDataChange() callback, where the pin contents are extracted from a DataSnapshot. This behaviour does not match with our understanding of a Junit test whereby the return value of a method is asserted to be equal to an expected value. In this case, there is no return value, but a callback method is called.

```
public void setUser(User mUser) {
    pushRef.child("UserProfile").setValue(mUser);
}
```

This method carries out the same functionality as createPin(), however instead of PinData, a User object is added at root/User UID/UserProfile, thus testing of this method runs into similar problems as createPin().

```
public void removeFriend(String friend){
    pushRef.child("Friends").child(friend).removeValue();
```

This method deletes the User object from root/User UID/Friends/friend. Again to test this, a new method has to be written to check if the friend data exists before removeFriend() is called and no longer exists after removeFriend() is called, which also is an asynchronous event that triggers a callback. What more is that to test the method, we have to code further methods which we do not require in the application.

```
public void checkFriendisUser(User friend){

    FriendListener friendListener = new FriendListener(pushRef, friend);

    rootRef.addListenerForSingleValueEvent(friendListener);

}
```

This method checks if an email address entered exists in the Firebase database ie. the friend exists, and on the return of the list of Firebase users to test the email input against, if an account matches, a nested call within the onDataChange to add the friend to list of User friends is sent to Firebase.

```
public void getPins(PinHeaderCallback callback, GoogleMap googleMap) {

    PinDownloadListener pinListener = new PinDownloadListener(callback, googleMap);

    pushRef.child("Friends").addListenerForSingleValueEvent(pinListener);

}
```

The getPins() method attaches a listener to the User's Friends node to receive the list of friends the user has. OnDataChanged(), a method is called to get all of the Friends' and the User's pins, passed into a callback method to display on the Google MapView.

To ensure that when called, these methods actually behaved as expected, due to real time nature of Firebase, whenever these methods are called, we observe real-time changes to the database i.e removal, adding of child nodes i.e. when User profiles, friends, Pins are added to the database and that no unexpected data, nodes were created on the database side. On the receiving side, when getting data from Firebase, mainly in the form of the Friends List and displayed marker pins on the Google MapView, the testing of functionality was tested by inspection and prolonged use of the application.