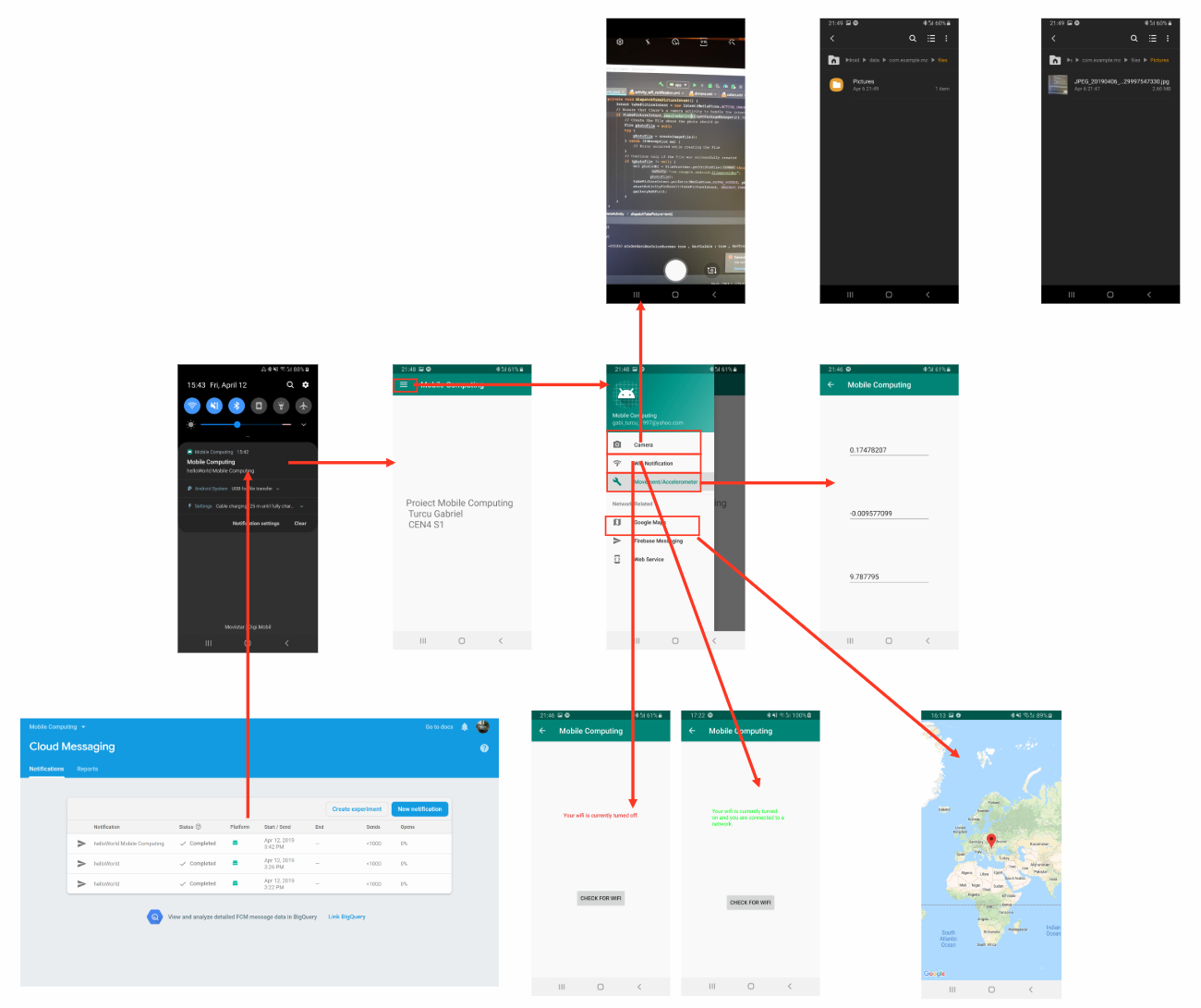
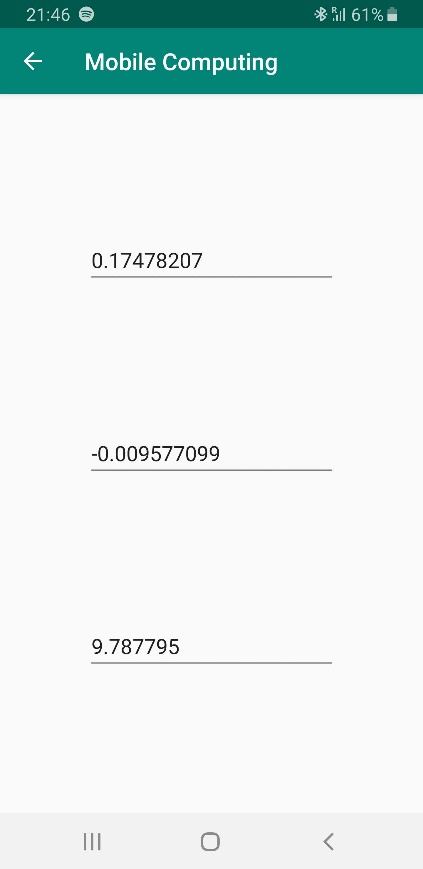
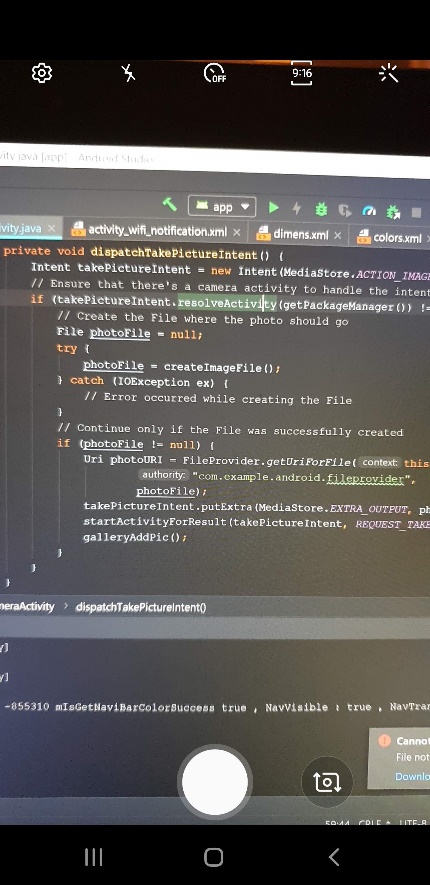
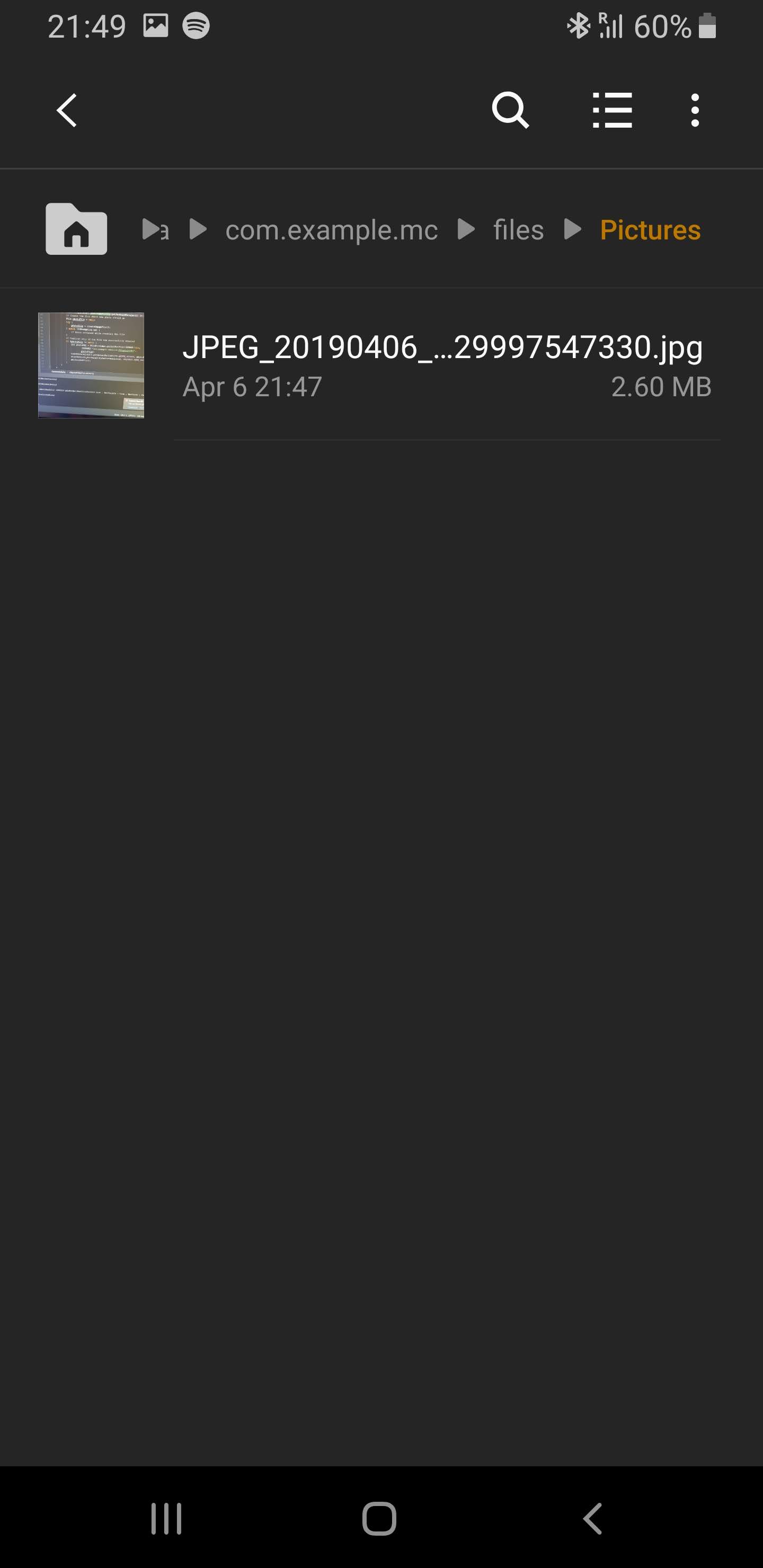
**Mobile Computing Report “helloAndroid”**

**Student: Turcu Gabriel**

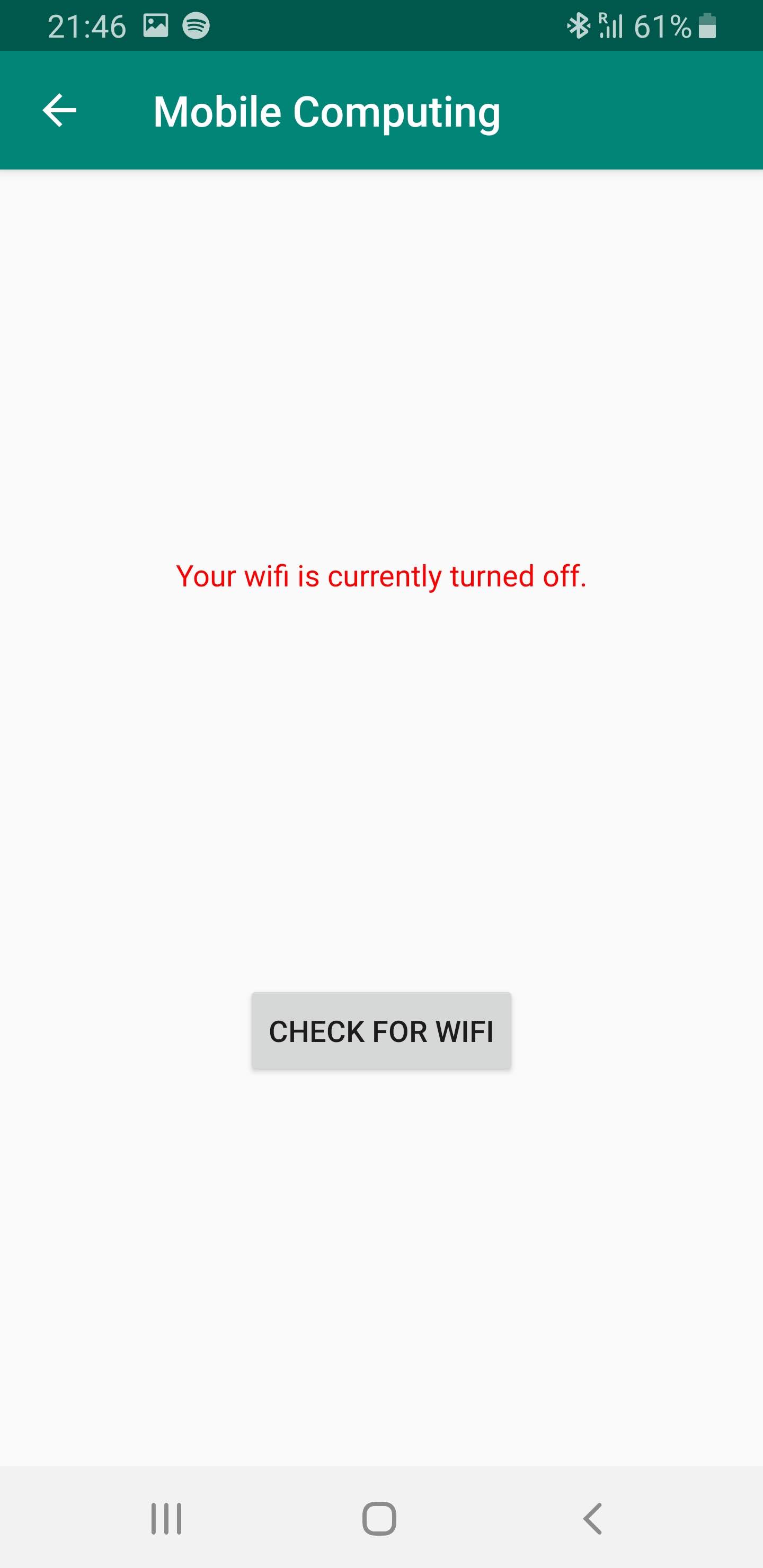
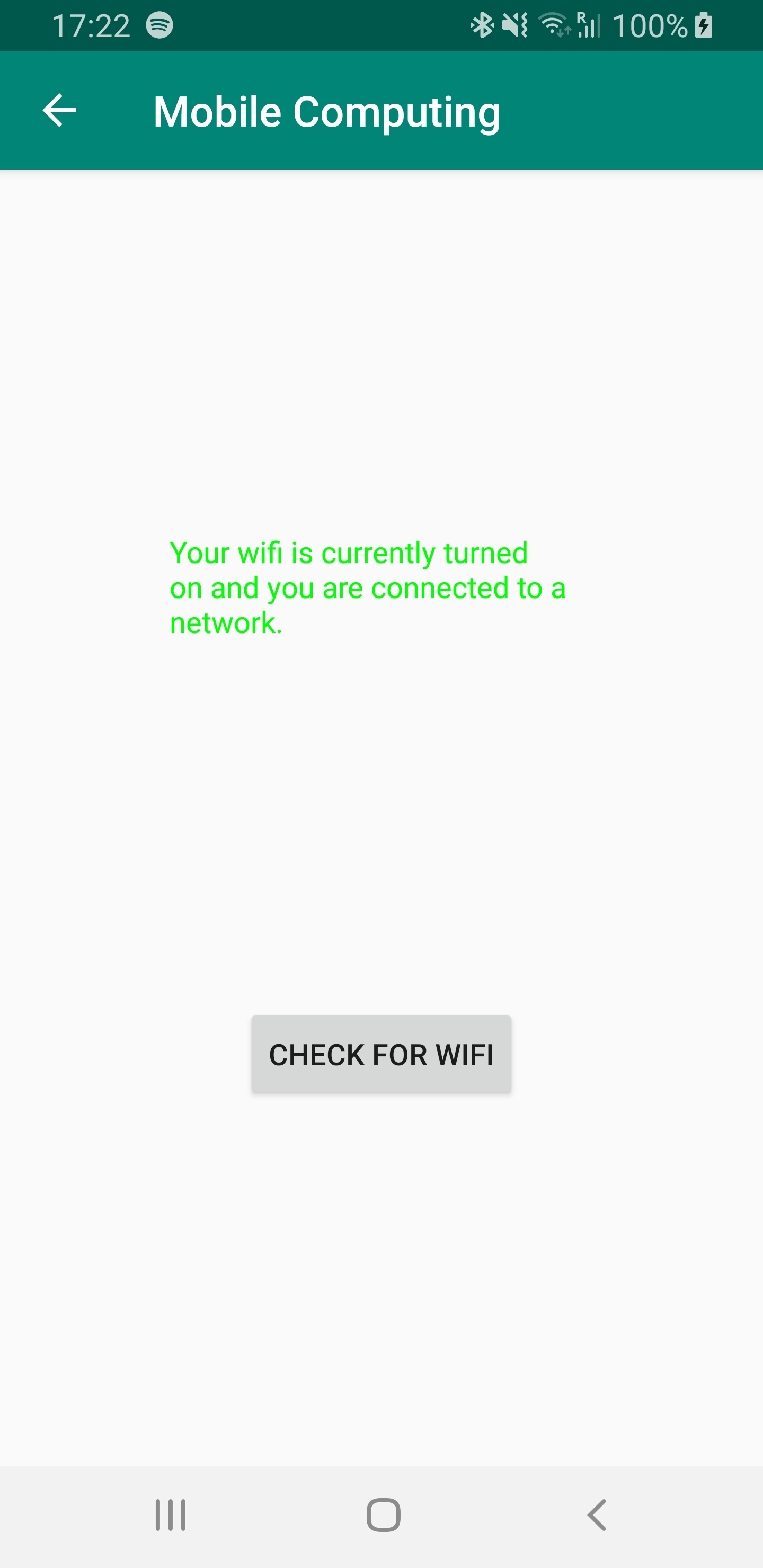
1. About the project
   * The project is a relatively simple Android app that includes all but one of the required features but whose features don’t necessarily have anything to do with each other because I wasn’t able to come up with something that actually combines all of them into something that makes sense, hence the name helloAndroid.
   * For this app, we are using the following built in sensors:
     1. Accelerometer for the accelerometer display activity.
     2. Location and mobile data/wifi for the Maps service.
     3. Wifi for the wifi notification activity.
     4. The camera for the activity that allows the user to take a picture that will be saved on the device.
2. Application Wireframe



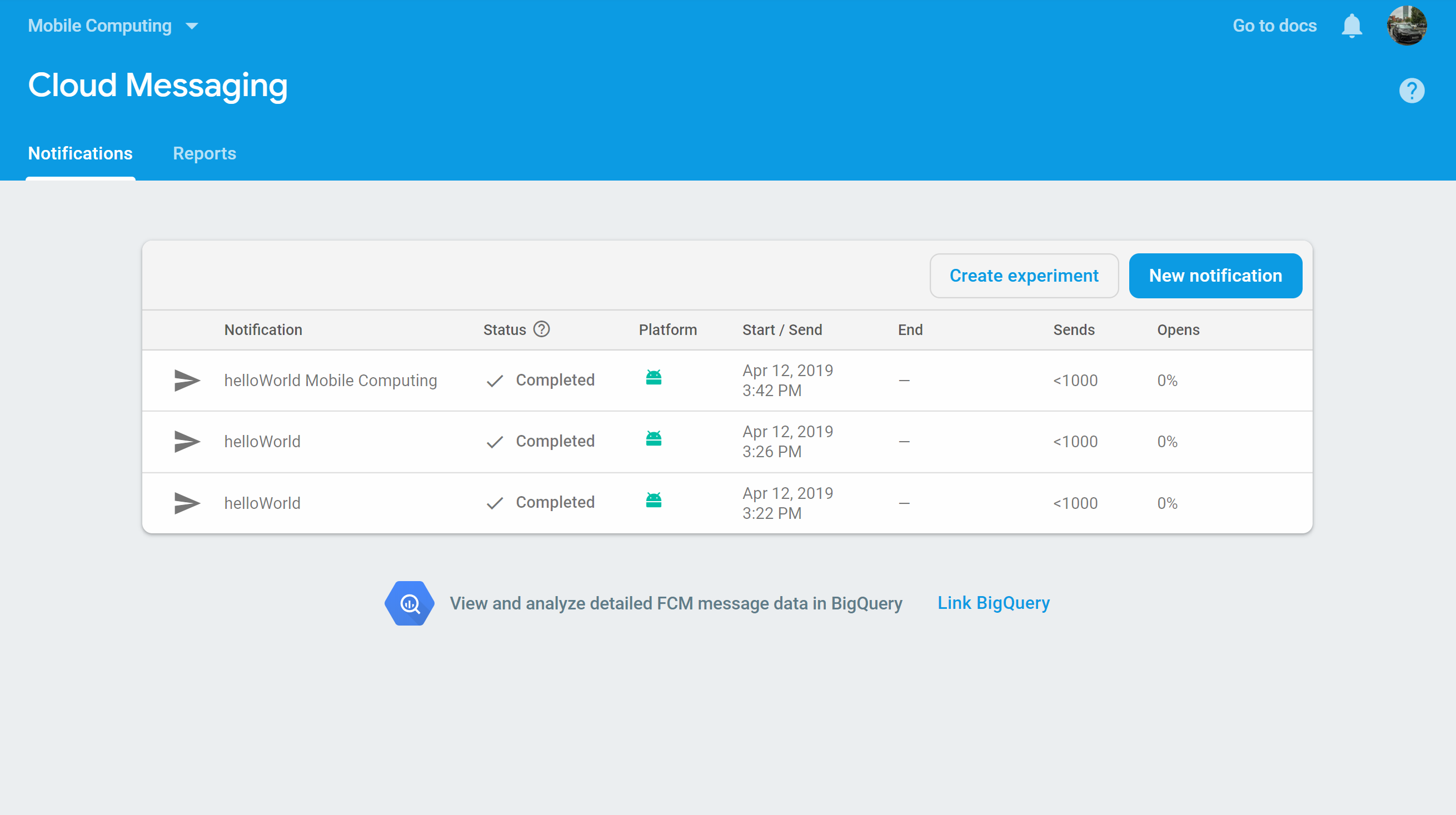
1. Server side functionality:
2. Movement(using the Accelerometer):
   * Github link to the **AccelerometerActivity**: <https://github.com/MickeyGT/Mobile-Computing---A4S2/blob/master/MC/app/src/main/java/com/example/mc/AccelerometerActivity.java>
   * We are going to showcase the accelerometer functionality in our app by displaying the information retrieved from the accelerometer sensor in an activity.
   * In order to access the information from the accelerometer, we need to create a **SensorManager** and **Sensor** object and register a listener that will use a **broadcast** **receiver** to listen for changes in the values reported by the accelerometer, we do this in the **onCreate** function of the activity.
   * Now that we have a listener registered, we need to have a function that will react accordingly to the changes in values. This will be the **onSensorChanged** function that will be automatically called every time the sensor has any change to report. In this function, we can modify the **textViews** that display the X, Y, Z axis of our accelerometer.
   * 
   * One thing that we should also do is make sure to **unregister the listener** in the **onPause** function and register the listener again in the **onResume** function. We don’t want to waste resources by keeping the listener on when we don’t need it.
3. Images(using the Camera):
   * Github link to the **CameraActivity**: <https://github.com/MickeyGT/Mobile-Computing---A4S2/blob/master/MC/app/src/main/java/com/example/mc/CameraActivity.java>
   * We are going to showcase the camera’s functionality in our app by allowing the user to take a picture using the onboard camera that will be saved to their device.
   * In order to take a picture and save it to the device, we first need **permissions** from the user to access the **camera** and to **read** and **write** files to the phone’s internal storage. In order to do this, we have to add a couple of lines in the **AndroidManifest** about using the camera feature and requesting the read and write permissions. <https://github.com/MickeyGT/Mobile-Computing---A4S2/blob/4b0f4e604de178278aecf617ad634cb939623cc5/MC/app/src/main/AndroidManifest.xml>
   * Now, we when the user enters the **CameraActivity**, we want the camera module to open so that they could proceed with taking a picture which would then be saved on their device. In order to accomplish this, in the **onCreate** function of the activity, we call the function **dispatchTakePictureIntent** which does exactly what you might suspect, it dispatches the intent to take a picture using the camera. Before it starts the activity respective to that intent, it first checks if there is a camera activity that is able to resolve that intent. If there is, it creates a File object using the **createImageFile** function, it starts the activity that uses the camera to take the picture and then it calls the **galleryAddPic** that creates and broadcasts an intent of the **ACTION\_MEDIA\_SCANNER\_SCAN\_FILE**type.
   * 



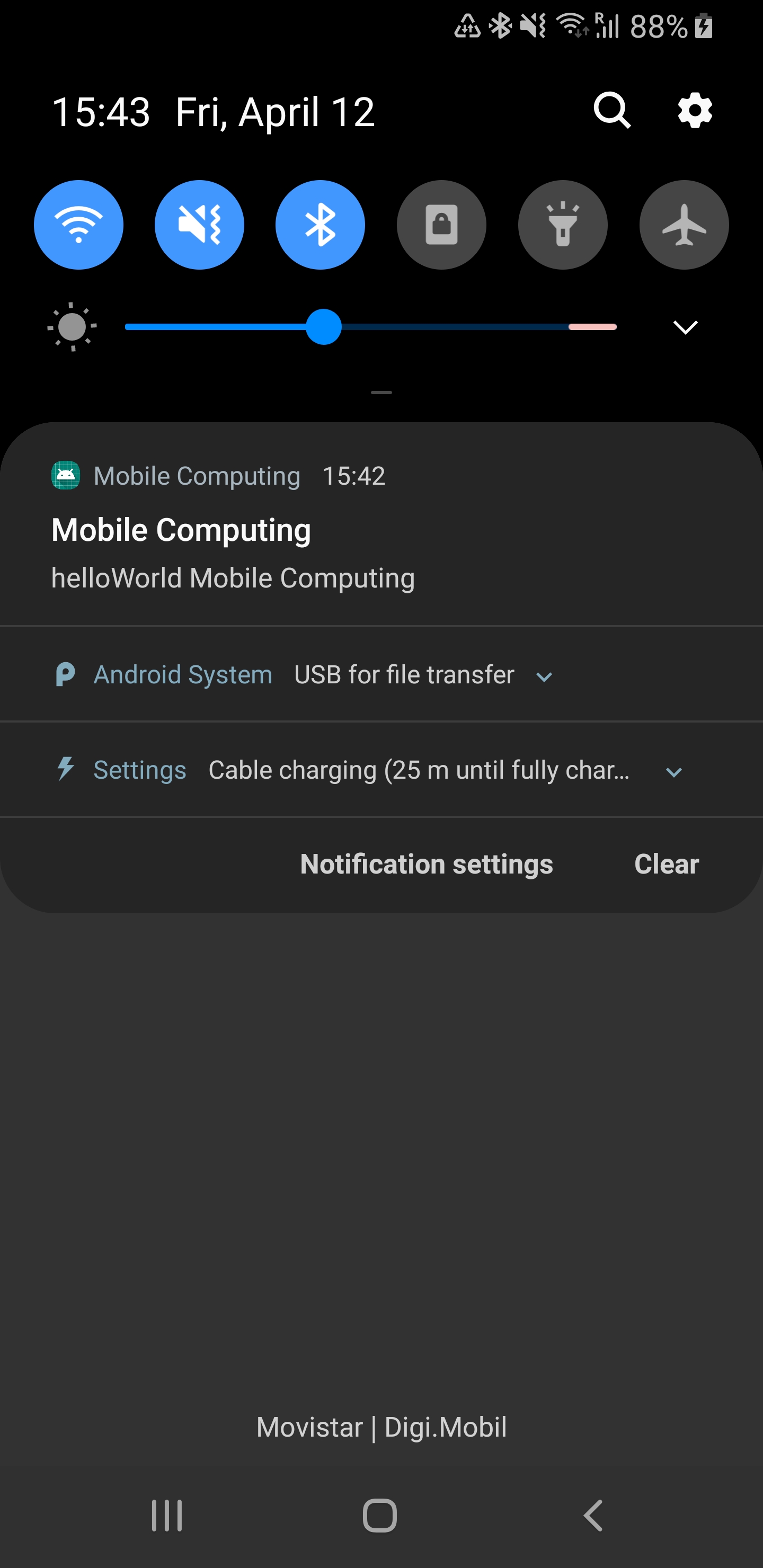
1. Wifi Listener:
   * Github link to WifiActivity: <https://github.com/MickeyGT/Mobile-Computing---A4S2/blob/master/MC/app/src/main/java/com/example/mc/WifiNotificationActivity.java>
   * We are going to showcase the wifi module in our app by having an activity that checks whether or not the wifi is on and actually connected to a network.
   * In order to do this, our activity will have a **textView** and a **button**. The button will act as a refresh button for the checker. The **textView** will display a text saying whether or not the phone is connected. We will also run the function that checks if the phone is connected when we first enter the activity.
   * We will create a function that checks if the phone is connected to a wifi network and that alters the **textView** accordingly. This function has a **connectivity manager** from which we can get **the active network**. We can then check if the **activeNetwork** is **not null** and if the activeNetwork.**isConnectedOrConnecting** and if the activeNetwork is of the **Wifi type** (see lines 39-41).
   * After we do the above check, if it is **true** then we modify the content of the textView to reflect that and change the color of the text to **green**. If the above condition is **false**, we modify the text to reflect the fact that the phone is not connected and change the **color** of the textView to **red**.



1. Notifications:
   * Github link to the service responsible for firebase notifications: <https://github.com/MickeyGT/Mobile-Computing---A4S2/blob/master/MC/app/src/main/java/com/example/mc/MyFirebaseMessagingService.java>
   * In order to send a notification using **firebase**, we need to configure a notification from the Cloud Messaging website.



* + Now, in order for us to receive notifications, our app needs to have **firebase** added to it. Once we did that using this tutorial <https://firebase.google.com/docs/android/setup>, we needed to have a **service** that listens for notifications coming from the **firebase web service**.
  + This **service** needed to override the **onMessageReceived** function and create a notification using the text from the **message**. In this function, we create the notification channel using the **createNotificationChannel** function and then build a notification using the text retrieved from the message. We then show the notification to the user.



1. Maps and Directions:
   * Github link to MapsActivity: <https://github.com/MickeyGT/Mobile-Computing---A4S2/blob/master/MC/app/src/main/java/com/example/mc/MapsActivity.java>
   * Configuring and setting up the **MapsActivity** was quite simple since Google has made it very easy to import a new Activity that contains **Google** **Maps**. I’ve followed this tutorial: <https://developers.google.com/maps/documentation/android-sdk/start>
   * From there, adding a new marker is very easy since all we have to do is create a **LatLng** object with the coordinates of the place we want to mark and then add a marker to the map with that position using the **addMarker** function and then we tell the camera to move to that position when the map is first launched using the **moveCamera** function.

