Technical Documentation

Android Application

Code Base

The code can be found in two locations either in a zipped folder in the handover documents or at github.com/WaiNZ/RiverWatch-Android.

Updating the Android App

This guide will give you a brief overview of how you can upload an app to the Play store or update the current one. For more information please visit the official documentation at https://support.google.com/googleplay/android-developer/answer/113469?hl=en and https://developer.android.com/distribute/googleplay/start.html

The keystore and account details can be found in the handover documentation in the app store folder.

Note: Things to keep in mind before publishing:

1. The package name for a app cannot be changed once it is uploaded, therefore make sure you choose a appropriate option
2. Part of uploading is signing your app, and any future updates to the apk has to have the same signature. Therefore it is crucial to keep your keystore in a safe place and the two passwords that are needed to use it.

Uploading a new APK

1. Generate a signed apk
2. Navigate to https://play.google.com/apps/publish/
3. On the main page in that upper right there is a button **Add New Application**
4. This will bring up a dialog to enter the name of the app, select language. Enter that information and then either select **Upload APK** or **Prepare Store Listing**
5. Fill out each of the **APK**, **Store Listing**, **Pricing & Distribution** and **Content Rating** sections
6. Click on **Publish App**, in the upper right

Updating the Current App

As stated above to update the app it must have the same package name, in our case **com.vuw.project1.riverwatch**, and be signed by the same keystore. The keystore can be found in the same directory as this document.

1. Update the versionCode and versionName(What is displayed in Google Play), found within *build.gradle*
2. Generate your signed apk
3. Navigate to https://play.google.com/apps/publish/
4. Click on RiverWatch 2.0 and navigate to the **APK** section
5. Click on **Upload new APK** and fill out the required fields, then click **Submit update** in the upper right.

Google Maps API Key

In order for the app to connect to the google API services it needs an API key that is sent with requests so that google knows the request is coming from a legitimate source. This key is identified in the AndroidManifest.xml inside the application tag with a metadata tag as such

```
‘<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="API key goes here" />
‘
```

This defines the API key the app will use to connect to the google play services. However google play services also needs to restrict the usage of these API keys to versions of the app developed by approved sources. So members of a team will need to be added to the riverwatch project on the google api manager console, To do this a current member of the project needs to add you. The email of wainz is currently an owner so this email can be used to add others to the project. Once you have been added you will need to then add the SHA-1 signing-certificate fingerprint of every device the apps are being developed from in order for the google play

services to accept it. This can be done in the console manager when you click on the credentials tab and open up the Android key edit screen.

Application Structure

Each of the following sections will cover how a person could take up development and carry on with the app.

Incident Reporting

The incident report functionality is relatively straightforward as there are only 2 activities, the ReportActivity (Where the camera functionality is), and the ReportTabbedActivity where the user views and writes about the incident they are reporting.

All of the report classes excluding the ReportActivity are in the Report_Functionality package

ReportActivity
This is the activity for when the user wants to take a picture of an incident there are two main parts.
The first part is handling the camera which deals with, displaying the images that are being captured by the device camera, dealing with rotation, and actually taking the picture and saving it to the device. For creating a camera there is a method for a safe camera open which is required to ensure that the camera can be accessed and won't crash the app. Once the camera has been opened the custom preview view is passed the camera which it uses to display the images from the hardware camera.
Reacting to rotation of the phone is handled in 2 seperate classes. The first is the Report Activity which figures out the new orientation and deals with animating the camera button.
The second is the preview class which figures out the sizes that it needs to be displayed based on what the rotation is and the quality of the camera/phone size.
Taking a picture is pretty straightforward, the only slightly complex part is how the app uses an object (SaveImagetask) which extends Asynchronous task to capture the image and save it to a riverwatch folder on the device.
The second part of this activity is the google api which is separated from the rest of the sections by comments and is just a bunch of relatively straightforward methods that handle connecting and disconnecting from the google api.

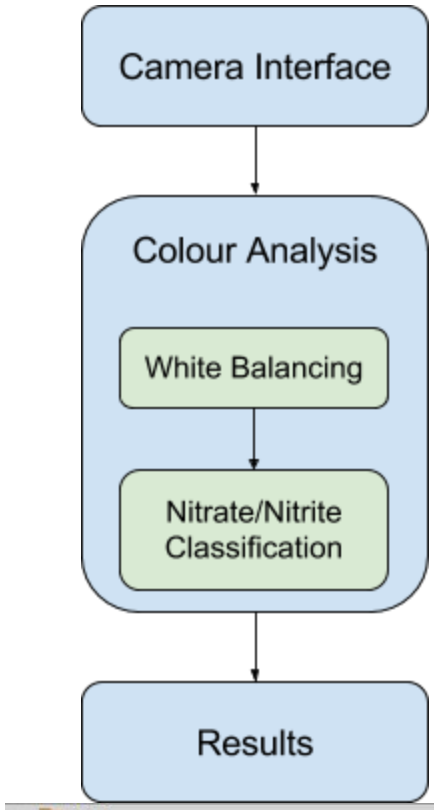
Important things to note is that this activity only sends the filepath of the of the image to the next activity as well as only the double values of the Latitude and longitude of the location (if there was one)

ReportTabbedActivity
This is the activity where the user enters the details of the incident and can also view the map and the image. As this activity is tabbed, 3 classes make up the functionality.

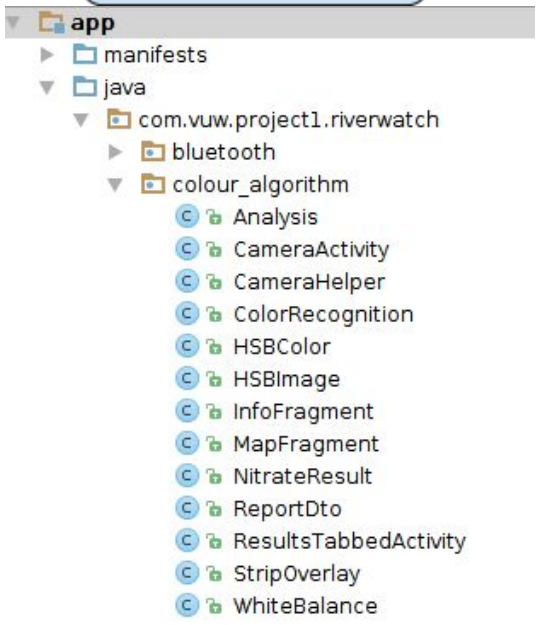
- ReportTabbedActivity - This deals with managing the tabs and the submission of all the details
- ReportMapFragment - This deals with displaying the google maps and the api service requests
- ReportInfoFragment - This deals with showing the image to the user and displaying the text fields and passing the information to the ReportTabbedActivity when submission is reached.

Nitrate/Nitrite Testing

The nitrate/nitrite testing functionality can be split into three sections, the camera and camera overlay, the algorithm and the results section.



Camera Interface
This section is made up of the CameraActivity, CameraHelper and StripOverlay classes. The main class being the CameraActivity, the main methods in this class are the *onCreate*, *surfaceChanged* and the field *picture*. On create gets called when the activity first gets created, this sets up the camera instance, the camera helper and adds a listener to the capture button which calls 'camera.takePicture(null, null, picture). The method *surfaceChanged* is called everytime something happens with the camera preview, ie if it changed rotation, and will do appropriate functions to change which is necessary. *Picture* defines a method *onPictureTake,* this is where we cut out the two rectangles in the overlay. We create three bitmaps, the two cutouts and the middle section which are then passed on to the next activity along with a URL to the image.



Algorithm
This is made up of *Analysis, ColorRecognition, HSBColor, HSBImage and WhiteBalance* classes. The algorithm starts when the *ResultsTabbedActivity* is created, at line 216 a new analysis task(line 174) is created and starts to execute. The *AnalysisTask* has a method *doInBackground* which creates a new *Analysis* class and calls the method *processImages* then

puts the results in a bundle to be passed on to the *onPostExecute* method.

When *processImages* gets called the algorithm will first white balance the cut outs by using the middle section. During white balancing, the algorithm extracts the RGB values of each pixel into seperate lists, which it then removes the lower and upper quartiles then averages the remaining values to find our estimated white point. We then find the multipliers in each of the red, green and blue channels which when applied to the white point will bring the white point to true white. Which we can then apply to each pixel in the two cutouts.

With the now white balanced images the algorithm extracts the median, HSBColor. The distance to each colour in HSB space is calculated, then the best two nitrate/nitrite classes are calculated and then the nitrate/nitrite levels can be calculated.

Results
The results are shown on a tabbed layout and uses fragments to display the information, these fragments are *InfoFragment* and *MapFragment*. The main parts are these fragment and the floating action button(FAB). The two fragments are blown up in the *SectionsPagerAdapter* inner class. *InfoFragment* takes care of displaying the nitrate/nitrite and give the user two field to fill out the name and description. The map fragment displays the google map and there current geolocation. The FAB is used to save the test.

Current known Bugs

- If a user touches their phone's home button while on the camera activity and then reenters said activity the app will crash. This is something todo with the *onResume* method in *CameraActivity.*

Water Reports (Bluetooth)

The Water Reports section is concerned with the following packages in the Riverwatch codebase:

bluetooth
database
service
util

As well as the hardware code that implements the function calls on the physical device. The device contains a simple API that accepts Strings sent over the serial port (Bluetooth) then sends back a response. In that way both the App and the device listen at all times then on

receiving a string, then they response accordingly. This was how it was implemented when our team received the devices.

The main functionality of the bluetooth package:

- Bluetooth activity and functionality
- Calibrate activity and functionality
- Store the needed Bluno library

BlunoLibrary

This is the backbones on the whole working App that is all done silently in the background. The majority of this class is taken from a previous years app which they modified and adapted the code that was given as a sample to the Bluno chip itself.

The main purpose of the Bluno library is to bind the BluetoothLeService to the MainBluetoothActivity. This allows the activity to access the Bluetooth serial functions which means communication can be established.

MainBluetoothActivity implements BlunoLibrary

In starting the Bluetooth activity the onCreate method is called then the class is instantiated following that the onCreate is called in the Bluno library superclass, creating the needed functionality to communicate to the Bluno chip. The method then initialises the GoogleMapAPI required to determine the location of the device.

The buttons on the app complete their following functions:

- **Test**
- Takes one test on the device by sending across “Test” and then the device replies with one test result in JSON form.
- The App shows this data and also parses the data and packages it up into a single sample and places it into a “water quality report”. This is then saved to the database, then if there is internet connection, the water quality report is sent to the website.

- **Status**
- This function is kind of useless because “Status” is sent across and then the device replies back with a confirmation that the device is running working, if no response is received that means the device is not working correctly
- Here the App only shows the outcome of the response.

- **Retrieve**
- Retrieve sends across “Retrieve” and then the water device sends back all of the data stored on the memory card of the device all in JSON form.

- This often can take a long time on the App, especially when there is lots of data on the card. The Bluetooth isn't super fast and depending on your internet connection it might take a long time.

MainCalibrateActivity implements BlunoLibrary

This class is a basic extension of the MainBluetoothActivity but acts as a settings activity for the operator of the app to use.

The only 100% working part of this section is the updating on the sample time interval and the other parts are just theoretical functions.

App

This is needed in order to send the reports to the website. The other classes in the service package are all related to connecting to the Riverwatch website API

In addition to the code, inside the Android manifest file there are a couple required statements giving the App access to the Bluetooth and Bluetooth services. In particular:

- uses permission
- service
- activity

The hardware code and implementation can be found within the handover documents

https://github.com/LiamDeGrey/RiverWatch
https://github.com/DFRobot/BlunoBasicDemo

History

The history section uses a SQLite database to store reports which can be found in the assets folder. I recommend using SQLite Browser to view and change the database, which can be downloaded here:

http://sqlitebrowser.org/

The graphs for the water tests were created using GraphView, more information found here:

http://www.android-graphview.org/

All of the classes for the history section are found in the ui folder. (The code could use some refactoring). These include the adapters for the lists, the activities for each page and the fragments for swipeable tab layouts.

There is also a Database folder containing a Database class which includes all the methods for interacting with the database.

Future Improvements

Nitrate/Nitrite Testing

Looking forward there are aspects that can be improved upon, mainly the algorithm, while it works, it does only produce a rough estimation which is mostly fine for the purpose of it. The algorithm could be improved either by using more advanced techniques or, using a completely different way of implementing it.
As well as this, sometimes it is not possible to get a accurate picture so an addition that would add some great functionality would be to add a way for the user to upload previously taken pictures which can then be processed by the algorithm.
At the moment the map cannot be manually changed on solely relies on location services, a good addition would be to allow the user to manually change their location if location services are not available.

Location
Allow the user to manually change the locations of each of the app functions in the history so that the user can deal with collecting information in places where the app is unable to find its location.

Bluetooth

Make the communications more robust, that is, sometimes the Bluetooth request is sent to the device, but the device freezes because it is handling too much data

Implement the easy calibration with the buttons provided in the Settings screen