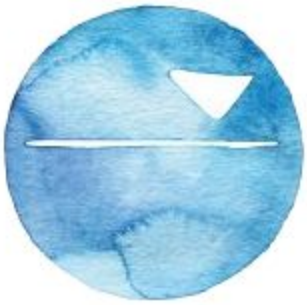

HEALING WATERS
INTERNATIONAL

Client:
Marc Malone

Developers:
Alex Kurzon
Gustavo Bejarano
Troy Woolbert
Brandon Shevela

Date:
6/20/17

1 Introduction

Healing Waters International is a charitable organization based in Golden, CO. They work with local communities in disadvantaged regions to provide safe drinking water, setting up water treatment systems and training for their use and maintenance.

The client has requested an Android application that can streamline management of the on-site water filters. It will allow users to manage a checklist of daily tasks, and to enter the gauge values on the filter to calculate the resistances, flow rates, and pressures. Using this information, the app will display recommended maintenance procedures to the operator. All of the results are uploaded to an online database.

Several masters-level CS students at the Colorado School of Mines have worked over the last few semesters to create computer vision algorithms to read the pressure gauges on the water treatment system. Additionally, a small script has been created to calculate internal resistances for the filter and display maintenance procedures. Our team is tasked with porting these algorithms into the Android application to improve usability.

2 Requirements

Our client wants an application that can be used by employees of Healing Waters as they monitor their water filtration systems in the field. Currently, workers need to maintain a physical checklist for daily tasks, and once a week, they need to manually record the values displayed on gauges on the filtration system. These values are used to determine if everything is functioning properly. Furthermore, all of this data is recorded by hand into tables that need to be compiled together.

The current system wastes a lot of paper with daily checklists and paper records of pressure data. It also introduces human error into calculating the values needed to check if the filters are working. Our proposed application will provide the checklists on the app, and will enable employees to submit the gauge values into the application, either manually or via a picture of water pump gauges.The app will determine the status of the system, then it will display helpful information to the employee. This information will be sent to a database for later retrieval.

List of functional requirements

- Make use of the computer vision algorithm to extrapolate data from the captured image(s)
- Use the algorithm for calculating internal resistances to determine the state of the water filter
- Show helpful messages concerning system maintenance to the user
- Display daily checklists and record answers entered by the user

- The app will have globally configurable values in the settings that affect the internal resistance calculations
- The app will upload the information to a database

List of non-functional requirements

- The database option must be as cheap as possible
- Anyone with basic smart phone/application knowledge should be able navigate and use the app with ease

3 System Architecture

As shown in the UML in appendix III, the application has a main menu that allows the user to access the checklists, gauge, and settings activities. The checklist activity gives the user access to the daily checklists that need to be completed and uploaded into the database. The user is then able to switch over to the gauge activity and input the value of each gauge into the text fields. When the data is submitted, all calculations are performed and the results are displayed on the same screen along with some helpful information for the user about the system. The results are also uploaded to the database simultaneously. Finally, the settings activity keeps track of the minimum and maximum thresholds which are used by the gauge logic handler to display the user certain messages regarding the calculation values.

Figure 2 describes the tree structure for the information that Healing Waters will need stored in the database. It consists of two subtrees: one for checklist submissions and one for gauge readings. Each entry is sorted first by submission date and then by franchise (i.e. the name of the group that manages the filter).

Figure 2: NoSQL Database

4 Technical Design

In designing the application, it was important that users were able to efficiently enter information and have it uploaded to the database with no overhead. Because we could not solely rely on the computer vision algorithm to fetch data, the application had to allow users to conveniently enter the gauge values manually. As well, the checklists needed to be structured to clearly differentiate between tasks that happened at the beginning and end of the day.

As shown in figure 3, entering the gauge readings was fairly straightforward. Even if the user were to have the cells filled using the camera, they are editable text fields that can be modified by interacting with them on the screen. For the flow meter, however, we needed to provide a more complex system for efficient data entry. To get the value for the flow meter, an operator at a water treatment system would keep track of the time it takes for the machine to fill a 10 liter jug. Our application provides a stopwatch for the operator to use. When filling the container, the stopwatch is started, and when the container is full, the stopwatch is stopped. As soon as it is stopped, the field for the flow meter information is automatically filled with the stopwatch value, rounded to the nearest second.

When the submit button is pressed, the program uses the calculations defined in figure 2 of the Appendix to calculate various properties of the system. It then matches these properties with the bounds defined in the settings and displays information and guidance for the user.

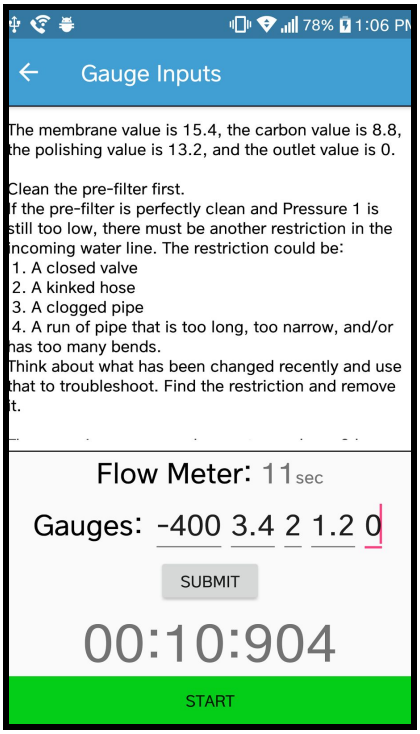

Figure 3: Screenshot of Stopwatch

Structuring the checklist provided an interesting challenge, because proper Android layout design called for a nested set of objects to hold the data and have it arranged in pages. This organization led to two difficulties: the data would later have to be retrieved and submitted to the database, and one of the checklists had a heterogeneous layout—it not only contained text with checkboxes, but also text fields for information about the water filters. A diagram describing the arrangement can be seen in Figure 4. The request for submission came from the top-level Activity that contained the two pages, and so the checklists had to convey their readings up through the intermediate layers for aggregation and submission.

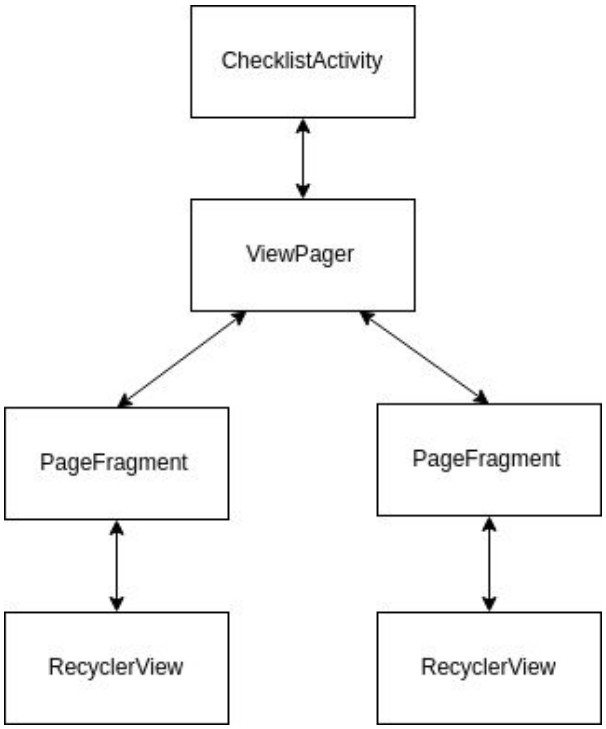

Figure 4: Sub-entities of Checklist Activity

5 Design Decisions

The project is divided into three subsystems: the front-end Android application, the cloud database, and the computer vision algorithm. The application itself has no flexibility in technical decisions, aside from which IDE to choose. The cloud database must be relatively fast and simple to implement, along with being extensible in the future. Additionally, it must have high availability (especially outside of the United States), and have reasonable cost compared to competitors. The computer vision algorithm, written in C++, must be inserted into the Java code in one of two ways: either it must be called directly using the Java Native Interface (JNI), or

reimplemented in Java, with library calls to the OpenCV library (a commonly used computer vision library) using the JNI.

Front-end Android Application

The aesthetics of the application were made to be as simple as possible with functionality of the algorithms as the main priority. This is because the application is strictly for employee use rather than the general public and simplicity would help increase productivity. The application features a main menu that allows the user to navigate through the checklist, camera, and settings activities.

Cloud database choice: Firebase

Initially, our group considered various cloud SQL options, but the costs were too high. We decided that the project would not benefit sufficiently from a relational model to warrant paying it. Firebase provides a NoSQL database for Android applications and is completely free if using the lowest tier of features. This tier only allows 100 concurrent connections, but because the app would only be used by site operators, this would not be an issue. It allows for 1 gigabyte of storage, which is more than sufficient for the small size of each daily upload.

Method of importing computer vision code: JNI and Android NDK

The JNI and the Android NDK were chosen for attaching the computer vision algorithm to our app. These were chosen because translating and rewriting the algorithm would have been pretty cumbersome, as it is fairly lengthy. Using JNI involved just compiling the already existing C++ code in Android studio rather than having to rewrite it. NDK and JNI were chosen to port the C++ computer vision algorithm into the Android application because it would have been too time consuming to translate all of the code into Java, and there could have been unforeseen errors that would arise due to translating from C++ to Java. Because of this we included a backup in the app where the operator can simply input the readings from the gauges if it wasn't working.

6 Results

Our application provides users with a checklist of their daily tasks for managing the company's water treatment systems around the world. It also lets them enter information about the gauge readings and flow meter on the water filter. When the data is submitted, the user is provided with information about the values, and if there is any discrepancy between read and expected values, the user will be provided with a description of the situation and told to contact an engineer. The thresholds used to determine these discrepancies are all loaded from the settings page, and the user can reset to defaults if necessary.

Our application successfully uploads all of the information that the user inputs into a cloud database. The number of completed values in the checklists are tallied, and the percentage is uploaded along with any other recorded values on the page. Due to unforeseen complications, we were not able to attach the computer vision code to the app in time. However, the user can also manually input the gauge readings, and there is a helpful stopwatch for entering the time for the flow meter.

Features not Implemented Due to Time

We were unable to have the app use the camera to calculate the gauge values or the flow meter. We tried to port the code without changes using the JNI, but the API provided in the C++ code did not give the sort of functionality we needed, so we had to rewrite significant portions of the native code so that it would interact with the JNI wrapper in the correct way. We ran out of time in having to backtrack and try to reimplement the computer vision code. One of the problems with the native C++ code was having to access files required by the algorithm within the Android architecture. It is exceedingly complicated to access files in Android using C++ code through JNI.

Summary of Testing

Each of us used the application in both an emulator and a mobile platform during development, using all of the app features and trying to break it if possible. The emulator helped with testing multiple screen sizes to make sure that the layouts remained reasonable for different resolutions.

Future Work

- Making full use of the computer vision code to enable automatic flow meter recognition
- Adding notifications if the user doesn't submit a weekly gauge reading to the database
- Improving overall flow and appearance of the app could be improved

Lessons Learned

Establishing consistent communication between developers and clients is crucial to ensure an efficient project. Our project started with a miscommunication about the scope of the work to be delivered, with the expectation being set that the computer vision algorithm was already accessible and could be easily attached to the application. Fully informing the client helps ensure that everyone is on the same page regarding expectations and results.

7 Appendices

I. Product Installation Instructions

The application will be available on the google play store. From there, it can be downloaded onto any Android device with an api version of 4.4.1 or higher.

II. Calculation Details

```
membrane  = (1st gauge value - 2nd gauge value) * # of secs to flow 10L
carbon    = (2nd gauge value - 3rd gauge value) * # of secs to flow 10L
polishing = (3rd gauge value - 4th gauge value) * # of secs to flow 10L
outlet    = 4th gauge value * # of secs to flow 10L
```

III. UML of Application Functionality