Chapter 1

User Interface

1.1 Introduction

The user interface is how the client will be able to make use of the system. The goal of this subsystem is to use wireless technology to allow the client to capture the weight of the monitored birds remotely.

1.2 Requirements

This subsystem uses a micro-controller to send the data to a mobile device. It uses an application on the mobile device to receive the data. The application has a Graphical User Interface (GUI) to save the weight, the bird's ID, and the time of recording. All the data is saved in a format that can be opened in a spreadsheet.

1.2.1 User Requirements

1. The weight reading must be displayed on phone.
2. The mobile must have interface for manual data.
3. The weight, the bird's ID, and the time of recording must be recorded.
4. Records must be saved to a spreadsheet.

1.2.2 Functional Requirements

1. Scale must be able to send data to the phone.
2. The GUI should have a text box to input the bird's ID.
3. A CSV file needs to be created where the data will be written.

1.3 Design Choices

1.3.1 Sending readings

There are several technologies to transmit data such as: Wi-Fi, and Bluetooth. Wi-Fi has the benefit of conneting to the internet. It requires minimal effort on the user's end if they are already connected to the internet . However, internet access is not feasible as weighing sessions span outside of *eduroam* coverage. Bluetooth is more power effecient, which is important because the scale must be portable. A comparative graph of power usage from Eridani et al. [?] is shown below in figure 1.1.

Figure 1.1: Power Usage Graph

In addition, Bluetooth's speed is sufficient because only several bytes of data are being transmitted at a time. There are two types of Bluetooth: Bluetooth Classic and Bluetooth Low Energy (BLE). BLE is a newer technology that is even more power efficient than Bluetooth Classic; therefore, Bluetooth Low Energy was chosen.

1.3.2 Graphical User Interface

A mobile app needed to be developed to receive and record the data. The client uses an Android device; therefore, an the app was developed for Android devices. The official native programming language is Java. Kotlin is a newer language, that Google will be giving more support to than Java, going forward [?]. In addition, Kotlin has more concise and safer code. It has less boilerplate and it has many language features to avoid common programming mistakes, such as null safety [?]. This it would mean this program would be easier to maintain for newer Android devices in the future and it would be less error prone. Therefore, Kotlin was chosen as the programming language.

Kotlin's UI App Development Toolkit, Jetpack Compose, was used to develop the GUI for the app. This was used instead of XML (Extended Markup Language), since it is more concise than XML, and is officially supported by Google.

1.3.3 Saving data

The data is required to be viewed in a spreadsheet form. The simplest format that can be read in a spread sheet is the CSV (Comma Separated Values) format. The other common option is the Excel Workbook format for Microsoft Excel. This format is more complicated and unnecessary, since only text values are being saved.

1.4 Subsystem Design

This subsystem exists in two environments: the Arduino, and the mobile device. The Arduino was programmed to send the weight readings to the mobile device. A mobile app was developed to receive the weight readings. The app has a GUI for the user to input the bird's ID and save the data to a spreadsheet for the user to view. A block diagram of the subsystem is shown in figure 1.2

Figures/Starling data block diagram.pdf

Figure 1.2: Block diagram of subsystem

1.4.1 Sending data

BLE works differently to Bluetooth Classic. Bluetooth Classic is based on an asynchronous serial connection. BLE, on the other hand, works like a community bulletin board [?]. The peripheral device posts data for central devices to read [?].This model is shown below in figure 1.3.

Figure 1.3: BLE Bulletin Board Model [?]

The information presented by the peripheral device is structured as services, which are subdivided into characteristics [?]. BLE includes a mechanism known as notify. This notifies central devices when the data changes.

A program was developed and uploaded to the Arduino. This program, send_data.ino, can be found under the *send_data* folder in the project repository. Firstly, in the setup function, the BLE name of the device is set to 'Arduino Nano 33 IoT'. Then, a custom service is advertised and a characteristic is added to the service. In an infinite loop, the program checks if a central device is connected. If there is, then the weight value is converted to a string and written in to the *weight* characteristic. The connected central device would then be able to read the *weight* reading as a string.

1.4.2 Graphical User Interface

The Graphical User Interface (GUI) is responsible for displaying the live reading coming from the Arduino, and allowing the user to capture that reading and input the bird ID. The GUI also needs a save button to save the data to a file, so that the user can view the data in a spreadsheet. The prototype of the GUI is shown below in figure 1.4.

Figure 1.4: GUI prototype

In the final GUI design, the bird mass reading was made editable so that the bird mass could be manually entered if needed. In the process of development, it was found that the saved data is stored in a directory inaccessible to the user. Therefore, an 'export data' button was added to export the data to a directory accessible to the user. Additionally, a 'clear data' button was added, so that user has a means of deleting the data, once exported. Figure 1.6 shows how the GUI looks like.

The application can be installed using the APK file, app-debug.apk, found in the repository under *BirdApp/app/build/outputs/apk/debug*. The code can be found in MainActivity.kt, which can be found under *BirdApp/app/src/main/java/com/example/BirdApp*.

1.4.3 Exporting data

The data is written to 'exported_data.csv' in the '/Android/data/com.example.myapplication/files/' directory. This is the only directory that can be written to from Android 10 and later.

1.5 Testing

1.5.1 Sending data

For the purpose of unit testing, the weight value was hard-coded to 346.2. *LightBlue* [?], a mobile app, was used to test the Arduino sending data. Screenshots of the process are shown below in figure 1.5.

Figure 1.5: Testing the Arduino sending data

On the homepage, a list of devices is displayed. The device 'Arduino Nano 33 IoT' was connected to. The service was selected and the characteristic was read. The data format 'UTF-8 String' was selected, and the weight reading was displayed. This reading corresponded with the hard-coded value in send_data.ino.

1.5.2 Graphical User Interface

The app was tested on a Samsung S9 phone running Android 10. The functionality of the buttons in the application were tested. Screenshots of the testing process are shown in figure 1.6.

Figure 1.6: Screenshots of Mobile App

Firstly, the 'Capture reading' button was pressed. This filled in the 'Bird mass' text box with the live reading successfully, as shown in figure 1.6 (b). After filling the 'Bird ID' text box, the 'Save data' button was pressed. This created an FID to uniquely identify each record. The FID, date, time, bird ID, and weight were recorded and displayed, as shown in figure 1.6 (c). The 'Export data' button was pressed, which exported the data, and notified that the export was successful, as shown in figure 1.6 (d).

1.5.3 Exporting data

Another record was made, as shown in figure 1.6 (e), before exporting again. Samsung's native file explorer was used to navigate to the '/Android/data/com.example.myapplication/files/' directory where the 'exported_data.csv' file was found. The file was opened in Google Sheets, where it was displayed as a spreadsheet, as shown in figure 1.7. The exported data corresponded with the saved data displayed in figure 1.6 (e).

Figure 1.7: Screenshot of Exported Data

Glossary