Final Report Weather Witness

Sense Umbrella Connection and Desensitisation project

by

Ronald van Driel
Justin van der Hout
Marco Houtman
Marissa van der Wel

as part of completing the bachelor project course (TI3806) of the Delft University of Technology.

Group members:	Ronald van Driel	
	Justin van der Hout	
	Marco Houtman	
	Marissa van der Wel	
Project duration:	April 24, 2017 – July 9, 2017	
Bachelor Project Committee:	Coach:	Cynthia Liem
	Client:	Zoltán Szlávik
	Project Coordinators:	Otto Visser
		Huijuan Wang

TU Delft

Foreword

The final course taken in the bachelor program of Computer Science at the TU Delft is the Bachelor End Project (BEP). This project requires students to form a group and choose a software project which will display the experience gained over the first few years of the Bachelor. The Sense Umbrella Connection and Desensitisation project was chosen by us because it would involve not only developing an app with advanced audio processing, but also designing the structure for a server/database back-end. This report was written to inform the reader about the entire project process from research phase to the final product. We would like to thank our Coach Cynthia Liem for being very helpful throughout the project and specifically for guiding us in the development of the audio processing part of the project. We would also like to thank our client's product owner Zoltán Szlávik for providing us with the required resources and the constructive feedback to improve our product to comply with the product requirements. Lastly we would like to thank the BEP coordinators Otto Visser and Huijuan Wang who helped us solve some problems we encountered during our project.

Contents

Summary

As of this moment there is a lack of data about rainfall in cities. To collect such data, IBM has started the Sense Umbrella Connection and Desensitisation project. For this project an umbrella was equipped with a piezoelectric sensor and a Bluetooth device to record the rain that falls on the surface of this umbrella. One downside of this umbrella is that, besides rain, it will also record other sounds which include recognisable human speech. Because IBM values privacy, one of the tasks was to make sure that no recording containing recognisable human speech would be uploaded to a server.

This bachelor project focuses on creating a mobile application to connect to the umbrella via Bluetooth and save the audio recordings of rain together with GPS data. After saving this data it would originally be analysed for presence of rain and it was also required to remove all human speech before it was sent from the phone to the server. Over the course of the project it became clear that because of a scarce set of sample data and the limited availability of audio processing libraries on Android, it would be difficult to process audio on Android devices. This is why the decision was made to upload all data, and the processing and analysing was moved to an external device supporting Java. The raw data will now be uploaded to a server which will make a database entry which includes GPS data, and saves the audio file. In the end an app has been developed that is able to gather data from the umbrella and send it to an external server. Additionally an implementation has been made to analyse the audio data gathered to classify which parts may contain rain.

This project focused on developing an Android app and not on other operating systems due to time constraints.

1 Introduction

In 2015 IBM acquired The Weather Company, a company which has been involved in making weather predictions. Because of the acquisition of this company IBM became interested in projects which track rainfall in urban areas. To collect data about rainfall, IBM plans to use umbrellas equipped with a piezoelectric sensor which is able to detect rain as it taps on the surface of the umbrella. This sensor acts like a microphone and makes it possible for the user of the umbrella to record the rain tapping on the umbrella as they are walking around. With enough people walking around with these umbrellas IBM believes that they might acquire valuable rain data within urban areas. Privacy is a main concern to consider when uploading recordings to a server, as these recordings will also include human speech.

In order to collect urban weather data, IBM has started the Sense Umbrella Connection and Desensitisation project. This project was an option for Computer Science students from the TU Delft as their Bachelor End Project. This paper was written by a team of student which have chosen to involve themselves with this project. The project was set up with the goal of creating a mobile app that would link the aforementioned umbrella with a server that would store the collected rain data as well as GPS data with timestamps. To combat the issue of recognisable speech being present in the recordings, the app was planned to prevent human speech from being uploaded to the server. Over the course of the project it became clear that because of a scarce set of sample data and the limited availability of audio processing libraries on Android, it would be difficult to remove human speech from the recordings. However, to value the privacy of the user, the user will be able to decide when to upload or delete their recordings. Additionally the user should be able to remove all the data they have uploaded to the server.

The first two weeks of the project were a research phase in which the problems are defined and possible solutions for these problems are discussed. This research report can be found in Appendix B. Chapter 2 will define the project's main problems and will list the requirements that need to be met to solve these problems. The design and implementation of the mobile application, the server and database, and the audio processing and classification are described in chapter 3 and 4 respectively. In chapter 5 the ethical issues regarding this project, such as privacy, are discussed. Finally the results, discussion/recommendations and conclusion will be discussed in chapter 6, 7 and 8 respectively.

2

Problem Analysis

In the research phase of the project the problem was defined and analysed. This chapter will define the project's main problems and will list the requirements that need to be met to solve these problems.

2.1. Problem background

IBM wishes to collect data of rainfall, particularly in urban areas. They believe this data could be valuable and want to fill the current lack of such data so that more research can be done on rainfall in urban areas. To collect this data they want to make use of an umbrella equipped with a piezoelectric sensor. This umbrella will record the sound of rain that falls on its surface and send its recordings to IBM for research and analysis.

2.2. Problem definition

For the data acquisition there needs to be a connection between the umbrella and the servers of IBM for the data to be transferred. IBM needs an application that can connect to both the umbrella and their servers, so that it can be the bridge between these two. Additionally the umbrella does not only record rain, but also the sound of its surroundings, including recognisable speech. IBM needs a way to analyse the data and a way to prevent sensitive audio from being uploaded to a server owned by IBM.

2.3. Development process

The development methodology used in this project is based on the SCRUM methodology[8]. Since none of the project group members had prior experience with Android development, it was difficult to estimate the time required to implement functionalities. For this reason no two week implementation deadline was used for prototypes, but the group will meet daily to discuss progress. A backlog was kept to decide which tasks were to be implemented.
The Definition of Done that was maintained throughout the project was to have a pull request made once the functionality met the requirements. If all project members approved the pull request after manual testing and review the functionality would be considered done.

2.4. Requirements

In this section the requirements will be discussed that have been agreed upon with the client. These requirements are defined using the MoSCoW method [1]. This methods splits requirements into Must Haves, Should Haves, Could Haves and Won't Haves. Must Haves consider the absolute required functionality of the final product. Should haves consider the functionalities that have a high priority, but are not minimally required like Must Haves. Could Haves consider the functionalities that have a low priority and will only be implemented if time allows it. Won't Haves are functionalities that will not be implemented over the course of this project.

The initial requirements have been defined in the research report, but are listed here again. Some of these requirements have been altered in the course of the project since there has been a change of direction.

Must Haves

M1 Connection between Bluetooth device and mobile phone.

M2 Trigger recording via mobile phone.

M3 Data transfer from phone to server.

M4 No fragments containing speech should be send to the storage.

M5 User must be able to remove themselves and all recorded data from the program and storage.

M6 App for Android.

M7 Send location and time as metadata.

Should Haves

S1 Data transfer uses WiFi.

S2 Use Bluemix for database service and server.

S3 Do not upload data that does not contain rain.

S4 Option to send data through 3G/4G.

S5 Efficient energy usage.

Could Haves

C1 App for iOS.

C2 QR codes to connect app to umbrella.

Won't Haves

W1 Immediate data transfer to cloud.

W2 Let user listen to the files before uploading.

W3 Support for other platforms than iOS and Android.

W4 Support for Android versions under 4.3.

2.4.1. Changes in requirements

Removal of M4

Instead of preventing the uploading of audio containing speech, it has been decided to upload all data. This decision has been made because the limited amount of sample data and the difficulty of audio processing on Android made this requirement not feasible within the allotted time.

Addition of M8

The user has to be able to decide whether they want to upload their recording sessions. This requirement has been added to give the user as much control over their data as possible and is meant as a replacement for M4.

Change of S2

Because there were problems in the process of acquiring an accessible Bluemix server for the project group, it has been decided to work using a local server.

Change of S3

As described before in M4, instead of selectively uploading data, all data is to be uploaded. Rain detection however is still a prioritised functionality. Instead of analysing audio before uploading, it will be analysed after uploading to annotate which parts of audio likely contain rain.

Removal of C1

Although an App for iOS was originally listed as a could have, it was soon apparent that there would be no time to make this app for iOS. For this specific project it has become a won't have but it is possible to create an app for iOS.

Addition of C3

A suggestion was made during a meeting later in the project to let the user have some form of feedback on their recording sessions. This would help stimulate users to use the Weather Witness app more. It is not a crucial feature and will only be implemented if there is time for it, therefore it is listed as a Could Have.

3

Design

This chapter will elaborate on our initial design for several components within this project. These components include the underlying app structure, audio processing and classification, data transfer from phone to database, the tracking of location, the graphical user interface and the connection with a Bluetooth device. As the project progressed, some of the initial design ideas were found to be inaccurate or unpractical and the design was updated accordingly.

3.1. App Structure

For programming it is good practice to maintain proper relationships between classes while having the right responsibility distribution, therefore it is necessary to design a good structure for the Weather Witness app. This section explains the usage of the separate components of our current design and describes how the design of the app structure evolved during the project.

3.1.1. Components of the structure

The structure design of the app comprises three different types of components: activities, fragments and non-UI classes. This section describes how each of these components fit in the design of the app's structure. An overview of how the life cycle of fragments and activities work in Android can be found in Appendix C.

Activities

An activity is described on the Android developers' page as: "a single, focused thing that the user can do."[9] As will be explained in subsection 3.1.2 the decision was made to use a single activity for the whole app. Everything stored in the main activity can be accessed by any interface related object, such as fragments. This means that a fragment has access to the same scope as its activity. Additionally the scope of an activity covers the scope of all of its fragments. Therefore all fragments have access to each other while still retaining their own responsibility.

Fragments

The Android developers' page describes a fragment as: "a behavior or a portion of user interface in an Activity."[11] Fragments can be attached to one or multiple activities and gain access to the same scope as its host activity. This activity has access to a fragment manager which is used to add fragments to or remove fragments from the layout of the app. An activity can contain multiple fragments of the same or different types simultaneously. All fragment instances are limited to the life cycle of its host activity.

Non-UI classes

Non-UI classes are designated to provide behaviour unrelated to the user interface. This implies that non-UI classes do not gain access to the scope of the app's main activity or its fragments.

3.1.2. Evolution of the design

At the start of the project the app was created with the philosophy of having a separate activity for each distinct feature, this would ensure a clear dependency separation as each activity maintains its own scope. Additionally non-UI classes, which are classes that do not interact with user interface, would contain functionality that is accessible by all of the activities. During development it turned out that there was a need for more advanced relationships between features. One of these relationships was the access to non-static sign-in data from Google's API. Therefore the decision was made to utilize only the scope of a single activity and introduce fragments for each feature while maintaining the non-UI classes.

3.2. Recording sound

The audio from the umbrella's Bluetooth device should be recorded, therefore a recorder had to be created for the Weather Witness app. This section will give a short description of the role of the recorder within the app and which recorder the application will use.

3.2.1. The recorder in the app

The recorder will primarily be used in combination with the GPS functionality because the app should have a clear overview of where the user is when the audio is recorded. To make sure the sound and GPS can easily be compared there will be a timestamp assigned to both files at the start of the recording so that they can easily be paired and overlapped. The recorder also depends on the Bluetooth functionality as it should only record when connected to the Bluetooth device. This is because the app should never record any sound which is not recorded by the Bluetooth device. This can be done by using listeners which check if the Bluetooth audio recorder is still connected.

3.2.2. MediaRecorder

After doing research on how to record audio on Android it became clear that using the *MediaRecorder* was recommended, therefore the decision was made to use the *MediaRecorder* provided by Android. The *MediaRecorder* is a basic recorder that can easily be set to save to standard audio/video formats used on mobile phones. It provides basic recording functionality, and it is possible to set the audio source, the output format and the preferred audio encoder. After changing these settings the recorder should be prepared and thereafter it can be started. After starting the recorder it will keep recording until the recording has been stopped by the stop command. If the recorder is interrupted or not stopped correctly it will save a corrupted file. This is why it is important to make sure the recorder is always stopped when leaving the activity or shutting down the application while recording.

3.3. Audio processing and classification

Since audio data recorded by the umbrella can contain more than just the sound of rain, this data needs to be processed and classified at some point so that no other data is shared by any user. This section will explain our initial view of this problem and how it changed as the project progressed.

3.3.1. Initial problem

At the start of the project the task was to upload rain audio to a server while making sure that no recognisable human speech would be uploaded with it. Aside from that, the rain audio that was to be uploaded was preferred to be as raw as possible.
This means that either any audio fragments containing human speech should be completely ignored, or the audio should be filtered in some way to remove the speech while keeping as much rain audio as possible.
This would be done on Android, since uploading audio that might contain speech to then process and classify was out of the question.

Plan

Research regarding audio processing and classification was mostly focused on filtering out human speech and being able to detect rain and human speech (Appendix B). To detect rain and human speech the plan was to make use of MFCCs (Mel-frequency cepstral coefficients) with machine learning to detect if either would

be present in an audio fragment. If rain would be detected in an audio fragment, the fragment should be considered for uploading. In the case of human speech detection an audio fragment should not be uploaded if there is no rain detected. If human speech is detected in an audio fragment where rain has been detected, the plan was to filter out the speech or separate the speech and rain audio.
For filtering, the plan was to use an audio separation method called REPET (REpeating Pattern Extraction Technique) [18] together with high-pass filters. REPET is a method that separates repeating audio from non-repeating audio and was originally meant to separate singing and background music. This method seemed to filter out speech from rain audio well when combined with a high-pass filter. Unfortunately the resulting rain audio would not be raw, but this way there would be no need to throw out audio that contains both rain and speech.
To actually perform audio processing the plan was to use the TarsosDSP [5] library. The reason that this library was chosen is because it is written entirely in Java and has a compatible version for Android.

3.3.2. Direction change

In the middle of the project there were two things that changed the direction and design of the audio processing and classification.

Acquisition of accurate sample data

Until the middle of the project there was no representative sample data available of audio recorded by the piezo sensor on the umbrella. This made the planned use of MFCCs difficult since machine learning requires training data. Also, once samples of such audio were acquired, it turned out that some of our assumptions were completely wrong. For instance the frequency range of the rain recorded by the piezo sensor was much lower than anticipated. The actual sound frequencies of rain falling on the umbrella covered the same range as that of human speech. Thus high-pass filtering would be out of the question. Additionally the volume of human speech was much lower than anticipated which would make it difficult to detect.

Change of app purpose

In a meeting with all parties involved in the project, the decision was made to change the purpose of the app. Instead of avoiding uploading any human speech, all audio would be uploaded regardless of whether it contains speech. Additionally as much information about the rain in the audio would be extracted and uploaded with the audio as metadata. Lastly since uploading speech audio to a server was no longer an issue, the audio processing and classification would no longer have to be done on Android.

3.3.3. Final design

For the final design the decision was made to use onset detection to extract data about the rain from audio fragments. After experimenting with onset detection on the newly acquired sample data and visualising the results, the conclusion was drawn that these results could be used to classify the rain in the sample data.

3.4. Transferring data from phone to database

An important part of this project is the submission of collected data to a database. There are two major decisions to be made regarding this submission. The first is the way this database should be structured, and the second is determining how and when a connection will be established. In the following sections the design choices for these decisions will be explained as well as the security risks involved in data submission.

3.4.1. Data submission

As explained in our research report (Appendix B) immediate data transfer would not be optimal from a user perspective. Data transfer costs energy, therefore the user might want to be charging their phone when uploading starts. It is also possible that the phone is not connected to WiFi when a recording is made, so the user might want to wait until they have a WiFi connection to start uploading. These are the reasons that the decision was made to have the user trigger the upload manually. By allowing the upload to be started manually, the user is given the ability to decide whether they have the battery power available to upload. Lastly, they should have the option to use Mobile Data for uploading if WiFi is unavailable. This has to be optional since it could cost the user money and battery power when uploading through Mobile Data.

3.4.2. Database structure

Which type of database is optimal depends on the type of data, and the operations that will be executed on the data. The data that will be submitted by the user will consist of:

- Audio data
- Date of creation
- GPS data
- UserID

The audio data will not be saved as an entry in the database because it would cost a great amount of database memory. This means the data either needs to be linked to the database entry, or the data location should be stored in the database.
The three operations which will be most commonly executed on the data will be: data insertion, deletion based on UserID and data selection grouping by either location or date of creation. The two main options for storing data are NoSQL or SQL. GPS is better suited to be stored as NoSQL data since there can be several GPS entries for one audio file, and in SQL that would mean a new row for each new GPS entry or a separate table for GPS which would need to be joined. If you are selecting on location it might be preferable to have a new entry for each GPS location but then duplicate data is stored in the other columns.
Since the way the data is accessed is very different in each situation, NoSQL data cannot be optimized for all of them. This is why the decision was made to use PostgreSQL with a single table for all data and a JSON field for GPS. This ensures that the GPS data can be stored in the same table without having to make a new entry for each coordinate.
The final design for the table is shown in Figure 3.1. The *fileID* will be a serial as is common for the primary

FileID (serial)	UserID(long)	Date (yyyy-MM-dd)	Time (hh:mm:ss)	Timezone (int)	GPS (json)
					[[time dif 1, long, lat], [time dif 2, long, lat]]

Figure 3.1: Database table format

key. The *userID* is quite large so it is necessary to store it as a long. Date is stored year first so it is easiest to sort. GPS is stored as json for the reason explained above.
In order to link the file data to the database entry, the id, created when inserting, is retrieved and used to name the file. This is chosen instead of storing the file location because it requires a column less in the table and no unique file name needs to be created as the id is already unique by default.

3.4.3. Connecting to the database

After the user decides to upload their data, it needs to be sent to the database and the audio file needs to be stored. It is possible to store the database information in the app and have it make the connection to the database. If the database information is stored in the app someone could easily issue any request they want if they are able to retrieve this information. This is the reason that the decision was made to send all requests to a server which verifies the user and only lets them delete their own data. This server will be used as a file server, and will contain the audio files. The first step on the server will be to verify the user. Thereafter the server will try to connect to the database and make the proper request using the data provided by the app and the locally stored database parameters.
The audio processing will no longer be executed on the phone as explained in subsection 3.3.2. Since the processing is now a more experimental and separate part of the project it was not included directly into the pipeline, but is a standalone process shown in Figure 3.2

3.4.4. Security

The app needs to have the server address stored to know where to send the data. If someone would decompile the app they would have access to the server address and without setting access rights they could access all files on the server. The address where the actual request is sent to, verifies the user first using a token. This means it should be difficult to see what the server does without this token.

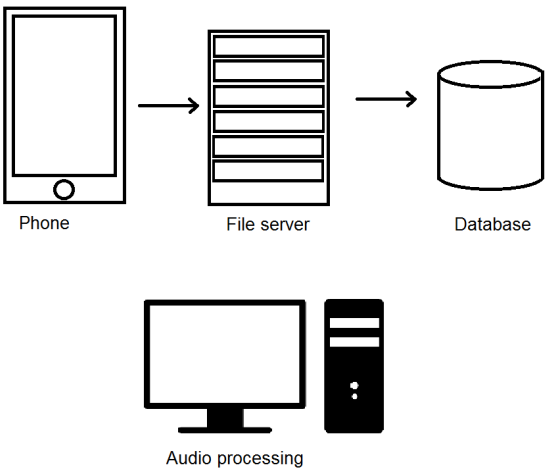

Figure 3.2: Data submission pipeline

3.5. Tracking the app location

The goal of the Weather Witness app is to give an accurate representation on the patterns of rain within urban areas. Urban areas are a tricky location to track these rain patterns as they have varying infrastructures. Within a few minutes you can walk from a street full of skyscrapers through an open park and finally arriving at the residential area you live in. This means that accurate location determination is very important to this app as the rain patterns will differ a lot depending on the location where it has been recorded.

3.5.1. Location Sources

There are two major ways in which Android can track a location, Network providers and GPS providers, The Network providers have a mediocre accuracy as they are sending the location of the broadcast location of the provider instead of the current location of the mobile phone, whereas GPS has a very high accuracy as it is pinging its location directly from the mobile phone. It makes GPS more accurate most of the times as it has a direct connection to the GPS satellites but it is also very battery-consuming and will not always work inside bigger buildings as it can not connect to the necessary satellites. For this situation Network Providers can be used. Although they are less accurate when it comes to walking around outside, they are more accurate when you can connect to nearby WiFi stations if they allow sharing their location from here. For this reason the decision has been made to use both Network and GPS providers to track the users location. This gives the user the possibility to disable GPS tracking and run solely on Network providers but also to provide the most accurate data possible if this has been allowed.

3.5.2. Tracking Frequency

Multiple tracking frequencies will be tested to get an idea of which tracking frequency is suitable for the application. The tracking update delay will not be longer than one minute as it will give a very inaccurate representation of where someone is walking, a user could walk into a dead end and back and the application would think they stood still the whole time. The delay will also not be shorter than ten seconds, updating locations is energy-consuming and while updating every five seconds gives an accurate representation of where someone was walking, it also means that the GPS has to be updated twice as much. A shorter delay would be ideal but is more energy consuming and this is something that should be restricted.
The final tracking frequency will be set during the implementation phase. During this phase location tracking will be tested by checking battery consumption compared to the accuracy the app will provide using this tracking frequency.

3.6. Interface design

The Graphical User Interface (GUI) is how the user will interact with the app. It should be designed in such a way that it is immediately clear to the user how to operate the app, however the main focus of this app is

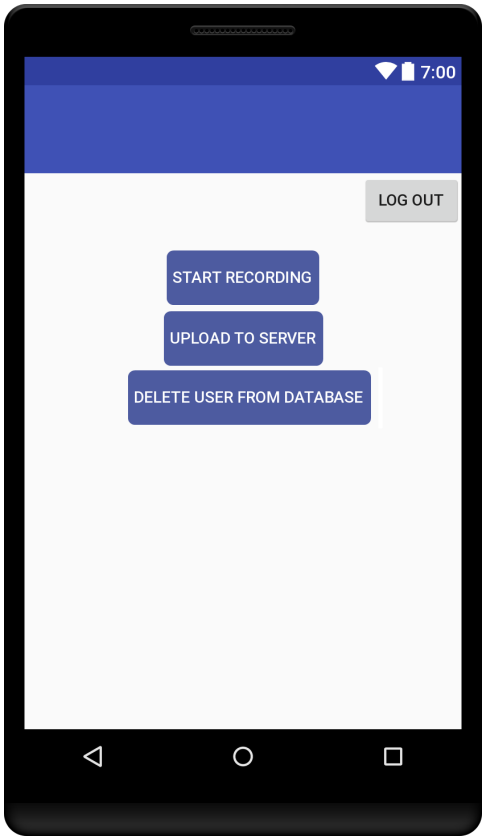

Figure 3.3: Original Interface Design

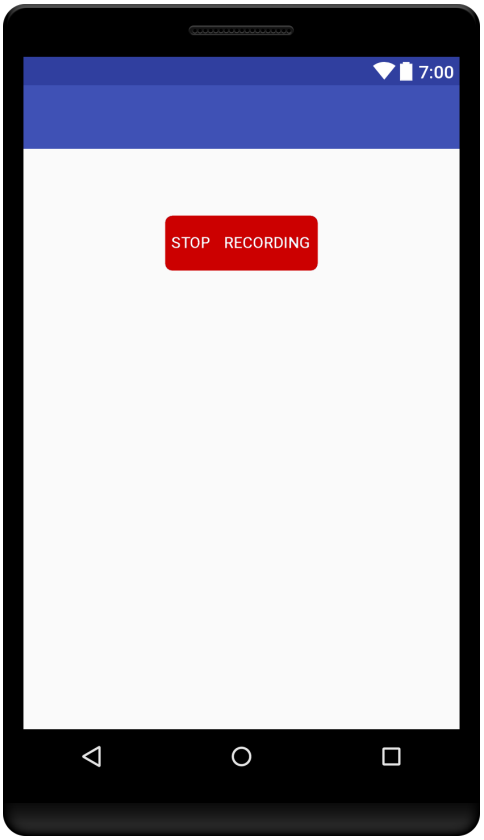

Figure 3.4: Button shown while recording in progress

based on functionality so the GUI is mostly a tool to access functionality.
Since the user will be connected to their data, they will have to log in before they enter the main screen. Google sign-in will be used for the log-in process which has a native layout. On start-up the app will only show the Google sign-in button

3.6.1. Original design

In the original interface design there was a very limited amount of actions available to the user. The main operations which were originally planned on being available to the user via the interface were: making a recording, uploading all recordings and requesting deletion from server. These could all be listed in a main menu page as shown in Figure 3.3.
After clicking on the record button in the main menu the user can choose the device they want to connect to. When they have connected to an umbrella they will enter the recording screen and will then be able to start recording. If they press the button that starts the recording it should give a visual indicator that it is currently recording. This can be done by changing the button text and coloring it red as shown in Figure 3.4. The upload button will let the user switch to an upload screen. In the upload screen the file uploading can be started which will show a progress bar so that the user has an indication of how fast it is progressing. The delete button will let the user delete themselves from the database. It is a requirement from the client for the user to be able to have their data removed. Since we later added some more actions the user could execute we had to update our original design as new functions where implemented.

3.6.2. Updated design

Some new functionalities were added later in the project that were not included in the original interface design. These functionalities are: showing a map of the latest recorded route, deleting GPS back-up files, and deleting all local files. Including all these buttons together with the original functionalities on one menu screen would make the screen rather cluttered. This is why the choice was made to create a main menu and a

sub menu. The main menu will still contain the button to go to the recording screen, but the button to show the map of the latest route will also be added to the main menu. Both the upload file and delete user button will be moved to the sub menu and a button that enters the sub menu will also be added to the main menu. The name chosen for this sub menu will be data menu since all operations on this menu manage local data or server data and will look as shown in Figure 3.5. The buttons for deleting all local data and removing the user from the database are both colored red and will display a pop-up message when pressed. The message indicates that the button the user pressed will start an operation they might not really want to complete and allows them to cancel it.

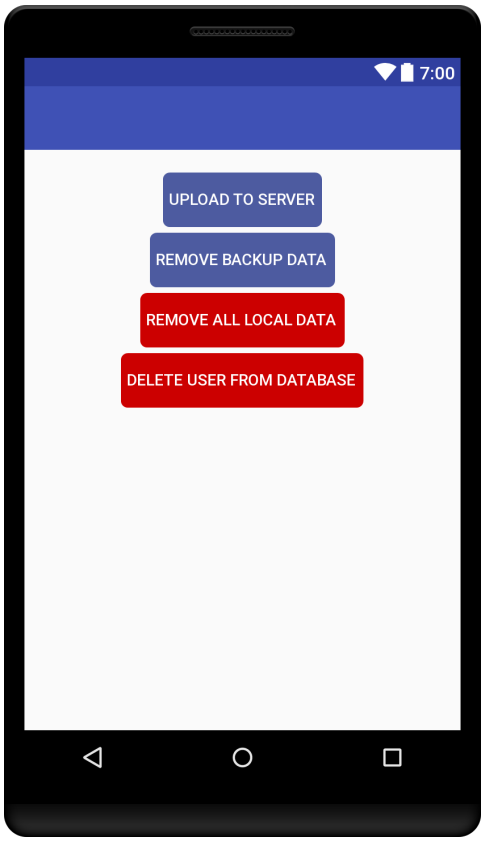

Figure 3.5: Data menu design

3.7. Connecting to the device using Bluetooth

The piezo sensor attached to the umbrella is connected to a Bluetooth recording device. By establishing a connection between this Bluetooth device and a mobile phone it is possible to enable this phone to record the sound that is captured by the device. First the discovery of devices will be discussed, followed by two ways of establishing a connection with the selected device.

3.7.1. Discovering Bluetooth devices

The app can discover Bluetooth devices by using the *BluetoothAdapter*. This adapter is essential for everything the app does with the Bluetooth functionality of the phone. The adapter has a functionality called *startDiscovery* which will add all discovered Bluetooth devices to the list of paired devices in the next 12 seconds. If a device has been discovered, the phone will remember this device until the user decides to remove the device from the list of paired Bluetooth devices.

3.7.2. Android Bluetooth implementation

Using the Android Bluetooth implementation provides a simple solution for device connection, the user is redirected to the Bluetooth settings menu in which he can connect to the shown devices. This implementation is best to use when there aren't too many devices around as it will display all discovered devices. Another disadvantage is that when using the redirected menu it does not automatically send the user back to the application when connected to a Bluetooth device. These problems can be solved by showing the user which device they should connect to and if the user is connected to the correct device, a notification should be sent that the Bluetooth Interface can be closed.

3.7.3. Autonomous Bluetooth connection

Another way of connecting to the Bluetooth device is by using the *BluetoothAdapter* to retrieve a list of all the paired devices to check if the desired Bluetooth device is listed. If this device is listed a connection can be established using a *BluetoothSocket*, this socket uses a UUID which is unique to a type of adapter. When this connection is established the app should try to connect using a profile proxy. Not all Bluetooth devices contain a profile proxy. When a connection is established between the Bluetooth device and the app, the Bluetooth device can be checked for this profile proxy. If it contains the proxy, it can connect to the Bluetooth device using this proxy. Since the umbrella uses a Bluetooth headset device to connect to the app it should be connected using the *BluetoothHeadset* proxy. If this connection is established the device can be used as a Bluetooth microphone.

4

Implementations

After designing most aspects of the product the designs had to be realised. This chapter will give a more in depth explanation on how the design decisions described in chapter 3 have actually been implemented, and some other aspects that were not originally designed such as the map were developed. Each functionality was developed on its own branch and merged with the master branch after peer review and manual testing. The following functionalities will be explicitly covered in the following subsections: audio processing and classification, server and database, connecting the Bluetooth device, Google sign-in, Map, tracking the app location and Recording sound.

4.1. Audio processing and classification

This section will explain more about the implementation details of the audio processing and classification in the project. As mentioned in section 3.3 our approach changed halfway through the project which means the implementation has changed accordingly.

4.1.1. Original implementation

At the beginning of the project the audio processing was to be done on Android since all speech should be removed before any audio was uploaded. The decision was made to first try to use the TarsosDSP library[5] for audio processing, because of its compatibility with Android.

TarsosDSP on Android

Before the change of approach, audio processing using the TarsosDSP library has been implemented for Android. Using the TarsosDSP library on Android requires some modifications compared to working off of Android, because the Java Sound library is not available. There is a release specifically for development on Android available on the TarsosDSP github [4] which has no references to the Java Sound library. Additionally, to be able to read audio from files on Android, FFMPEG binaries are required to be placed within the assets folder [3]. One big drawback of the TarsosDSP library is that the required FFMPEG binaries combine to almost 50 MB of space, which is a substantial amount for an Android app. Before the change of plans high-pass and MFCC extraction methods were implemented in the *AudioProcessing* class, however they are unused since the audio processing moved off of Android.

4.1.2. Change to off-phone processing

Since some audio processing was already implemented on Android using TarsosDSP, the decision was made to keep using the same library and programming language for external processing. The only functionality of TarsosDSP that is used in the end is onset detection. As can be read in section 3.3, there were plans to use MFCCs, REPET and highpass filtering for audio processing, but for reasons explained in that section these were not used in the final design and implementation.

TarsosDSP in Java

In order to use the implementation there is a public method called *processAudioFile* in the *OnsetProcessing* class, that simply takes the filepath of a selected audio file and a path of the directory in which the output gets saved to and then starts the processing/classification pipeline. There is also a simple main method in the *StartProcessing* class that opens a file browser which is used to choose a file. The output of the pipeline consists of several text files which contain the onset data and the classification of the data. All of these files can be imported into Audacity [2] which is a program that was used in this project to visualise the results.

Since the only umbrella that was available for acquiring sample data was a prototype this implementation might not work accurately with a different umbrella and the current parameters for onset detection. For that reason all parameters can be edited in the *Config* class.

4.1.3. Onset detection

Figure 4.1 contains an example of what the results of onset detection look like when visualised in Audacity. The figure shows a spectrogram of the audio and labels for two different types of onset detection. The first type is complex domain onset detection and the second type is percussion onset detection. As can be seen, the complex domain onset detection has labels with a number, this number indicates how salient the onset impact is. The percussion detection is only a Boolean indicator.

4.1.4. Classification

After tweaking the parameters for both types of onset detection, a set of parameters was found that seemed to produce useful results for rain classification. With these parameters the complex domain onset detection seemed to have a low false negative rate, but a high false positive rate. It would detect onsets for the rain, as well as random noises. However the opposite occurred with the percussion onset detection. It would miss rain sometimes, but would not detect an onset for random noises that aren't extremely loud (Figure 4.2).

To make the onset results easier to interpret, a moving average of the salience values is calculated. The amount of intervals and the size of the intervals can be adjusted in the *Config* class. The amount of percussion onsets is also counted for the same intervals.
In the implementation the salience of the complex domain onset detection was used to determine how much impact the rain has and the percussion onset detection was used to determine whether this is a true positive or a false positive. The salience has also been used to determine if detected rain was softer than usual and if there was something that would cause the audio to be overloaded, making detection impossible. The latter happens for instance when adjusting the umbrella or when wind shakes the umbrella too much. In such cases the salience would be incredibly high compared to other parts of the audio.

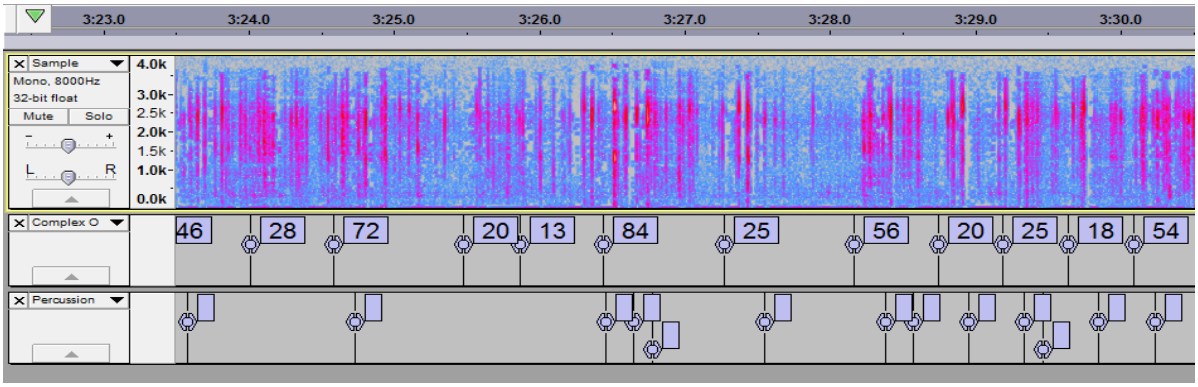

Figure 4.1: Onset detection results visualised in Audacity

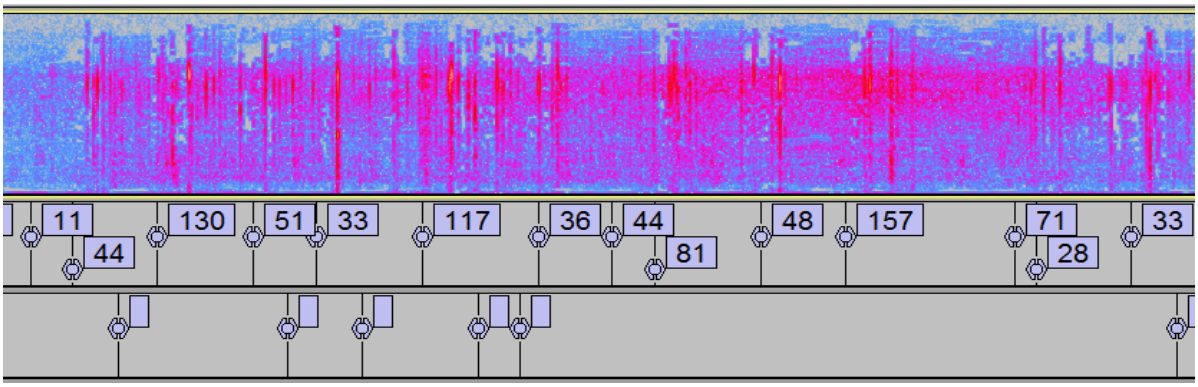

Figure 4.2: An example of the percussion onset detection ignoring a car passing by.

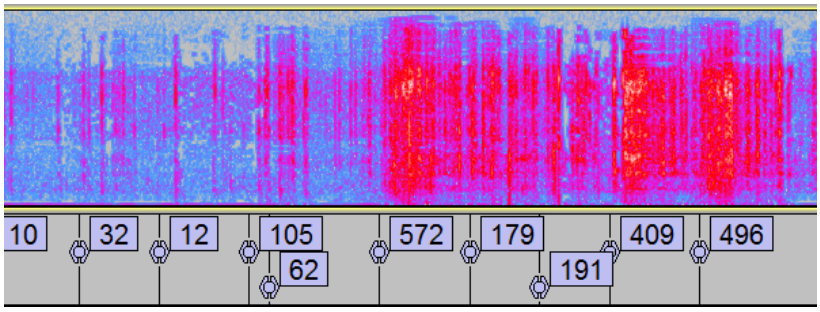

Figure 4.3: An example of extreme noise overload caused by adjusting the umbrella.
Note that the salience values are much higher than in the other examples.

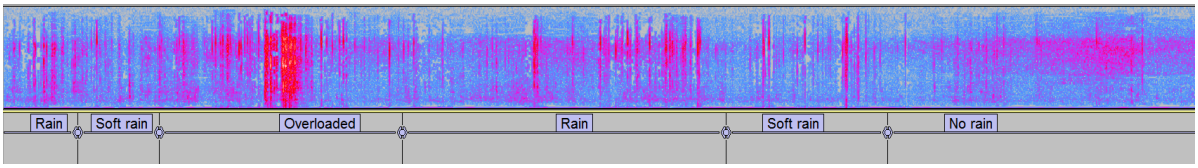

Figure 4.4: Classification of sample data, visualised in Audacity

4.2. Server

The client did not have a server currently available for us which is why server functionality was locally developed and tested. If this app is released by the client the app will need a server, and the server address will need to be changed in the Configuration file of the app. The server setup and implementation will be explained in the sections below.

4.2.1. IdToken verification on server

For user verification an idToken generated by Google is used. This token needs to be verified on the server by sending a request to Google for which the Google API Client Library for PHP was added[6]. To establish this connection the Google API Client library also needs cURL. After downloading the appropriate files the extension was loaded in the php.ini. The first step on the server is checking if an idToken has been send with the request, after which the token is verified. This makes it less likely that a random person can access the server, since they would need to correctly guess an active idToken.

4.2.2. App connection

The two main reasons for the user to connect to the database are to upload their local files, or request deletion from the server and database. Both requests are similar in the way they want to connect to the server using an idToken. This is why both requests in the implementation share an abstract class with a predefined *pro-*

cessResponse method that checks if it was successful and an abstract method *sendServerRequest* which takes a *requestContainer* as input and sends the request to the server. After the app has send a correct idToken to the address defined in the *Config* the server checks if all appropriate post variables are set and if any of them is not set it returns an error response with a message to the app. If all steps are successful the app receives error = false with a message and information about the request made. In case of a successful upload the app deletes all successfully uploaded audio files and the GPS files are moved to a back-up folder so that they can be used for user interaction (section 4.6). This ensures the same file will not be uploaded twice and the user will not have useless audio files stored on their device which take up memory space.

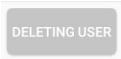

Figure 4.5: Button changes while request in progress

While the upload/delete request is being sent to the server no new request of the same kind can be made. This is shown by fading out the button, changing it's text and making it unclickable as shown in Figure 4.5.

Mobile Data and WiFi connection

The user should be able to send data via WiFi and it should be optional to do so via Mobile Data. For this reason the user receives an *alertDialog* if they are not connected to WiFi if they are trying to send a server request. This *alertDialog* gives them the option of sending it via Mobile Data but they are also free to decline as shown in Figure 4.6. If they choose to decline they will not upload and can choose to try again later if they do have a WiFi connection.

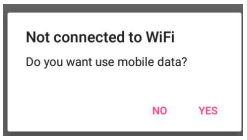

Figure 4.6: User is given the option to use mobile data

4.2.3. Database communication

The database user, password, host and database are all variables stored on the server, because they are needed to make a connection to the database. To connect to the PostgreSQL database the server needs PostgreSQL commands enabled. This was done by downloading the pgsql library and loading the downloaded extension in the php.ini[16]. The server communicates with the database by sending queries to the database. For an upload request the server first sends an insertion query and then retrieves the id created by this query so that it can rename the file to "fileID.wav". If the first query fails the database information on the server can be wrong, or the database is offline so the server registers this failure and tells the user the upload has failed. For a user deletion query the database queries all fileIDs belonging to the user requesting the deletion. After that the server tries to *unlink* (php method for delete) all the files with any of these fileIDs and if this fails to unlink one an entry is made in an error file so that it can later be deleted manually. After the *unlink* operation all entries belonging to the user are removed from the database.

4.3. Database

Since a PostgreSQL database was chosen as our database design and IBM prefers Bluemix if possible, an ElephantSQL database was created via Bluemix. It was not possible to connect to the Compose for PostgreSQL database so the decision was made to test on ElephantSQL. ElephantSQL is an external app listed in Bluemix so this might need to be switched to Compose for PostgreSQL later because this is made by IBM.

4.3.1. Database format

The database was intended to be created as explained in subsection 3.4.2 with a long for userID, which would be suited since the userID provided by Google is 21 numeric characters long. Unfortunately PostgreSQl does not support the long type, its largest numeric column type is bigint which is 8 bytes. That is why the column type of userID was changed to varchar. The gps column is has a jsonb type which stands for json binary. The main difference between json en json binary is that json binary has slightly longer input time but allows indexing and faster processing (PostgreSQL documentation [7]). The date and time columns have been merged

into one timestamp column since it requires less storage size according to the PostgreSQL documentation. The timezone could have been added in the timestamp but is instead stored in a separate column. The reason for this is that it is less important to know what the UTC time was, and more important what the local time was when the rain was recorded. It is still stored since it might be necessary later. The timezone column is an integer but the way the data is submitted to the server is 0200 so that the entry for +2 becomes 200 and for +10 is 1000 so some processing is required if the timezone is going to be used. A table entry will in the end look similar to this:

id [PK]...	datetime timestamp without time zone	gps jsonb	userid char...	ti... in...
5	2017-06-20 14:15:32	{"gps":[["1497961001068","52.00185783","4.37397331"],[...	1133...	200
6	2017-06-20 14:18:56	{"gps":[["1497961164300","52.00181123","4.37399929"]]}	1133...	200

Figure 4.7: Data table with entries

4.3.2. Optimizing data retrieval

Of course the data will need to be queried at some point as well. To decrease retrieval time of all files on a certain date or time an index on the datetime column has been made. With this index the insertion time will be longer, but the retrieval time will be faster. It is possible to index on the GPS column since it is of the type jsonb. Since as of yet the exact queries that will be used on the data retrieval are not known, these indexes have not been created but could be in the future.

4.4. Connecting the Bluetooth Device

This section will cover the implementation of the Bluetooth Device. The first section covers the discovery of devices. The next section describes the usage of the standard Android Bluetooth implementation which requires user input, this is followed by a section on an implementation in which the app can possibly autonomously establish a connection. The final section will describe why we have chosen for the Android Bluetooth implementation (mentioned in 3.7.2).

4.4.1. The initial implementation

The initial idea was to show a list of umbrellas in the app on which the user could click to connect the device within the app. After first trying to connect to the Bluetooth device inside the application it became clear that there was a problem using this implementation.
One problem became apparent with the *AudioRecorder* once the app could successfully connect to the Bluetooth device of the umbrella. It would keep recording when the connection with the Bluetooth device was lost and start recording sound from the phone's microphone instead of the piezo-sensor. Because there is a huge difference between the way the phone's microphone and the piezo-sensor record it became clear that the recorder should be shut down the moment the connection to the umbrella was lost.

4.4.2. Encountered problems

When connecting to the Bluetooth Device through the application it became clear that there was an issue at the moment the device notified it was connected. It could not be recognized as a Bluetooth Microphone which made recording from the device impossible. Since during the initial implementation it became clear that if already connected to the device the *AudioRecorder* would work fine the choice was made to redirect the user to the Bluetooth interface implemented by Android.

Another problem was that the recorder was still running while the umbrella disconnected, this could be solved by using a listener.

4.4.3. The final implementation

By changing to the Android Bluetooth implementation the problem, that the Bluetooth Microphone was not recognized as a microphone, has been solved. The user is redirected to the Android Bluetooth Prompt and a message clarifies the user to which device he should connect. After the connection is established a message

prompts which clarifies to the user that they can use the return button to return to the app. After this connection is established the AudioManager is called. The AudioManager is first prompted to start the possibility to record from the Bluetooth device using startBluetoothSco(), followed by activating BluetoothSco and adding an Intentfilter. This filter checks if the BluetoothDevice is connected as a microphone. If this connection is established the user can choose to start recording through the umbrella.

4.5. Google sign-in

During the research phase the agreement was made with the client to require the user to log in using a Google account. We considered that since Android users already have a Google account, this was no drawback while providing advantages. One of these advantages for using a sign-in was the ability to link a user to the data recorded by that user. This link would enable users to delete their data. Most of the functionality of the sign-in is handled by Google's back end implementation which resides outside of the scope of our app, therefore it makes sense for the app to follow Google's intended log-in design instead of creating our own special design. The following subsections each cover a different aspect in relation to Google sign-in.

4.5.1. Google integration

The Google sign-in requires the Weather Witness app to integrate the Google services API for authorization. For this integration the following guide was used [12]. Google's API needs the following dependencies to be present: Android 2.3 or newer, Google play services SDK, Android keystore, google-services.json configuration file, and the OATH 2.0 client ID from the server. Android studio automatically generates a keystore file for Google to verify at the default location: %USER%\android\debug.keystore. Google sign in will not work if the registered keystore is not present on the system. In the case Google's API is going to be used when further development is being made to the project it would require to register a new project at `https://console.developers.google.com/` as well as registering a new keystore in accordance with [13]. The OATH 2.0 client ID is simply defined as a variable in the file strings.xml matching the address of the local server we have been using, if another server is being used this string should be changed accordingly.

4.5.2. LoginFragment

All code related to signing in to Google is placed inside the LoginFragment. This fragment is also responsible for the layout of the log-in screen. Currently this layout only contains the sign-in button. This fragment is displayed as soon as the app is started and dismissed on successful sign in which means that the user is always logged in while navigating through the other fragments.

4.5.3. Sign-in activity

When a user presses the sign-in button Google's API requests the user to sign in using the provided sign-in activity outside of the app. The scope of the app does not cover any data from the sign-in activity and only receives a callback when the user signs in successfully or leaves the activity. Inside the sign-in activity Google validates the users credentials and returns a sign-in result which contains the Google account information.

4.5.4. Google account information

When the user has signed in with valid credentials a reference to this accounts information is stored within LoginFragment so that it is accessible to all other fragments. It is important that other fragments have access to this information because several problems were encountered when a separate activity was used for sign-in as described in subsection 3.1.2. The most significant problem was that generating a token for server-side verification, which requires access to a Google account, would happen in a different activity. Passing the account data between activities was not possible in an appropriate manner. The Google account information is also used to show the name of the currently logged in user in the main menu.

4.5.5. Revoking Access

The owner of a Google account is authorized to revoke access to the account from any device. The user is able to revoke access for the device that is being used for the Weather Witness app. If the user does so it is no longer possible to gain access to that account within the app. It is however, for a single time, still possible to

use this account to get past the log-in screen without actually being logged in. This behaviour seems to be caused by verification on Google's side and might cause the Weather Witness app to stop working until the user has reconnected after logging off.

4.6. Map

Initially there were no plans to add a map feature to the the Weather Witness app until the mid-term meeting. During this meeting the agreement was made that, for the app to adopt a user base, there should be incentives for the user such as feedback or interaction with social media. As a template for providing feedback to the user a map feature has been implemented and added to the app.

4.6.1. Dependencies

For the map feature to work properly a Google play services version of 11.0.0 or higher should be installed on the Android device. And the application should be allowed access to Google's map API via Google API project on the developers dashboard at `https://console.developers.google.com/`. Google provides a guide for the initial setup process [15].

4.6.2. Maps layout

The map is drawn on a separate window controlled by the Map Fragment. This Map Fragment contains a by Google defined Support Map Fragment which is where the map is drawn in. The reason two fragments are used for the map is because the layout of the Support Map Fragment can not be expanded. The Map Fragment on the other hand is able to be expanded with additional layout elements while the Support Map Fragment can only display the map. With the current implementation the Map Fragment to the main activity automatically initiates both fragments.

4.6.3. Features

When the user enters the map they gain access to the features as listed in the following subsections.

GPS reader

When the map is opened it will display a path based on the most recent recording from the user. The paths are created by reading coordinates from the GPS data files as they were stored when the user has had a recording session. Not having a GPS file stored in either the backup folder or the GPS folder will prohibit access to the map fragment. When a file with at least two coordinates is found the map fragment will be opened and a map will be shown with a path drawn between the coordinates of the file.

Path

The user will see a path drawn corresponding to the locations of the latest recording session. After the path has been drawn the screen will fit the path to the screen. This path consists of individually clickable line segments. An example of what such path looks like can be found in Figure 4.8. The line segments in this path are of Google's standard polyline type as defined in [14] and can be customized to a certain extend. It is possible to listen to clicks on the lines by the user and execute any code desired in such event. In the implementation for example the segment that is clicked is highlighted with a different color. As seen in Figure 4.8 a marker is drawn at the start of the path, which corresponds to the first coordinate in the GPS file.

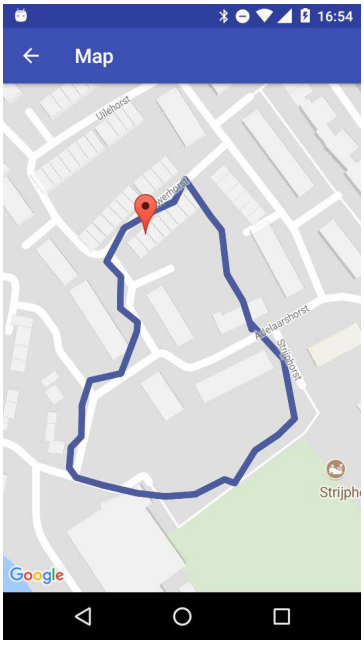

Figure 4.8: Map with drawn path.

Information boxes

Clicking on the line segments of the path will not only change the line color, but also show an information box. This information box is a small

window that shows up on the screen at the location of the clicked line segment. This box can be used to display information on the weather conditions. The intended use case for the information boxes is displaying the classification of the rain recorded at that location. The Google maps API enforces these boxes to be attached to a marker. In this case a line is clicked instead of a marker and therefore each line has access to an invisible marker located at the middle of the line segment, this marker will open the information box when the user clicks the line.

4.7. Testing

To check if all parts of the app work as intended, two types of tests where used. For most non-GUI parts of the app jUnit tests were written, and for GUI parts manual testing was used. The following sections will describe how we approached both jUnit testing and manual testing.

4.7.1. Manual testing

Each functionality that was deemed as ready to be merged with the master branch was manually tested with the following questions in mind:

- Does the branch function as is stated in the pull request and is the way it is described desirable.
- Do the old functionalities still work as before
- Can a crash be caused by executing unusual behaviour

If these questions could be answered in a satisfactory manner the pull request would be approved (if proper documentation was present).

4.7.2. Testing using jUnit and Powermock

Some classes in the project had no GUI elements in them at all. These classes were tested using jUnit tests. Due to a lot of dependencies on the basic functionality of a mobile phone it is difficult to test the application, for example the logging command would cause an error while testing. This is why Powermock[17] has been added to our project library. Powermock gives developers the possibility to mock static and private objects and variables. This should be done with care as mocking a class sometimes means that every step should be manually verified during testing. It does however provide the possibility to check if every functionality is working as intended and it can improve the testing coverage of certain classes by more than fifty percent.

4.8. Tracking the app location

As discussed in the design chapter the app uses the location tracker provided by Android. The implementation is based on the documentation found on the Android website `https://developer.android.com/guide/topics/location/strategies.html`. The first section will focus on the initial tracking implementation, the next section will provide some feedback on the problems found when working with the initial implementation and the final section will focus on the final implementation of the tracker.

4.8.1. Initial implementation

In the initial implementation the user had to give permission to use location tracking and allow the application to use one of the possible location tracking functionalities, this means allowing either GPS or Network location tracking. When both tracking methods were activated the application would always try to use GPS and if not available use Network locating. The choice was made to display the location of the user every ten seconds and write the location to a text file. This text file used the convention: $latitude : val1 - longitude : val2 - time : val3$.

.

4.8.2. Encountered problems

It became clear that when the GPS was always preferred, it was not always trustworthy. This is because in certain situations the GPS source would not always be the most accurate source. For example when walking

through a building the accuracy of a network provider would be higher but the GPS location was still used. This could be solved by introducing criteria. These criteria are sent to a *LocationManager* which will choose the most viable location tracker for the current situation. Another problem was that when looking back to someone's locations it became clear that when GPS was not enabled the accuracy was way lower than when it was enabled. This resulted in sending the text file with a new convention so that a certain location which was tracked with GPS or Network locating would be labeled as such.

4.8.3. Final implementation

In the final implementation two major things changed: The Criteria were added and were adjusted according to the fact if GPS was enabled, if this is enabled it would always look for the most accurate location and would allow a high energy usage (as the user allowed using GPS). If GPS was not enabled the accuracy was set to a coarse accuracy and the power requirement would be set to low. These criteria are updated before an update is asked from the *LocationManager* to make sure that when the GPS is enabled or available it will choose for the GPS over the Network tracking. The other change was adding the last provider source to the text file with the following convention: $time : val1 - latitude : val2 - longitude : val3 - provider : val4$. This makes it possible for the database to just use the recordings on accurate locations by only filtering on GPS providers.

4.9. Recording sound

During the implementation of sound recording several parts of the code have changed. The next sections will sketch an overview of what happened to the code concerning audio recording. The first section will focus on the initial implementation, this section is followed by a short overview of what problems were encountered during the usage of the implementation and the last section will cover the final implementation.

4.9.1. First implementation

The first implementation used the *MediaRecorder* and saved files in 3gp. The *MediaRecorder* would always directly write to the final file. The usage of the *MediaRecorder* and the 3gp format caused some problems however. The extension of the file is not a very common sound-extension and is not always supported by programs which were required for audio processing. This is why the decision was made to look for a new audio recorder which would be able to save files to a wav format which is a very common format to use in audio processing programs.
Another problem was that when spamclicking on the button to start and stop recording it would result in several useless files as they were corrupted because they could not correctly be processed.
The final problem was that by directly writing to the final file it would become corrupt when not processed correctly but it would still be saved on the final location as a corrupted file.

4.9.2. Final implementation

After some searching an implementation was found which can record sound using the *AudioRecord* from Android and which adds a header which is necessary for a wav audio file. The time needed to process the files to save them correctly was not that long, this is why a timer of half a second was added to the button before a new recording could be started or stopped, this solved the possibility of corrupt files because they were not being processed correctly if interrupted prematurely. This implementation also was less error-prone as it would save the file to a temporary location and when the audio processing was finished it would be saved with the new header to the final location. This solved the issue where corrupt files would be available in the upload folder.

5 Ethics

A well designed software project not only solves the problem that was meant to be addressed but also takes into account the ethical implications of the design. This chapter will talk about the ethical issues that have arisen during the course of this project.

5.1. Privacy of the user

One of the most prevalent issues in this project is the privacy. This section will discuss the privacy of the user of the Weather Witness.

5.1.1. Data recorded

Since the piezo sensor can also record speech, conversations of the user could be uploaded to the IBM server. This would be a major breach of privacy if done without being able to ensure the user that such recordings will not be listened to by any party other than the user without their knowledge and consent. For this, the initial task was to prevent uploading any audio fragments that contain recognisable human speech. The user would also have to be able to remove all their data from the server and database.
During a meeting with all parties involved with the Weather Witness project, it was pointed out that the requirement to prevent the uploading of audio fragments containing recognisable human speech might not be broad enough. After all there could be other things recorded than speech that might be considered private for the user. Additionally if this requirement would be more broad it could become too restrictive. Therefore the decision was made to change this requirement. Instead of preventing uploading audio containing private information, the user would be the one to decide whether to upload the recording. However the user would still have the option of pulling out of the project should they wish to do so. It is of course possible some analysis has already been executed on the gathered data while the user still had their data uploaded to the database. It is impossible to ensure that their data gets retracted from this analysis.

5.2. Privacy of people other than the user

This section is dedicated to discussing the privacy of other parties than the user. While the user has the option to decide whether to upload or remove the data that their umbrella has recorded, this does not apply to other people that may be recorded by the umbrella. Obviously the user cannot be asked to be considerate of what they could be recording and uploading when it comes to someone other than the user. This would mean that recordings of conversations of people could be uploaded without the consent of those people. Unfortunately as of now there is no solution provided to combat this issue.

5.3. Usage of Google ID

The usage of Google ID has made it easy to make the connection between a user and their data in the database. However this exposes the user to certain privacy risks. With the current implementation there is a direct reference to the Google ID of the user in the database. This ID for instance can be used to find

the Google plus account that is connected with the ID. This means that everyone with access to this database can link GPS data to an actual person, thus it is possible to find where a person lives or works. In case of a database breach this information could become available to the hacker who could use it in malicious ways.

6

Results

This chapter will evaluate the final product by going over the list of requirements that were defined in section 2.4 and evaluating the development process. The evaluation of the requirements will show for every requirement if it has been met or not and an explanation will be given on how or why that is the case.

6.1. Process evaluation

This section will reflect on the development process of the project. It will cover how the development process was maintained and what issues in the process have been encountered.

Even though there was no delivery deadline for functionality, the group delivered functionalities on a regular interval. Because the group planned biweekly meetings with the client, implementation was planned to be delivered before this meeting. If a functionality took longer than expected the progress that had been made was pull requested and merged after testing so that it could be showed to the client.

One major problem in the development process was the limited availability of the umbrella used in the project. The project group only had one umbrella available and this umbrella was ready for use late in the project. Because there was only one umbrella, thorough manual testing by all project members could only happen at the daily meetings. The Bluetooth device could also easily break off of the piezoelectric sensor, which means that it would have to be soldered onto it again. Functionality that did not require the umbrella were prioritised in the first half of the project because of these issues.
Another problem was that once the umbrella was ready for use, it took a long time before there was a day with rain to acquire accurate sample data. Instead the sound of rain on the umbrella was simulated using a shower, however this was not an accurate representation of actual rain falling on the umbrella.

These issues slowed down the development process because some functionalities would not have been implemented if the connection to the umbrella had been implemented earlier so that accurate sample data would be available.

6.2. Evaluation of requirements

The following section will for each requirement list: a brief explanation, if the requirement was met or not, and an explanation as to how or why this was the case.

Requirement	Met	Explanation
M1 Connection between Bluetooth device and mobile phone.	Yes	The app lets the phone connect to the device.
M2 Trigger recording via mobile phone.	Yes	After connecting, the app will give the user the option to record via the Bluetooth device.
M3 Data transfer from phone to server.	Partially	After recording, the user can upload the recording session to a server. This server is local so an external server needs to be deployed by the client.
M4 No fragments containing speech should be send to the storage.	No, changed	In the middle of the project it was decided with the client that this requirement would be changed and instead the user is be able to delete recorded data.
M5 User must be able to remove themselves and all recorded data from the program and storage.	Yes	The user is able to do this in the data managing menu.
M6 App for Android.	Yes	The mobile application is compatible with Android.
M7 Send location and time as metadata.	Yes	During recording the time and GPS coordinates are saves and when uploading, this data is also uploaded.
M8 The user has to be able to decide whether they want to upload their recording sessions.	Yes	The user has the option to delete all files they have recorded and can choose when to upload their files.
S1 Data transfer uses WiFi.	Yes	When uploading data, WiFi is used when it is available.
S2 Use Bluemix for database service and server.	Partially, changed	Because of problems in the process of acquiring an accessible Bluemix server for the project group, it has been decided to work using a local server. The database however is a Bluemix ElephantSQL database
S3 Do not upload data that does not contain rain.	No, changed	In the middle of the project it was decided to perform the audio processing and classification outside of Android. This means that it will not happen before uploading.
S4 Option to send data through 3G/4G.	Yes	When WiFi is not available the user will be prompted to ask if they wish to use their mobile data instead.
S5 Efficient energy usage.	Yes	Certain coding practices described in the research report have been used to reduce energy usage. Energy usage has also been taken into account when implementing the GPS tracking.
C1 App for iOS.	No	It soon became apparent that this requirement was not realistically feasible. It had been listed as a Could Have which means that it would only be implemented if there was time for it.
C2 QR codes to connect app to umbrella.	No	There was not enough time to do research in this area and implement it.
C3 Let the user have feedback on their recording sessions	Provisional, one form of feedback	A map functionality that lets the user see where they have walked using the GPS files has been implemented, since it was relatively easy to do. However because of a late change in how GPS files are saved, this functionality needs changes to work properly.

7

Discussion and recommendation

This chapter will explain why certain decisions where made for this project, and will list recommendations for further continuation of this product.

7.1. User incentives

The focus of the project so far was on gathering data suited to IBM's needs. Currently there is no real incentive for the user to use the app except for wanting to help with the research. Possible ways to attract users would be to give feedback to the user based on recordings, share rain information via social media or provide accurate weather data to the user. An example usage of user feedback is implemented in form of a map such as described at section 4.6. The map can be extended by providing the user with feedback on the collected rain data when it has been processed. Social media can function as an incentive by receiving data from other people as well as sharing personal data with others.

7.2. User sign-in

Google sign-in is currently used to connect the user to the data. An important question is why the user should even be connected to their data. One of the clients demands was for a user to remove their data from the database, which means a connection should be established between the user and their data. Another way that the userID is currently used is for the server to check if a legitimate request was made by using an idToken generated using Google sign-in. If the demand for a user to be able to remove their data would be dropped by the client, the log-in would still be necessary for the server verification, or another way of making this connection more secure would be required. Another possibility is to upload all data to storage and an external server can watch this and process it as it comes in. There is one significant drawback of using Google sign-in since it can be used to establish a direct connection to public data using Google Plus.

7.2.1. Google Plus ID

The userID received from Google when using their sign-in can be directly linked to the user via Google Plus. Going to `plus.google.com/userid` will lead you directly to the user's Google Plus page. This means it can be possible to access their first and last name, date of birth and other information. This is why we highly recommend changing the ID used since the database stores both GPS data and the userID which can be combined to track a person's often visited locations (like their home and office).

7.2.2. Open ID

During the midterm meeting of the project Otto Visser, one of the project coordinators, pointed out that open ID can be a better substitute for Google sign-in. Mostly to enable the app to utilize social media to provide incentive for using the app from user perspective as mentioned in section 7.1. Open ID has not been included in the Weather Witness app because we had to focus on other functionality due to the time constraints of the project. If social media interaction is something considered valuable we recommend researching the possibilities for an open ID integration with the app.

7.2.3. Unique phone ID

The Weather Witness app uses Google accounts to link a user to the data as described in section 4.5. In the future, the unique phone ID should be considered, as opposed to using an account, if the following two conditions are met. The first condition is that there is no need for the app to distinguish between different users on the same phone and the second is that the app has no interaction with either a Google account or an open ID account. The unique phone ID is easily accessible on all types of mobile phones and can also be used to link data to a user.

7.3. Connecting the processing

There is currently no real connection between the audio processing part of this project and the app/back-end because the processing is still very experimental. Processing will need to be integrated into the process. Our recommendation would be to keep the current structure but instead of storing the files on a file server maybe they should be stored somewhere where the audio processing can occur so that it does not need to be retrieved when the processing has to take place. If the best way to analyse the data is finalized it can be moved to a different point in the pipeline and the processing can be done before the file is saved to the server.

7.4. Java for audio processing

When it comes to audio processing, Java is certainly not the most efficient language to use. This section will explain why the decision was made to use Java for audio processing.

During the research phase it was decided that the TarsosDSP library would be used for audio processing. The most important reason was because initially the audio processing was to be done on Android and TarsosDSP has a version that is compatible with Android. TarsosDSP was also the only compatible library with sufficient documentation and it had the widest variety of functionalities. Lastly the library is written entirely in Java which allows for potentially adding functionalities.
Once the decision was made to no longer proceed with the audio processing on Android, the decision was made to keeping using the TarsosDSP library. This was done because there was already an implementation using the library on Android, which made it easier to implement it again. Aside from that there was no research done on audio processing libraries without Android compatibility and since the remaining time was limited it seemed the best choice to keep using the same library, even if it may not be the most efficient library. By still using the TarsosDSP library it is also possible to later move the processing back to the app.

7.5. Bluetooth

7.5.1. Multiple umbrella's

As we only have one prototype umbrella it is possible to save one address and compare this address to the discovered devices, if the umbrella is available the user can be prompted to connect to this umbrella or the app can connect to the umbrella without user input. In the case of multiple umbrella's it becomes a bit more complicated: If there are multiple umbrella's which are discovered it is possible that the app will try to prompt the user to pair to a wrong umbrella. This can be solved by having a clear Bluetooth ID on the umbrella, giving the user a clear ID to which they can connect in their device. Another possible solution is providing the umbrella with a QR code which can be scanned by the phone. This makes it more transparent for the user how they connect to the right umbrella and solves the problem of looking through multiple discovered IDs.

7.5.2. Bluetooth Low Energy

One of the should haves is Efficient Energy Usage, this is why the choice was made to develop the app on API 18 as it allows Bluetooth Low Energy. Unfortunately the Bluetooth Device does not support the use of Bluetooth Low Energy so it became unnecessary for the application to support Bluetooth Low Energy. With the current implementation the device would most likely be connected with Bluetooth Low Energy as it is currently connected through the Android Bluetooth implementation. For future umbrellas it would be a smart idea to use a Bluetooth device which implements Bluetooth Low Energy.

7.6. Location Tracking

7.6.1. Location Accuracy

When looking back on the Location tracking data it becomes clear that there is a severe error in accuracy when not using GPS, this is because the location given by the Network data is the closest Network transmitter. From data acquired it became clear that these transmitters are spread and the distance between seemed to be at least twenty meters in Delft. This means that the accuracy of the phone when not using GPS can easily differ twenty meters from the actual user's location. Because of this discrepancy between GPS and Network locations it is currently set that the acquired data has a label for every location which states if it was tracked using GPS or Network tracking.

Because the goal of this umbrella is to acquire accurate data within cities it would deliver better results if the user was forced to at least activate their GPS when recording with this umbrella. This is recommended due the discrepancy the Network tracking gives as it would compromise the data because even a distance as short as twenty meters might be the difference between walking through a park or walking next to a skyscraper.

7.6.2. Tracking Frequency

With the current settings of the app it is tracking the location of the user every ten seconds. This is because while testing it seemed that the battery did not suffer too much from retrieving a location every ten seconds and it was easy to couple a sound bite to a location.

In the future the user might want to have a little bit more freedom on choosing their tracking frequency. A possible solution is providing the option to track every twenty, ten or five seconds. This would give the user a option to save battery or to help provide the most accurate data. A possibility to give more incentive to use the most accurate data the app might return extra information on the data retrieved.

7.7. Audio processing and classification

7.7.1. Prototype

Currently the audio classification is tweaked for a prototype that is likely not representative for the actual umbrellas that will be used. Therefore it might be necessary to tweak the parameters of the audio processing and classification again. Depending on how much difference there is with the sound recorded by the prototype and an actual Weather Witness, the classification design may also have to be changed. All parameters can easily be changed in the Config class.

7.7.2. Visualisation of output

For tweaking the processing and classification, it would be helpful to have a visualisation of the output. During the course of this project the Audacity [2] program was used for this. All of the output, which includes not only classifications but also raw onset data and average onset data over intervals, is formatted in a way that it can easily be visualised by importing them as labels in Audacity. The format of the output is very simple, every line contains a start time, end time and a tag. In the raw onset data the start time and end time are the same because they are detections of single moments. The tag represents the salience, average of classification in a label.

7.7.3. Other audio processing

Currently only data of onset detection is used for classification. However other forms of audio processing may prove useful for classification. For instance if enough training data can be assembled, MFCC features could be used with machine learning to classify rain. The library used for audio processing in this project, TarsosDSP[4], also has MFCC functionality.

7.7.4. Android audio processing

In the case that IBM wishes to use a form of audio processing on Android, there is still an implementation using TarsosDSP present for basic forms of audio processing, which can server as an example. Note that to be able to read audio from files on Android, almost 50 MB of FFMPEG binaries are required[3]. Another drawback is the speed of processing, it can take seconds to several minutes before the processing is done.

This duration depends on how many and which forms of processing are used at a time. An implementation of audio processing on Android is still present and if IBM does not wish to do any audio processing on Android they are advised to remove these to reduce required app space.

7.8. Recording intervals

With the addition of recording intervals it became possible to upload shorter audio and GPS files. These different fragments make sure that when corruption in a fragment occurs that the other fragments can still be uploaded. One of the disadvantages of the fragments is that there was no time to refactor for the Map functionality. When a recording has multiple fragments it is currently impossible for the Map functionality to read all the fragments belonging to a recording. To make sure the Map functionality is fully functional a high recording interval could be set. It is possible to update the Map in future projects.

7.9. Option to use Mobile Data

Currently the user has the option to upload data through mobile data if WiFi is unavailable. The method that checks what type of internet connection is currently active will have to be updated as new types of connections are released. It can be changed to assume all classes that are undefined are actually Mobile Data, but this might not be the best approach.

7.10. Testing

The three main ways of testing an Android application are jUnit tests, Android Tests and Manual Testing. As stated in section 4.7 jUnit testing and Manual Testing where used in this project, but Android Tests where not. Android Tests are tests which are run on a simulated app with visual elements instead of on the code and are also called instrumented unit tests.[10] The decision was made to not use these tests since we manually tested all app functionality that was seen as ready to merge with the master. Furthermore since all of the app functionality is locked until a user has logged in and logging in opens a screen which is not easy to click past in instrumented unit tests we decided these types of tests where not necessary. If in the future it is decided that instrumented unit tests are something that should be utilized there is already code for log-in bypass in the appropriate log-in class. This function should never be used in the release of this app since it would break functionality. Test coverage using jUnit tests can still be improved by using Powermock.

7.11. Design of the umbrella

One issue that arose while recording and classifying sample data with the prototype was that the strap meant to bind the umbrella together would repeatedly touch the umbrella while recording. This caused a sound that is indistinguishable from actual rain, even for human hearing. As a result this caused some false positives in the classification. Additionally the wiring between the Bluetooth device and the piezoelectric sensor was very fragile and they often had to be reattached. It would be useful to keep this into account with the design of an umbrella for collecting rain data.

8

Conclusion

IBM wanted a mobile application that can send the data collected by specially equipped umbrellas to a server that they can access. The Weather Witness app that has been developed in this project is what provides this connection. The data that IBM is most interested in is audio containing rain. This is why aside from the Weather Witness app, audio processing and classification for detecting rain have also been implemented outside of the app.

Most of the final important requirements have been met. Because the requirements have changed during the course of the project, some of the initial Must Haves and Should Haves are not met. However this has been discussed with the client and a mutual agreement has been reached to change these requirements.

All things considered we find that this project has been a success.

Bibliography

[1] S. Bradner. *MoSCoW method*. https://www.ietf.org/rfc/rfc2119.txt [Accessed:24/06/2017].

[2] D. Mazzoni. Audacity. http://www.audacityteam.org/ [Accessed: 23/06/2017].

[3] J. Six. Decode MP3s and other Audio formats the easy way on Android, . https://0110.be/posts/Decode_MP3s_and_other_Audio_formats_the_easy_way_on_Android [Accessed: 25/06/2017].

[4] J. Six. TarsosDSP, . https://github.com/JorenSix/TarsosDSP [Accessed: 23/06/2017].

[5] J. Six, O. Cornelis, and M. Leman. TarsosDSP, a Real-Time Audio Processing Framework in Java. In *Proceedings of the 53rd AES Conference (AES 53rd)*, 2014.

[6] Unknown. *Authenticate with a backend server*, . https://developers.google.com/identity/sign-in/android/backend-auth [Accessed:21/06/2017].

[7] Unknown. *PostgreSQL Documentation*, . https://www.postgresql.org/docs [Accessed: 30/05/2017].

[8] Unknown. *Scrum Begrippen*, . http://www.scrum.nl/scrum-begrippen-agile-scrum/ [Accessed:24/06/2017].

[9] Unknown. *Activity*, . https://developer.android.com/reference/android/app/Activity.html [Accessed:03/05/2017].

[10] Unknown. *Getting Started with Testing*, . https://developer.android.com/training/testing/start/index.html [Accessed:24/06/2017].

[11] Unknown. *Fragments*, . https://developer.android.com/guide/components/fragments.html [Accessed:30/05/2017].

[12] Unknown. *Start integrating Google Sing-in into your Android App*, . https://developers.google.com/identity/sign-in/android/start-integrating [Accessed:30/05/2017].

[13] Unknown. *Sign Your App*, . https://developer.android.com/studio/publish/app-signing.html [Accessed:23/06/2017].

[14] Unknown. *Shapes*, . https://developers.google.com/maps/documentation/android-api/shapes [Accessed:24/06/2017].

[15] Unknown. *Getting Started*, . https://developers.google.com/maps/documentation/android-api/start [Accessed:24/06/2017].

[16] Unknown. *Installation PostgreSQL*, . http://php.net/manual/en/pgsql.installation.php [Accessed:21/06/2017].

[17] Unknown. *Powermock*, . https://github.com/powermock/powermock [Accessed:24/06/2017].

[18] B. Pardo Z. Rafii. REpeating Pattern Extraction Technique (REPET): A Simple Method for Music/Voice Separation. IEEE, 2012. URL http://ieeexplore.ieee.org/abstract/document/6269059/?reload=true.

Infosheet

General Information

Project Name: Sense Umbrella Connection and Desensitisation
GitHub page Android: `https://github.com/justinvdh/Sense-Umbrella`
GitHub page Server: `https://github.com/justinvdh/Sense-Umbrella-ServerSide`
The final report for this project can be found at: `http://repository.tudelft.nl`.

Description

IBM has a number of Sense Umbrellas. These are umbrellas equipped with Bluetooth device and a sensor to detect rain that falls on the surface of the umbrella by recording it as audio. Our task was to create a mobile app to connect to the umbrella and collect the data from the sensor via the Bluetooth device and to analyse the collected data. The main challenge of the project was the analysis of the data, even more so because this was initially to be done on Android. The research phase was mostly focused on how to collect the audio data and analyse it on Android. Options for processing and analysing audio on Android turned out to be very limited. In the end because of the limited availability of such resources, the analysis was moved off of Android. The product created is an Android app that is able to gather data from the umbrella and send it to an external server. Additionally an implementation has been made to analyse the audio data gathered to classify which parts may contain rain. The product is meant to be further developed in the future by IBM. In this case the app should be modified to work with the IBM servers instead of a local one and the audio processing should be tuned to work with the umbrellas that will be used, rather than the prototype.

Members of the project team

Name: Ronald van Driel
Contribution: Google sign-in, fragment utilization, map display

Name: Justin van der Hout
Contribution: Audio recording, audio processing/classification

Name: Marco Houtman
Contribution: GPS, Bluetooth, audio recording

Name: Marissa van der Wel
Contribution: Data submission and storage, software testing.

Client

Name: Z. Szlávik
Affiliation: IBM Netherlands

Coach

Name: C.C.S. Liem
Affiliation: Multimedia Computing Group

Contact

Ronald van Driel, ronald_hey_@hotmail.com
Justin van der Hout, justinvdh@gmail.com
Marco Houtman, M.Houtman@student.tudelft.nl
Marissa van der Wel, M.M.vanderWel@student.tudelft.nl

A

Project description

Task

Using IBM technology (Bluemix and its services, primarily) to tackle the problem below:

We have a number of so called Sense Umbrellas (Delft development). These are umbrellas with a sensor on them, able to detect rain as it taps on the surface (see link below for image). What we need is an app on a phone that uses the phone's Bluetooth connection to get connected to the umbrella, then collect the data from the umbrella and upload it to the (Bluemix) cloud. This is fairly straightforward so far, but we have an additional challenge here: the recorded data can be interpreted as sound, and if we play it, unfortunately, it also plays what people are saying while carrying the umbrella. This is obviously not acceptable, so what we need to tackle as well is the problem of wanting to upload data to the cloud that does not contain (recognisable) human speech. So ideally speech gets removed from the signal before it gets uploaded to the cloud.

Company description

At IBM Center for Advanced Studies Benelux, enthusiasm for innovation and research reflects everything we do. We are an IBM entity that drives and empowers the development of partnerships between IBM, clients and the scientific community, creating a lively ecosystem of the business world and the academia. Our mission is to innovate with the latest technologies in the areas of earth, health and work environment. Our goal is to become the biggest driving force of innovation and research in the Benelux, striving to disrupt what we consider as given.

B

Research report

Research Report
Sense Umbrella

by

Ronald van Driel
Justin van der Hout
Marco Houtman
Marissa van der Wel

as part of completing the bachelor project course (TI3806) of the Delft University of Technology.

Group members:	Ronald van Driel	
	Justin van der Hout	
	Marco Houtman	
	Marissa van der Wel	
Project duration:	April 24, 2017 – July 9, 2017	
Bachelor Project Committee:	Coach:	Cynthia Liem
	Client:	Zoltán Szlávik
	Project Coordinator:	Otto Visser
		Huijuan Wang

TU Delft

Contents

1 Introduction

This report has been created with the intention to summarize the findings of the research phase for the Sense Umbrella project. The research phase fully spans the first two weeks of the project.

1.1. Sense Umbrella project

The Sense Umbrella is an umbrella with a sensor which is able to detect rain as it taps on its surface, this sensor acts like a microphone and makes it possible for the user of the umbrella to record the rain tapping on the umbrella as they are walking around. The data of the rainfall will be collected in the cloud which IBM can access to use in several experiments to try and determine the rainfall's properties and possibly improve weather predictions. Because the sensor acts like a microphone it is possible that the umbrella records fragments of people talking. Because it would be a breach of privacy to record the voices and send this data to IBM, it is our job to develop an app which will make sure the recordings of rainfall sent to IBM do not contain any human speech. The goal of the app is to make the umbrella easy to use for people whom are interested to collaborate with IBM on this experiment and to find an efficient way to record the rain and send the data to IBM without breaching the privacy of the user and their surroundings while using the app.

1.2. Research purpose

The primary purpose of the research phase of the project is to define the problems that need to be solved for the project and to provide insight in the relevant solutions to these problems. Crucial to this research phase is a comparison of these alternative solutions which should result in a clear motivation for the approach chosen for the project.

1.3. Structure of this document

The next chapter will list our problem description which includes a project definition and project requirements with an explanation for specific decisions. The last chapter will contain all research done in the research phase of the project.

2

Problem Description

During the research phase the main requirements for the project have been established with the client. This chapter aims to define these requirements.

2.1. Problem Definition

Our client has a number of so called Sense Umbrellas. These umbrellas have a sensor on them that will record the sound of raining. Our task is to create a mobile phone app that will connect the phone to the umbrella through a Bluetooth device attached to the sound sensor and then have the data collected by the sensor uploaded to the Bluemix cloud. There is one additional challenge and that is that the sound sensor also records other things than rain, including human speech. This may not be freely uploaded to the cloud because that would be a major breach of privacy. Therefore we have make sure that no recognizable human speech gets uploaded to the cloud.

2.2. Requirements

Having spoken with our client in person, we have been able to define a provisional list of requirements for this project:

- Must haves
  - Connection between Bluetooth device and mobile phone.
  - Trigger recording via mobile phone.
  - Data transfer from phone to server.
  - No fragments containing speech should be send to the cloud.
  - User must be able to remove themselves and all recorded data from the program and cloud.
  - App for Android.
  - Send location and time as metadata.
- Should haves
  - Data transfer uses WiFi.
  - Use Bluemix for database service and server.
  - Do not upload data that does not contain rain.
  - Option to send data through 3G/4G.
  - Efficient energy usage.
- Could haves
  - App for iOS.

  - QR codes to connect app to umbrella.

- Won't haves
  - Immediate data transfer to cloud.
  - Let user listen to the files before uploading.
  - Support for other platforms than iOS and Android.
  - Support for Android versions under 4.3.

2.3. Requirement Justification

Besides the essential requirements the research phase also provided incentive to make some other important decisions which will have a major influence for the development of the app. This section will contain justifications for all non-obvious requirements.

2.3.1. Must have

User removal from program and cloud

After speaking with our client we have agreed that users must be able to opt out of the program, which means that all of the data connected to them will be removed from the program and cloud. For this we connect the user to their data with an account.

2.3.2. Should have

Do not upload data that does not contain rain

Because the client is interested in data containing rain there is little value in sending data fragments to the cloud that do not contain rain. Additionally, this data has a risk of containing human speech, so not uploading this data would reduce the risk of uploading human speech to the cloud.

Option to send data through 3G/4G

If a user does not have a regular WiFi connection they should still be able to upload their data. This should however always be optional since it will use internet data which has increased battery usage and could cost the user money. The primary focus will be WiFi since this is free for the user and costs less energy.

Use Bluemix for database service.

Since our client company is IBM, it is preferred if we use a Bluemix database which was developed by IBM.

2.3.3. Could have

App for iOS

We will start out developing for Android since this is the preferred platform for our client and has the larger userbase.[5] Since we will be making an app that uses functionalities with a very different implementation in Android vs iOS (Bluetooth, GPS, WiFi), we will not be able to develop a hybrid app which will be usable on both devices. This is why we will only develop an iOS app if our other requirements are implemented first.

QR codes to connect app to umbrella

This is considered a "nice to have" if we have time for it. It would be convenient if the user can connect the app to the umbrella by scanning a QR code on it.

2.3.4. Won't have

Immediate data transfer to cloud

The input data will first have to be processed before it can be sent to the cloud, which means it will not be immediately transferred. Since it will cost energy to transfer data to the cloud either via WiFi or data it would be better for the user to trigger this process themselves when they are charging their phone.

Let the user listen to the file before uploading

The recorded data might not just contain the speech from the user, but also people surrounding the user. This means they cannot decide if a file should be submitted or not by listening to the recording, which is why we will not implement this functionality.

Support for platforms other than Android and iOS

Most mobile phone users own either an Android or iOS operated phone[5]. Because of the limited scope of this project we will not be developing an application for any other operating system.

Android Version under 4.3

Target operating systems have major implications for the app development process, therefore it is important to agree on the operating for which to develop. It has already been determined that the app will be required to be developed for Android. 2.2 The difference between the versions is that recent Android versions provide more functionality but they can exclude users of older versions. For the Sense Umbrella app the most relevant functionality that has been added to Android is the Bluetooth low energy feature. This feature has been added in Android 4.3. [12] As it turned out 91.4% of Androids user base uses version 4.3 or higher. [14] This means that developing for Android 4.3 will not exclude a large amount of users and provides the most recent relevant functionality. Therefore the app will be developed for Android 4.3 and higher.

3

Research Topics

This chapter will have a separate section for each research topic and will present the findings for this specific topic. For each of these research topics the relevance for this project will be explained in the section itself.

3.1. Processing rain audio

Our task in this project is to record and upload rain audio to the IBM cloud while making sure that no recognisable human speech gets uploaded, because that would obviously be a major breach of privacy. Furthermore the uploaded rain audio should be as raw as possible, which means that the uploaded rain audio should be as close to the corresponding recorded audio as possible. Usually noise such as rain is filtered out of speech audio, but now the opposite is desired. This makes the task challenging since the existing solutions for filtering rain from speech are not necessarily applicable.

3.1.1. Detecting rain

When someone uses a Sense Umbrella while it is raining and it suddenly stops raining, the audio that the recording device is recording will not be useful as it does not contain any rain audio. In fact it might also contain speech which must not be uploaded to the cloud. If we prevent any audio that does not contain rain from being uploaded, we will be uploading less data, which is desirable for data and energy consumption, and we will have a smaller risk of uploading any speech to the cloud. To do this we must be able to detect whether an audio fragment contains any rain.
Most proposed methods for recognising environmental sounds such as rain make use of MFCC features, which are often used for voice recognition. (Chu, et al.) [8] shows that MFCC features with a Gaussian Mixture Model (GMM) classifier have a high recognition rate for the sound of rain. The recognition rate is even higher when MFCC features as well as Matching Pursuit (MP) features are used, however the difference is marginal. Similarly (Uzkent et al.) [1] shows that with a Support Vector Machine (SVM) classifier or a Nearest Neighbour classifiers (which was used as a baseline) the classification rate using MFCC features is very high.

3.1.2. Filtering speech from rain

Because it is unconventional to filter out speech audio and to keep audio that is often considered noise, there are few methods that do exactly this. One promising method is REPET (REpeating Pattern Extraction Technique) (Zafar Rafii, Bryan Pardo) [18], which separates repeating audio from non-repeating audio. This method was originally meant to separate singing and background in music, however this method might also be useful for filtering out speech audio while keeping rain audio. Zafar Rafii has provided a MATLAB implementation of this method [19].
We have tried this implementation out on audio samples containing both speech and rain. Unfortunately this method by itself does not completely separate speech and rain. The rain audio usually still contains recognisable speech, however when the output goes through a highpass filter then the speech can become unrecognisable as a result. This result still varies and the rain will become somewhat distorted, which is undesirable since the rain audio should be as raw as possible. But with certain tweaking this method may be a solution to filtering out speech from audio that contains both rain and speech.

3.2. Audio processing in Android

For this project we will need to filter out speech from an audio signal received from a microphone, on a mobile phone. To save and process an audio signal within a Android application, there are several libraries available. Just visualizing the audio data is not the type of functionality we are looking for, so all libraries focused only on audio visualization are not considered useful for this project.
Libraries are available in several programming languages including C/C++, Python and Java. General consensus seems to be that the C/C++ libraries have the potential to perform faster than Java libraries [17] when executing audio transformations, but since this would require a JNI to be compiled in Android and each JNI call would also have some overhead, the implementation would decide if a Java or C++ library would be faster. Because our project group has more experience with programming in Java, we will try to use a Java library for our project. The aim is to implement this in such a way that if the performance is too slow, it can be easily replaced with another library at a later time.

3.2.1. Java libraries

Which Java library would be optimal for our application heavily depends on the functions we will be using. The chosen library should also have good documentation, and a fast implementation of the algorithms needed. Below some Java libraries will be listed with their positive and negative aspects.

TarsosDSP

TarsosDSP is a Java library for audio processing which was created with the intention to be easy to use and is written in pure Java (no C++/C back end).[9] The authors of this article admit that this library was not optimized for fast performance which means this library might not be what we are looking for. This library does have a version which is suited for Android. Since Java and Android use different audio objects this is not the case for all libraries.

Available functions This library has basic implementations for many functions including the following:

- FFT transformation
- High and low pass filters
- Onset detection
- MFCC
- Pitch detection

JSyn

JSyn is an audio synthesis library with a Java front end and C++ back end.[4] Because this program has a C++ back end it should be better optimized than a program written in pure Java, but also means that adding new functions to this library might be difficult. Although this library seems like it has many usable features, it's documentation (including Java doc) [3] isn't always very extensive which means it might be difficult to learn how to use this library.

Available functions

- FFT transformation
- High and low pass filters
- Pitch detection

JASS

JASS or "Java Audio Synthesis System" is an audio synthesis library written in pure Java. [16]Because it is written in just Java it is easier to add new features than if it had a C++ back end. Not all available classes have proper documentation [15] to explain its use which means that we will probably not be able to use these functions.

Available functions

- FFT transformation
- High and low pass filters
- Pitch detection

Beads

The Beads Java library was written mainly as a music creation library. All audio is added to an AudioContext class and has to be "played" in order to have operations executed on the audio, but it is possible to play without actually making sound. This library does have some example applications, but not all functions are intuitive in use and/or have an extensive Javadoc [2].

Available functions

- FFT transformation
- Pitch detection
- MFCC
- Peak detection

There are few Java libraries for processing audio. Although TarsosDSP is not the fastest implementation, it is the only candidate that is properly documented and provides the required functionalities. Our plan is to use TarsosDSP for audio processing, but if it turns out it is too inefficient we will resort to another library or write our own implementation.

3.3. Location Determination

As the goal for the app is to determine rain patterns it is important to know where the rain is falling as the location is an essential part of the pattern. For example, when you are walking around in a city with a lot of high buildings and the rain comes from a certain direction, this may mean that direct rainfall has been obstructed by one of the buildings. This will result in a very different dataset than someone who is using the umbrella in an open space. As it is decided to develop this app in Android the choice has been made to solely focus on determining locations on Android phones.

3.3.1. The Challenges of Location Determination

There are several challenges in determining the location of the user as can be found on the developer page of Android[13]: The multitude of location sources, the movement of the user and the varying accuracy of the location.

Multiple Location Sources

To effectively use multiple location sources it is important to take the accuracy of the location, the speed of acquiring the location and the battery efficiency of acquiring the location into account. To determine the most accurate location the best solution would be to use all location sources and determine which of the sources both has the highest accuracy and has proven to be trustworthy. However, this would mean that every possible source should be continuously updated and this will take time and would be detrimental to the battery usage. The battery usage can be reduced with the help of short time frames in which the location is pinpointed instead of real-time updates and to restrict the mobile phone to a set amount of providers. When it comes to speed the best way to improve this is again by restricting to a set amount of location providers.

Movement of the User

The movement of the user is important as we want to determine an accurate location, but when a user is moving while collecting the data it might be sent with an out of date location.

Varying Accuracy of the Location

Accuracy is the biggest problem in determining the location, the accuracy may vary over a small timespan and it is important to determine which of the received locations is the most accurate with respect to the trustworthiness of a source.

3.3.2. Location Determination Implementations

There are several implementations which differ in the way the fragments are recorded, two of these implementations will be discussed with regards to the implementation of GPS. The first implementation focuses on short recordings (30 to 60 seconds) over several time intervals, the second implementation is focused on a continuous recording over several minutes.

Short recordings

In case of short recordings with intervals it is affordable to call upon a multitude of location sources as the location of a user can not change dramatically within half a minute and a location can be estimated on about half of this duration to get a good approximate user location. This gives the possibility to use enough time to determine the exact location of the user and with correctly determined intervals it will not interfere too much with the battery usage of the phone. Every interval can be sent to the server with a simple timestamp containing time, location and location offset.

Continuous recording over several minutes

The continuous recording reduces the amount of location sources as it takes time to determine the location with a location source and you want to update the location continuously. When using multiple location sources this will have implications on the battery duration and it can be assumed that the app will not update the location fast enough due to the time it takes for a location source to return a location. This also means that more overhead is needed when sending data to the server as timestamps should be added with a time, location and location offset due to inaccuracy.

3.4. User data connectivity

Since the user should to be able to delete all the data they have provided from the database, they should stay connected to their data. A way to achieve this is by implementing a user account. With user accounts the user will stay connected to their data even if they switch phones. Every person who has downloaded this app on Android will have done so through the Google Play Store, which means they will have a Google account which can be used as a user account for our application. By using their Google account the user won't have to create a separate account for this application which should be more user friendly.

3.4.1. Adding Google Sign-In to Android

To add Google Sign-In to an Android application you need to add a configuration file to your project which needs to be specifically generated for your project using a key to get an SHA-1 certificate.[10] This functionality needs to be shared between all developers that want to work on the user account part of the project, so we will need to share the keystore and key for the developing process. After adding the configuration file to the project and adding the corresponding dependencies Google services can be used.
For optimal security the user's email address should not be directly used when communicating with the database since this allows outsiders easy access. Google allows you to generate a token for each user which can be verified to have originated at the right source and to be for your server.

3.4.2. iOS

It is also possible to use Google Sign-In on iOS[11], so if this project gets expanded to iOS this should be easy to extend. It is possible that an iOS user does not have a Google account since they do not need a Google account to download the application through the App Store. For this situation we can have the user create a Google account directly in the application, or expand our sign-in option to use the user's Apple ID. For the user the second option would probably be preferable since creating a new account for just this application is not very user friendly.

3.5. Efficient Energy Usage

A major concern for mobile applications is the energy consumption. It is very possible that an app that satisfies the client's requirements will drain a lot of energy. In order to prevent the app from draining such a large amount of energy that it undermines the usability of the app it should be developed with energy efficiency in mind. This part of the research phase is aimed at identifying and understanding known energy saving development practices. Specifically, this chapter elaborates on the energy-saving practices related to data transmission via network in section 3.5.1 as well as energy-saving practices related to Android programming in section 3.5.2.

3.5.1. Network Utilisation

For the Sense Umbrella project any appropriate processed data that resides on the mobile phone is required to be send to the IBM cloud. It is beneficial to delay data transmission until WiFi is available. [6] This is explained using the following two findings. First, energy usage is, for both WiFi and 3G, similarly proportional to the transmission time. Second, compared to 3G networks, WiFi has a much higher data rate which means that WiFi requires significantly lower transmission times. Since there is no requirement for data to be sent immediately after processing or recording, it will be beneficial to have an option to delay data transmission until WiFi connection is available.

3.5.2. Coding Practices

Despite modern hardware related advances in mobile energy efficiency, the implementation of an app can still largely influence energy consumption. There is no straightforward connection between energy consumption measurements and energy consumption optimization. Therefore it is difficult for developers to identify for what parts of their code an alternative implementation exists that consumes less energy. This section is aimed to assist in providing the knowledge required to identify such parts. For this purpose a research containing an empirical investigation into the effectiveness of commonly recommended energy-saving practices is consulted. [7] Based on this research the following programming practices will be adopted during development of the app for the Sense Umbrella project.

- Approximately 10% energy can be saved by placing references to array length within a loop outside of the loop.

Listing 3.1: Recommended

```
Object[] array;
int l=array.length;
for(int i=0;i<l;i++)

    array[i]=null;
}
```

Listing 3.2: Baseline

```
Object[] array;
for(int i=0;i<array.length;i++)

    array[i]=null;
}
```

- Approximately between 30% and 35% energy can be saved by using direct access to object fields for reading and setting rather than using *getters* and *setters*.

Listing 3.3: Recommended

```
class obj{
    public int a=11;
    public void Setter(int b)
    {
        this.a=b;
    }
    public int Getter()
    {
    return this.a;
    }
}
```

```
class fieldAccess{
    public int Read()
    {
        return obj.a;
    }
    public void Write(int b)
    {
        obj.a = b;
    }
}
```

Listing 3.4: Baseline

```
class obj{
    public int a=11;
    public void Setter(int b)
    {
        this.a=b;
    }
    public int Getter()
    {
    return this.a;
    }
}
```

```
class fieldAccess{
    public int Read()
    {
        return obj.Getter();
    }
    public void Write(int b)
    {
        obj.Setter(b);
    }
}
```

- Approximately 15% energy can be saved by using static invocations rather than using virtual invocations

Listing 3.5: Recommended

```
class methods{
    public static int Foo()
    {
        return 0;
    }
}
```

Listing 3.6: Baseline

```
class methods{
    public int Foo()
    {
        return 0;
    }
}
```

Bibliography

[1] H. Cevikalp B. Uzkent, B. D. Barkana. Non-speech environmental sound classification using SVMs with a new set of features. 2012. URL https://www.researchgate.net/profile/Hakan_Cevikalp/publication/267782696_Non-speech_environmental_sound_classification_using_SVMs_with_a_new_set_of_features/links/54b7bf9f0cf24eb34f6ed7ff/Non-speech-environmental-sound-classification-using-SVMs-with-a-new-set-of-features.pdf.

[2] O. Brown. *Beads Java doc*, 2008. http://www.beadsproject.net/doc [Accessed: 28/04/2017].

[3] P. Burk. *JSyn Java doc*, 1998. http://www.softsynth.com/jsyn/docs/javadocs/ [Accessed: 28/04/2017].

[4] P. Burk. Jsyn-a real-time synthesis api for java. 1998.

[5] StatCounter global stats. *Top 8 Mobile & Tablet Operating Systems*, 2016. http://gs.statcounter.com/#mobile+tablet-os-ww-monthly-201611-201611-bar [Accessed: 07/05/2017].

[6] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong. Mobile data offloading: How much can wifi deliver? In *Proceedings of the 6th International COnference*, page 26. ACM, 2010.

[7] D. Li and W. G. J. Halfond. An investigation into energy-saving programming practices for android smartphone app development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, pages 46–53. ACM, 2014.

[8] C. C. Jay Kuo S. Chu, S. Narayanan. Environmental Sound Recognition With Time–Frequency Audio Features. IEEE, 2009. URL http://ieeexplore.ieee.org/abstract/document/5109766/.

[9] J. Six, O. Cornelis, and M. Leman. TarsosDSP, a Real-Time Audio Processing Framework in Java. In *Proceedings of the 53rd AES Conference (AES 53rd)*, 2014.

[10] Unknown. *Add Google Sign-In to Your Android App*, . https://developers.google.com/identity/sign-in/android/ [Accessed: 01/05/2017].

[11] Unknown. *Start integrating Google Sign-In into your iOS app*, . https://developers.google.com/identity/sign-in/ios/start-integrating [Accessed: 02/05/2017].

[12] Unknown. *Introducing Android 4.3, a sweeter Jelly Bean*, 2013. https://android.googleblog.com/2013/07/introducing-android-43-sweeter-jelly.html [Accessed: 02/05/2017].

[13] Unknown. *Location Strategies*, 2017. https://developer.android.com/guide/topics/location/strategies.html [Accessed: 02/05/2017].

[14] Unknown. *Android Dashboards*, 2017. https://developer.android.com/about/dashboards/index.html [Accessed: 02/05/2017].

[15] K. van den Doel. *JASS Javadoc for Class Silence*. http://people.cs.ubc.ca/~kvdoel/jass/doc/jass/generators/Silence.html [Accessed:02/05/2017].

[16] K. van den Doel and D. K. Pai. Jass: a java audio synthesis system for programmers. Georgia Institute of Technology, 2001.

[17] S. Verovets. *Tools for Audio Processing in Android Development*, 2016. https://anadea.info/blog/tools-for-audio-processing-in-android-development [Accessed: 27/04/2017].

[18] B. Pardo Z. Rafii. REpeating Pattern Extraction Technique (REPET): A Simple Method for Music/Voice Separation. IEEE, 2012. URL http://ieeexplore.ieee.org/abstract/document/6269059/?reload=true.

[19] R. Zafar. REPET, 2017. http://zafarrafii.com/repet.html.

C

Activity and fragment lifecycle

Activities are a single container usually used to present a user interface. Fragments contain a set of layout elements. Together they are responsible for the appearance of an app. The next page shows a figure containing the life-cycle of both fragments and activities. The figure is created from behaviour analysis and can be found on the following github adress: `https://github.com/xxv/android-lifecycle`.

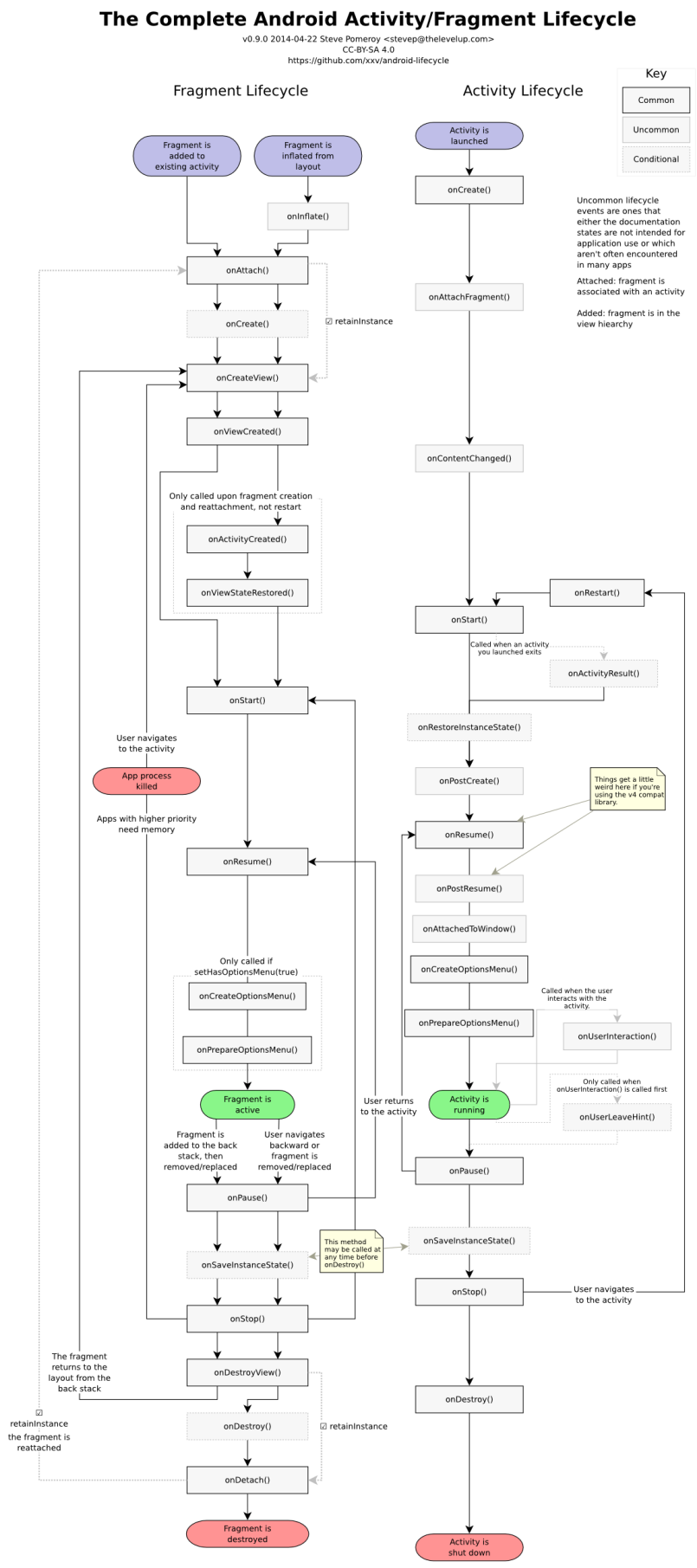

Figure C.1: Lifecycle of fragments contained within activity

D

SIG feedback

Feedback received

[Analyse]
De code van het systeem scoort 3,5 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'GPSFragment.onActivityCreated'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld 'Asks for permissions needed by the GPS' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

Adressing the feedback

As a response to the feedback, extra attention has been given to reducing the Unit Size of methods in the code. Long methods have been split into smaller methods when reasonably possible. There are some exceptions where this simply could not be done. For example writing a header for a wave file requires a byte array with 44 specific entries or certain methods for audio classification that require long for loops with a lot of variables. Splitting such methods further would only decrease readability.

Test-code has also been added for as many new functionalities as possible. Testing on Android with automatic tests is usually a challenge, because there are likely a lot of GUI elements in the code. For this reason, GUI elements have been split from non-GUI elements as much as possible in the code. This allows for testing most of the non-GUI related code.
Since the audio processing and classification did not have to be on Android anymore after the change of plans, the testability of this part of the code has increased significantly as there are almost no GUI elements.

Second feedback

[Hermeting]
In de tweede upload zien we dat zowel de omvang van het systeem is gestegen, terwijl de score voor onderhoudbaarheid licht is gedaald. Deze daling komt voornamelijk door een stijging in de hoeveelheid gedu-

pliceerde code. Op het gebied van Unit Size zien we dat jullie een deel van de voorbeelden hebben refactored, maar dat er in de nieuwe code ook weer lange methode zijn bijgekomen.

Wel is het goed om te zien dat jullie naast nieuwe productiecode ook aandacht hebben besteed aan het schrijven van nieuwe testcode.

Uit deze observaties kunnen we concluderen dat een deel van de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.