CA400 TECHNICAL GUIDE

Chirp – Audio Social Updates

Eoin O'Brien & Kieran Flynn
15324971 & 16334663

Date Completed: 15/05/2019

Abstract

Chirp is an android mobile application that will enrich the experiences of users on the popular social media platform Twitter. The application at its core allows a user to record & share a voice message for their followers to listen to, as well as receive audio updates from selected Twitter users.

1. Introduction

1.1 Overview

The aim of this application is to add a new dimension to Twitter which allows users to interact with each other in novel ways. It enables users to keep up to speed with various social, political & sporting functions without the need to scroll through a Twitter feed looking for relevant content. It does this by playing back recorded audio files that have been previously shared via our application without any custom server infrastructure.

1.2 Glossary

API - *An **A**pplication **P**rogram **I**nterface is a set of routines, protocols and tools for building software applications.*

REST - *A Representational State Transfer is an architecture designed for web services and distributed hypermedia systems. The transfer sends hypertext to and from online platforms such as Twitter and Google.*

Android Studio - *Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.*

JSON - ***J**ava**S**cript **O**bject **N**otation is an open standard file format, and data interchange format, that uses human-readable text to store & transmit data objects consisting of attribute-value pairs and array data types (or any other serializable values).*

JUnit - *JUnit is a unit testing framework for the Java programming language.*

Mockito - *A mocking framework for unit tests written in Java.*

GET Request - *A **GET** request is used to request data from a specified resource using a given URL. GET requests are one element of the greater REST architectural style.*

POST Request - *A **POST** request is used to send data to a specified resource using a given URL. POST requests are one element of the greater REST architectural style.*

Hashtag - *A keyword or phrase used to describe a topic or theme, which is immediately preceded by the pound sign* (#)*.*

Feed - *A social media **Feed** is an updated list of content created by people that specific users follow.*

Tweet - *A Tweet is a social media post created for the social media platform Twitte*r.

Toast - *An user feedback tool which displays messages to the user of the application.*

URL - *A **U**niform **R**esource **L**ocator is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. URLs are a subset of the broader URI mechanism.*

IDE - *An **I**ntegrated **D**evelopment **E**nvironment is a software application that provides comprehensive facilities to computer programmers for software development.*

2. General Description

2.1 Motivation

The motivation for this project came from a desire to learn more about mobile application development, but also to create something that we would actually use in our day to day lives. It seemed fruitless to create a project that would never be used or thought about again, and as such we endeavoured to come up with a real solution to a real problem.

We believe the application could be useful for users who may want to stay informed about a particular event or topic, whilst being too busy to keep up to speed in the traditional manner.

2.2 Business Context

The application definitely has a potential use in the social media industry as it enables a new level of communication between users that Twitter does not currently facilitate.

A major appeal of Twitter is how seemingly lightweight it is when compared to some of its' competitors such as Facebook or Instagram. From start to finish, it is incredibly easy to post and share tweets. We tried to replicate this simplicity when developing our application whilst trying to add something that is currently lacking.

2.3 Research

2.3.1 Android Application Development

Having never developed an android application before, this was quite a steep learning curve when it came to implementing features within the application. Throughout the development process we made use of the Android developer documentation & tutorials to learn about key topics and ideas in relation to Android development. We also regularly created test projects to develop and validate the implementation of a specific feature. This allowed us to make mistakes, break things, and therefore fix things in a controlled environment. The Android developer documentation was particularly useful when creating our initial "Hello World" application. The Google Developers online documentation aided the development process when performing actions such as HTTP requests & asynchronous tasks.

Throughout the application development process we made use of well-moderated blogs and forums such as StackOverflow. We found a lot of useful information that helped to bolster our understanding of various topics we were dealing with. With that being said however, we definitely also came across a lot of stale or outdated information that slowed down some development areas.

2.3.2 Twitter & Google Drive API

The Twitter and Google Drive API are both forms of REST API. These services allow for registered applications to interact with content on their platforms in the manner of HTTP requests. To make an application that implements these platforms, the app needs to be registered on the developer platforms for both Twitter and for Google Developers. These developer platforms also come with guides on how to correctly perform actions with their interfaces. You are provided with an API Key and a secret consumer token that is assigned to your application specifically. This is then called in the build of the client used to interact with either platform.

While the Google Developers platform had extensive examples of how to perform actions with Google Drive using languages such as Java, the Twitter documentation was solely focused on the relevant URLs. Thankfully, there were extensive resources available on online platforms such as Stack Overflow discussing clients to use such as the Apache HTTP Client and the open source Java resource Twitter4J. These discussed how to perform HTTP Requests with the Twitter platform using languages such as PHP, Python and Java.

Due to the nature of the collaboration between the two platforms, there was little documentation available on how to integrate the two web services. This meant a significant amount of time on our part was used to experiment with trial and error to integrate the two platforms.

2.4 System Functions

The system is an Android application with four distinct functions that share some implementation methods.

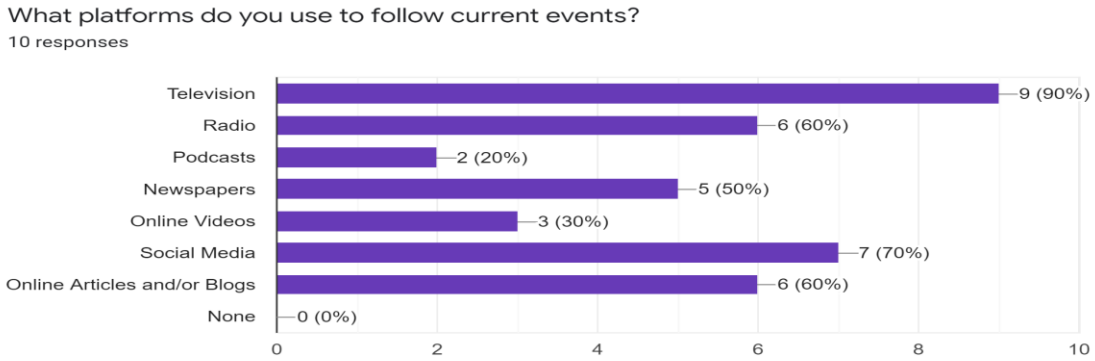
The functions are as follows:

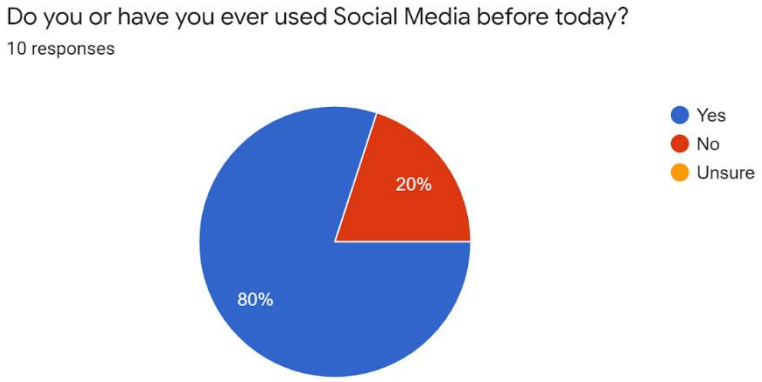
1. Record & Share audio
2. Go Live - Receive updates from a specific Twitter account with the aid of a user entered hashtag
3. Browse Chirp - Browse the user's home timeline of Chirp tweets
4. Users Own Tweets - Browse the user's own Chirp tweets

These functions are laid out in the form of four buttons on the application's home page with each button corresponding to a feature. These functions are explained in detail in both the User Manual & Video Walkthrough.

2.5 User Characteristics & Objectives

We imagine that the characteristics of our user base would be like those of consumers of social media and current events media. The two formats have been closely tied together for more than a decade with the popularity of online content rising compared to traditional platforms like television and newspapers. In this growth period for online media, platforms have tried to integrate their existing knowledge for the traditional multimedia platforms such as Radio into their online presence. The objective of this project was to provide a platform of use to content creators that can integrate their output into their social media. We also wanted to use the platform as a means for content followers to have a means to gather quick multimedia updates on the events they wish to follow such as sport or politics. This avoids them having to use long format, dedicated platforms such as Podcasts and programmes.



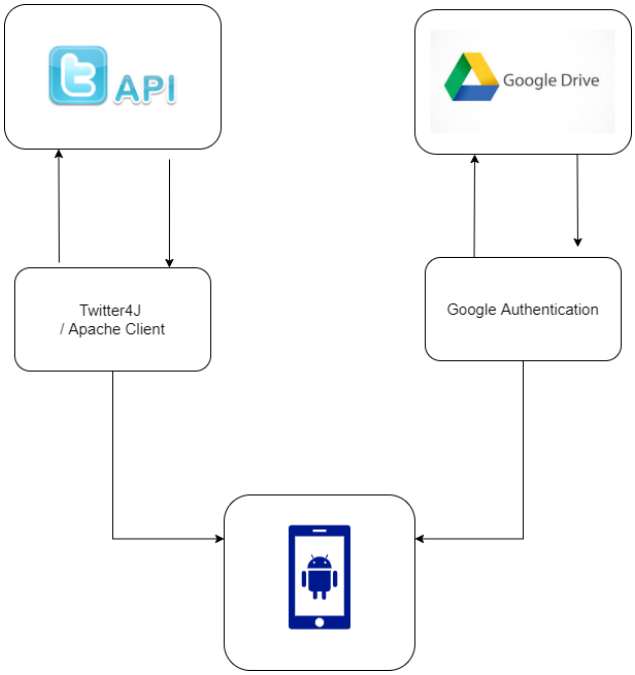


3. Design

The general design of the overall architecture did not radically change during the development process; however, it became significantly more refined than what was outlined in the functional specification.

One change that did have to be made was the iOS version of the application being dropped. It became apparent during the research period that there were not sufficient resources to implement native integration compared to that of Android OS. Any implementation using Google Drive or Java resources such as Twitter4J would have to be done using Maven. That's on top of any issues encountered with accessing Apple devices in DCU due to the COVID 19 lockdown. However, this did not alter the design or implementation of the system as designed.

3.1 System Architecture



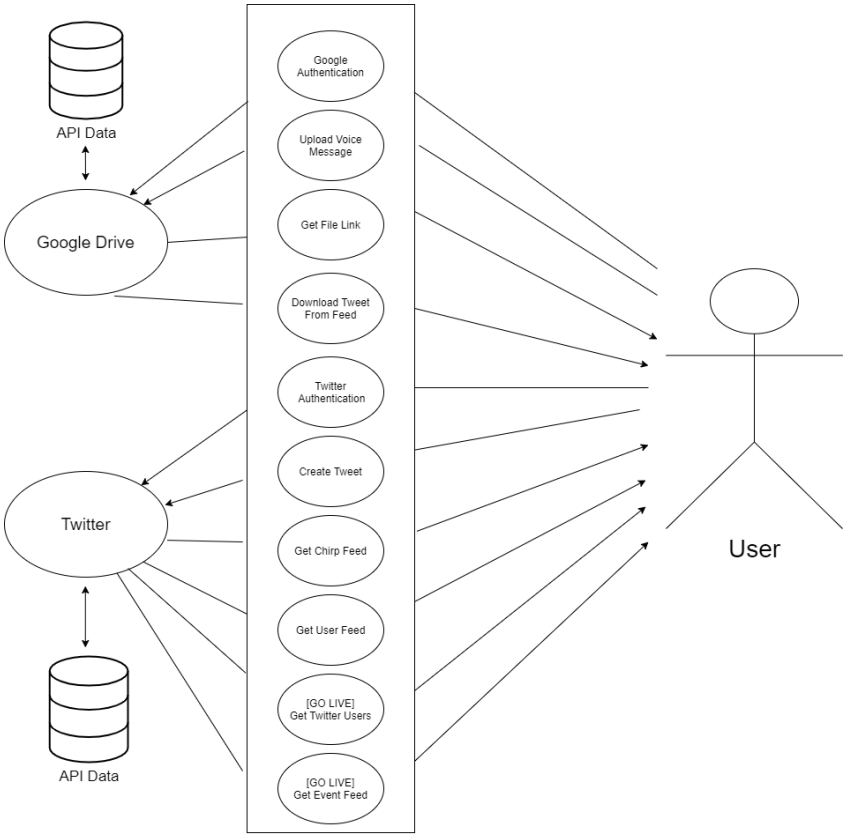
As can be seen above, the system architecture primarily consists of the Android application, which the user interacts with. The application itself makes a series of calls to both the Twitter & Google Drive APIs to perform the various functions detailed previously.

A major aim of the project was to implement the idea without the use of any custom storage infrastructure. That is to say, we aimed to make use of already existing, everyday platforms such as Twitter & Google Drive.

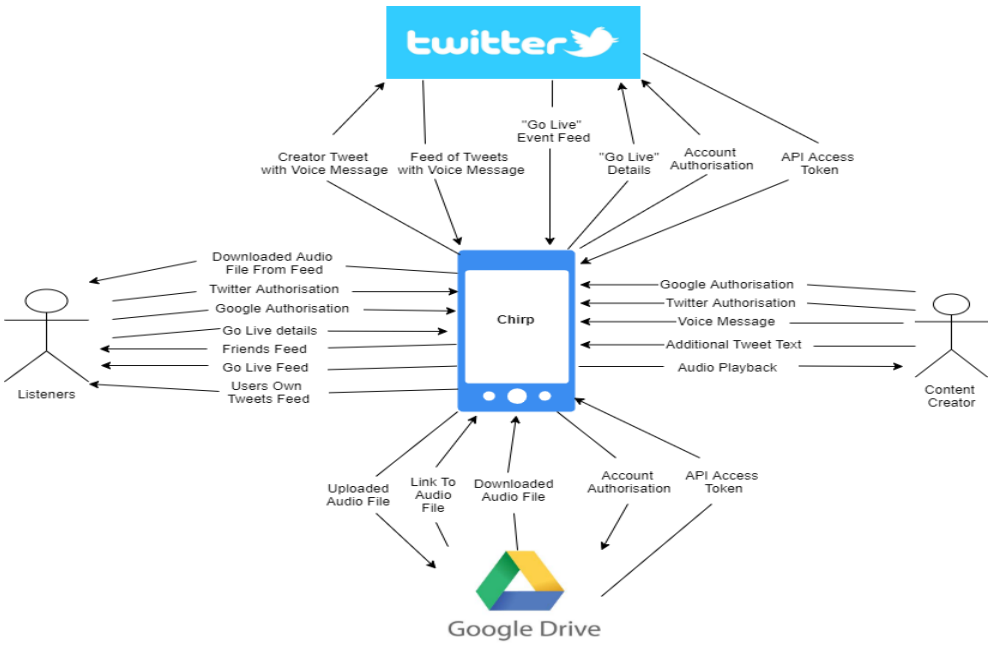
We chose Twitter for our sharing platform as it seemed more straightforward to interact with than other social media platform APIs such as Facebook or Instagram.

We chose Google Drive over some alternatives such as Dropbox due to Google Drive being far more common for everyday users and the ease in which it could be implemented into Maven based Java.

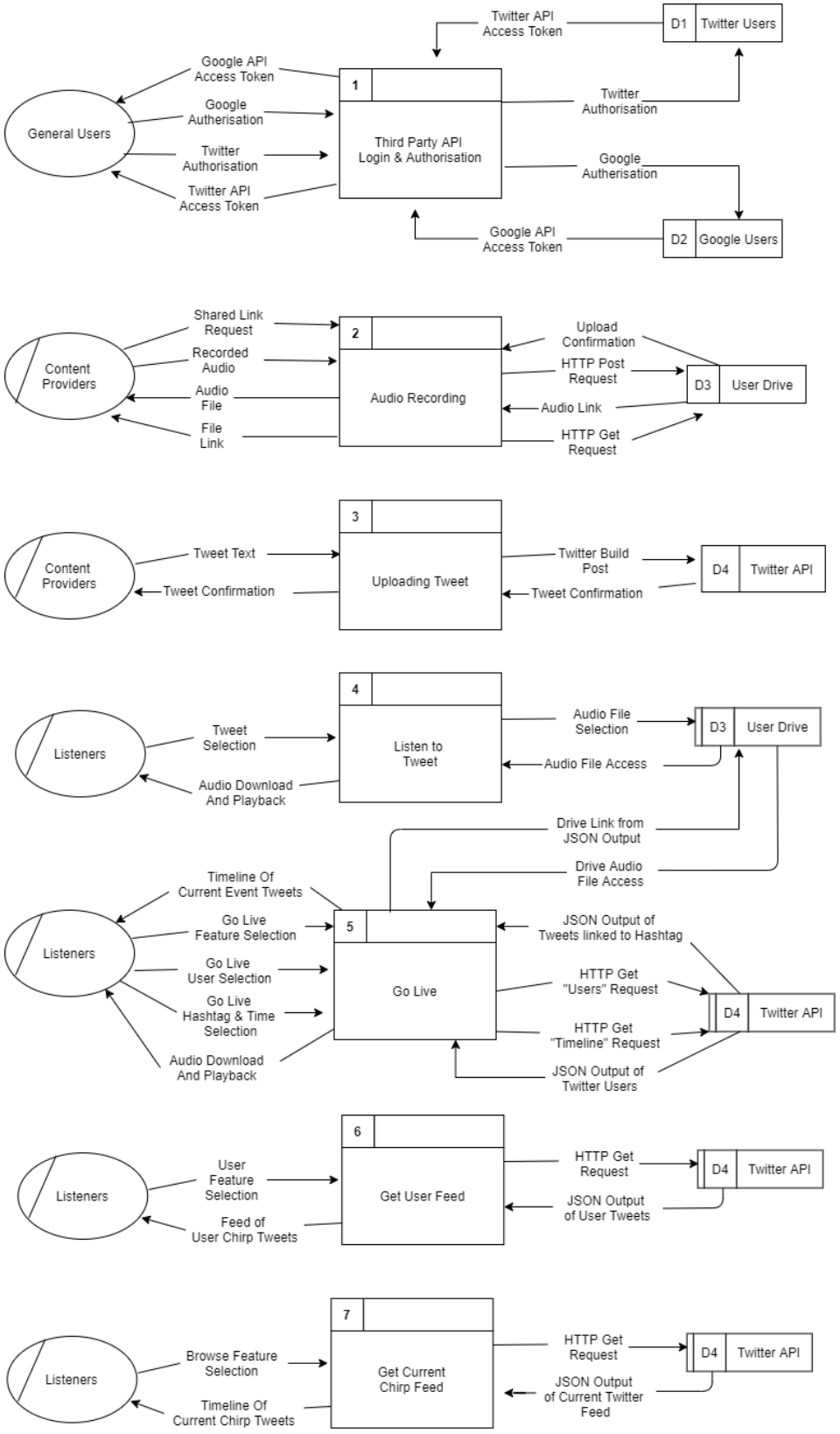
3.2 Use Case Diagram



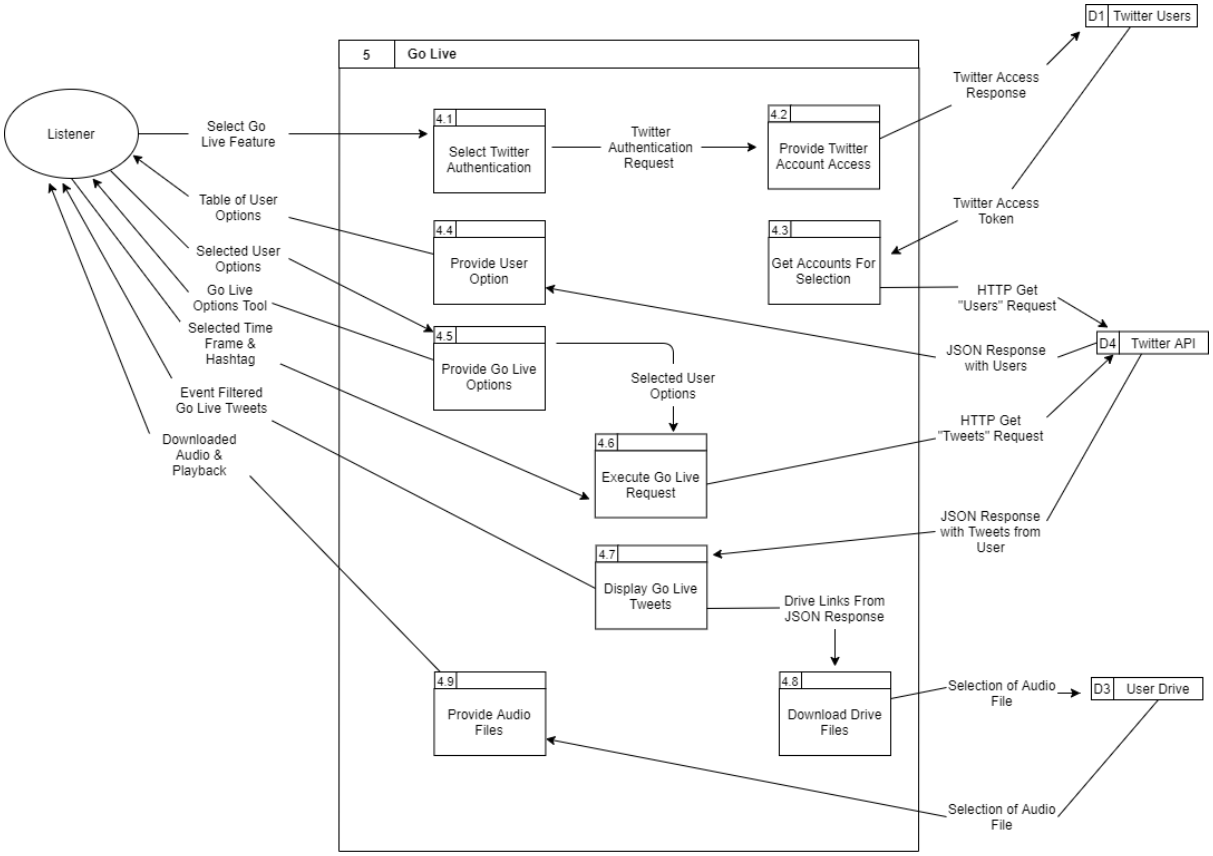
3.3 Context Diagram



3.3 Data Flow Diagram - Level One



3.3 Data Flow Diagram - Level Two



4. Implementation

4.1 Application Framework

The application was created using Java with Android Studio used as the IDE. Android Studio is the official integrated development environment for Google's Android operating system, with many features that aided us during development. It ensured that the development, integration, build and deployment phases were smooth and reliable.

In a general sense, the application consists of a series of Activities that perform various functions. The front end of the application is displayed via xml layout files, which different activities can reference when rendering the display.

4.2 Record & playing Audio

Recording & playing audio was one of the first features that was implemented due to it being such a core element of the application. This functionality makes use of the Android MediaPlayer & MediaRecorder classes. The two features are implemented

via a series of methods that each perform a number of various tasks. Taking the method *startRecording()* for example:

```
private void startRecording() {
    recorder = new MediaRecorder();
    recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    recorder.setOutputFile(fileName);
    recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_WB);
    try {
        playButton.setEnabled(false);
        recorder.prepare();
    } catch (IOException e) {
        Log.e(LOG_TAG, msg: "prepare() failed");
    }
    recorder.start();
}
```

The MediaRecorder object is first initialised, after which a series of set attribute methods are called. These methods set the source of audio to the device microphone, set the output format to .3gpp, set the output file to the *fileName* variable and set the audio encoder to the AMR Wideband audio codec.

The other methods for recording & playing perform similar tasks related to their own specific contexts.

4.3 Sharing a recording (Uploading to Drive & Posting to Twitter)

4.3.1 Uploading to Google Drive

Uploading to Google Drive is implemented in the *GoogleDriveService* class. *GoogleDriveService* consists of a series of methods that can be used to interact with the Google Drive API. An example of one of these methods is the *uploadFile* method, which is called from *ShareRecordingActivity*, which is where the user will be uploading the recording they have previously created.

```
public Task<GoogleDriveFiles> uploadFile(final java.io.File localFile, final String mimeType, @Nullable final String folderId) {
    return Tasks.call(mExecutor, new Callable<GoogleDriveFiles>() {
        @Override
        public GoogleDriveFiles call() throws Exception {
            // Retrieve the metadata as a File object.
            String fileName = RecordAudioActivity.fileName;

            List<String> root;
            if (folderId == null) {
                root = Collections.singletonList("root");
            } else {

                root = Collections.singletonList(folderId);
            }

            File metadata = new File()
                    .setParents(root)
                    .setMimeType(mimeType)
                    .setName(fileName)
                    .setCopyRequiresWriterPermission(false)
                    .setWritersCanShare(true);

            FileContent fileContent = new FileContent(mimeType, localFile);

            File fileMeta = mDriveService.files().create(metadata, fileContent).execute();
            GoogleDriveFiles GoogleDriveFiles = new GoogleDriveFiles();
            GoogleDriveFiles.setId(fileMeta.getId());
            file_id = fileMeta.getId();

            //Used to download the latest recording - mostly for testing 28/04/2020
            ShareRecordingActivity.file_id=file_id;
            GoogleDriveFiles.setName(fileMeta.getName());
            return GoogleDriveFiles;
        }
    });
}
```

Within this method the ID of the uploaded File is also retrieved, which will be used in the *getShareableLink* method that is called from *ShareRecordingActivity* on successful completion of the above.

4.3.2 Posting to Twitter

Posting to Twitter from the application can only be performed provided a file has been successfully uploaded to Google Drive. This is to prevent situations where the Tweet Composer is passed a null Drive Sharing Link.

The activity used to post tweets can be seen below. It simply creates a TweetComposer.Builder object and passes it the pre-defined variables *driveSharingLink* & *chirpHashtag*.

```
public class PostTweetActivity extends AppCompatActivity {
    String driveSharingLink = "\n" + ShareRecordingActivity.driveSharingLink; //Sharing link used to share the google drive file
    String chirpHashtag = "\n#chirpaudioupdates"; //Our designated Hashtag to find tweets by this application easily

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_post_tweet);

        TweetComposer.Builder builder = new TweetComposer.Builder( context: this)
                .text(chirpHashtag + " " + driveSharingLink);
        builder.show();
    }

}
```

The TweetComposer.Builder will default to opening in the device's Twitter application, however if this is not installed or missing it will open the Tweet Composer in a browser window instead.

4.4 Go Live

The core feature of Go Live is the GET request to find Tweets by a specified user along with a specific user-entered hashtag. This takes place in the *GoLiveTweetsActivity* class.

We had initially intended to implement GoLive as a continuous, regularly refreshing feature that would GET tweets every 5 minutes or so. However, due to the GET statuses/user_timeline route being rate limited to 1500 requests per 15 minutes, we acknowledged that this may not be the best approach. We instead implemented the pulling of Tweets statically so that it occurs as a part of the core feature pipeline. Although this is not what we had originally planned, it is a necessary evil to modify the implementation of features as more information is gathered about the tools being used.

Several steps are involved with creating & sending this GET request successfully with the correct authentication. Doing this required a lot of reading documentation & trial and error type experiments.

The results from the above GET request are returned in a JSON format. These results are parsed, looking for patterns in the text objects that match the pattern of a Drive Sharing link. This pattern is explicitly defined as is seen below.

```
//Used in pattern matching below to find the file id
String pattern = "expanded_url\":\"https:\\/\\/drive.google.com\\/file\\/d\\/";
```

From here the JSON result is parsed, looking for instances of the specified hashtag. When a relevant tweet is found, it is added to an *ArrayList* which is eventually passed to a *TwitterAdapter* for rendering.

```
// send the tweets to the adapter for rendering
obj_adapter = new TwitterAdapter(getApplicationContext(), al_text);
lv_list.setAdapter(obj_adapter);
```

The pattern used allows us to extract file IDs from the links found in the tweets. From here, the relevant file ID's (namely the tweets containing the user entered hashtag) can be passed to the download function. This download is designed to take place automatically rather than relying on a user to explicitly tap on a tweet, as there would be a delay between the user tapping the tweet and the tweet being played otherwise. During this time the UI is still usable for the user which is something we intentionally considered when designing these features.

Once the recordings have been downloaded, which typically takes between 45 and 60 seconds, the user can tap on any of the rendered tweets to have them played.

```
lv_list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        startPlaying(position);
    }
});
```

Lastly, Go Live contains a feature whereby when the user-entered number of hours has elapsed they will be taken back to the main application menu. This is done by using the Java *Time*r class and its built-in schedule method.

4.5 Additional Functionality

The other two major functions within the application, namely Browsing a user's chirp home timeline & viewing the user's own tweets, were performed similarly to above using different GET routes. Similarly, to the Go Live platform, it uses a HTTP Client to gather an access token to access Twitter content and then collects a JSON Response of tweets based upon the URL used within the GET request. This consisted of User Timeline for the users own tweets and home timeline to get an updated feed of tweets from the users' friends.

4.6 Source Control Management

We used the prescribed DCU Gitlab domain to manage all source code. During development, we tried our best to make use of branches for different features to prevent breaking pre-existing functionality. One thing that we feel we could have used, and should have used, is some sort of code review system on pull requests. This would ensure that code is of a higher standard & quality throughout the development process. It would also benefit learning to work in a team environment as the majority of enterprise software development will take place in large teams, with code review processes.

5. Problems & Resolutions

5.1 External Web Service Defects

The idea of this app is that it implements online web services like Google Drive and Twitter in order to avoid the need of maintaining a “back end” database infrastructure. All tasks involved with creating and accessing data is handled between the HTTP Clients and the API for both web services. But the major problem that became apparent with this approach is that you must rely on the quality of a system that we could not change. As a result, there were a few problems we encountered due to defects in both services.

With Twitter, we occasionally had issues signing in as the website would throw an error when the user tried to give the app access. This was narrowed down to issues with the API credentials for our Twitter app and to do with the callback URL for the app. We also had issues using certain operations provided by Twitter in their API. The search tool in the API is designed to search the platform for tweets with given parameters such as user name, hashtags or location. However, these tweets are filtered based on what the Twitter system deems to be relevant. And in the case of our apps hashtag “#chirpaudioupdates”, it would ignore tweets that contained a link to Google Drive. This causes the process of playing and downloading the audio files to fail. As a result, we had to focus on the user timeline methods that get content associated with specific users.

Meanwhile, the Google Drive API had its own unique issues. Part of the functionality of this app is that you can provide a shareable link to the file that is uploaded to the Drive platform. However, while the app can create the file link and create the permissions needed to make the file available to anyone that has a link, the link can only be shareable by selecting the “Get Shareable Link” button in Google Drive’s own website. The API does not allow external applications to change the configurations and settings to allow this to happen. When the get shareable link button is pressed for the specific file, the link provided by our app will work and the audio files can be downloaded. However, this is an annoying obstacle in the content creation process and it is an issue we feel Google should address in future.

5.2 Deprecated API’s and ToolKits

A major issue we encountered when developing this application was the changes both Google and Twitter made to their external API’s. In the case of Twitter, the task of interactions between the platform and a mobile device would have been simple due to the availability of the Twitter Mobile Kit and its documentation. However, this mobile kit was deprecated in 2019. Although the app was still able to use this kit to perform actions such as logging in to the site, it meant that alternatives would need

to be sought in order for data to be pulled from the Twitter API, such as the open source Java tool Twitter4J. Google also has made substantial changes to their existing tools. Google not only shut down their own mobile kit for Google Drive in 2019, they also shut down their Apache services for HTTP Requests. This not only impacted how our app would interact with Google Drive, but also how we would perform interactions with the Twitter API. Thankfully, the overall Google Drive API was easily integratable into Android Studio and Apache HTTP Requests were still possible by calling Java Apache services itself.

One major downside of the Google API changes, for example, was that it made gathering information on implementation of features very challenging. Many of the Stack Overflow posts, Google forum posts etc. were outdated or contained information that was no longer relevant or correct. As a result, a lot of time and energy was spent reading the official documentation and trying to gather other pieces of information from reliable sources.

5.3 Ensuring correct parameters / authentication for Post & GET requests

As mentioned previously, correctly implementing the various HTTP Post & Get methods required a large amount of time to be spent reading the documentation for each API as well as some element of trial and error. Having limited experience with APIs before the project meant that this was quite a steep learning curve for both of us. However, with the use of some (albeit limited) worked examples & the relevant documentation we feel that we have done a good job in implementing the various API calls throughout the application.

5.4 Impact of Covid-19

The lockdown procedures as a result of the COVID-19 pandemic no doubt had an impact on our ability to perform. Both members of this project group are from different counties in Ireland and have not the same facilities available at home as they would on the DCU campus. The lack of availability of Apple devices in our homes meant that the Android version of the application had to be prioritised. Our initial plan also had intended on performing User Testing with members of the industry that could give clear insight into how to improve our user interface. Meetings also had to be performed online which varied in quality as the demands on our bandwidth came under pressure as other family members tried to work from home. It was and still is an unprecedented obstacle that had a major impact on our project.

Another major downside of the Covid-19 pandemic in relation to our project, which we have only fully realised in recent days, is that a lot of casual communication between the two project partners was lost. This might seem like a small issue; however, we feel that in such a small development team, tacit knowledge is

incredibly valuable. This sharing of tacit knowledge is often what leads to effective defect resolution and progress tracking. With the social distancing measures that were enforced by the Irish government greatly reducing this sharing of tacit knowledge, we feel that in hindsight the use of error tracking boards or even general progress tracking boards are something that we should have made more use of.

5.5 API Call Implementation

Throughout our application, API calls and requests are made statically and repeatedly. When a user navigates to GoLive, a GET request is formed. Looking back this was not a particularly scalable solution and given the application is an extension to a social media platform, this is an issue.

If we were to undertake a project such as this for a second time, I think we would have put far more emphasis on scalability & portability of the application. For example, having a core server with a series of thin clients would mean that we could implement solutions such as caching GET request data for users to pull from rather than performing the requests multiple times.

In essence, this would make the application more robust, usable & testable.

6. Validation & Testing

Android Studio proved very useful when it came to testing our application. Built into the IDE is the functionality to create & update build configurations. With the click of a button, and selected configuration, the entire application can be built & installed onto a mobile device or emulator. Naturally this was particularly helpful at identifying any build errors that may have been introduced during development.

The IDE also allows a user to create configurations to run suites of tests. We often used a configuration that ran all of our unit tests with code coverage throughout development.

On reflection of our development process, we feel that some sort of bug tracker would have proved to be crucial given the collaborative nature of development. Even something as basic as a Trello board tracking tasks & bugs would have aided us in our endeavour to create a well-rounded application.

6.1 Ad-Hoc Testing

Throughout development of the application there was frequent ad-hoc testing undertaken. This came in the form of specifically trying to break the application’s functionality from an end-user’s standpoint. For example, tapping the “Start Playing”

button while a file was being recorded. The solution to this example was disabling the audio player button while recording was taking place, and vice versa.

```
private void startPlaying() {
    player = new MediaPlayer();
    try {
        recordButton.setEnabled(false);
        player.setDataSource(fileName);
        player.prepare();
        player.start();
    } catch (IOException e) {
        Log.e(LOG_TAG, e.toString());
    }
}
```

This type of testing was very valuable to us as it allowed us to attempt to identify some of the errors that would inevitably be encountered by an actual end user, as opposed to someone who is intimately familiar with the design, structure and implementation of each feature within the application.

Another example of this type of development focused ad-hoc testing comes in the form of using Listeners throughout the application. Listeners proved to be an invaluable tool to ensure specific actions have taken place, or specific results have been returned. Some examples can be seen below.

```
player.setDataSource(createdFileName);
player.prepare();
player.setOnCompletionListener(mp -> stopPlaying());
player.start();
```

The above onCompletionListener calls the *stopPlaying()* method when the audio file being played is completed. This listener ensures that errors related to multiple data sources being set are kept to a minimum.

```
mDriveServiceHelper.uploadFile(new java.io.File(fileName),  mimeType: "audio/mp3",  folderId: null
        .addOnSuccessListener(GoogleDriveFiles -> {
            Gson gson = new Gson();
            Log.d(TAG,  msg: "onSuccess: " + gson.toJson(GoogleDriveFiles));

            //Get the shareable Drive link
            mDriveServiceHelper.GetShareableLink()
                    .addOnSuccessListener(s -> {
```

The above onSuccessListener ensures that the *getShareableLink()* method is only called provided the *uploadFile()* method is performed successfully. This is vital in this particular scenario as the *getShareableLink()* method will obviously fail if there is no file to get a sharing link for.

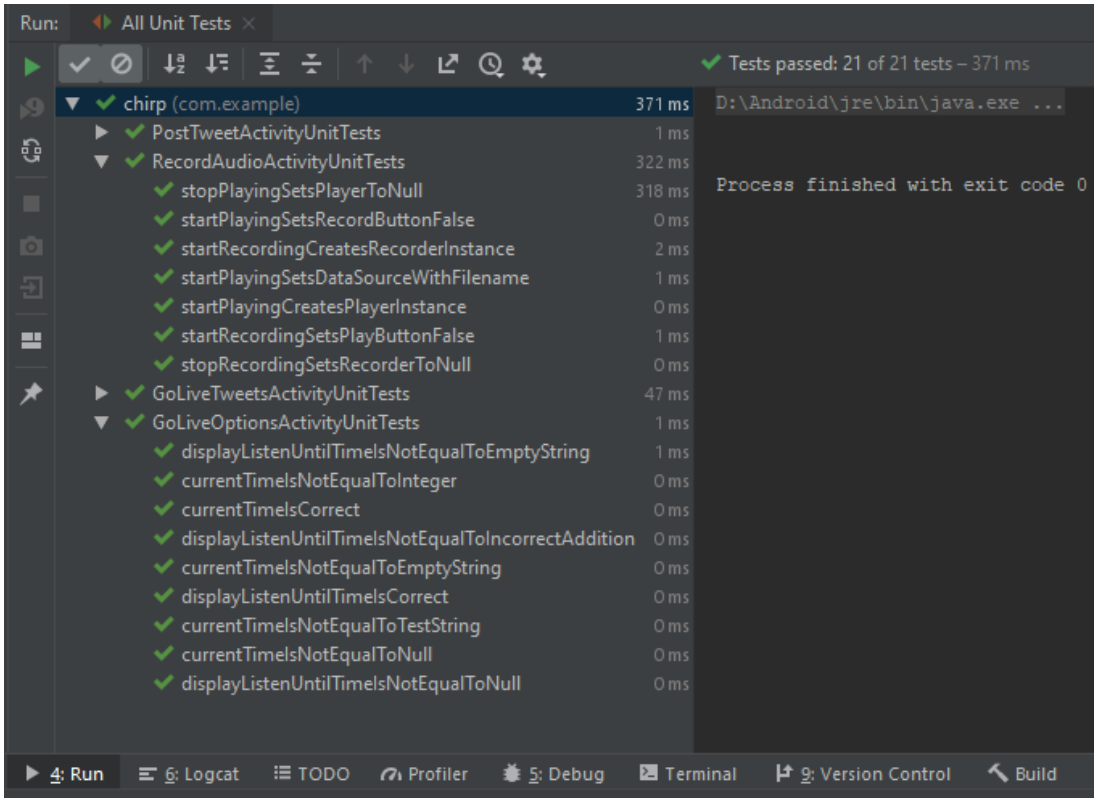
6.2 Unit Testing

We made use of the JUnit framework along with Mockito for our unit tests. These unit tests proved invaluable during the development life cycle to ensure that when adding new features we did not break or modify previous functionality unexpectedly. These unit tests definitely helped us to have some peace of mind in this regard.

One Issue that was encountered whilst unit testing our application is that we struggled to accurately test methods or classes that were private. In order to unit test them, given we were unable to get some of the mocking features implemented, we were forced to make some of these subjects under test public. We fully acknowledge that the process of making some methods & classes public in order to test them is far from best practice. Rather than this we would

As mentioned previously, using a “Run All Unit Tests” configuration, running these unit tests at the click of a button was very easy & straightforward.

Due to time constraints our test coverage was quite a bit lower than we would have liked and if we were to start a new project we would definitely aim to test more, test more often & aim for higher code coverage percentages.



One change that we would definitely make going forward, or when starting a new project, is making use of a dedicated UI testing framework. We found it extremely challenging at times to test UI elements with how they had been implemented. This whole process demonstrated quite clearly how much thought needs to be put into

testing before development of features begins, and certainly highlights the importance of development methods such as Test-Driven development.

6.3 Integration Testing

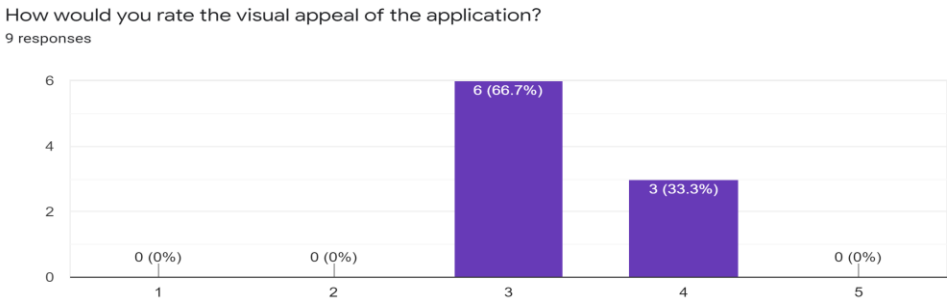
Although time constraints & implementation methods made integration testing not feasible for us to carry out in a robust sense, given more time we would have loved to try to improve this area of testing within our application. The majority of integration testing we performed was Ad-hoc or manual testing.

For example, during development, when uploading a file to google drive the resulting shareable link was output to the Log within the IDE. A simple test we performed was to ensure that the link actually worked, that it linked to the recording previously uploaded and that the link was passed to the TweetComposer correctly.

Performing tasks such as the above automatically would have greatly aided the quality of our application and is definitely something we would endeavour to improve upon moving forward.

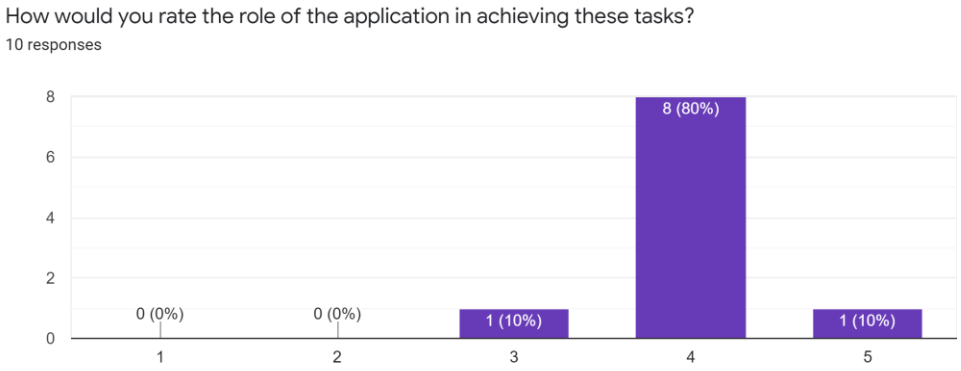
6.4 User Testing

For our User Testing, we decided to survey a sample of acquaintances, friends and family on their experience with our application. We gave each user a number of tasks to perform in our app and they would then answer a number of questions that discussed their experience with performing these tasks. We had initially planned on performing this study on site with members of the computing industry. However, this plan was not possible due to the restrictions of the COVID 19 lockdown. These restrictions meant that our sample size consisted of people who were within two kilometres of both members' residence. When testing was performed outside of our household, there were no more than two people in each room and a social distance of at least two metres was observed. Face masks & gloves were also worn at all times during the interaction. Here are our findings from the resulting survey:

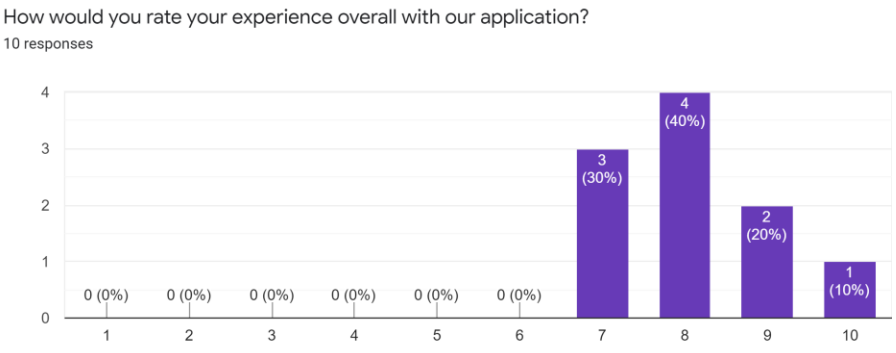


While discussing the user interface used in the application, the average user rating was 66 out of a hundred. Overall, this was a quite positive reaction to the

implemented interface. When asked for feedback on this score, the users cited issues with the color scheme of the app and the spaces between fields for user input. While those with more experience with the computing industry recommended resources that could have been used to improve the interface.



The users from the study were far more positive about the role our application played in performing the tasks they were given. They felt that the tasks were easy to perform and each member of the sample was able to complete the provided tasks successfully. That said, the users did find room for improvement in the application. Some users found the names given to features such as Browse and Go Live to be quite vague. They were not sure what the terms meant which caused an obstacle to get past to in the task completion process. Certain members of the study who deemed themselves to be “not tech savvy” also made mistakes performing the tasks such as misselecting tweets to listen to. They recommended a tutorial tool be implemented in order to assist those with less social media experience.



The users were also asked about their general thoughts about the idea as a whole. They were quite positive in their thoughts about the idea. They felt it was an effective combination of multimedia for current events and social media. Though some did not see it as a potential replacement for existing news media, they felt it was a

worthwhile expansion to both platforms and would happily use the Chirp platform again.

Overall we found the process of this user study to be quite beneficial to the development process. It served as verification of the existing implementation of the application as well as scope for potential improvements in a future expansion of the idea.

6.5 Future Testing / Learning Outcomes

Looking back on our approach to testing, we feel that a more Test-Driven Development approach would have greatly aided our efforts to implement robust testing suites and suitable code coverage percentages. We felt that our design choices somewhat hindered this & it is definitely a far bigger consideration than we gave weight to. Moving forward this is without a doubt an issue we would seek to rectify and improve upon.

7. Results

7.1 Future Work / possible expansion of project

As a media service, there would be quite a bit of expansion that could be made to this idea to make it a commercial grade product. While the Go Live feature aims to provide users with live updates on the events they want to follow, there is the barrier of it being based around individual tweets and voice messages. So perhaps a method around that in a commercial setting would be livestreaming of voice recordings similar to radio. Then perhaps adding a live comment section for listeners to participate in the discussion with their favourite creators would also add to the experience. Although that would certainly not be possible using Google Drive, there may be an implementation to base it upon judging on the success of platforms for video live streaming such as Twitch and Twitter's own Periscope. These additions would also require content moderation systems so that potential online harassment could be avoided.

Since the app is implementing a lot of features from Twitter, with more time we wish we could have implemented some common features for Twitter users such as the ability to search for tweets, like and retweet of tweets and deleting tweets. We experimented with using search as part of other features but it would ignore tweets that had our sharable link to the audio file. And integrating likes and retweets would involve modifications into how our tweets would be displayed within the app.

An issue that was raised in our project proposal might also be addressed in terms of accessibility. Obviously the process of listening to audio files is an issue for people with hearing issues and/or disabilities. Although a speech to text algorithm was

outside the possible scope of this project, it would be a good way of providing subtitles to make the Chirp platform more accessible to those affected users. This may have also been solved with a user form where volunteers could manually input text to serve as close captions or for translations for different languages.

The app could also be possibly integrated with platforms for podcasts such as Spotify or Castbox. In a commercial setting, the ability to instantly export the audio recording to those platforms would be an amazing way to entice content creators to our platform & would encourage live participation with their content. The application could also use certain features used by those platforms such as custom playlists without the need to search for specific users on Twitter. We could have also helped content creators by implementing displays for statistics on how users interact with their content such as how many people listen to their tweets & for how long.

7.2 Conclusion

Reflecting back on the entire process of designing, developing, testing and using Chirp has reminded us both of how far we have come during the past year. Even when comparing the finished application to the design set out in the Functional Specification, we both noted how much the original idea has been expanded upon. Neither of us had any or much experience working with API's and developing mobile applications. However, throughout the process of developing Chirp, we gained a lot of insight into the demands of using external web services, developing user friendly mobile apps and creating a social media and content creation platform.

We made plenty of mistakes along the way and there are issues that we wish were dealt with sooner throughout development. However we truly feel that this has been a fantastic experience with a number of learning outcomes to be digested and is a good reflection of the knowledge we have gained throughout our years completing this undergraduate degree.