

# Adaptive Internet Radio

ICS2000 – Group Assigned Practical Task



UNIVERSITY OF MALTA  
L-Università ta' Malta

JENNY ATTARD	16696G
KRISTINA CATANIA	85196M
EDWARD FLERI SOLER	247696M
KENNETH ZERAFA	27596G

## Contents

1. Introduction .....	2
2. Aims and Objectives.....	2
3. Procedure .....	3
3.1 Online Bidding.....	3
3.2 Scheduled Jobs.....	5
3.3 Java .....	6
3.4 Music Preferences .....	6
3.5 Playlist Generation.....	7
4. Problems Encountered .....	7
4.1 Extraction from Facebook.....	7
4.2 Mailer Script.....	8
4.3 Downloading of Audio Files from Server .....	8
5. Future Improvements .....	8
6. Conclusion.....	9
7. References .....	10
8. Appendix.....	11
1 <sup>st</sup> Meeting 29 <sup>th</sup> February 2016, 4 hours: Idea.....	11
2 <sup>nd</sup> Meeting 3 <sup>rd</sup> March 2016, 5 hours: Basic Layout.....	11
3 <sup>rd</sup> Meeting 7 <sup>th</sup> March 2016, 3 hours: Facebook Page & Research .....	11
4 <sup>th</sup> Meeting 10 <sup>th</sup> March 2016, 5 hours: Facebook Page likes, Login & Style .....	11
5 <sup>th</sup> Meeting 14 <sup>th</sup> March 2016, 8 hours: JavaScript, PHP, Links & Bidding .....	11
6 <sup>th</sup> Meeting 21 <sup>st</sup> March 2016, 8 hours: Continued Bidding & Facebook data .....	11
7 <sup>th</sup> Meeting 4 <sup>th</sup> April 2016, 5 hours: Music likes & download files .....	11
8 <sup>th</sup> Meeting 11 <sup>th</sup> April 2016, 2 hours: Bid Acceptance & Emails.....	12
9 <sup>th</sup> Meeting 19 <sup>th</sup> April 2016, 4 hours: PHP and Online .....	12
10 <sup>th</sup> Meeting 25 <sup>th</sup> April 2016, 6 hours: Music Player and Design .....	12
11 <sup>th</sup> -15 <sup>th</sup> Meetings 9 <sup>th</sup> – 15 <sup>th</sup> May 2016, 30 hours: Finishing touches, testing and documentation .....	12

## 1. Introduction

Since the turn of the 21<sup>st</sup> century, the accessibility of technological devices has sky rocketed, with ownership of a computing or mobile device becoming the norm. Social media has taken the world by storm. Nowadays people may post, tweet, pin, upload and stream everything from the meal they had for lunch to their personal opinions and daily routines. At first glance, one may not find any proper use for social media past entertainment. However, one's social page may say a lot about their character and personal habits and, as we shall demonstrate through this project, this data may be used in a non-intrusive manner to improve services and individually engage customers.

This project shall focus on the implementation of an Adaptive Internet Radio system, which we shall refer to as AIR. This system applies the above mentioned tactic to better sales and engage customers in outlets, ranging from food stores and supermarkets to shopping malls and clothing stores, through the generation of a tailor made playlist. AIR also introduces an automated advert bidding system, which allows external businesses to bid for airspace. This system therefore not only better the service provided to customers, but is an alternative source of income.

We shall start by stating the aims and planned outcome of this system, followed by the general outline of meetings held between the involved team members. We shall then delve into the inner workings of the system, understanding the approaches taken to tackle a specific problem. Testing of the system and the outcome of these tests shall then be outlined, followed by a discussion of future implementations and the conclusion of this project.

## 2. Aims and Objectives

The aim of this project is to build a fully functional and interactive system for the generation of tailor made playlists. AIR dynamically extracts user preferences for music from social networks, building a unique, personalized playlist for each outlet. Data regarding music preferences is extracted from the clients that 'like' the respective outlets Facebook Page and this data is then analyzed, computed and used to construct a tailor made playlist. Further functionality involves the addition of an online bidding system, which allows prospective businesses/companies to bid for an advertisement slot in the air time. Our system then incorporates the winning bids into the playlist at the discretion of the administrator.

A key principle at the forefront of our design is the non-intrusive manner in which the system functions. All data extracted by the system is public, and in no way or manner tied to a specific individual upon storage.

This system may bring about a variety of benefits, with the possibility of increased sales, personal client engagement and a means of income for the said outlet through advertisement charges. Future implementations may see the consideration of excess data for analysis to improve the systems capabilities of building a truly dynamic system.

### 3. Procedure

In this section we shall introduce and explain the different parts of the system, their roles and the ways in which they are implemented.

#### 3.1 Online Bidding

The web-end of this system [1] is split into three main parts. The first part is the bidding page. This page consists of a form, asking the prospective bidder to submit their credentials, such as name, surname, email and company name. The user is also asked to upload a .mp3 audio file which shall be the advert to be played on air. Finally, a bidding price ranging between €1.00 and €10.00 is specified. Once all credentials are correctly input, the form is submitted and the user is notified that their bid has been submitted.

The structure and design of the page is implemented in HTML and CSS respectively, with basic <form> tags being made use of to outline the form. The required input fields are listed, and are noted to take in a specified type of input (such as text, email, range, etc.). A basic JavaScript function was set up to specify a window in which bidding may take place. Any attempts to place a bid between 8pm and 8am will be prevented, so as to ensure that the remaining bids and playlist can be processed successfully and

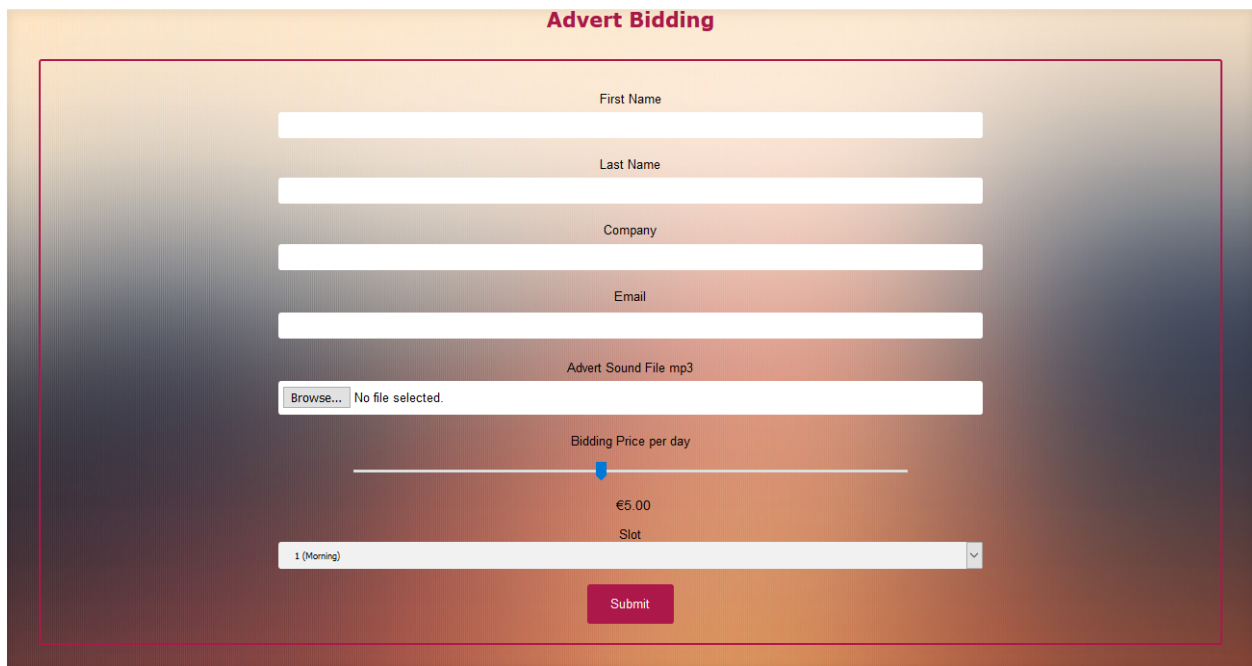


Figure 1. Print screen of bidding page form

in sync.

On submission of the bid form, a POST request is sent to a PHP script which handles the form data. The uploaded file is tested for validity in this script. Namely, the file is tested to be of the right type (.mp3), size and uniqueness, which we shall discuss below. If any inconsistencies are found at this point, an error is returned to the user. Otherwise, the script goes on to upload the audio file to a server directory categorized by the advertisement slot selected, a connection to the server database is established and the bid data is input into the said database. The uniqueness criteria mentioned specifies that no two

audio files in the advertisement slot shall be the same, in order to prevent excessive repetition of adverts.

Once all of the above operations are successfully completed, an email is sent out to the bidder, highlighting the submission of their bid, by means of the SendGrid plugin available through Microsoft Azure. This marks the end of the submission process.

The second section of the web-end of this project is the login page. This page serves as an access point to the bid reviewer page. Authorized members (administrators) may enter their username and password credentials to view the submitted bids. On entry of correct credentials, a session variable is initialized marking their authorization, and the user is redirected to the bid reviewer page.

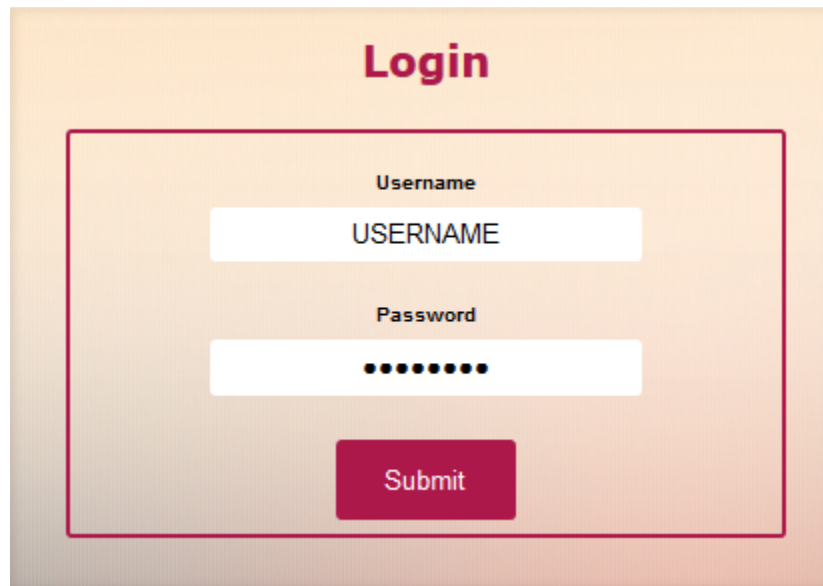


Figure 2. Login Page

In this page (figure 3), the top 5 bids from each slot are output on screen in a list, allowing the administrator to view the bid details and play the audio file advert. This feature was included to ensure that all included adverts are assessed by an administrator, and do not directly enter the playlist without screening of any sort. The administrator has an option to accept the bid or decline it. If the former is selected, then the acceptance PHP script is run by means of Ajax and the bid is hidden from the list. If the latter is selected, the bid is replaced by the next bid in the series (if present) and the decline PHP script is called, once again, by Ajax. Ajax was found to be ideal in this scenario as it allowed for the script to be run without the need to refresh the page.

The acceptance PHP script is passed the id of the accepted bid as reference, so as to perform operations upon it. Firstly, the said bid is referenced in the database and is altered to show that it has been accepted. Next, the advert file for that bid is moved to a new folder, holding accepted bids. This folder will be downloaded and inserted into the playlist at a later stage. Finally, an email is sent out to the respective bidder, informing them that their bid has been accepted.

In the case of the decline script, a similar procedure is followed, however the audio file is not moved to any folder, but is left in the same place. This folder therefore holds all declined or pending files, and will be cleaned out at a later stage.

These three pages, together with their styling, make up the web-accessible part of this project. However, through the Azure job scheduler, further scripts are run at set times to clean out and organize file data.

**Slot Number: 1**

Name: Norman	Surname: Dimech
Company: Conimex	Email: info@conimex.com
Bidding Price: 7.81 euro	<input type="button" value="accept"/> <input type="button" value="decline"/>
<div><div>0:00</div><div></div><div>4:22</div><div></div></div>	

Name: Joseph	Surname: Borg
Company: Your Stationers Ltd.	Email: joseph.borg@yourstationers.com
Bidding Price: 7.50 euro	<input type="button" value="accept"/> <input type="button" value="decline"/>
<div><div>0:00</div><div></div><div>4:22</div><div></div></div>	

Figure 3. Bid review page

### 3.2 Scheduled Jobs

As was previously mentioned, scheduled jobs are used to clean out and organize site data. Two PHP scripts have been assigned to the scheduler to run on a daily basis at set times. The first script is set to run at 3:30 am every day. This script is of great importance to the remainder of the system, and aids in the cross over from the Web to Java. This script begins by accessing the credentials of all entries in the database which have been listed as accepted. These credentials are used to create a text file called **bids.txt** which is stored on the server. This text file stores the server directory of all of the accepted bids for the respective slots and is updated every day early in the morning.

The script then goes on to truncate the database of all data, creating a new empty table for the next day to come. Alternatively, at the request of a client, the data could be stored and marked by date, so as to list its relevance.

Next, the three upload folders holding the declined or pending adverts are emptied. If an advert was declined, then it is no longer needed to be stored, and if it is still pending, then it has not made the top 5 bid space, or the administrator has simply decided not to accept all adverts. At this point, an email is sent out to all bidders whose advert was still pending, stating that it has not been selected. Following this task, the job has completed and will go into a resting state until it is called again.

The second scheduled job is set to run at 3:30pm every day. The sole function of this job is to clear the folder holding previously accepted adverts. This ensures that the folder is empty and that no files from the previous day remain in the folder. This prepares the folder for the new adverts to be accepted.

### 3.3 Java

Java was used in order to be able to gather all the information needed from the associated store's Facebook page. Here, ratios are used in order to generate the playlist. Five different classes were used, two of which are JLayer package imports. These two classes were used to generate the GUI that is displayed when playing the song from the .jar file. If the user would like to make an announcement, there is also the functionality of pausing and resuming the playlist with a button found here.

### 3.4 Music Preferences

The Java program begins by gathering the Facebook ID of each user who has liked the specified outlet's page. Once these are found another parse is implemented to obtain the music likes per individual. Figure 5 below shows the Java program traversing the list of users that have liked the outlets page, and visiting them to extract band data. This results in a list of bands which the user likes. Next, the Facebook page for each band is found. The band's Facebook page is parsed to find out what genre the said band falls under. Figure 4 below shows the Java program accessing each band's Facebook page to extract their genre type.

These are then generalized appropriately according to the 5 different genres we had preset; Rock, Pop, Classical, Indie and House. After performing this process for each user, a text file is generated that stores the ratios calculated from the popularity of the genres found. This will then be used shortly to generate the random playlist.

Due to the fact that several requests are made when extracting information from Facebook, such as the page likes as well as the music likes, a wait function was added into the code which delayed each request. This temporarily bypasses the problem caused by the blocking of our IP address by Facebook.

```
https://www.facebook.com/DemiLovato
https://www.facebook.com/WeAreMako
https://www.facebook.com/Techno
Error in link: https://www.facebook.com/Techno
https://www.facebook.com/ladygaga
https://www.facebook.com/blackeyedpeas
https://www.facebook.com/stonewallmovie
https://www.facebook.com/FedericaMusic
https://www.facebook.com/Rock-Music
https://www.facebook.com/PVD
```

Figure 4. Java output of band page parsing

```
run:
RUNNING INTERNAL SCRIPTS
-----
https://www.facebook.com/100001583051550
Error in link: https://www.facebook.com/100001583051550
https://www.facebook.com/100000081187793
https://www.facebook.com/100003673193591
Error in link: https://www.facebook.com/100003673193591
https://www.facebook.com/589214133
Search failed
https://www.facebook.com/10000196156786
https://www.facebook.com/1402731046
https://www.facebook.com/100002019058810
Error in link: https://www.facebook.com/100002019058810
https://www.facebook.com/1150374129
https://www.facebook.com/10000158902920
Error in link: https://www.facebook.com/10000158902920
https://www.facebook.com/1106113209
Error in link: https://www.facebook.com/1106113209
https://www.facebook.com/841910674
https://www.facebook.com/1431393695
Search failed
https://www.facebook.com/100001743924535
Error in link: https://www.facebook.com/100001743924535
https://www.facebook.com/1524126860
Error in link: https://www.facebook.com/1524126860
https://www.facebook.com/10000021612938
https://www.facebook.com/10000011600252
https://www.facebook.com/773366616
https://www.facebook.com/1694392519
https://www.facebook.com/1454940581
Error in link: https://www.facebook.com/1454940581
https://www.facebook.com/100001051080326
Error in link: https://www.facebook.com/100001051080326
Traversal Succeeded
```

Figure 5. Java output of user page parsing



The Java algorithm required a certain level of flexibility when dealing with data extracted from Facebook. As may be observed in figure 5 above, when visiting the pages of each user that has liked the outlet's page, certain inconsistencies exist. In the mentioned figure, 'Search Failed' results when the indexed user's profile is not publicly accessible, and therefore absolutely no data may be obtained from such a user. Lines starting with 'Error in Link:' relates to users whose profiles are public but have no public music preferences. Therefore, no data may be extracted from such users. Meanwhile, all other lines relate to users which have set their music tastes to show publicly. These users' pages are then parsed for band names.

Similarly, figure 4 above shows the Java program searching each band's page to find the band genre. An output starting with 'Error in Link:' relates to pages where no genre is listed. These bands must therefore be discarded as no useful information may be extracted from them.

### 3.5 Playlist Generation

In order to generate a random playlist, we gathered a list of .mp3 format songs and built a repository. Before the process begins a test is made to see whether or not an update on the genre ratios is required. At the moment, updates occur every month where the above mentioned Facebook scraping classes are run. This could be changed to suit the preferences of such companies making this more personalized. Next the adverts currently stored on the machine are deleted in order to be able to download the new ones of that current day.

A text file storing the names of the adverts accepted by administration (of the previous day) is downloaded from the Azure server and the corresponding adverts are downloaded too and saved on the current machine. Next, a text file containing the ratios is read and the songs are chosen at random from the repository built in the beginning, with each song having a probability of being chosen equal to their ratio. Whilst generating this playlist a check was used to ensure that no song was repeated. A variable is then used to keep count of the approximate length of each song so that adverts from each slot can be included within their respective time slot.

Another counter tracks the number of songs played for each advert. This sets the song-advert ration, which is initialized to 3. However, this can also be changed according to the company's preferences. This is done for all three slots used. The final playlist is then stored in a linked list, which is then passed to a play function, playing each song after the next until the playlist is done. The playlist may then be controlled by means of the pause/resume button available through the GUI.

## 4. Problems Encountered

Throughout the generation of the system, a number of hurdles were encountered, requiring research, modifications and a great deal of testing. In this section we shall list a few of the greater problems encountered, and the measures taken to overcome them.

### 4.1 Extraction from Facebook

One of the first major problems encountered related to the extraction of data from Facebook. The extraction process involves a number of calls to the Facebook server in quick succession. Firstly, the source code for the outlet page is requested, followed by a request from each individual who has liked the page, followed by a request for each band liked by each individual. This results in an exponentially



large number of requests within a small amount of time, which any human is incapable of manually doing. Therefore, following the first handful of tests conducted, the IP on which we were testing was blocked from sending requests to Facebook. This posed a problem as no further testing of the system was possible.

To overcome this problem, the outlet source code page was manually downloaded, and was saved in a text file, which was referenced as if it was normal source code returned from a request. Furthermore, a random delay function was introduced to randomly delay each call to a user profile or band profile. This randomness helps prevent the requests from following a trend which is easily caught out by Facebook servers. Reducing the running frequency of this function to once a month also helps prevent this from happening again.

Another noted problem was the change in the layout of Facebook pages whilst the program was being developed. The manner in which Facebook pages listed the individuals which liked a page was changed, deeming our initial approach incapable. Therefore, it is required that the text file be used to prove the concept, and changes would have to be made to update that section of the program to run at the moment. This is a common problem given the fast rate of change of sites such as Facebook.

#### 4.2 Mailer Script

Another encountered problem was the implementation of a function PHP mail sending script. Initially, PHPMailer was used to send out emails to clients. However, once the web application was deployed onto Azure, PHPMailer was no longer supported, resulting in a search for a new mailing function. Following research, the SendGrid mailer was found to be the favorite. This mailer was problematic due to the fact that it only runs following a POST request. Following a great deal of research and testing, this issue was sorted and found to work efficiently.

#### 4.3 Downloading of Audio Files from Server

The team found it challenging to download audio files from the Azure server through the Java application, so that the adverts could be included within the playlist. The first approach involves using Azure Blobs, which were the most frequently suggested approach when research was conducted. However, following dozens of attempts to get this system to work, this approach was abandoned in place of a new idea. It was found that by making the server directory for the audio public, and accessing the files through a URL specifying the directory, it is possible to download these audio files.

The next hurdle was passing the addresses of these audio files to Java. This was solved by generating the **bids.txt** text file which specifies the directory of each file. Once the Java program is initiated, this file is the first thing that it looks for. The directories of the songs are then found, the songs are downloaded and included in their respective folders. This approach was therefore found to work well and was ultimately implemented within the system.

### 5. Future Improvements

Although the system that we have created, tested and implemented is fully functional and performs its defined task properly, a number of possible future improvements have been discussed and noted. These could help make the system more efficient and effective in meeting its demands.

Gathering further information about clientele could strengthen the dynamicity and effect of the system. Gathering cues such as the preferred time of shopping; through eating or work habits, could help play songs preferred by individuals at the times at which they frequent the outlet. Analysis of sales data such as the type and time of objects being sold could also be incorporated, as studies have shown that different music influences different spending patterns [2]. Sales data may also be cross referenced with advertisement, providing feedback to advertisers regarding the reach and effectiveness of their adverts, attracting new businesses of the right fit.

The complete automation of the bidding process could also be possible following the application of the right techniques. Audio files uploaded by bidders could be analyzed for sentiment and content. Files containing explicit content could be filtered out and removed, while detailed analysis may be able to tell whether the uploaded ad is genuine or spam, or possibly even whether it is of competing outlet. The system could scan the audio file for keywords relating to competing brands, ensuring that no advertisements cause detrimental harm to the outlet they are being played in. This would make the system completely autonomous, removing the need for an administrator to manually scan through the ads.

## 6. Conclusion

Following the rigorous testing, analysis, modification and documentation of this system, it may be concluded that all the specified needs have been met. AIR manages to dynamically create a playlist, void of boring repetitions with a mix of songs relating to genres preferred by the specified clientele. Figure 6 below demonstrates how the system generates the playlist, starts playing the song and is controllable by the pause/resume button. The first lines are the song and advert titles in the playlist order. As may be observed, the different genres are incorporated into the playlist according to their popularity ratio, with adverts being output after every 3 songs. The final line in the screen shot is the title of the song currently playing (just 1 since the playlist has just started). To the right one may also observe the GUI pause/resume button which may be used to control the playlist.

We have seen this system working when only based upon a small control group of 20-30 individuals, therefore it is expected that with a greater data set for music preferences, the system would be better able to truly engage the customers. We have observed how the local program works in tandem with the web application, and how the two parts efficiently interact with each other. The system is designed to be easily personalized and adapted for specific client needs, from long term data keeping, to the number of adverts selected, their price range and their frequency in the playlist. With a few minor adjustments and improvements, this project could be marketed as a fully functional system to outlets.

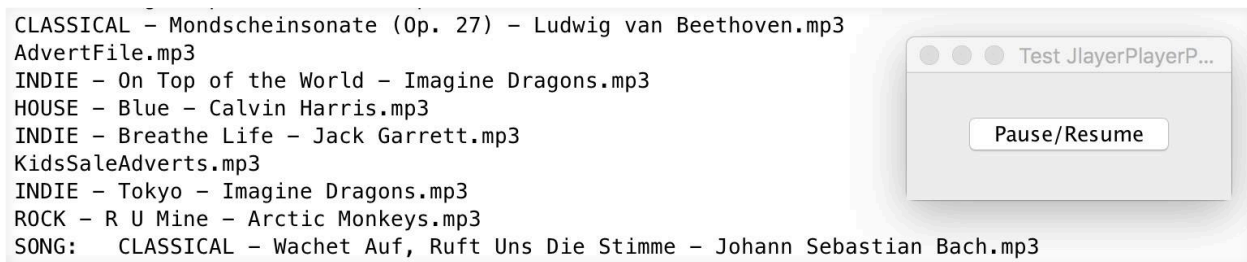


Figure 6. Java output of playlist

## 7. References

[1] <http://airgapt.azurewebsites.net/>

[2] <http://www.emeraldinsight.com/doi/full/10.1108/08876049610114249>

- Azure SendGrid Tutorial:  
<https://azure.microsoft.com/en-us/documentation/articles/sendgrid-dotnet-how-to-send-email/>
- JavaZoom/JLayer Library:  
<http://www.javazoom.net/index.shtml>

## 8. Appendix

### 1<sup>st</sup> Meeting 29<sup>th</sup> February 2016, 4 hours: Idea

Being the first meeting between all team members for such a project, the first task assigned to familiarize ourselves with each other's working habits and style. The general aims and objectives for this system were laid out, with sub-goals and deadlines being estimated for the duration of the project. Considering that the concept of this system was new and possibly abstract to us, a decent portion of time was dedicated to researching for ideas and means of implementations for this system.

### 2<sup>nd</sup> Meeting 3<sup>rd</sup> March 2016, 5 hours: Basic Layout

This meeting was dedicated to the planning and design of the web end of this project. A list of required features was made, and the general layout of the various sections of the web page were discussed and designed. Further research was conducted to investigate the programming requirements for the desired features.

### 3<sup>rd</sup> Meeting 7<sup>th</sup> March 2016, 3 hours: Facebook Page & Research

Considering that none of the involved members had ever dealt with the extraction of data from a social network, this meeting was dedicated to this task, being the primary sub-goal to be accomplished. A mock outlet page was set up, in the hope of gathering a small group of mock-clients to build and test the system upon.

### 4<sup>th</sup> Meeting 10<sup>th</sup> March 2016, 5 hours: Facebook Page likes, Login & Style

Further planning and basic development of the web-end of the system was started, with a login portal being set up to allow the administrator(s) of the proposed system to gain authorization to accept or decline prospective bids. Basic page styles and colour schemes were selected and research regarding the remainder of the system was also conducted at the same time. The above tasks were all split between the members of the team.

### 5<sup>th</sup> Meeting 14<sup>th</sup> March 2016, 8 hours: JavaScript, PHP, Links & Bidding

This meeting was solely dedicated to the development of the web-end of the system. The bidding form was set up and handled through the use of PHP and JavaScript, handling file uploads for the advert files. Following multiple attempts and research, the audio files allowed to be uploaded were restricted to the .mp3 file type. This decision was taken so as to ensure a level of consistency throughout the system. Considering the size of this task, some work was not completed in time, and a further meeting to complete this was set.

### 6<sup>th</sup> Meeting 21<sup>st</sup> March 2016, 8 hours: Continued Bidding & Facebook data

A continuation of the previous meeting, the bidding system was worked upon until completion. Work was delegated and research regarding the extraction of data through Facebook was conducted simultaneously. A great deal of research regarding the ways and means in which page likes and music preferences may be extracted from Facebook was conducted, and the best approaches were drawn out and discussed.

### 7<sup>th</sup> Meeting 4<sup>th</sup> April 2016, 5 hours: Music likes & download files

Concentration and effort was diverted to the Java end of this project, with a program being set up to download page data from Facebook and extract the preferences of each user. Music preferences were

split into five genres, with each genre being weighted according to its popularity. A sample repository of music was created with downloaded .mp3 music files. This repository was used to test and improve the algorithm.

#### 8<sup>th</sup> Meeting 11<sup>th</sup> April 2016, 2 hours: Bid Acceptance & Emails

A group decision on the manner in which bidding slots should be split up and bids being accepted was taken. This task was heavily discussed, as it would greatly change the approach to the remainder of the system. An emailing system to notify ad customers on the submission, acceptance or declination of their bid was also investigated.

#### 9<sup>th</sup> Meeting 19<sup>th</sup> April 2016, 4 hours: PHP and Online

A mailing system was set up to initiate the mailing of clients regarding their bids. It was also decided to deploy the web part of the system through hosting, so as to test that all work up to this point functions properly on the web. The Microsoft Azure Web Service platform was decided to be the best means of deployment, and time was dedicated to the setting up of a hosting account and the transfer of all files. The Job Scheduler capability of this web service was a key selling point, as it would allow jobs to be run remotely, which we felt would be of great use in the future.

#### 10<sup>th</sup> Meeting 25<sup>th</sup> April 2016, 6 hours: Music Player and Design

Focus was once again shifted towards to Java end of the assignment, while further testing was implemented upon the web-end. Remaining tasks involved the setting up of a music player which could play the queued songs in order. The idea of a GUI to play pause and stop the playlist was researched, so as to make the Java-end of the application more intuitive and inviting.

#### 11<sup>th</sup>-15<sup>th</sup> Meetings 9<sup>th</sup> – 15<sup>th</sup> May 2016, 30 hours: Finishing touches, testing and documentation

The separate parts of the project were brought together with multiple small errors and bugs being encountered on the way. The merging of the web-end with the Java-end demanded a great deal of time and testing to establish a synchronous workflow. Multiple minor improvements and fine tuning were made to the system, including the randomness and dynamicity of the playlist and inclusion of adverts. Multiple tests on all parts of the system were conducted, with adjustments being made each time a problem was encountered. The documentation was also drafted and worked upon, together with the showcasing project video. The final adjustments were made until the outcome of the project was satisfactory.