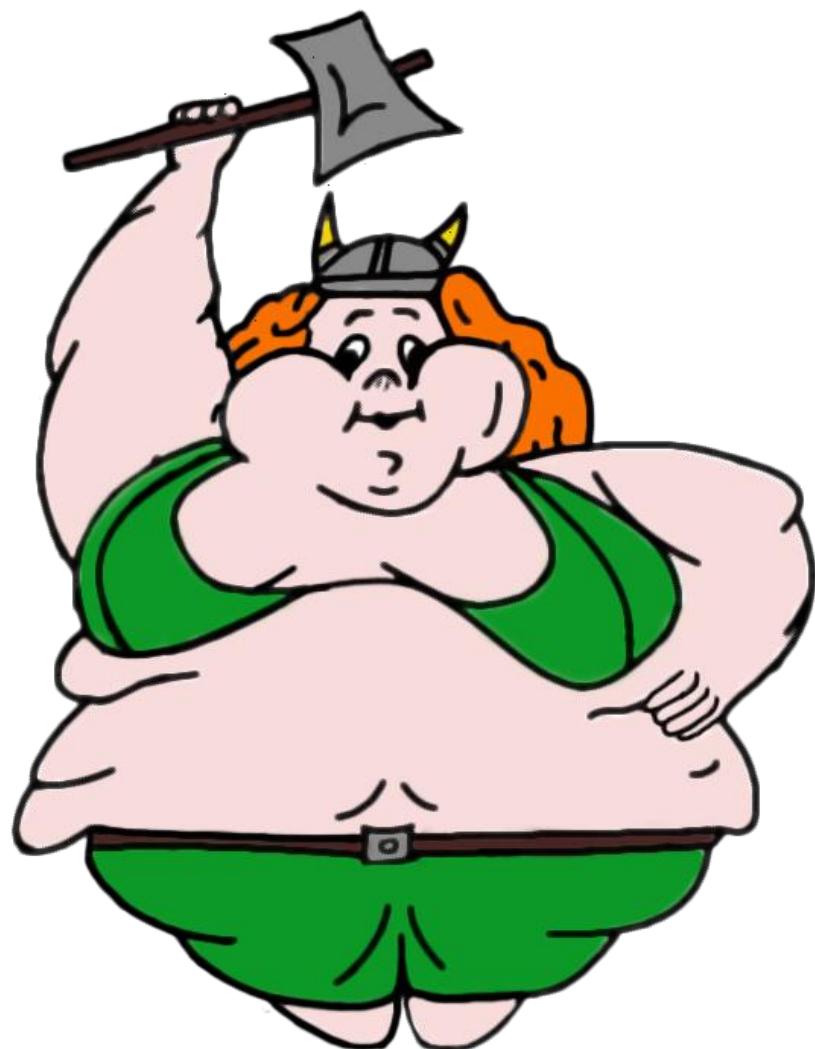


Big Julie

by John Jennings



For Stuart, my heart will go on.

“I love challenges. If you don't have any and can do whatever you want, then it's probably time to die”

- Jean-Claude Van Damme

Current System

Last year in the Sixth Form common room, we had a speaker system that we enjoyed playing music on during lunch and in our free periods. There were, however, a few problems with this. For music to play you had to use a wired connection to your mobile phone or iPod. I noticed that this would drain the battery and leave my phone unusable until I stopped using the speakers. Because you could only connect one device at a time, there were often disagreements if multiple people wanted to play their music and eventually the speakers were taken away as a result.

Project Proposal

In a preliminary meeting with Mrs Marston Smith, Head of Sixth Form, I was told that the students would benefit from being able to play their music but until the current problems were resolved, the option would not be available. I then asked a variety of Sixth Form pupils their opinion on the proposal and the vast majority agreed that an improved music system would be very useful to the year. I aim to develop a system with the staff and pupils that will overcome these issues and bring music back to the common room.

My initial idea is a multi-user music streaming service. The user will send song requests using their mobile phones to a website which will fetch their song from Spotify and add it to a queue. A networked PC will then take the song from the server and play it through some speakers.

End Users

The product will be used by pupils in the Sixth Form so I will choose a sample of students that can act as beta testers focussing on the user interface and suggest functions to be added. The product must also adhere to the rules set out by Mrs Marston-Smith.

Pupils

There are around 200 pupils in the Sixth Form. They have a variety of musical tastes and devices from iPods to Android phones. Many of them will find see it as a challenge to disrupt the normal operation of the system, such as playing very long songs or the same song repeatedly. Most pupils have a good grasp with technology and can easily navigate a website by themselves. They do not have access to the school's Wi-Fi and must use mobile data for internet access which can often be limited by their mobile contract.

Liam

Liam is the Deputy Head Boy and leader of the 'Improvements Committee' for the Sixth Form. He is very enthusiastic about the project and looks forward to testing the system. He will act as spokesman for the pupils, suggesting new features that the committee would like to add as well as any general problems with the system. Having studied Computer Science at AS level, Liam should be able to give a more technical opinion than the average end user. As such, his input will be very valuable when we design the main functions of the system since he is aware of what we can realistically achieve.

Mrs Marston-Smith

As Head of Sixth Form, Mrs Marston-Smith wants to maintain a good working environment for the pupils. She would not be happy if the music was very loud and would prefer it if music was only played during break-times since the Sixth Form Centre should be a quiet working environment during free periods. She cannot allow explicit music to be played in the common room.

Relevant Data

The school assigns each pupil with a username in the form of the year they entered the school, their first initial and their last name e.g. 08jjennings. This may be useful to my system if I want to create accounts or keep track of who is using the service as the pupils already know their username and it will be trivial export them from the network's Active Directory to import into my database.

The system will need to interface with a music service to fetch the relevant data related to the songs requested by the users. Spotify is a streaming service with a robust API that would allow me to query their database, and access information on the thousands of songs that the service can stream. The API also returns a unique URI, (HTTP address), which could be given to a Spotify client to play a specific song. The additional information accessible through the API such as cover art and track names will aid in giving feedback to the user such as informing them of what song is currently playing.

Here is an example of some information returned from Spotify on a single track:

```
{
  "tracks" : {
    "href" :
      "https://api.spotify.com/v1/search?query=hello+lionel&offset=0&limit=1&type=track",
    "items" : [ {
      "album" : {
        "href" :
          "https://api.spotify.com/v1/albums/5U0NU0T1JKIJwgq2ZDWb2T",
        "id" : "5U0NU0T1JKIJwgq2ZDWb2T",
        "images" : [ {
          "height" : 634,
          "url" : // link to album art
          "width" : 640
        }, // information on album
        "name" : "Can't Slow Down",
        "type" : "album",
        "uri" : "spotify:album:5U0NU0T1JKIJwgq2ZDWb2T"
      },
      "disc_number" : 1,
      "duration_ms" : 250200, // length of song in milliseconds
      "explicit" : false, // whether song is explicit
      "href" : "https://api.spotify.com/v1/tracks/6HMvJcdw6qLsyV1b5x29sa",
      "id" : "6HMvJcdw6qLsyV1b5x29sa", // unique URI of song
      "name" : "Hello", // name of song
      "track_number" : 8,
```

Initial Requirements

Here are the initial requirements for that I have ascertained from my initial meetings with clients and observation of the current system in use.

<u>Requirement</u>	<u>Possible Solution</u>
- Compatibility with multiple mobile devices	- AUX lead connection / web interface
- Wireless connection	- Web interface over mobile data
- Solve disagreements on music taste	- Change between different user's music often
- Songs must not be too long	- Filter songs by length
- Songs must not be explicit	- Filter songs by age rating
- Music must not be too loud	- Set maximum volume for system output

Current Products

I researched current products that performed similar functions to the project proposed, listing their strengths and weaknesses so that I could improve on them with my system.

Motorola Deck

The Motorola Deck is a Bluetooth speaker with some simple multi-user functionality. The users can pair their Bluetooth device to the speaker and send their music to it from their mobile's storage. Whenever a new user sends a new song, the device will immediately end the current song and begin playback.



GOOD

- The deck is compatible with any Bluetooth enabled device.
- Bluetooth is wireless so the connected devices could still be used while playing.
- It is quick and simple to connect a device to the speaker.

BAD

- The multi-user system is very basic and does not let songs finish. We would still have a problem with people disagreeing on song choice.
- The device costs around £130 which would be too expensive for the Sixth Form.
- There is a maximum of 5 users which would be unsuitable for a Sixth Form of ~100 students.

Apple Remote

Apple remote is an iOS app that allows the user to link apple devices through a Wi-Fi connection. This then allows the user to control the devices wirelessly from a different device e.g. controlling an iPod connected to a speaker from an iPhone.



GOOD

- The app has most of the functionality of iTunes, allowing for movies and pictures to be streamed also.
- The playlist feature of iTunes will allow for basic multi-user functions.
- The user interface is simple and well designed

BAD

- The user is limited to one account and must buy the music before playing.
- The app is only compatible with apple devices which are expensive and not everyone owns an apple device
- There is no purpose-built multi-user functionality

In order to develop a high quality product, I must first analyse the needs of my clients and end users and develop a list of requirements that must be fulfilled in order to create a satisfactory product.

Information Gathering Plan

To ensure that I efficiently gather useful information, I have created a general strategy of the meetings that I will attend and the surveys that I will carry out. This will help me to organise the project effectively and collect relevant data in a reasonable amount of time.

<u>Date</u>	<u>Plan</u>
29/09/14 → 6/10/14	Hand out questionnaire to sixth formers. This will find useful numerical information such as how many people have iPhones and how many people would want to use the system.
7/10/14	Interview with Liam discussing the general functions of the system that he would like with consideration of the data collected from the questionnaire.
13/10/14	Interview with Mrs Marston-Smith to ensure that the ideas agreed with Liam will be allowed in school.

Questionnaire

1. I own a...

- a. iPhone b. Android c. Blackberry d. Other (Please State)

2. My phone supports...

- a. Wi-Fi b. 3G c. 4G d. None

3. My phone can store GB.

4. I listen to music from...

- a. YouTube b. Music Streaming c. My Phone d. Other (Please State)

5. I prefer...

- a. Pop b. Rock c. Dance d. Other (Please State)

6. I like the music that my peers listen to...

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

7. I prefer listening to music with...

- a. My Headphones b. Shared Headphones c. Speakers d. Other (Please State)

8. I spend most of my study periods in...

- a. The Endeavour Room b. The Study Area c. The Canteen d. Other (Please State)

9. I spend most of my study periods...

- a. Doing Homework b. Playing Games c. Talking d. Other (Please State)

10. I would prefer my study periods to have...

- a. Silence b. Talking c. Music d. Anything

11. I would enjoy having music played in study periods.

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

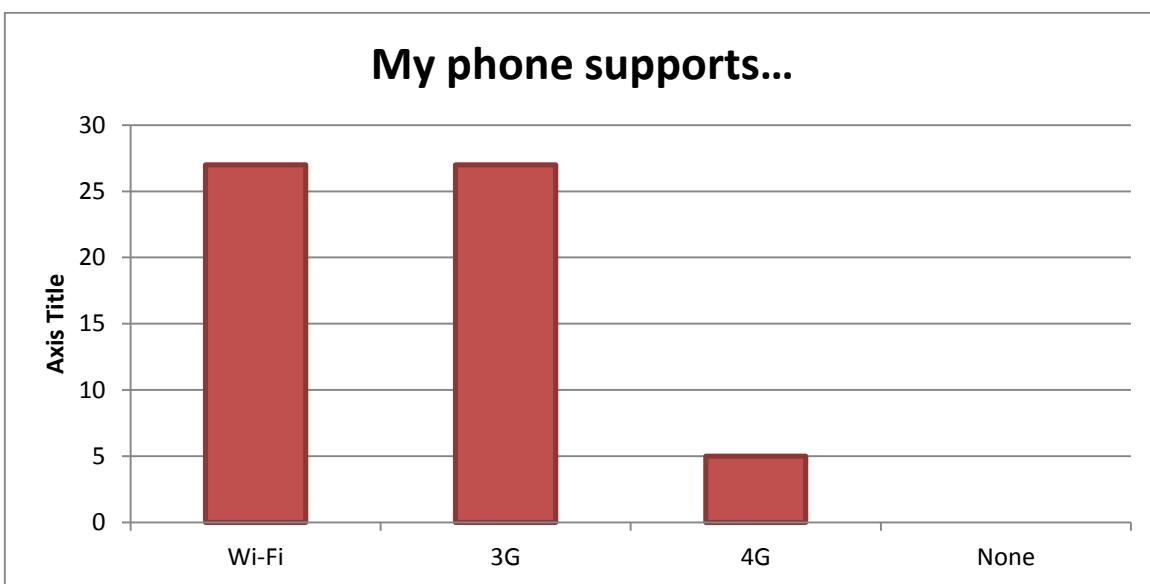
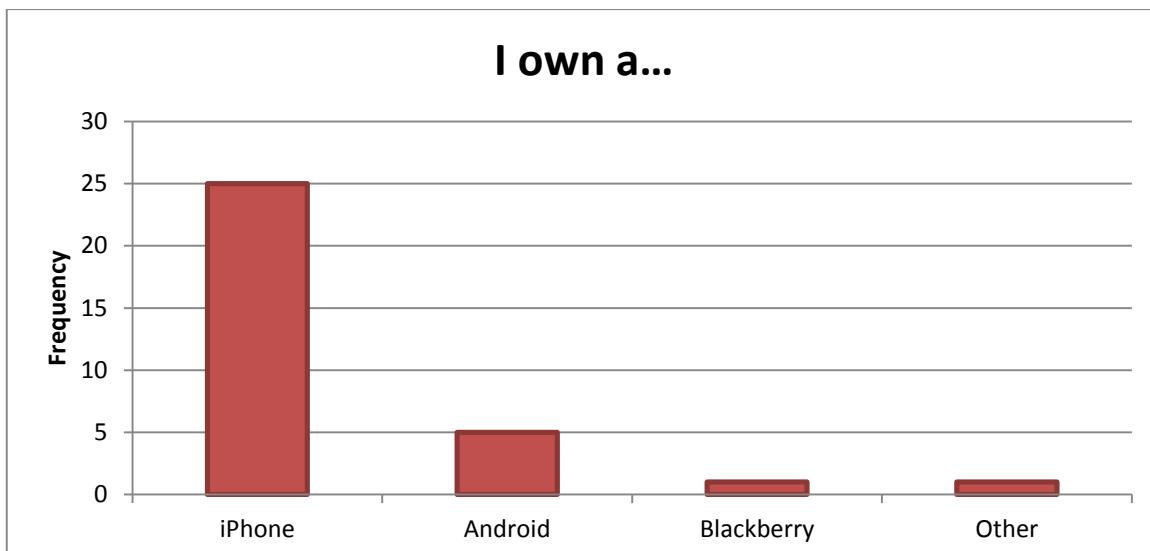
Questionnaire Justification

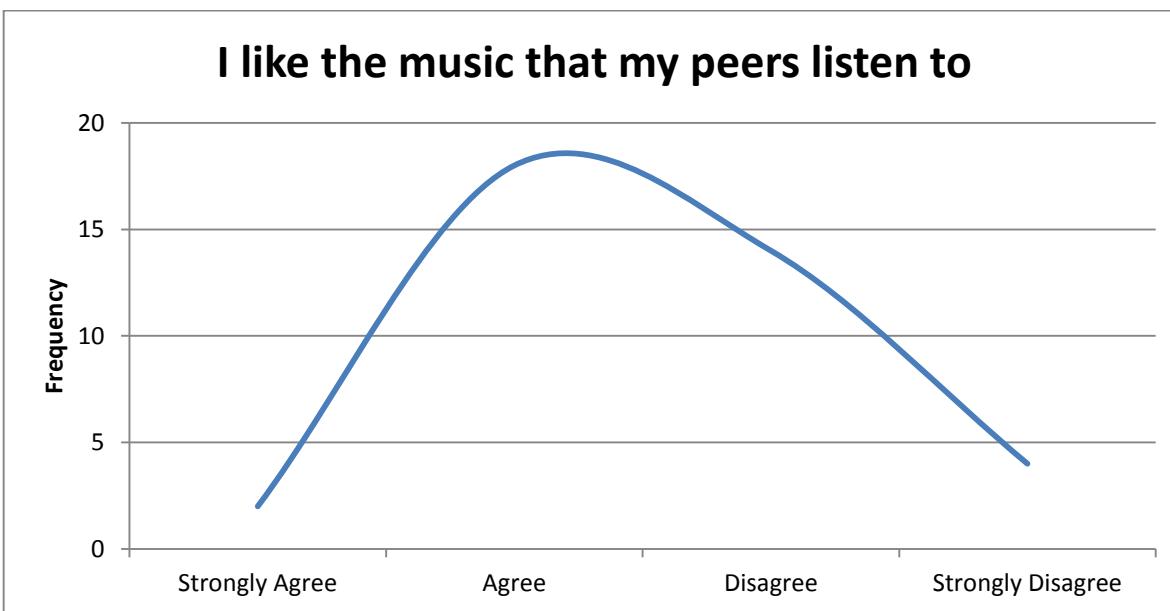
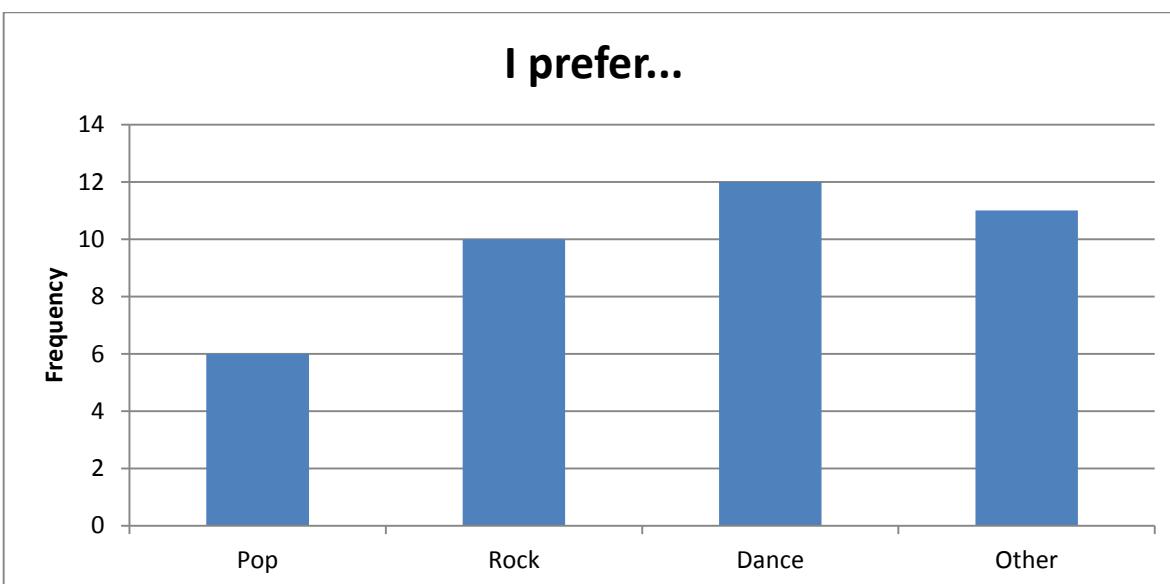
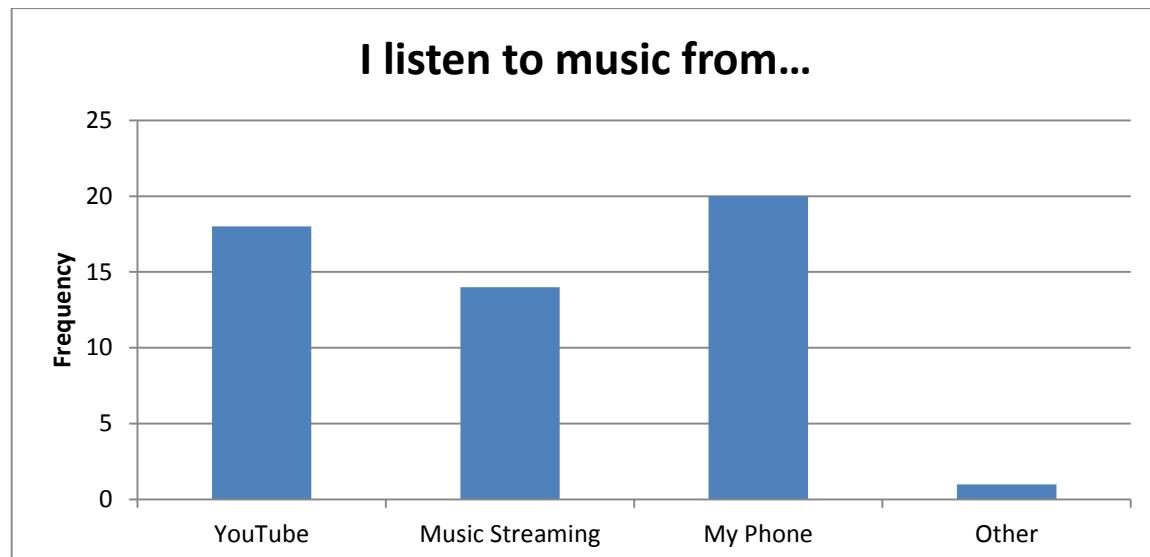
Questions 1-3 will help me ascertain the different types of device that my product should be compatible with. If the majority of users have iPhones then I could develop an application for iOS. Likewise, I will have to limit my design by the average speed of the user's internet and their average storage capacity.

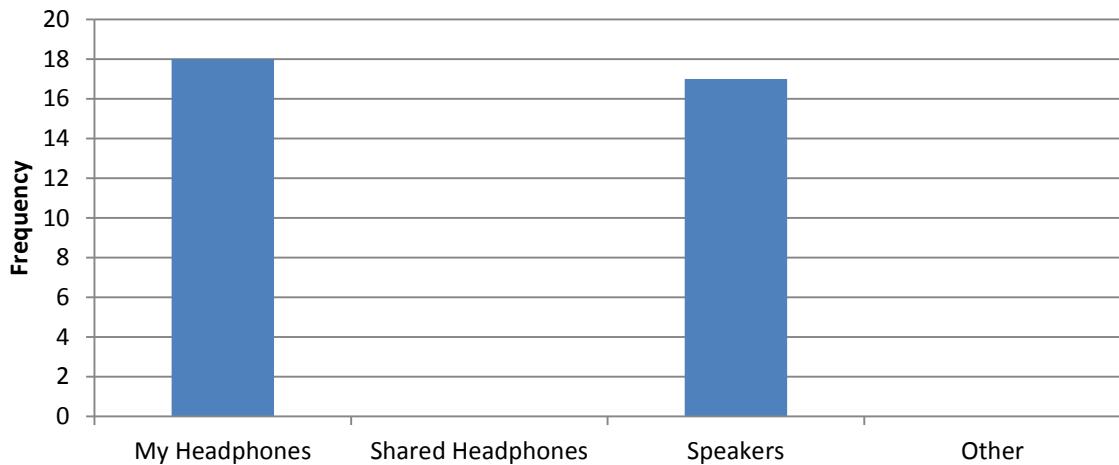
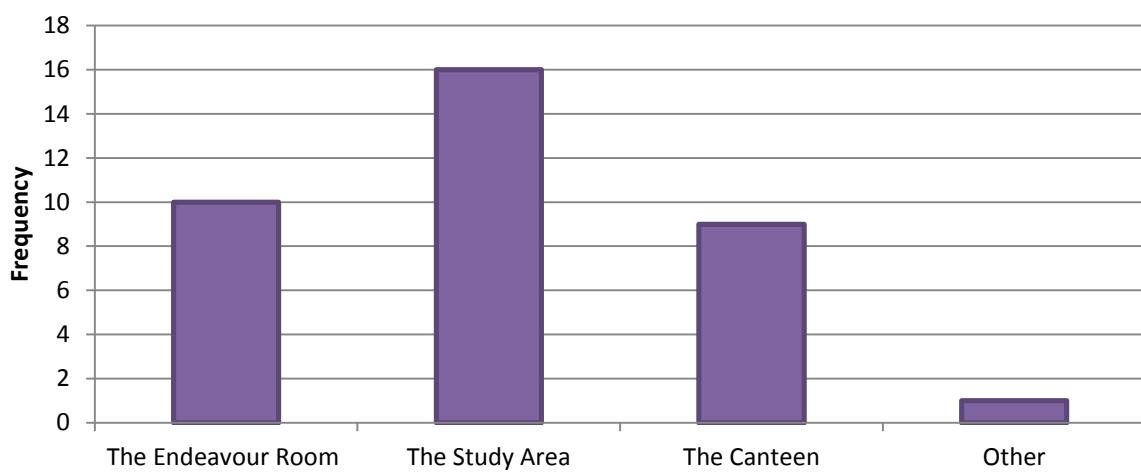
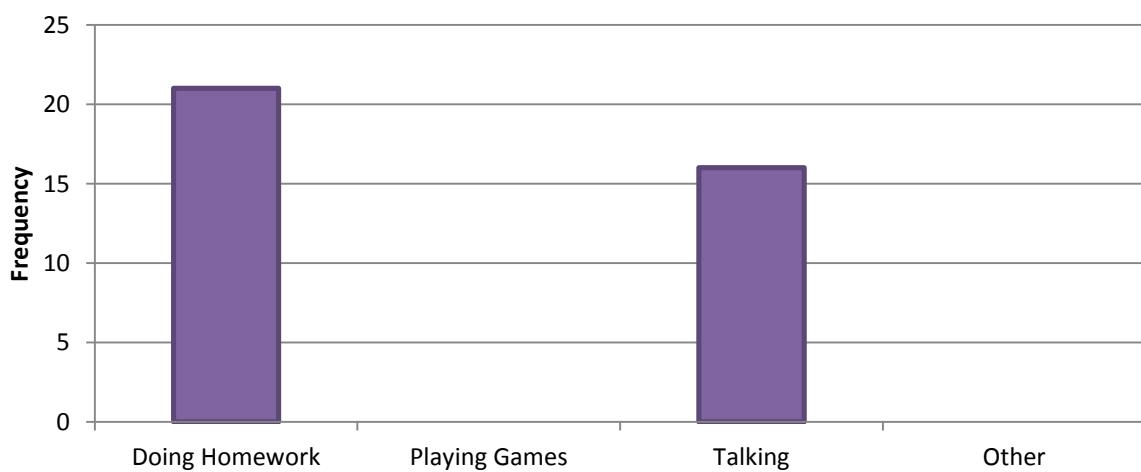
I used questions 4-7 to discern the specific methods that I will use to play the music. The source of the music will be decided by the most common way that users get their music from currently. If there is little agreement between the users' musical taste then I will have to implement a system that plays a variety of genres so that everyone gets their share of the music. If a sizeable number of users say that they don't enjoy music being played in their free periods then I should have the system use headphones instead of speakers. If the popular source of music is a streaming service such as YouTube then it would be advisable to use that service as the music source for my system as that is what the users are used to.

Questions 8-11 were used to determine whether the Sixth Form as a whole would appreciate the system. If a large amount of students use their free periods to do homework in silence then this system will not be useful to them. On the contrary, if the majority of people use their free periods to relax and think that playing music would be enjoyable then my system will integrate well with the user group's lifestyle.

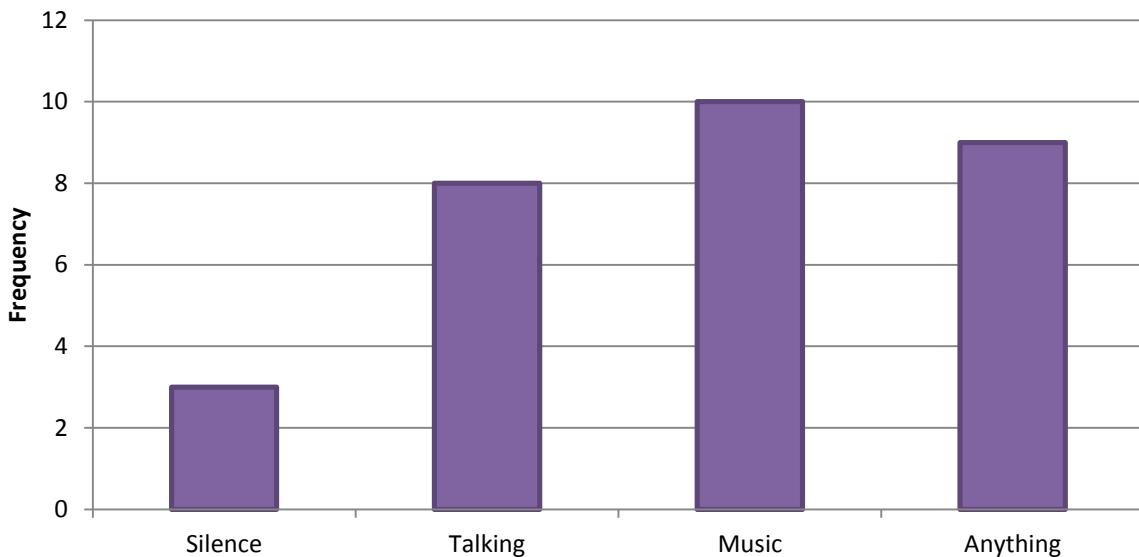
Questionnaire Results



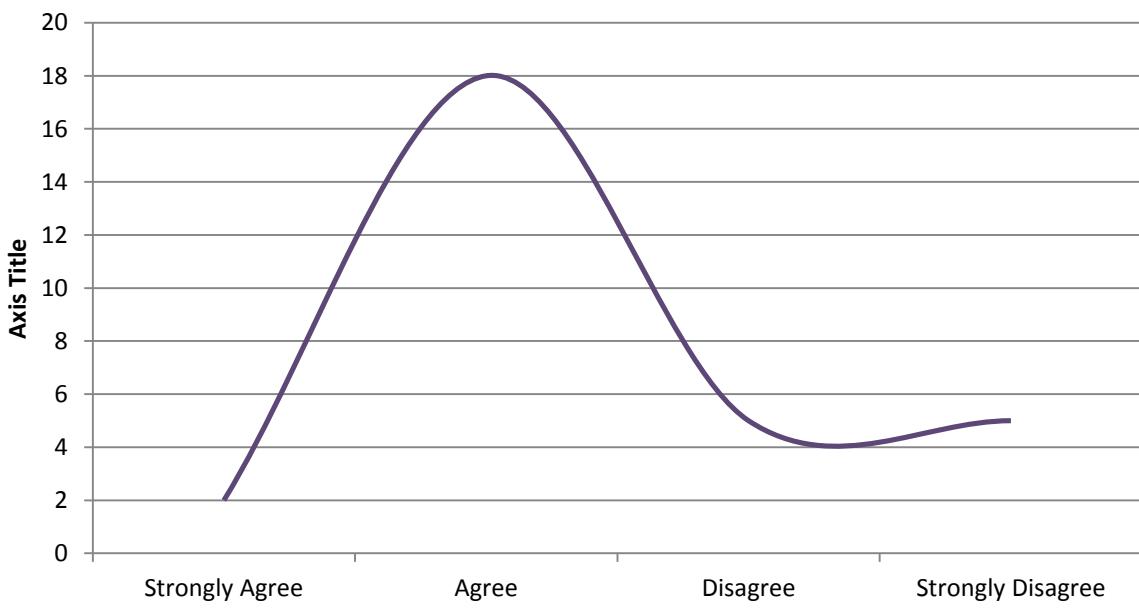


I prefer listening to music with...**I spend most of my study periods in...****I spend most of my study periods...**

I would prefer my study periods to have...



I would enjoy having music played in study periods



Questionnaire Results Analysis

The answers to the first three questions gave a clear picture of the hardware that users have available to them. The vast majority of users owned an iPhone which was 3G and Wi-Fi enabled, meaning that both an iOS app and mobile website would be a generally accessible method of interfacing with the system (**i1, i2**)*. The average storage capacity of a user's mobile device came to be around 16GB so I should not have to worry about reducing the file size of any data handled by the system.

Questions 4-7 gave a strong indication that there are varying music tastes in the Sixth Form. Pop, Rock and Dance music were fairly equally popular while many users opted to choose the 'other' option. On average, a single user does not mind the musical taste of their peers so there is little need to artificially cause a variety of genres to play. However, there were enough answers that showed a dislike of other people's music to warrant the need for a 'skip' request for unpopular choices (**i5**). The skip function would work by counting the number of skip requests and once a certain threshold was reached, end the currently playing song and move forward in the queue (**p5**). There was a 50/50 split between users who prefer listening with headphones and those that prefer a speaker system so I may have to develop a way to cater for both parties; perhaps an option to switch between headphone and speaker 'mode.' There was no clearly preferred method of accessing the music so I should be able to choose whichever method is the most practical to develop with. This will most likely be Spotify as it has a versatile API that can easily be integrated into the code for my system.

The remaining questions gave reassurance that this system would be received well by the Sixth Form as a whole. There is a 50/50 split between pupils that do homework in the Study Area and pupils that talk in the Canteen and Endeavour Room. Since the Study Area is a quiet working area, the system should not be implemented there and instead be used in the other rooms where people like to relax and talk. Overall, people responded positively to the idea of music being played in free periods and those that disliked the idea tended to work in the Study Area where they will not be affected anyway.

* Requirements with code i1 and i2 are sourced from this information

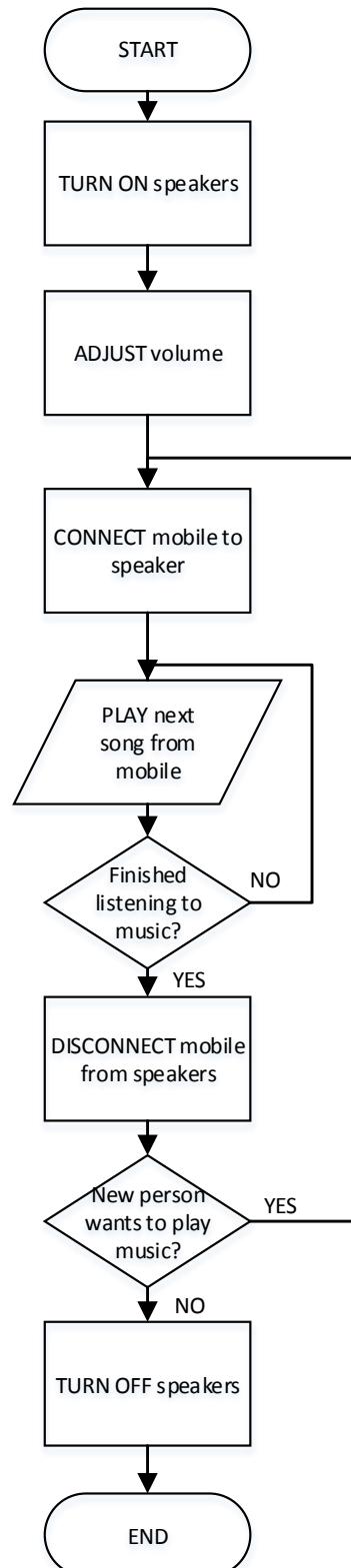
Observation

Last year I observed the use of the current system during my free periods. The system consisted of a set of hi-fi speakers and an AUX lead. The user would connect their mobile device with the AUX lead to the speakers and use the media player software on the device to play the music on their phone's storage. Some users would use streaming services such as Spotify and YouTube instead of playing from local storage.

I noticed that people would often disagree on the music they wanted to play. A single song would almost always finish before the end because another user would connect their phone to the system and play their choice of music. Often, the music would be interrupted by text alerts and other notifications on the user's phone. The user would then have to stand up from their work and walk over to the system where they would disconnect their phone. Many people did not have the opportunity to play their music and decided to use headphones instead. On a few occasions, there were arguments because multiple people wanted to play their music which eventually led to Mrs Marston-Smith taking the speakers away from us.

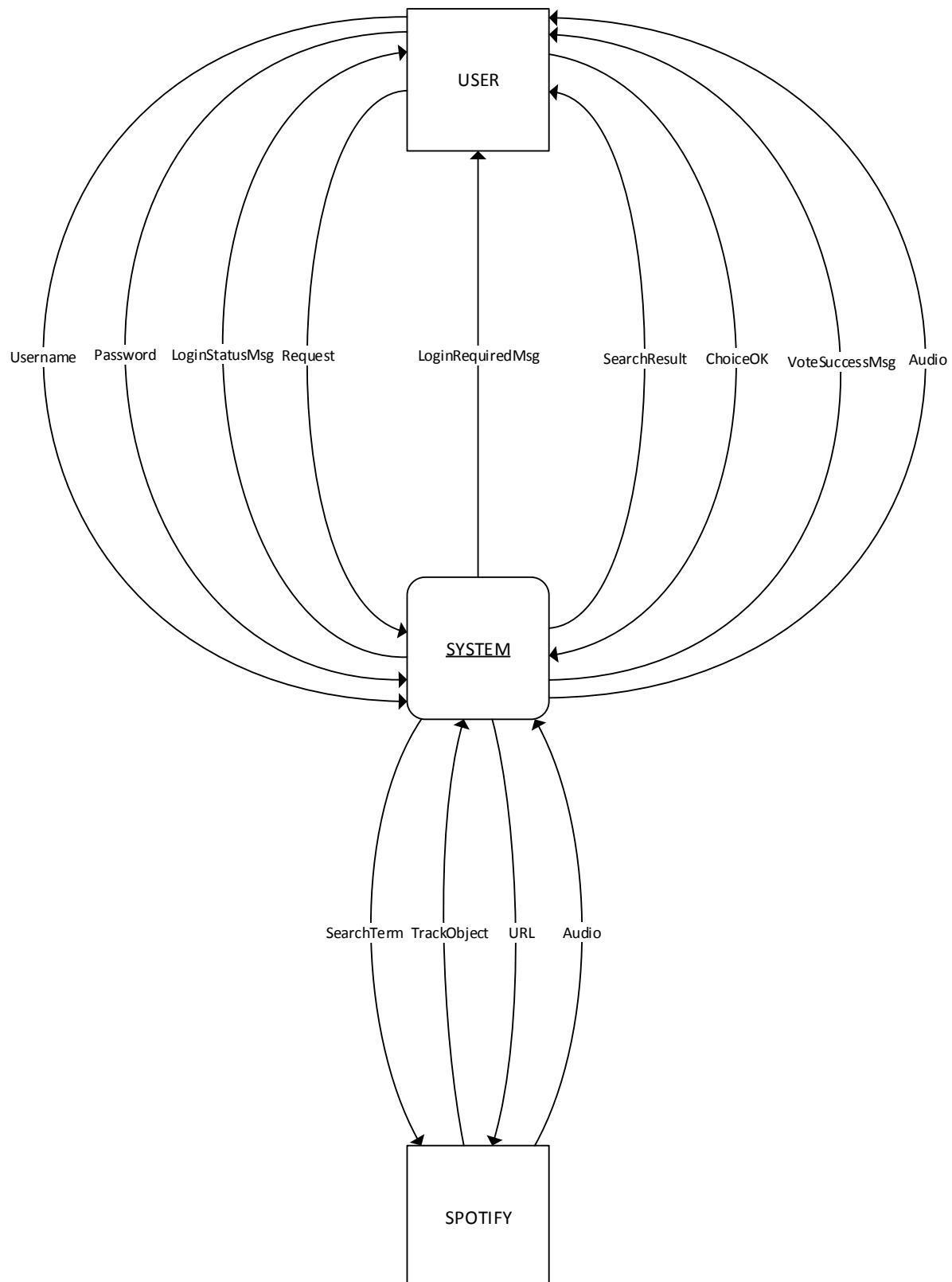
Current System Flowchart

Following my observation of the current method that is used to play music in the Sixth Form Centre, I created a flowchart to display the process in a visual manner. This will allow me to easily see the functions that I should be emulating and improving upon when designing the new system.



Context Diagram

From the flowchart I then created a context diagram to show the general interaction of data with the entities in the system.



Interviews

I asked Liam a set of seven questions that were designed to give a general overview of how he wanted the system to be designed. I put specific emphasis on the situations in which music should be played and the general functions of the system that Liam would like. I noted down any ideas that he suggested and will use this to develop the requirements specification.

Since I finished my interview with Liam earlier than I expected, I also interviewed Joe as he was a good example of the average end user of the system. He is a year 13 prefect and is very interested in the system as he studies A Level music.

Below are the questions that I asked:

- 1. When should we be allowed to play music and why?**
- 2. Who should be allowed to use the system and why?**
- 3. What should be the functions of the system and why?**
- 4. Where should the music be played and why?**
- 5. What problems could you foresee in the system?**
- 6. What are your views on the system in general?**
- 7. AOB**

Liam

1. When should we be allowed to play music and why?

Liam thinks that we should always be allowed to play music in the Sixth Form Centre because it is our free time and we can do what we want. In response to the idea that music may distract us from our work, he says that in many cases, music has been shown to help studying.

2. Who should be allowed to use the system and why?

Liam proposes the idea of a 'rota'. A fixed number of people should be allowed to request music in one period. This is in an effort to prevent people from overloading the system with requests. A specific user can book in advance for their preferred period of controlling the music so that they don't get assigned control when they are in a lesson. He thinks that there should be two separate systems for year 12 and 13 with students in each year only having access to 'their' system. He thinks that there should be unique accounts for each pupil and one for teachers that will have extra features. This will allow the staff to control the volume and skip or mute the system when they feel it is appropriate.

3. What should be the functions of the system and why?

Liam would like the system to be controlled from a single point using a mouse and keyboard. There would be a simple user interface displayed on a monitor where the user could input their song choice and call other functions. He would like for the interface to consist of a single page that updates when new information is available. When asked for an example of these functions, he proposed controls such as skipping the song, shuffling the song queue and voting for a song to move forward in the queue. When there are no more song requests to be fulfilled, he wants the system to pause and wait for the next request. His preferred method of sourcing the music is a streaming service such as Spotify since there is a large variety of music to be found there. Liam is aware that this method depends on having an internet connection so he would like a backup function that would be active when the system cannot access the streaming services. He proposes that the number of requests for each song is stored permanently and when a certain song is requested a set number of times, it is downloaded and stored on the system's memory. In the event of a connection loss, the system should revert to playing randomly from this collection of popularly requested songs.

4. Where should the music be played and why?

Liam would like there to be two separate systems in the Canteen and Endeavour Room. He suggests that the Study Area is left without music as that room tends to be a silent area where people need to focus. He knows that there is a stereo system in the Endeavour Room that we are currently not allowed to use but he recommends that we ask Mrs Marston-Smith for permission to connect the music system to it as the resources are already in place.

5. What problems could you foresee in the system?

Liam thinks that there may be a problem with people continually skipping songs that they don't like. He says that this will annoy the users as it is better to listen to one complete song than the first ten seconds of many. His proposed solution to this is a hard cap on the amount of times the system can skip a song as this will promote playing the music until the end but still allow users to skip what that they really do not like. He is worried about the signal strength of the Wi-Fi in the Sixth Form Centre as he often loses connection when he works on a laptop there. There is also the problem of people choosing explicit songs which he says can be mitigated by letting teachers have the power to skip any song they deem inappropriate and possibly disable the system for a while if it is not used sensibly.

6. What are your views on the system in general?

Liam thinks that playing music in the Sixth Form Centre is a "cracking idea" and looks forward to seeing its development. However, he does feel that it will be a challenge to implement and highlights the importance of controlling the system so that it cannot be abused or it will likely be taken away by the staff.

7. AOB

Liam had no other comments on the system.

Liam Feedback Analysis

Liam gave satisfying answers to every question and his suggestions were generally useful and either agreed with my current ideas or improved upon them. In particular, I liked his idea of having two separate systems for each year group. With this approach, only one set of speakers will be needed for each system, eliminating any synchronisation problems that would occur between multiple speakers if they were both playing the same music. When discussing the basic functions of the system, he supported my previous idea of a queueing structure (**p3**) with a 'skip' function and also proposed other useful utilities such as 'vote' requests (**i4, p4**) and the offline backup which was an elegant solution to a problem that I had not foreseen. His request for the interface to consist of a single dynamically updating page is feasible with JavaScript and would give a more natural user experience (**o4**). While the majority of his suggestions were very helpful and will most likely be used, I have chosen to discard some of his ideas. In question two he suggested a 'rota' system which while functional, it limits the number of users that can control the system at one time. This goes against the aim of having a system that everyone in the Sixth Form can control at once although it is admittedly more practical so I may end up using this idea if the system does not scale well to a larger user base. The potential problems that he identified were valid and I aim to proactively solve them although his suggested solutions often required a teacher to moderate the system. I will try to reduce the amount of interaction that the staff will need to have with the system as they are very busy. I will instead focus on automated filters for song length and explicit content that are built into the main functions of the system (**p7, p8**).

Joe**1. When should we be allowed to play music and why?**

Joe fully supports the idea of playing music during lunch and break as it is already quite noisy during that time and people do not need to concentrate as they are not doing work. He thinks people would appreciate being able to use the system during their free periods but since some people have expressed that they would prefer to work in silence, it would be good to leave one room without speakers for those who need to concentrate.

2. Who should be allowed to use the system and why?

Joe thinks that everyone should have some access to the system as it would be unfair to leave people out. He suggests giving each user an equal say in what music is played so that nobody can monopolise the system although he supports the idea of letting teachers temporarily give certain users more 'votes' as a reward for good work or behaviour. He doesn't like the idea of giving teachers an overriding control of the music as he believes we should be old enough to act responsibly. If the system is not being used correctly however, he would prefer for the teachers to restrict the functionality instead of banning it outright. Since the two years included in the Sixth Form tend to congregate in separate rooms, he suggests having two separate systems for each room. This would mean that the year 13s can only play music in their room and vice versa; thus increasing the chance of a certain user having their music played.

3. What should be the functions of the system and why?

Joe proposes that the basic function of the system should be a queue. This would work on a first come first served basis where a user sends in a song request which will be played when all the previous song requests have finished. He understands that in order to prevent 'spam', there should be a limit to the amount of times that a specific user can make a request. This limit should scale with the number of active users so that the less active users, the more power a single user has. To prevent music that people don't like from being played, he suggests a 'skip' function where the system will end the current song and move onto the next in the queue if a certain number of users send a request for the song to be skipped. He also says that there should be a limit to the number of times that a specific user can use this function, ensuring that people will not just skip every song until their request is played. He does not like the idea of filtering the songs for explicit content as we should be trusted to be responsible. However, he does think that the option should exist in case the users abuse the system with overly-offensive content. He says that there must be a maximum length for any suggested songs in order to allow more users' requests to be fulfilled. When the queue is empty (there are no more requests in the system) he would like for the music to continue playing from a list of previously requested songs.

4. Where should the music be played and why?

Joe suggests that the main system should exist in the canteen area as it tends to be the louder room where people are more likely to enjoy music. This is also the area where people gather at break and lunch. He thinks that the study area should remain quiet although he is aware that there is interest in the system from people that use this room. He suggests that there could be music there but it should be quieter and restricted to more relaxing genres such as classical and folk.

5. What problems could you foresee in the system?

Joe focussed specifically on the different ways that users could ‘break’ the system by using it in a way that was not intended. He makes it clear that some people would find it funny to play overly offensive or incredibly long songs to annoy other students. Joe also points out that users could physically unplug the system in order to use the speaker for their phones or disable the music entirely. He suggests that we try to make the system unreachable by the users i.e. placing the system in a locked box.

Furthermore, Joe acknowledges that a group of people could try to ‘break’ the system by using the software incorrectly. He advises that the system should limit the amount of requests that one user can make in a small period of time in order to prevent a makeshift Denial of Service attack. He also raises the issue of a certain group of students having a monopoly on the music due to the way that the majority vote is used. He proposes that the system gives less ‘weight’ to votes for a song that is very similar to the music that was played recently in order to promote variety.

6. What are your views on the system in general?

Joe thinks that the inclusion of music in our study periods will be beneficial to both the pupils and teachers. He looks forward to seeing the system develop and hopes it will be released before he leaves school.

7. AOB

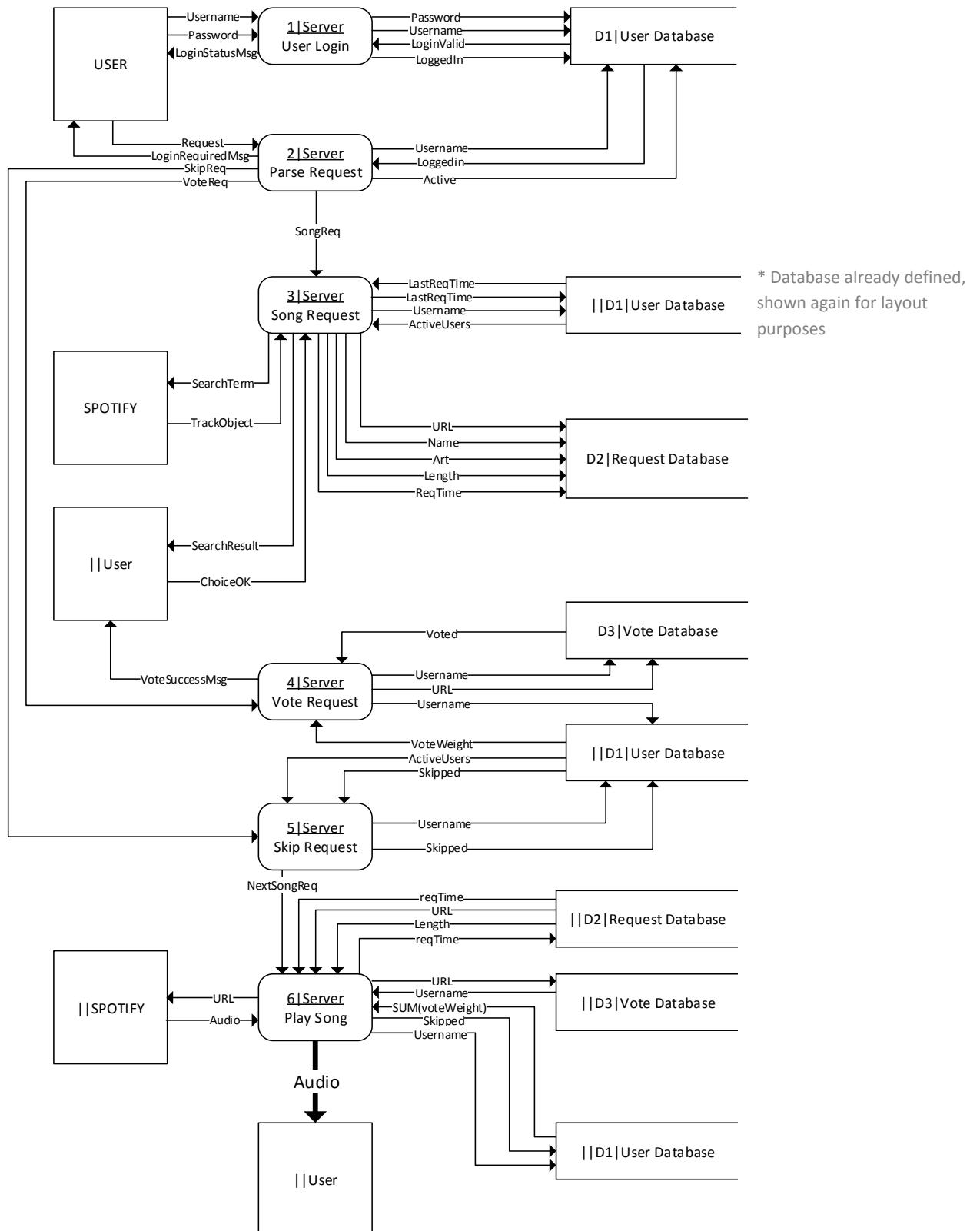
Joe requests an ‘Easter egg’ where the system will play the same song repeatedly on April the 1st.

Joe Feedback Analysis

Considering the fact that I had not originally planned to interview Joe, his input was particularly impressive. In general, he supported the current ideas for the project and added some unique and useful ideas to the plan. He refined the previously outlined 'queueing' format by including a function for 'weighting' the votes of certain users so that we can vary the control that certain users or groups of users will have over the system (**p6**). In order to achieve this, the users must log in to unique accounts before they can send requests (**i6, i7, p1, p2**). I especially liked his idea of allowing teachers to give certain pupils a higher 'weighting' as a reward for good work. His approach to potential problems with the system was also very useful. Another improvement that he suggested was for the system to continue playing previously requested songs when all requests had been fulfilled. This would give a better user experience than simply waiting for new requests as there would always be music playing (**p9**). Joe's proposed methods for preventing users from abusing the system were intuitive and I should be able to implement functions such as the limit on users frequently sending multiple requests (**p10**). The idea that some of these safeguards may be disabled as a default is particularly interesting and I look forward to discussing this with Mrs Marston-Smith. My only issue with his suggestions is that the weighting system may work out to give certain users an unfair advantage if it is not set up perfectly.

Data Flow Diagram Level 1

Using the information gained from the analysis, I extended the context diagram by creating my initial data flow diagram. I described the general features of the system and the data passed through it. This will be the basis of my design.



Requirements

Using the data collected in the analysis, I have created a table of requirements that should be fulfilled in order to have delivered a satisfactory product to my clients.

Input Requirements

ID	Requirement
i1	Must be controllable with an iPhone
i2	Users can control the system through a web interface
i3	Allow users to input their song requests as text
i4	Allow users to vote on songs in queue with buttons
i5	Allow users to skip unfavourable songs with buttons
i6	Every Sixth Form student and teacher will have a unique username and password
i7	Allow users to login with username and password

Processing Requirements

ID	Requirement
p1	Log user in if username and password are correct
p2	Allow users to send requests only when logged in
p3	Add requested songs to the end of a queue
p4	Move song request forward in queue based on number of votes received
p5	Skip the currently playing song if > 40% of active users request skip
p6	Allow certain users to have more influence in the voting algorithm
p7	Filter song requests for explicit content
p8	Only play song requests that are under 7 minutes long
p9	Return to start of queue when final song ends
p10	Allow one song request per user in a certain amount of time dependent on number of active users

Output Requirements

ID	Requirement
o1	Stream requested songs from Spotify
o2	Display currently playing track name and cover art
o3	Display track name and cover art of next songs in queue
o4	Display all outputted information without refreshing or changing page
o5	Display all information in an appropriate format on desktop and mobile devices

Hardware/Software Requirements

Server

The system will comprise of an externally hosted website that allows users to input requests (e.g. add song to queue, skip song, vote) into a MySQL database. This database will feed into the main logic of the system that will be written in PHP and processed on the external server. The server side algorithms will use the database in conjunction with an API for Spotify to output the URL for the next song when the 'next song' request is made. The minimum requirements for the server ensure that a MySQL database can be maintained with PHP code being executed concurrently.

ID	Requirement
s1	Linux x86 32bit
s2	PHP 5.6.5
s3	MySQL 5.7
s4	2 CPU cores
s5	2GB free disk space
s6	2 GB RAM
s7	1 GB bandwidth per month

Player

The 'player' will be a PC connected to a set of speakers that runs a script to utilise the Spotify web API to stream the song through the Spotify client. The only operations that the player will perform will be playing the current song, sending a 'next song' request to the server when the song ends and periodically checking with the server to see if the current song has been skipped.

ID	Requirement
pl1	256 MB RAM
pl2	2GB free disk space
pl3	Windows XP with Service Pack 3
pl4	Connection to 2.1 speaker system
pl5	Spotify 2.1.1 client

Client

In general, I expect most users to access the website through their mobile phones' browser. Their browser must support JavaScript so that the client-side logic will be performed correctly.

ID	Requirement
c1	Internet browser with JavaScript support

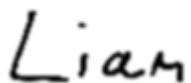
Client Agreement

I agree that the requirements specified in this document describe the product that I wish to be developed. A product that meets all of these requirements will satisfy the needs of the party that I represent.

Signed



Mrs Marston Smith



Liam Venencia:



John Jennings

Design Specification

I have compiled a list of criteria that my system should fulfil in order to meet the requirements specification. In my testing I will ensure that each objective is met. Some requirements could not be fulfilled until development so the status was marked as N/A

Input Requirements

<u>ID</u>	<u>Requirement</u>	<u>Objective</u>	<u>Status</u>
i1	Must be controllable with an iPhone	Must not use flash player and CSS must fit iPhone dimensions.	N/A
i2	Users can control the system through a web interface	Send requests from website with JavaScript into server	N/A
i3	Allow users to input their song requests as text	Create form on website where users can input song requests. When 'send' button is pressed, content of textbox will be sent to server as song request	Planned
i4	Allow users to vote on songs in queue with buttons	Create vote button for next 5 songs in queue. When pressed, send vote request to server with username and song name	Planned
i5	Allow users to skip unfavourable songs with buttons	Create skip button for currently playing song. When pressed, send skip request to server with username.	Planned
i6	Every Sixth Form student and teacher will have a unique username and password	Create users table with fields for username and password. Generate unique usernames from school register and randomly generate passwords with a VB.net script. Print out usernames and corresponding passwords and give to Mrs Hayes.	N/A
i7	Allow users to login with username and password	Users can obtain login information from Mrs Hayes. Input information through textboxes and press login button to send login request.	Planned

Processing Requirements

ID	Requirement	Objective	Status
p1	Log user in if username and password are correct	Get login request and compare username and password to records in user database. If a record with matching username and password is found, set user as logged in on client-side JavaScript and in user database.	Planned
p2	Allow users to send requests only when logged in	When user initiates request, check in user database whether user is logged in. If user is logged in then continue with request. Else output an error message.	Planned
p3	Add requested songs to the end of a queue	Create request database. When song request is received, use song name to get album art, URL and song length from Spotify with the API. Add all information to request database along with time that request was sent. Order database by request time.	Planned
p4	Move song request forward in queue based on number of votes received.	Create vote database. When vote request received, add username and URL of song to vote database. When song ends, increment request time of each song in database by number of votes for that song in vote database.	Planned
p5	Skip the currently playing song if > 40% of active users request skip	When skip request received, in user database flag user specified in request as having voted to skip song. Get number of active users and number of skip votes from user database and if number of skips is greater than (number of active users / 2.5), signal to player to skip song.	Planned
p6	Allow certain users to have more influence in the voting algorithm	Add field in user database for vote weight which can take values between 0 and 9. When updating queue with new votes, increment request time of each song by sum of vote weights of users that voted for that song.	Planned
p7	Filter song requests for explicit content	When Spotify returns search result object during song request, if song is flagged as explicit, do not continue with song request.	Planned
p8	Only play song requests that are under 7 minutes long	When Spotify returns search result object during song request, if song length is greater than 42000ms, do not continue with song request.	Planned
p9	Return to start of queue when final song ends	When song ends, in request database set request time to current time + 10000000 so that the play order is retained but new song requests will be added to top of queue.	Planned
p10	Allow one song request per user in a certain amount of time dependent on number of active users	When song request received, get number of active users from request database and input value into function to calculate appropriate cooldown time. Check that difference between last request time and current time is greater than cooldown time. If true, set last request time to current time and continue with request. Else, output error message.	Planned

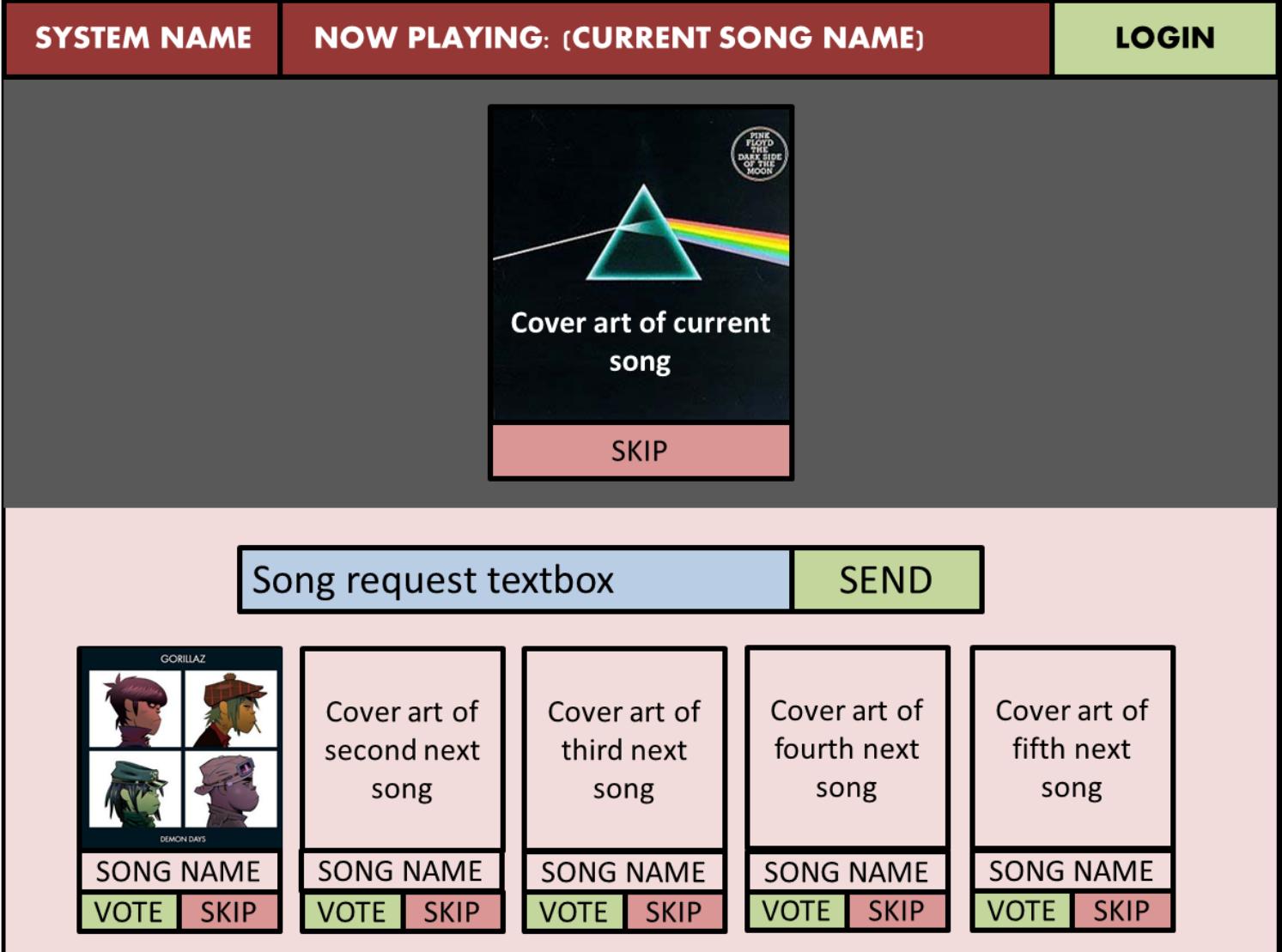
Output Requirements

ID	Requirement	Objective	Status
o1	Stream requested songs from Spotify	In request database, get URL and length of song with lowest request time. Play song in Spotify and repeat after length of song or if song is flagged as skipped.	Planned
o2	Display currently playing track name and cover art	In request database, get track name and cover art URL of song with lowest request time. Update HTML elements with this new data.	Planned
o3	Display track name and cover art of next songs in queue	In request database, get track name and cover art URL of songs with 2 nd -> 5 th lowest request times. Update HTML elements with this new data.	Planned
o4	Display all outputted information without refreshing or changing page	Use AJAX to call PHP functions and update HTML asynchronously without needed to refresh page. Have all elements of system available on one HTML document to be loaded only once.	N/A
o5	Display all information in an appropriate format on desktop and mobile devices	Detect whether user is on desktop or mobile device and apply separate style sheets accordingly.	N/A

User Interface Initial Design

Main Page

This is the homepage of the website that the user will use to control the system. From here, the user must click the 'Login' button in the top right corner in order to sign in and make the 'vote', 'skip' and 'send' buttons active. The modules for the next songs in the queue in the bottom right will be animated.



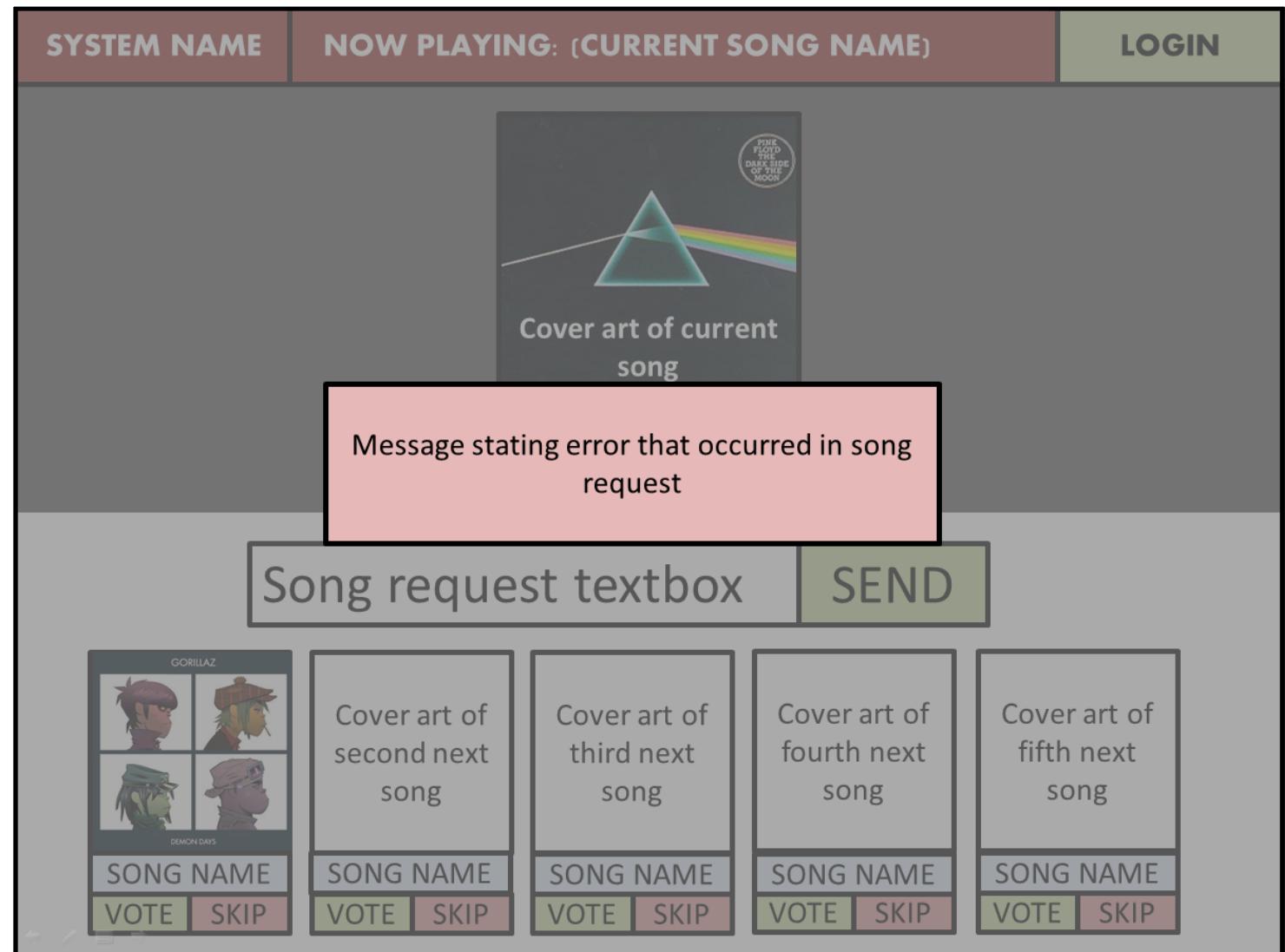
The main page interface is divided into several sections:

- Top Bar:** A red header bar containing "SYSTEM NAME", "NOW PLAYING: (CURRENT SONG NAME)", and a green "LOGIN" button.
- Center:** A large dark grey area featuring a "Cover art of current song" placeholder with a "SKIP" button below it.
- Bottom Section:** A light pink area containing:
 - A "Song request textbox" with a "SEND" button.
 - Five smaller modules showing "Cover art of second next song", "Cover art of third next song", "Cover art of fourth next song", and "Cover art of fifth next song". Each module includes a "SONG NAME" label and "VOTE" and "SKIP" buttons.
 - A separate module for "GORILLAZ DEMON DAYS" with four small character portraits and "VOTE" and "SKIP" buttons.

System Output

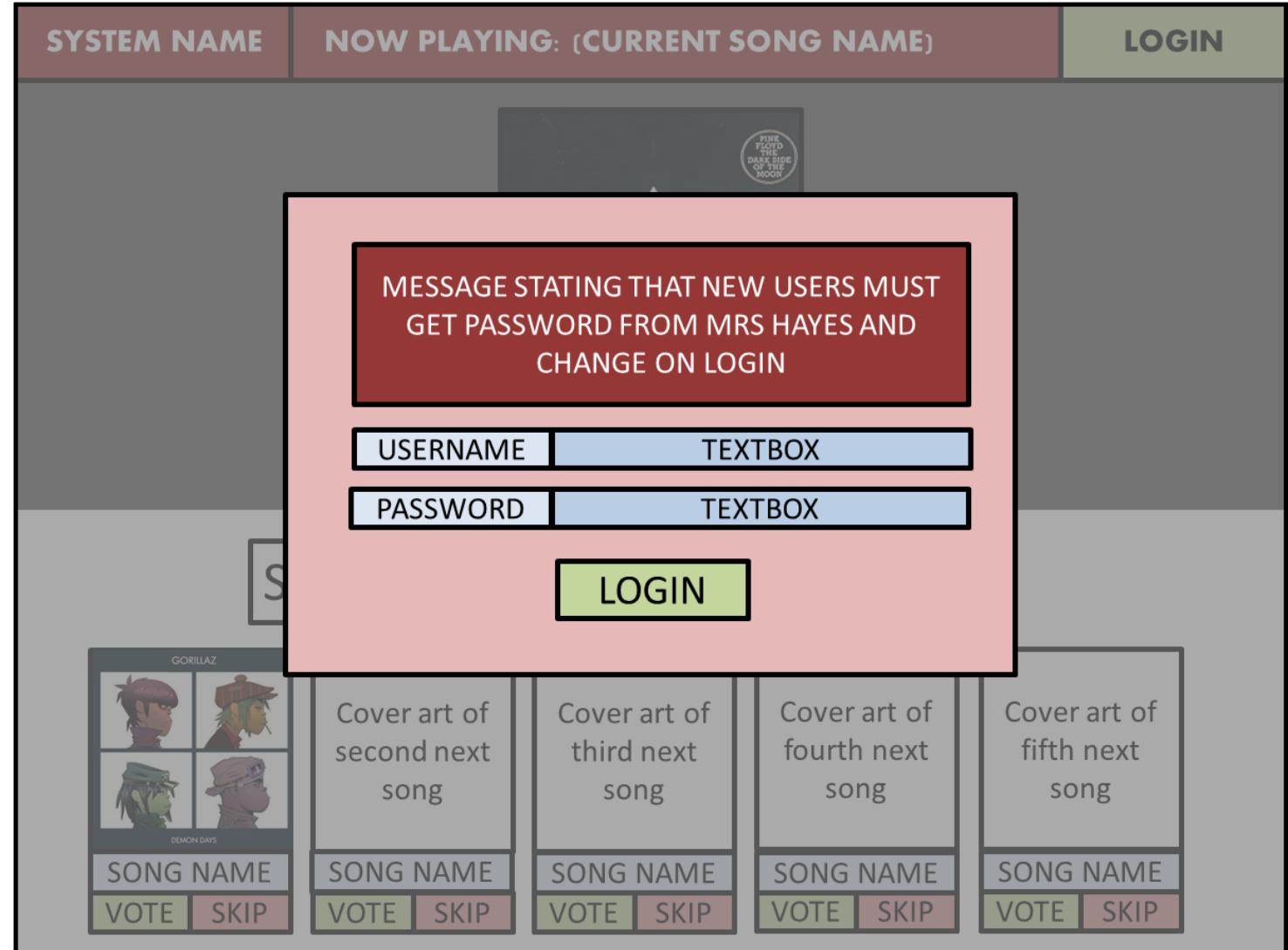
If the user performs an illegal action or an error occurs at any time, the website will be greyed out and all buttons will become inactive. A messagebox will be displayed on the screen giving details of the problem. The website will be in this state for 10 seconds before reverting to its original design.

A similar message box will appear when a request has been successfully processed. This colour will be green to differentiate the updates from the errors.



Login

When the user clicks the login button in the top right side of the window, the website will be greyed out identically to when a message is displayed. A form will appear with a message detailing where users can find their username and original password. There will also be two text boxes for inputting the username and password and a login button that will send the login request to the system and return the website to its default state when pressed.



Design Requirements

After creating the mock-ups of the user interface, I formalised the design requirements with my clients. This will be used as the main reference point when developing the HTML and CSS of the system.

ID	Requirement
d1	Dark red horizontal navbar at top of page containing d2,d3,d4
d2	System name in first 30% of navbar. White font colour
d3	Currently playing song name as horizontal marquee in next 40% of navbar, white font colour
d4	Green login button in final 30% of navbar
d5	Dark grey jumbotron below navbar, 50% page height containing d6,d7
d6	Thumbnail of currently playing song art, 25% page height, horizontally centred
d7	Red skip button below thumbnail, font colour white, 10% page width
d8	Search module, 50% page width horizontally centred below jumbotron. Containing d9, d10.
d9	Text box for requesting song, black font colour, 40% page width
d10	Green search button adjacent to search box, white font colour, 10% page width
d11	Five next song modules, 15% page width, containing d12, d13, d14, d15. Arranged in equally spaced row.
d12	Thumbnail of next song to play, at top of next song module, 90% module width.
d13	Name of next song to play, below thumbnail.
d14	Green vote button, below song name, first 50% of module width, font colour white.
d15	Red skip button, below song name, last 50% of module width, font colour white.
d16	Light pink page background

Data Dictionary

This table contains a concise profile of every variable and field established in the design process. It was updated and referred to throughout the development, ensuring concurrency between all different aspects of the document.

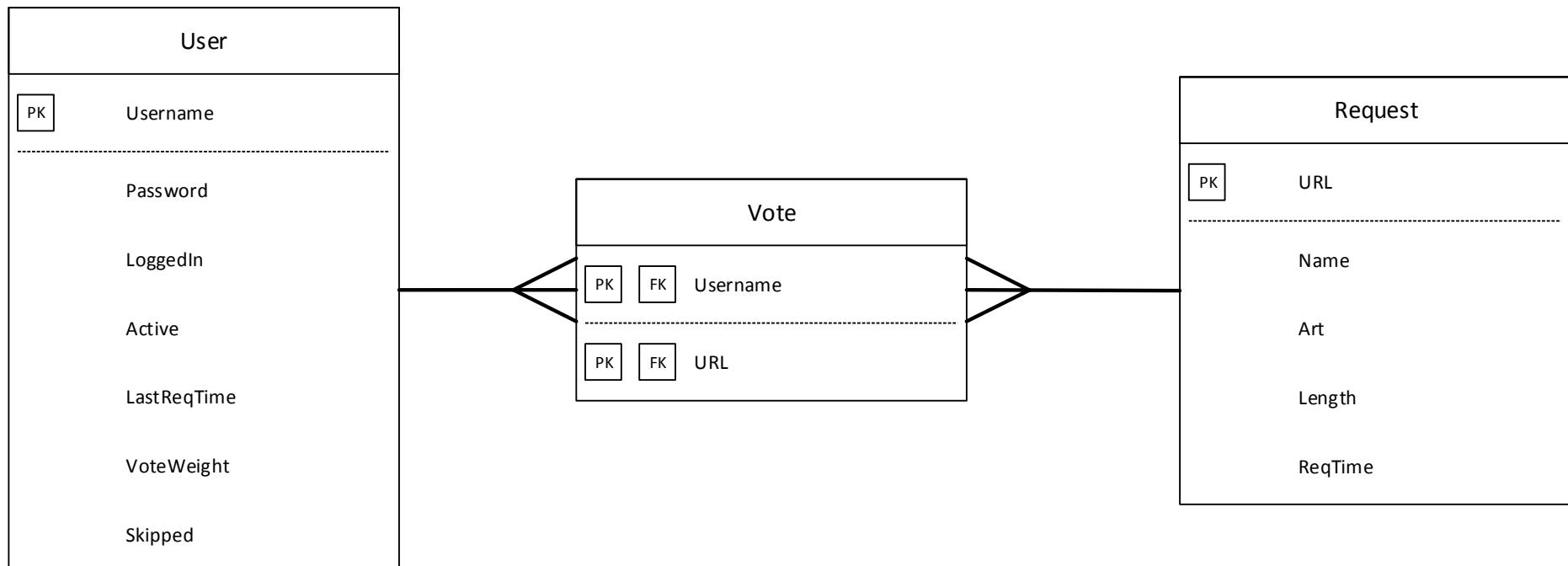
DataID	Field/Variable Name	Used By Database/Process								Data Type	Description	Example
		User	Request	Vote	1	2	3	4	5			
1a	LoginValid	■			■					Boolean	TRUE if user login is successful	TRUE,FALSE
1b	LoginStatusMsg				■					String	Message sent to notify user of login status	"Login Successful", "Login Unsuccessful"
2a	Request				■					String	Contains data of generic request	"08jennings,song,smoke on the water"
2b	LoginRequiredMsg				■					String	Message sent to user if request sent prior to user being logged in	"Login Required"
2c	RequestType				■					String	Part of request used to decide how it should be processed	"song", "vote", "skip"
3a	SongReq				■	■				String	Contains requested song name and username that sent request	"08jennings,smoke on the water"
3b	SearchTerm				■	■				String	SongReq with user removed. Used to search Spotify for song	"smoke on the water"
3c	TrackObject				■	■				JSON Object	Returned data of first result in Spotify search for song request.	{URL, Name ,Art ,Length ,Explicit}

DataID	Field/Variable Name	Used By Database/Process									Data Type	Description	Example
		User	Request	Vote	1	2	3	4	5	6			
3d	Length										Integer	Length in milliseconds of track	34342354
3e	Explicit										Boolean	TRUE if Spotify identifies track as containing explicit content	TRUE, FALSE
3f	SearchResult										JSON Object	Data sent to user to confirm that correct song request is found	{Name, Art}
3g	ChoiceOk										Boolean	Return TRUE if user confirms request for song	TRUE, FALSE
3h	ActiveUsers										Integer	Number of records in User database with field UserActive > 0	45
4a	VoteReq										String	Contains URL of song voted for and username that sent vote request	{Username, URL}
4b	VoteSuccessMsg										String	Sent to user to inform that vote request was fulfilled	"Vote success"
4c	Voted										Boolean	Indicates whether user has voted on the song already	TRUE
5a	SkipReq										String	Contains URL of song to be skipped and Username that sent request	{Username, URL}
5b	Skipped										Boolean	Returned TRUE if user has already skipped song	TRUE, FALSE
6a	NextSongPass										Boolean	Return TRUE if next song has passed all validation	TRUE, FALSE
6b	Audio										Audio	Audio stream of song from Spotify	N/A

DataID	Field/Variable Name	Used By Database/Process								Data Type	Description	Example
		User	Request	Vote	1	2	3	4	5			
R1	URL									String	Unique ID of track from Spotify	"6rqhFgbbKwnb9MLmUQDhG6"
R2	Name									String	Track title of song from Spotify	"Smoke on the Water"
R3	Art									String	URL of cover art of song	"https://i.scdn.co/image/39e0f9d708957f38361d7c391b4860c3894eb266"
R4	Length									Integer	Length of song from Spotify in milliseconds	2222652
R5	reqTime									Single	UNIX timestamp for time that song request is sent	141880556
U1	Username									String	Unique username assigned to each pupil in Y13	"08jjennings"
U2	Password									String	Password of corresponding user	"password1"
U3	LoggedIn									Boolean	Whether the user is logged in	TRUE, FALSE
U4	Active									Integer	Used to determine if user is active within 3 songs.	3 , 0
U5	LastReqTime									Integer	UNIX timestamp of last time user sent request	2345345345
U6	VoteWeight									Single	Value assigned to each user that indicates influence of their vote. Default = 1	1.3
U7	Skipped									Boolean	Whether user has voted to skip current song	TRUE

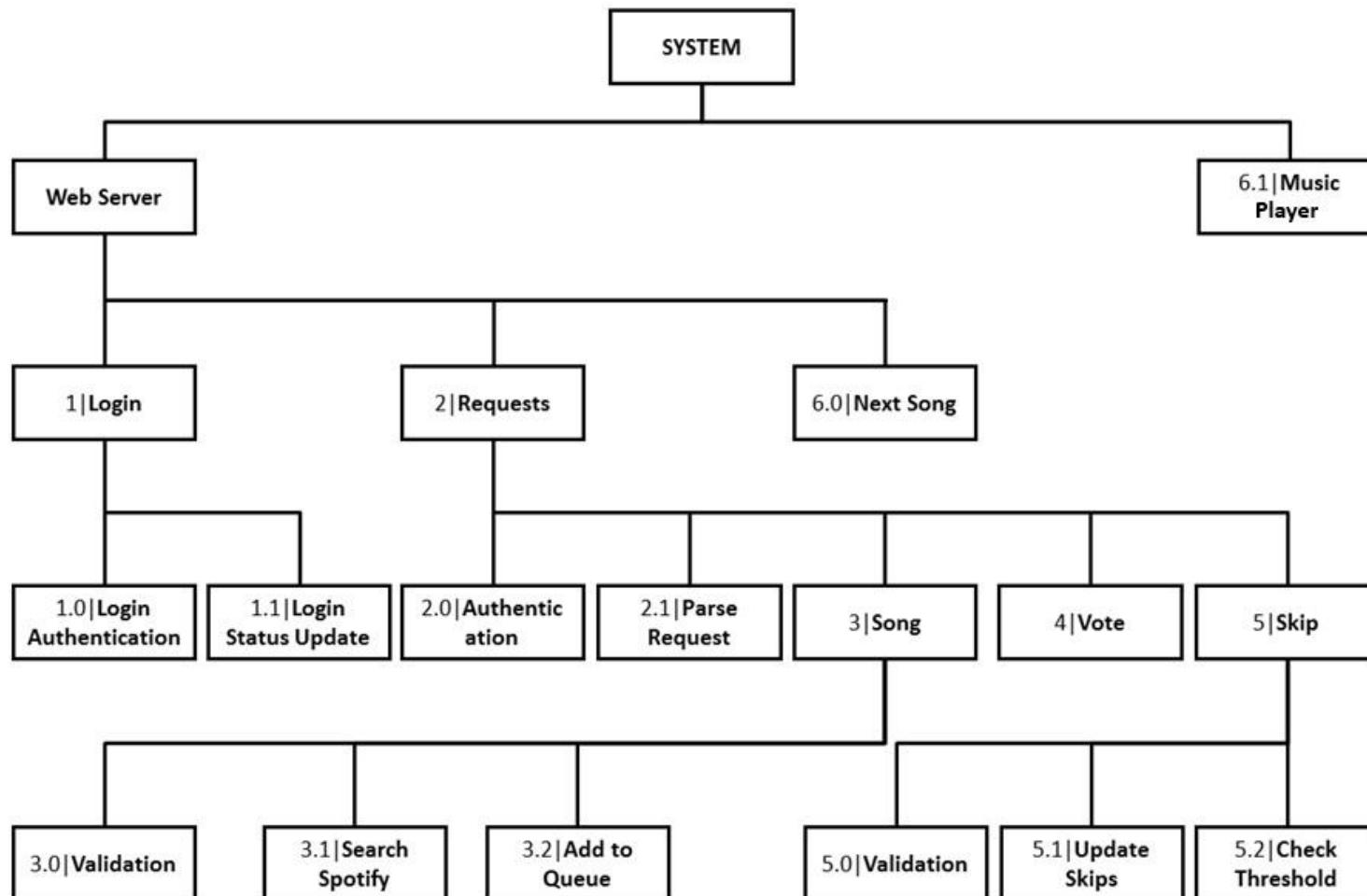
Entity Relationship Diagram

The User and Request databases do not have a direct relationship but a user can make many vote requests for many songs in the request database. This means that a single user and a single request can have a relationship as long as the user has voted for that song. This relationship allows for more complex calculations that require information from both the User and Request databases.



Hierarchy Chart

The original problem was divided into 6 processes which were again divided into their logical sub-processes. This allowed me to deconstruct the system into more manageable modules that could be treated as black boxes in the general design before working out the specific operations of each process later on.



Process Specification

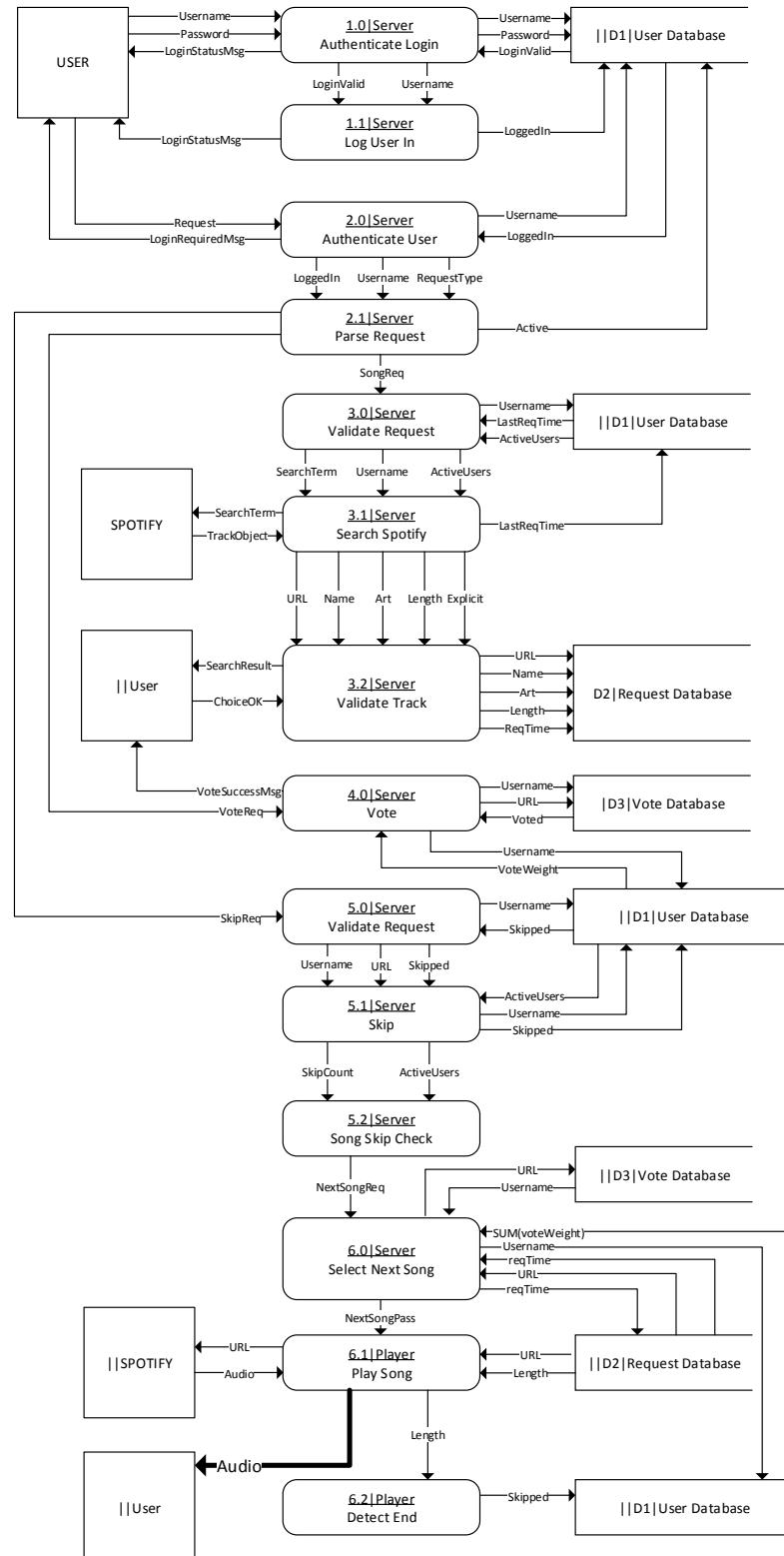
I then used the hierarchy chart to formally describe each process; showing the inputs, outputs and the basic manipulation of the data. This will be referred to when designing the algorithms for each process.

<u>ID</u>	<u>Process Name</u>	<u>Requirements Met</u>	<u>Process Description</u>	<u>User Input</u>	<u>System Input</u>	<u>User Output</u>	<u>System Output</u>
P1	Login	i7, p1	Takes username and password as input. Evaluates whether matching username and password are present in database. If true, flags user as logged in. Else displays error message.	Username, Password	LoginValid	LoginStatusMsg	LoggedIn
P2	Request	i3, i4, p2	Takes request as input. Splits data in request into separate variables. Checks whether user is flagged as logged in on database. If true, flags user as active in database and calls relevant request function for specified request type. Else, displays error message.	Request	LoggedIn	LoginRequiredMsg	SongReq, VoteReq, SkipReq, Active
P3	Song	i3, p3, p7, p8, p10, o2, o3	Takes songReq as input. Splits songReq into username and searchTerm. Gets last time of song request by user from database. Gets count of active users from database. Calculates cooldown time dependent on number of active users and evaluates whether cooldown time has passed. If true, set lastReqTime for user as current time, search Spotify with searchTerm. Splits returned object into URL, name, art, length and explicit. If song less than 7 minutes long and not explicit, output searchResult. If user inputs that choice is correct, create row in request database with URL, name, art, length and reqTime of song. If cooldown not elapsed, displays error message.	ChoiceOK	SongReq, LastReqTime, ActiveUsers, TrackObject	SearchResult	SearchTerm, LastReqTime URL, Name, Art, Length, ReqTime

P4	Vote	i4	Takes voteReq as input. Splits voteReq into username and URL. Evaluate whether matching username and URL already present in vote database. If false, create new row in database with URL and username.		VoteReq, Voted, VoteWeight	VoteSuccessMsg	
P5	Skip	i5, p5	Takes skipReq as input. Splits skipReq into username and URL. Gets whether user has skipped already from database. If false, gets count of active users, flags user as having skipped in database, gets count of skip requests. If skipCount > activeUsers/2.5, sends nextSong request.		SkipReq, Skipped, ActiveUsers		Skipped, NextSongReq
P6	Next Song	p4, p6, p9, o1	For first five songs in request database sorted by time of request in descending order, gets URL and adds sum of vote weights of each vote for that song to request time. Sorts database with new request times and gets first URL and its length. Plays song in Spotify client with URL until end of song length. Sets all users to have not skipped song. Sets request time of finished song to current time + 100000. Repeats.		NextSongReq, SUM(VoteWeight), ReqTime, URL, Length, Audio	Audio	ReqTime, Skipped

Data Flow Diagram Level 2

I then constructed this diagram to show the transfer of data within the system between the processes. Knowing what data is passed into and output by a process will give a starting point for designing its algorithm.

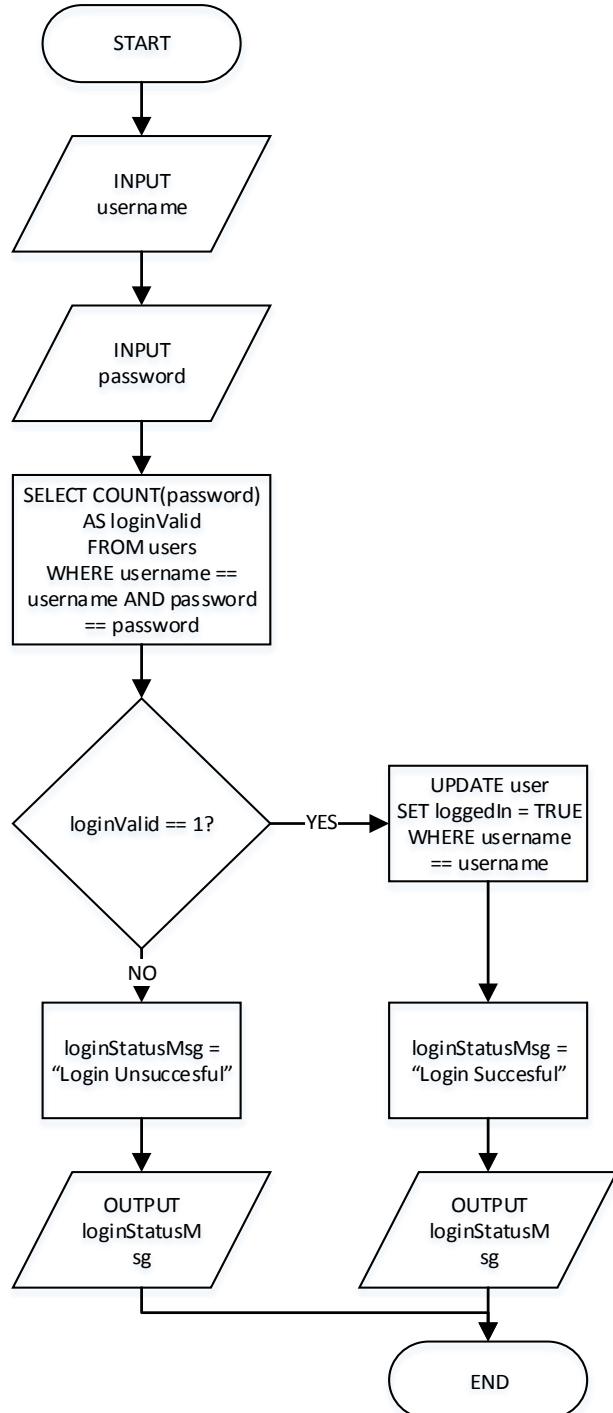


Finally, I used the DFDs and process specification to plan out algorithms for each process. I created Pseudocode and flowcharts in unison as one helps develop the other. Once I had finished a process, I would check that it correctly performed all functions defined in the design specification through dry runs recorded in trace tables.

Process 1 (Login)

Flowchart P1

This is the flowchart for the login process.



Pseudocode P1

This is the pseudocode for the login process.

```
01    START
02        INPUT username
03        INPUT password
04        SELECT COUNT(password) /* Returns 1 if username and password
are correct
05            AS loginValid
06            FROM user
07            WHERE username == username
08            AND password == password
09        IF loginValid == 1 THEN
10            UPDATE user /* Flag user as logged in
11            SET loggedIn = TRUE
12            WHERE username == username
13            loginStatusMsg = "Login Successful"
14            OUTPUT loginStatusMsg
15        ELSE /* if username or password incorrect
16            loginStatusMsg = "Login Unsuccessful"
17            OUTPUT loginStatusMsg
18        END IF
19    END
```

Trace Table P1

These are the trace tables for the login process. The first table shows that valid login credentials will log the user in while the second table shows that invalid login credentials will result in an error message being output.

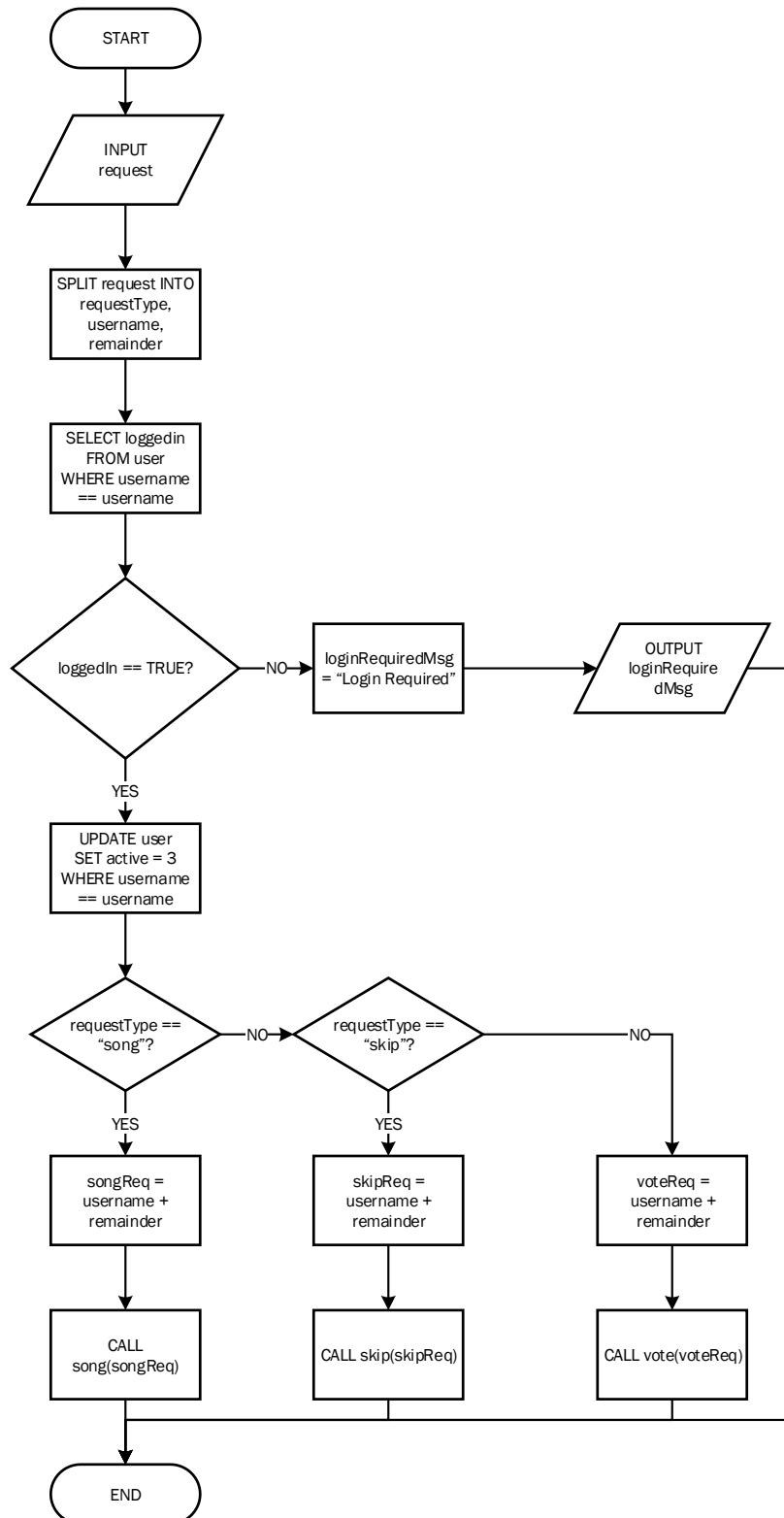
Process Variables					
Line	username	password	loginValid	loginStatusMsg	OUTPUT
00	""	""	0	""	
02	08jjennings				
03		pass1			
08			1		
12					
13				"Login Successful"	
14					"Login Successful"

Process Variables					
Line	username	password	loginValid	loginStatusMsg	OUTPUT
00	""	""	0	""	
02	08jjennings				
03		oops			
08			0		
16				"Login Unsuccessful"	
17					"Login Unsuccessful"

Process 2 (Request)

Flowchart P2

This is the flowchart for the request process.



Pseudocode P2

This is the pseudocode for the request process.

```
01      START
02          INPUT request
03          SPLIT (request,",") /* split request by comma
04          username = request[0]
05          requestType = request[1]
06          FOR i = 2 to (request[].length -1)
07              remainder = remainder + "," + request[i] /* collates all
excess data into comma separated string
08          NEXT
09          SELECT loggedIn /* returns TRUE if user is logged in
10             AS loggedIn
11             FROM user
12             WHERE username == username
13          IF loggedIn THEN
14              UPDATE user
15                  SET active = 3 /* flag user as active for 3 songs
16                  WHERE username == username
17          SELECT CASE requestType /* call appropriate procedure,
passing username and remainder string
18              CASE IS "song"
19                  songReq = username + "," + remainder
20                  song(songReq)
21              CASE IS "skip"
22                  skipReq = username + "," + remainder
23                  skip(skipreq)
24              CASE IS "vote"
25                  voteReq = username + "," + remainder
26                  vote(voteReq)
27          END SELECT
28          ELSE /* if user is not logged in
29              loginRequiredMsg = "Login Required"
30              OUTPUT loginRequiredMSg
31          END IF
32      END
```

Trace Table P2

These are the trace tables for the request process. The first table shows how a song request will be sent to the song request procedure if the user is logged on. The second table shows that an error message will be output if the user is not logged in. The third table shows how a vote request will be sent to the vote request procedure if the user is logged on.

Process Variables												
Line	request	i	request().length	username	requestType	remainder	loggedin	songReq	skipReq	voteReq	loginRequiredMsg	Output
0												
2	"08jennings,song,smoke on the water"											
4				"08jennings"								
5					"song"							
6		2	3									
7						"smoke on the water"						
12							TRUE					
19												"08jennings,smok e on the water"

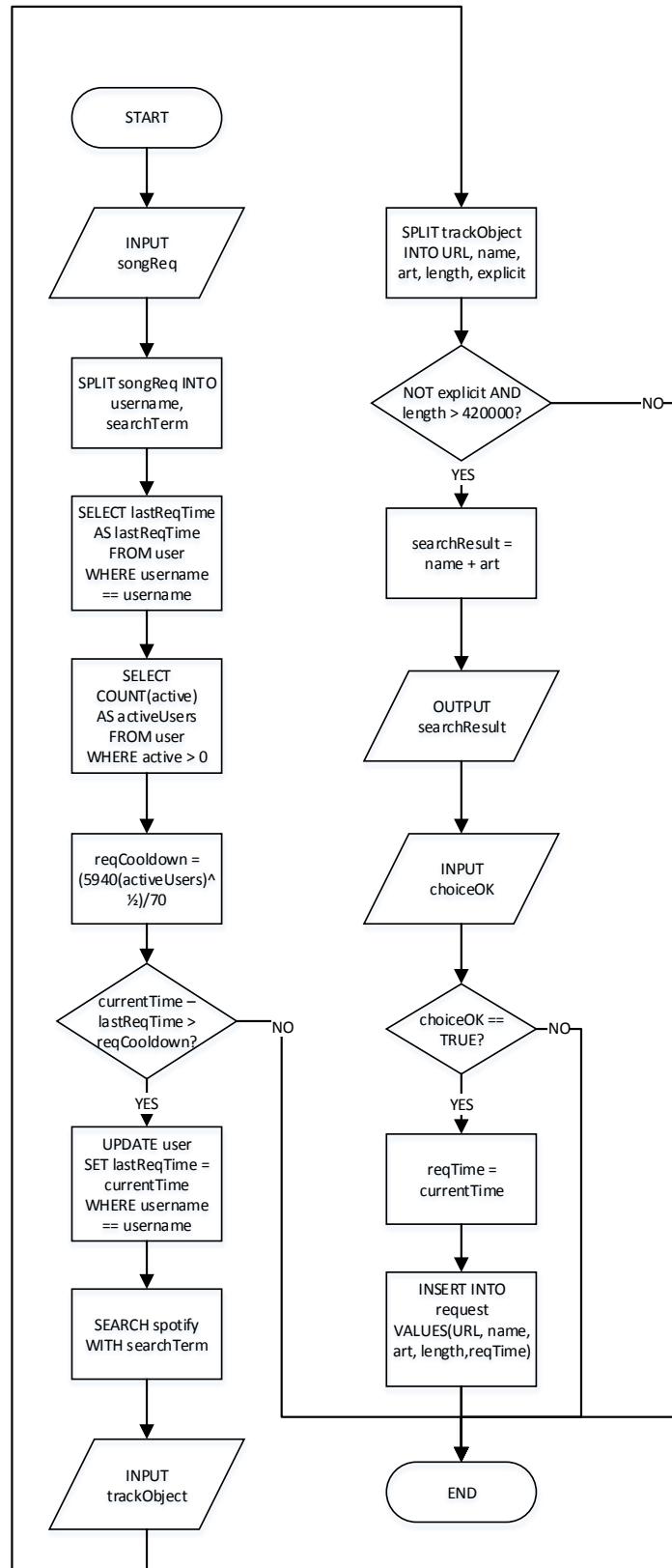
Process Variables												
Line	request	i	request().length	username	requestType	remainder	loggedin	songReq	skipReq	voteReq	loginRequiredMsg	Output
0												
2	"08jennings,song,smoke on the water"											
4				"08jennings"								
5					"song"							
6		2	3									
7						"smoke on the water"						
12							FALSE					
29											"Login Required"	
30												"Login Required"

Process Variables												
Line	request	i	request().length	username	requestType	remainder	loggedin	songReq	skipReq	voteReq	loginRequiredMsg	Output
0												
2	"08jennings,vote,spotify.c om/smoke"											
4				"08jennings"								
5					"vote"							
6		2	3									
7						"spotify.com/smoke"						
12							TRUE					
25												"08jennings,spotif y.com.smoke"

Process 3 (Song)

This is the flowchart for the song request process.

Flowchart P3



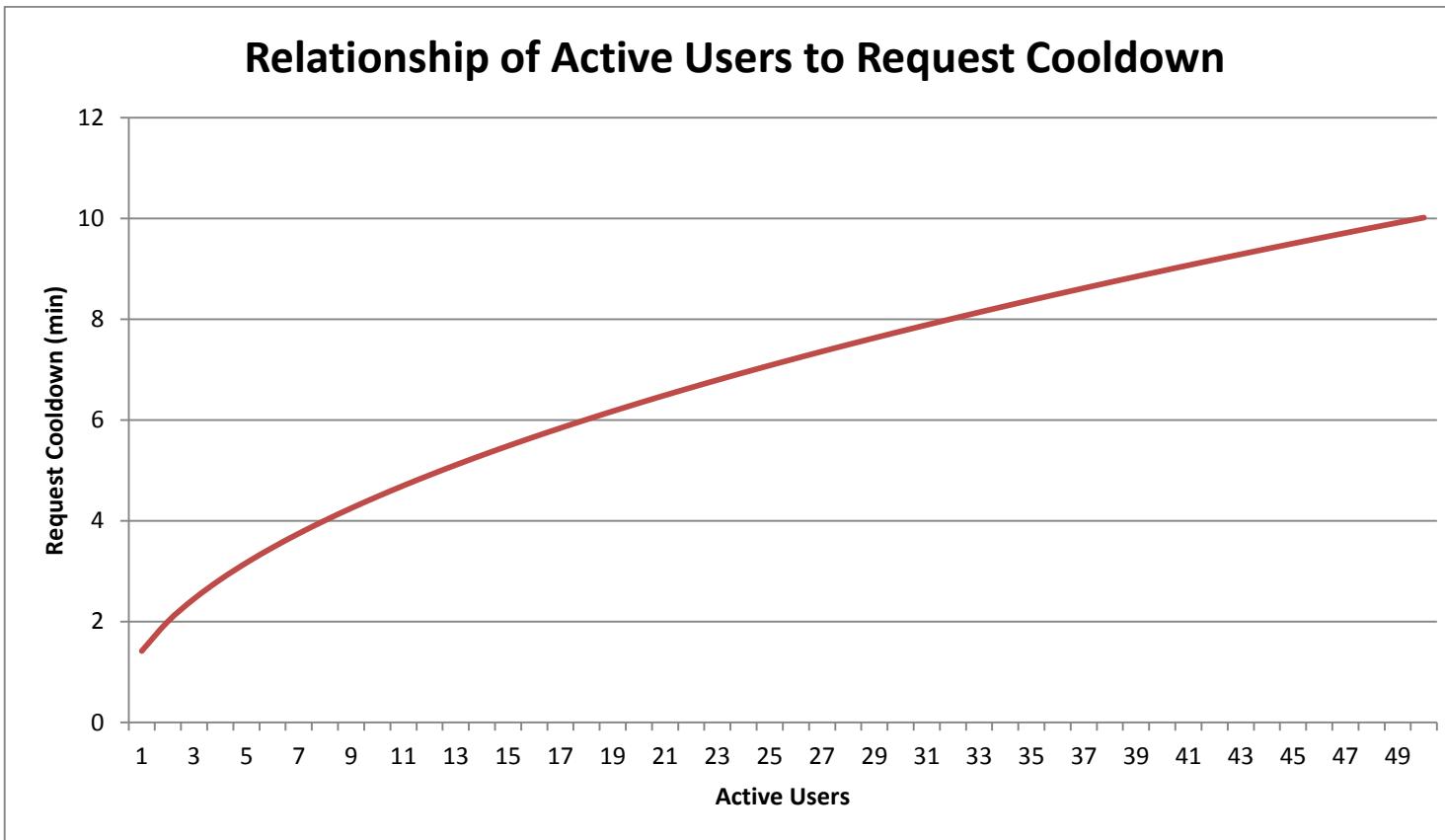
Pseudocode P3

This is the pseudocode for the song request process.

```
01      START
02          INPUT songReq
03          SPLIT(songReq,",")
04          username = songReq[0]
05          searchTerm = songReq[1]
06          SELECT lastReqTime /* returns unix timestamp of last songreq
by user
07              AS lastReqTime
08              FROM user
09              WHERE username == username
10          SELECT COUNT(active) /* returns number of active users
11              AS activeUsers
12              FROM user
13              WHERE active > 0
14      reqCooldown = 85*(activeUsers)^0.50 /* determines cooldown time (s)
from graph
15      IF (currentTime - lastReqTime) > reqCooldown THEN /* if
cooldown elapsed
16          UPDATE user
17              SET lastReqTime = currentTime
18              WHERE username == username
19              spotify.search(searchTerm)
20          INPUT trackObject
21          SPLIT (trackObject,",") /* split trackObject into
component parts
22          URL = trackObject[0]
23          name = trackObject[2]
24          art = trackObject[3]
25          length = trackObject[4]
26          explicit = trackObject[5]
27          IF NOT explicit AND length < 420000 THEN /* if song not
explicit and below 7 mins
28              searchResult = name + art
29              OUTPUT searchResult
30              INPUT choiceOK
31              IF choiceOK THEN /* if user confirms song choice
32                  reqTime = currentTime
33                  INSERT INTO request
34                      VALUES(URL,name,art,length,reqTime)
35              END IF
36          END IF
37      END IF
38  END
```

Trace Table P3

This is the trace table for the song request process. It shows how a song request will be sent to Spotify and the resulting object will be split into separate variables and added to the request database.

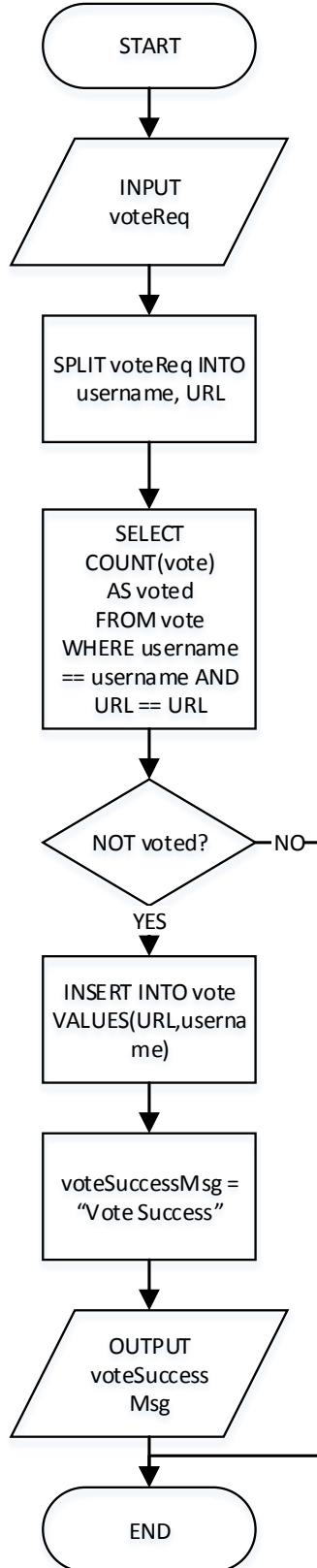


The system should allow users to request songs more frequently when they are low in number while also not creating an unfair restriction at higher user counts. The graph $y = 85(x)^{1/2}$ provides this functionality with a maximum cooldown time of 10 minutes and allowances for consecutive requests at user counts of 5 or less. I have therefore used this equation to generate the reqCooldown variable in the vote process in order to achieve the target relationship.

Process 4 (Vote)

This is the flowchart for the vote request process.

Flowchart P4



Pseudocode P4

This is the pseudocode for the vote request process.

```
01      START
02          INPUT voteReq
03          SPLIT(voteReq, ",")
04          username = voteReq(0)
05          URL = voteReq(1)
06          SELECT COUNT(vote) *\  
           \ returns 1 if user has voted for song
already
07              AS voted
08              FROM vote
09              WHERE username == username
10                  AND URL == URL
11          IF NOT voted THEN *\  
           \ if user has not voted for song
12              INSERT INTO vote
13                  VALUES(URL,username)
14              voteSuccessMsg = "Vote Success"
15              OUTPUT voteSuccessMsg
16          END IF
17      END
```

Trace Table P4

These are the test tables for the vote request process. The first shows how a user who has not yet voted for a song will have their vote added to the system. The second shows how a user who has already voted for a song will be rejected by the system.

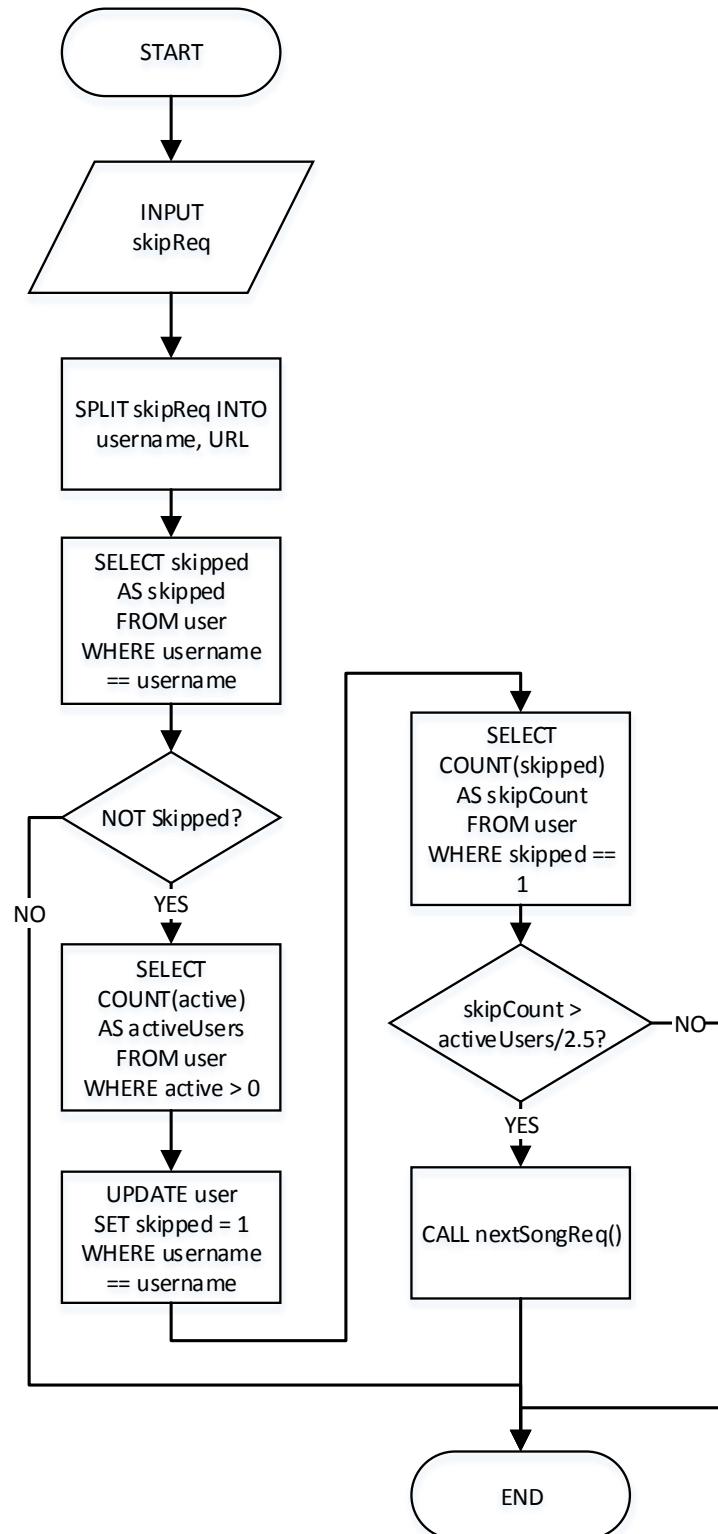
Process Variables							
Line	voteReq	username	URL	voteWeight	voted	voteSuccessMsg	OUTPUT
00	""	""	""	0	""	""	
02	"08jjennings,spotify.com/smoke"						
04		"08jjennings"					
05			"spotify.com/smoke"				
09				0.7			
14					FALSE		
18						"Vote Success"	
19							"VoteSuccess"

Process Variables							
Line	voteReq	username	URL	voteWeight	voted	voteSuccessMsg	OUTPUT
00	""	""	""	0	""	""	
02	"08jjennings,spotify.com/smoke"						
04		"08jjennings"					
05			"spotify.com/smoke"				
09				0.7			
14					TRUE		

Process 5 (Skip)

Flowchart P5

This is the flowchart for the skip request process.



Pseudocode P5

This is the pseudocode for the skip request process.

```
01      START
02          INPUT skipReq
03          SPLIT(skipReq,",")
04          username = skipReq[0]
05          URL = skipReq[1]
06          SELECT skipped /* returns 1 if user has skipped current song
07              AS skipped
08              FROM user
09              WHERE username == username
10          IF NOT skipped THEN /* if user has not skipped yet
11              SELECT COUNT(active) /* returns number of active users
12                  AS activeUsers
13                  FROM user
14                  WHERE active > 0
15              UPDATE user
16                  SET skipped = 1 /* flags user as having skipped
song
17                  WHERE username == username
18              SELECT COUNT(skipped) /* returns number of skip votes
for current song
19                  AS skipCount
20                  FROM user
21                  WHERE skipped == 1
22              IF skipCount > (activeUsers/2.5) THEN /* if number of
skip votes over 40% threshold
23                  nextSongReq()
24              END IF
25          END IF
26      END
```

Trace Table P5

These are the test tables for the skip request process. The first shows how a user who has not yet voted to skip a song will have their skip vote added to the system. The second shows how a user who has already voted to skip a song will be rejected by the system.

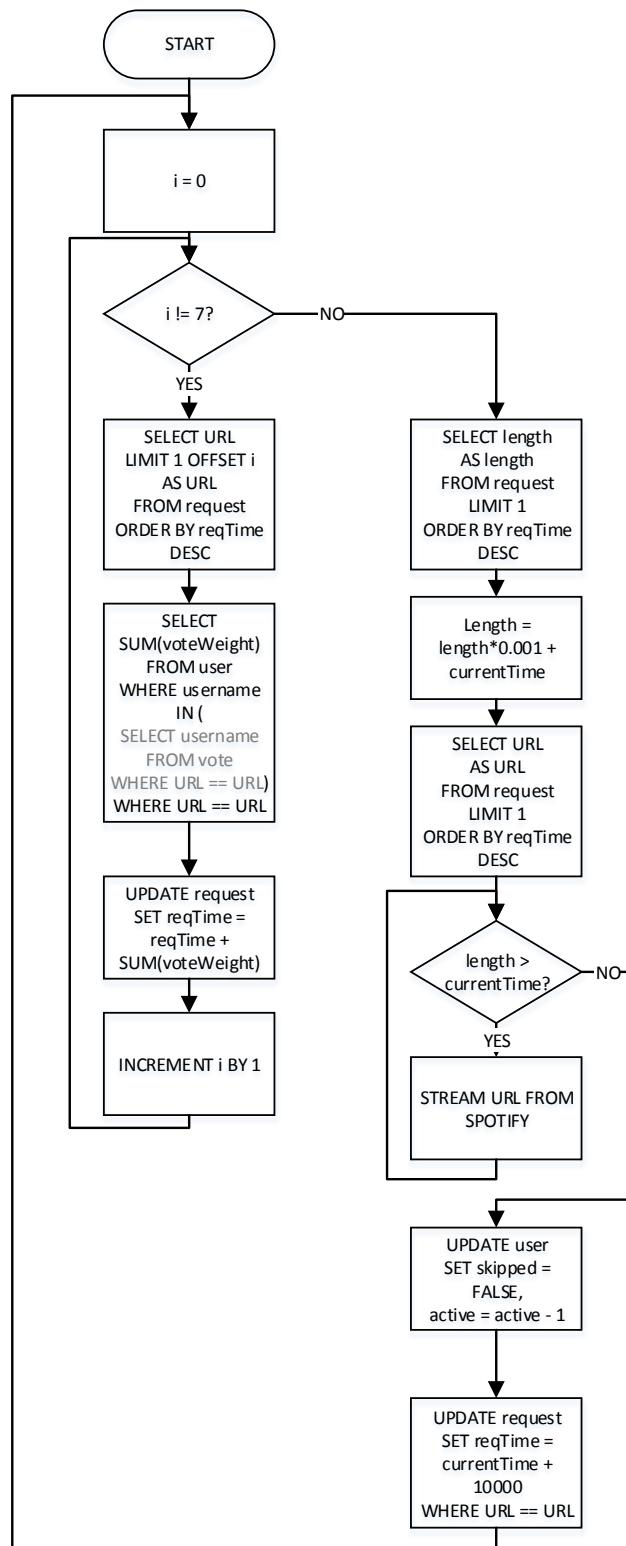
Process Variables							
Line	skipReq	username	URL	skipped	activeUsers	skipCount	OUTPUT
0	""	""	""				
2	"08jjennings,spotify.com/smoke"						
4		"08jjennings"					
5			"spotify.com/smoke"				
9				0			
14					17		
18						13	

Process Variables							
Line	skipReq	username	URL	skipped	activeUsers	skipCount	OUTPUT
0	""	""	""				
2	"08jjennings,spotify.com/smoke"						
4		"08jjennings"					
5			"spotify.com/smoke"				
9				1			

Process 6 (Next Song)

Flowchart P6

This is the flowchart for the next song process.



Pseudocode P6

This is the pseudocode for the next song process.

```

01  START
02      DO
03          FOR i = 0 to 4 /* for each candidate next song
04              SELECT URL /* returns URL of ith song
05                  LIMIT 1 OFFSET i
06                  AS URL
07                  FROM request
08              UPDATE request
09                  SET reqTime = reqTime + ( /* increment
reqTime of song by sum of vote weights
10                     SELECT SUM(voteWeight)/* sums all
voteweights of returned usernames
11                     FROM user
12                     WHERE username IN ( /* returns
all usernames that have voted for song
13                     SELECT username
14                         FROM vote
15                         WHERE URL == URL)
16                     WHERE URL == URL
17             NEXT
18             SELECT length /* returns length of song (ms)
19                 AS length
20                 FROM request
21                 LIMIT 1
22                 ORDER BY reqTime DESC
23                 length = length*0.001 + currentTime /* converts song
length to seconds and adds to current UNIX timestamp
24                 SELECT URL /* returns URL of song to play
25                     AS URL
26                     FROM request
27                     LIMIT 1
28                     ORDER BY reqTime DESC
29                     WHILE length > currentTime /* loop until song finished
30                         spotify.stream(URL)
31                     END WHILE
32                     UPDATE user
33                         SET skipped = FALSE, active = active - 1 /* flags
all users as not having skipped song and decrements their active value
34                     UPDATE request /* moves finished song to end of queue
35                         SET ReqTime = CurrentTime + 10000
36                         WHERE URL == URL
37             LOOP
38     END

```

Trace Table P6

This is the trace table for the next song process. It shows how before playing a song the system re-orders the queue with respect to the votes that each song has received and the plays the song at the top of the queue.

Process Variables						
Line	i	URL	reqTime	SUM(voteWeight)	length	currentTime
00	0					
03	0					
07		"spotify.com/smoke"				
15				345		
16			1334353536			
03	1					
07		"spotify.com/test1"				
15				23		
16			1111353536			
03	2					
07		"spotify.com/test2"				
15				11		
16			123353536			
03	3					
07		"spotify.com/test3"				
15				45		
16			1001353536			
03	4					
07		"spotify.com/test4"				
15				3		
16			1111300102			
22					120004	
23					1334496540	1334376536
28		"spotify.com/smoke"				

Testing Plan

To ensure that my end product satisfies the requirements outlined in this document, I must perform tests on the system. My alpha testing stage will consist of the tests described in the table below. I developed these tests to verify that the algorithms were working correctly and the system was behaving in the correct manner to satisfy the process requirements. I also added other tests that did not explicitly test a requirement but ensured that the system was robust and handled a varying set of inputs correctly. I will perform these tests at the end of the development of each process and the development of a system as a whole.

As soon as the system is in a usable state, I will also beta test the user experience and compatibility by letting a focus group of 20 students use the product. I will conduct a survey to gauge each user's personal opinion of the system as a whole, focussing on responsivity, ease of use and aesthetics. I will also discuss any possible improvements with each user and try to make adjustments to the product based on their feedback.

Test Number	Process Tested	Requirement Tested	Type	Data Entered / Action Performed	Expected Result
1	P1 (Login)	Check that valid user credentials lead to login (p1)	Normal	Username: 08cliggins Password: 2773	Website changes to logged in mode. User flagged as logged in in database
				Username: 08amaher Password: 655117DINKUM	
				Username: 08cpreece Password: 44235427205MAYO	
2		Check that invalid passwords for valid usernames are rejected.	Invalid	Username: 08cliggins Password: invalid	Message stating that login was unsuccessful. No change in database
				Username: 08amaher Password: INVALID	
				Username: 08cpreece Password: 2345345346	
3		Check that invalid usernames are rejected	Invalid	Username: Invalid Password: Invalid	Message stating that login was unsuccessful. No change in database
				Username: 45645766 Password: 123123	
				Username: INVALID Password: qwerty	
4		Check that empty username and password fields are rejected	Invalid	Username empty Password empty	Message stating that login was unsuccessful. No change in database
				Username empty Password: test	
				Username: test Password empty	

5	P1 (Login)	Check that the shortest username and password are accepted if valid	Borderline	Username: 08scay Password:LOTIONICEMEN664 Username: 08troberts Password: 2049	Website changes to logged in mode. User flagged as logged in in database
6		Check that the longest username and password are accepted if valid		Username: 08kbroadhurst Password: 985404 Username: 08bmcgrail Password: GLOBEDKUDZUSMEA	
7	P2 (Request)	Check that requests are accepted for users that are logged in (p2)	Normal	Username: 08troberts Request: Skip	Request is processed successfully
8				Username: 08troberts Request: Vote	
9				Username: test1 Request: Song	
8	P2 (Request)	Check that requests are rejected for users that are not logged in	Normal	Username: 08tmacklin Request: Vote	Message stating that login is required
9				Username: 08tmacklin Request: Skip	
9				Username: 08vadcock Request: Song	
9	P2 (Request)	Check that users are flagged as active when request is accepted.	Normal	Username: 08troberts Request: Skip	Active value in user's record is set to 10 in database
9				Username: 08troberts Request: Vote	
9				Username: test1 Request: Song	

15	P3 (Song)	Check that song is added to queue when requested by user(p3)	Normal	Search: Hello by Lionel Richie	New record for Hello by Lionel Richie created in database
				Search: Tip Toe Thru' The Tulips With Me by Tiny Tim	New record for Tip Toe Thru' The Tulips With Me by Tiny Tim created in database
				Search: Smoke on the Water by Deep Purple	New record for Smoke on the Water by Deep Purple created in database
16		Check that no song is added to queue when no song is selected	Invalid	Search empty	No activity
17		Check that no song longer than 7 minutes is added to queue (p8)	Invalid	Artist: Deadmau5	New record for radio edit versions of songs by Deadmau5
				Artist: Daft Punk	New record for radio edit versions of songs by Daft Punk
18		Check that songs just over 7 minutes long are rejected	Borderline	Search: When the levee breaks	New record for shorter versions of When the Levee Breaks by Led Zeppelin
				Search: Around the World	New record for radio edit versions of Around the World by Daft Punk

19	P3 (Song)	Check that songs just under 7 minutes long are accepted	Borderline	Search: Ghosts 'n' Stuff Search: Musique	New record for Nero remix but not original song New record for Musique by Daft Punk
20		Check that explicit songs are rejected (p7)	Invalid	Search: Alphabet Lil Wayne	Message states that no results were found
				Search: Oh my Darling Don't Cry	Message states that no results were found
21		Check that songs already in queue are rejected	Invalid	Search: Don't Look Back into the Sun	Song not added to queue and message states that song already requested
21		Check that songs already in queue are rejected	Invalid	Search: T-Shirt Weather	Song not added to queue and message states that song already requested
22		Check that songs similar to songs already in queue are accepted	Borderline	Search: Hello J Cole Search: Can't Stand me Now	Song added to queue
23		Check that requests are rejected if sent within cooldown time (p10)	Invalid	Send two song requests consecutively	First request accepted, second request rejected with message stating still in cooldown
24	P4 (Vote)	Check that vote requests are added to database wen vote button pressed	Normal	Username: 08cliggins Vote button pressed	Vote added to database and vote button disabled
				Username: 08amaher Vote button pressed	
				Username: 08cpreece Vote button pressed	

25	P4 (Vote)	Check that duplicate votes are rejected	Invalid	Username: 08cliggins Vote button pressed twice	Vote not added to database
				Username: 08amaher Vote button pressed twice	
				Username: 08cpreece Vote button pressed twice	
26	P5 (Skip)	Check that user is flagged as skipped when skip button pressed	Normal	Username: 08cliggins Vote button pressed	User flagged as skipped in database, skip button disabled
				Username: 08amaher Vote button pressed	
				Username: 08cpreece Vote button pressed	
27	P5 (Skip)	Check that duplicate skip requests are rejected	Invalid	Username: 08cliggins Vote button pressed twice	Skip request not added to database
				Username: 08amaher Vote button pressed twice	
				Username: 08cpreece Vote button pressed twice	
28	P6 (Next Song)	Check that songs are played in order of queue	Normal	Open player	Spotify plays Smoke on the Water followed by Me Julie and Bring it All Back
29		Check that songs are played in their entirety	Normal	Open player	Spotify plays a new song only when the current song has ended
30		Check that songs are played without gaps	Normal	Open player	Spotify plays a new song immediately after current song has ended

31	P6 (Next Song)	Check that song is skipped if >40% of active users have voted to skip (p5)	Normal	Set 100 users as active, 40 as skipped. Press skip button	Current song ends and next song begins playing	
			Borderline	Set 100 users as active, 39 as skipped. Press skip button	Current song continues playing and skip button disabled	
32		Check that songs move forward in queue depending on how many votes they have received (p4)	Normal	Set 100 users as having voted for song at bottom of queue	When current song ends, song at bottom of queue moves forward	
33		Check that certain users have more influence in the voting algorithm (p6)	Normal	Set 1 user with weight 500 as having voted for song at bottom of queue, 100 users with weight 1 as having voted for second song from bottom of queue	When current song ends, both songs move forward in queue with first song now above second	
				Set 5 users with weight 10 as having voted for song at bottom of queue, 20 users with weight 2 as having voted for second song from bottom of queue		
33	P6 (Next Song)	Check that certain users have more influence in the voting algorithm	Borderline	Set 10 users with weight 50 as having voted for song at bottom of queue, 9 users with weight 50 as having voted for second song from bottom of queue	When current song ends, both songs move forward in queue with first song now above second	
34		Check that player returns to start of queue when final song ends (p9)	Normal	Open player	When Lovesick ends, Smoke on the Water begins	
35		Check that song is removed from queue when skipped	Normal	Set 100 users as active, 60 as skipped. Press skip button	Current song ends and is removed from database	
36		Check that users are flagged as having not skipped when next song begins playing	Normal	Open player, wait for first song to end	All users' skipped value is set to 0 in database	

Questionnaire

1. I can navigate the website easily

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

2. I feel like my actions have an effect on the system

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

3. I can easily find the songs that I want

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

4. The music playing on the system is of a high quality

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

5. The website is visually appealing

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

6. The website scales well on my device

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

7. The website feels responsive and dynamic

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

8. I rarely get frustrated when using the system

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

9. As a whole I feel that using the system is a positive experience

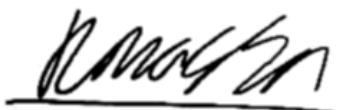
- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

To ensure that I had designed the system properly, I showed this documentation to the clients and asked for their signature to confirm that they were satisfied with my design.

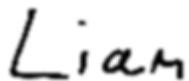
Client Agreement

I agree that the system outlined in this document describes the product that I wish to be developed. A product that meets all of these requirements will satisfy the needs of the party that I represent.

Signed



Mrs Marston Smith



Liam Venencia:



John Jennings

Webpage

Developing the client-side of the system consisted of three main steps. Outlining the general layout in HTML, describing the aesthetic elements in CSS and creating a dynamic experience with JavaScript. Since each stage built on the foundations of the previous, I ensured that I had completed all of the HTML code before I began the CSS and so forth. This allowed me to focus on one language at a time which made debugging considerably easier as I knew that only one file would change at a time; only having to make minor tweaks to the previous code as small problems were discovered.

Markup 1 (Head)

The first step in creating my website was ensuring that the libraries I would be using were correctly installed. I decided to use Twitter's open source CSS library, "bootstrap" as it would allow me to quickly create the basic page layout by assigning certain class identifiers to my HTML elements. Bootstrap would then take these classes such as ".btn" and apply a default style that I could then override and customise with my own style sheet, main.css. I also installed the jQuery library for JavaScript which adds extra functionality such as selecting a group of HTML elements by a shared class identifier and running an animation on them. This would allow me to create a dynamic experience for the user with more concise and readable client-side code.

To set up this environment I linked to the external resources in the head. Linking in the head ensures that the initial code is loaded before the page is displayed, meaning that the correct adjustments will be applied to the body before it is visible to the user. I loaded the files in a specific order to ensure that the corresponding code was executed in the correct order. My CSS file must be executed after bootstrap so that it can override the styles applied by bootstrap. The jQuery library must be loaded before my JavaScript, main.js, since my code calls jQuery functions that must be previously defined.

HTML

```
<head>
  <!-- load twitter bootstrap css template, used to create dynamic components e.g.
  navbar, jumbotron etc -->
  <link rel="stylesheet" href="dist/css/bootstrap.css">
  <!-- load css style for page -->
  <link rel="stylesheet" href="main.css">
  <!-- load jquery library, used to add functionality to and fix bugs in vanilla
  javascript e.g. animation and DOM object selection -->
  <script
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
  <!-- load javascript for page -->
  <script src="main.js"></script>
</head>
```

Requirements Met: i2, 04

Markup 2 (Navbar)

From the design, the horizontal bar containing the login button and currently playing song easily transferred into the navbar layout supported by bootstrap. Simply encasing the elements in a nav tag with the class “navbar” and some additional arguments caused bootstrap to automatically assign them with the generic layout that I required. It also added some extra functionality that I had not designed such as the navbar staying at the top of the screen as the user scrolled down the page. Another function of bootstrap that was used in the navbar and extensively in the website as a whole was the ‘row’ layout. In the navbar, encasing the elements in a ‘row’ div allowed me to quickly describe the horizontal positioning of each element with ‘column’ divs containing their size and width as an integer out of 12.

HTML

```
<nav class="navbar navbar-default navbar-fixed-top" role= "navigation">
  <div class="container">
    <!-- create bootstrap row with width 12, used to arrange elements into dynamic
columns -->
    <div class="row">
      <!-- create column with width 3, xs ensures that display is consistent on all
devices -->
      <div class="col-xs-3">
        <!-- Big Julie text -->
      </div>
      <!-- create column with width 6 -->
      <div class="col-xs-6">
        <!-- currently playing song-->
      </div>
      <!-- create column with width 2 to contain login textboxes -->
      <div class="col-xs-2">
        <!-- login forms-->
      </div>
      <!-- create column with width 6 -->
      <div class="col-xs-1">
        <!-- login button-->
      </div>
    </div>
  </div>
</nav>
```

Once the layout was described it was simply a case of adding in the appropriate elements from the design to their respective columns. Some elements of the design however, did not transfer well to HTML. For example, it was considerably harder than expected to have the login information appear as a separate window on top of the page. In every situation where I encountered this problem, I created a simpler to code alternative, explained the problem to my clients and worked with them to find a compromising solution. In this case, we decided that the login textboxes should appear in the navbar, next to the login button and only be visible if the user is not logged in. A client also decided that there should be a logo instead of text with the system name so that element was also changed. When creating the HTML, I commented each ‘dynamic’ element with the features that my JavaScript should provide them. This was to ensure that when I came to developing the JavaScript, I could easily refer to the HTML to see what functions I should implement.

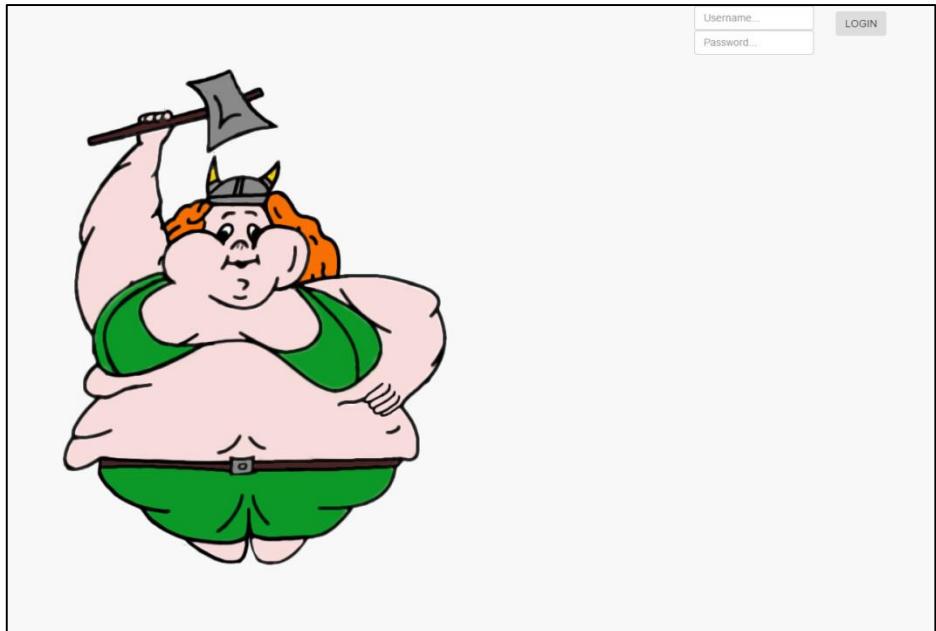
Final HTML

```

<nav class="navbar navbar-default navbar-fixed-top" role="navigation">
    <div class="container">
        <div class="row">
            <div class="col-xs-3">
                <!-- display Big Julie logo in column -->
                <div class="BJ">
                    
                </div>
            </div>
            <div class="col-xs-6">
                <!-- display marquee moving right to left with currently playing song name-->
                <!-- main.js updates text on page load and refresh function called -->
                <p id="nowPlaying" class="nowPlaying toRefresh"><marquee> </marquee></p>
            </div>
            <div class="col-xs-2">
                <div class="login form-group">
                    <div class="username">
                        <!-- create textbox with default text "Username..." -->
                        <!-- main.js gets text entered when login function called, fades
textbox out if login successful, calls login function when enter key pressed while in
textbox -->
                        <input id="username" type="text" class="form-control toToggle"
placeholder="Username..."/>
                    </div>
                    <div class="password">
                        <!-- create textbox with default text "Password..." -->
                        <!-- main.js performs same logic as to username -->
                        <input id="password" type="password" class="form-control toToggle"
placeholder="Password..."/>
                    </div>
                </div>
                <div class="col-xs-1">
                    <div class="btnLogin">
                        <!-- create button with text "LOGIN" -->
                        <!-- main.js calls login function when pressed and adds class "logged",
changes text to LOGOUT if login successful, main.css changes display accordingly-->
                        <button id="btnLogin" type="button" class="btn btn-default navbar-
btn">LOGIN</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</nav>

```

I then tested the HTML by hosting the file online and visiting the page through Google Chrome on my desktop and mobile phone. I used the inspect element feature of the browser to check that each element was positioned correctly.



Navbar in plain HTML

Once the HTML was finished, I created a style sheet that followed the design outlined by my clients and again tested by uploading the file and checking it's compatibility with different devices. I also temporarily added certain classes to the HTML to simulate different states of the website such as the user being logged in so that I could test that the style changed accordingly.

CSS

```
/* for entire page */
body{
    /* set background colour to light pink */
    background-color: #F7DEDC;
}

/* for entire navbar */
.navbar{
    /* set background colour to dark red */
    background: #9C3B35;
    /* match bottom border with jumbotron background */
    border-bottom-color: #2E2E2E;
    /* set navbar to be slightly transparent */
    opacity: 0.8;
    /* size adjustments for correct layout on desktop */
    padding-top: 5px;
    height: 85px;
}
/* for Big Julie logo */
.BJ img{
    /* set consistent size to fit in navbar */
    height: 70px;
}
/* for marquee displaying currently playing song title */
.nowPlaying {
    /* position in middle of navbar */
    text-align: center;
    /* set text colour to dark grey */
    color: #2E2E2E;
    /* size adjustments for correct layout on desktop */
    font-size: 30px;
    height: 70px;
    line-height: 75px;
}
/* for div containing login button */
.btnLogin {
    /* align button to far right of navbar, vertically centre */
    padding-top: 1px;
    text-align: right;
}
/* for login button */
.btnLogin button{
    /* set text colour to white, background colour to green */
    color: #FFF;
    background: #83BD6C ;
    /* size adjustments for correct layout on desktop */
    margin-top: 0px;
    font-size: 30px;
    height: 70px;
}
/* for login button when logged in */
.logged {
    /* override background to match navbar background */
    background: #9C3B35 !important;
    /* size adjustments for longer text */
    margin-left: -30px;
```



Navbar with CSS, user logged out



Navbar with CSS, user logged in

I then wrote the dynamic elements of the navbar such as the transition when a user logs in successfully. I used jQuery events to call animation functions that faded certain elements out, toggled the “logged” class in their HTML and faded them back in with the new styles applied from the CSS. This method was used throughout the site in situations where the style of an element would change without the layout being affected. I did not implement the actual login function yet as I wanted to finalise the layout of the entire page before I interacted with the backend. Instead, I designed the ‘visual’ functions so that they could easily be called by the frontend logic once it was implemented. Again, I tested the code by loading the webpage on different devices, clicking the login button and checking that the animations were executed correctly.

I designed the login/logout functions so that they behaved as a toggle instead of having two separate login and logout functions. This meant that system would never try to log in if it was already logged in and vice versa, removing any possible bugs that may occur in that situation.

JavaScript

Login button pressed

```
// if login button clicked
$('#btnLogin').click(function() {
    // call login function
    login();
});
// if enter key pressed in password box
$("#password").keypress(function(e) {
    if(e.which == 13) {
        login();
    }
});
```

Login button toggle

```
// toggle button between login and logout
$("#btnLogin").fadeToggle("slow",function() {
    // toggle button text
    if ($("#btnLogin").text() == "LOGIN") {
        $("#btnLogin").text("LOGOUT");
    }
    else {
        $("#btnLogin").text("LOGIN");
    }
    // toggle CSS of button
    $("#btnLogin").toggleClass("logged");
});
$("#btnLogin").fadeToggle("slow");
```

Requirements Met: i7, d1, d2, d3, d4, d16

Markup 3 (Jumbotron)

Once the navbar was completed, I continued through the design, creating the ‘jumbotron’. In web design, a jumbotron is the main focus of the site; a header that stretches horizontally across the page. This easily translated into the area in the design which contained the currently playing song. Using the same method as with the navbar, I used bootstrap to get the basic layout correct and then applied my own CSS to further refine the style. To test the layout I used placeholder cover art and text since the actual sources were not to be developed until later. When testing the site with a focus group, it was noted that the users did not like the fact that the skip button gave no feedback so they couldn’t tell whether they had actually pressed the button. To remedy this, I added another toggleable class, “skipped”. When the skip button was pressed, the button would become disabled and the colour would change to grey to indicate its inactivity. The button can then be reset to its default state in JavaScript when the next song is played. This also gave the added benefit of preventing multiple skip requests from one user being sent for one song. Although any request following the first would have no effect, they would use up bandwidth so it is useful to stop the requests from being sent at all instead of only validating the request on the server.

HTML

```
<div class="jumbotron">
  <div class="container">
    <!-- create thumbnail in centre of jumbotron, bootstrap.css handles layout -->
    <div class="thumbnail">
      <!-- display cover art of currently playing song -->
      <!-- main.js updates image on page load and refresh function called -->
      
    </div>
    <div class="btnSkip">
      <!-- create button with text "SKIP", not displayed until user logged in -->
      <!-- main.js calls skip function when pressed -->
      <button id="btnSkip" type="button" class="btn btn-default navbar-btn toToggle
locked" style="display: none;">SKIP</button>
    </div>
  </div>
</div>
```



Jumbotron in plain HTML

CSS

```
/* for password textbox */
.password{
    /* size adjustments for correct layout on desktop */
    padding-top: 5px;
}

/* for entire jumbotron */
.jumbotron {
    /* set background to dark grey */
    background: #2E2E2E;
    /* align elements in centre */
    text-align: center;
    /* size adjustments for correct layout on desktop */
    padding-top: 135px;
    height: 508px;
    margin-bottom: 15px;
}

/* for thumbnail containing currently playing song cover art */
.jumbotron .thumbnail {
    /* size adjustments for correct layout on desktop */
    display: inline-block;
    height:300px;
    width:300px;
    padding:3px;
}

/* for currently playing song cover art */
.thumbNow{
    /* size adjustments for correct layout on desktop */
    height:292px !important;
    width:300px;
}

/* for skip button */
.btnSkip button {
    /* set text to white, background to match navbar background */
    background-color: #9C3B35;
    color: #FFF;
    /* size adjustments for correct layout on desktop */
    font-size:30px;
    width: 250px;
}

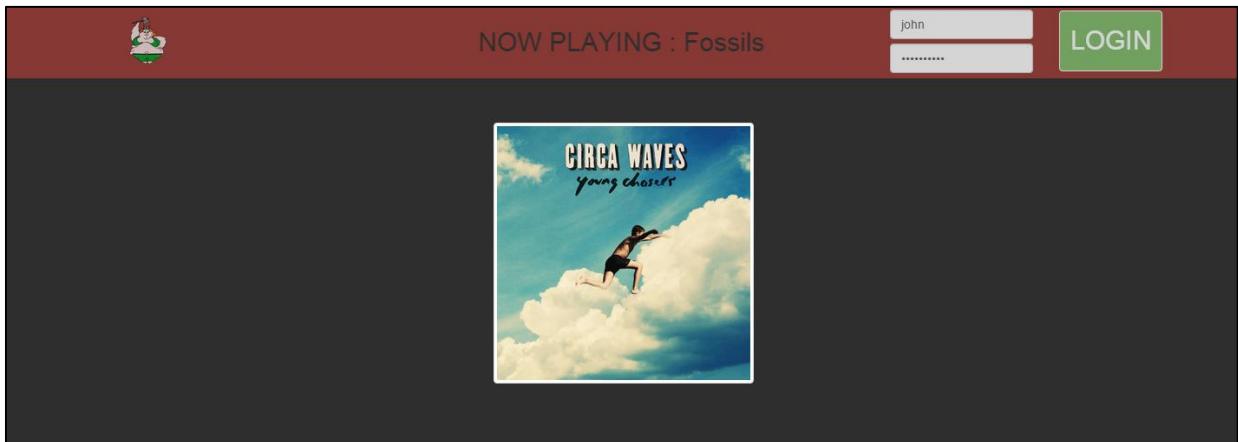
/* for skip button when skipped */
.skipped {
    /* override background colour to grey */
    background-color: #bdbdbd !important;
}
```

JavaScript

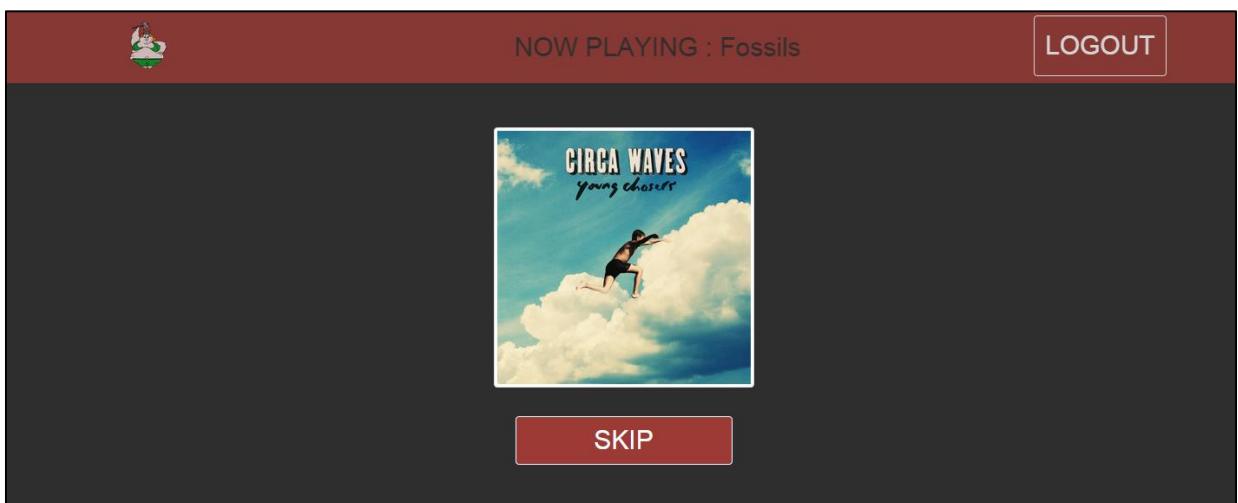
Skip button pressed

```
// if skip button pressed
$('#btnSkip').click(function(){
    // call skip function
    skipReq();
});

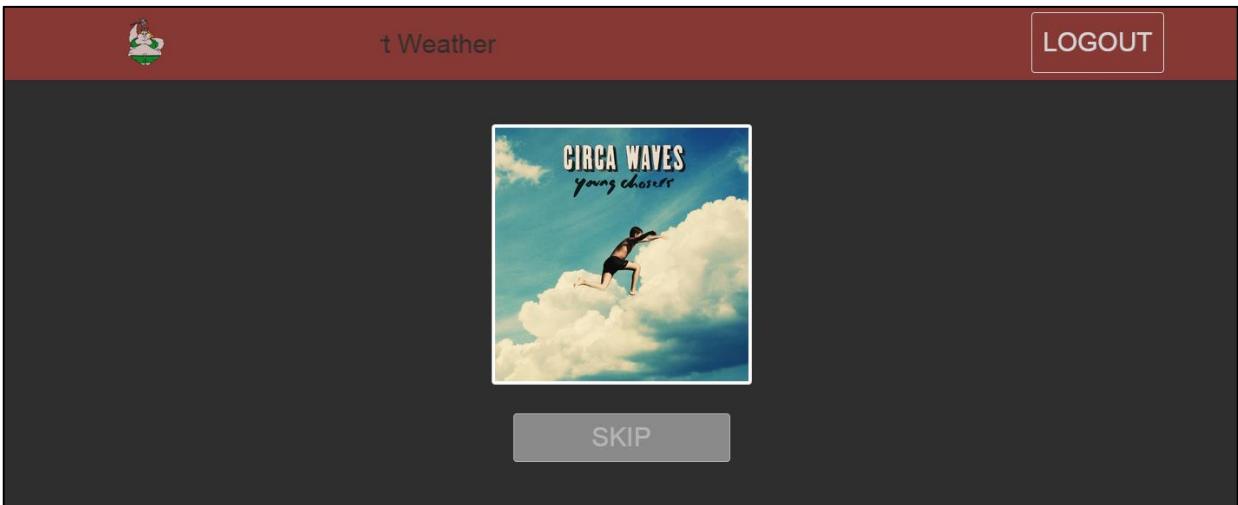
// if skip request successful
// change skip button colour to grey
$("#btnSkip").toggleClass("skipped");
// toggle skip button disabled
$("#btnSkip").prop('disabled', function(i, v) { return !v; });
```



Jumbotron with CSS, user logged out



Jumbotron with CSS, user logged in



Jumbotron with CSS, user skipped song

Requirements Met: i5, d5, d6, d7, o2

Markup 4 (Search)

The module for searching for a song in the design translated perfectly to HTML with the use of bootstrap's support for forms. To prevent requests from being sent while the user is not logged in, I initialised these elements as disabled and invisible with those properties being toggled appropriately when the login is successful. When a focus group tested the working system, they found that they could often not find their chosen song when there was only one textbox. As a result, I split the textbox into two separate inputs for track name and artist in order to allow more specific searches that are more likely to return the correct song.

HTML

```
<!-- contains all elements related to searching and requesting songs -->
<div class="search">
  <div class="container">
    <!-- all contained elements not displayed until user logged in successfully -->
    <!-- main.js temporarily disables elements when song successfully requested -->
    <div class="initSearch">
      <div class="row-fluid">
        <div class="col-sm-5">
          <div class="form-group">
            <!-- create textbox with default text "Track name..." -->
            <!-- main.js gets text entered when songReq function called, calls
            songReq function when enter key pressed while in textbox -->
            <input id="tbSong" type="text" class="toSongReqDisable form-control
            toToggle searchBar" placeholder="Track name..." style="display: none;">
          </div>
        </div>
        <div class="col-sm-5">
          <div class="form-group">
            <!-- create textbox with default text "Artist..." -->
            <!-- main.js performs same logic as to tbSong -->
            <input id="tbArtist" type="text" class="toSongReqDisable form-control
            toToggle searchBar" placeholder="Artist..." style="display: none;">
          </div>
        </div>
        <div class="col-sm-2">
          <!-- create button with text "SEARCH" -->
          <!-- main.js calls songReq function when pressed -->
          <button id="btnSearch" type="button" class="toSongReqDisable btnSearch btn
          btn-default navbar-btn toToggle" style="display: none;">SEARCH</button>
        </div>
      </div>
    </div>
  </div>
</div>
```

CSS

```
/* for div containing all search elements */
.search .container{
    /* size adjustments for correct layout on desktop */
    text-align: center;
    padding-bottom: 10px;
}

/* for track and artist textboxes */
.search input{
    /* size adjustments for correct layout on desktop */
    font-size:20px;
    height:50px;
}

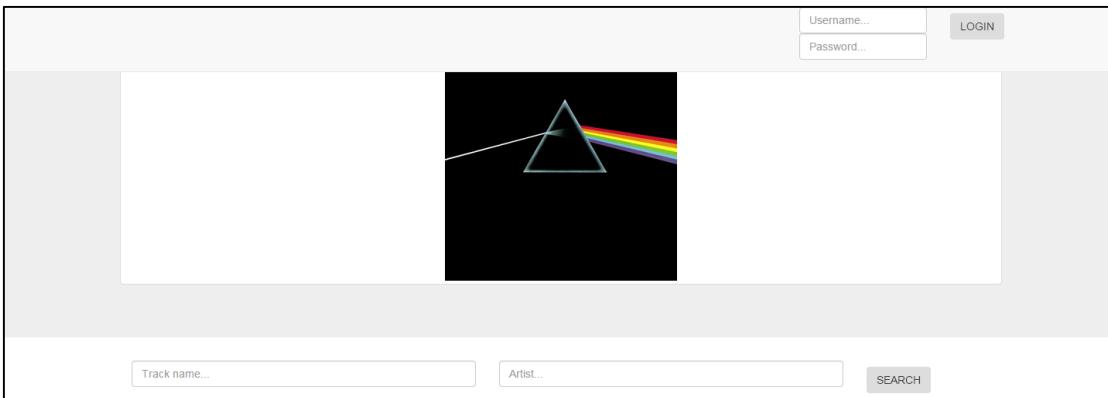
/* for search button */
.btnSearch{
    /* set text to white, background to green */
    color: #FFF;
    background: #83BD6C ;
    /* size adjustments for correct layout on desktop */
    font-size:20px;
    height:50px;
    width:150px;
    margin-top:0px;
}
```

JavaScript

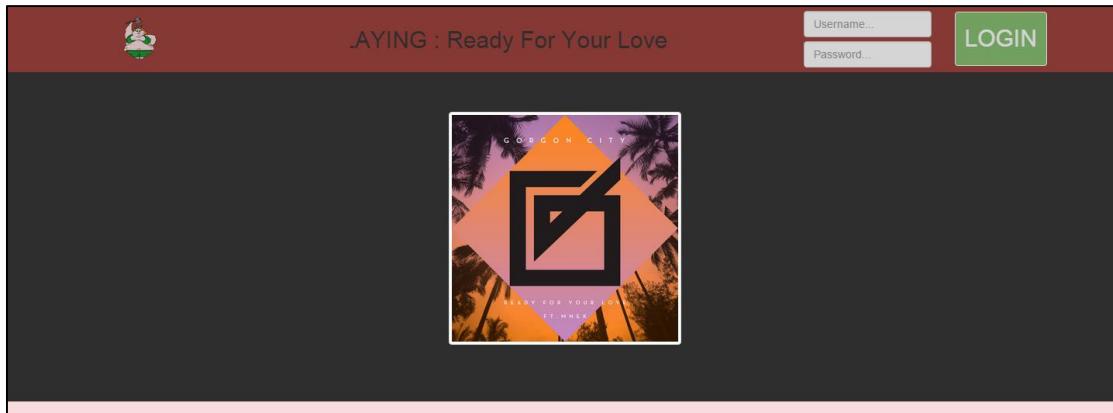
Search button pressed

```
// if search button pressed
$('#btnSearch').click(function(){
    // reset search counter so 1st result is shown
    localStorage.setItem("searchResCount",0)
    // open search results module
    searchOpen();
    // send search request
    songSearch();
});

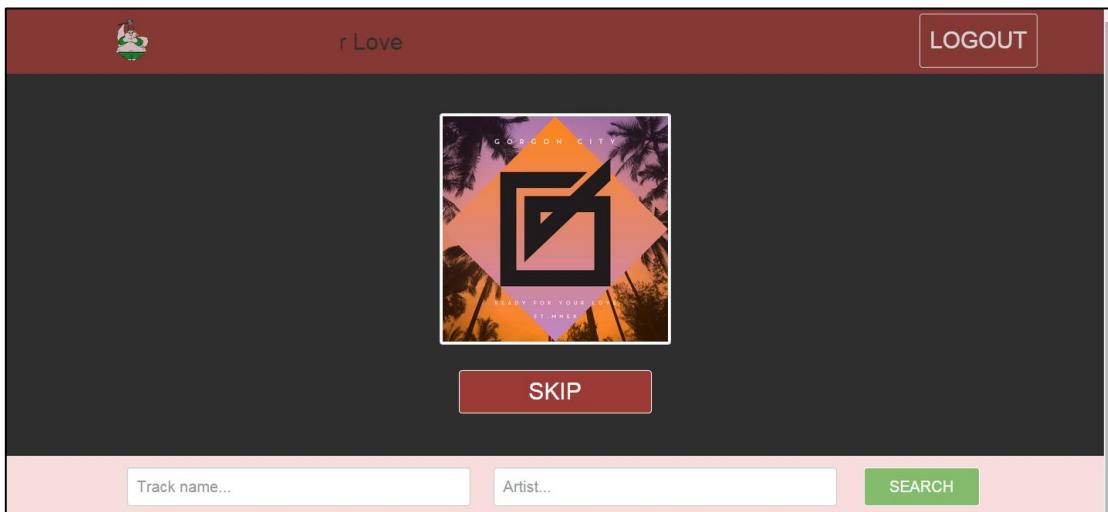
// if enter key pressed in track or artist box
$(".searchBox").keypress(function(e) {
    if(e.which == 13) {
        localStorage.setItem("searchResCount",0)
        searchOpen();
        songSearch();
    }
});
```



Search in plain HTML



Search with CSS, user logged out



Search with CSS, user logged in

Requirements Met: i3, d8, d9, d10

Markup 5 (Search Results)

When the system was designed, there was no way for the user to see the song that they had requested before it appeared in the queue. During testing, it was clear that some sort of verification was needed before the song was added to the queue in order to prevent unwanted songs from being played and users ‘wasting’ their request. As a result, I discussed this problem with my clients and developed a simple interface that appeared when a search was successfully made. It would display the information of the first result returned from Spotify and two buttons to allow the user to input whether the song was correct or not. If the user indicated that the correct song was returned then the song request would be sent and the results module would close. If the user indicated that the song displayed was incorrect then the module would update with the information of the next search result. All elements that will be updated are assigned the class “toUpdate” so that they can be processed as a group easily in JavaScript.

HTML

```
<!-- main.js animates fade in of contained elements and moves next song modules
to fit when searchOpen function called -->
<div class="searchResult" >
  <div class="searchToggle toSearchToggle" style="display: none;">
    <div class="row-fluid">
      <div class="col-sm-10">
        <div class="row-fluid">
          <div class="col-sm-4">
            <!-- create thumbnail to left of page, bootstrap.css handles basic
layout -->
            <div class="thumbnail toUpdate">
              <!-- display cover art of search result song -->
              <!-- main.js updates image when new search query made -->
              
            </div>
          </div>
          <!-- contains information on search result song -->
          <!-- main.js updates text when new search query made -->
          <div class="col-sm-8">
            <div class="trackInfo ">
              <p id="titleResult" class="toUpdate">Title</p>
              <p id="artistResult" class="toUpdate">Artist</p>
            </div>
          </div>
        </div>
      </div>
    </div>
    <!-- contains buttons for selecting whether search result is correct song -
-->
    <!-- main.js disables buttons temporarily when song request successfully
made -->
    <div class="col-sm-2">
      <div class="reqConfBtns">
        <!-- main.js calls songReq function when pressed, closes search result
module -->
        <button id="btnYes" type="button" class="toSongReqDisable btn btn-
default navbar-btn reqConfBtn btnYes" disabled>YES</button>
        <!-- main.js calls search function again, gets information of next
search result and updates accordingly -->
        <button id="btnNo" type="button" class="toSongReqDisable btn btn-
default navbar-btn reqConfBtn btnNo" disabled>NO</button>
      </div>
    </div>
  </div>
</div>
```

css

```
/* for div containing all search result elements */
.searchToggle {
    /* set background to match jumbotron background */
    background: #2E2E2E;
    /* round corners */
    border-radius: 10px;
    /* set elements to be slightly transparent*/
    opacity: 0.8;
    /* size adjustments for correct layout on desktop */
    margin-bottom: 15px !important;
    margin-top: 65px;
    height:330px;
}

/* when search results hovered over */
.searchToggle:hover{
    /* disable transparency */
    opacity:1;
}

/* for thumbnail containing search result album art */
.searchResult .thumbnail {
    /* ensure that thumbnail is constant size */
    height: 300px !important;
    width: 300px !important;
    /* size adjustments for correct layout on desktop */
    margin-left:-5px;
    margin-top:15px;
    margin-bottom:10px;
}

/* for search result album art */
.thumbResult{
    /* ensure that image fits in thumbnail */
    height: 290px !important;
    width: 300px !important;
}

/* for text displaying search result information*/
.trackInfo p{
    /* set colour to white */
    color: #fff;
    /* size adjustments for correct layout on desktop */
    font-size: 20px;
    text-align: left;
}

/* for YES and NO buttons */
.reqConfBtn{
    /* set text to white */
    color: #FFF;
    /* size adjustments for correct layout on desktop */
    font-size:60px;
    height:150px;
    width:150px;
}

/* for YES button */
.btnYes{
    /* set colour to green */
    background: #83BD6C ;
}

/* for NO button */
.btnNo{
    /* set colour to red */
    background: #9C3B35;
}
```

JavaScript

Yes button pressed

```
$('.btnYes').click(function() {
    songReq();
    // update page after request sent
    setTimeout(refresh,1000);
    // close search results module
    searchClose();
});
```

No button pressed

The system cycles through the different search results with the use of the variable 'searchResCount' which points to the index in the object returned from Spotify. When the user presses the no button to indicate that the incorrect song was displayed, searchResCount will increment and the songSearch() function will be called again, this time displaying the next song in the returned object. To ensure that the first and most likely to be correct result is displayed when a user requests a new song, searchResCount is reset to 0 when the search button is pressed.

```
$('.btnNo').click(function() {
    // increment search result index and update search
    localStorage.setItem("searchResCount", (parseInt(localStorage.getItem("searchResCount"))+1));
    // search for song again with updated index
    songSearch();
});
```

Open and close search results module

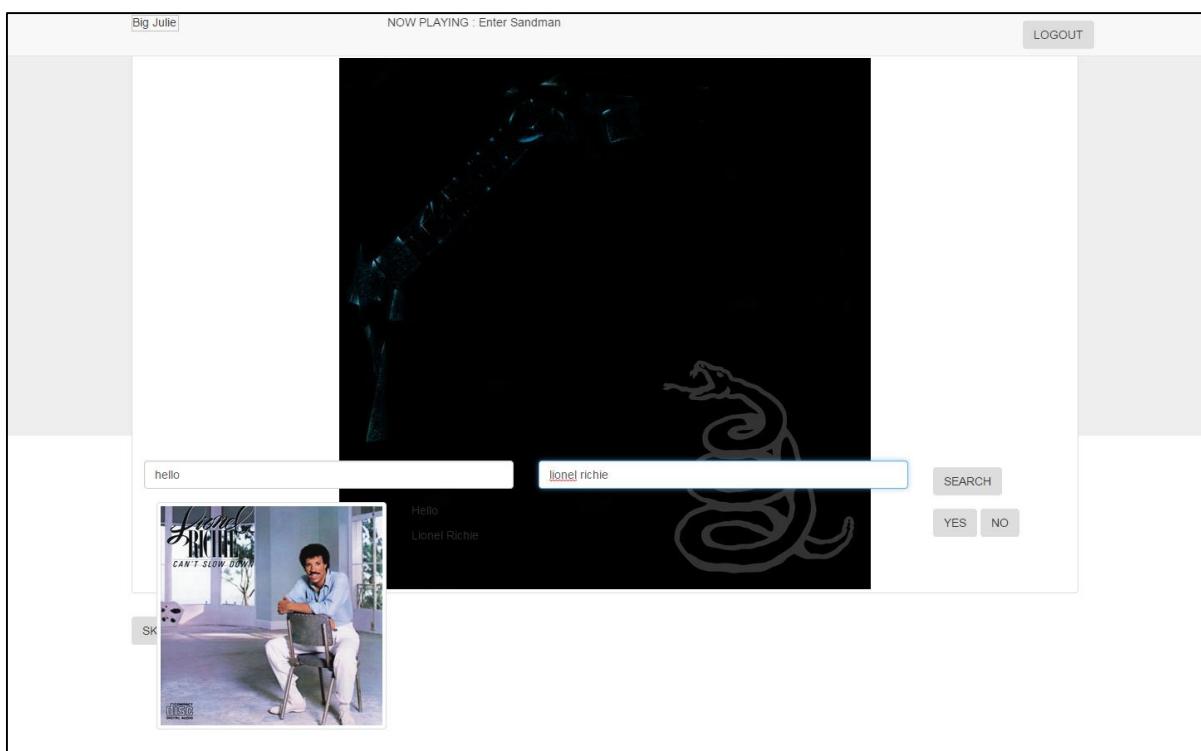
As opposed to the majority of the animations that changed the page layout, the opening and closing of the search result module cannot be written as a toggle-able function. The function was originally written as a toggle but when I tested the page, the module would not open or close at the correct times. For instance, if the user logged out when the search results were not displayed, then the module would open when it should remain closed. When the user pressed the no button, the module would close when it should remain open. To solve this issue, I wrote separate open and close functions that could be called after meeting certain criteria e.g. when the user logs out, call the searchClose() function if the results module is open.

When testing the page with a focus group, they noted that it was annoying when previously entered text remained in the search box after a song was confirmed. When they went to make a second request, they would have to delete the text in the search box and then input their new query. To improve the user experience, I added code to clear the search boxes of all text when the searchClose() function is called.

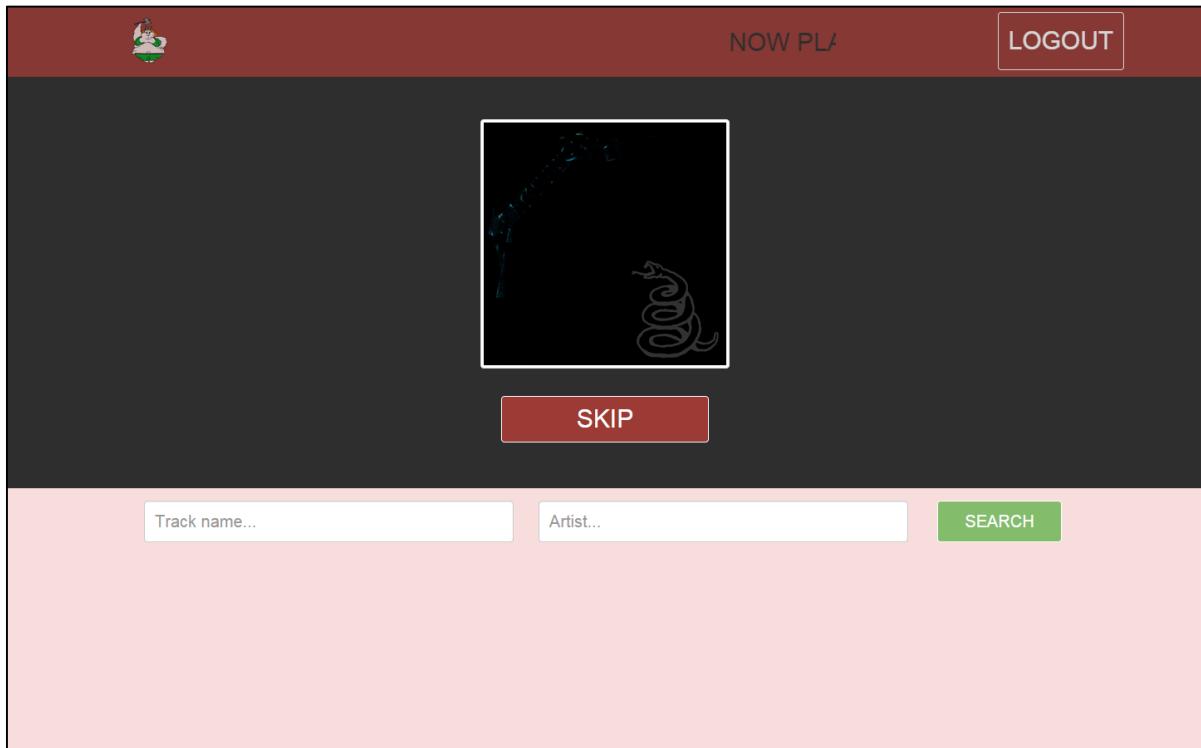
```
function searchOpen(){
    // if search results not already displayed
    if ($('#btnYes').prop('disabled') == true){
        // animate opening of search results
        $(".reqConfBtn").prop('disabled', function(i, v) { return !v; });
        $('.nextSong').animate({top:345},500);
        $(".toSearchToggle").fadeToggle("slow");
    }
}
function searchClose(){
    // empty search boxes
    $('#tbSong').val("");
    $('#tbArtist').val("");
    // animate closing of search results
    $(".reqConfBtn").prop('disabled', function(i, v) { return !v; });
    $('.nextSong').animate({top:0},500);
    $(".toSearchToggle").fadeToggle("slow");
}
```



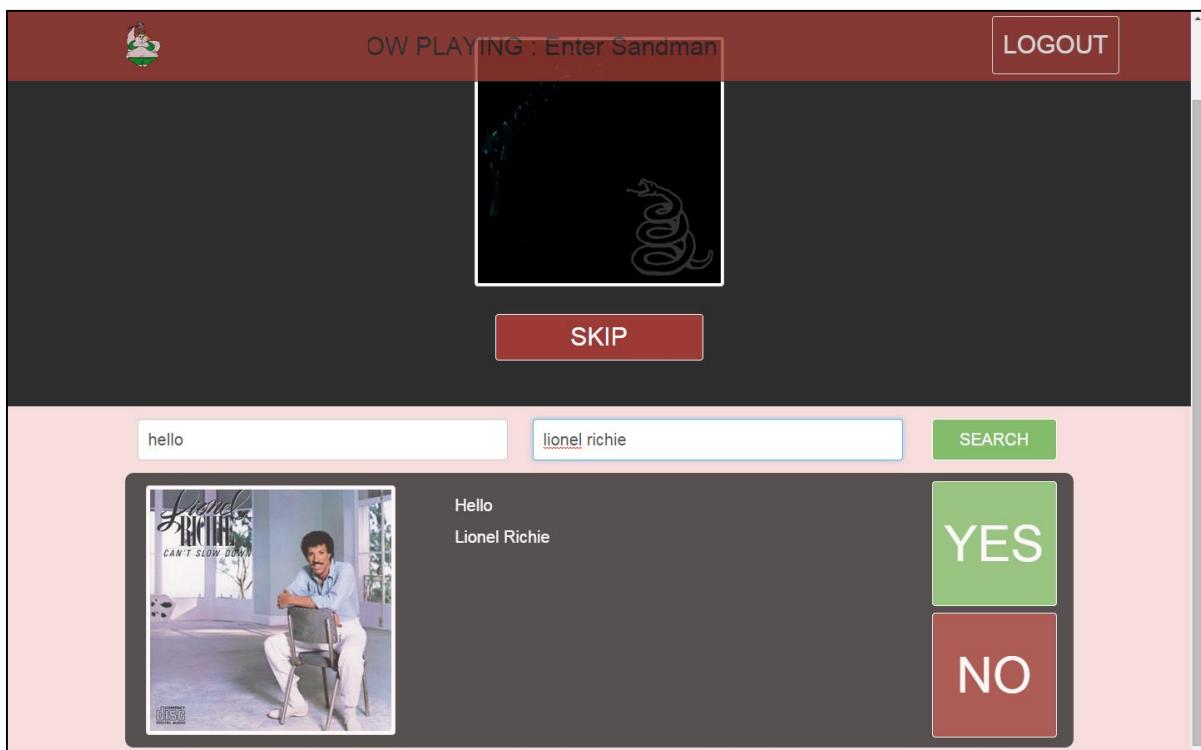
Search results closed in plain HTML



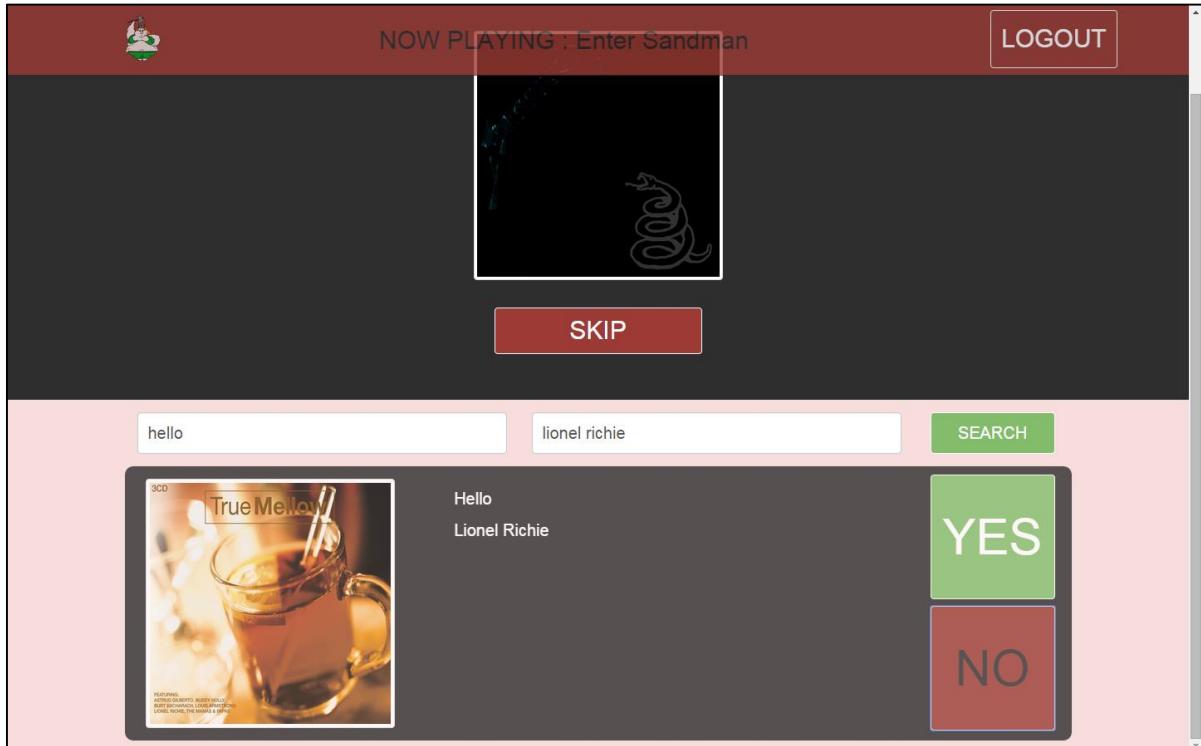
Search results open in plain HTML



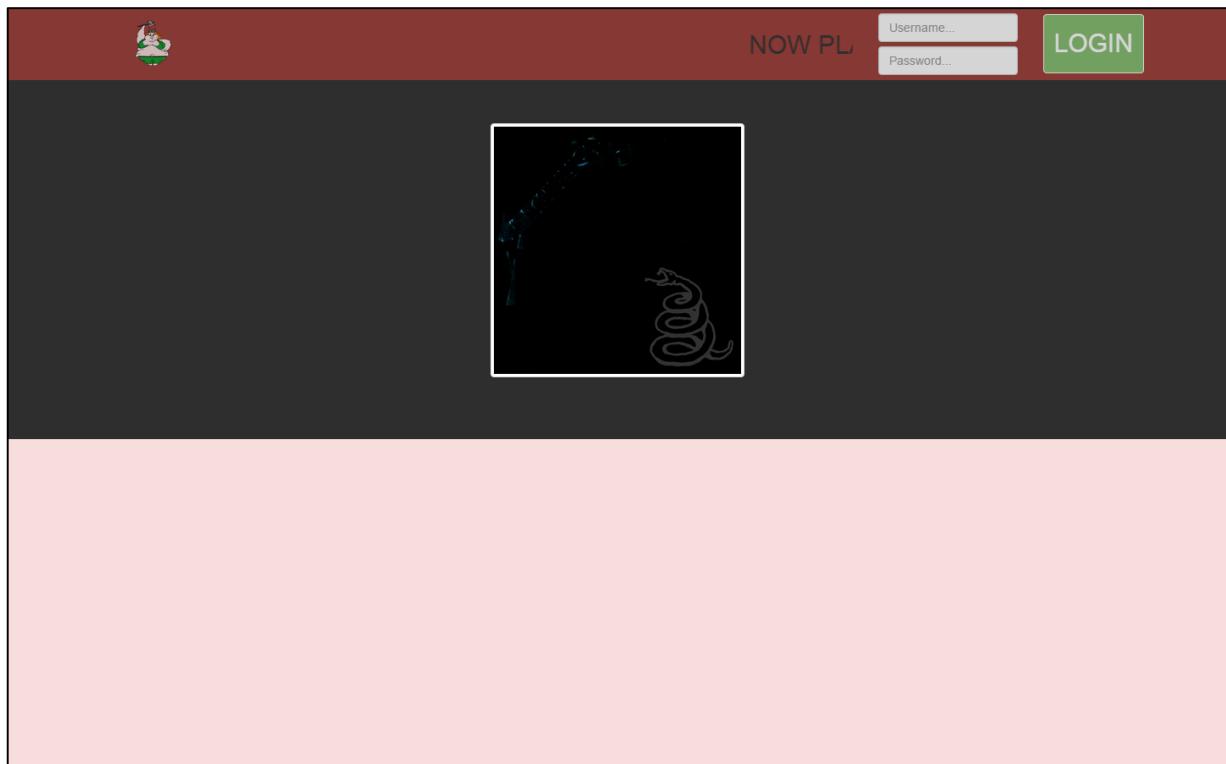
Search results closed, with CSS



Search results open, with CSS



No button pressed, with CSS



User logged out, with CSS

Markup 6 (Next Songs)

The final section of the page is the module containing the next songs to be played with buttons to vote for each song. The overall design of the next song module translated well to HTML but in testing, some changes were required. Since the bootstrap row layout has a width of twelve, it would be impractical to have five next song modules of constant width as five does not evenly divide into twelve. The clients agreed that four modules of width three would give the same functionality with considerably simpler code so the page was changed to that layout. Using different types of column, sm and md, allowed me to create separate layouts for desktop and mobile devices. With this configuration, the modules are changed to display in a grid pattern when viewed on a mobile device. This vertical layout is more appropriate for mobile devices as it matches the orientation of the screen.

HTML

```
<!-- contains all elements related to upcoming songs -->
<div class="nextSongs">
  <div class="container">
    <div class="row-fluid">
      <!-- md column causes modules to display horizontally on desktop, sm column creates grid layout on mobile -->
      <div class="col-sm-6">
        <div class="col-md-6">
          <!-- contains all elements for first next song module -->
          <div class="nextSong">
            <!-- create thumbnail at top of module, bootstrap.css handles basic layout -->
            <div class="thumbnail">
              <!-- display cover art of next song -->
              <!-- main.js updates image when getNextSongs function called if next song has changed -->
              
            </div>
            <!-- display title of next song -->
            <!-- main.js updates text when getNextSong function called if next song has changed, if text will not fit in module, text is truncated to 20 characters and ellipses appended -->
            <p id="pNext1" style="white-space: nowrap; text-overflow: ellipsis;" class="pNext toRefresh"> - </p>
            <span class="btnVote">
              <!-- create button with text "VOTE" -->
              <!-- main.js calls vote function when pressed-->
              <button id="btnVote1" type="button" class="voteBtn locked btn btn-default navbar-btn toToggle" style="display: none;">VOTE</button>
            </span>
          </div>
        </div>
      </div>
    </div>
  </div>
<!-- same markup for other 3 modules -->
```

css

```
/* for div containing next song modules */
.nextSongs{
    /* ensure that mouse hover targets correct area */
    pointer-events: none;
    /* size adjustments for correct layout on desktop */
    margin-left: -10px;
}
/* for each next song module */
.nextSong {
    /* align elements in centre of module */
    text-align: center;
    /* set background to dark grey */
    background: #2E2E2E;
    /* round corners */
    border-radius: 10px;
    /* set module to 50% transparency */
    opacity: 0.5;
    /* ensure that mouse hover targets correct area */
    pointer-events: fill;
    /* ensure that modules align correctly when search result module is open */
    position: relative !important;
    /* size adjustments for correct layout on desktop */
    padding: 5px;
    height: 300px;
    margin-top: -10px;
}
/* when mouse hovers over specific module */
.nextSong:hover {
    /* disable transparency */
    opacity: 1;
}
/* for each thumbnail containing next song cover art */
.nextSong .thumbnail {
    /* ensure that thumbnail is constant size */
    height: 220px !important;
    width: 220px !important;
    /* size adjustments for correct layout on desktop */
    display: inline-block;
    margin-top: 3px;
}
/* for each next song cover art */
.nextSong img {
    /* ensure that image fits in thumbnail */
    height: 210px !important;
    width: 220px !important;
}
/* for each next song title */
.nextSong p {
    /* set colour to white */
    color: #FFF;
    /* size adjustments for correct layout on desktop */
    font-size: 20px;
}
/* for each vote button */
.btnVote button {
    /* set text colour to white, background to green */
    color: #FFF;
    background: #83BD6C;
    /* size adjustments for correct layout on desktop */
    width: 100px;
    font-size: 18px;
    width: 200px;
```

```

}
/* for vote button when corresponding song already voted for */
.voted {
    /* override background colour to grey */
    background-color: #BDBDBD !important;
}
/* for each next song module on desktop */
.col-md-6 {
    /* size adjustments for correct layout on desktop */
    padding-bottom: 20px;
}

```

JavaScript

Move next songs on login and logout

Since the next song modules remain visible on the page at all times, the animations for logging in and out are considerably more complicated as the modules must move in accordance to other modules appearing and disappearing. The function detects whether the client is flagged as mobile or desktop and then toggles the height of the module so that it moves up or down with relation to the jumbotron which is in a fixed position. This gives the effect of the next song modules sliding down while other modules such as search fade in and vice versa.

```

// animate resize for next song module
// if user on mobile
if (localStorage.getItem("mobile") == 'TRUE') {
    // expand module to fit vote button
    if ($.nextSong").height() == 360){
        $('.nextSong').animate({height:450},500);
    }
    // retract module to initial height
    else{
        $('.nextSong').animate({height:370},500);
    }
}
// if user on desktop
else{
    // expand module to fit vote button
    if ($.nextSong").height() == 290){
        $('.nextSong').animate({height:355},500);
    }
    // retract module to initial height
    else{
        $('.nextSong').animate({height:300},500);
    }
}

```

Truncate song name to fit in module

I also added in a small function that ensured that the song name would fit inside its next song module. When a name is greater than 20 characters long, the page displays the first 20 characters followed by ellipses.

```

// if track name will overflow module
if (name.length > 20) {
    // set text to first twenty characters and append ellipses
    name = (name.substr(0,20)+'...');
}

```

NOW PLAYING: I Still Bel

Username... Password...

FRANK TURNER
ENGLAND KEEP MY BONES

SWIM DEEP WHERE THE HEAVEN ARE WE

DILLON FRANCIS DJ SNAKE GET LOW

FRANK TURNER THE WAY I TEND TO BE

wolf blue Moaning Lisa Smile

King City Get Low The Way I Tend To Be Moaning Lisa Smile

Next song in plain HTML, logged out

I Still Believe

FRANK TURNER

Track name... Artist...

SWIM DEEP WHERE THE HEAVEN ARE WE

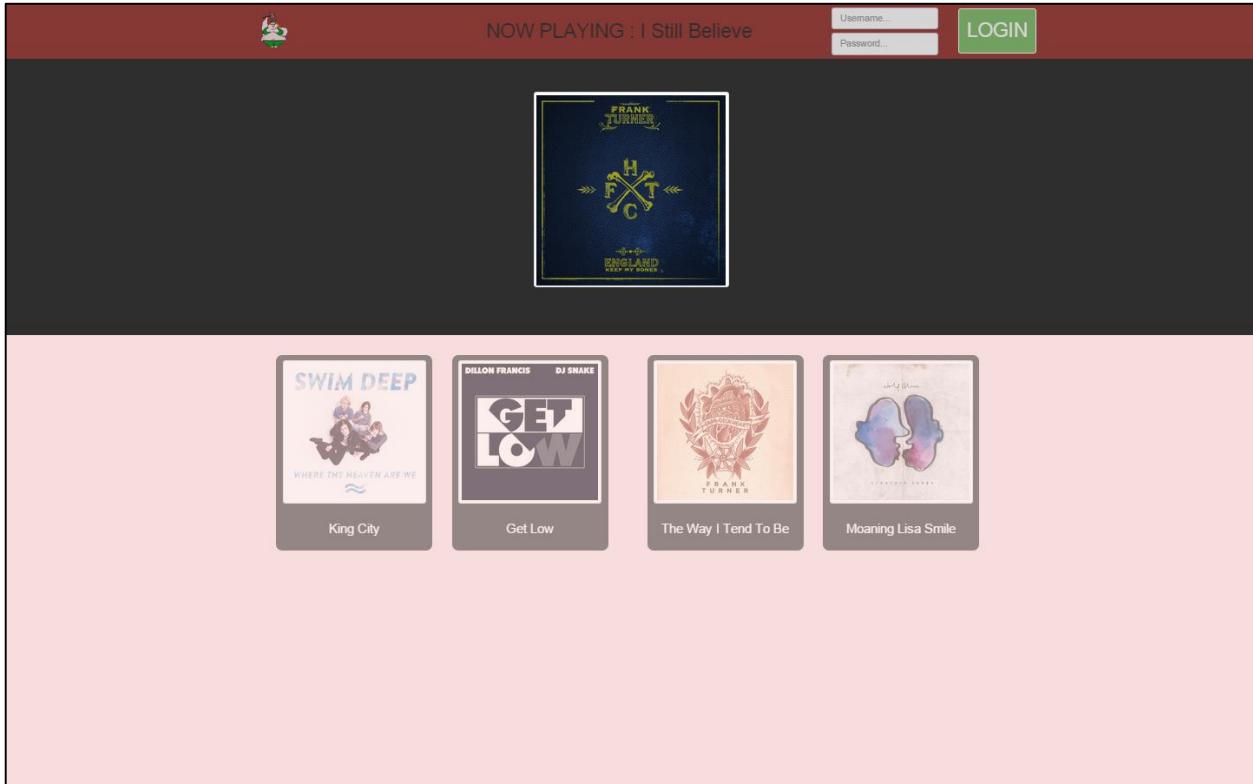
DILLON FRANCIS DJ SNAKE GET LOW

FRANK TURNER THE WAY I TEND TO BE

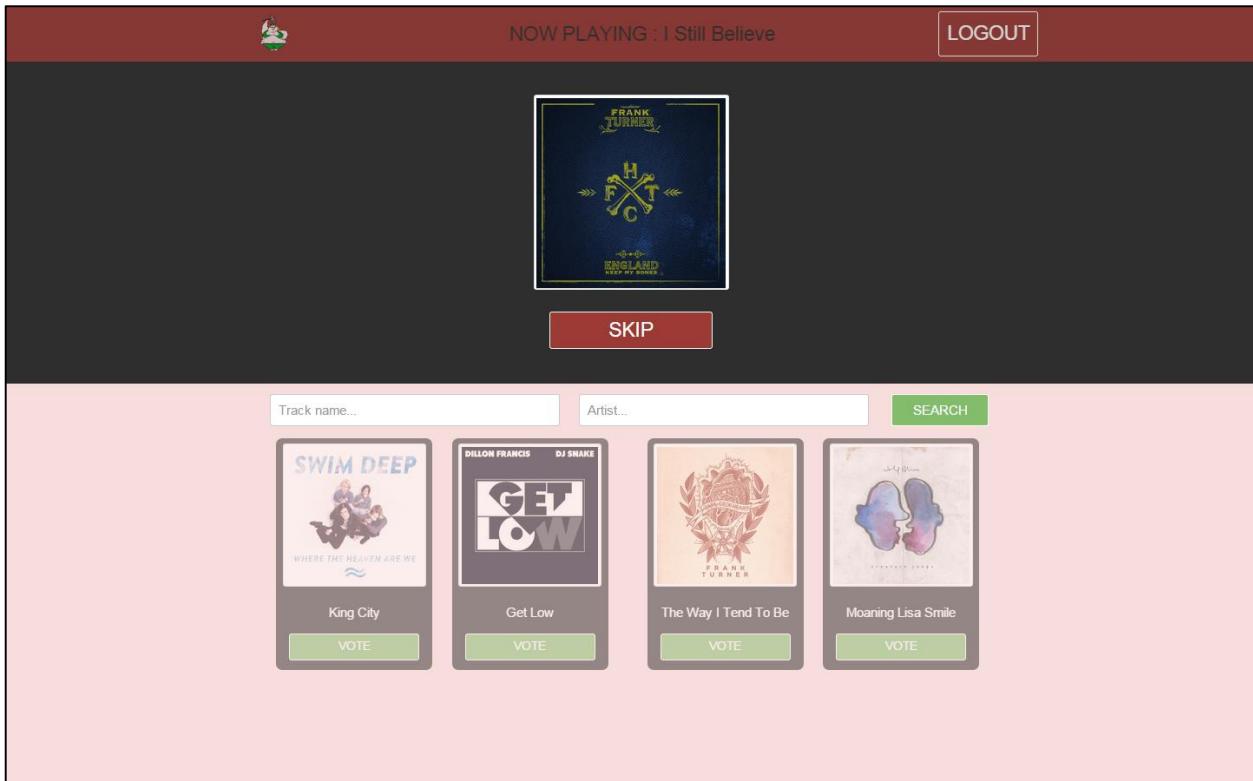
wolf blue Moaning Lisa Smile

King City Get Low The Way I Tend To Be Moaning Lisa Smile

Next Song in plain HTML, logged in



[Next song with CSS, logged out](#)



[Next song with CSS, logged in](#)

Requirements Met: i4, d11, d12, d13, d14, ,o3

Markup 7 (Mobile Device Compatibility)

To ensure that the page was displayed in a suitable format on mobile devices I used a media query in my CSS. The query will use properties such as pixel density and screen orientation to determine if the client is a mobile device and apply additional styles if true.

CSS

```
/* apply following CSS only when media query met, i.e. when page loaded from mobile
device */
@media="only screen and (max-device-width: 480px) , only screen and (-webkit-min-
device-pixel-ratio: 2) , screen and (-webkit-device-pixel-ratio:1.5){
/* override previous layout for correct display on mobile */
.navbar{
  padding-top: 10px;
  height: 95px;
}
.BJ {
  padding-top: 25px;
  font-size: 35px;
}
.nowPlaying {
  font-size:30px;
  height: 80px;
  line-height: 75px;
}
.btnLogin {
  margin-left:-10px;
}
.btnLogin button{
  font-size: 30px;
  width: 150px;
}
.jumbotron {
  padding-top: 135px;
}
.jumbotron .thumbnail {
  width: 300px;
  padding:3px;
}
.btnSkip button{
  font-size: 30px;
  width: 250px;
}
.search .container{
  padding-bottom: 15px;
}
.initSearch{
  margin-left: -50px;
}
.search input{
  font-size:20px;
  height:50px;
}
.searchResult {
  margin-left:-100px;
  margin-right:-100px;
}
.search .thumbnail{
  padding:3px;
}
.trackInfo{
```

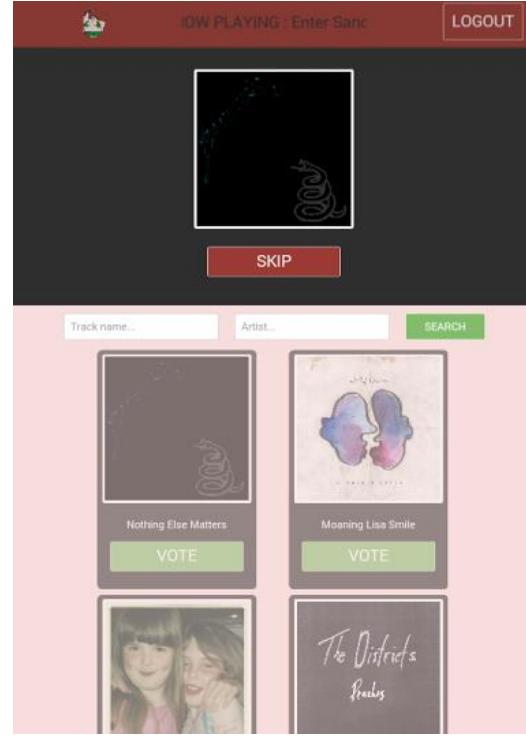
```

margin-top: 25px;
margin-left: 100px;
}
.reqConfBtns{
margin-left: -40px;
}
.nextSongs{
margin-left:-5px;
}
.nextSong {
padding:5px;
border-radius: 10px;
height: 370px;
}
.nextSong .thumbnail {
height: 280px !important;
width: 280px !important;
}
.nextSong img {
height: 270px !important;
width: 280px !important;
}
.nextSong p {
font-size: 20px;
}
.btnVote button{
font-size: 30px;
width: 250px;
}
.col-md-6 {
padding-bottom: 20px;
}
.logged {
background: #9C3B35 !important;
margin-left: 0px;
}

```



Page on mobile without media query



Page on mobile with media query

Requirements Met: i1, o5

Process 1 (Login)

The login functionality was the first system logic to be implemented since all requests require the user to be logged in before they are processed.

Database creation

Before I could begin writing the code for the login function, I needed to set up the database environment that the system required. I used the data dictionary and database diagram from the design to create the request, user and vote tables in phpMyAdmin, the natively supported database package on my hosting service.

I then populated the user table with the usernames and passwords that the login function needed. I obtained a list of usernames for all students in my year and inserted them into the user table, creating a new record for each pupil in the year. I used Visual Basic .NET to generate a list of unique passwords before assigning each user a password from the file using MySQL.

Table	Action	Rows	Type	Collation	Size
Request	Browse Structure Search Insert Empty Drop	1	MyISAM	utf8_general_ci	3.1 KiB
User	Browse Structure Search Insert Empty Drop	1	MyISAM	utf8_general_ci	2 KiB
Vote	Browse Structure Search Insert Empty Drop	1	MyISAM	utf8_general_ci	4 KiB
3 tables		Sum			

Creating the request, user and vote tables in phpMyAdmin

#	Name	Type	Collation	Attributes	Null	Default
1	username	varchar(30)	utf8_general_ci		No	
2	password	varchar(15)	utf8_general_ci		Yes	NULL
3	loggedin	tinyint(1)			Yes	NULL
4	active	int(1)			Yes	NULL
5	lastReqTime	bigint(15)			Yes	NULL
6	voteWeight	decimal(3,2)			Yes	NULL
7	skipped	tinyint(1)			Yes	NULL

Creating each field within the user table in accordance to the design

username	password	loggedin	active	lastReqTime	voteWeight	skipped
08apodraza	GABLE607702DENN	0	0	0	1.00	0
08aomoigui	SEAMCHICAS1081	0	0	0	1.00	0
08amarlow	612FRORENRAPED	0	0	0	1.00	0
08amaher	655117DINKUM	0	0	0	1.00	0
08ajohnson	652751BIUKS	0	0	0	1.00	0
08bbritton	06IMIDICUMBERY	0	0	0	1.00	0
08bcook	SNAWS212	0	0	0	1.00	0
08bdickenson	BATATA45592WINS	0	0	0	1.00	0
08bholland	23DEBUS	0	0	0	1.00	0
08bmgrail	GLOBEDKUDZUSMEA	0	0	0	1.00	0
08brobinson	RECANT305KNOBBY	0	0	0	1.00	0
08btreadwell	6860344339960	0	0	0	1.00	0
08ccaines	ERASPREGGY110	0	0	0	1.00	0
08cclay	8388OBEAHS936	0	0	0	1.00	0
08cdunne	556698717	0	0	0	1.00	0
08cfee	WOALD4031941532	0	0	0	1.00	0
08cfurnell	704509DOSTANVIL	0	0	0	1.00	0
08chanrahan	7810225ACCUSE	0	0	0	1.00	0
08cliggins	2773	0	0	0	1.00	0
08cnash	MYXOID627034469	0	0	0	1.00	0
08cperry	93995435225	0	0	0	1.00	0
08cpreece	44235427205MAYO	0	0	0	1.00	0
08cruane	WAREZ31426OBTEC	0	0	0	1.00	0
08cstewardson	41BEMADS46	0	0	0	1.00	0
08czako	SCENAS91DAULT	0	0	0	1.00	0

The user table populated with username, password and other field values defaulted

Visual Basic .NET

Generate unique passwords for each user

```
Imports System.IO
Module Module1
    Sub Main()
        ' Load password seed list into bank variable
        Dim reader As StreamReader = New
StreamReader("N:\Documents\Dev\P1\PassGen\JuliePassGen\passSeed.txt")
        Dim bank As String = reader.ReadLine
        ' Split each word in wordbank into separate indices in wordbank()
        Dim wordbank() As String = bank.Split(",")
        Dim word(wordbank.Length - 1) As String
        reader.Close()
        ' Open output file to write passwords to
        Dim writer As StreamWriter = New
StreamWriter("N:\Documents\Dev\P1\PassGen\JuliePassGen\passwords.txt")
        ' Assign n with number specified by user (number of passwords required)
        Dim n As Integer = Console.ReadLine
        ' Generate new random seed
        Randomize()
        ' Repeat for number of passwords requested
        For i = 0 To n - 1
            ' Assign currently generating password as empty string
            word(i) = ""
            ' Repeat 3 times
            For j = 0 To 2
                ' Randomly generate value of 1 or 0
                ' If value is 1
                If CInt(Math.Floor((2) * Rnd())) = 1 Then
                    ' Select random word from wordbank() and append to password
                    word(i) = word(i) & wordbank(CInt(Math.Floor((wordbank.Length + 1) * Rnd())))
                Else
                    ' Generate random integer from between 0 and 5 as k
                    ' Repeat k times
                    For k = 0 To CInt(Math.Floor((6) * Rnd()))
                        ' Append random digit to password
                        word(i) = word(i) & (CInt(Math.Floor((10) * Rnd())))
                    Next
                End If
            Next
            ' write new password to output file
            writer.WriteLine(word(i))
        Next
        ' close output file
        writer.Close()
    End Sub
End Module
```

JavaScript

On the client-side, the login function was fairly simple and followed the designed logic almost exactly. When called, jQuery assigns username and password variables with the corresponding values of their input textboxes. It then checks whether the user is locally flagged as logged in and sends either a login or logout request to the server depending on that evaluation.

All requests to the server are sent in accordance with AJAX, a convention that allows for PHP to operate asynchronously. Using AJAX, the PHP functions can be called repeatedly on a single page without the need for refreshing, meaning that the website can update dynamically and perform other functions while the requests are being processed.

If the server returns that the request was successful then the user is locally flagged as logged in and the toggleLogged() function is called. This function changes the website between logged in and logged out mode with the animations shown previously in the document. If the request is unsuccessful then the page will display an alert box with the error message returned by the server.

```
function login() {
    // get login values from textboxes
    var username = $('#username').val();
    var password = $('#password').val();
    // if user not logged in
    if (localStorage.getItem("loggedIn") == 'FALSE') {
        // send login request to login.php with username and password
        $.ajax({
            type: "POST",
            url: "login.php",
            dataType: "text",
            data: {action:"login", PHPusername: username, PHPpassword: password},
            // return if login was successful
            success: function (loginValid) {
                // if login successful
                if (loginValid == 1) {
                    // update page to logged in mode
                    toggleLogged();
                    // set user as logged in
                    localStorage.setItem("loggedIn", "TRUE");
                    // save username in localstorage for use in requests
                    localStorage.setItem("username", username);
                }
                // if login unsuccessful
                else {
                    // output login failed message
                    alert("Login unsuccessful, check username and password are correct");
                }
            }
        });
    }
    // if user logged in
    else {
        // send logout request to login.php with username
        $.ajax({
            type: "POST",
            url: "login.php",
            dataType: "text",
            data: {action:"logout", PHPusername: username},
        });
        // update page to logged out mode
    }
}
```

```
toggleLogged();
// set user as logged out
localStorage.setItem("loggedIn", "FALSE");
}

}

function toggleLogged(){
// if search box is open then close
if ($("#btnYes").prop('disabled') == false){
searchClose();
}
// toggle vote buttons and search box visible
if ($.nextSong").height() == 360 || $(".nextSong").height() == 290{
$(".toToggle").delay(200).fadeToggle("slow");
}
else{
$(".toToggle").delay(0).fadeToggle("slow");
}
// animate resize for next song module
if (localStorage.getItem("mobile") == 'TRUE') {
// expand module to fit vote button
if ($.nextSong").height() == 360{
$('.nextSong').animate({height:450},500);
}
// retract module to initial height
else{
$('.nextSong').animate({height:370},500);
}
}
// if user on desktop
else{
if ($.nextSong").height() == 290{
$('.nextSong').animate({height:355},500);
}
else{
$('.nextSong').animate({height:300},500);
}
}
// animate resize for jumbotron and search box
// expand to fit search box and skip button
if ($.jumbotron").height() == 325{
$('.jumbotron').animate({height:579},500);
$('.search .container').animate({height:79},500);
}
// retract to initial height
else{
$('.jumbotron').delay(100).animate({height:508},500);
$('.search .container').delay(100).animate({height:25},500);
}
// toggle button between login and logout
$("#btnLogin").fadeToggle("slow",function(){
// toggle button text and CSS
if ($("#btnLogin").text() == "LOGIN") {
$("#btnLogin").text("LOGOUT");
}
else {
$("#btnLogin").text("LOGIN");
};
$("#btnLogin").toggleClass("logged");
});
$("#btnLogin").fadeToggle("slow");
```

PHP

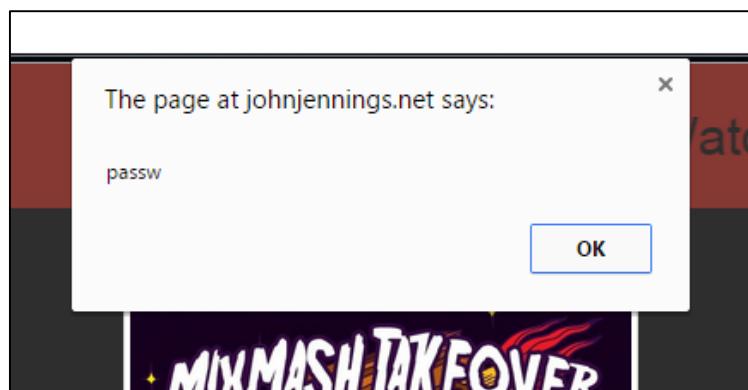
The server-side logic of the entire system consists mostly of manipulating the database using SQL. Before any logic for logging in could be developed, I had to be able to connect to my database, send in SQL queries and process the results accordingly.

This was achieved with the connect() and query() functions. The connect() function is called at the start of each request and uses my login credentials to establish a connection to the database for the duration of that request. The query() function acts as a wrapper for a generic MySQL query. It executes the query and processes the resulting object, ultimately returning the raw values. By writing this generic function I greatly simplified the server-side coding in the long run. Instead of having to use multiple PHP functions and JSON objects each time I needed a return value from a query, I only had to call the query() function and specify what data I wanted to be returned in the returnAs parameter.

Main.php

```
// establishes connection with mySQL server
function connect() {
    $con = mysql_connect('censored', censored, 'censored');
    mysql_select_db('Big_Julie', $con);
}
// generalised mySQL query function, unwraps data from mySQL array before returning
function query($query, $returnAs) {
    return (mysql_fetch_array(mysql_query($query)) [$returnAs]);
}
```

I tested that the queries were being correctly executed by uploading a temporary file to the website that would send basic queries and output the resulting values. I could then check that these results matched the expected results and if they did not, I would place alerts throughout the problematic code in order to trace how the variables changed with each new line. This allowed me to pinpoint the area of code that was causing the error and make appropriate changes. I used this approach throughout the development of the server-side logic as a precursor to formal testing.



An alert containing the results of a test query, indicating that the SQL query was executed correctly

With the server interfacing correctly with my database, I could then complete the login functionality. I used the logic and queries designed in the pseudocode, testing each query with alerts to ensure that they were working correctly before adding them to the code.

When login.php is executed from an AJAX call, the ‘action’ parameter specified in the call is used in a switch statement to identify whether a login or logout request has been received. In the case of a login request, the loginValid() function is called which queries the database with the username and password received in the request and determines whether there is a matching record. If the username and password are correct then the toggleLogged() function is called to update the user’s loggedIn value as 1. The server then returns whether the user has logged in successfully as a Boolean value. If a logout request is sent then the toggleLogged() function is called again with a loggedIn parameter of 0 so that the user is flagged as logged out.

Login.php

```
<?php
include 'main.php';
// return whether username and password match in user database
function loginValid($username,$password) {
    return (query('SELECT COUNT(password) AS loginValid FROM User WHERE username =
\'' .mysql_real_escape_string($username).'\' AND password =
\'' .mysql_real_escape_string($password).'\'', 'loginValid'));
}
// set value of loggedIn in user database
function toggleLogged($username,$loggedIn) {
    mysql_query('UPDATE User SET loggedIn = \'' .($loggedIn). '\' WHERE username =
\'' .mysql_real_escape_string($username). '\'');
}
// use 'action' from login.js to branch to different logic
switch ($_POST['action']) {
    // login request
    case "login":
        // establish connection with mySQL
        connect();
        // validate login details
        $loginvalid = loginValid($_POST['PHPusername'],$_POST['PHPpassword']);
        // if validation passed
        if( $loginvalid == 1){
            // flag user as logged in and active
            toggleLogged($_POST['PHPusername'],1);
            updateActive($_POST['PHPusername'],10);
        }
        // return if user login is valid to login.js
        echo $loginvalid;
        break;
    // logout request
    case "logout":
        connect();
        // flag user as logged in
        toggleLogged($_POST['PHPusername'],0);
        break;
}
?>
```

Requirements Met: i6, i7, p1

Test Number	Process Tested	Requirement Tested	Type	Data Entered / Action Performed	Expected Result	Actual Result	Test Passed?	Corrective Action
1	P1 (Login)	Check that valid user credentials lead to login (p1)	Normal	Username: 08cleggins Password: 2773	Website changes to logged in mode. User flagged as logged in in database	Website changes to logged in mode. User flagged as logged in in database	Yes	None
				Username: 08amaher Password: 655117DINKUM				
				Username: 08cpreece Password: 44235427205MAYO				
2	P1 (Login)	Check that invalid passwords for valid usernames are rejected.	Invalid	Username: 08cleggins Password: invalid	Alert stating that login was unsuccessful. No change in database	Alert stating that login was unsuccessful. No change in database	Yes	None
				Username: 08amaher Password: INVALID				
				Username: 08cpreece Password: 2345345346				
3	P1 (Login)	Check that invalid usernames are rejected	Invalid	Username: Invalid Password: Invalid	Alert stating that login was unsuccessful. No change in database	Alert stating that login was unsuccessful. No change in database	Yes	None
				Username: 45645766 Password: 123123				
				Username: INVALID Password: qwerty				
4	P1 (Login)	Check that empty username and password fields are rejected	Invalid	Username empty Password empty	Alert stating that login was unsuccessful. No change in database	Alert stating that login was unsuccessful. No change in database	Yes	None
				Username empty Password: test				
				Username: test Password empty				

5	P1 (Login)	Check that the shortest username and password are accepted if valid	Borderline	Username: 08sclay Password:LOTIONICEMEN664	Website changes to logged in mode. User flagged as logged in in database	Website changes to logged in mode. User flagged as logged in in database	Yes	None
				Username: 08troberts Password: 2049				
6		Check that the longest username and password are accepted if valid	Borderline	Username: 08kbroadhurst Password: 985404	Website changes to logged in mode. User flagged as logged in in database	Website changes to logged in mode. User flagged as logged in in database	Yes	None

All tests passed so development of next process can begin.

Process 2 (Request)

Once the user could log in, the next step in the development was allowing users to send requests to the system.

JavaScript

In my planning, I designed process 2 to act as a parser, deconstructing a generic request into its component parts, identifying the request type and calling the appropriate functions accordingly. However, the event driven nature of jQuery meant that this functionality could be achieved in a much more elegant manner with the use of button press events. Instead of sending each request through a filter to determine the request type, the vote, skip and song buttons could call their respective functions from their own separate event handlers.

Example of event handlers for different request types

```
// if search button pressed
$('#btnSearch').click(function() {
    // reset search counter so 1st result is shown
    localStorage.setItem("searchResCount",0)
    // open search results module
    searchOpen();
    // send search request
    songSearch();
});

// if enter key pressed in track or artist box
$(".searchBox").keypress(function(e) {
    if(e.which == 13) {
        localStorage.setItem("searchResCount",0)
        searchOpen();
        songSearch();
    }
});
$('.btnYes').click(function() {
    songReq();
    // update page after request sent
    setTimeout(refresh,1000);
    // close search results
    searchClose();
});
$('.btnNo').click(function() {
    // increment search result index and update search result
    localStorage.setItem("searchResCount", (parseInt(localStorage.getItem("searchResCount")) + 1));
    songSearch();
});
// if skip button pressed
$('#btnSkip').click(function() {
    skipReq();
});
// if vote button pressed
$('.btnVote').click(function() {
    // get position of song voted for as offset from left side
    var offset = $(this).data('offset');
    voteReq(offset);
});
```

PHP

On the server-side however, some operations still needed to be performed on receipt of any request regardless of type. These functions were added to main.php and called within each request when needed.

At the start of each request, the system must check that the user is logged in and will either continue processing the request or return an error message depending on the result of that check. This was easily achievable with a simple SQL query of “SELECT loggedIn FROM User WHERE username = x”.

To accurately keep track of the number of active users on the system at that time, the user would be flagged as active for the next three songs whenever the system received a request from them. When testing the system, my focus group stated that they should be given a longer period of activity after a request because although they were not interacting with the website, they were still listening but being treated as if they were no longer present. As a result, on receipt of any request, a query of “UPDATE User SET active= 10 WHERE username = x” is executed with the active values being decremented by 1 at the end of every song.

Main.php

```
// update active counter of specific user
function updateActive($username,$active){
    mysql_query('UPDATE User SET active = '.$active.' WHERE username =
\''.mysql_real_escape_string($username).'\'');
}
// return loggedIn flag of specific user
function userLogged($username){
    return(query('SELECT loggedIn FROM User WHERE username =
\''.mysql_real_escape_string($username).'\',\'loggedIn\'));
}
```

The design of the page should prevent logged out users from sending requests but I should ensure that they are not processed regardless

Requirements Met: p2

Test Number	Process Tested	Requirement Tested	Type	Data Entered / Action Performed	Expected Result	Actual Result	Test Passed?	Corrective Action
7	P2 (Request)	Check that requests are accepted for users that are logged in (p2)	Normal	Username: 08troberts Request: Skip	userLogged() in PHP returns true	userLogged() in PHP returns true	Yes	None
				Username: 08troberts Request: Vote				
				Username: test1 Request: Song				
8	P2 (Request)	Check that requests are rejected for users that are not logged in	Normal	Username: 08tmacklin Request: Vote	Alert stating that login is required	Alert stating that login is required	No	Added if statement to check that user is logged in. Continue if true else output login required
				Username: 08tmacklin Request: Skip		Alert stating that login is required	Yes	None
				Username: 08vadcock Request: Song				
9	P2 (Request)	Check that users are flagged as active when request is accepted.	Normal	Username: 08troberts Request: Skip	Active value in user's record is set to 10 in database	Active value in user's record is set to 10 in database	Yes	None
				Username: 08troberts Request: Vote				
				Username: test1 Request: Song				

All unsuccessful tests have been corrected so development of next process can begin.

Process 3 (Song)

The next stage was allowing the user to add their choice of songs to the database. Earlier in the development, it was decided that the user should be able to confirm that the song found by the system is correct. This meant that the song request process would be split into two parts, searching for a song and pushing the user's selected track to the queue.

JavaScript

Search

The songSearch() function is called when the user clicks the search button. It assigns track and artist variables with the values of their corresponding text boxes and constructs a URL in the correct format for interfacing with the Spotify API. I tested that the URLs were constructed correctly by pasting them into a web browser and checking that the returned JSON object matched the results of searching for the same song in the Spotify client. During testing, it was noted that searching for a track where the artist parameter is set to an empty string returned less results than if the parameter was not included. As a result, a set of if statements during the URL construction were added to ensure that a search parameter is only included if its text box is not empty.

```
function songSearch(){
    // get search result index
    var i = (localStorage.getItem("searchResCount"));
    // get track and artist to search, convert string to valid format for spotify
    var searchTrack = ($('#tbSong').val()).replace(/\ /g,"%20");
    var searchArtist = ($('#tbArtist').val()).replace(/\ /g,"%20");
    // construct query depending on which data was sent by user
    var query = "https://api.spotify.com/v1/search?q=";
    // if track and artist specified
    if (searchTrack != "" && searchArtist != ""){
        // construct request with track and artist
        query = query + "track:" + searchTrack + "+artist:" + searchArtist;
    }
    // if only track specified
    else if (searchTrack != ""){
        // construct request with only track
        query = query + "track:" + searchTrack;
    }
    // if only artist specified
    else if (searchArtist != ""){
        // construct request with only artist
        query = query + "artist:" + searchArtist;
    }
    query = query + "&type=track&market=GB";
```

I then added the URL to an AJAX GET request which would return the search results to my JavaScript as a JSON object. The validateTrack() function would then be called which would iterate through each track in the object and construct a list containing only indices for tracks that were below 7 minutes long and not flagged as explicit. JQuery could then strip the required data from the first valid track object and assign it to the matching HTML elements in the search results module. If the user signalled that the displayed song was incorrect by pressing the no button then the process would be repeated again with the second valid track object and so forth.

```
$ .ajax(
  {
    type: "GET",
    url: query,
    success: function (trackObject) {
      var validTracks = [];
      // get array of valid indices for trackObject
      validTracks = validateTrack(trackObject);
      var j = validTracks[i];
      $(".toUpdate").fadeToggle("1200",function(){
        // if object with index j present in trackObject
        if (typeof(trackObject.tracks.items[j]) != "undefined"){
          // assign web elements with corresponding values from trackObject

      $('#thumbResult').attr("src",trackObject.tracks.items[j].album.images[0].url);
      $('#titleResult').html(trackObject.tracks.items[j].name);
      $('#artistResult').html(trackObject.tracks.items[j].artists[0].name);

      $('#lengthResult').html(parseInt(trackObject.tracks.items[j].duration_ms));
      $('#idResult').html(trackObject.tracks.items[j].id);
    }
  });
  // update display on page
  $(".toUpdate").fadeToggle("1200");
}

});

function validateTrack(trackObject){
  var returnTracks = [];
  var i = 0;
  // iterate through each item in trackObject
  while (typeof(trackObject.tracks.items[i]) != "undefined"){
    // if song is under 7 minutes long and not explicit
    if (((trackObject.tracks.items[i].duration_ms) < 420000) &&
    ((trackObject.tracks.items[i].explicit) == false))){
      // push corresponding index to returnTracks
      returnTracks.push(i);
    }
    i = i + 1;
  }
  // return array of indices for valid songs in trackObject
  return returnTracks;
}
```

Push to queue

If the user indicates that the song displayed is correct by clicking the yes button then the songReq() function will be called. This function gets the username and current time and passes it to the server so that the song request can be validated. If the request is valid then the song request button will be disabled for a duration specified in the data returned by the server and the pushSongReq() function is called. This function gathers the track information from the search results module and sends it to the server to be added to the database.

```
function songReq(){
    // get request data from html
    var username = (localStorage.getItem("username"));
    var currentTime = getCurrentTime();
    var id = $('#idResult').text();
    // send request data to songReq.php for validation
    $.ajax(
        {
            type: "POST",
            url: "songReq.php",
            dataType: "text",
            data: {action:"validateSongReq", PHPusername: username, PHPcurrentTime:
currentTime, PHPURL: id},
            success: function (songReqValid) {
                // if request has passed all validation
                if ($.trim(songReqValid).charAt(0) == "p"){
                    // get cooldown time from returned data
                    var cooldown = parseInt($.trim(songReqValid).substring(1));
                    // disable song request button and enable after cooldown
                    toggleSongReqDisabled();
                    setTimeout(toggleSongReqDisabled,cooldown);
                    // send song request
                    pushSongReq(id,username);
                }
                // if request has failed validation
                else {
                    // display error message
                    alert(songReqValid);
                }
            }
        });
}

function pushSongReq(id,username){
    // get request data from html
    var name = $('#titleResult').text();
    var artist = $('#artistResult').text();
    var art = $('#thumbResult').attr("src");
    var length = parseInt($('#lengthResult').text());
    var reqTime = getCurrentTime();
    var skipCount = 0;
    // send track information to songReq.php to add song to queue
    $.ajax(
        {
            type: "POST",
            url: "songReq.php",
            dataType: "text",
            data: {action:"pushSongReq", PHPURL: id, PHPusername: username, PHPname: name,
PHPartist: artist, PHPart: art, PHPlength: length, PHPreqTime: reqTime, PHPskipCount:
skipCount},
            success: function () {
                }
        });
}
```

PHP

The majority of the server-side code for a song request is concerned with validation. The validateSongReq() function checks that the user is logged in, has not requested a song already present in the queue and has not sent a request within a specific amount of time. This cooldown time is calculated by taking the number of users with an active value greater than 0 and inputting that value to the formula outlined in the design. It then gets the time of the last request by that user and checks that the difference with the current time is greater than the cooldown. If all validation is passed then the server returns the letter “p” along with the cooldown time as it is required by the toggleSongReqDisabled() function in JavaScript.

Once the JavaScript has gathered all of the required data, it will send a pushSong request to the server. Here, an SQL query will be constructed and executed to insert a new record to the request table containing the parameters specified in the request. The updateLastReqTime() function will then be called to set the user’s last request time to the current time and the song request will then be completed with a new song successfully added to the queue.

```
<?php
function validateSongReq ($username, $currentTime, $URL) {
    // get if user logged in
    $loggedIn = userLogged ($username);
    //if user logged in
    if( $loggedIn == 1){
        // get time of last request by user
        $lastReqTime = getLastReqTime ($username);
        // get number of active users
        $activeUsers = (getActiveUsers ());
        // calculate cooldown time for request
        $reqCooldown = 20000*($activeUsers)^0.50;
        // if cooldown is over
        if ( ($currentTime - $lastReqTime) > $reqCooldown) {
            // if song not already requested
            if ((checkPresent ($URL)) == 0 ){
                // signal to push request to database
                echo ("p".$reqCooldown);
            }
            // if song already requested
            else{
                echo "already requested";
            }
        }
        // if cooldown still active
        else{
            echo ($reqCooldown);
        }
    }
    // if user not logged in
    else{
        echo "not logged in";
    }
}

function getLastReqTime ($username) {
    return(query ('SELECT lastReqTime FROM User WHERE username =
\'' .mysql_real_escape_string ($username) . '\'', 'lastReqTime'));
}

function checkPresent ($URL) {
    return (query ('SELECT COUNT(URL) AS present FROM Request WHERE URL =
\'' .mysql_real_escape_string ($URL) . '\'', 'present'));
}
```

```

function pushSongReq ($URL, $name, $artist, $art, $length, $reqTime, $skipCount) {
    // prevents request for default song values
    if ($name != "Title") {
        mysql_query('INSERT INTO `Big_Julie`.`Request` (`URL`, `name`, `artist`, `art`,
`length`, `reqTime`, `skipCount`) VALUES (\'' .mysql_real_escape_string($URL) .'\',
\''.mysql_real_escape_string($name) .'\', \''.mysql_real_escape_string($artist) .'\',
\''.mysql_real_escape_string($art) .'\', \''.mysql_real_escape_string($length) .'\',
\''.mysql_real_escape_string($reqTime) .'\',
\''.mysql_real_escape_string($skipCount) .'\')');
        mysql_query('INSERT INTO `Big_Julie`.`Vote` (`URL`, `username`) VALUES
(\'' .mysql_real_escape_string($URL) .'\', \'julie\')");
    }
}

function updateLastReqTime ($username, $currentTime) {
    mysql_query('UPDATE User SET lastReqTime = \'' . $currentTime . '\' WHERE username =
\''.mysql_real_escape_string($username) . '\'');
}

include 'main.php';

switch ($_POST['action']) {
    case "validateSongReq":
        // establish connection with MySQL
        connect();
        // reset user active counter
        updateActive($_POST['PHPusername'], 10);
        echo validateSongReq($_POST['PHPusername'], $_POST['PHPCurrentTime'],
$_POST['PHPURL']);
        break;
    case "pushSongReq":
        connect();
        // add song request to database
        pushSongReq($_POST['PHPURL'], $_POST['PHPname'], $_POST['PHPartist'], $_POST['PHPart'],
$_POST['PHPlength'], $_POST['PHPreqTime'], $_POST['PHPskipCount']);
        // update last request time of user to current time and reset active
        countdown
        updateLastReqTime($_POST['PHPusername'], $_POST['PHPreqTime']);
        break;
}
?>

```

URL	name	artist	art	length	reqTime	skipCount
0XHWCIsz0v6RlaRSGqJH3X	Smoke On The Water - 1971 Recording	Deep Purple	https://i.scdn.co/image/ab8f5aa91af053eebd4dd42737...	340133	1428695608710	0
0tIQGF7hchcbAWwSBxZFdg	Me Julie	Ali G	https://i.scdn.co/image/62c82b8b842b95b8c27b802838...	225293	1428695620553	0
1fn6EFBNidVhL3DxDJTD	Bring It All Back	S Club 7	https://i.scdn.co/image/68e5f2f7edb1edcf4eee879f9...	213600	1428695692430	0
6HMVJcdw6LsyV1b5x29sa	Hello	Lionel Richie	https://i.scdn.co/image/297c3872cefa48afea08456fe...	250200	1428695849552	0
1z0KstU8zTDpSu3WsQD0RF	Titanium (feat. Sia)	David Guetta	https://i.scdn.co/image/2c7c85408140ec370f3d5e3c95...	245053	1428695866161	0
5EZImpJljqXxfqA9RuClkf	Homesick	Catfish and the Bottlemen	https://i.scdn.co/image/62efbe3da2b38ae52a23957ad...	148092	101427450548571	0
4qlMTMS6QlqSx0TvN6PqiS	Stuck In My Teeth	Circa Waves	https://i.scdn.co/image/fb896cbcd09851b24a2bf7b7087...	183874	101427450674350	0
2EPwH1bWUcre7x8100EAky	T-Shirt Weather	Circa Waves	https://i.scdn.co/image/56644972dfa1f5761ac47bf8be...	194288	101427450929763	0
3M0YwiWNPNX8ctWQjYsXTpu	Don't Look Back Into The Sun	The Libertines	https://i.scdn.co/image/abe4af82b7ff01c6f3b6e23f6...	178266	101427451108811	0
71D4LseR0RADPbyvMeTiHR	Can't Stand Me Now	The Libertines	https://i.scdn.co/image/9434065bfbca7a6d280a198d...	203733	101427451314198	0
7DJemldOsadY7j7Bv8HKV3	Giant Peach	Wolf Alice	https://i.scdn.co/image/5454a2165badd92bf625134f6b...	275493	101427451590973	0
2LlvrdnLa3Xb1b4jYuCnl	R U Mine?	Arctic Monkeys	https://i.scdn.co/image/4d9ec146e3a257b10634d9a413...	201726	101427451797210	0
5Yvxbl6LxhQ6mTUv0Vpk	Young Chasers	Circa Waves	https://i.scdn.co/image/da0865476d7b45dc45937c01eb...	130132	101427451930645	0
1aB3C7GKpCFgsdhqzZW0U	Bloodshake	Peace	https://i.scdn.co/image/646e39793d240e380df756b5a7...	235586	101427452171091	0
5Ahv76gqMWgbLKjuc9rFB	Gold	JAWS	https://i.scdn.co/image/a6bf0ff177e3010aa8f9c2181a...	238918	101427452414643	0
49ccDOSipalJ4PcmovJy4h	Recovery	Frank Turner	https://i.scdn.co/image/b2b320355e1d3b948181e5fd3a...	208040	101427452913542	0
3AIKWmEGeQze4rw5krTY0	Chlorine	The Districts	https://i.scdn.co/image/c7abf7a3c3722b87a1e7784030...	241680	101427453157158	0
4semBrqop202J6V0iU0NRc	A Rush of Blood	Coasts	https://i.scdn.co/image/68bd1c31196aaef12aae41e4e...	248082	101427453408562	0
5FkJQLJukvrvtmi1QlgmzY	Be Slowly	JAWS	https://i.scdn.co/image/a6bf0ff177e3010aa8f9c2181a...	163347	101427453608843	0
1cFwUMRIsE6jEMRUDyBONA	Rabbit Hole	Jamie T	https://i.scdn.co/image/a2085ce48b6279dadf3313b435...	219853	101427453865347	0
1khoSOxg6VHzRIVkczrbl	Sticks 'n' Stones	Jamie T	https://i.scdn.co/image/eaab9c0aabdfb65f8110605265...	240832	101427454111322	0
2NqVgVZstbhp3FF8RieCa	Lovesick	Peace	https://i.scdn.co/image/1a8ff08bc49c3fc24695477f4...	132866	101427454248372	0
64Jrz6yWomVTxFe82CYky	Dibby Dibby Sound - Extended	DJ Fresh	https://i.scdn.co/image/dadff15759d519bbad0f4905f2a...	238319	101427454498824	0
7viqrQhPnI3YYgJxwMoGt3	Moaning Lisa Smile	Wolf Alice	https://i.scdn.co/image/41fb8f70a35dffbd779131cab...	160747	101427456408139	0

The successfully populated song queue

Requirements Met: p3, p7, p8, p10

Test Number	Process Tested	Requirement Tested	Type	Data Entered / Action Performed	Expected Result	Actual Result	Test Passed?	Corrective Action
10	P3 (Song)	Check that search function displays relevant results for search query with track name and artist specified	Normal	Track name: Hello Artist: Lionel Richie	Search result module displays Hello by Lionel Richie	Search result module displays Hello by Lionel Richie	Yes	None
				Track name: Smoke on the Water Artist: Deep Purple	Search result module displays Smoke on the Water by Deep Purple	Search result module displays Smoke on the Water by Deep Purple		
				Track name: Tip Toe Thru' The Tulips With Me Artist: Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim		
11	P3 (Song)	Check that search function displays relevant results for queries with only track name specified	Normal	Track name: Hello	Search result module displays popular track titled Hello	Search result module displays Hello by J. Cole	Yes	None

11	P3 (Song)	Check that search function displays relevant results for queries with only track name specified	Normal	Track name: Smoke on the Water	Search result module displays popular track titled Smoke on the Water	Search result module displays Smoke on the Water by Deep Purple	Yes	None
				Track name: Tip Toe Thru' The Tulips With Me	Search result module displays popular track titled Tip Toe Thru' The Tulips With Me by Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim		
12		Check that search function displays relevant results for queries with only artist specified	Normal	Artist: Lionel Richie	Search Result module displays popular track by Lionel Richie	Search result module displays All Night Long by Lionel Richie	Yes	None
				Artist: Deep Purple	Search Result module displays popular track by Deep Purple	Search Result module displays Smoke on the Water by Deep Purple		
				Artist: Tiny Tim	Search Result module displays popular track by Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim		

				Track name: Smoke water	Search Result module displays popular track with name similar to smoke water	Search Result module displays Smoke on the Water by Deep Purple		
13	P3 (Song)	Check that search function displays relevant results for queries where track or artist is not exact	Borderline	Artist: Lionel	Search result module returns popular track by artist name similar to Lionel	Search result module displays All Night Long by Lionel Richie	Yes	None
				Track name: Rabbit Artist: T	Search result module returns popular track with name similar to Rabbit and artist similar to T	Search result module displays Rabbit Hole by Jamie T		
				Track name: sidfiosdjfiosdfjsdfi Artist: sduifhdugf	Search result module does not open and alert states that no results were found	Search result module opens and displays default values		
14		Check that search function shows no results when Spotify cannot find a song	Invalid	Track name: sidfiosdjfiosdfjsdfi Artist: sduifhdugf	Search result module does not open and alert states that no results were found	Search result module opens and displays default values	No	Changed code to check if object returned by Spotify is empty. If true, display error message else display results.

15	P3 (Song)	Check that song is added to queue when yes button is pressed (p3)	Normal	Yes button clicked on track Hello by Lionel Richie	Search result module closes and new record for Hello by Lionel Richie created in database	Search result module closes and new record for Hello by Lionel Richie created in database	Yes	None
				Yes button clicked on track Tip Toe Thru' The Tulips With Me by Tiny Tim	Search result module closes and new record for Tip Toe Thru' The Tulips With Me by Tiny Tim created in database	Search result module closes but no new record is created	No	Sanitise input in PHP so that apostrophes are escaped
				Yes button clicked on track Smoke on the Water by Deep Purple	Search result module closes and new record for Smoke on the Water by Deep Purple created in database	Search result module closes and new record for Smoke on the Water by Deep Purple created in database	Yes	None
16		Check that no song is added to queue when no song is selected	Invalid	Yes button clicked when no results are returned from Spotify	No activity	Record added to database with default values	No	Added code so if default values are sent to server, do not process request

17	P3 (Song)	Check that no song longer than 7 minutes is displayed in search results (p8)	Invalid	Artist: Deadmau5	Search result module displays only radio edit versions of songs by Deadmau5	Search result module displays only radio edit versions of songs by Deadmau5	Yes	None
				Artist: Daft Punk	Search result module displays only radio edit versions of songs by Daft Punk	Search result module displays only radio edit versions of songs by Daft Punk		
18		Check that songs just over 7 minutes long are rejected	Borderline	Track name: When the levee breaks Artist: Led Zeppelin	Search result module displays only shorter versions of When the Levee Breaks by Led Zeppelin	Search result module displays shorter cover versions of When the Levee Breaks	Yes	None
				Track name: Around the World Artist: Daft Punk	Search result module displays only radio edit versions of Around the World by Daft Punk	Search result module displays only radio edit versions of Around the World by Daft Punk		

19	P3 (Song)	Check that songs just under 7 minutes long are accepted	Borderline	Track name: Ghosts 'n' Stuff	Search result module displays Nero remix but not original song	Search result module displays Nero remix but not original song	Yes	None
				Track name: Musique Artist: Daft Punk	Search result module displays Musique by Daft Punk	Search result module displays Musique by Daft Punk		
20		Check that no explicit songs are displayed in search results (p7)	Invalid	Track name: Alphabet Artist: Lil Wayne	Search result module does not open and alert states that no results were found	Search result module does not open and alert states that no results were found	Yes	None
				Track name: Oh my Darling Don't Cry Artist: Run the Jewels	Search result module does not open and alert states that no results were found	Search result module displays Oh my Darling Don't Cry by Run the Jewels		
21		Check that songs already in queue are rejected	Invalid	Track name: Don't Look Back into the Sun Artist: The Libertines	Song not added to queue and alert states that song already requested	Song not added to queue and alert states that song already requested	Yes	None

21	P3 (Song)	Check that songs already in queue are rejected	Invalid	Track name: T-Shirt Weather Artist: Circa Waves	Song not added to queue and alert states that song already requested	Song not added to queue and alert states that song already requested	Yes	None	
22		Check that songs similar to songs already in queue are accepted	Borderline	Track name: Hello Artist: J. Cole	Song added to queue	Song added to queue	Yes	None	
				Track name: Can't Stand me Now Artist: The Libertines					
23		Check that requests are rejected if sent within cooldown time (p10)	Invalid	Send two song requests consecutively	First request accepted, second request rejected with alert stating still in cooldown	First request accepted, second request rejected with alert stating still in cooldown	Yes	None	

All unsuccessful tests have been corrected so development of next process can begin

Refreshing

With requests successfully being added to the queue, the placeholder images and text could now be replaced with real information from the database.

JavaScript

To ensure that the user did not have to refresh the page to update the data, I set a timer that would call the refresh() function every 5 seconds. This function would not refresh the entire page, but detect if any song titles or cover art had changed from those that were currently displayed and update them if needed. The getNowPlaying() and getNextSongs() functions take the track name and cover art from the HTML along with the latest data from various AJAX calls to the server and compare the values. If the values are not equivalent then the song featured in that module must have changed and the setThumb() and setName() functions are called to update that module with the new data. If the client detects that the “currently playing” song has changed then the skip and vote buttons are re-enabled since a new song must be playing. This creates a dynamic experience for the user with the page changing as new songs are played, requested and voted for.

```
window.setInterval(function(){
    // update page with latest data every 5s
    refresh();
}, 5000);
function refresh(){
    getNowPlaying();
    getNextSongs();
}
function getNowPlaying(){
    // get album cover of currently playing song
    $.ajax({
        type: "POST",
        url: "init.php",
        dataType: "text",
        data: {action:"getNextSongCover", PHPindex:0},
        success: function (cover) {
            var thumb = $('#thumbNow');
            // if thumbnail has changed
            if (($(thumb).prop('src')) != $.trim(cover)){
                // enable skip button if disabled
                if ($("#btnSkip").prop('disabled') == true ){
                    $("#btnSkip").toggleClass("skipped");
                    $("#btnSkip").prop('disabled', function(i, v) { return !v; });
                }
                // enable vote buttons if disabled
                $('.voteBtn').each(function(currentElement, index) {
                    if ($(this).prop('disabled') == true ){
                        $(this).toggleClass("voted");
                        $(this).prop('disabled', function(i, v) { return !v; });
                    }
                });
            }
            // get name of currently playing song
            $.ajax({
                type: "POST",
                url: "init.php",
                dataType: "text",
                data: {action:"getNextSongName", PHPindex:0},
                success: function (name) {
                    // update marquee with new name
                    $('#nowPlaying').fadeToggle("slow",function(){

```

```
$('#nowPlaying').html('<marquee>NOW PLAYING : '+name+'</marquee>');
});
$('#nowPlaying').fadeToggle("slow");
}
});
}
// update thumbnail with new cover art
setThumb(thumb,$.trim(cover));
}
});
}
}

function getNextSongs(){
// for each thumbnail with class 'thumbNext'
$('.thumbNext').each(function(currentElement, index) {
// get nth album cover from request database
$.ajax(
{
type: "POST",
url: "init.php",
dataType: "text",
data: {action:"getNextSongCover", PHPindex:(currentElement+1)},
success: function (cover) {
// set thumbnail to new cover
var thumb = $('#thumbNext'+String(currentElement+1));
setThumb(thumb,$.trim(cover));
}
});
// for each paragraph with class 'pNext'
$('.pNext').each(function(currentElement, index) {
// get nth name from request database
$.ajax(
{
type: "POST",
url: "init.php",
dataType: "text",
data: {action:"getNextSongName", PHPindex:(currentElement+1)},
success: function (name) {
name = $.trim(name);
// update page with new song name
var para = $('#pNext'+String(currentElement+1));
setName(para,name);
}
});
});
}
);

function setThumb(thumb,art){
if (($(thumb).prop('src')) != art){
$(thumb).fadeToggle("slow",function(){
$(thumb).attr("src",art);
});
$(thumb).fadeToggle("slow");
}
}

function setName(para,name){
if ($(para).text() != name){
$(para).fadeToggle("slow",function(){
$(para).html(name);
});
$(para).fadeToggle("slow");
}
}
```

PHP

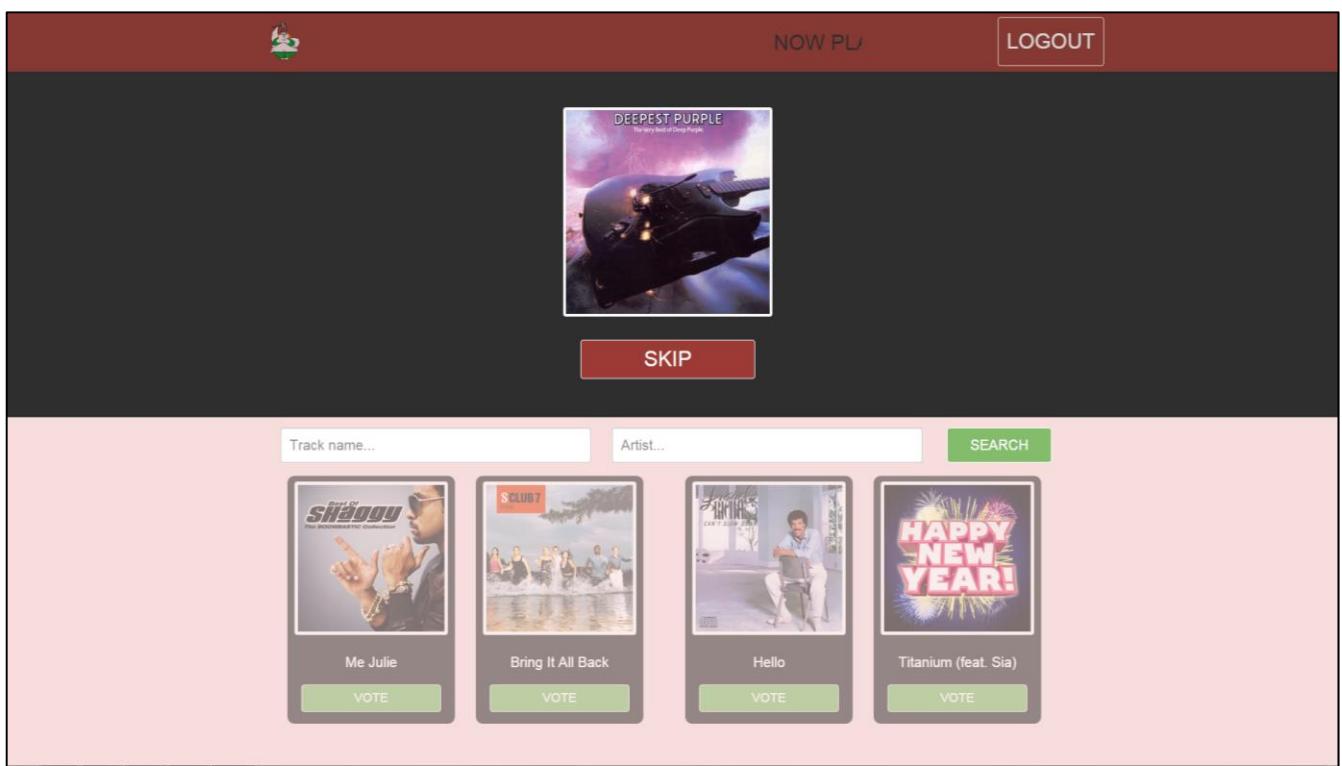
The server side code needed for updating a page is very simple. All the JavaScript needs is the latest cover art and track names for the currently playing song and the next four songs to be played. This can be handled by two simple queries that order the request table by request time in ascending order. An offset can then also be applied as a parameter with 0 pointing to the currently playing song, 1 pointing to the next song to be played and so forth.

```
<?php

function getNextSongCover ($index) {
    return query('SELECT art FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT 1
OFFSET '.mysql_real_escape_string($index), 'art');
}

function getNextSongName ($index) {
    return query('SELECT name FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT 1
OFFSET '.mysql_real_escape_string($index), 'name');
}

include 'main.php';
switch ($_POST['action']) {
    case "getNextSongCover":
        connect();
        echo getNextSongCover($_POST['PHPindex']);
    break;
    case "getNextSongName":
        connect();
        echo getNextSongName($_POST['PHPindex']);
    break;
}
?>
```



The page correctly displaying songs in the queue

Requirements Met: o2, o3, o4

Process 4 (Vote)

With the user able to see the next songs in the queue, the voting system could now be implemented. The logic for sending vote requests was essentially a simpler version of that for song requests so the development process consisted mostly of copying and modifying existing code. Each next song module is given an offset value starting from the left of the screen. When the vote button is pressed, the offset value for that module is sent with the vote request to the server. The server can then sort the request table by request time in ascending order and use the offset to determine the URL that is being voted for. That URL and username can then be added as a new record to the vote table. SQL automatically rejects records that are exact duplicates so no validation is required. Once the server returns that the vote request was successful, the corresponding vote button is disabled until the next song is played.

JavaScript

```
function voteReq(offset){  
    var username = (localStorage.getItem("username"));  
    $.ajax(  
    {  
        type: "POST",  
        url: "vote.php",  
        dataType: "text",  
        data: {action:"voteReq", PHPusername:username, PHPoffset:offset},  
        success: function (voted) {  
            // if vote request not successful  
            if ($.trim(voted) != "1"){  
                // display error message  
                alert(voted);  
            }  
            else {  
                // disable vote button  
                $('#btnVote'+String(offset+1)).toggleClass("voted");  
                $('#btnVote'+String(offset+1)).prop('disabled', function(i, v) { return !v;});  
        }  
    });  
}
```

PHP

```
<?php
function getURL($offset) {
    return query('SELECT URL FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT 1
OFFSET '.($offset+1), 'URL');
}

function voteReq($username,$offset) {
    // get whether user is logged in
    $loggedIn = userLogged($username);
    // if user logged in
    if ($loggedIn == TRUE) {
        $URL = getURL($offset);
        pushVote($username,$URL);
        return 1;
    }
    // if user not logged in
    else{
        echo "not logged in";
    }
}

function pushVote($username,$URL) {
    mysql_query('INSERT INTO `Big_Julie`.`Vote`(`username`, `URL`) VALUES
(\'' .mysql_real_escape_string($username). '\',
\'' .mysql_real_escape_string($URL). '\' )');
}

include 'main.php';
switch ($_POST['action']) {
    case "voteReq":
        connect();
        updateActive($_POST['PHPusername'],10);
        echo voteReq($_POST['PHPusername'],$_POST['PHPopffset']);
        break;
}
?>
```

Test Number	Process Tested	Description	Type	Data Entered / Action Performed	Expected Result	Actual Result	Test Passed?	Corrective Action
24	P4 (Vote)	Check that vote requests are added to database when vote button pressed	Normal	Username: 08cliggins Vote button pressed	Vote added to database and vote button disabled	Vote added to database and vote button disabled	Yes	None
				Username: 08amaher Vote button pressed				
				Username: 08cpreece Vote button pressed				
25	P4 (Vote)	Check that duplicate votes are rejected	Invalid	Username: 08cliggins Vote button pressed twice	Vote not added to database	Vote not added to database	Yes	None
				Username: 08amaher Vote button pressed twice				
				Username: 08cpreece Vote button pressed twice				

All tests passed so development of next process can begin.

Process 5 (Skip)

The skip request logic was very similar to that for a vote request so the implementation was again a case of redesigning existing code. Since only the currently playing song could be skipped, the client-side code consisted only of a button press event calling an AJAX query which upon success, would disable the skip button until the next song was played. The server did need to perform some validation to check that the user was logged in and had not already skipped the song but this was essentially the validation process for a song request so it was trivial to adjust the code from that module. Most of the complexity in the vote and skip functionality did not lie in making the requests but in the way that they are processed later.

JavaScript

```
function skipReq() {
    var username = (localStorage.getItem("username"));
    $.ajax(
        {
            type: "POST",
            url: "skip.php",
            dataType: "text",
            data: {action:"skipReq", PHPusername:username},
            success: function (skipped) {
                // if skip request not made
                if ($.trim(skipped) != "1") {
                    // output error message
                    alert(skipped);
                }
                // if skip request success
                else{
                    // change skip button colour to grey
                    $("#btnSkip").toggleClass("skipped");
                    // disable skip button
                    $("#btnSkip").prop('disabled', function(i, v) { return !v; });
                    setTimeout(refresh,1000);
                }
            }
        });
}
```

PHP

```
<?php
function skipReq($username) {
    // get whether user is logged in
    $loggedIn = userLogged($username);
    // if user logged in
    if ($loggedIn == TRUE) {
        // if user not already skipped song
        if (getSkipped($username) == 0){
            // set user to have skipped song
            updateActive($username,10);
            setSkipped($username);
            return 1;
        }
        else {
            return ("already skipped song");
        }
    }
    else {
        return ("not logged in");
    }
}

function getSkipped($username) {
    return query('SELECT skipped FROM User WHERE username =
\'' .mysql_real_escape_string($username) . '\'', 'skipped');
}

function setSkipped($username) {
    mysql_query('UPDATE User SET skipped = 1 WHERE username
=\'' .mysql_real_escape_string($username) . '\'');
}

include 'main.php';
switch ($_POST['action']) {
    case "skipReq":
        connect();
        // reset active counter for user
        updateActive($_POST['PHPusername'],10);
        echo skipReq($_POST['PHPusername']);
        break;
}
?>
```

Test Number	Process Tested	Description	Type	Data Entered / Action Performed	Expected Result	Actual Result	Test Passed?	Corrective Action
26	P5 (Skip)	Check that user is flagged as skipped when skip button pressed	Normal	Username: 08cliggins Vote button pressed	User flagged as skipped in database, skip button disabled	User flagged as skipped in database, skip button disabled	Yes	None
				Username: 08amaher Vote button pressed				
				Username: 08cpreece Vote button pressed				
27	P5 (Skip)	Check that duplicate skip requests are rejected	Invalid	Username: 08cliggins Vote button pressed twice	Skip request not added to database	Skip request not added to database	Yes	None
				Username: 08amaher Vote button pressed twice				
				Username: 08cpreece Vote button pressed twice				

All tests passed so development of next process can begin.

Process 6 (Next Song)

The player was by far the most complicated aspect of the system to develop. It must operate completely autonomously without any user interaction for extended periods of time. This means that the algorithms developed must be fault tolerant and ensure consistency with the information displayed on the client-side.

JavaScript

The player is a separate entity residing on the same domain as the user interface. It interacts with the server, getting the URL of the next song to be played along with its track length with the getURL() and getLength() functions. The playSong() function is then called to signal to the Spotify client to play that URL and a countdown timer is set with the length of the track. When the song has finished playback, the songEnd() function is called to update the queue and the playing flag is set to false. At the next update tick of the player, the play() function will be called which will repeat the process.

Originally, the system was designed so that at the end of playback, the play() function would be called immediately but during testing, it was found that the recursive nature of that solution caused a memory leak and did not lend itself to being interrupted when the song is skipped. Instead, I chose to have a timer that checked every second if the system was flagged as playing. This meant that the chain of functions for playing a single song could terminate by flagging playing as false, freeing up memory for use in playing the next song.

This solution also decreased the complexity of the skip functionality on the player-side. Within the timer, the system could also send a request to the server to check that the song had reached the criteria to be skipped. If the server returned that the song should be skipped then the countdown timer started when the song began playing could be cancelled and the songEnd() function would be called. The dropSong() function would also be called to signal to the server that the skipped song should be deleted from the queue. In the next second, the play() function would be called and the next song would begin playback.

```
var Spotify = {
    // initialise Spotify as an object
    // source: http://khlo.co.uk/stuff/spotify/
    currentSong: '',
    play: function(songId) {
        if (songId != this.currentSong) {
            if (!document.getElementById('spotifyPlayer')) {
                spotPlayer = document.createElement('div');
                spotPlayer.id = 'spotifyPlayer';
                document.body.appendChild(spotPlayer);
            }
            document.getElementById('spotifyPlayer').innerHTML = '<iframe src="spotify:track:' + songId + '" height="1" width="1" style="visibility: hidden;"></iframe>';
            this.currentSong = songId;
        }
    }
}
function init() {
    // ensure that first song in queue will play on startup
    localStorage.setItem("playing", "false")
}
function play() {
    // set song playing flag as true
    localStorage.setItem("playing", "true")
    // get URL of song to play
    getURL();
}
```

```
function getURL() {
    // get URL of song to play from request database
    $.ajax(
    {
        type: "POST",
        url: "play.php",
        dataType: "text",
        data: {action:"getURL"},
        success: function (song) {
            // get length of song to play
            getLength($.trim(song));
        }
    });
}

function getLength(URL) {
    // get Length of song to play from request database
    $.ajax(
    {
        type: "POST",
        url: "play.php",
        dataType: "text",
        data: {action:"getLength", PHPURL:URL},
        success: function (length) {
            // reduce length of song by 1 second to allow for processing
            time between song changes
            length = length - 1000;
            playSong(URL,length);
        }
    });
}

function playSong(URL,length) {
    Spotify.play(URL);
    // set timer to execute end of song logic when song ends
    var timer = setTimeout(function(){songEnd(URL);},length);
    localStorage.setItem("URL",URL);
    // store id of timer so that it can be cancelled if song is skipped
    localStorage.setItem("timer",timer);
}

function songEnd(URL) {
    // push song to end of queue by adding 1000000000000ms to request time
    var currentTime = getCurrentTime() + 1000000000000000 ;
    $.ajax(
    {
        type: "POST",
        url: "play.php",
        dataType: "text",
        data: {action:"nextSong", PHPURL:URL, PHptime:currentTime},
        success: function () {
            // flag that no song is playing
            localStorage.setItem("playing","false");
        }
    });
}
```

```
function getSkipped(){
    var URL = localStorage.getItem("URL");
    // get whether song has met criteria for skipping
    $.ajax(
        {
            type: "POST",
            url: "play.php",
            dataType: "text",
            data: {action:"getSkipped", PHPURL:URL},
            success: function (skipped) {
                // if song has met skipping criteria
                if ($.trim(skipped) == "true"){
                    // disable countdown timer for song end
                    var timer =
parseInt(localStorage.getItem("timer"));
                    clearTimeout(timer);
                    // run logic as if song had ended
                    songEnd(localStorage.getItem("URL"));
                    dropSong();
                }
            }
        });
}

function dropSong(){
    var URL = localStorage.getItem("URL");
    $.ajax(
        {
            type: "POST",
            url: "play.php",
            dataType: "text",
            data: {action:"dropSong", PHPURL:URL},
            success: function () {
            }
        });
}

$(document).ready(function(){
    init();
    // every second
    window.setInterval(function(){
        // check whether song has been skipped
        getSkipped();
        // if song not currently playing
        if (localStorage.getItem("playing") == "false"){
            // play song
            play();
        }
    }, 1000);
});
```

PHP

A large portion of the player's logic is performed on the server. Every second, the player sends a getSkipped request which calls the getSkipped() function in PHP. This function first calls the getNewReq() function to check whether the song currently playing is not at the top of queue. This situation occurs when the system is playing previously requested songs and a new song is requested. In that case, the server will signal to the player to end playback and begin playing the song that has just been requested. If getNewReq() returns that the condition has not been satisfied then the getSkipCount() and getActiveUsers() functions are called to fetch the number of users that have voted to skip the currently playing song and the number of active users in total. The server then checks whether the skip count is greater than 60% of the active users and if true, the server signals to the player that the song has been skipped. After the player has processed skipping the song, the server receives a dropSong request and a query will be executed to delete the skipped song from the queue.

```
function getSkipped ($URL) {
    if (getNewReq ($URL) == 1) {
        return "true";
    }
    // get number of votes to skip current song
    $skipCount = getSkipCount ();
    //get number of active users
    $activeUsers = getActiveUsers ();
    // if more than 60% of users have voted to skip
    if ($skipCount > ($activeUsers*0.6)) {
        return "true";
    }
    else{
        return "false";
    }
}

// get if new song at top of queue
function getNewReq ($URL) {
    $newURL = getURL(0);
    // if currently playing song is not at top of queue
    if ($URL != $newURL) {
        return 1;
    }
    else {
        return 0;
    }
}

function getSkipCount () {
    return query('SELECT COUNT(skipped) AS skipCount FROM User WHERE skipped =
1','skipCount');
}

function dropSong ($URL) {
    mysql_query ('DELETE FROM Request WHERE URL =
\'' .mysql_real_escape_string($URL) . '\'');
    mysql_query ('DELETE FROM Vote WHERE URL = \'' .mysql_real_escape_string($URL) . '\'
AND username != \'julie\'');
}
```

Before the player begins playing a new song, it sends a nextSong request to the server. The server will then call the nextSong() function to move the song at the top of the queue to the bottom. The votes() function is then called which executes a nested SQL statement that calculates the sum of the vote weights of each user that has voted for a specific song. It then reduces the request time of that song by the total vote weight, effectively moving the song further up the queue depending on the number and weight of the votes received. To reset the logic for the new song, the activeDec() and setSkipped() functions are called to decrement the active counter of each user by 1 and flag all users as having not voted to skip.

```
<?php

function nextSong ($URL,$time) {
    mysql_query ('UPDATE Request SET ReqTime = '.mysql_real_escape_string($time).'
WHERE URL = \'' .mysql_real_escape_string($URL).'\'');
}

function votes () {
    // for top 4 songs in queue
    for ($i = 0; $i <=3 ; $i++) {
        // get URL of song
        $URL = getURL($i);
        // calculate new request time for song
        mysql_query ('UPDATE Request SET reqTime = reqTime - ((SELECT
SUM(voteWeight) FROM User WHERE username IN ( SELECT username FROM Vote WHERE URL =
\''.mysql_real_escape_string($URL).'\' ) )*60000) WHERE URL =
\''.mysql_real_escape_string($URL).'\'');
    }
}

function activeDec () {
    mysql_query ('UPDATE User SET active = active - 1 WHERE active > 0');
}

function setSkipped () {
    mysql_query ('UPDATE User SET skipped = 0');
}

include 'main.php';
switch ($_POST['action']) {

    case "nextSong":
        connect();
        // update request table since song has ended
        nextSong($_POST['PHPURL'],$_POST['PHPtime']);
        votes();
        // decrement active counter of every user
        activeDec();
        // set each user to have not voted to skip
        setSkipped();
        break;
    case "getSkipped":
        connect();
        // get whether song has met all skip criteria
        echo getSkipped($_POST['PHPURL']);
        break;
    case "dropSong":
        connect();
        dropSong($_POST['PHPURL']);
        break;
}
?>
```

Requirements Met: o1, p4, p5, p6, p9

28	P6 (Next Song)	Check that songs are played in order of queue	Normal	Open player	Spotify plays Smoke on the Water followed by Me Julie and Bring it All Back	Spotify plays Smoke on the Water followed by Me Julie and Bring it All Back	Yes	None
29		Check that songs are played in their entirety	Normal	Open player	Spotify plays a new song only when the current song has ended	Spotify plays a new song only when the current song has ended	Yes	None
30		Check that songs are played without gaps	Normal	Open player	Spotify plays a new song immediately after current song has ended	Spotify plays a new song 3-5 seconds after current song has ended	No	Reduced track length by 3 seconds to allowed for processing between songs
31		Check that song is skipped if >60% of active users have voted to skip (p5)	Normal	Set 100 users as active, 60 as skipped. Press skip button	Current song ends and next song begins playing	Current song ends and next song begins playing	Yes	None
			Borderline	Set 100 users as active, 59 as skipped. Press skip button	Current song continues playing and skip button disabled	Current song continues playing and skip button disabled	Yes	None

32	P6 (Next Song)	Check that songs move forward in queue depending on how many votes they have received (p4)	Normal	Set 100 users as having voted for song at bottom of queue	When current song ends, song at bottom of queue moves forward	When current song ends, song at bottom of queue moves forward	Yes	None
33		Check that certain users have more influence in the voting algorithm (p6)	Normal	Set 1 user with weight 500 as having voted for song at bottom of queue, 100 users with weight 1 as having voted for second song from bottom of queue	When current song ends, both songs move forward in queue with first song now above second	When current song ends, both songs move forward in queue with first song now above second	Yes	None
33		Check that certain users have more influence in the voting algorithm	Borderline	Set 5 users with weight 10 as having voted for song at bottom of queue, 20 users with weight 2 as having voted for second song from bottom of queue	When current song ends, both songs move forward in queue with first song now above second	When current song ends, both songs move forward in queue with first song now above second	Yes	None
34		Check that player begins at start of queue when last song ends (p9)	Normal	Open player	When Lovesick ends, Smoke on the Water begins	When Lovesick ends, Smoke on the Water begins	Yes	No

35	P6 (Next Song)	Check that song is removed from queue when skipped	Normal	Set 100 users as active, 60 as skipped. Press skip button	Current song ends and is removed from database	Current song ends and is removed from database	Yes	No
36		Check that users are flagged as having not skipped when next song begins playing	Normal	Open player, wait for first song to end	All users' skipped value is set to 0 in database	All users' skipped value is set to 0 in database	Yes	No

Input Requirements

<u>ID</u>	<u>Requirement</u>	<u>Achieved During Development of</u>
i1	Must be controllable with an iPhone	Markup 7 (Mobile Device Compatibility)
i2	Users can control the system through a web interface	Markup 1 (Head)
i3	Allow users to input their song requests as text	Markup 4 (Search)
i4	Allow users to vote on songs in queue with buttons	Markup 6 (Next Songs)
i5	Allow users to skip unfavourable songs with buttons	Markup 3 (Jumbotron)
i6	Every Sixth Form student and teacher will have a unique username and password	Process 1 (Login)
i7	Allow users to login with username and password	Markup 2 (Navbar)

Processing Requirements

<u>ID</u>	<u>Requirement</u>	<u>Achieved During Development of</u>
p1	Log user in if username and password are correct	Process 1 (Login)
p2	Allow users to send requests only when logged in	Process 2 (Request)
p3	Add requested songs to the end of a queue	Process 3 (Song)
p4	Move song request forward in queue based on number of votes received.	Process 6 (Next Song)
p5	Skip the currently playing song if > 60% of active users request skip	Process 6 (Next Song)
p6	Allow certain users to have more influence in the voting algorithm	Process 6 (Next Song)
p7	Filter song requests for explicit content	Process 3 (Song)
p8	Only play song requests that are under 7 minutes long	Process 3 (Song)
p9	Return to start of queue when final song ends	Process 6 (Next Song)
p10	Allow one song request per user in a certain amount of time dependent on number of active users	Process 3 (Song)

Output Requirements

<u>ID</u>	<u>Requirement</u>	<u>Achieved During Development of</u>
o1	Stream requested songs from Spotify	Process 6 (Next Song)
o2	Display currently playing track name and cover art	Markup 3 (Jumbotron) / Refreshing
o3	Display track name and cover art of next songs in queue	Markup 6 (Next Songs) / Refreshing
o4	Display all outputted information without refreshing or changing page	Refreshing
o5	Display all information in an appropriate format on desktop and mobile devices	Markup 7 (Mobile Device Compatibility)

Design Requirements

ID	Requirement	Achieved During Development of
d1	Dark red horizontal navbar at top of page containing d2,d3,d4	Markup 2 (Navbar)
d2	Big Julie logo in first 30% of navbar	Markup 2 (Navbar)
d3	Currently playing song name as horizontal marquee in next 40% of navbar, white font colour	Markup 2 (Navbar)
d4	Green login button in final 30% of navbar	Markup 2 (Navbar)
d5	Dark grey jumbotron below navbar, 50% page height containing d6,d7	Markup 3 (Jumbotron)
d6	Thumbnail of currently playing song art, 25% page height, horizontally centred	Markup 3 (Jumbotron)
d7	Red skip button below thumbnail, font colour white, 10% page width	Markup 3 (Jumbotron)
d8	Search module, 50% page width horizontally centred below jumbotron. Containing d9, d10.	Markup 4 (Search)
d9	Text box for requesting song, black font colour, 40% page width	Markup 4 (Search)
d10	Green search button adjacent to search box, white font colour, 10% page width	Markup 4 (Search)
d11	Five next song modules, 15% page width, containing d12, d13, d14, d15. Arranged in equally spaced row.	Markup 6 (Next Songs)
d12	Thumbnail of next song to play, at top of next song module, 90% module width.	Markup 6 (Next Songs)
d13	Name of next song to play, below thumbnail.	Markup 6 (Next Songs)
d14	Green vote button, below song name, first 50% of module width, font colour white.	Markup 6 (Next Songs)
d15	Red skip button, below song name, last 50% of module width, font colour white.	N/A
d16	Light pink page background	Markup 2 (Navbar)

Alpha Testing

During the development process, I tested the system to ensure that it behaved in the way that I had designed it. Using the test plan from the design, I created a test table that allowed me to verify that each process requirement was fulfilled along with other requirements that were intrinsic to the operation of my system.

I carried out each test as they had been described in the test plan, entering the data specified and performing any necessary actions. I then observed the outcome of the test and compared it to the outcome that was expected if the requirement was fulfilled. If the outcomes were not correct then I would edit my code and perform the test again until the result was correct and the test was passed.

At the end of the development I carried out each test again to ensure that the system had fulfilled every requirement in the specification. Some requirements such as those for the design could not be quantified and were instead verified by my observation of the system.

Test Number	Process Tested	Requirement Tested	Type	Data Entered / Action Performed	Expected Result	Actual Result	Test Passed?	Corrective Action
1	P1 (Login)	Check that valid user credentials lead to login (p1)	Normal	Username: 08cliggins Password: 2773	Website changes to logged in mode. User flagged as logged in in database	Website changes to logged in mode. User flagged as logged in in database	Yes	None
				Username: 08amaher Password: 655117DINKUM				
				Username: 08cpreece Password: 44235427205MAYO				
2	P1 (Login)	Check that invalid passwords for valid usernames are rejected.	Invalid	Username: 08cliggins Password: invalid	Alert stating that login was unsuccessful. No change in database	Alert stating that login was unsuccessful. No change in database	Yes	None
				Username: 08amaher Password: INVALID				
				Username: 08cpreece Password: 2345345346				
3	P1 (Login)	Check that invalid usernames are rejected	Invalid	Username: Invalid Password: Invalid	Alert stating that login was unsuccessful. No change in database	Alert stating that login was unsuccessful. No change in database	Yes	None
				Username: 45645766 Password: 123123				
				Username: INVALID Password: qwerty				
4	P1 (Login)	Check that empty username and password fields are rejected	Invalid	Username empty Password empty	Alert stating that login was unsuccessful. No change in database	Alert stating that login was unsuccessful. No change in database	Yes	None
				Username empty Password: test				
				Username: test Password empty				

5	P1 (Login)	Check that the shortest username and password are accepted if valid	Borderline	Username: 08sclay Password:LOTIONICEMEN664	Website changes to logged in mode. User flagged as logged in in database	Website changes to logged in mode. User flagged as logged in in database	Yes	None			
6				Username: 08troberts Password: 2049							
7	P2 (Request)	Check that requests are accepted for users that are logged in (p2)	Normal	Username: 08kbroadhurst Password: 985404	Website changes to logged in mode. User flagged as logged in in database	Website changes to logged in mode. User flagged as logged in in database	Yes	None			
				Username: 08bmcgrail Password: GLOBEDKUDZUSMEA							
8				Username: 08troberts Request: Skip	userLogged() in PHP returns true	userLogged() in PHP returns true	Yes	None			
				Username: 08troberts Request: Vote							
				Username: test1 Request: Song							
				Username: 08tmacklin Request: Vote	Alert stating that login is required	Alert stating that login is required	No	Added if statement to check that user is logged in. Continue if true else output login required			
				Username: 08tmacklin Request: Skip							
				Username: 08vadcock Request: Song							

9	P2 (Request)	Check that users are flagged as active when request is accepted.	Normal	Username: 08troberts Request: Skip	Active value in user's record is set to 10 in database	Active value in user's record is set to 10 in database	Yes	None	
10	P3 (Song)	Check that search function displays relevant results for search query with track name and artist specified		Username: 08troberts Request: Vote					
				Username: test1 Request: Song					
				Track name: Hello Artist: Lionel Richie	Search result module displays Hello by Lionel Richie	Search result module displays Hello by Lionel Richie	Yes	None	
11	P3 (Song)	Check that search function displays relevant results for queries with only track name specified	Normal	Track name: Smoke on the Water Artist: Deep Purple	Search result module displays Smoke on the Water by Deep Purple	Search result module displays Smoke on the Water by Deep Purple			
				Track name: Tip Toe Thru' The Tulips With Me Artist: Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim			
				Track name: Hello	Search result module displays popular track titled Hello	Search result module displays Hello by J. Cole			

11	P3 (Song)	Check that search function displays relevant results for queries with only track name specified	Normal	Track name: Smoke on the Water	Search result module displays popular track titled Smoke on the Water	Search result module displays Smoke on the Water by Deep Purple	Yes	None
				Track name: Tip Toe Thru' The Tulips With Me	Search result module displays popular track titled Tip Toe Thru' The Tulips With Me	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim		
12		Check that search function displays relevant results for queries with only artist specified	Normal	Artist: Lionel Richie	Search Result module displays popular track by Lionel Richie	Search result module displays All Night Long by Lionel Richie	Yes	None
				Artist: Deep Purple	Search Result module displays popular track by Deep Purple	Search Result module displays Smoke on the Water by Deep Purple		
				Artist: Tiny Tim	Search Result module displays popular track by Tiny Tim	Search result module displays Tip Toe Thru' The Tulips With Me by Tiny Tim		

				Track name: Smoke water	Search Result module displays popular track with name similar to smoke water	Search Result module displays Smoke on the Water by Deep Purple		
13	P3 (Song)	Check that search function displays relevant results for queries where track or artist is not exact	Borderline	Artist: Lionel	Search result module returns popular track by artist name similar to Lionel	Search result module displays All Night Long by Lionel Richie	Yes	None
				Track name: Rabbit Artist: T	Search result module returns popular track with name similar to Rabbit and artist similar to T	Search result module displays Rabbit Hole by Jamie T		
				Track name: sidfiosdjfiosdfjsdfi Artist: sduifhdufg	Search result module does not open and alert states that no results were found	Search result module opens and displays default values		
14		Check that search function shows no results when Spotify cannot find a song	Invalid	Track name: sidfiosdjfiosdfjsdfi Artist: sduifhdufg	Search result module does not open and alert states that no results were found	Search result module opens and displays default values	No	Changed code to check if object returned by Spotify is empty. If true, display error message else display results.

15	P3 (Song)	Check that song is added to queue when yes button is pressed (p3)	Normal	Yes button clicked on track Hello by Lionel Richie	Search result module closes and new record for Hello by Lionel Richie created in database	Search result module closes and new record for Hello by Lionel Richie created in database	Yes	None
				Yes button clicked on track Tip Toe Thru' The Tulips With Me by Tiny Tim	Search result module closes and new record for Tip Toe Thru' The Tulips With Me by Tiny Tim created in database	Search result module closes but no new record is created	No	Sanitise input in PHP so that apostrophes are escaped
				Yes button clicked on track Smoke on the Water by Deep Purple	Search result module closes and new record for Smoke on the Water by Deep Purple created in database	Search result module closes and new record for Smoke on the Water by Deep Purple created in database	Yes	None
16		Check that no song is added to queue when no song is selected	Invalid	Yes button clicked when no results are returned from Spotify	No activity	Record added to database with default values	No	Added code so if default values are sent to server, do not process request

17	P3 (Song)	Check that no song longer than 7 minutes is displayed in search results (p8)	Invalid	Artist: Deadmau5	Search result module displays only radio edit versions of songs by Deadmau5	Search result module displays only radio edit versions of songs by Deadmau5	Yes	None
				Artist: Daft Punk	Search result module displays only radio edit versions of songs by Daft Punk	Search result module displays only radio edit versions of songs by Daft Punk		
18		Check that songs just over 7 minutes long are rejected	Borderline	Track name: When the levee breaks Artist: Led Zeppelin	Search result module displays only shorter versions of When the Levee Breaks by Led Zeppelin	Search result module displays shorter cover versions of When the Levee Breaks	Yes	None
				Track name: Around the World Artist: Daft Punk	Search result module displays only radio edit versions of Around the World by Daft Punk	Search result module displays only radio edit versions of Around the World by Daft Punk		

19	P3 (Song)	Check that songs just under 7 minutes long are accepted	Borderline	Track name: Ghosts 'n' Stuff	Search result module displays Nero remix but not original song	Search result module displays Nero remix but not original song	Yes	None
				Track name: Musique Artist: Daft Punk	Search result module displays Musique by Daft Punk	Search result module displays Musique by Daft Punk		
20		Check that no explicit songs are displayed in search results (p7)	Invalid	Track name: Alphabet Artist: Lil Wayne	Search result module does not open and alert states that no results were found	Search result module does not open and alert states that no results were found	Yes	None
				Track name: Oh my Darling Don't Cry Artist: Run the Jewels	Search result module does not open and alert states that no results were found	Search result module displays Oh my Darling Don't Cry by Run the Jewels		No changes to be made, Spotify does not class song as explicit
21		Check that songs already in queue are rejected	Invalid	Track name: Don't Look Back into the Sun Artist: The Libertines	Song not added to queue and alert states that song already requested	Song not added to queue and alert states that song already requested	Yes	None

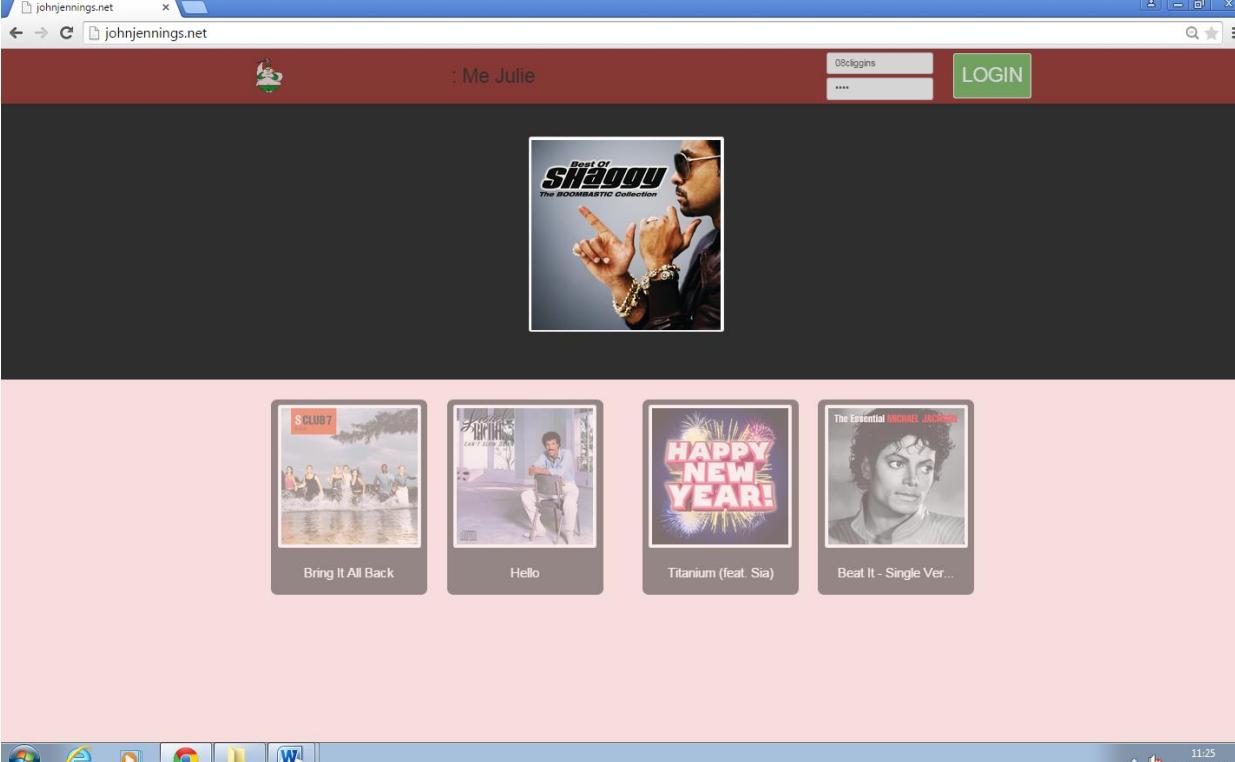
21	P3 (Song)	Check that songs already in queue are rejected	Invalid	Track name: T-Shirt Weather Artist: Circa Waves	Song not added to queue and alert states that song already requested	Song not added to queue and alert states that song already requested	Yes	None
22		Check that songs similar to songs already in queue are accepted	Borderline	Track name: Hello Artist: J. Cole	Song added to queue	Song added to queue	Yes	None
				Track name: Can't Stand me Now Artist: The Libertines				
23		Check that requests are rejected if sent within cooldown time (p10)	Invalid	Send two song requests consecutively	First request accepted, second request rejected with alert stating still in cooldown	First request accepted, second request rejected with alert stating still in cooldown	Yes	None
24	P4 (Vote)	Check that vote requests are added to database when vote button pressed	Normal	Username: 08cliggins Vote button pressed	Vote added to database and vote button disabled	Vote added to database and vote button disabled	Yes	None
				Username: 08amaher Vote button pressed				
				Username: 08cpreece Vote button pressed				
25		Check that duplicate votes are rejected	Invalid	Username: 08cliggins Vote button pressed twice	Vote not added to database	Vote not added to database	Yes	None
				Username: 08amaher Vote button pressed twice				
				Username: 08cpreece Vote button pressed twice				

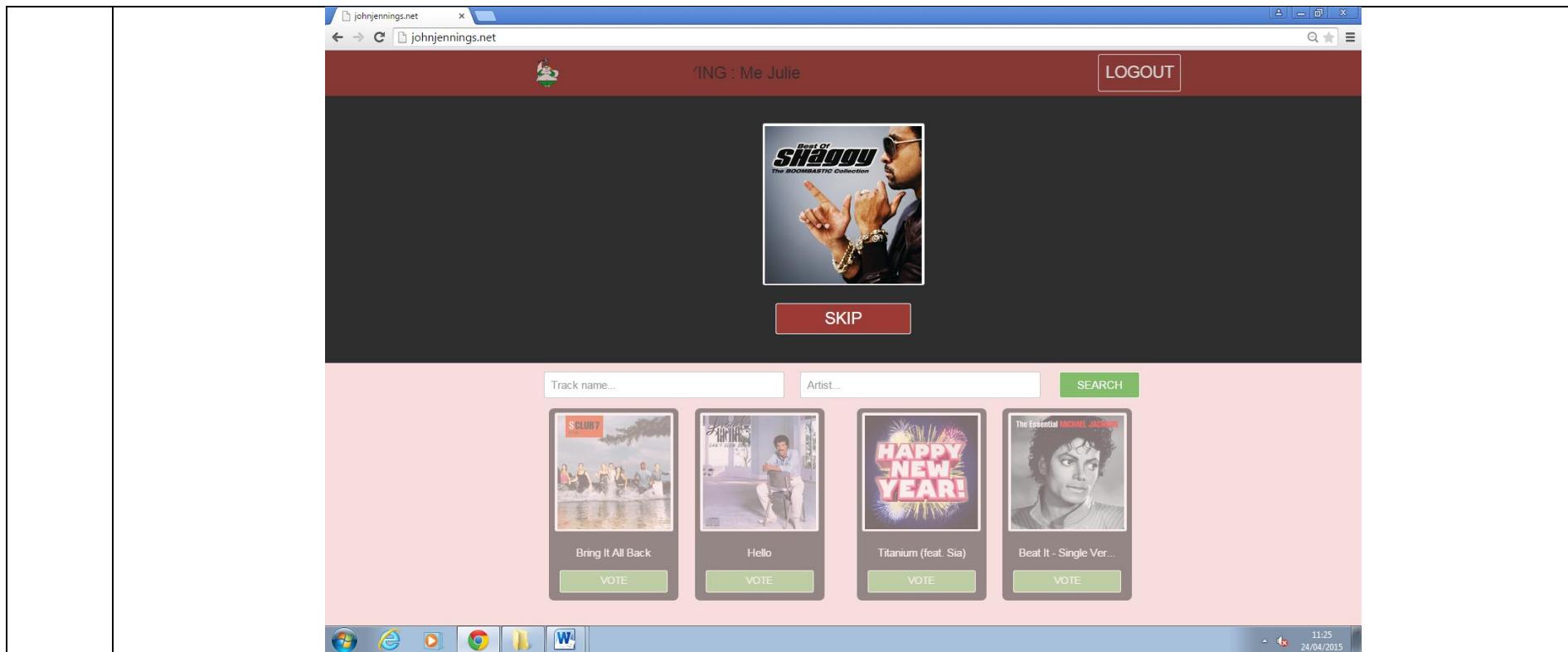
26	P5 (Skip)	Check that user is flagged as skipped when skip button pressed	Normal	Username: 08cliggins Vote button pressed	User flagged as skipped in database, skip button disabled	User flagged as skipped in database, skip button disabled	Yes	None
27				Username: 08amaher Vote button pressed twice				
				Username: 08cpreece Vote button pressed				
27	P6 (Next Song)	Check that duplicate skip requests are rejected	Invalid	Username: 08cliggins Vote button pressed twice	Skip request not added to database	Skip request not added to database	Yes	None
28				Username: 08amaher Vote button pressed twice				
29				Username: 08cpreece Vote button pressed twice				
28	P6 (Next Song)	Check that songs are played in order of queue	Normal	Open player	Spotify plays Smoke on the Water followed by Me Julie and Bring it All Back	Spotify plays Smoke on the Water followed by Me Julie and Bring it All Back	Yes	None
29		Check that songs are played in their entirety	Normal	Open player	Spotify plays a new song only when the current song has ended	Spotify plays a new song only when the current song has ended	Yes	None
30		Check that songs are played without gaps	Normal	Open player	Spotify plays a new song immediately after current song has ended	Spotify plays a new song 3-5 seconds after current song has ended	No	Reduced track length by 3 seconds to allowed for processing between songs

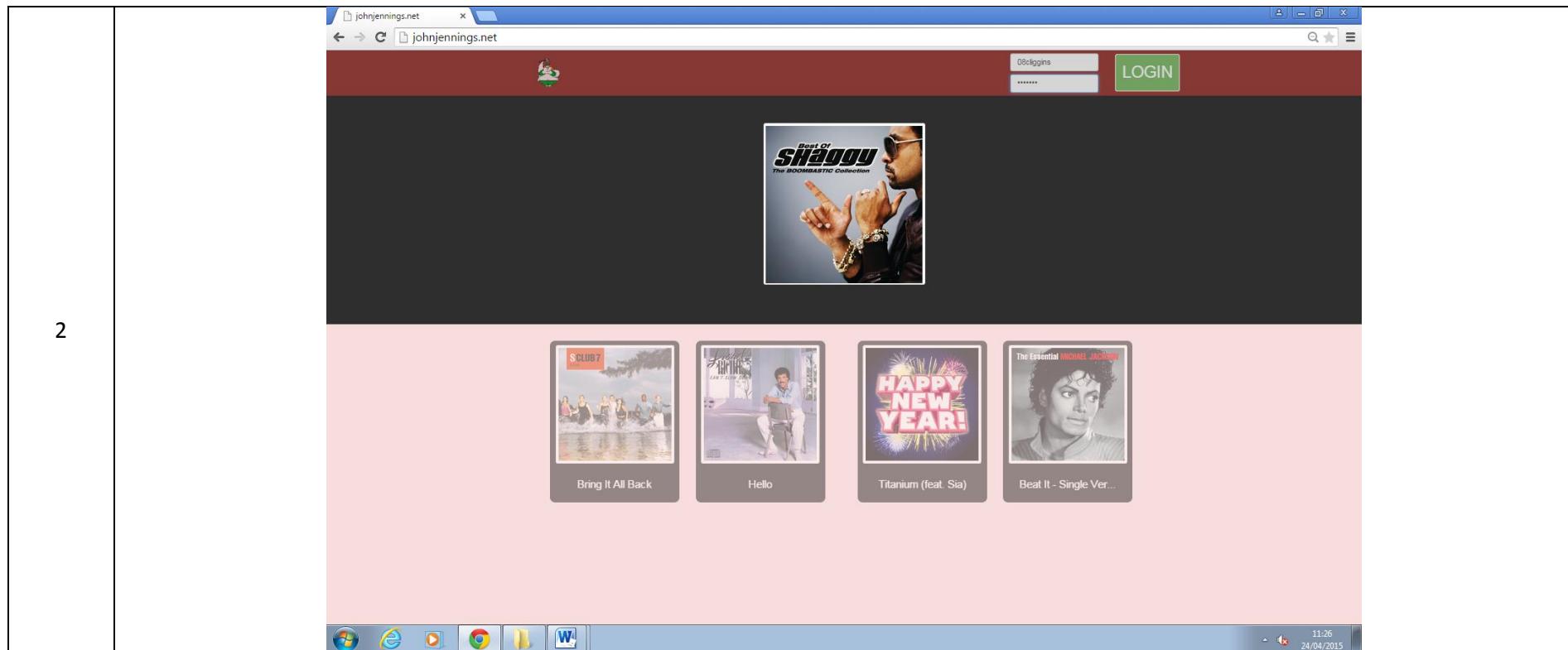
31	P6 (Next Song)	Check that song is skipped if >60% of active users have voted to skip (p5)	Normal	Set 100 users as active, 60 as skipped. Press skip button	Current song ends and next song begins playing	Current song ends and next song begins playing	Yes	None	
			Borderline	Set 100 users as active, 59 as skipped. Press skip button	Current song continues playing and skip button disabled	Current song continues playing and skip button disabled	Yes	None	
32		Check that songs move forward in queue depending on how many votes they have received (p4)	Normal	Set 100 users as having voted for song at bottom of queue	When current song ends, song at bottom of queue moves forward	When current song ends, song at bottom of queue moves forward	Yes	None	
33		Check that certain users have more influence in the voting algorithm (p6)	Normal	Set 1 user with weight 500 as having voted for song at bottom of queue, 100 users with weight 1 as having voted for second song from bottom of queue	When current song ends, both songs move forward in queue with first song now above second	When current song ends, both songs move forward in queue with first song now above second	Yes	None	
				Set 5 users with weight 10 as having voted for song at bottom of queue, 20 users with weight 2 as having voted for second song from bottom of queue					

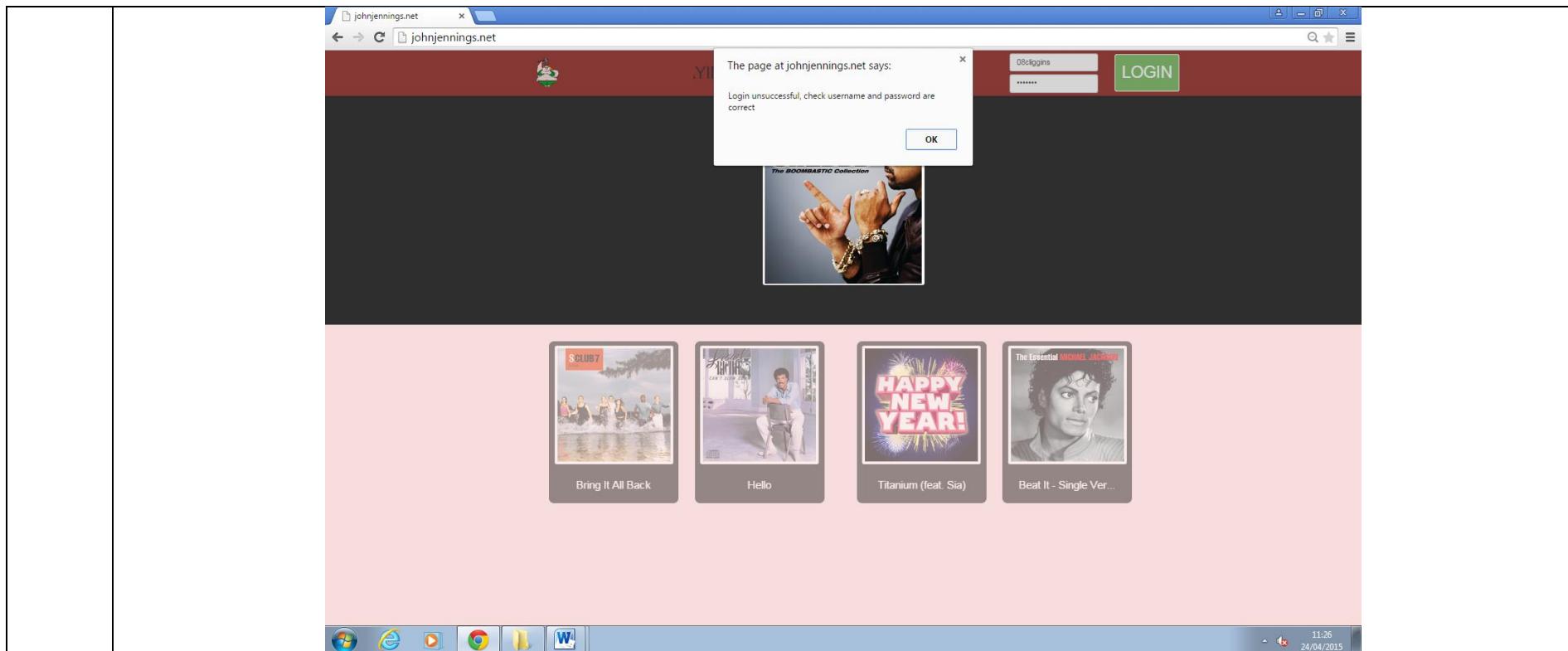
33	P6 (Next Song)	Check that certain users have more influence in the voting algorithm	Borderline	Set 10 users with weight 50 as having voted for song at bottom of queue, 9 users with weight 50 as having voted for second song from bottom of queue	When current song ends, both songs move forward in queue with first song now above second	When current song ends, both songs move forward in queue with first song now above second	Yes	None
34		Check that player returns to start of queue when final song ends (p9)	Normal	Open player	When Lovesick ends, Smoke on the Water begins	When Lovesick ends, Smoke on the Water begins	Yes	No
35		Check that song is removed from queue when skipped	Normal	Set 100 users as active, 60 as skipped. Press skip button	Current song ends and is removed from database	Current song ends and is removed from database	Yes	No
36		Check that users are flagged as having not skipped when next song begins playing	Normal	Open player, wait for first song to end	All users' skipped value is set to 0 in database	All users' skipped value is set to 0 in database	Yes	No

Alpha Evidence

Test Number	Screenshot
1	







3

NOW PLAYING : Me Jul

invalid
.....

LOGIN

Best of Shaggy
The BOOMBASTIC Collection

S CLUB 7
Bring It All Back

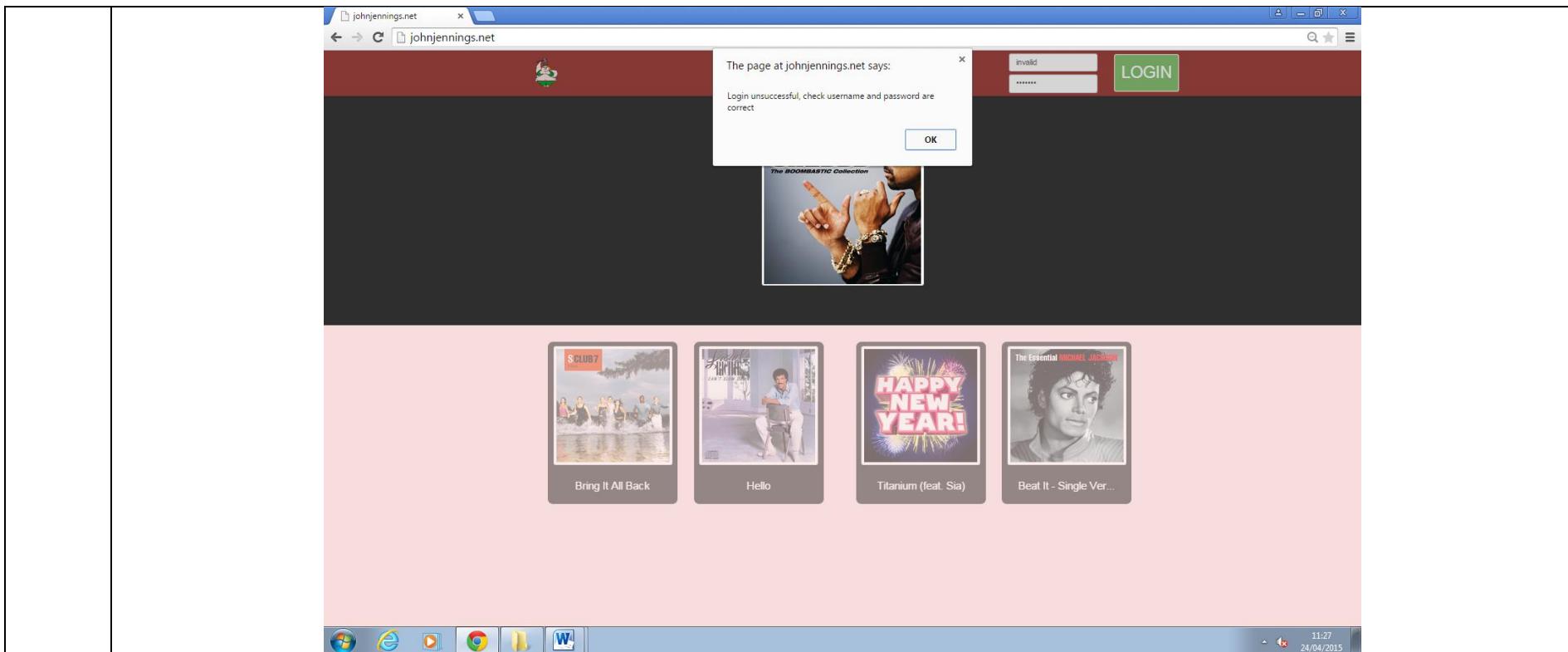
Hello

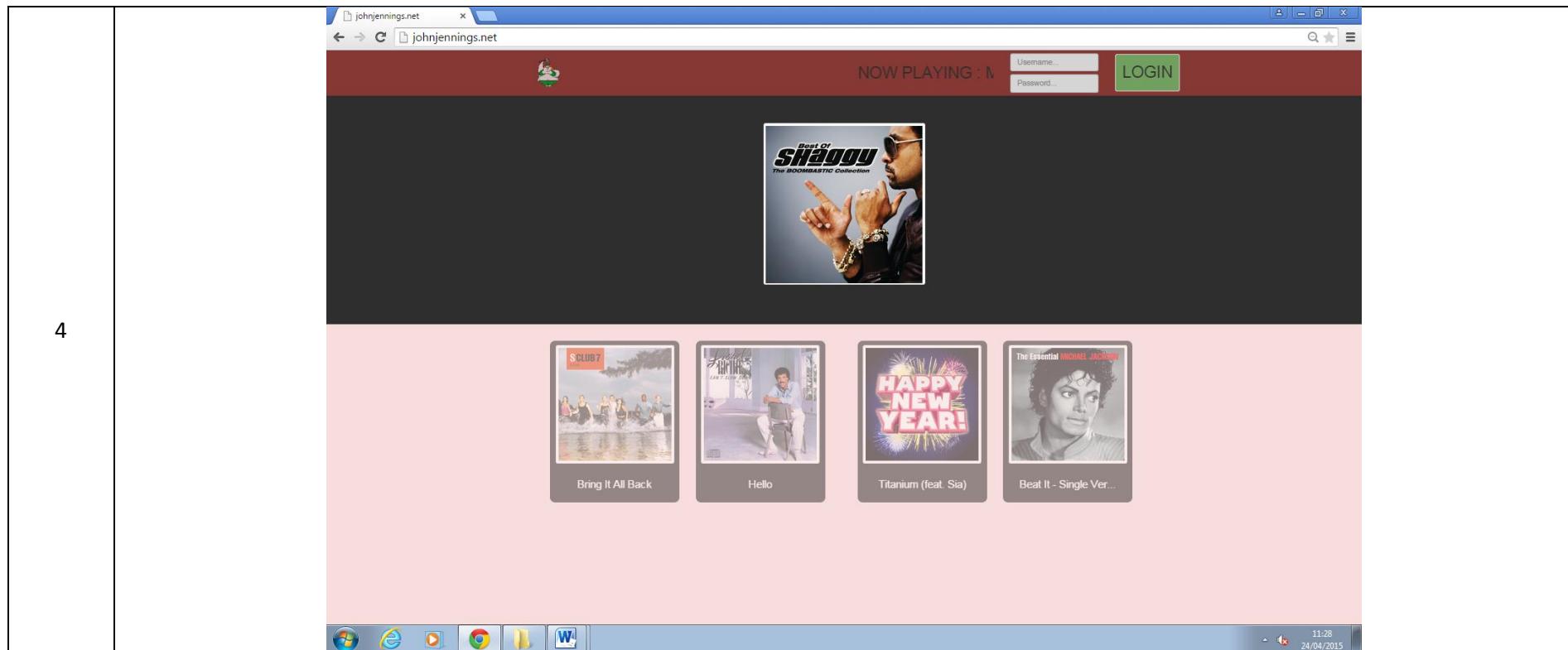
HAPPY NEW YEAR!

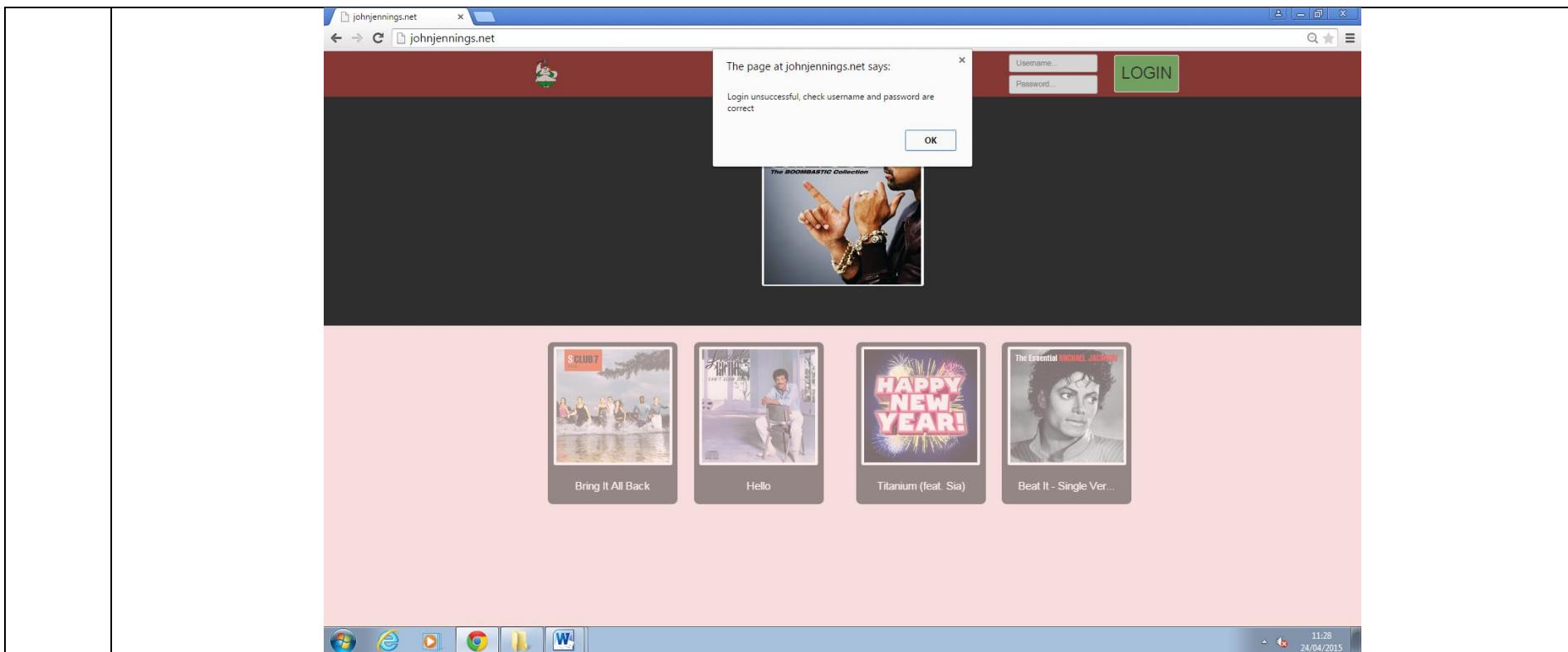
Titanium (feat. Sia)

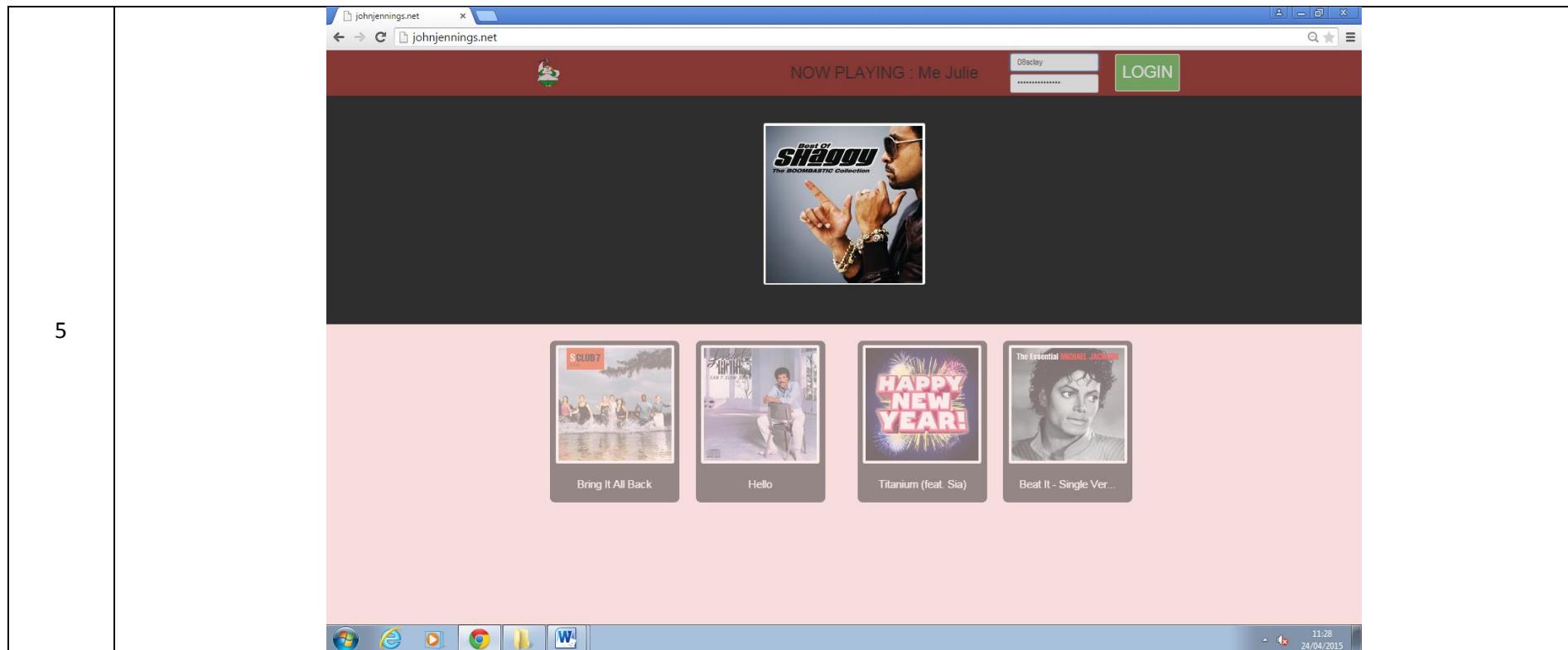
The Essential MICHAEL JACKSON
Beat It - Single Ver...

11:27
24/04/2015

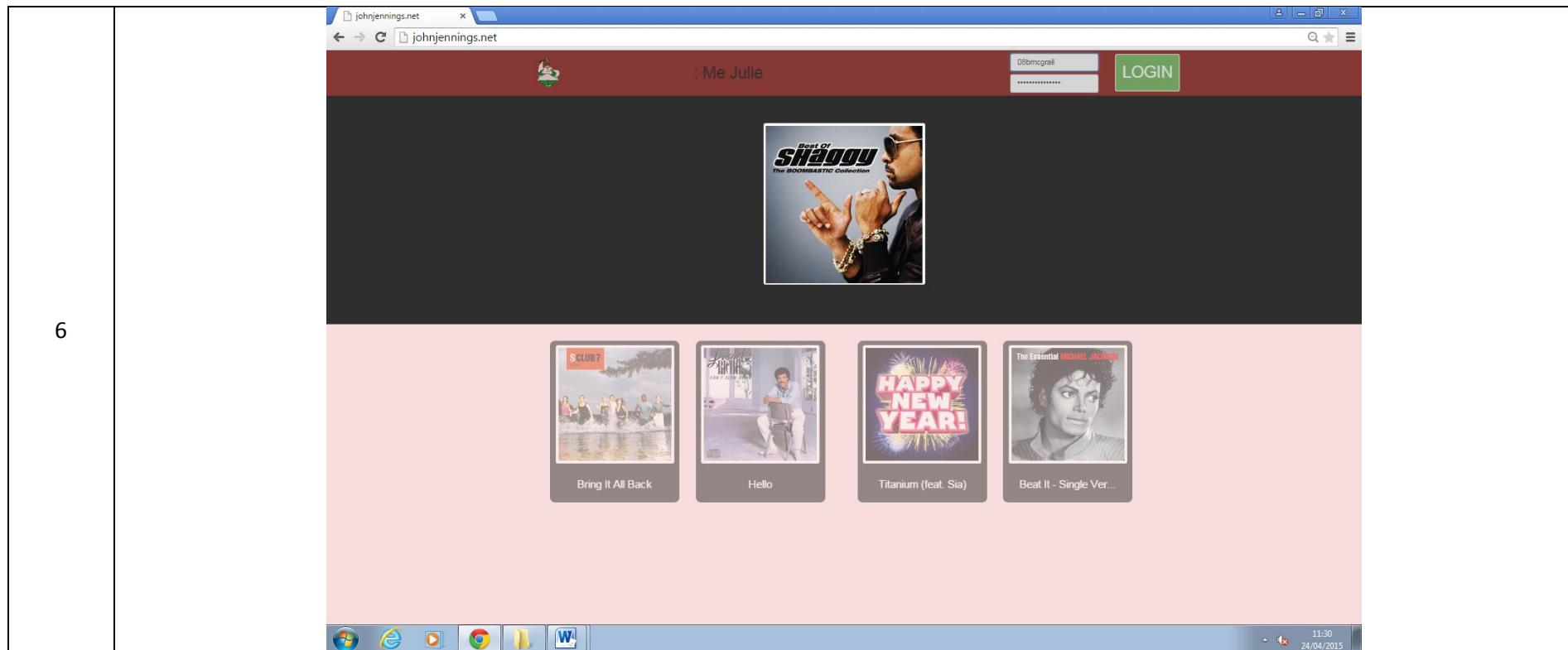


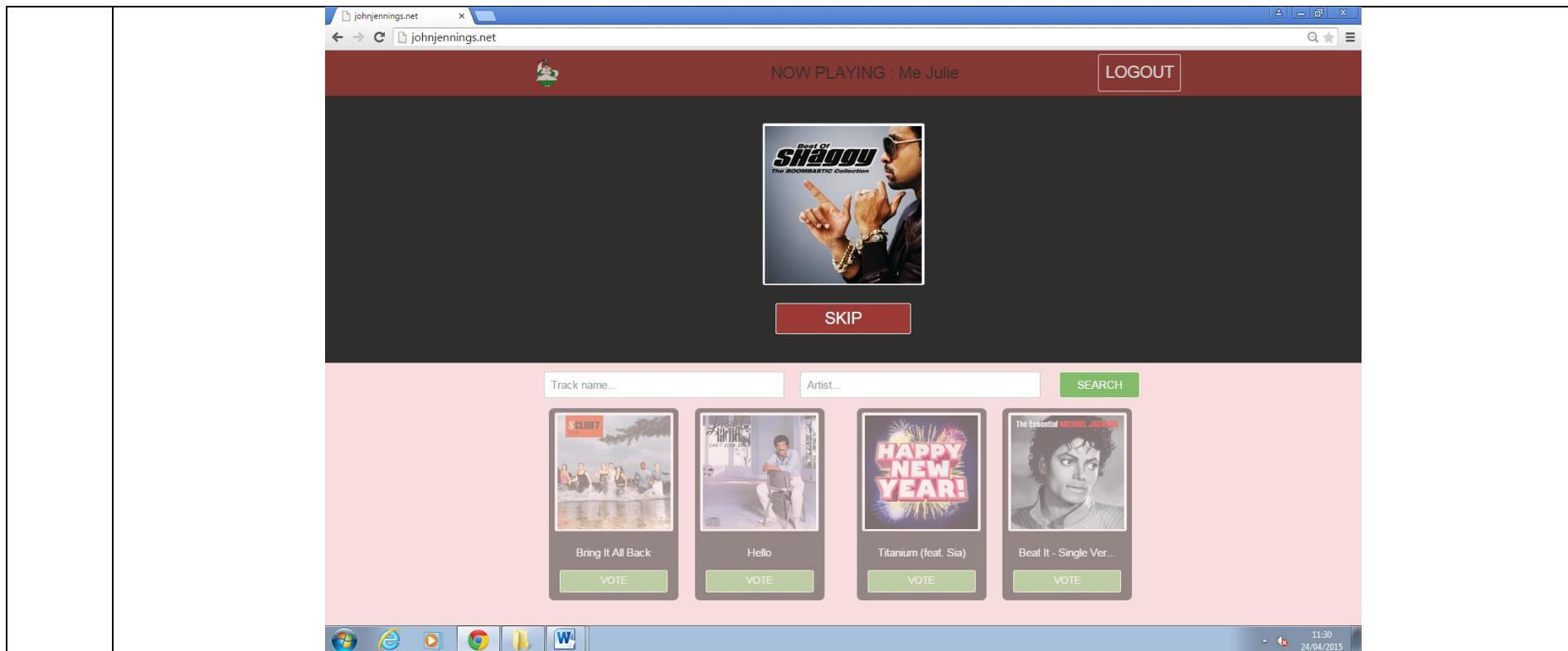






The screenshot shows a web browser window for 'johnjennings.net'. The title bar displays the URL 'johnjennings.net'. The main content area has a dark header bar with a small profile icon on the left, the text 'NOW PLAYING : Me' in the center, and a 'LOGOUT' button on the right. Below this is a large image of Shaggy from his album 'Best of Shaggy: The BOOMBASTIC Collection'. A red 'SKIP' button is centered below the image. At the bottom of the page is a pink search bar with fields for 'Track name...' and 'Artist...', a 'SEARCH' button, and a track listing section. This section contains four tracks with album art, track names, and 'VOTE' buttons: 'S Club 7 - Bring It All Back', 'Hello by Adele', 'HAPPY NEW YEAR!' (featuring Sia), and 'The Essential Michael Jackson - Beat It - Single Version'. The bottom of the screen shows a Windows taskbar with icons for Internet Explorer, Google Chrome, and Microsoft Word, along with system status icons.



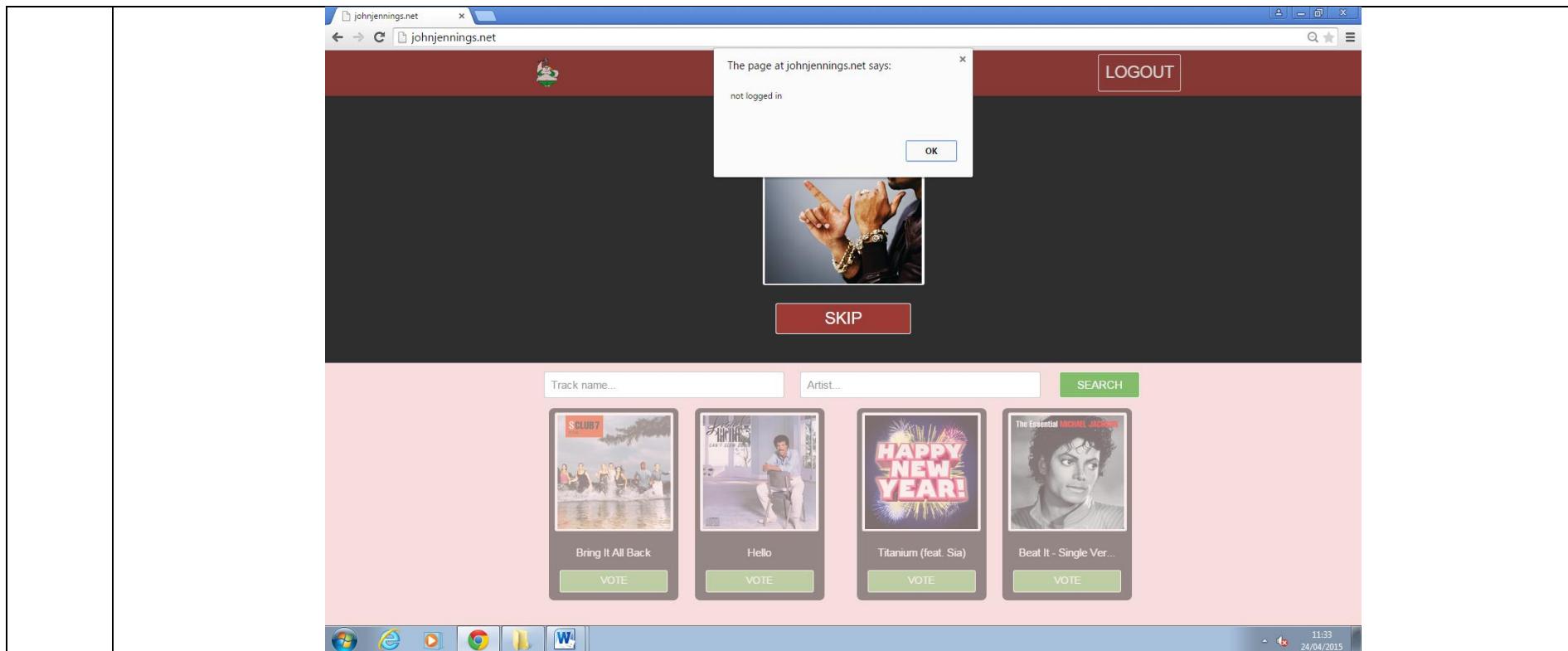


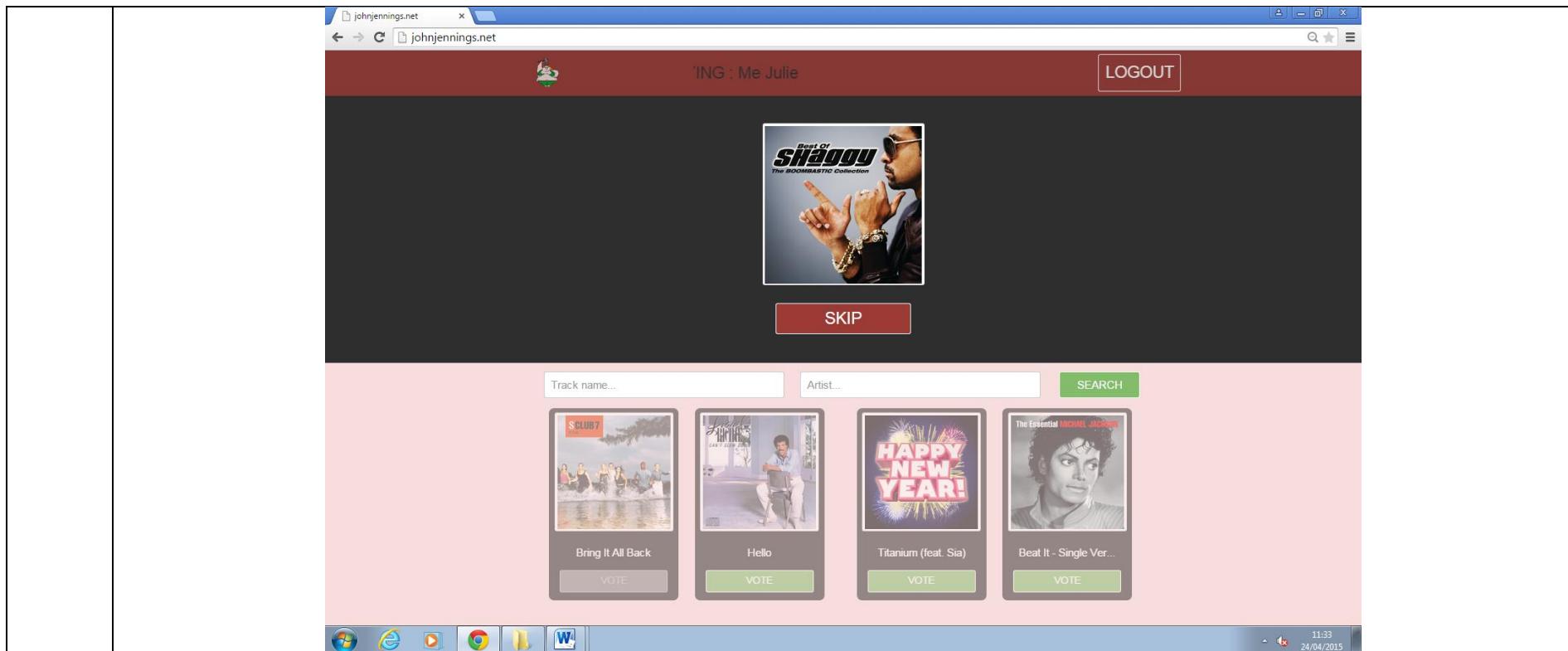
7

The screenshot shows a web browser window with the URL johnjennings.net. The page has a dark theme with a red header bar. In the header, there is a small user icon, the name "Me Julie", and a "LOGOUT" button. Below the header, there is a large image of Shaggy from the album "Best Of Shaggy The Bodastik Collection". A "SKIP" button is located below the image. At the bottom of the page, there is a search bar with the word "test" in it, followed by another search bar labeled "Artist...", a "SEARCH" button, and a "Test Drive" section. The "Test Drive" section displays an album cover for "How To Train Your DRAGON" by John Powell. To the right of the album cover, there are two buttons: a green "YES" button and a red "NO" button. The bottom of the screen shows a Windows taskbar with various icons and the system clock indicating 11:31 on 24/04/2015.

8

The screenshot shows a web browser window with the URL johnjennings.net. On the left, there is a search interface with a placeholder 'test' in a text input field and a 'SEARCH' button below it. Below the search bar is a small image of a movie poster for 'The Best of Shaggy'. A 'SKIP' button is located just above the poster. On the right side of the screen, the developer tools are open, specifically the 'Resources' tab under the 'Local Storage' section. It displays a table with one entry: 'username' with a value of 'OBfmacKlin'. At the bottom of the developer tools, the 'Console' tab is active, showing a single error message: 'Failed to load resource: the server responded with a status of 404 (Not Found) http://johnjennings.net/favicon.ico'. The browser's taskbar at the bottom shows various pinned icons.





9

The screenshot shows the phpMyAdmin interface for the 'User' table in the 'Big_Julie' database. The table has columns: username, password, loggedIn, active, lastReqTime, voteWeight, and skipped. There are 14 rows of data. The last row is selected.

	username	password	loggedIn	active	lastReqTime	voteWeight	skipped
1	08bateman	57KITSCH338	0	0	0	1.00	0
2	08Tcooke	290FLUS627	0	0	0	1.00	0
3	08tcummings	LOBESUTAS0	0	0	0	1.00	0
4	08tmackin	89074SEARCH	0	3	0	1.00	0
5	08troberts	2049	1	10	0	1.00	0
6	08vadcock	26822927	0	2	0	1.00	0
7	08zdalton	060236594480	0	0	0	1.00	0
8	08zmadden	34534634	0	0	0	1.00	0
9	09clancy	what	1	0	0	4.99	0
10	julie	fgdfgsdfgdfhgh	0	0	0	0.00	0
11	test1	123456789	1	0	1424896994356	9.99	0
12	Test2	testy	1	3	1429193680418	9.99	1
13	test3	testy	1	0	1424894530314	9.99	0

Number of rows: 25 Filter rows: Search this table

Query results operations:

- Print view
- Print view (with full texts)
- Export
- Display chart
- Create view

11:36
24/04/2015

10

The screenshot shows a web browser window with a dark theme. At the top, there's a navigation bar with tabs for 'johnjennings.net' and 'Databases - Parallels Plesk'. The main content area has a red header bar with the text 'PLAYING: Me Julie' and a 'LOGOUT' button. Below this is a large image of Shaggy from his album 'Best Of Shaggy The BOOMTASTIC Collection'. A red 'SKIP' button is centered below the image. In the middle section, there's a search bar with two input fields: one containing 'hello' and another containing 'lionel richie', followed by a green 'SEARCH' button. Below the search bar, there's a result card for 'Hello' by Lionel Richie, featuring a photo of him sitting in a chair. To the right of the result card are two large buttons: a green 'YES' button and a red 'NO' button. At the bottom of the screen, there's a taskbar with icons for various applications like Internet Explorer, Google Chrome, and Microsoft Word. The system tray shows the date and time as '11:37 24/04/2015'.

11

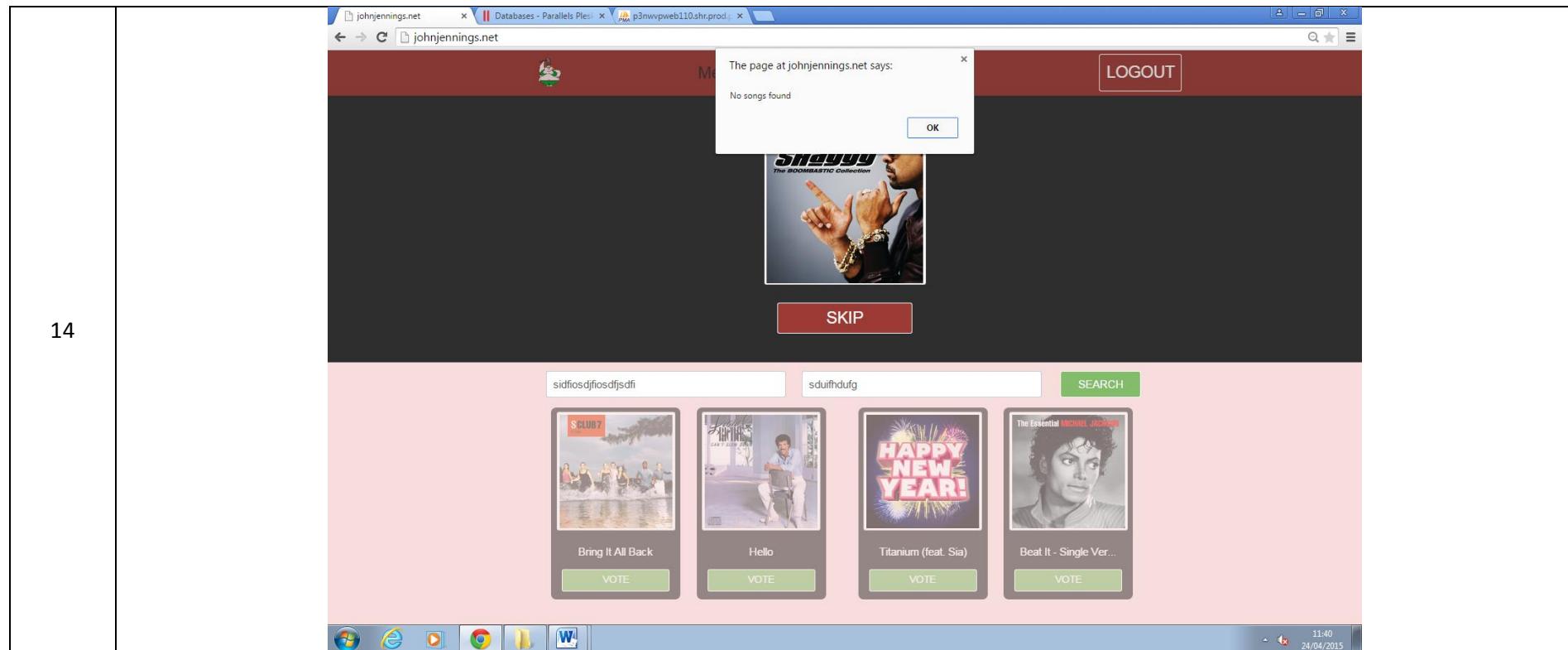
The screenshot shows a web-based application interface. At the top, there's a dark red header bar with the text '[JOHN JENNINGS]' on the left and 'C2: Testing' on the right. Below this is a browser window with three tabs: 'johnjennings.net', 'Databases - Parallels Ples', and 'p3nwvypweb110.shr.prod...'. The main content area has a dark background. In the center, there's a small image of a person wearing sunglasses and a gold chain, with the text 'Best Of SHAGGY The BOOMBASTIC Collection' above it. Below this image is a red 'SKIP' button. To the right of the image is a 'LOGOUT' button. At the bottom of the screen, there's a pinkish-red footer bar with a search bar containing 'hello' and 'Artist...', a 'SEARCH' button, and two large buttons labeled 'YES' and 'NO'. The footer also features several small album covers and logos for S CLUB 7, Kylie Minogue, and Michael Jackson. The system tray at the bottom shows icons for the Start button, Internet Explorer, File Explorer, Google Chrome, and Microsoft Word, along with a date and time stamp: '11:38 24/04/2015'.

12

The screenshot shows a web-based music player interface. At the top, there's a red header bar with the title '[JOHN JENNINGS]' on the left and 'C2: Testing' on the right. Below this is a dark grey header with a logo on the left, 'NOW PLAYING : Mi...' in the center, and a 'LOGOUT' button on the right. The main content area features a large image of Shaggy from his 'Best Of Shaggy The BOOMTASTIC Collection' album. Below the image is a red 'SKIP' button. Underneath the album art, there are two search input fields: 'Track name...' containing 'tiny tim' and a 'SEARCH' button. To the right of these fields is a green 'YES' button and a red 'NO' button. A preview image of 'GOD BLESS TINY TIM' by Tiny Tim is shown. The track information 'Tip Toe Thru' The Tulips With Me (Remastered Version)' and 'Tiny Tim' is displayed above the preview image. At the bottom of the screen, there's a blue taskbar with various icons (Windows, Internet Explorer, etc.) and a system tray showing the date and time (24/04/2015, 11:39).

13

The screenshot shows a web browser window with the URL johnjennings.net. The title bar also displays "Databases - Parallels Plesk" and "p3nwvypweb110.shr.prod". The main content area has a red header bar with the text "/ PLAYING : Me Julie" and a "LOGOUT" button. Below this is a dark gray section featuring a thumbnail of Shaggy's album "Best Of Shaggy: The Bodomastix Collection" and a "SKIP" button. At the bottom of this section is a pink search bar with fields for "smoke" and "Artist...", and a "SEARCH" button. To the right of the search bar are two large buttons: a green "YES" button and a red "NO" button. In the center of the pink area is a thumbnail for Deep Purple's album "DEEPEST PURPLE" with the title "Smoke On The Water - 1971 Recording" and the artist "Deep Purple". Below the search bar, there are several small album thumbnails, including "S CLUB 7", "LADY GAGA", "The Essential MICHAEL JACKSON", and others. The bottom of the screen shows a Windows taskbar with icons for Internet Explorer, Google Chrome, and Microsoft Word, along with the system clock (11:39) and date (24/04/2015).



15

15

ID	Name	Song	URL	Length	ID
1	Me Julie	All I G	https://i.scdn.co/image/62c82b8b842b95b8c27b802838...	225293	1428695620553
2	S Club 7	Bring It All Back	https://i.scdn.co/image/68e5f2f7edb1edcf4eec879f9...	213600	1428695692430
3	Lionel Richie	Hello	https://i.scdn.co/image/297c3872cefa48afea08456e...	250200	1428695849552
4	David Guetta	Titanium (feat. Sia)	https://i.scdn.co/image/2c7c85408140ec370f3d5e3c95...	245053	1428695866161
5	Michael Jackson	Beat It - Single Version	https://i.scdn.co/image/c4f2f8fcbe714dd8d9a5c2af56...	258040	1429193680418
6	Tiny Tim	Tip Toe Thru' The Tulips With Me (Remastered Version)	https://i.scdn.co/image/3eb4dc2845501c549a3e8a5346...	110613	1429872070149
7	Catfish and the Bottlemen	Homesick	https://i.scdn.co/image/62ebe3da2b38ae52a23957ad7...	148092	101427450548571
8	Circa Waves	Stuck In My Teeth	https://i.scdn.co/image/fb896cbd09851b24a2bf7b7087...	183874	101427450674350
9	Circa Waves	T-Shirt Weather	https://i.scdn.co/image/56644972dfa1f5761ac47bf8be...	194288	101427450929763
10	The Libertines	Don't Look Back Into The Sun	https://i.scdn.co/image/abe4af82b7ff01c6f3b6e23f6...	178266	101427451108811
11	The Libertines	Can't Stand Me Now	https://i.scdn.co/image/9434065bf0fc7a7a6d280a198d...	203733	101427451314198
12	Wolf Alice	Giant Peach	https://i.scdn.co/image/5454a2165badd92bf625f34f6b...	275493	101427451590973
13	Arctic Monkeys	R U Mine?	https://i.scdn.co/image/d49ec146e3a257b10634d9a413...	201726	101427451797210
14	Circa Waves	Young Chasers	https://i.scdn.co/image/da0865476d7b45dc45937c01eb...	130132	101427451930645
15	Peace	Bloodshake	https://i.scdn.co/image/646e39793d24e3080d756b5a7...	235586	101427452171091
16	JAWS	Gold	https://i.scdn.co/image/a6bf0ff7e3010aa8f9c2161a...	238918	101427452414643

17

The screenshot shows a web browser window with the URL johnjennings.net. The page has a dark theme with a red header bar. In the header, there is a small logo, a 'NOW PLAYING' button, and a 'LOGOUT' button. Below the header, there is a large image of Shaggy from the album 'Best Of Shaggy'. A red 'SKIP' button is positioned below the image. At the bottom of the screen, there is a search bar with the placeholder 'Track name...' and a search input field containing 'Deadmau5'. To the right of the search input is a green 'SEARCH' button. Below the search bar, there is a track preview for 'deadmau5 - The Veldt - Radio Edit' by deadmau5. To the right of the track preview are two large buttons: a green 'YES' button and a red 'NO' button. The bottom of the screen features a blue navigation bar with various icons, including a Windows logo, Internet Explorer, Google Chrome, Microsoft Word, and others. The date and time '24/04/2015 11:44' are visible in the bottom right corner.

18

NOW PLAYING : Me Julie

LOGOUT

SKIP

Around the World

Artist...

SEARCH

Best Of Shaggy
The BOOMTASTIC Collection

Around The World - Radio Edit
Daft Punk

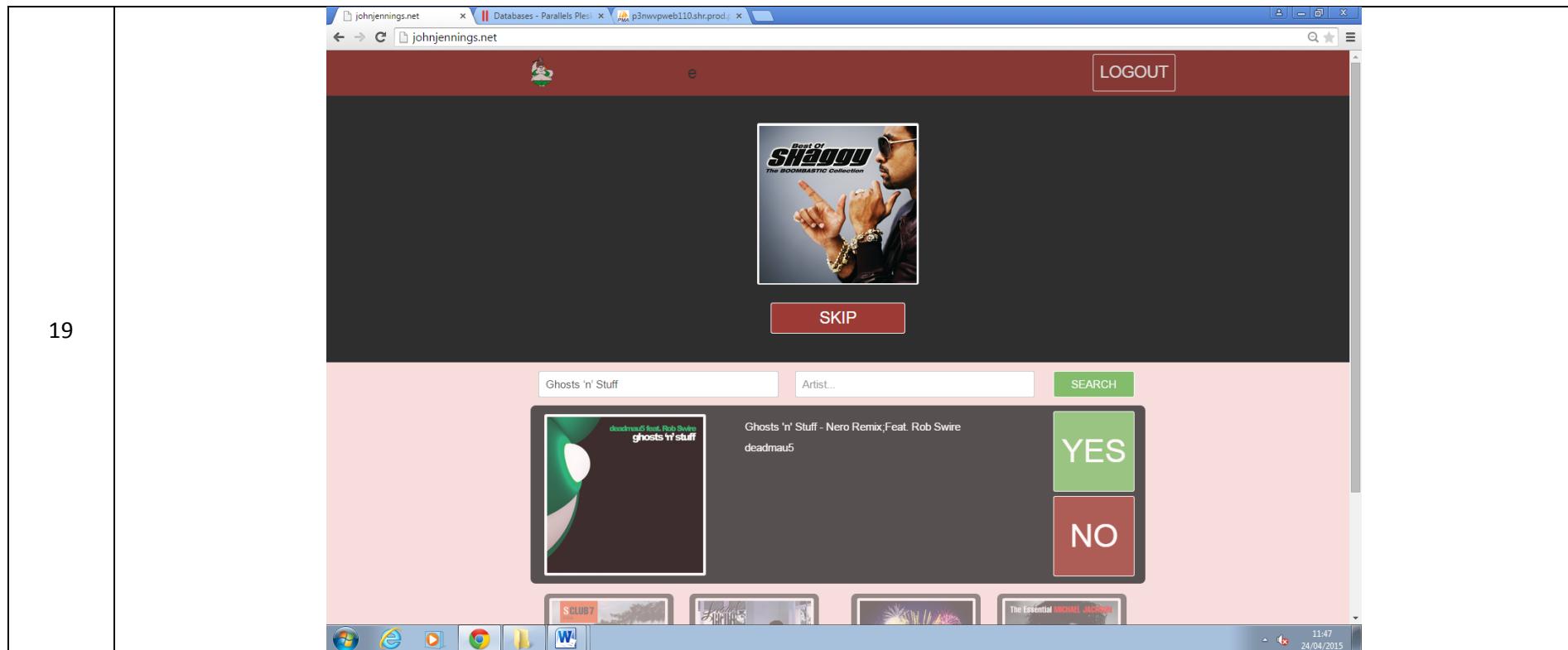
YES

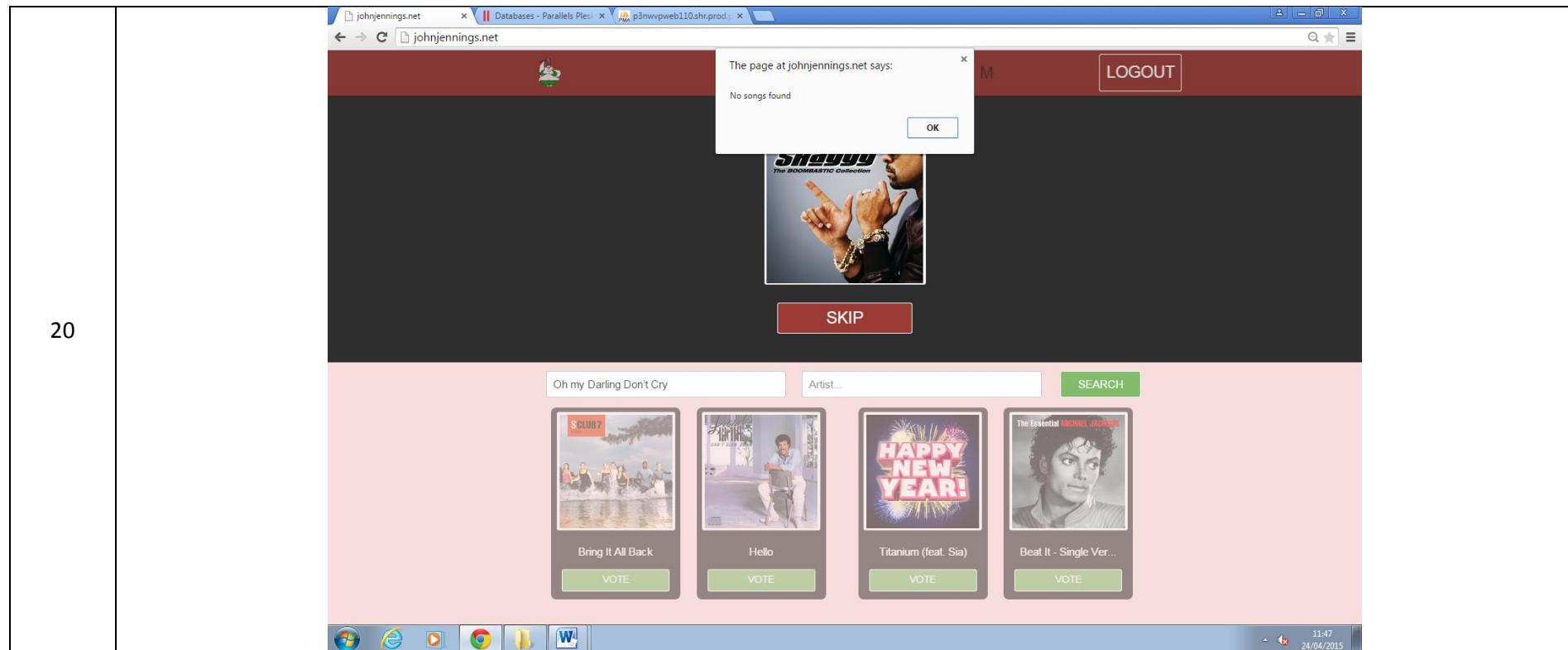
NO

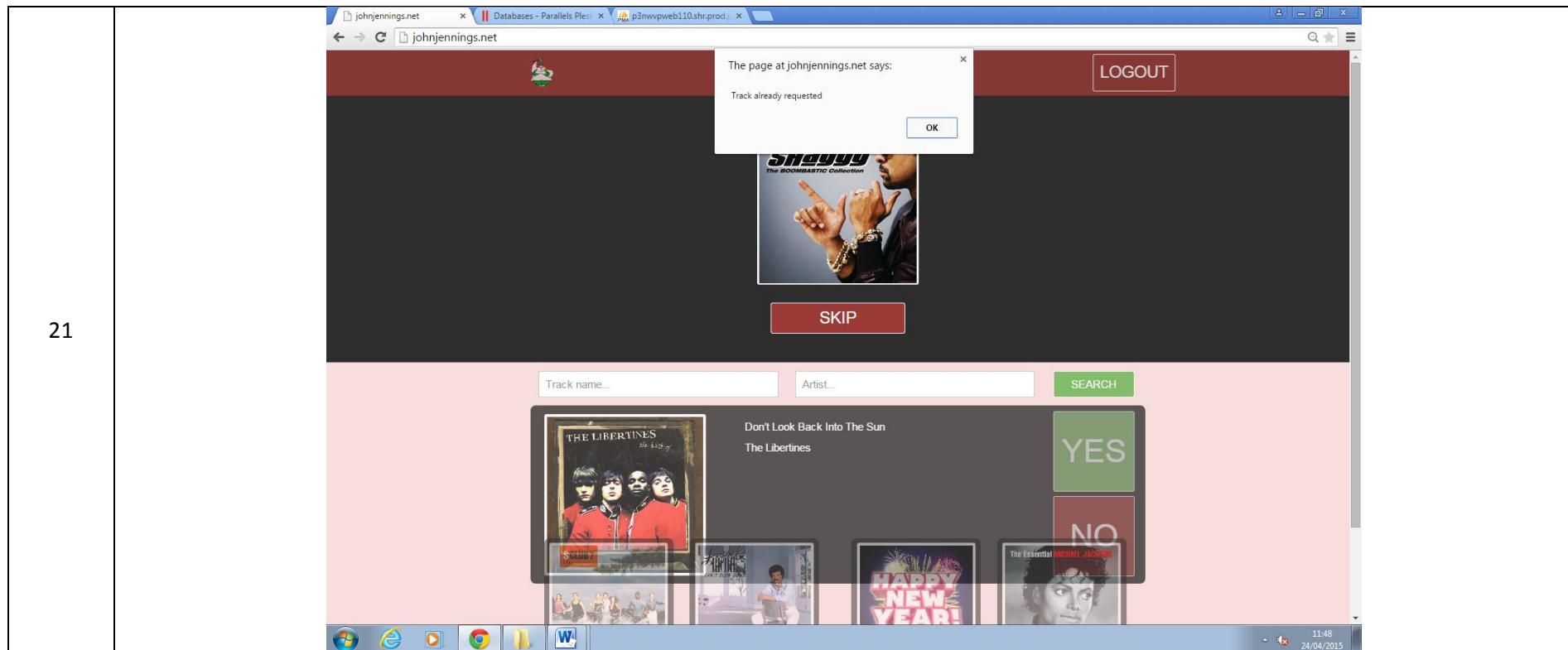
Daft Punk
MUSIQUE VOL. I 1993 - 2005

S CLUB 7 | Kylie | Daft Punk | The Essential MICHAEL JACKSON

11:45
24/04/2015

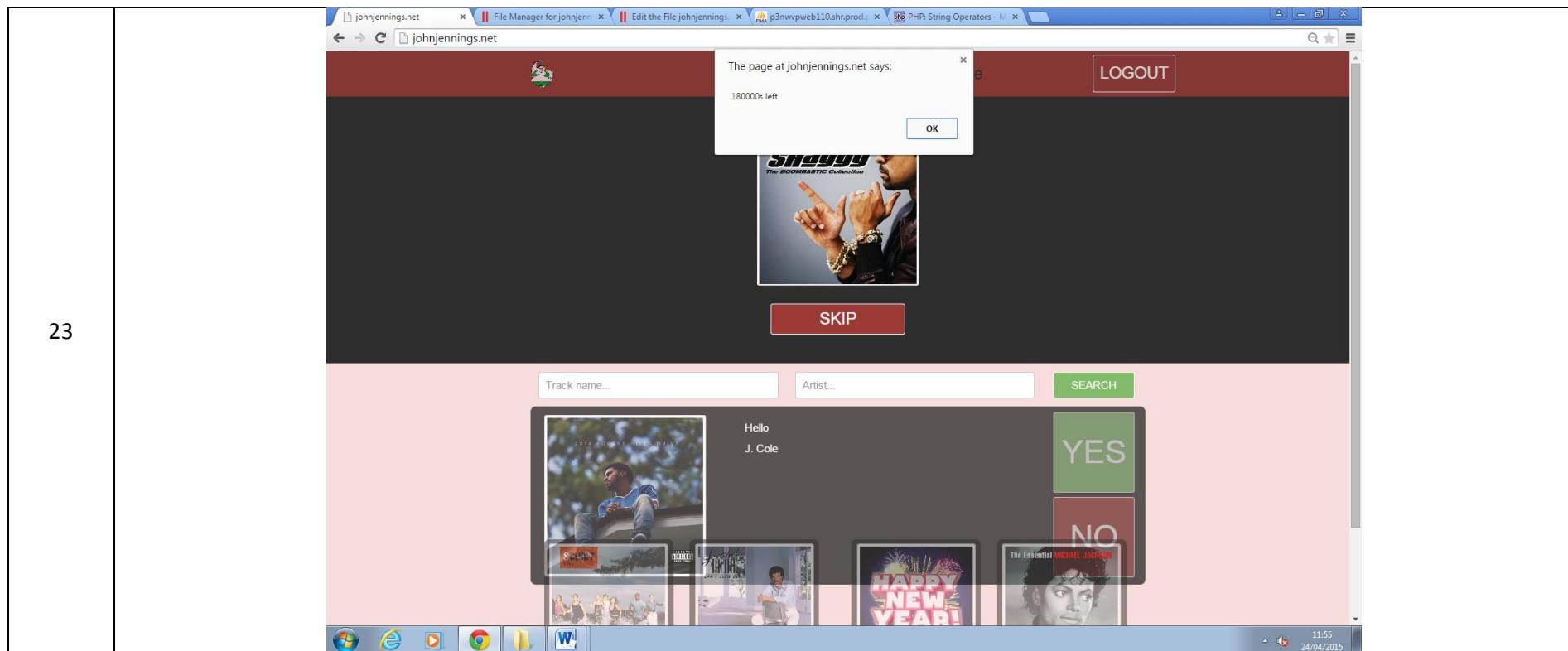






22

URL	name (Remastered Versi...)	artist	art	length	req
it 5EZImpJljQXxfqA9RuCikf	Homesick	Catfish and the Bottlemen	https://i.scdn.co/image/62efbe3da2b38ae52a23957ad7...	148092	10+
it 4qIMtMS6QlqSxOTvN6PqiS	Stuck In My Teeth	Circa Waves	https://i.scdn.co/image/fb896cbd09851b24a2bf7b7087...	183874	10+
it 2EPwh1bWUcre7x8l00Eaky	T-Shirt Weather	Circa Waves	https://i.scdn.co/image/56644972dfa1f5761ac47bf8e...	194288	10+
it 3M0YwlWNPNX8cfWQjYsXTpu	Don't Look Back Into The Sun	The Libertines	https://i.scdn.co/image/abe4af82b78ff01c6f3b6e23f6...	178266	10+
it 7D4LseR0RADPbyyMeTiHR	Can't Stand Me Now	The Libertines	https://i.scdn.co/image/9434065bf0fbca7a6d280a198d...	203733	10+
it 7DJemidOSadY7j7Bv8HKV3	Giant Peach	Wolf Alice	https://i.scdn.co/image/5454a2165badd92bf625f34f6b...	275493	10+
it 2LrvrdnLa3XbB1b4jYuCnl	R U Mine?	Arctic Monkeys	https://i.scdn.co/image/4d9ec146e3a257b10634d9a413...	201726	10+
it 5Yxb116LxhxQ6mTVuDpk	Young Chasers	Circa Waves	https://i.scdn.co/image/da0865476d7b45dc45937c01eb...	130132	10+
it 1aB3C7GKpCFgsdhqzIZWOU	Bloodshake	Peace	https://i.scdn.co/image/646e39793d240e380df756b5a7...	235586	10+
it 5Ahov76gqMWgbLKjuc9FB	Gold	JAWS	https://i.scdn.co/image/a6bf0ff177e3010aa8f9c2181a...	238918	10+
it 49ccDOSipalJ4PcmovJy4h	Recovery	Frank Turner	https://i.scdn.co/image/b2b320355e1d3b948181e5fd3a...	208040	10+
it 3AIKWmEGeQze4nrw5krTY0	Chlorine	The Districts	https://i.scdn.co/image/c7abf7a3c3722b87a1e7784030...	241680	10+
it 4semBrqop202J6V0iU0NRc	A Rush of Blood	Coasts	https://i.scdn.co/image/68bd1c31196aaed1f2aae41e4e...	248082	10+
it 5FkJQLJukvrrtmi1QlgmzY	Be Slowly	JAWS	https://i.scdn.co/image/a6bf0ff177e3010aa8f9c2181a...	163347	10+
it 1cFwUMRIsE6jEMRUDyBONA	Rabbit Hole	Jamie T	https://i.scdn.co/image/a2085ce48b6279dadf3313b435...	219853	10+



24

The screenshot shows a browser window with multiple tabs open. The active tab is 'phpMyAdmin' showing the 'Vote' table from the 'Big_Julie' database. The SQL query 'SELECT * FROM `Vote`' has been run, resulting in 151 rows displayed. The columns are 'username' and 'URL'. The rows show various user names and their corresponding URLs.

	username	URL
1	08bmccgrail	1fn6EFBBNiDVhL3DxDJJTD
2	08cliggins	1fn6EFBBNiDVhL3DxDJJTD
3	08tmacklin	1fn6EFBBNiDVhL3DxDJJTD
4	john	1fn6EFBBNiDVhL3DxDJJTD
5	julie	0ClvVAXFMRuo31Dopv4e3Z
6	julie	0iPncBNh7VdW1krk41bnoU
7	julie	0l6s307JRgXEGIzh8sPdCx
8	julie	0QIYINh2AwmOmdu8CRYvlw
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
100		
101		
102		
103		
104		
105		
106		
107		
108		
109		
110		
111		
112		
113		
114		
115		
116		
117		
118		
119		
120		
121		
122		
123		
124		
125		
126		
127		
128		
129		
130		
131		
132		
133		
134		
135		
136		
137		
138		
139		
140		
141		
142		
143		
144		
145		
146		
147		
148		
149		
150		
151		

26

The screenshot shows a Windows desktop environment with several open windows. In the center is a phpMyAdmin interface for a MySQL database named 'Big_Julie'. The 'User' table is selected, displaying a list of users with their details: username, password, loggedIn, active, lastReqTime, voteWeight, and skipped. The table contains 20 rows of data. The 'Operations' menu bar is visible at the top of the phpMyAdmin window. The status bar at the bottom right of the screen shows the time as 11:58 and the date as 24/04/2015.

	username	password	loggedin	active	lastReqTime	voteWeight	skipped
08cunne	556698717		0	0	0 1.00	0	
08cfree	WOALD4031941532		0	0	0 1.00	0	
08cfurnell	704509DOSTANVIL		0	0	0 1.00	0	
08chanrahan	7810225ACCUSE		0	0	0 1.00	0	
08cliggins	2773		1	10	0 1.00	1	
08cnash	MYXOID6277034469		0	0	0 1.00	0	
08cperry	93995435225		0	0	0 1.00	0	
08cpreece	44235427205MAYO		0	0	0 1.00	0	
08crueane	WAREZ31426OBTEC		0	0	0 1.00	0	
08cstewardson	41BEMADS46		0	0	0 1.00	0	
08czako	SCENAS91DAULT		0	0	0 1.00	0	
08dboyle	18CHAFESFLINTS		0	0	0 1.00	0	
08dportman	hunter2		1	0	1424969897321 5.00	0	
08dtaggart	0380FLIT503		0	0	0 1.00	0	
08ebarron	PHYSIO2LUCKY		0	0	0 1.00	0	
08eburke	9117939894DOGES		0	0	0 1.00	0	
08ecelella	iamtiny		1	0	1424970841297 0.01	0	
08eholmes	42030003KOALA		0	0	0 1.00	0	
08emagill	041056SEAZED		0	0	0 1.00	0	
08emartin	123		1	0	0 1.00	0	
08emcdonald	KUFISUNTRUEEMBA		0	0	0 1.00	0	
08eperrins	105777175481475		0	0	0 1.00	0	
08epocock	ZINC81865876		0	0	0 1.00	0	

Beta Testing

Along with the alpha testing, I had a focus group of 20 pupils use my system during their free periods. This allowed me to verify that the system was stable when multiple users were sending requests concurrently and also gave me valuable feedback on usability and compatibility with different mobile devices.

Questionnaire

I supplied each member of the focus group with a questionnaire to collect data on topics that would indicate whether the system was polished and easy to use. I focussed on the feel of the system instead of the function since each user would have a slightly different experience and the mechanics were already the focus of my alpha testing.

1. I can navigate the website easily

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

2. I feel like my actions have an effect on the system

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

3. I can easily find the songs that I want

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

4. The music playing on the system is of a high quality

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

5. The website is visually appealing

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

6. The website scales well on my device

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

7. The website feels responsive and dynamic

- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

8. I rarely get frustrated when using the system

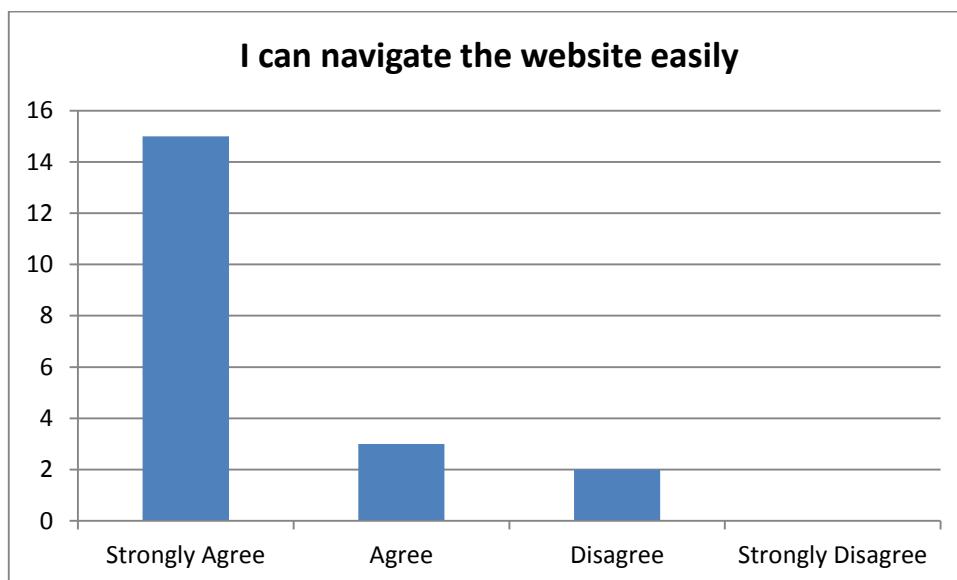
- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

9. As a whole I feel that using the system is a positive experience

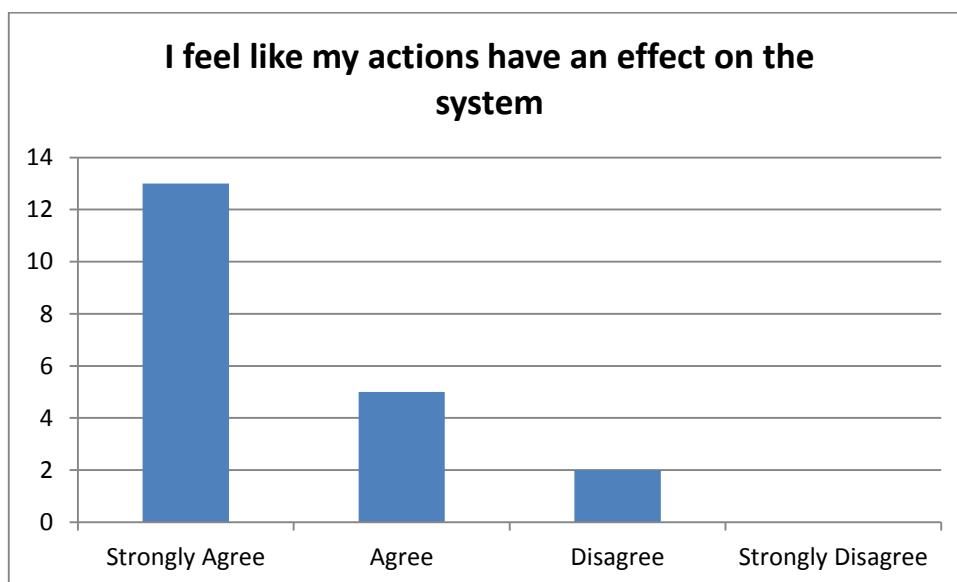
- a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

Questionnaire Results

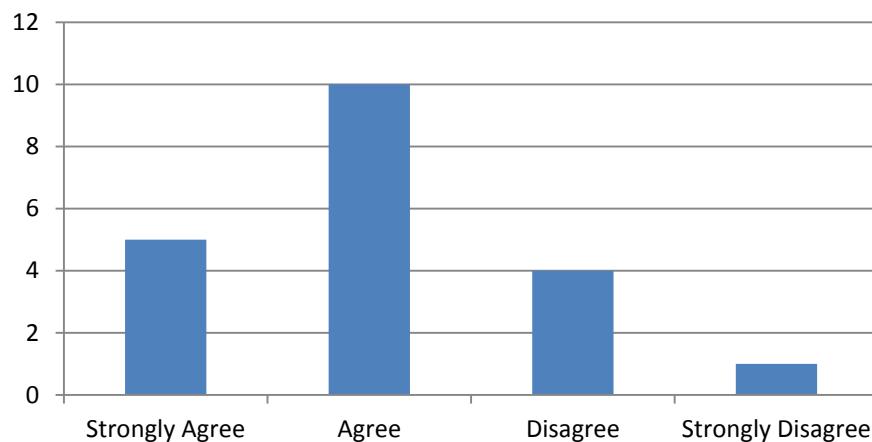
The overall feedback from my focus group was very positive. The results indicated that the usability and aesthetics of the website were very well received. In particular, the users agreed almost unanimously that the website was easy to navigate, visually appealing and responsive. While the majority of users responded positively to the quality of music and compatibility with mobile devices, some users were not satisfied. When discussing those topics with the group, it was noted that the filtering of song requests by explicit content and track length caused some users to be unable to request the music that they desired. This is an unfortunate side effect of the requirements outlined for the system although other results indicate that it does not impact the user experience to a significant level.



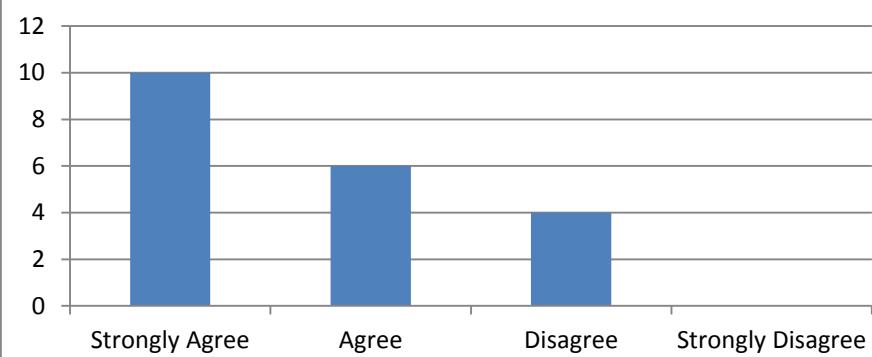
The majority of users feel that the website is easy to navigate



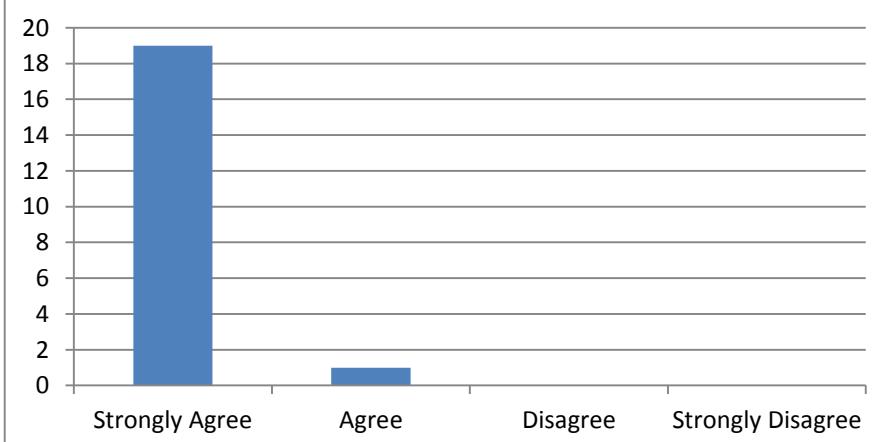
The majority of users feel that their actions have an effect on the system

I can easily find the songs that I want

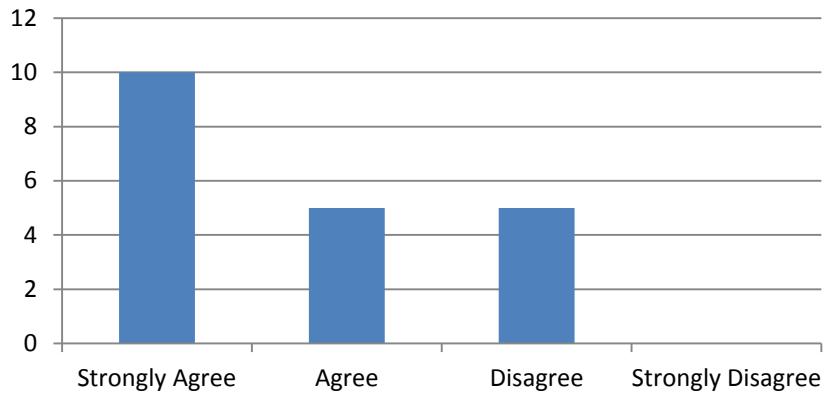
Most users can easily find the songs that they want

The music playing on the system is of a high quality

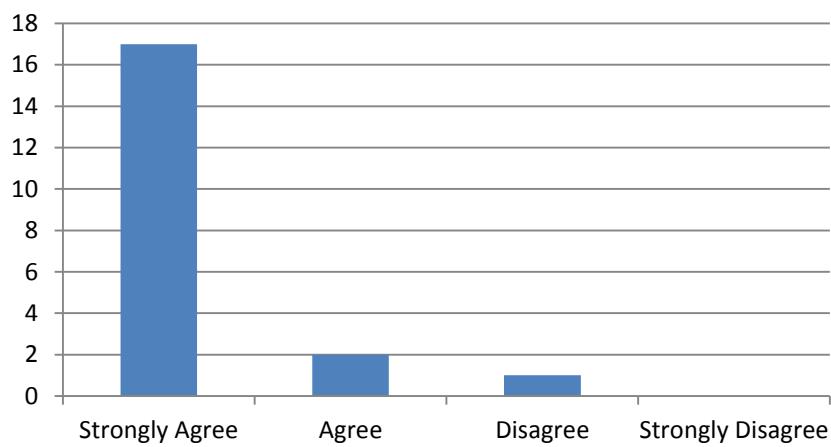
Most users feel that the music playing is of a high quality

The website is visually appealing

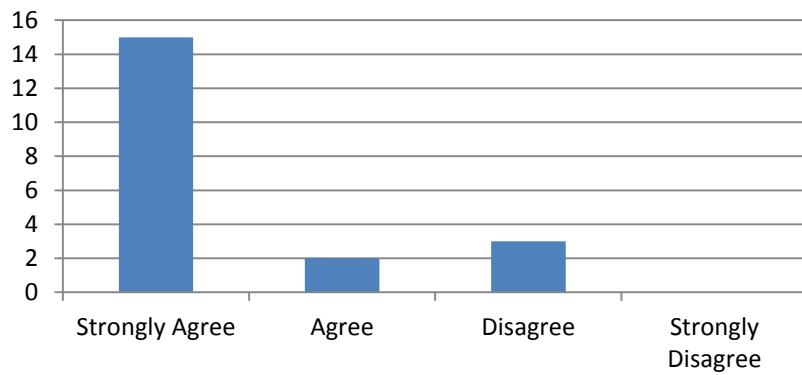
The majority of users feel that the website is visually appealing

The website scales well on my device

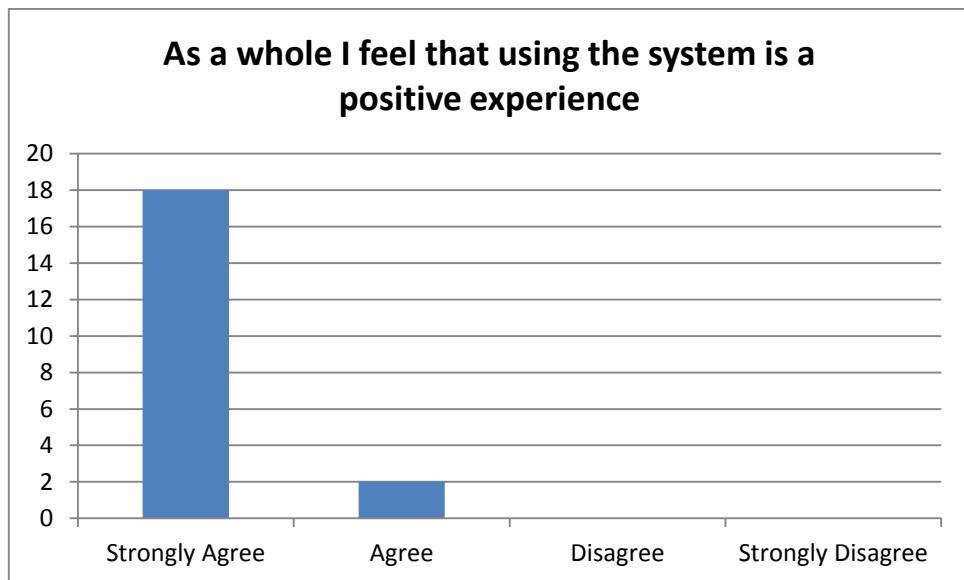
Most users feel that the website scales well on their devices

The website feels responsive and dynamic

The majority of users feel that the website is responsive and dynamic

I rarely get frustrated when using the system

The majority of users feel they are rarely frustrated by the system



The majority of users feel that using the system is a positive experience as a whole

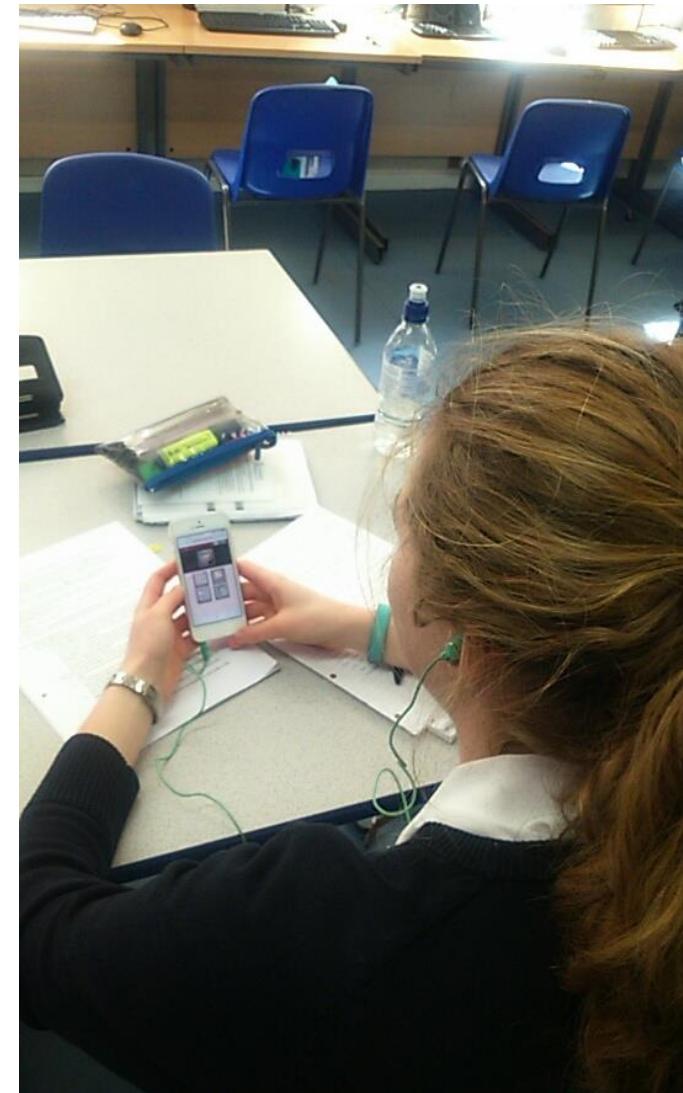
Feedback from Focus Group

During beta testing, I discussed the product with each member of the group in an informal conversation. I collated the suggestions into a table and improved the system based on feedback provided by the users.

<u>Feedback</u>	<u>Corrective Action</u>
Skip button does not show whether it has been pressed.	Added code to change skip button colour to grey to indicate that button has been pressed.
Previously entered text in search box remains after song request sent	Added code to clear track name and artist textboxes when yes button is pressed
Users are considered inactive too quickly	When user is flagged as active, value is set to 10 instead of 3. Meaning that the user will be considered active for the next 10 songs.
System “slows down” over time	Fixed memory leak in play function
Users could not search for songs by a specific artist	Split search box into track name and artist. Allowing for users to also search for a specific track and artist or any song by a specific artist



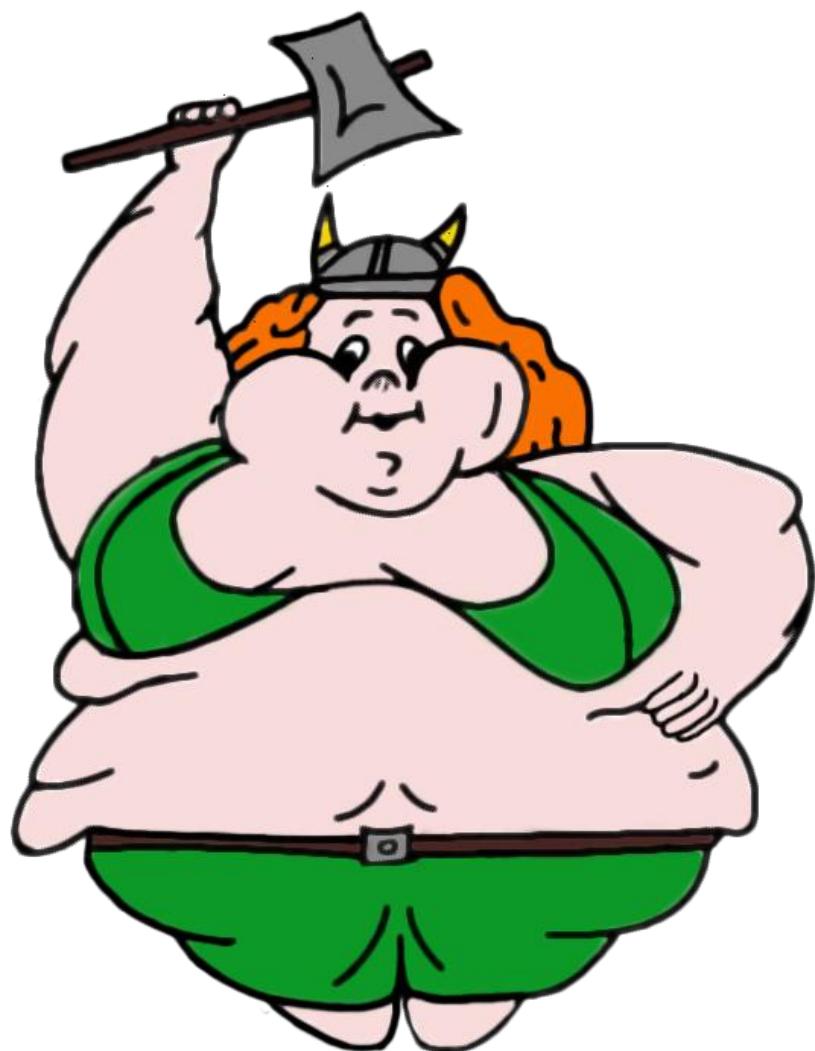
The Big Julie player during beta testing



A beta tester using the system.

BIG JULIE

Student User Guide



Contents

Introduction (2)

Setup and Installation

- Client Setup (4-6)

Typical Use

- Logging in (8-9)
- Song requests (10-15)
- Voting (16-18)
- Skipping (19-21)

Troubleshooting

- Enabling JavaScript (23-24)
- Login unsuccessful (25)
- No song found / incorrect results (26)
- Track already requested (27)
- Requested track not visible on page (28)
- No songs playing (29)

Glossary (31)

Introduction

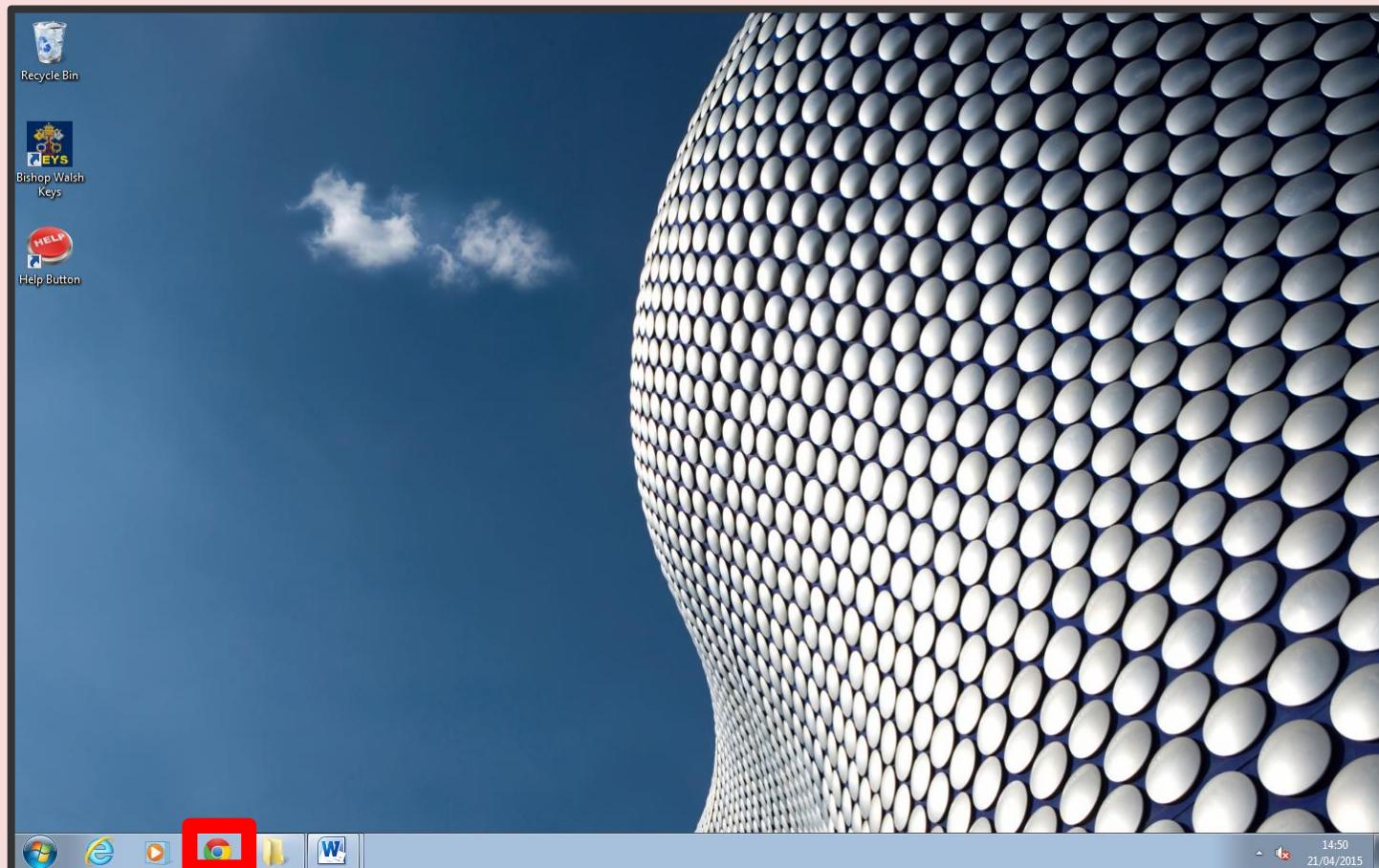
Big Julie is a multi-user music service available for all members of the Sixth Form at Bishop Walsh School. It streams music through Spotify and allows students to democratically control the tracks that are played. Students can request their favourite tracks, vote for tracks that other students have suggested and vote to skip songs that they do not like.

Setup and Installation



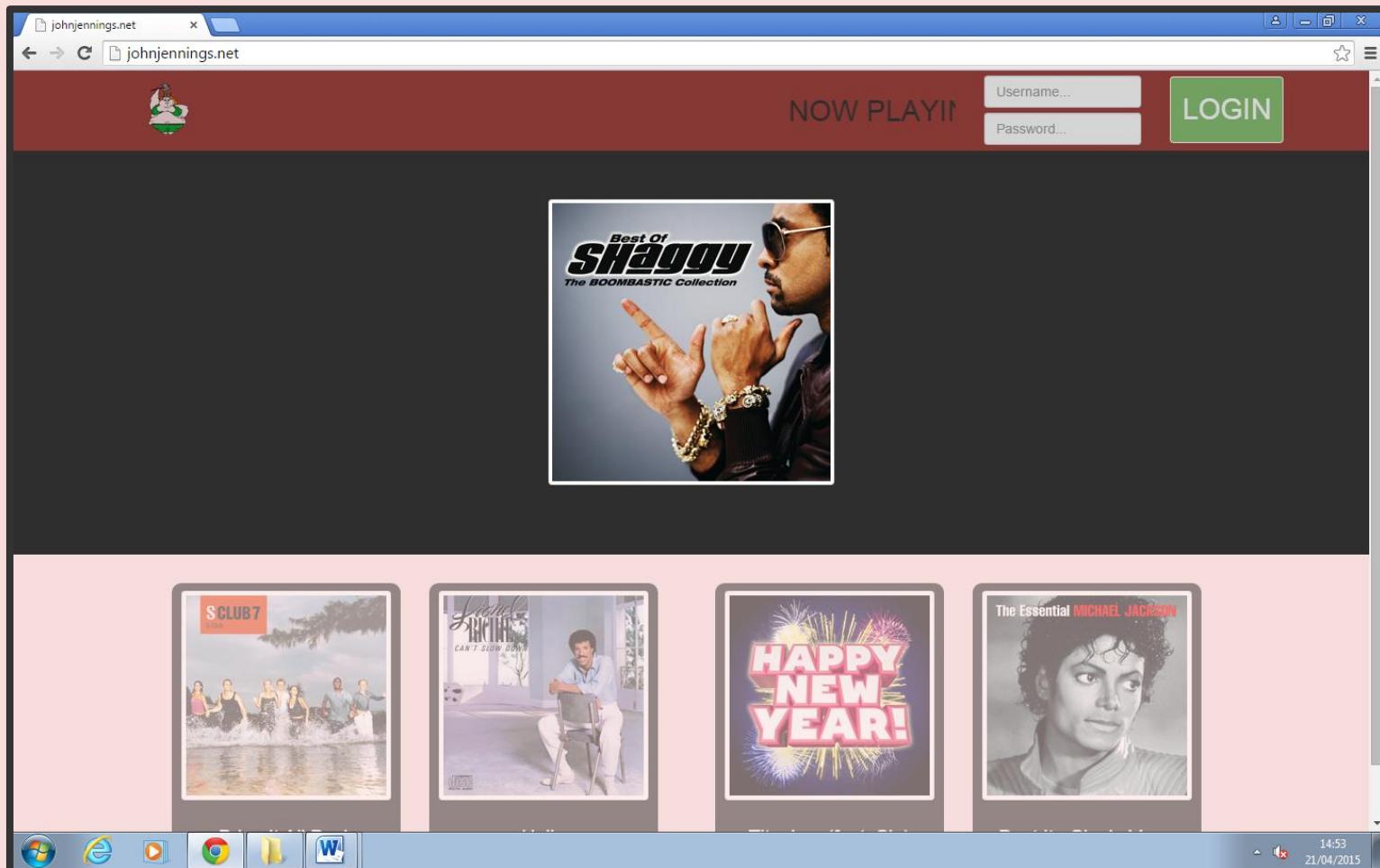
Client Setup

1. Sign in to a computer in the Sixth Form centre and open Google Chrome by clicking the icon at the bottom of the screen.



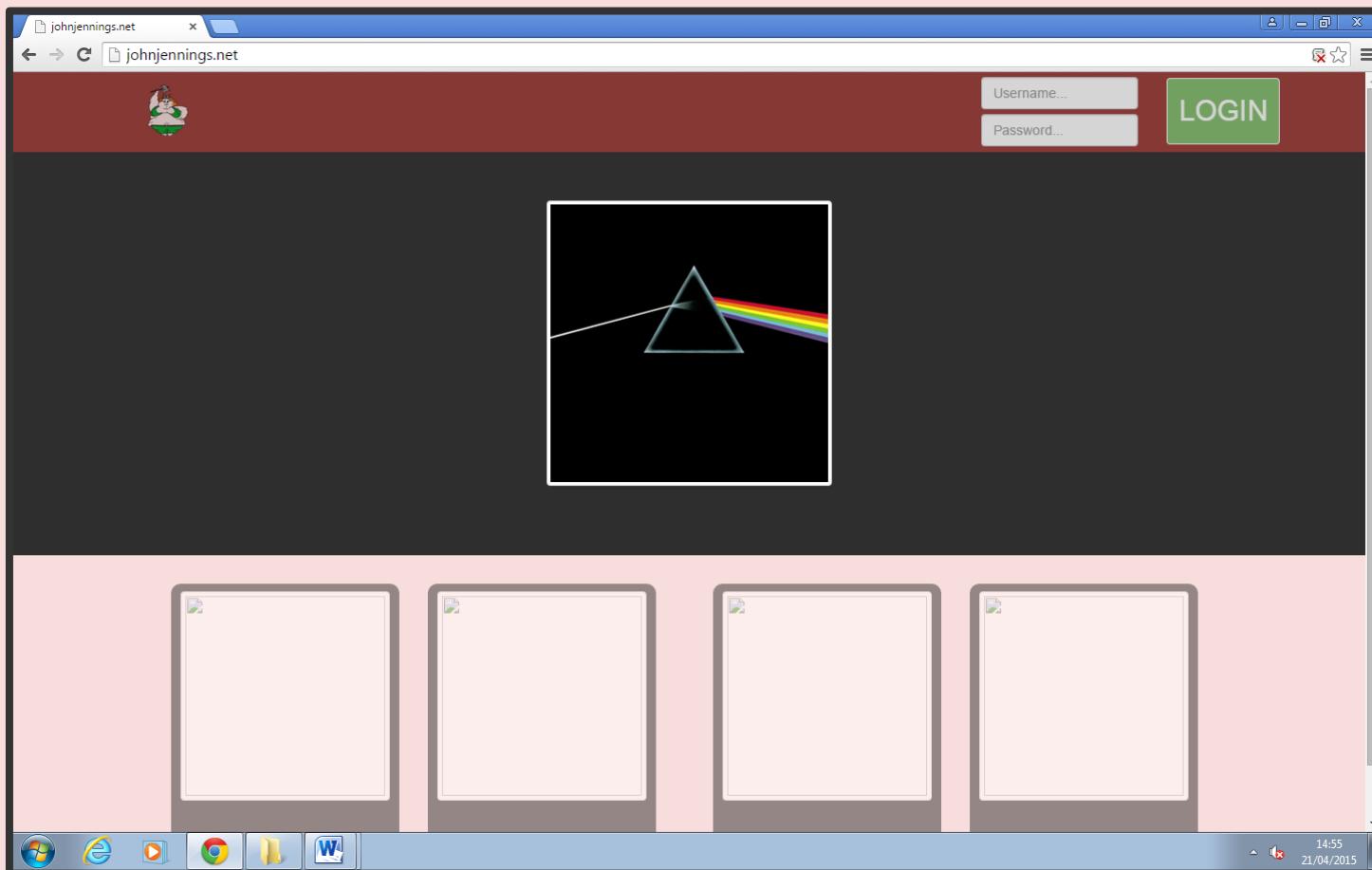
Client Setup

2. Visit www.johnjennings.net.



Client Setup

3. If the page does not display images in the modules at the bottom then enable **JavaScript** on the site by following the guide beginning at page 23.



Typical Use



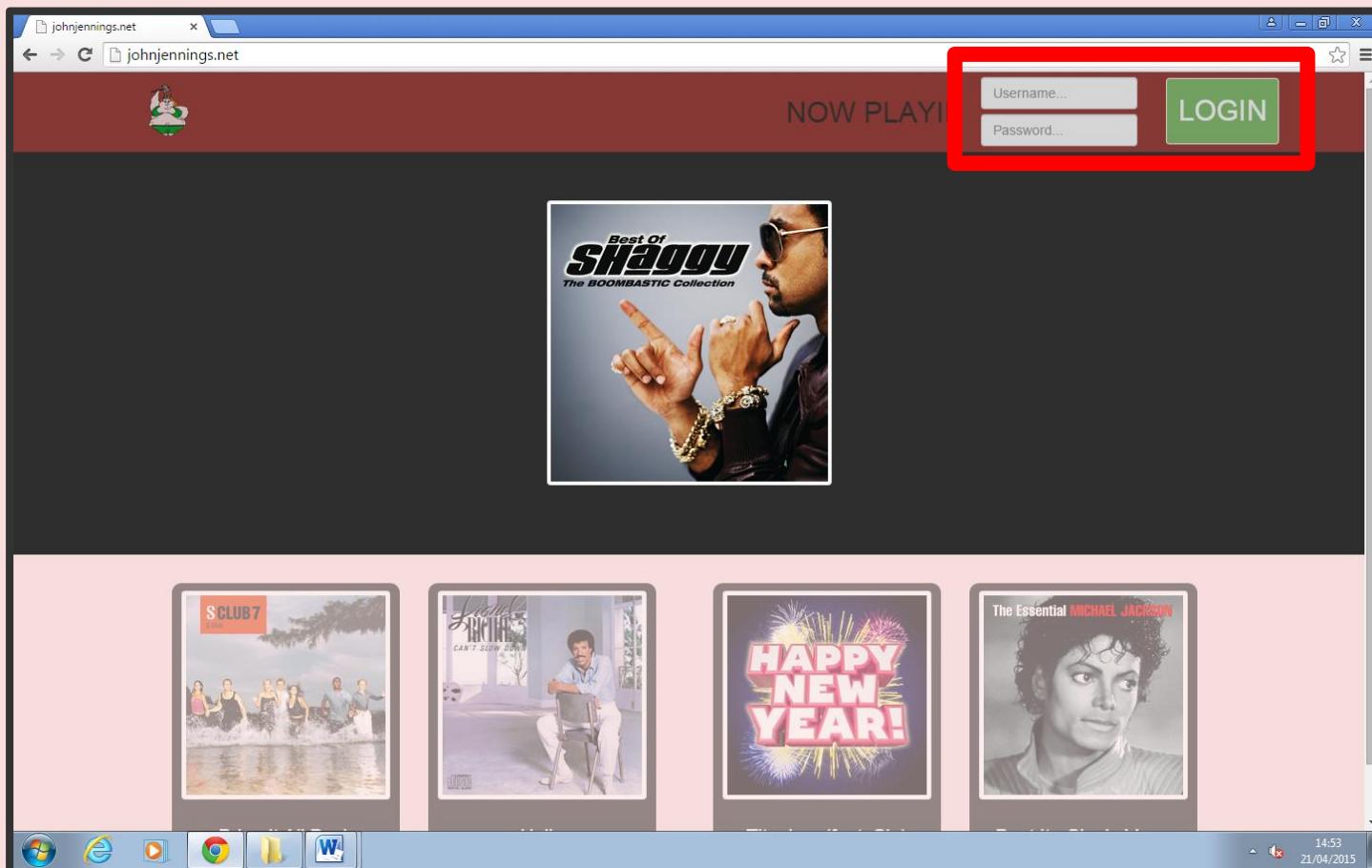
Logging in

1. Ask Mrs Hayes for your password and note it down in your homework diary.
Your username is your school username e.g. 08jjennings



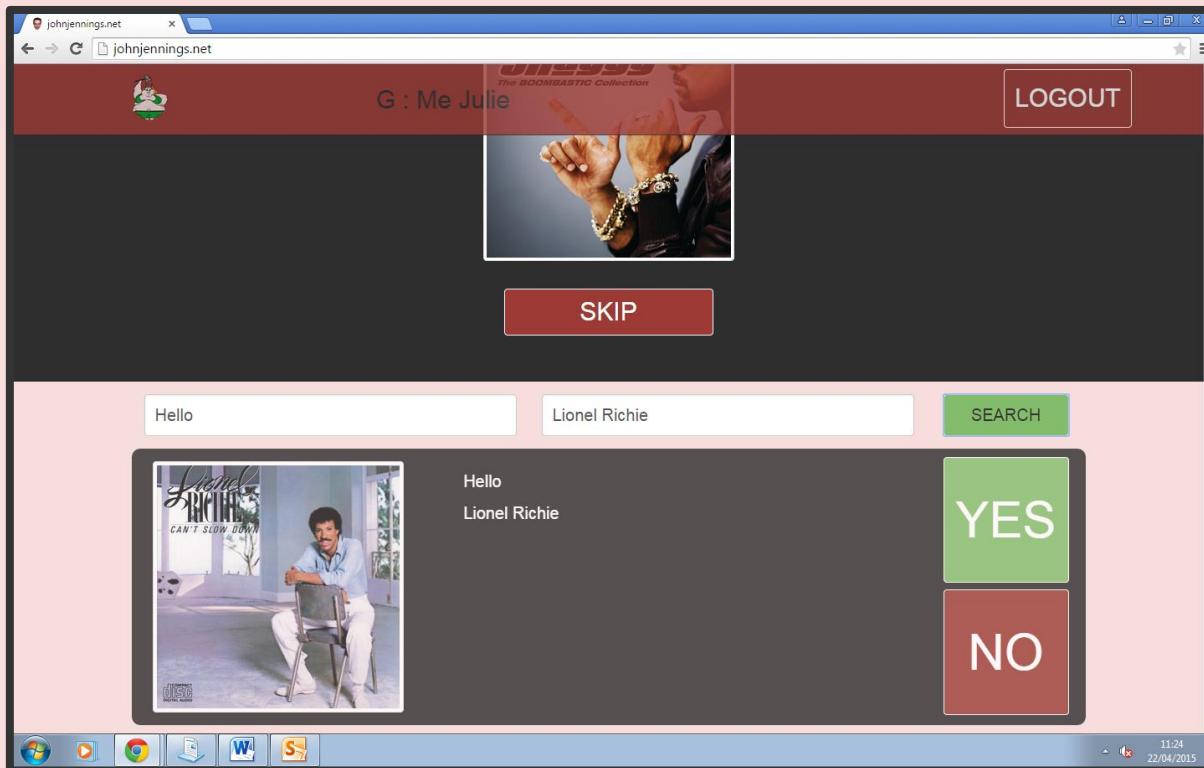
Logging in

2. Enter your username and password into the textboxes in the top right corner of the page and click the login button.



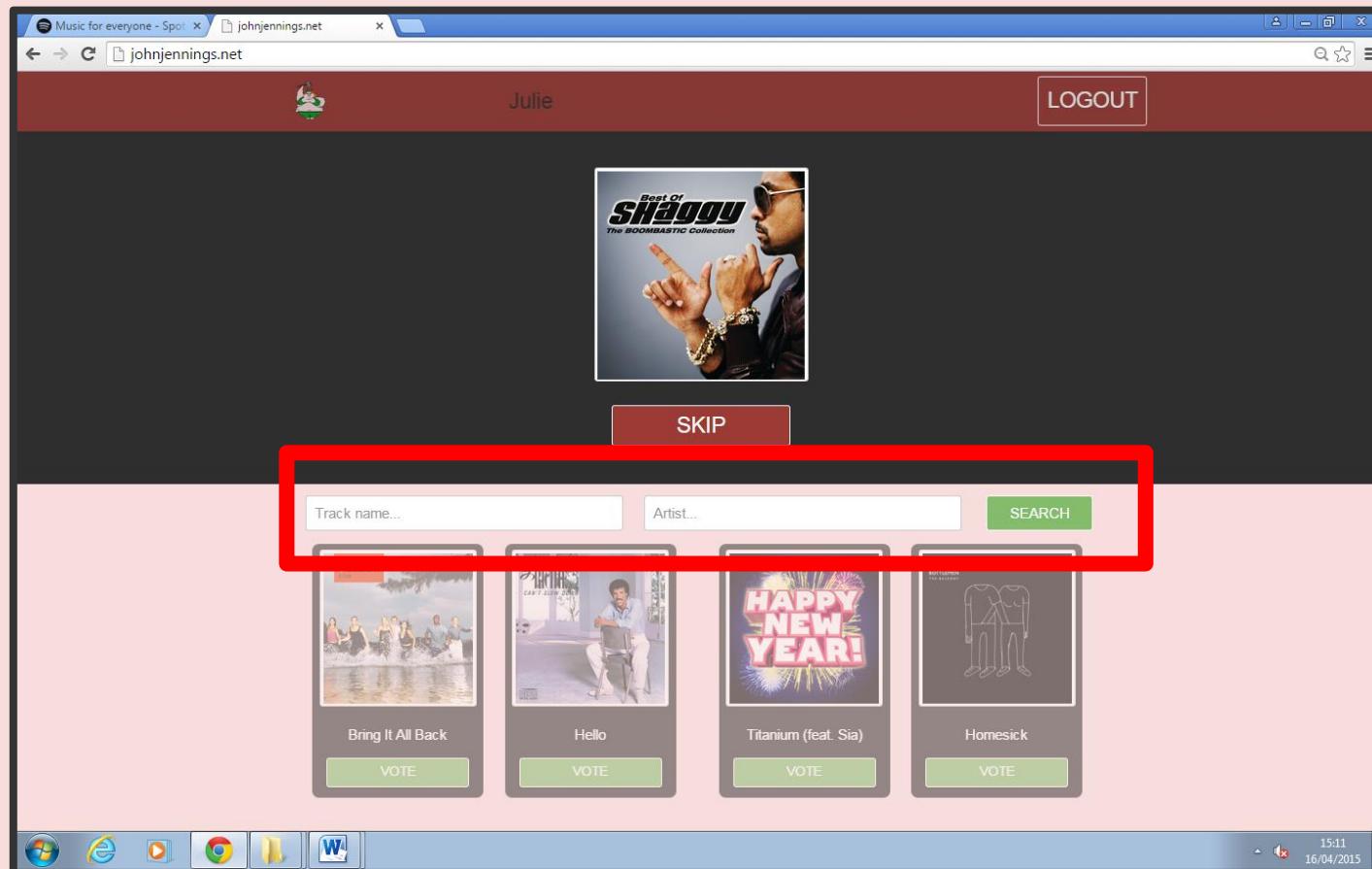
Song requests

A song request is a way for you to add a song of your choice to the play **queue**. You can request any song available on **Spotify** that is shorter than 7 minutes and not **explicit**. When you successfully request a song, the track will be added to the end of the queue and played once it reaches the front.



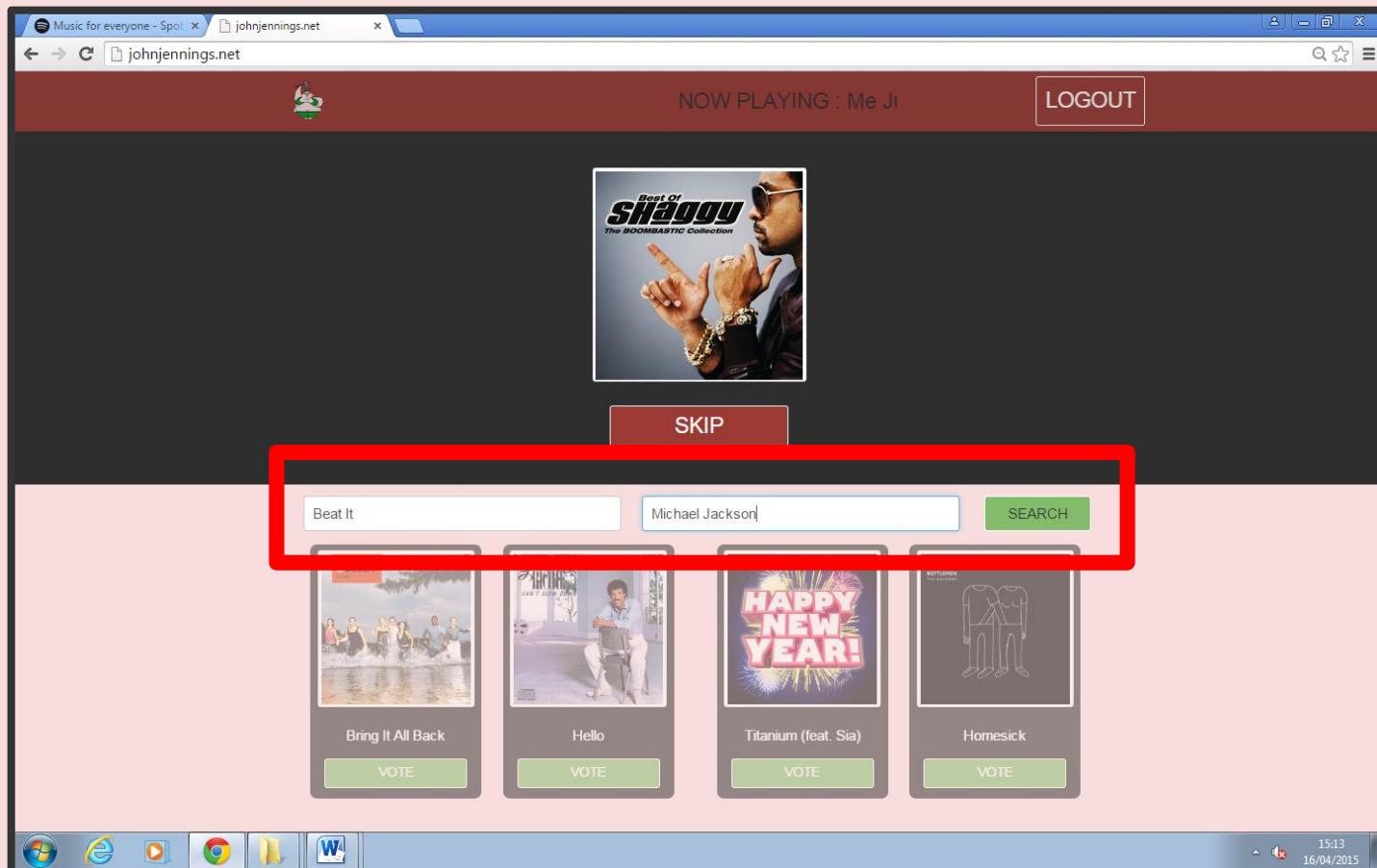
Song requests

1. To make a song request you must first be logged in. If song requests are available then the two search textboxes and the search button will be visible.



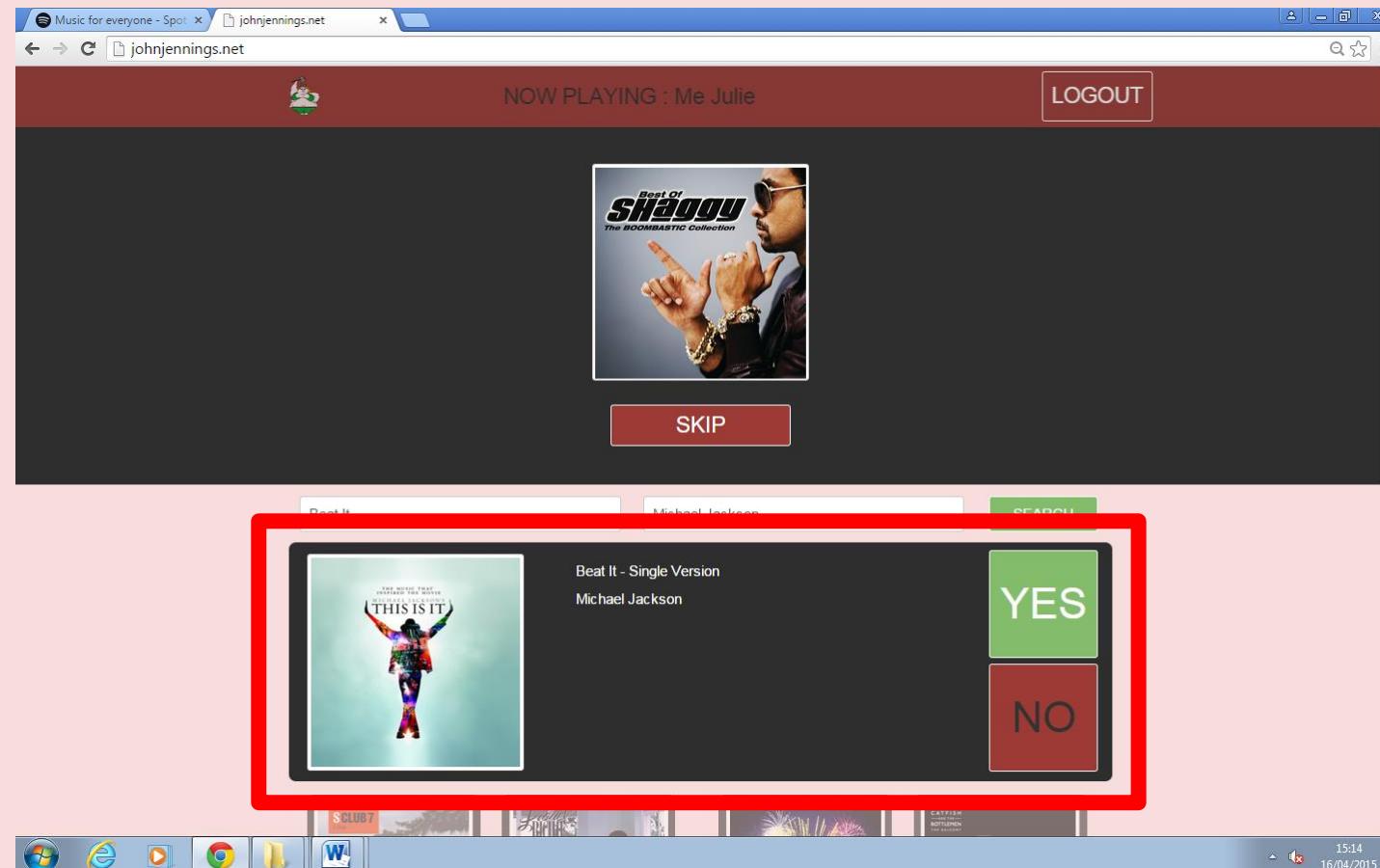
Song requests

2. Enter the track name and artist of the song you want to request into the textboxes and click the search button.



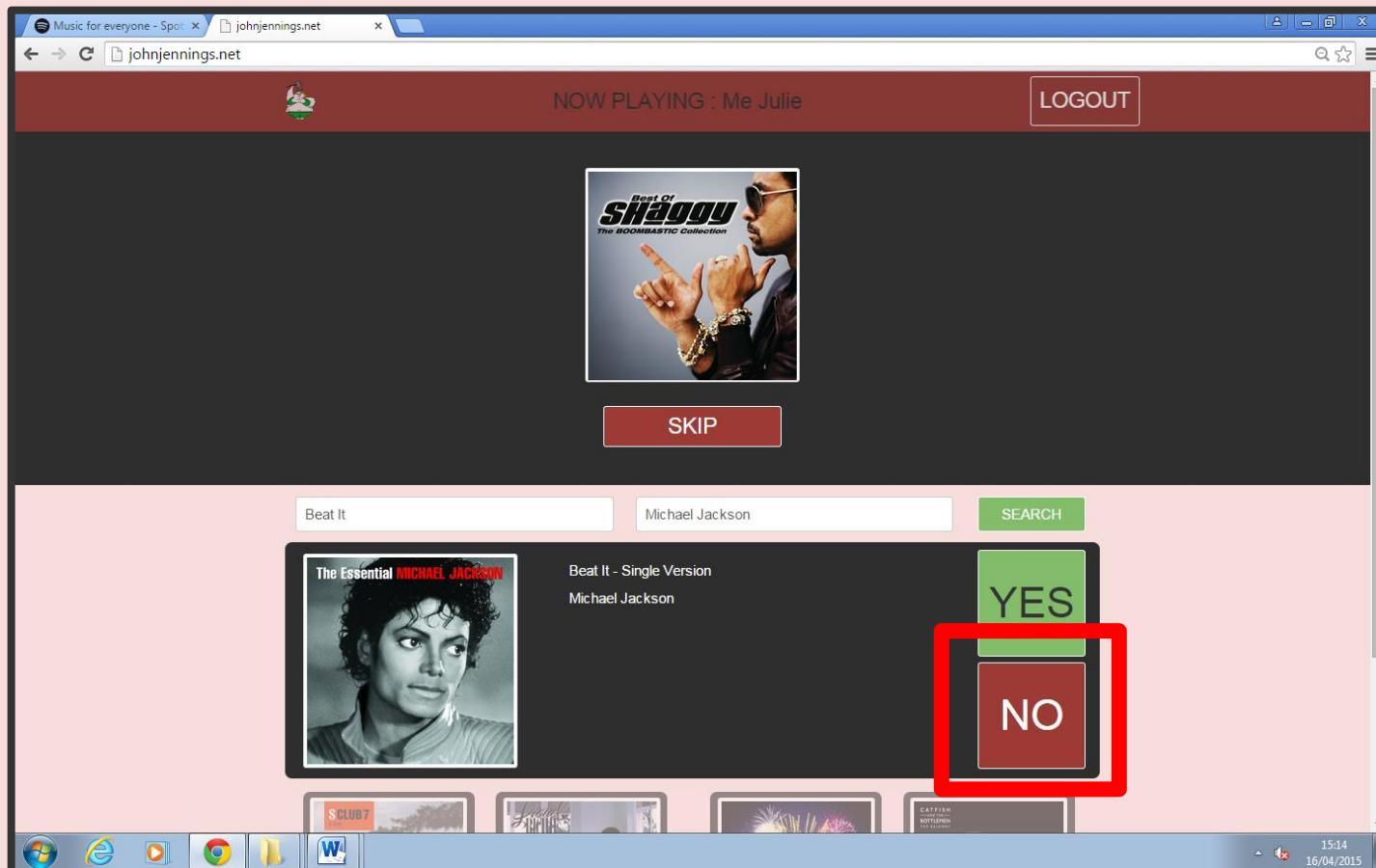
Song requests

3. The search result module will now open and display the first track that you could request. If this is the correct track then click the yes button to request that track.



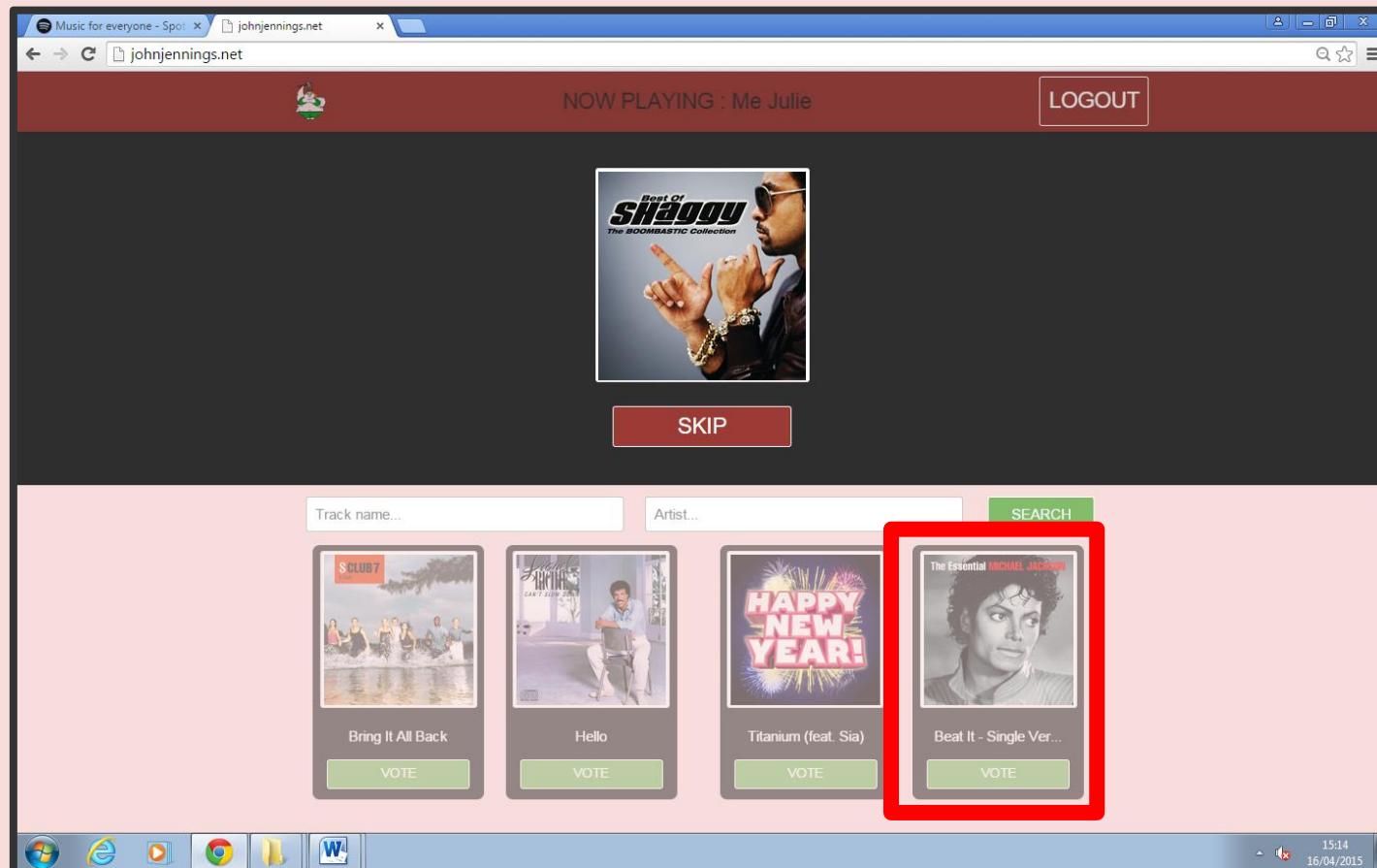
Song requests

4. If the track displayed is not correct then click the no button to cycle through other available songs.



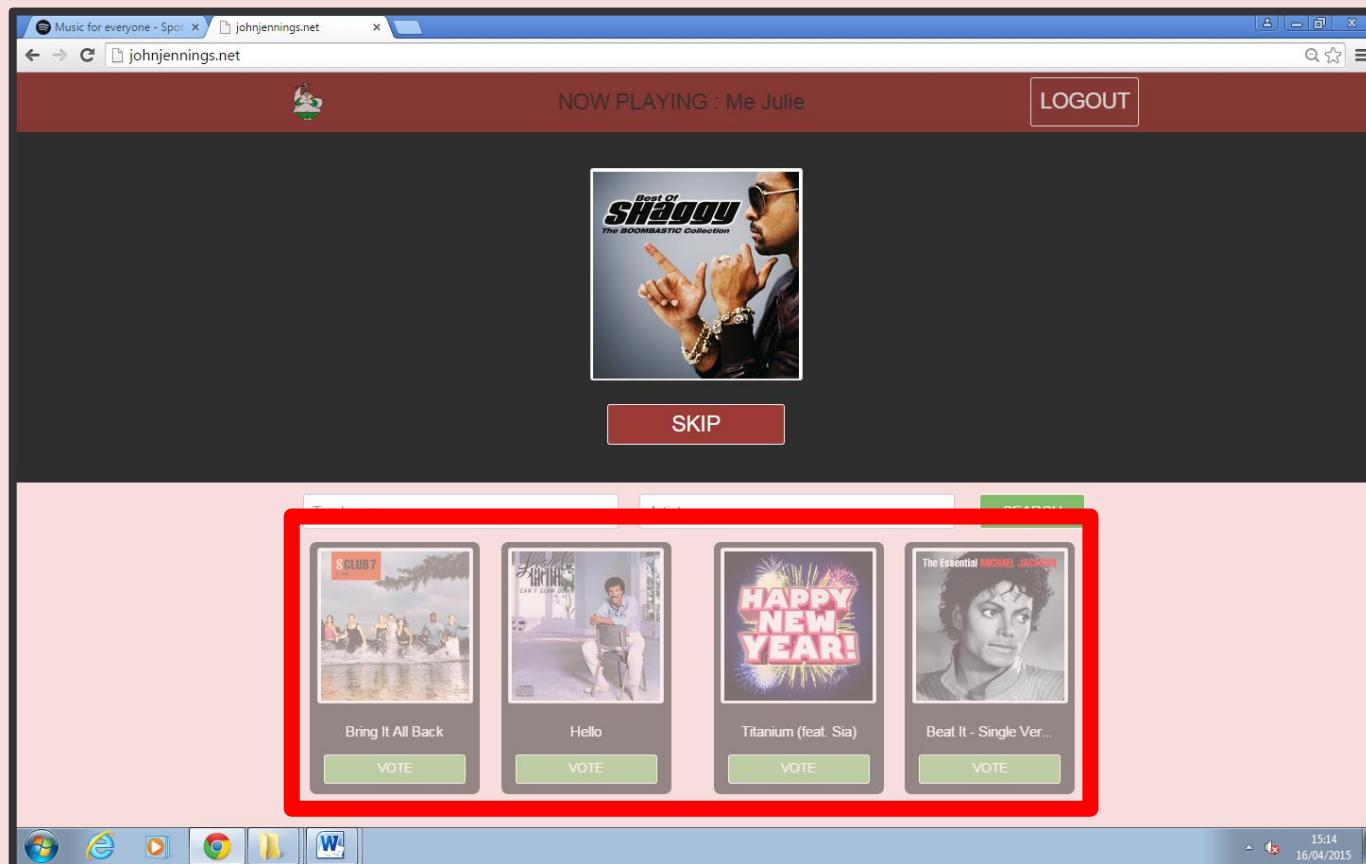
Song requests

5. When you click the yes button, the search results module will close and the song will be added to the queue.



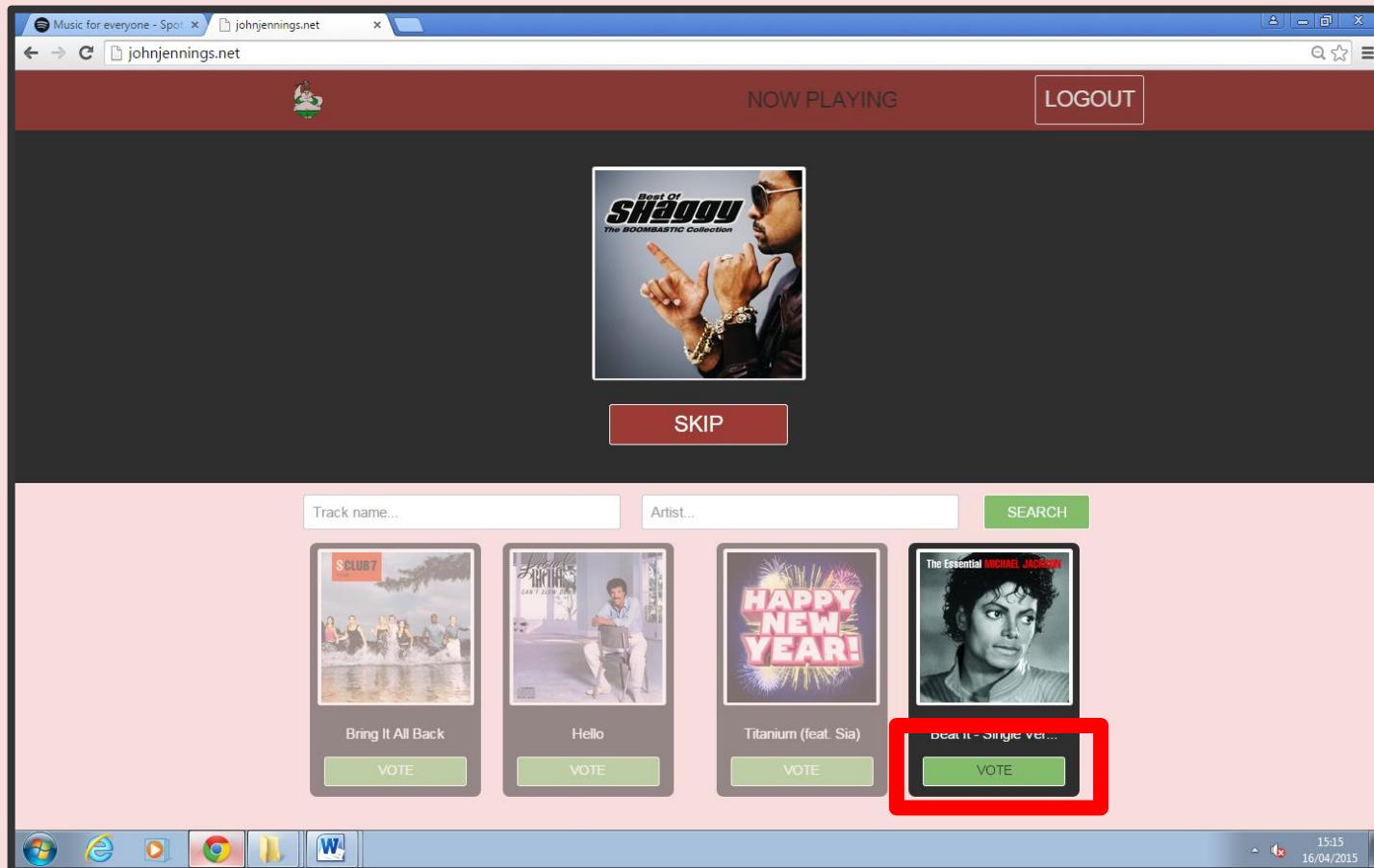
Voting

A vote request allows you to move a song forward in the **queue** so that it is played earlier. You can vote for any of the next four songs in the queue. The amount you can move a song forward is determined by your **vote weight**.



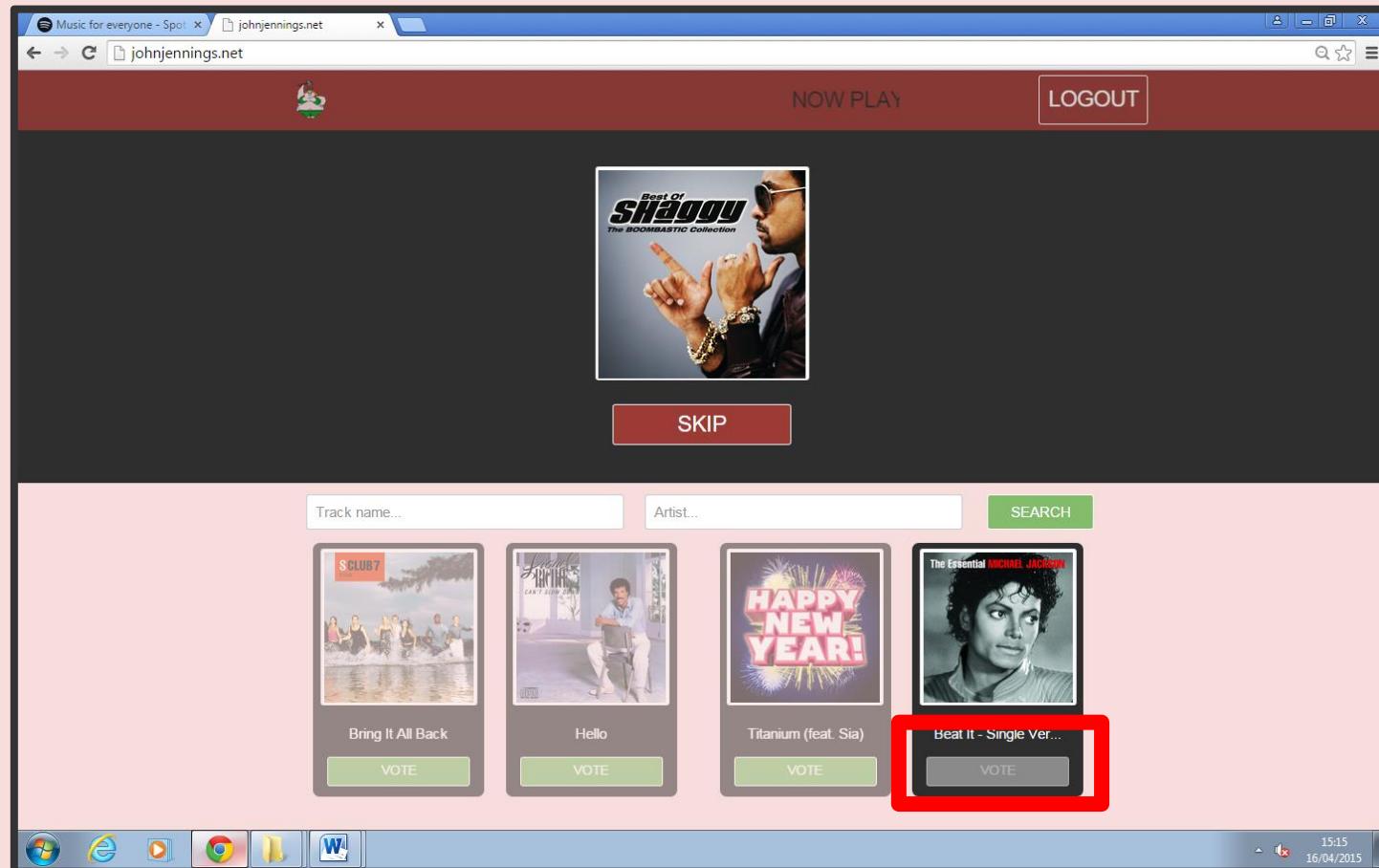
Voting

1. To vote for a specific song, click the vote button at the bottom of that song's module.



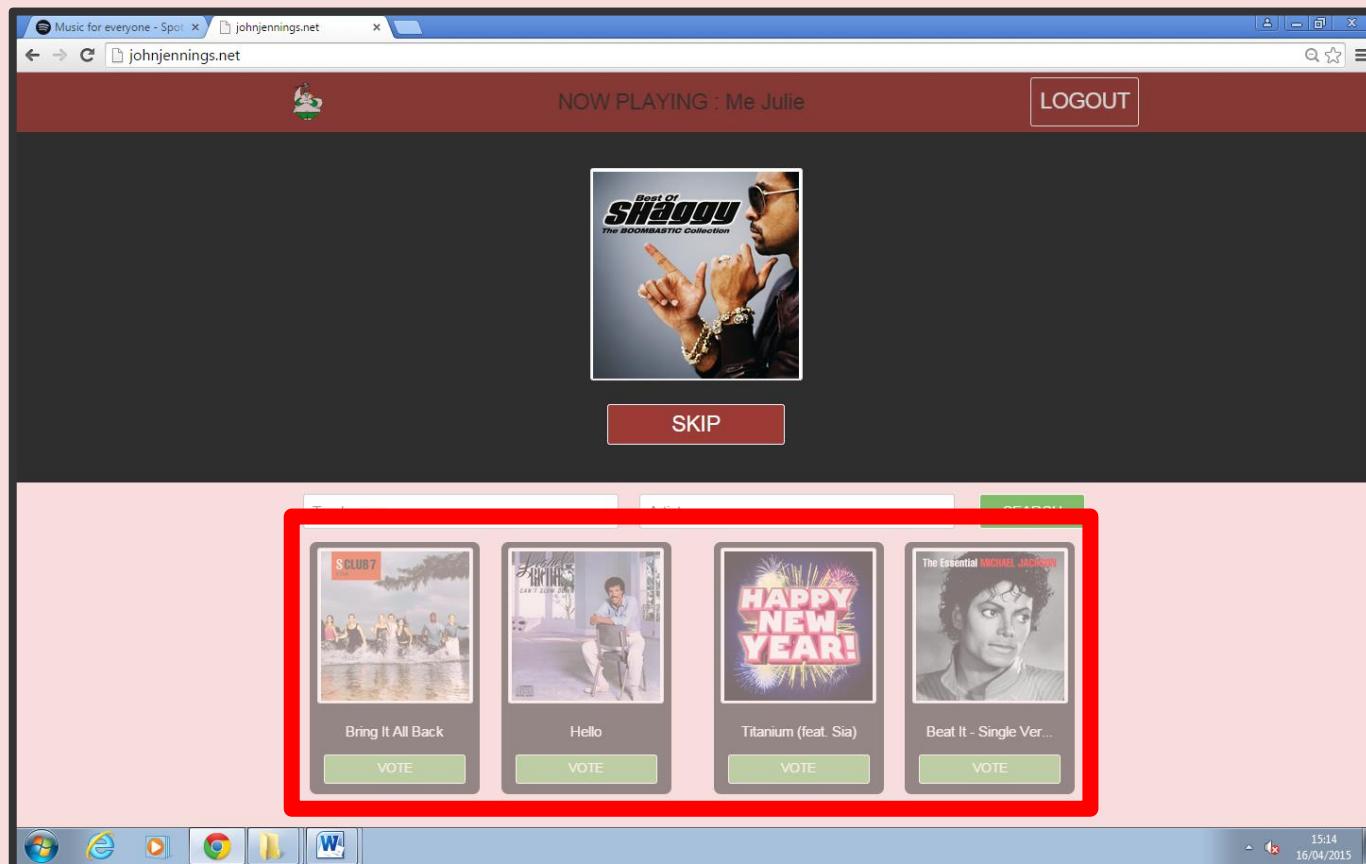
Voting

2. A vote request will be sent for the specified song and the vote button will now be disabled for that song until the currently playing song ends.



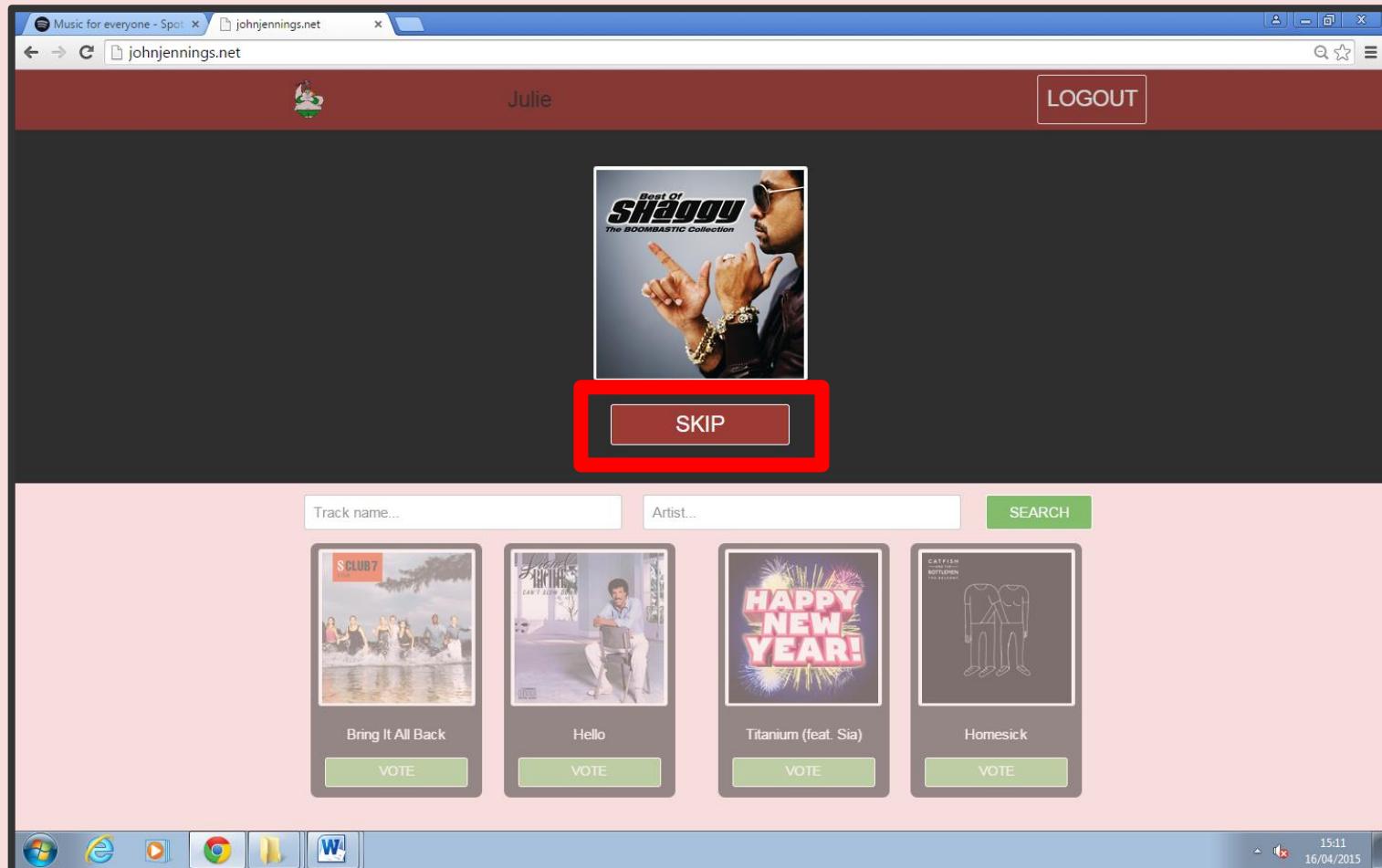
Skipping

A skip request allows you to skip songs that you do not like. If the majority of users currently using the system have sent a skip request then the currently playing song will end and the next song in the **queue** will begin playing.



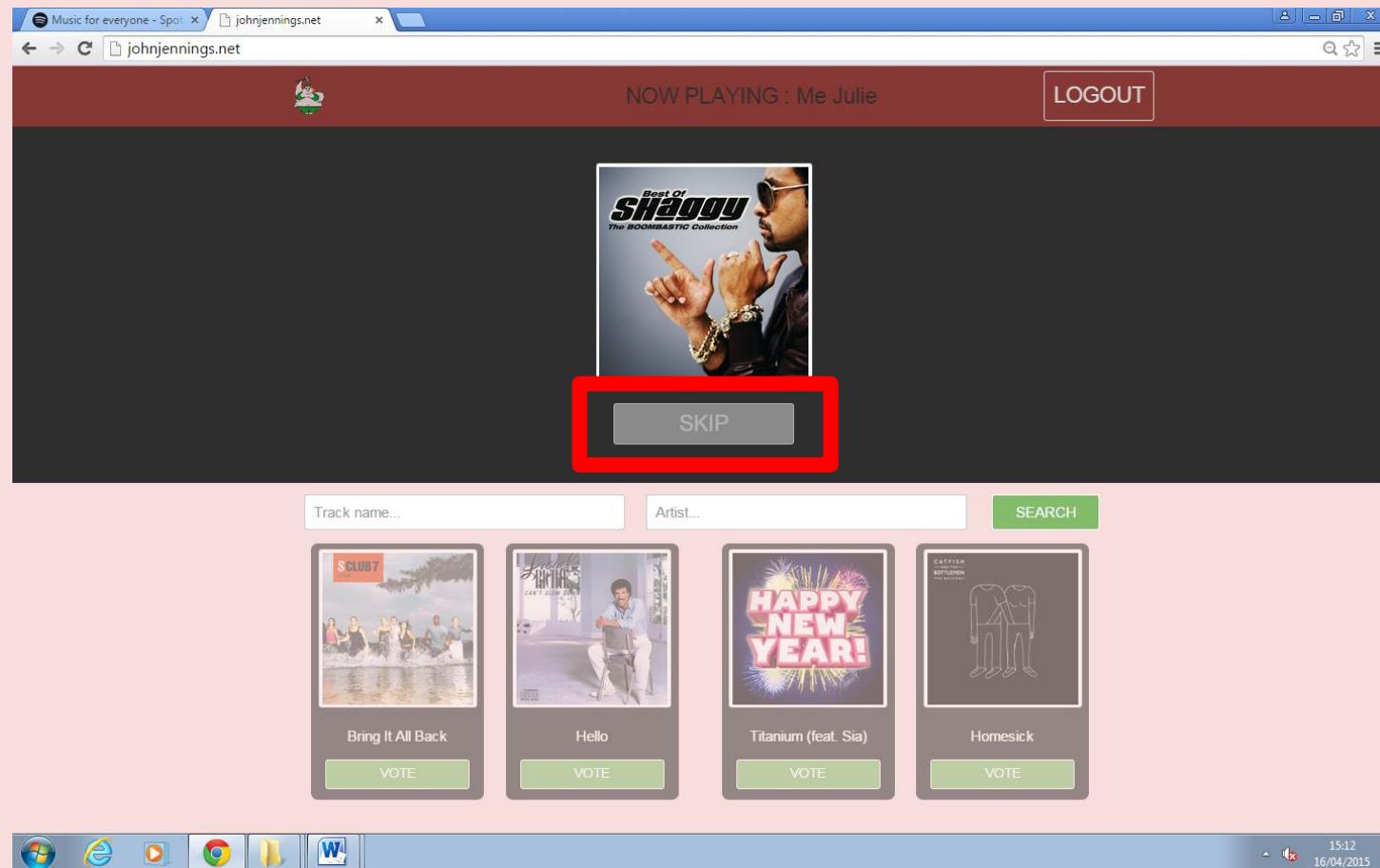
Skipping

1. To send a skip request, click the skip button in the middle of the page.



Skipping

2. A skip request will be sent and the skip button will now be disabled until the currently playing song ends.



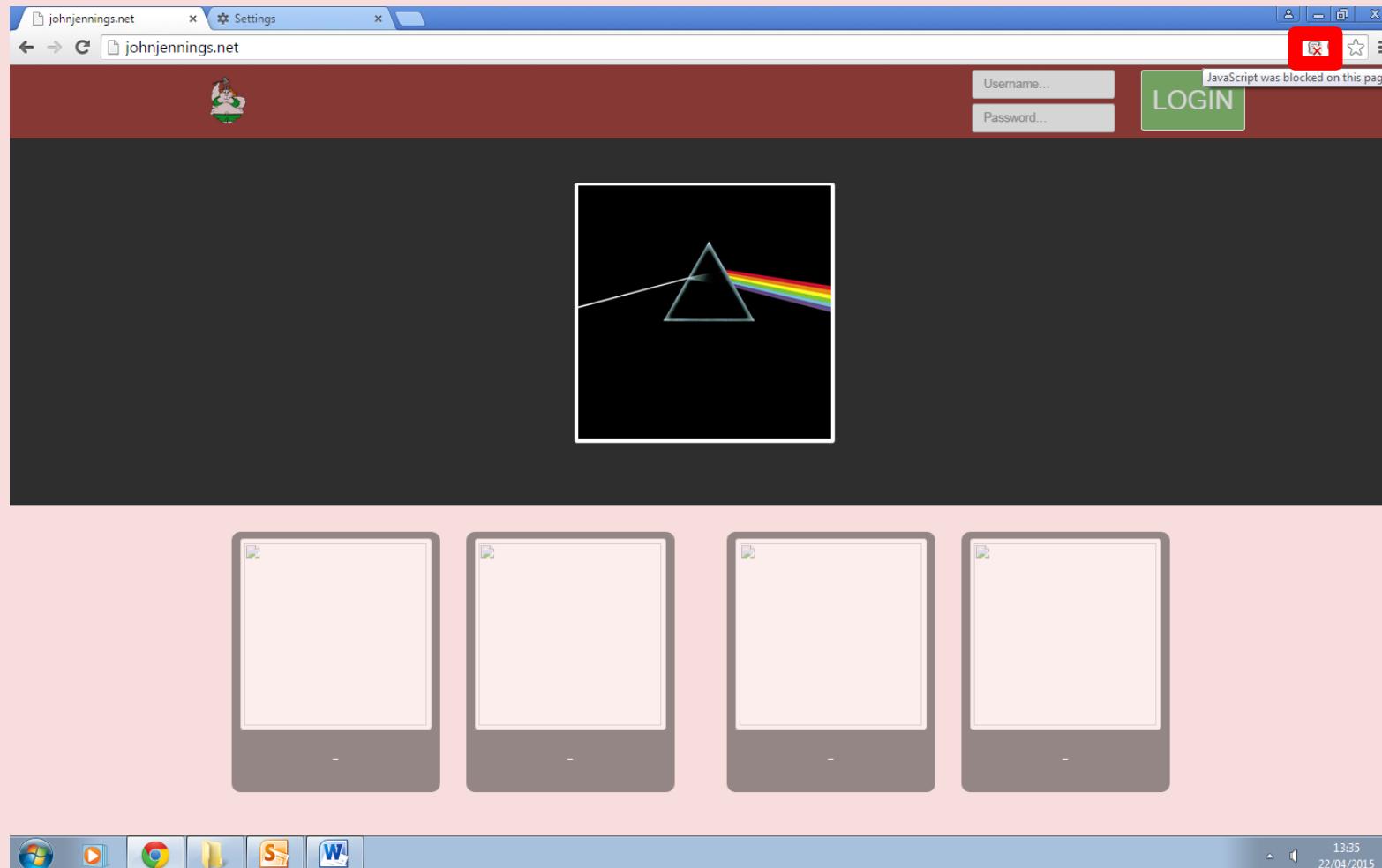
Troubleshooting

This section aims to solve some of the common problems that you may experience when using Big Julie:

- Enabling JavaScript (23-24)
- Login unsuccessful (25)
- No song found / incorrect results (26)
- Track already requested (27)
- Requested track not visible on page (28)
- No songs playing (29)

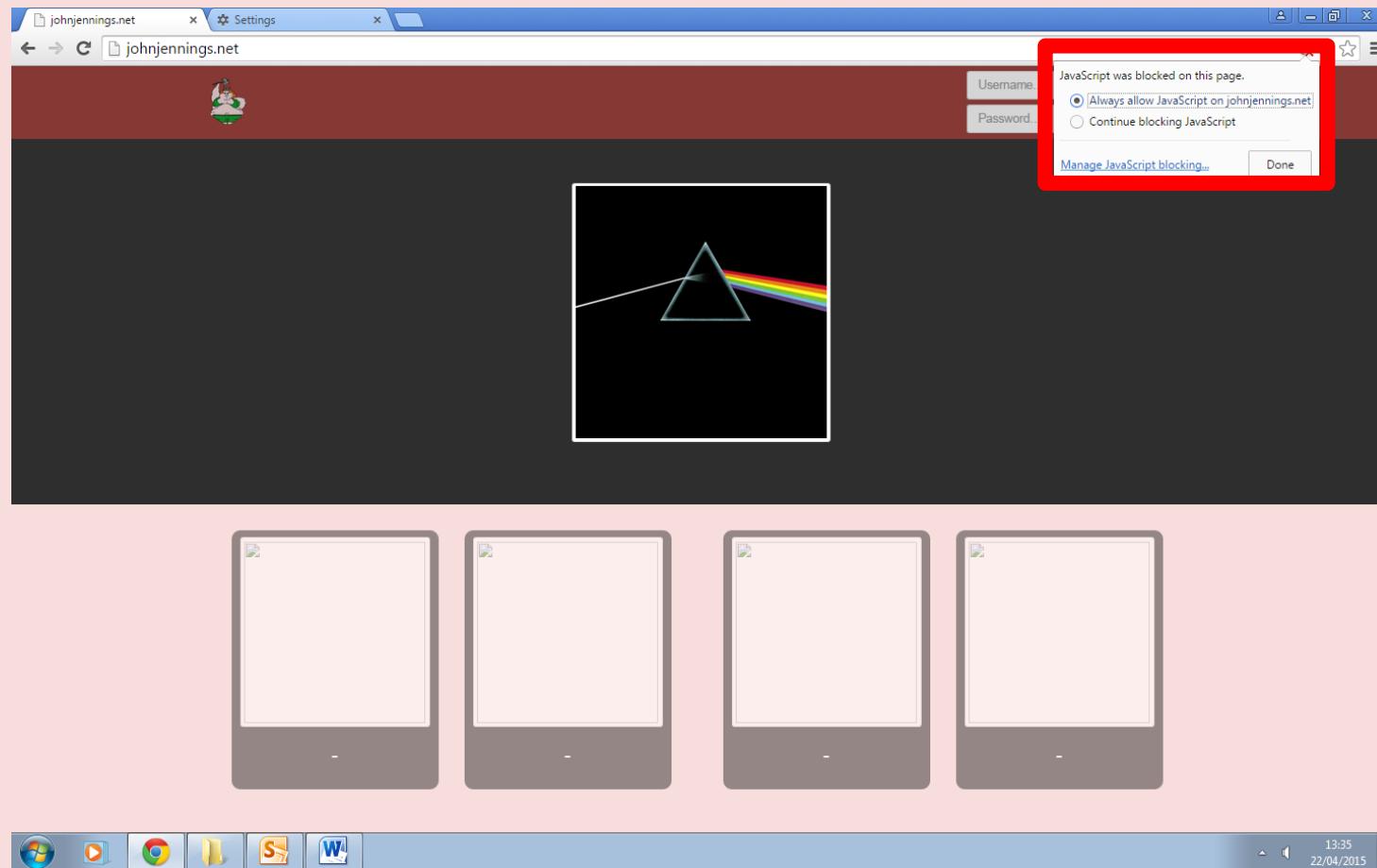
Enabling JavaScript

1. Click the **JavaScript** icon to the left of the magnifying glass in the **address bar**.



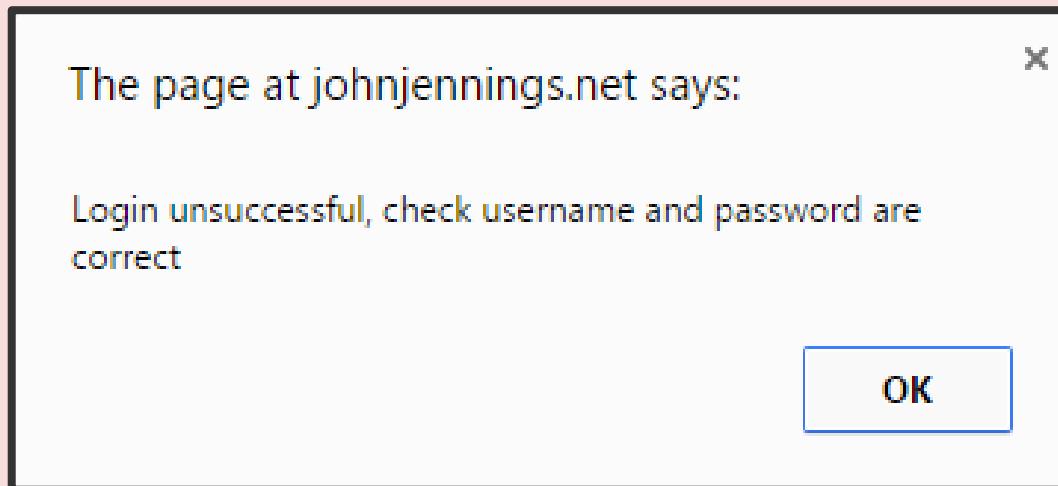
Enabling JavaScript

2. Click the radio button at the top of the alert then click the done button in the bottom right corner of the **alert**. Press the F5 key to refresh the page.



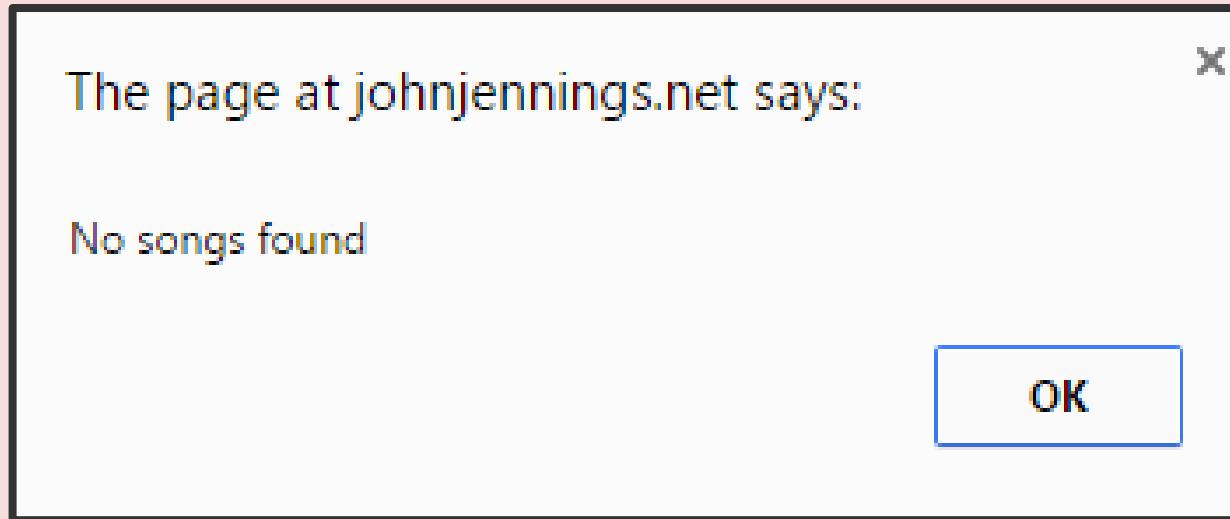
Login unsuccessful

When logging in, this **alert** may be displayed. This indicates that you have not entered a correct username and password and as a result have not been logged in to Big Julie. Check that you are entering your school username e.g. “08jjennings” and the password given to you by Mrs Hayes and try logging in again.



No song found / incorrect results

When searching for a song to request, this **alert** may be displayed or the results displayed may not be the song you wanted. Check that the track name and artist are spelled correctly and that the song can be found on the **Spotify** website. If the song is available then it may not meet the rules of Big Julie. The system automatically filters out songs that are **explicit** or over seven minutes long from the search results. If the track you wish to request does not meet these criteria then it cannot be played by Big Julie.



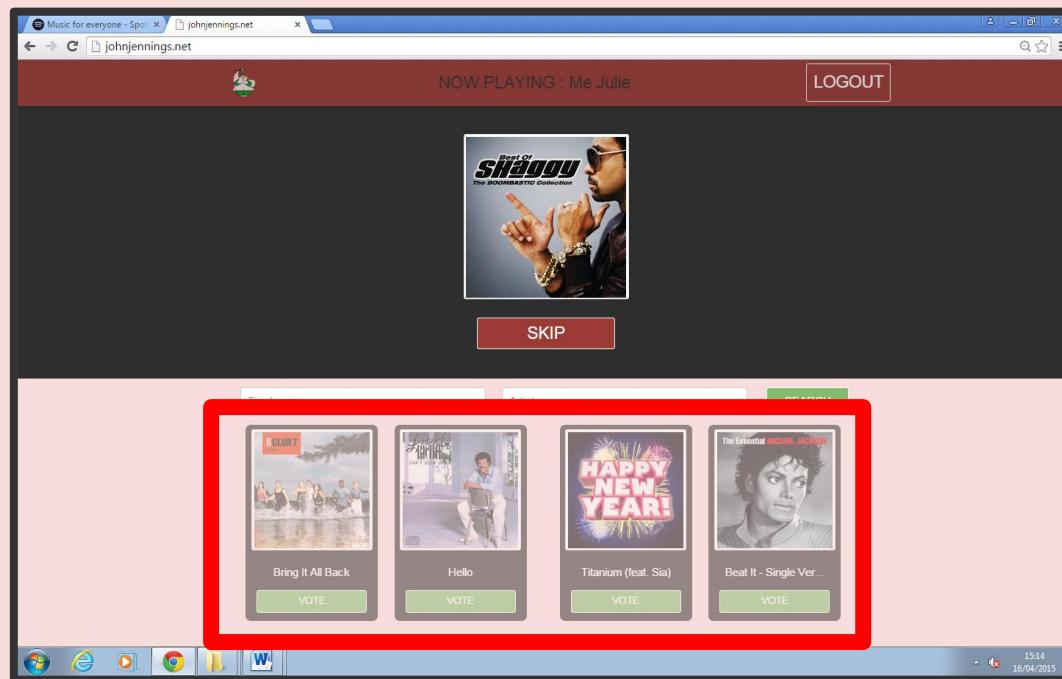
Track already requested

When requesting a song, this **alert** may be displayed. This indicates that the track is already present in the **queue** and cannot be requested again. Instead, you can vote for the track to move it forward in the queue or request a different version of that song.



Requested track not visible on page

When you have successfully requested a track, it will usually appear in one of the next song modules at the bottom of the page. When the system has received many requested in a short time frame, this may not be the case. Unless any error **alerts** have been displayed, the track will have still been added to the **queue** and will appear once the system plays through the songs requested before your track.



No songs playing

Big Julie should always be playing music while it is running. If no songs are being played then the system may have been disabled by a staff member or an error has occurred. Inform a member of staff that the system is not working properly and if it has not been disabled on purpose, they will try to solve the problem for you. The request system is separate from the player so you can still successfully make song, vote and skip requests even if no music is playing.

Glossary



Glossary

Address bar – a text box in a web browser displaying the address of the web page that is currently being viewed.

Alert – A small box that appears on the display screen to give you information or to warn you about a potentially damaging operation.

Explicit – Swearing and adult themes

JavaScript – a programming language commonly used to create interactive effects within web browsers. Must be installed for Big Julie to operate correctly

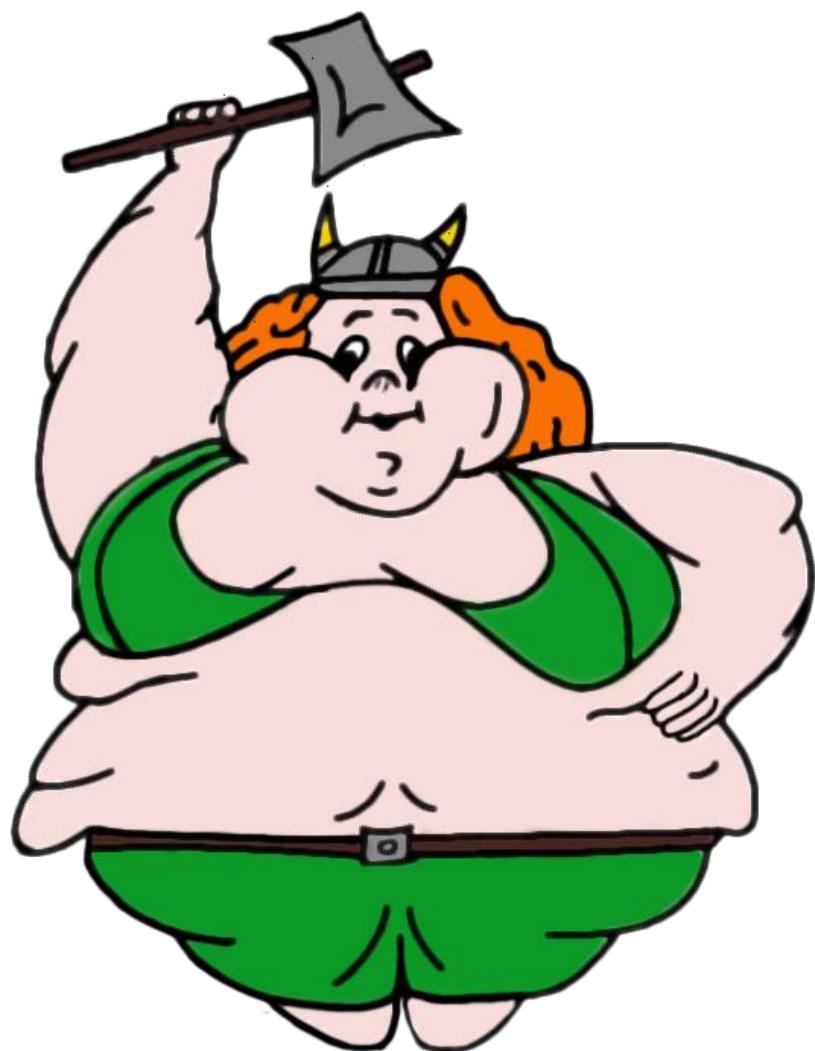
Queue – a part of Big Julie that ensures that songs are played in the order that they are requested.

Spotify – a music streaming service used by Big Julie to play tracks

Vote Weight – a value assigned to each user that determines how much effect their vote request has on the queue

BIG JULIE

Teacher User Guide



Contents

Introduction (2)

Setup and Installation

- Installing Spotify (4-12)
- Player Setup (13-15)
- Backup Routine (16-22)

Typical Use

- Adding Users (24-29)
- Removing Users (30-34)
- Turning Player On / Off (35)

Troubleshooting

- Enabling JavaScript (37-39)
- No songs playing (40-41)

Glossary (43)

System Requirements (44)

Introduction

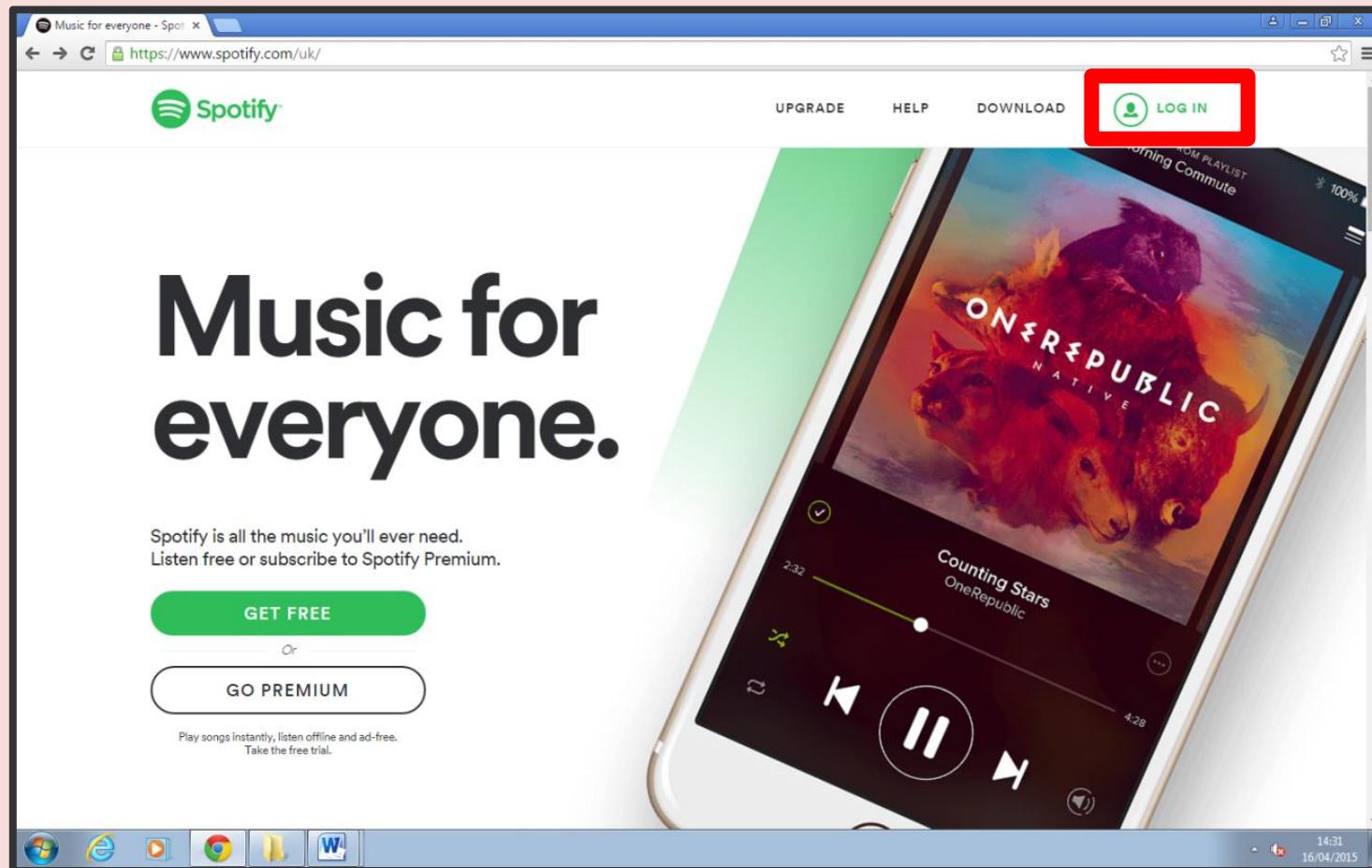
Big Julie is a multi-user music service available for all members of the Sixth Form at Bishop Walsh School. It streams music through **Spotify** and allows students to democratically control the tracks that are played. Students can request their favourite tracks, vote for tracks that other students have suggested and vote to skip songs that they do not like.

Setup and Installation



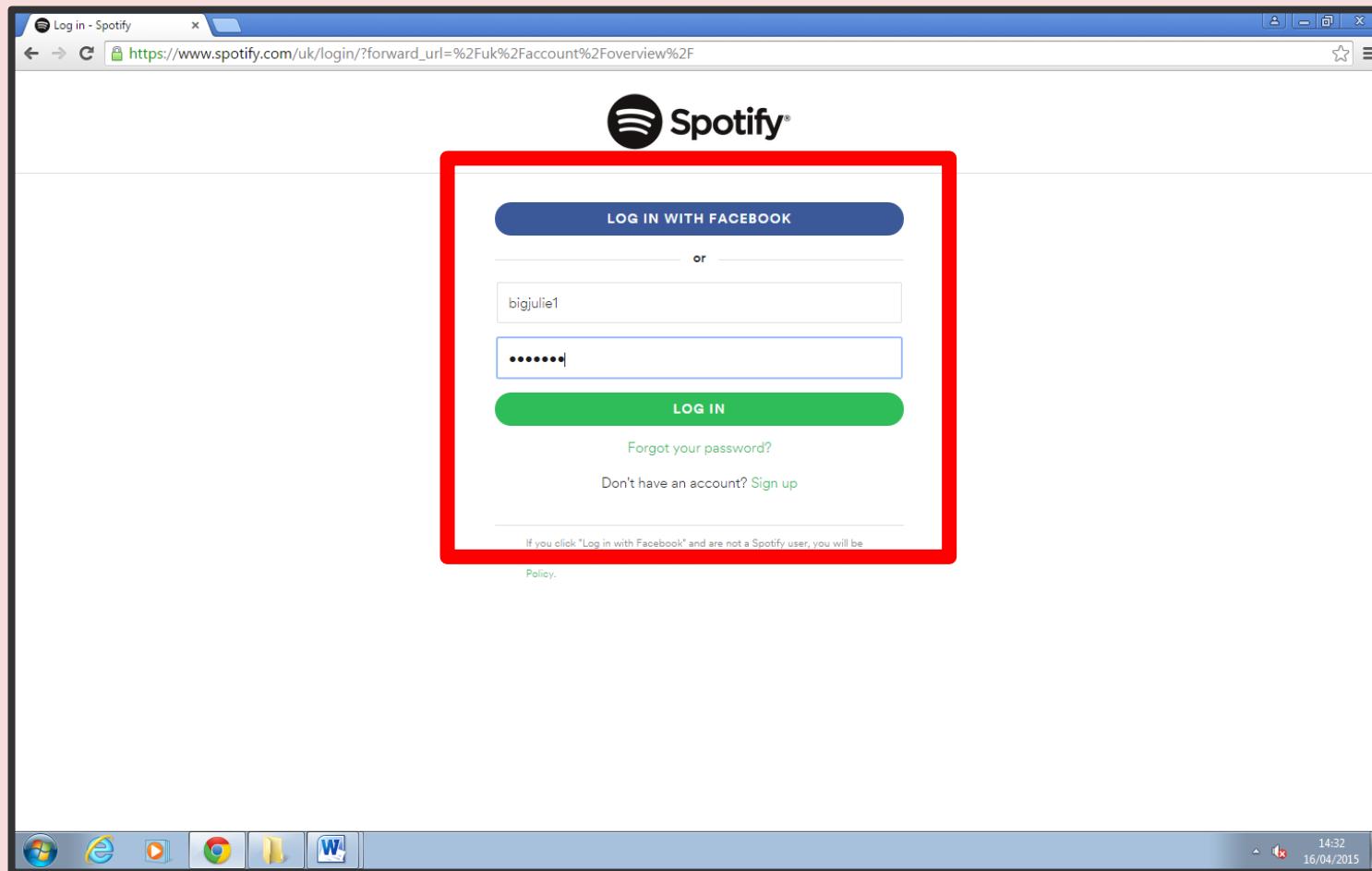
Installing Spotify

1. Visit www.spotify.com in your **browser** and click the log in link at the top right corner of the page.



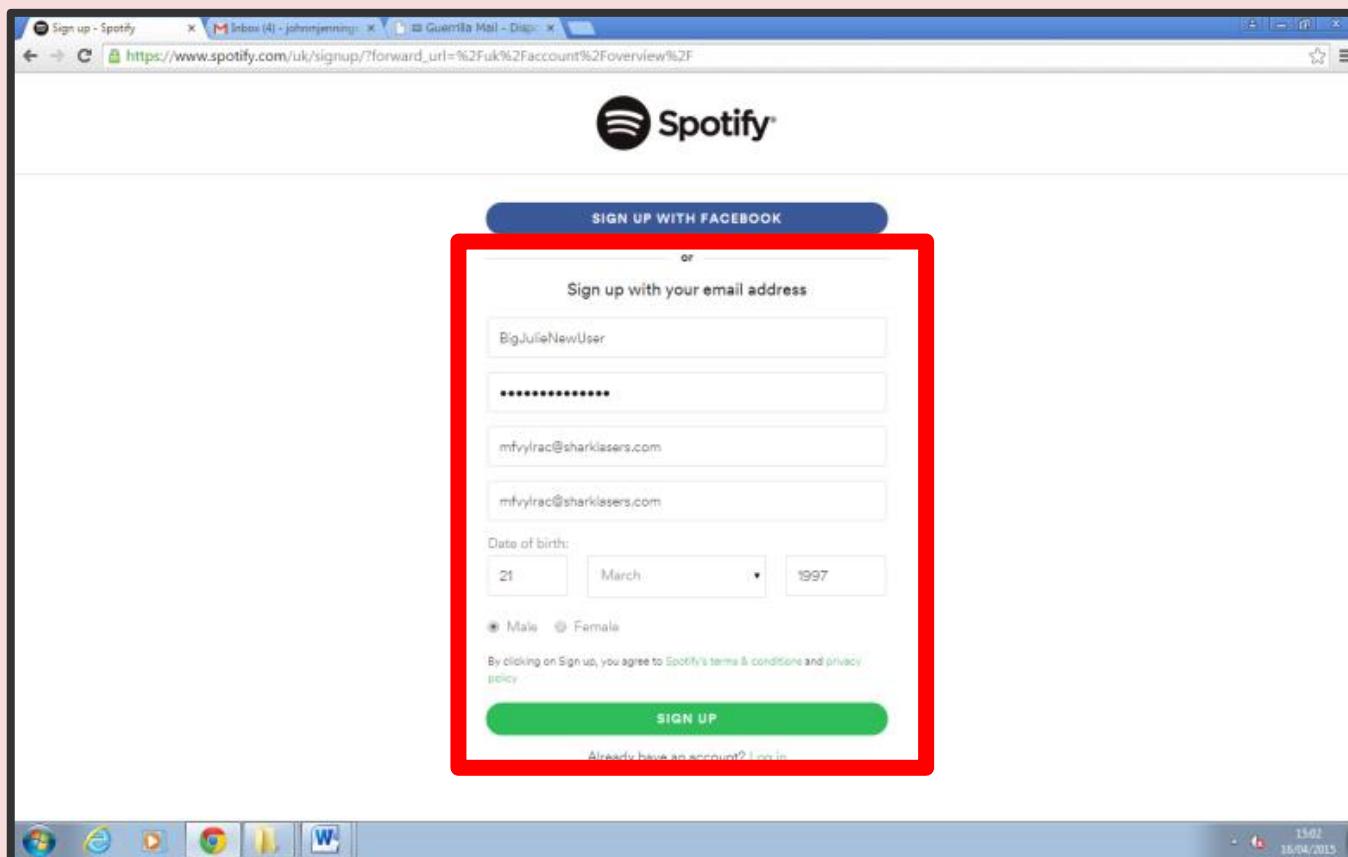
Installing Spotify

2. Enter your Spotify username and password and click the login button to continue.



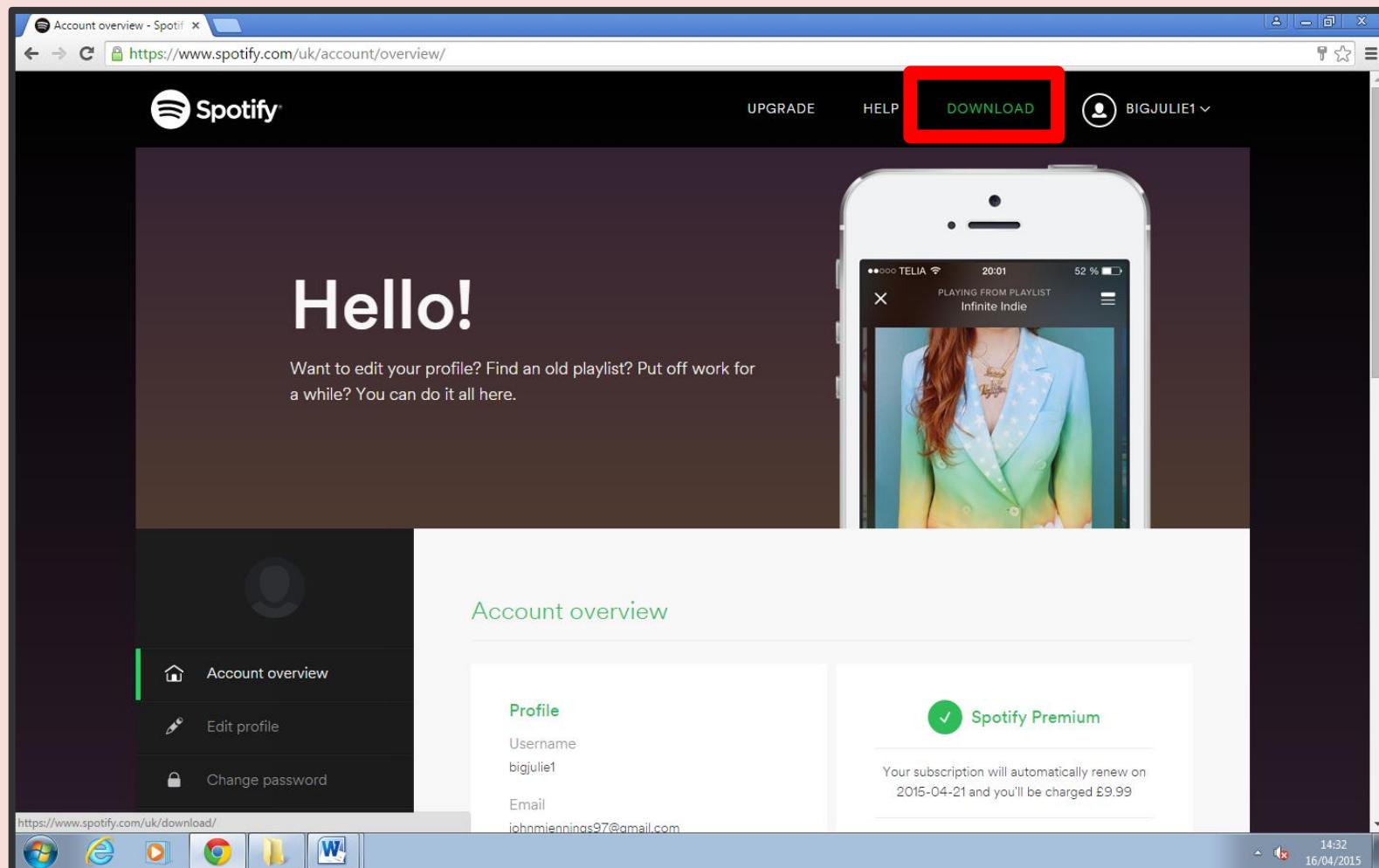
Installing Spotify

3. If you do not have a Spotify account then click the “sign up” link below the login button and either sign up with your Facebook account or create a new account by filling in the form below.



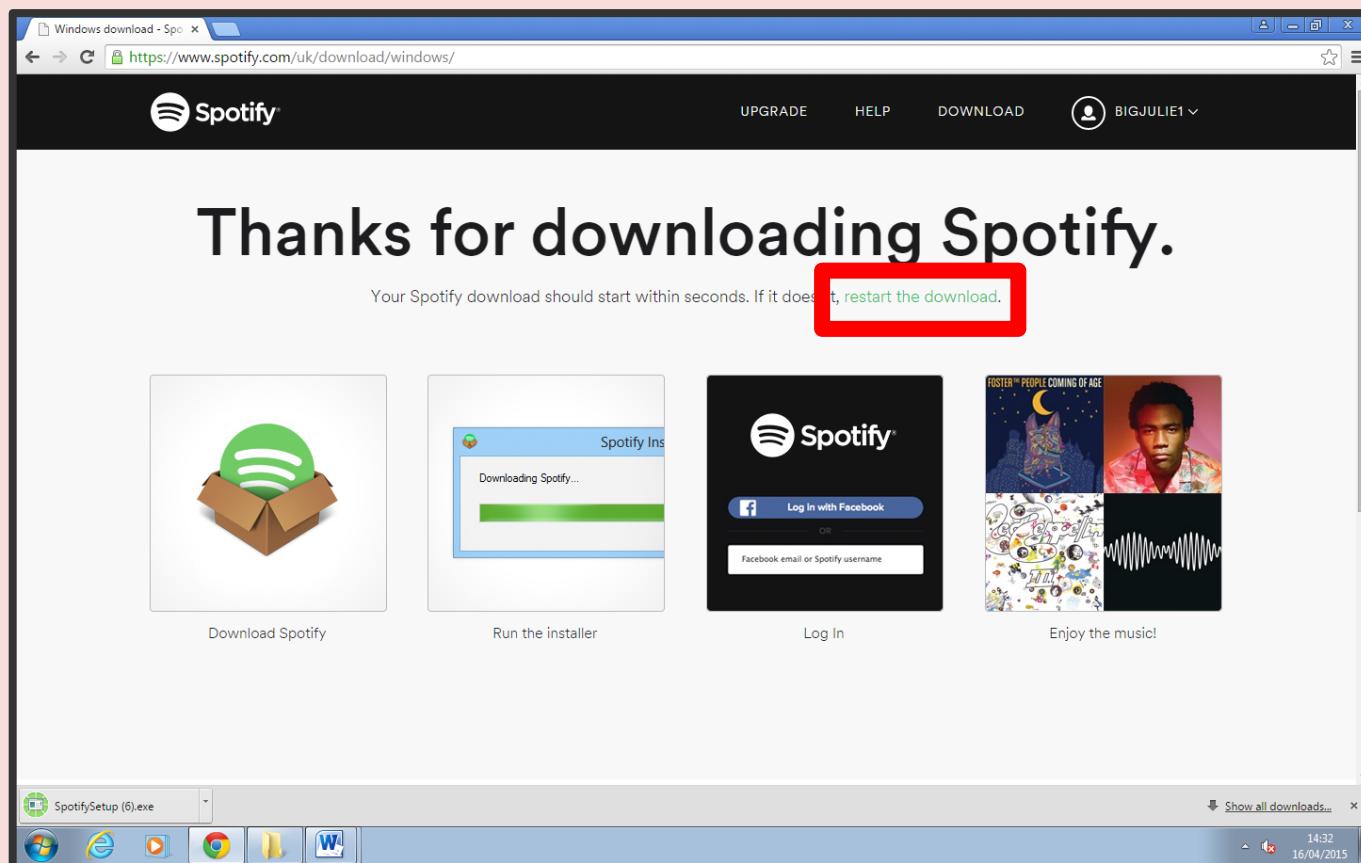
Installing Spotify

4. Click the download link in the top right corner of the page



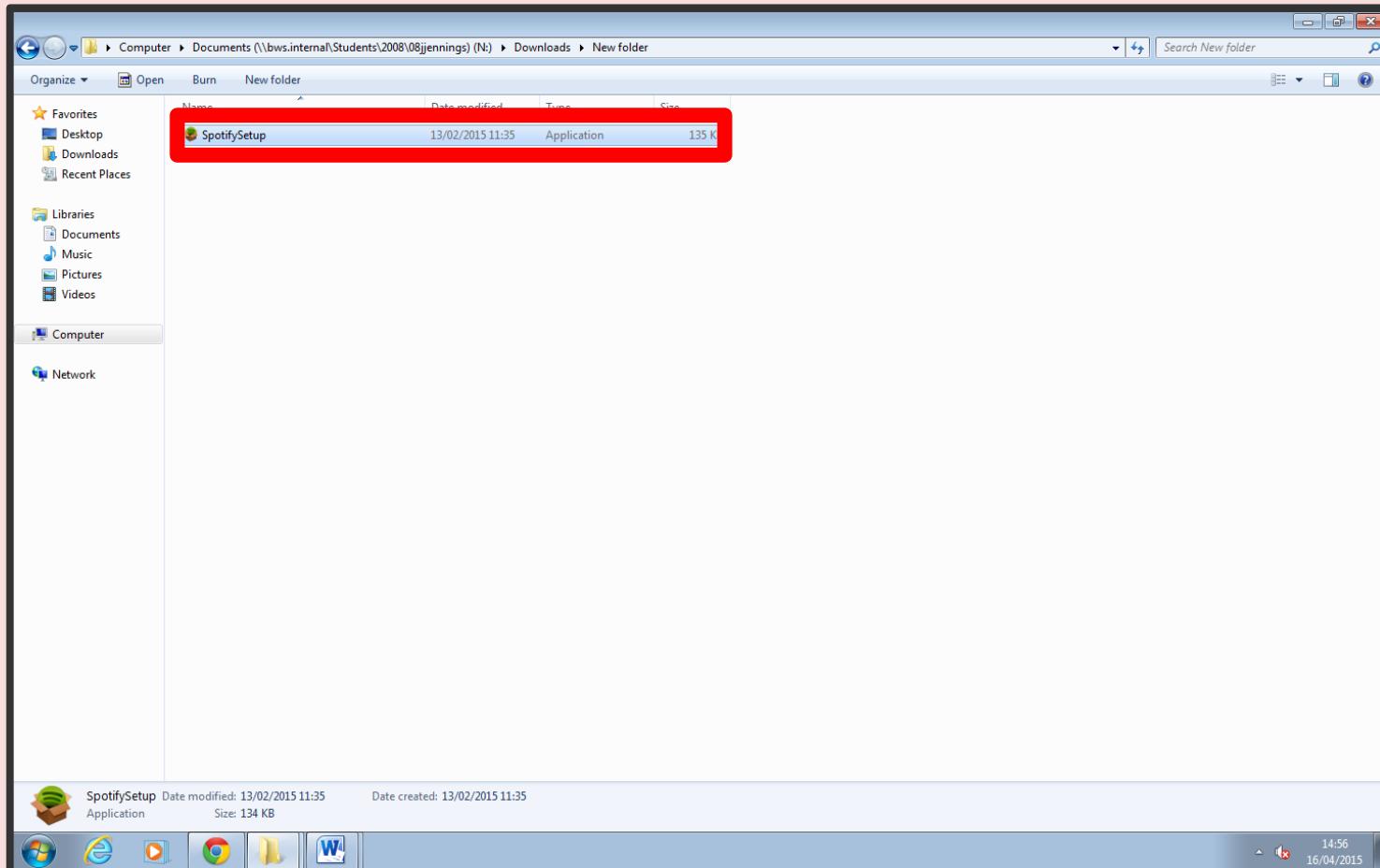
Installing Spotify

5. SpotifySetup.exe should begin downloading automatically. If the Spotify installer does not begin downloading then click the “restart the download” link in the centre of the page.



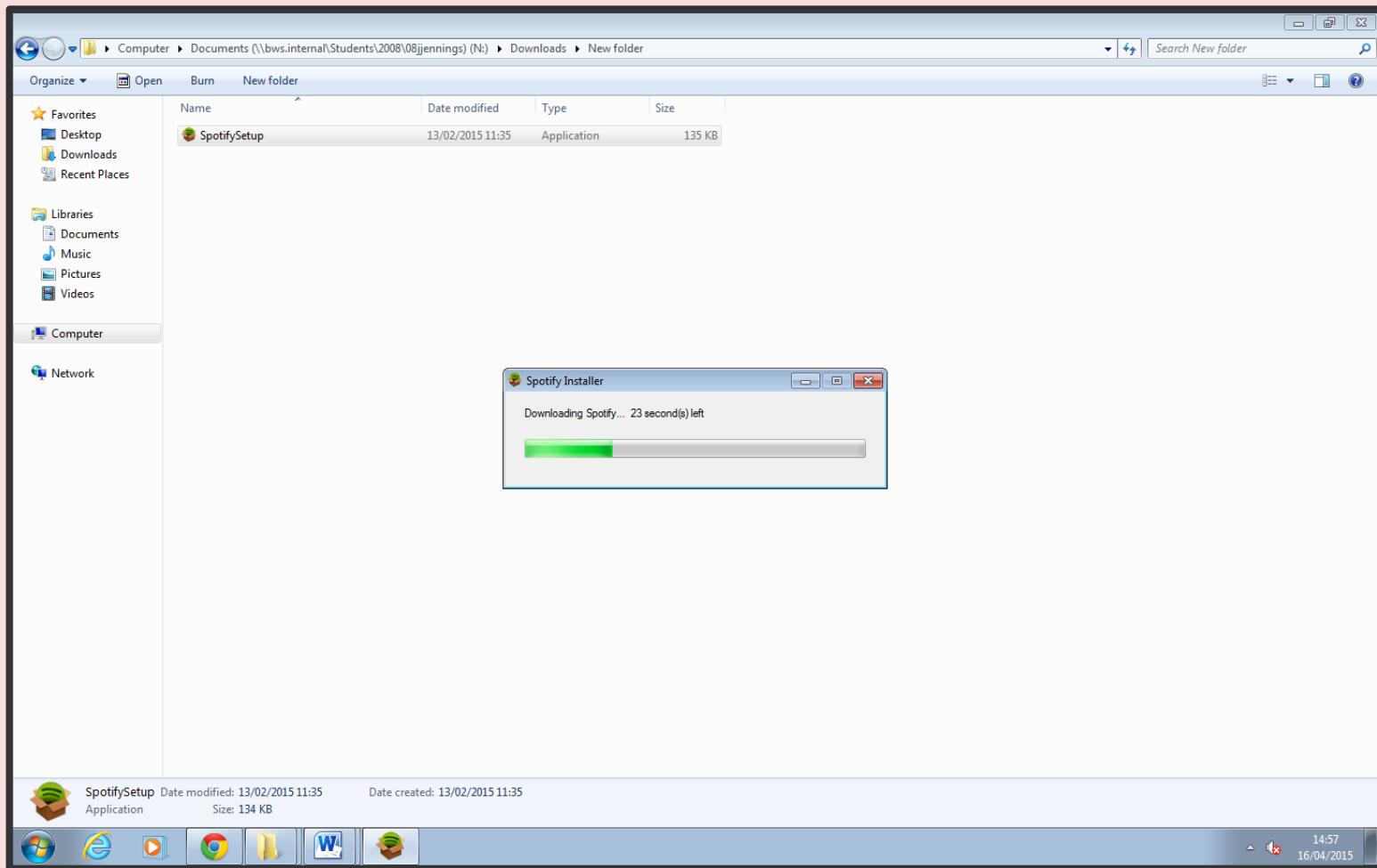
Installing Spotify

6. Navigate to SpotifySetup.exe in your file manager. Double click the file to start the installer.



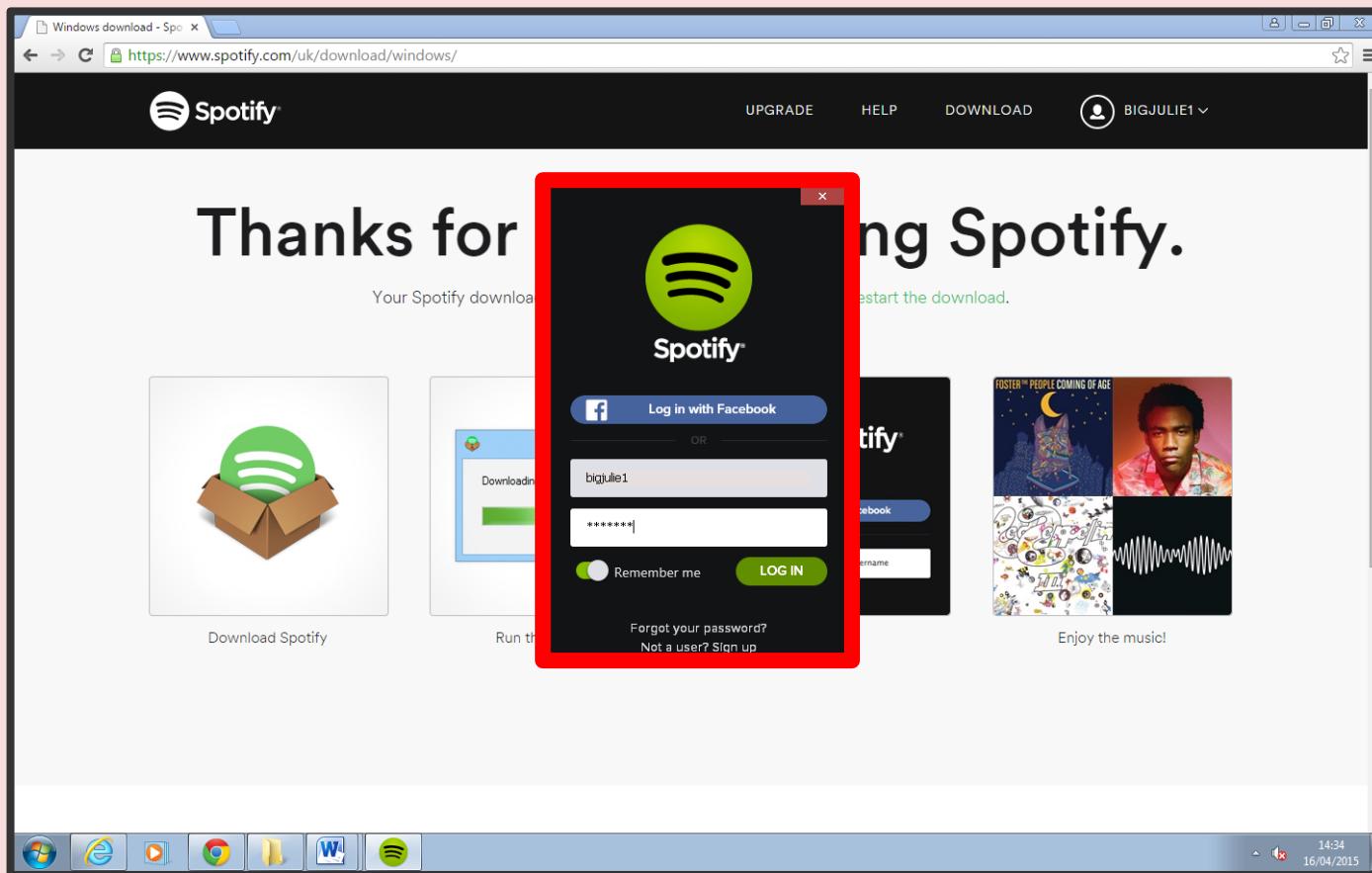
Installing Spotify

7. Wait until the Spotify installer has finished.



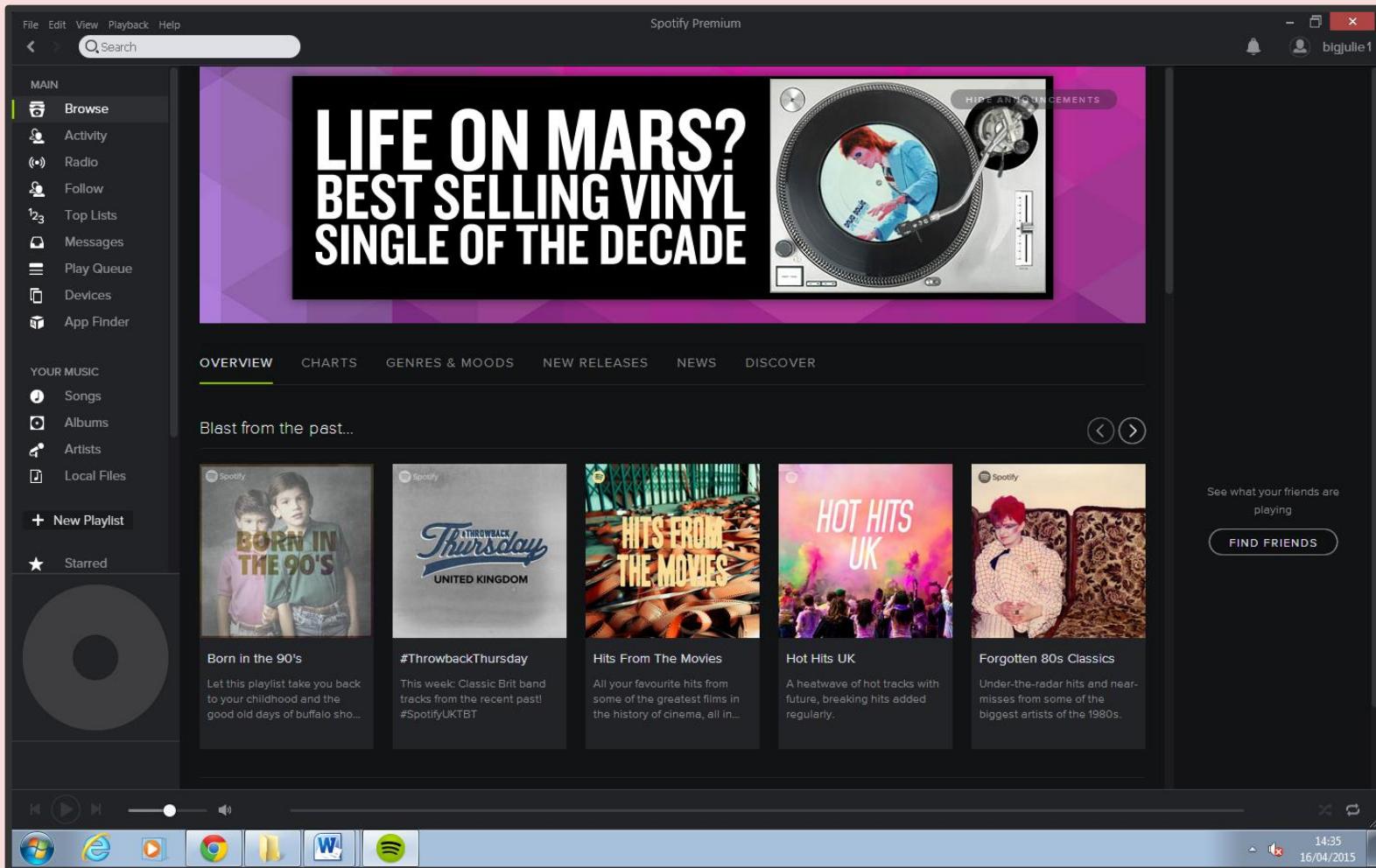
Installing Spotify

- When the installer has finished, Spotify will open at the login screen. Enter your account details and click the login button.



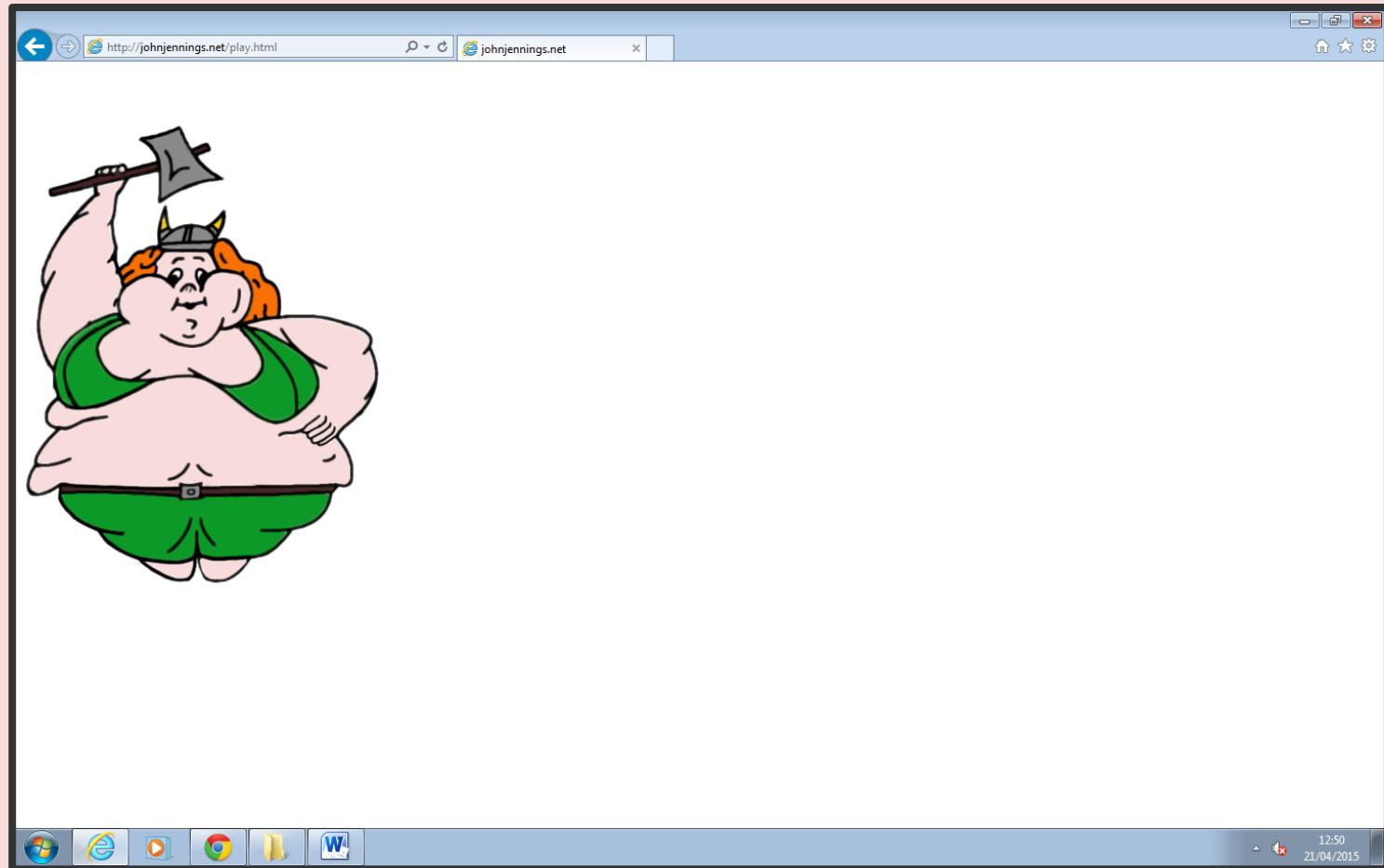
Installing Spotify

9. Spotify will now be open and ready to be controlled by Big Julie.



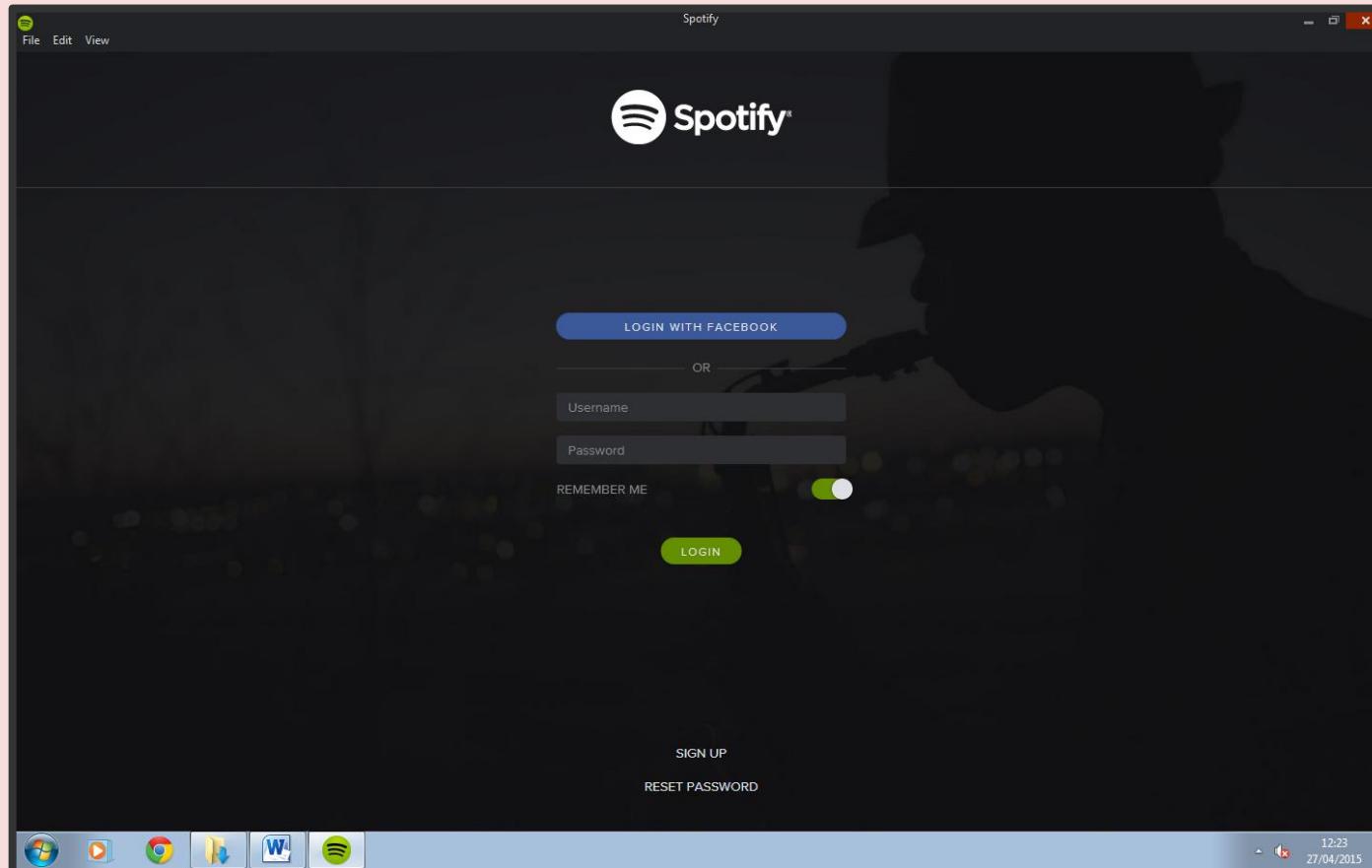
Player Setup

1. Visit www.johnjennings.net/play.html in your **browser**



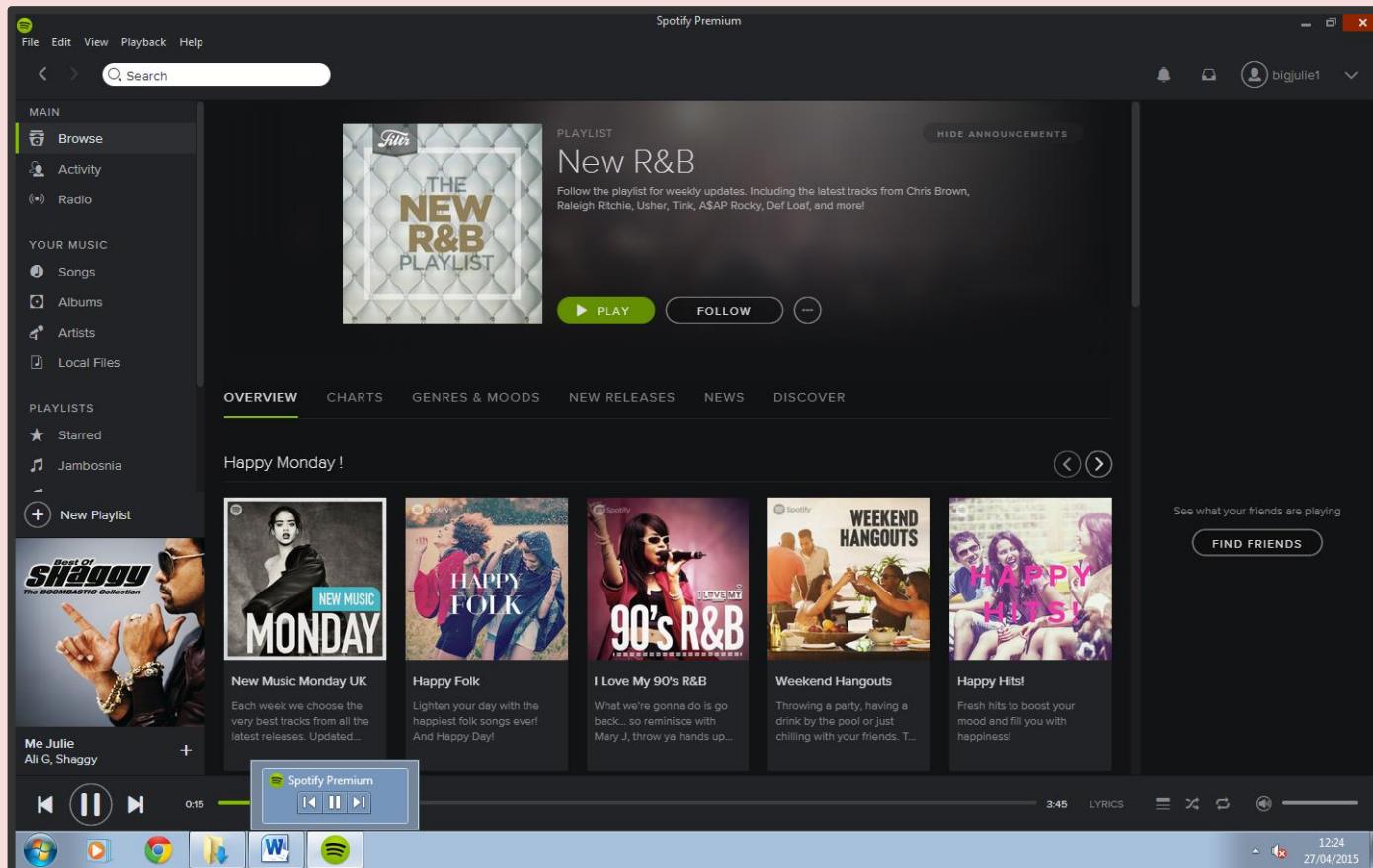
Player Setup

2. The **Spotify client** may open automatically. If it does not then ensure that **JavaScript** is enabled in your **browser** and refresh the page.



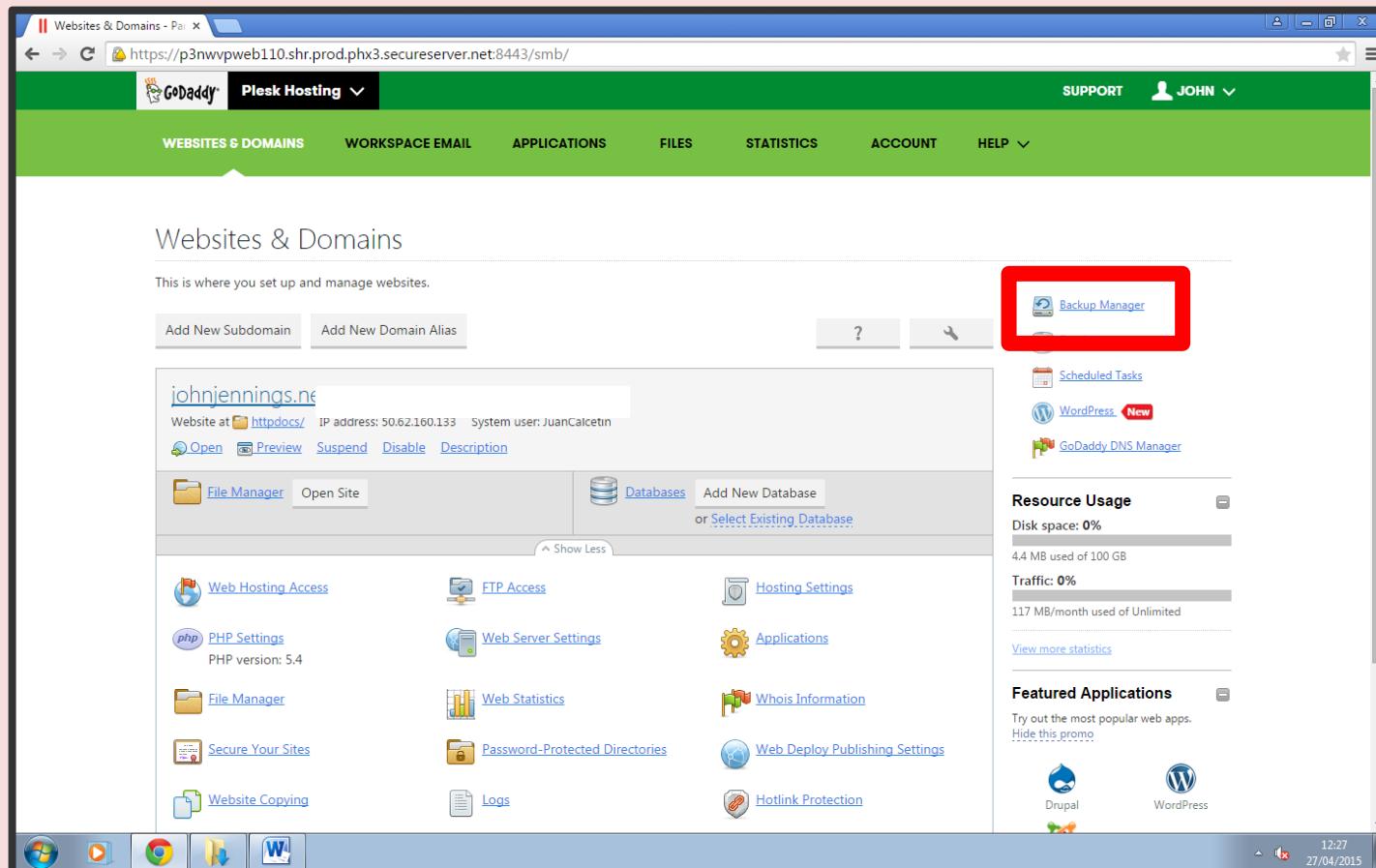
Player Setup

3. Spotify should now begin playing the music that has been requested. Ensure that speakers are connected to the system and the volume is not zero.



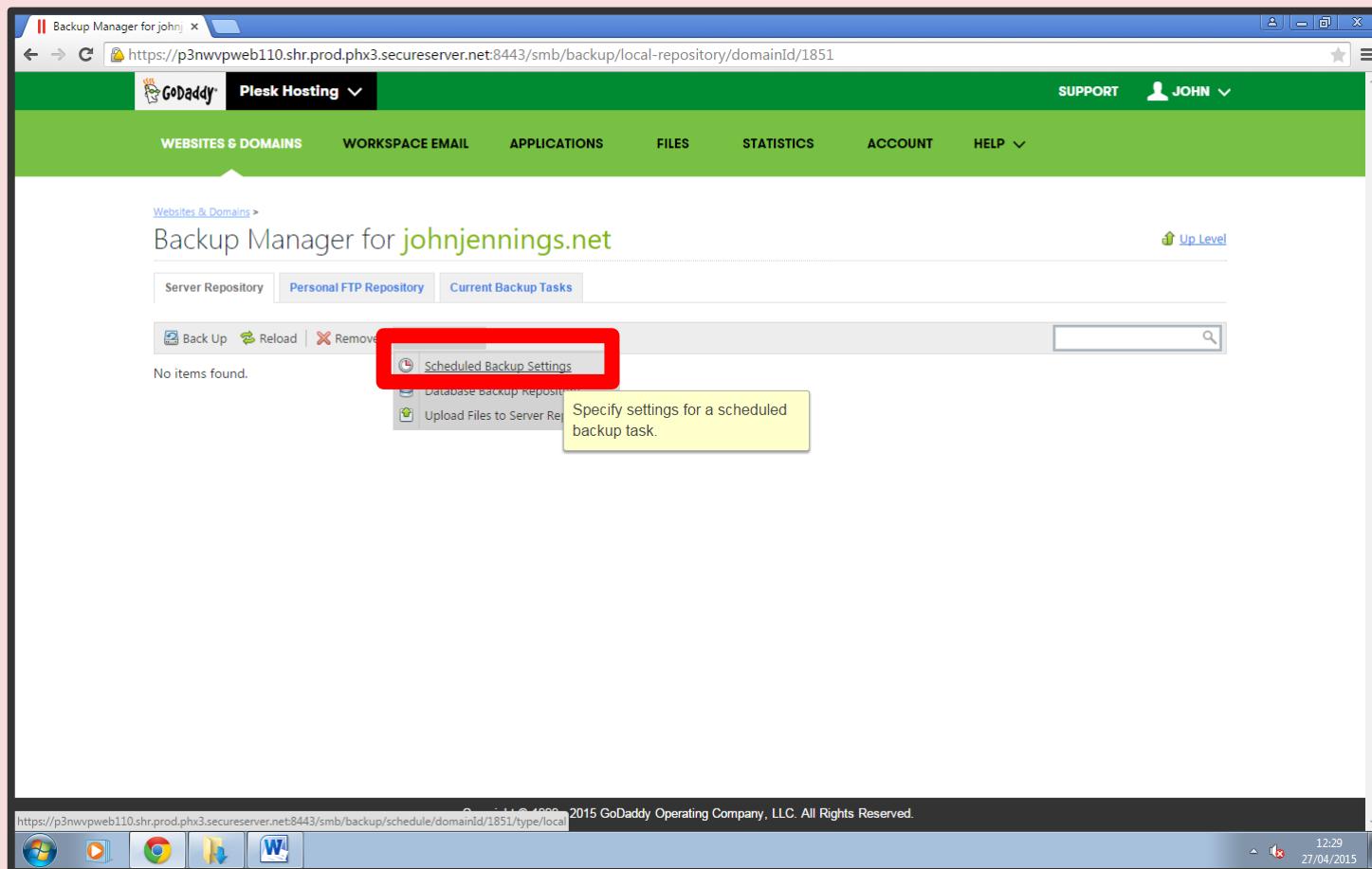
Backup Routine

1. Navigate to the johnjennings.net hosting on **GoDaddy** and click the “Backup Manager” link



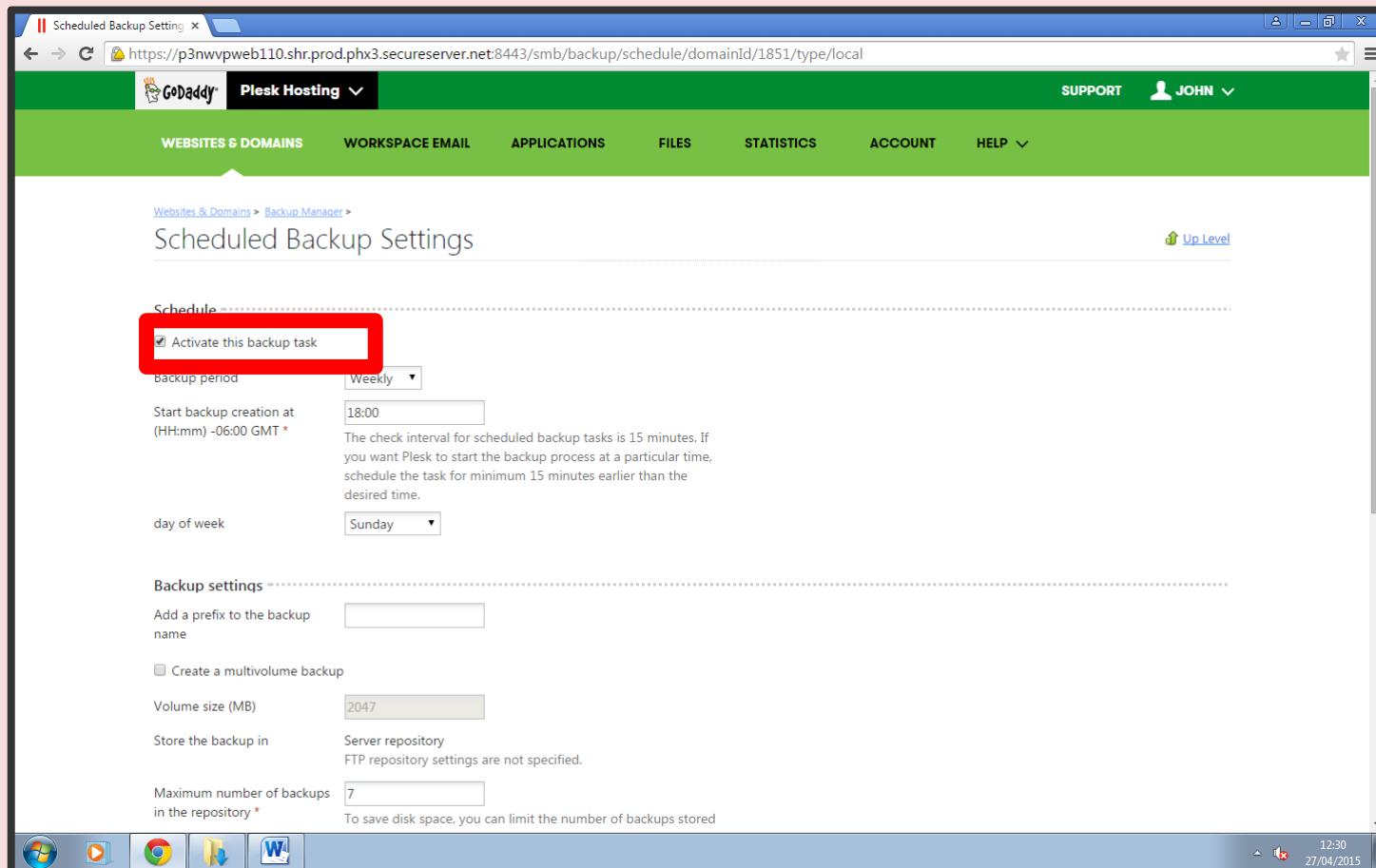
Backup Routine

2. Click the “More Actions” tab and select “Scheduled Backup Settings” from the drop down menu.



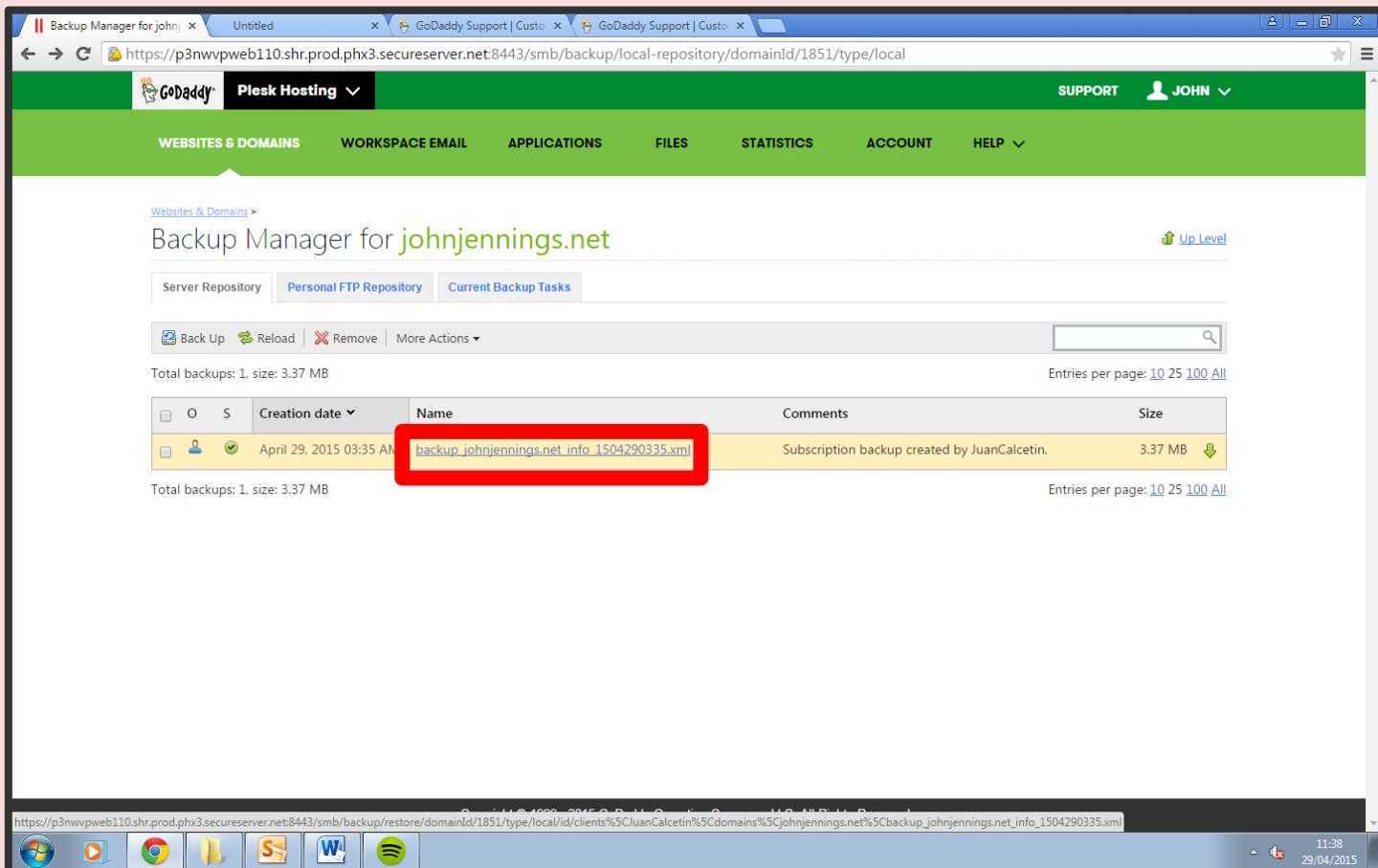
Backup Routine

3. Select the “Activate this backup task” tick box to begin weekly backups at 6 PM on a Sunday. You can configure your own **backup routine** with the form below.



Backup Routine

4. To restore from a backup, repeat step 1 and click on the backup that you wish to restore.



The screenshot shows a web browser window for the GoDaddy Backup Manager. The URL is <https://p3nwwpweb110.shr.prod.phx3.secureserver.net:8443/smb/backup/local-repository/domainId/1851/type/local>. The page title is "Backup Manager for johnjennings.net". The navigation bar includes links for WEBSITES & DOMAINS, WORKSPACE EMAIL, APPLICATIONS, FILES, STATISTICS, ACCOUNT, and HELP. The main content area displays a table of backups:

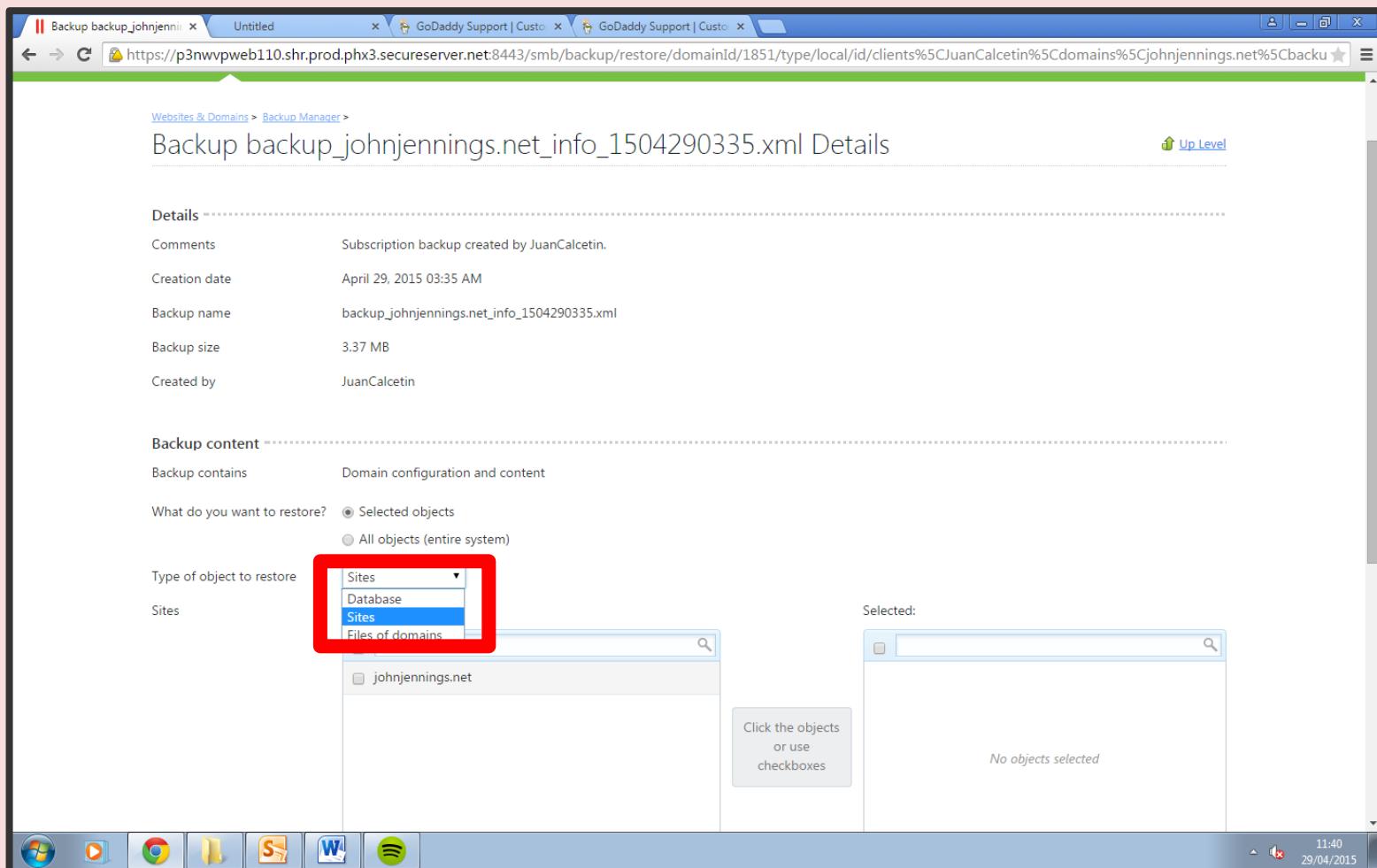
O	S	Creation date	Name	Comments	Size
<input type="checkbox"/>	<input checked="" type="checkbox"/>	April 29, 2015 03:35 AM	backup_johnjennings.net_info_1504290335.xml	Subscription backup created by JuanCalcentin.	3.37 MB 

Total backups: 1, size: 3.37 MB Entries per page: 10 25 100 All

A red box highlights the row for the backup file "backup_johnjennings.net_info_1504290335.xml".

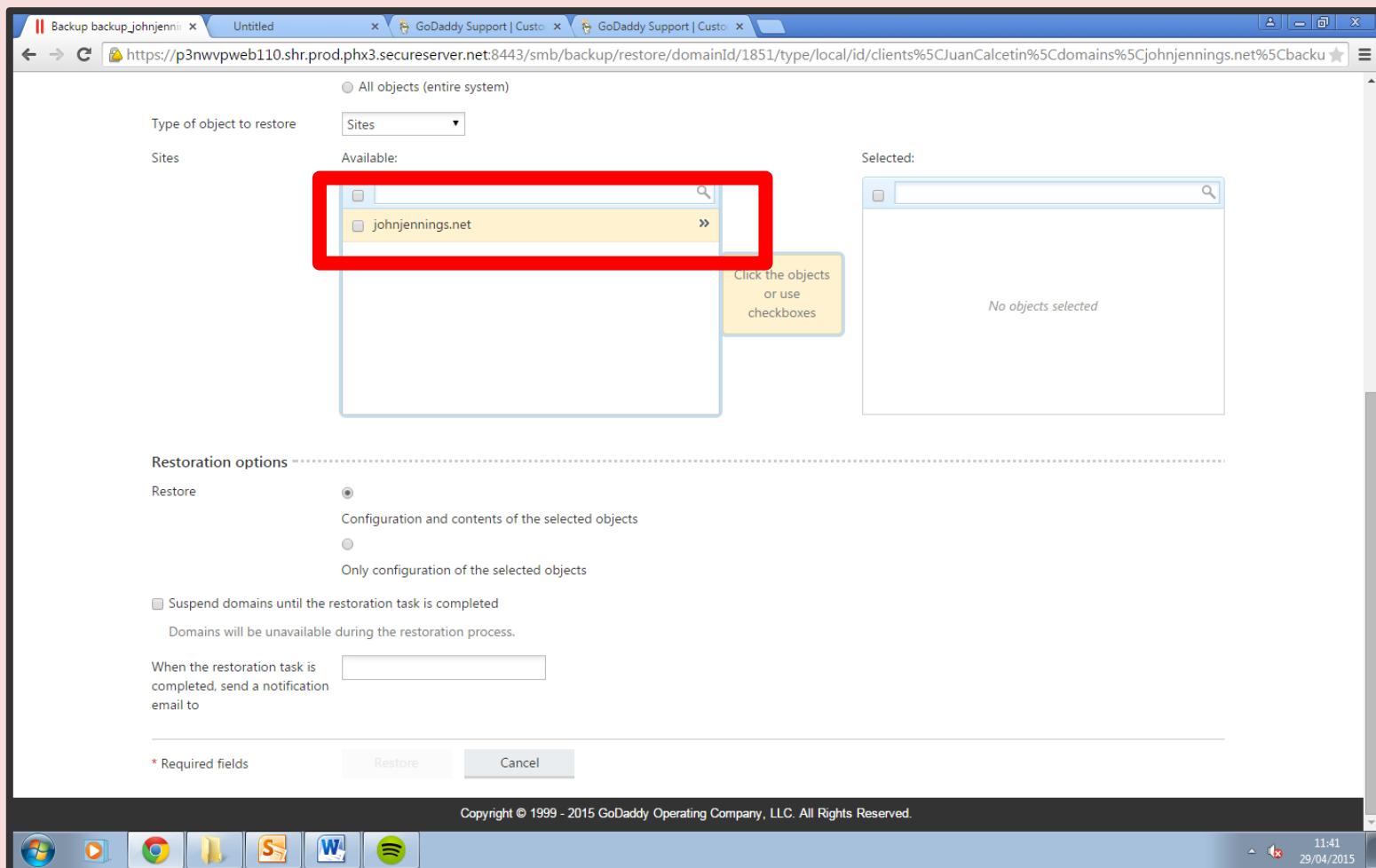
Backup Routine

5. Select “Sites” from the “Type of object to restore” drop down menu .



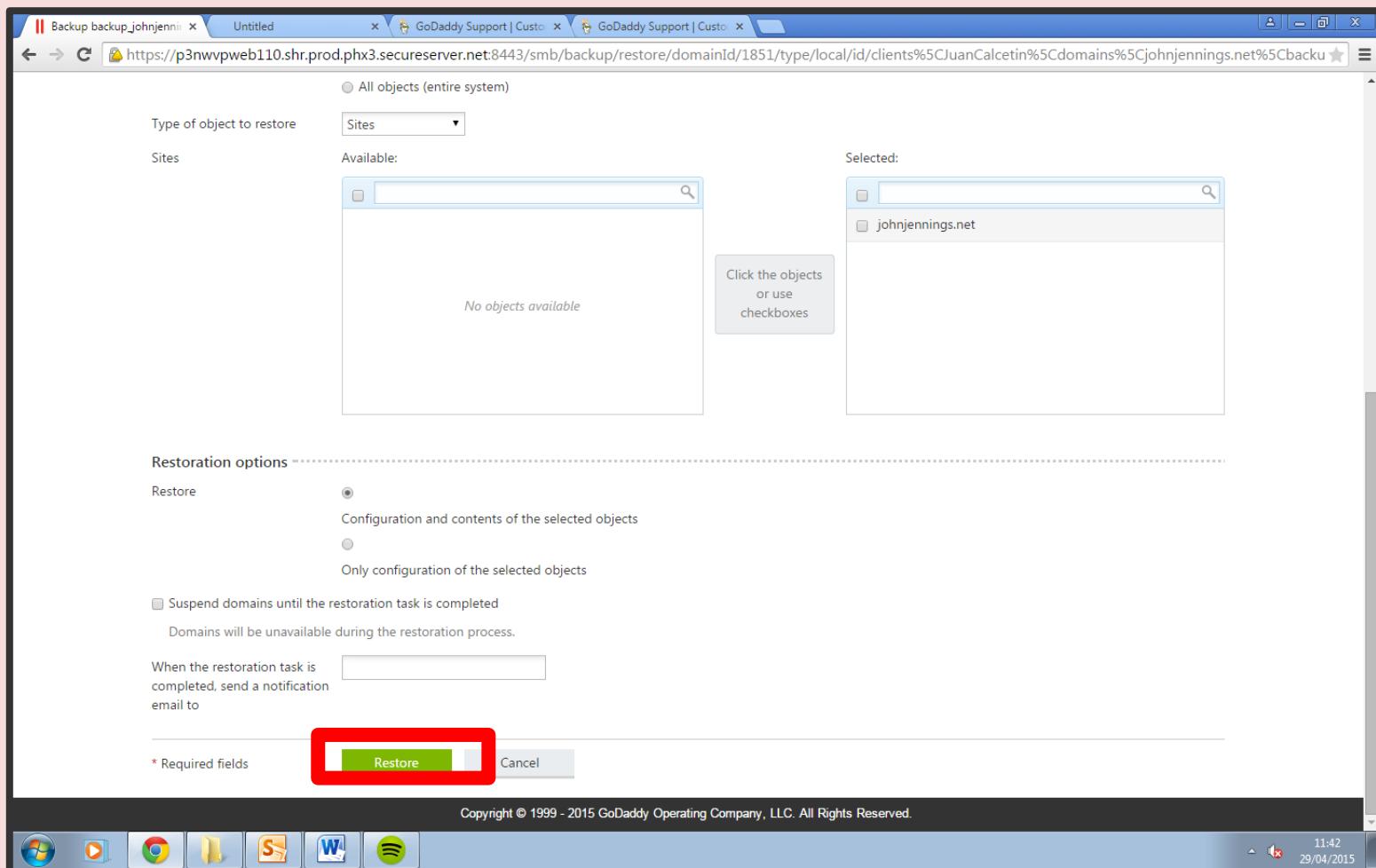
Backup Routine

6. Select johnjennings.net from the list that appears



Backup Routine

7. Click the Restore button to restore Big Julie to the backup you have selected.

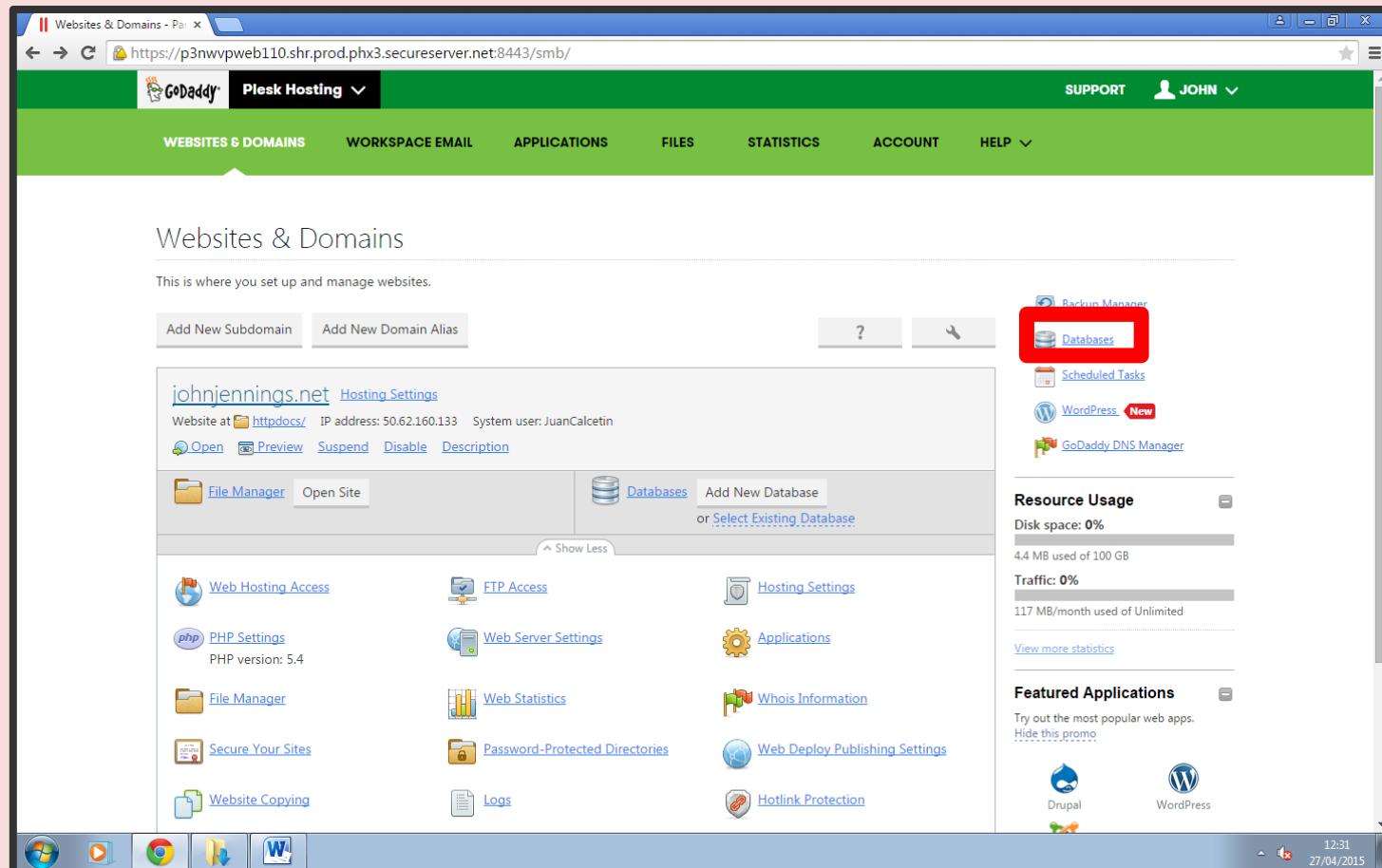


Typical Use



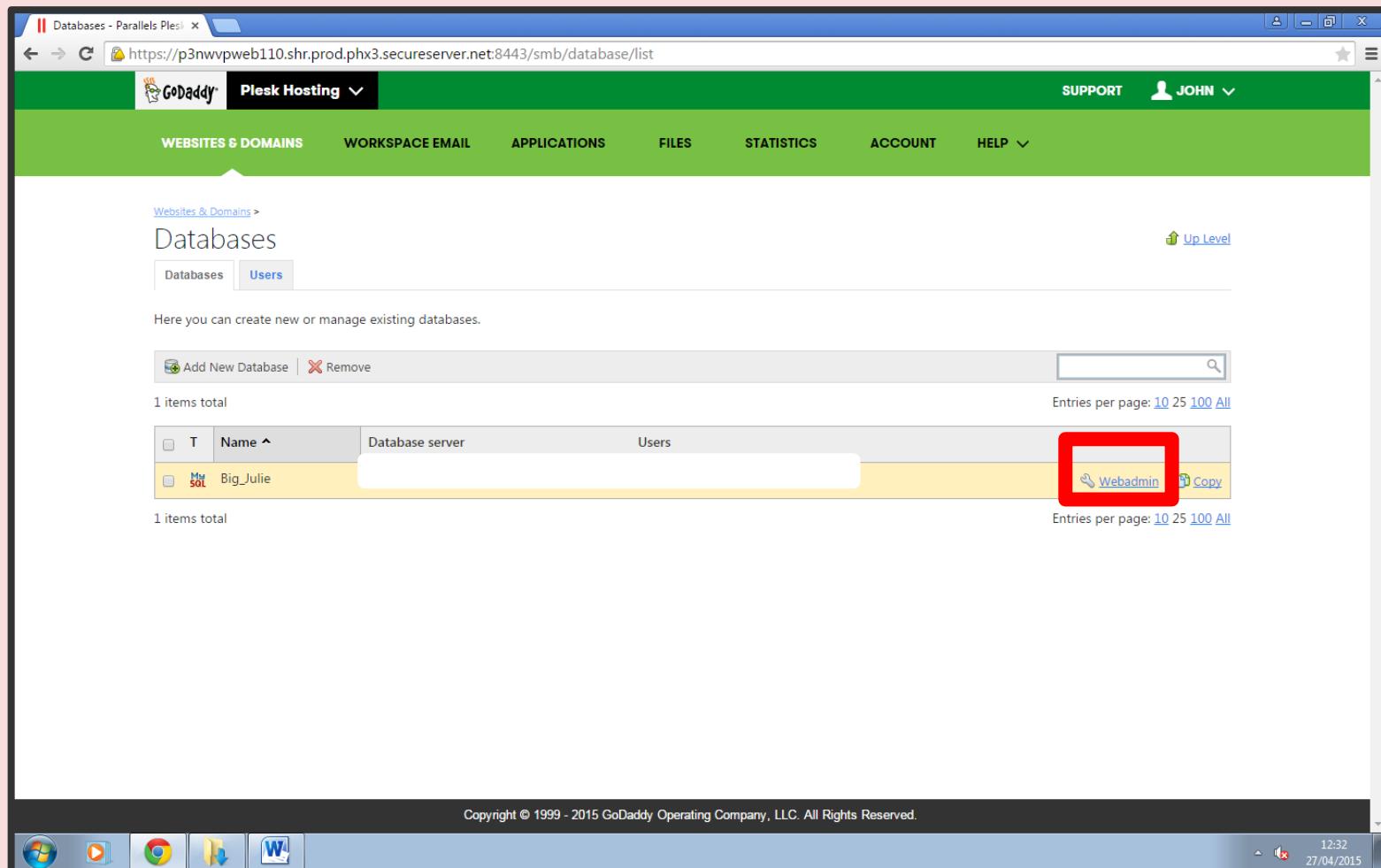
Adding Users

1. Navigate to the johnjennings.net hosting on **GoDaddy** and click the “Databases” link.



Adding Users

2. Click the “Webadmin” link on the right side of the page.



Adding Users

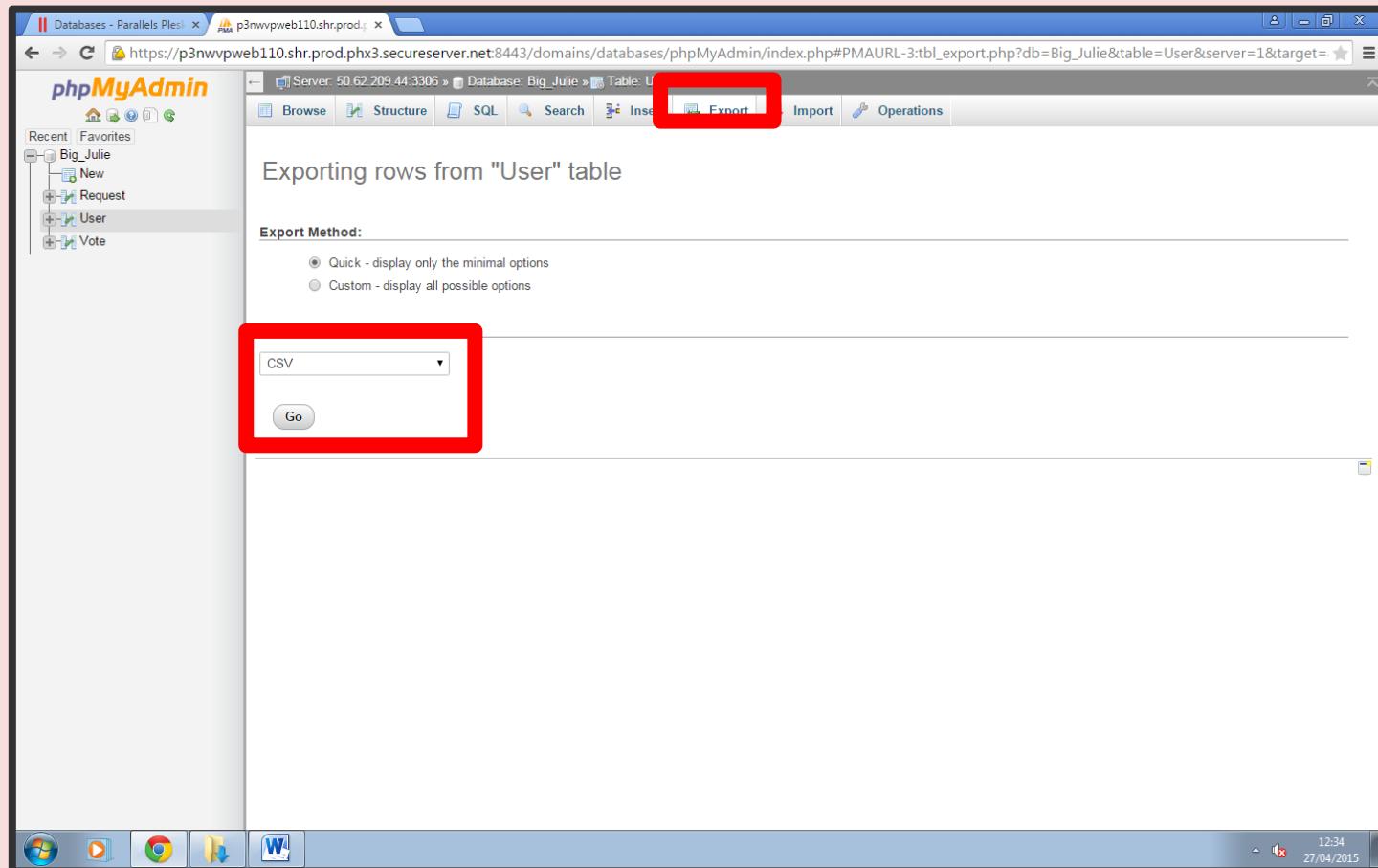
3. On the sidebar, click “Big_Julie” and then “User”

The screenshot shows the phpMyAdmin interface for the database 'Big_Julie'. The sidebar on the left has a red box around the 'Request' item under the 'Big_Julie' section. The main area displays a table of three tables: Request, User, and Vote. The 'User' table is selected, showing 116 rows. A modal window titled 'Create table' is open in the foreground, asking for a table name and number of columns (set to 4). The status bar at the bottom shows the URL 'p3nwvpweb110.shr.prod.phx3.secureserver.net:8443/domains/databases/.../sql.php?ser...' and the date/time '27/04/2015 12:33'.

Table	Action	Rows	Type	Collation	Size	Overhead
Request	Browse Structure Search Insert Empty Drop	29	MyISAM	utf8_general_ci	26.2 Kib	7.8Kib
User	Browse Structure Search Insert Empty Drop	116	MyISAM	utf8_general_ci	7.9 Kib	-
Vote	Browse Structure Search Insert Empty Drop	151	MyISAM	utf8_general_ci	16.3 Kib	-
3 tables	Sum	296	MyISAM	utf8_general_ci	50.4 Kib	7.8 Kib

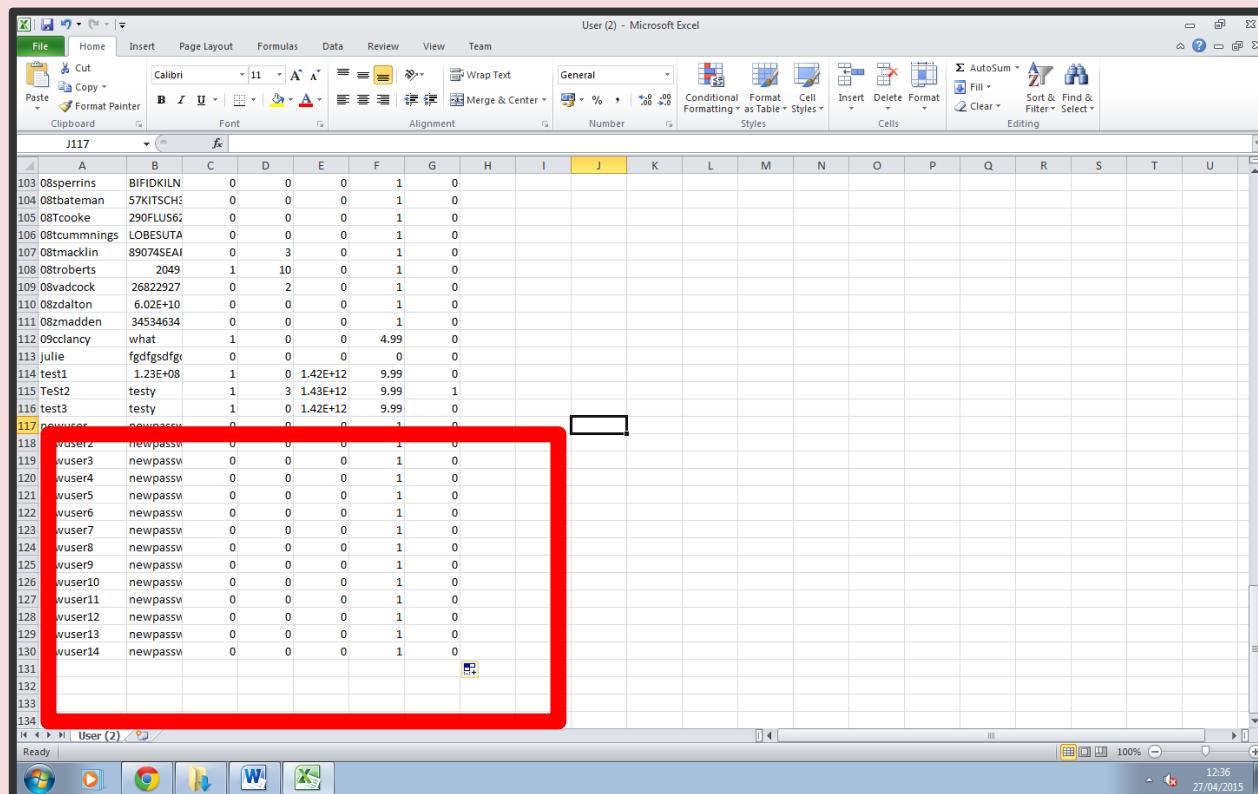
Adding Users

4. Click the “Export” tab and select **CSV** from the format drop down menu then click the Go button.



Adding Users

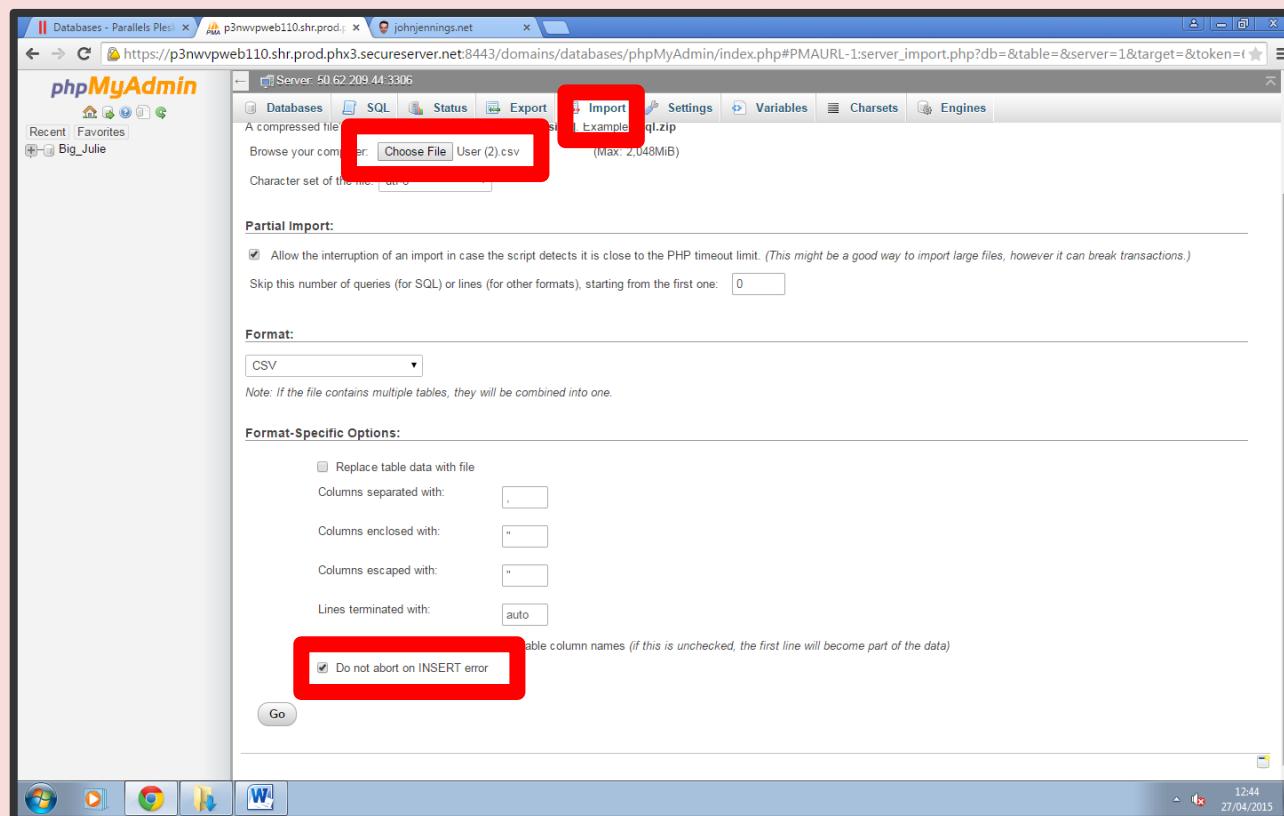
5. Open the **CSV file** in Microsoft Excel and insert a row for each new user. A list of usernames can be obtained from the school database and passwords can be randomly generated by **juliePassGen.exe**. The other columns should be defaulted to 0,0,0,1,0.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
103	08sperrins	BiFDIKLN	0	0	0	1	0													
104	08tbateman	57KITSCH	0	0	0	1	0													
105	08Tcooke	290FLUS62	0	0	0	1	0													
106	08tcummings	LOBESUTA	0	0	0	1	0													
107	08tmacklin	89074SEAI	0	3	0	1	0													
108	08troberts	2049	1	10	0	1	0													
109	08vadcock	26822927	0	2	0	1	0													
110	08zdalton	6.02E+10	0	0	0	1	0													
111	08zmadden	34534634	0	0	0	1	0													
112	09cclancy	what	1	0	0	4.99	0													
113	julie	fgdfgsdfg	0	0	0	0	0													
114	test1	1.23E+08	1	0	1.42E+12	9.99	0													
115	TeSt2	testy	1	3	1.43E+12	9.99	1													
116	test3	testy	1	0	1.42E+12	9.99	0													
117	newuser	newpassw	0	0	0	1	0													
118	wuser2	newpassw	0	0	0	1	0													
119	wuser3	newpassw	0	0	0	1	0													
120	wuser4	newpassw	0	0	0	1	0													
121	wuser5	newpassw	0	0	0	1	0													
122	wuser6	newpassw	0	0	0	1	0													
123	wuser7	newpassw	0	0	0	1	0													
124	wuser8	newpassw	0	0	0	1	0													
125	wuser9	newpassw	0	0	0	1	0													
126	wuser10	newpassw	0	0	0	1	0													
127	wuser11	newpassw	0	0	0	1	0													
128	wuser12	newpassw	0	0	0	1	0													
129	wuser13	newpassw	0	0	0	1	0													
130	wuser14	newpassw	0	0	0	1	0													
131																				
132																				
133																				
134																				

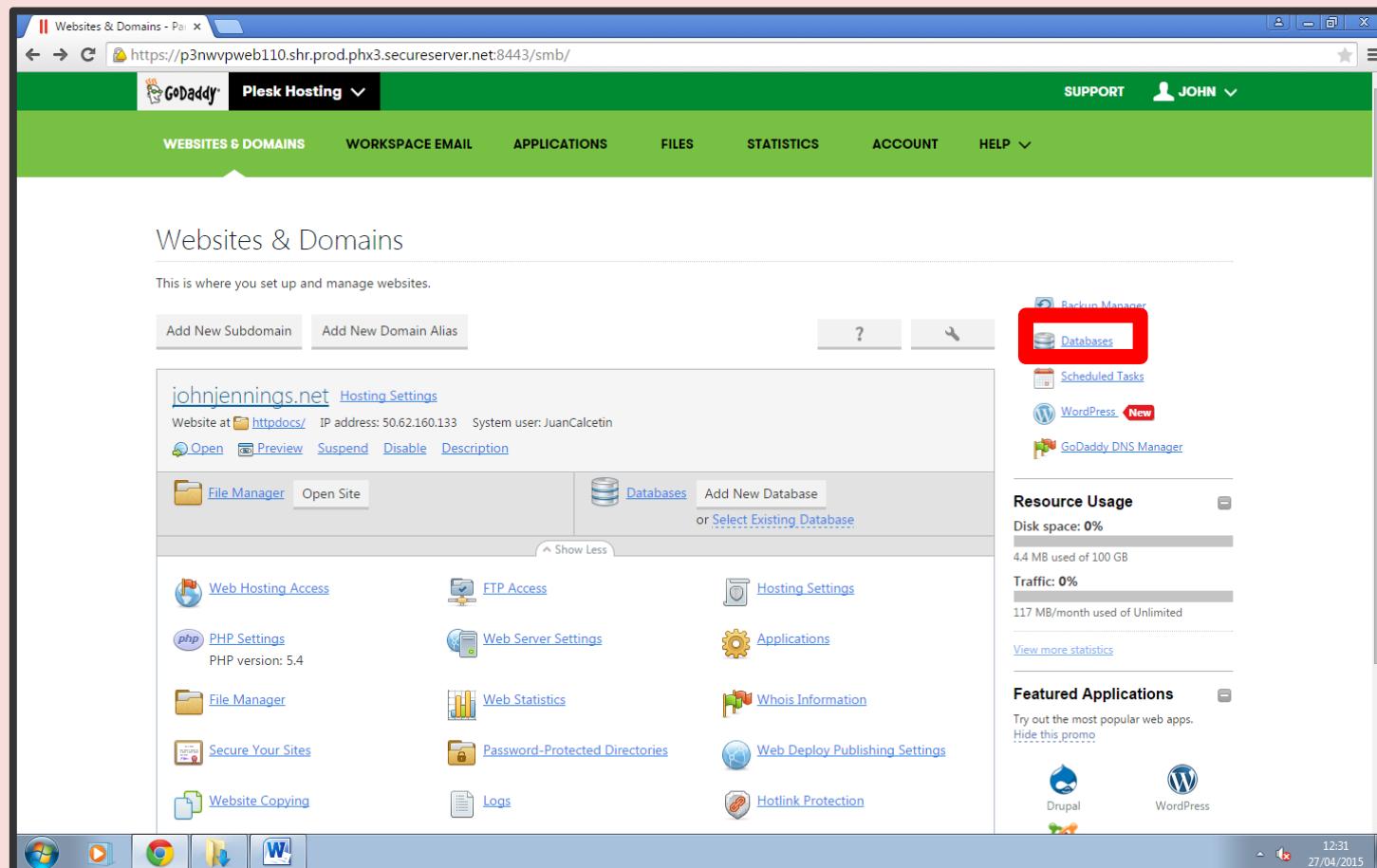
Adding Users

6. Save the **CSV file** and return to the **browser**. Click the “Import” tab and upload the new CSV file by clicking the “Choose file” button. Select the “Do not abort on INSERT error” checkbox and press the Go button.



Removing Users

1. Navigate to the johnjennings.net hosting on **GoDaddy** and click the “Databases” link.



Removing Users

2. Click the “Webadmin” link on the right side of the page.

The screenshot shows a web browser window for 'Databases - Parallels Plesk'. The URL is <https://p3nwvpweb110.shr.prod.phx3.secureserver.net:8443/smb/database/list>. The interface includes a top navigation bar with 'GoDaddy' logo, 'Plesk Hosting', 'SUPPORT', and 'JOHN'. Below it is a green header with links for 'WEBSITES & DOMAINS', 'WORKSPACE EMAIL', 'APPLICATIONS', 'FILES', 'STATISTICS', 'ACCOUNT', and 'HELP'. The main content area is titled 'Databases' with tabs for 'Databases' and 'Users'. It displays a message: 'Here you can create new or manage existing databases.' Below this are two sections: one for 'Database server' and one for 'Users'. In the 'Database server' section, there is a table with one item: 'My SQL' under 'Name' and 'Big_Julie' under 'Database server'. To the right of this table are 'Webadmin' and 'Copy' links, with 'Webadmin' being highlighted by a red box. The 'Users' section below also has a 'Webadmin' link. At the bottom, there are links for 'Entries per page: 10 25 100 All' and 'Copyright © 1999 - 2015 GoDaddy Operating Company, LLC. All Rights Reserved.' The status bar at the bottom right shows the time as 12:32 and the date as 27/04/2015.

Removing Users

3. On the sidebar, click “Big_Julie” and then “User”

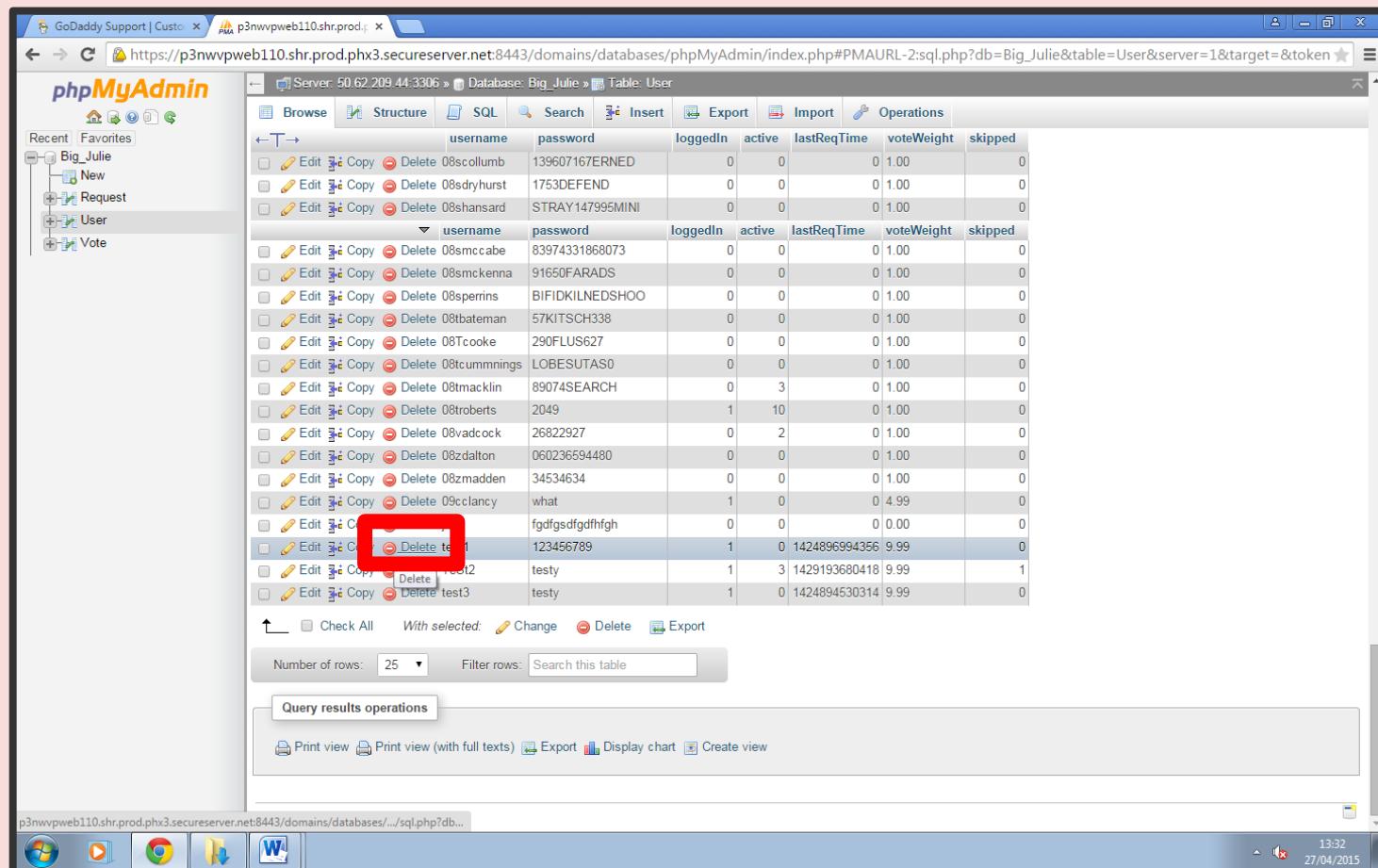
The screenshot shows the phpMyAdmin interface for the 'Big_Julie' database. The left sidebar lists tables: 'Request', 'User', 'Vote', and '3 tables Sum'. The 'User' table is selected and highlighted with a red box. The main area displays the 'User' table structure with the following data:

Table	Action	Rows	Type	Collation	Size	Overhead
Request	Browse Structure Search Insert Empty Drop	29	MyISAM	utf8_general_ci	26.2 Kib	7.8Kib
User	Browse Structure Search Insert Empty Drop	116	MyISAM	utf8_general_ci	7.9 Kib	-
Vote	Browse Structure Search Insert Empty Drop	151	MyISAM	utf8_general_ci	16.3 Kib	-
3 tables Sum		296	MyISAM	utf8_general_ci	50.4 Kib	7.8 Kib

Below the table, there is a 'Create table' form with fields for 'Name:' (empty) and 'Number of columns:' (set to 4). A 'Go' button is located to the right of the form.

Removing Users

4. Navigate to the record for the user you wish to remove and click the delete button

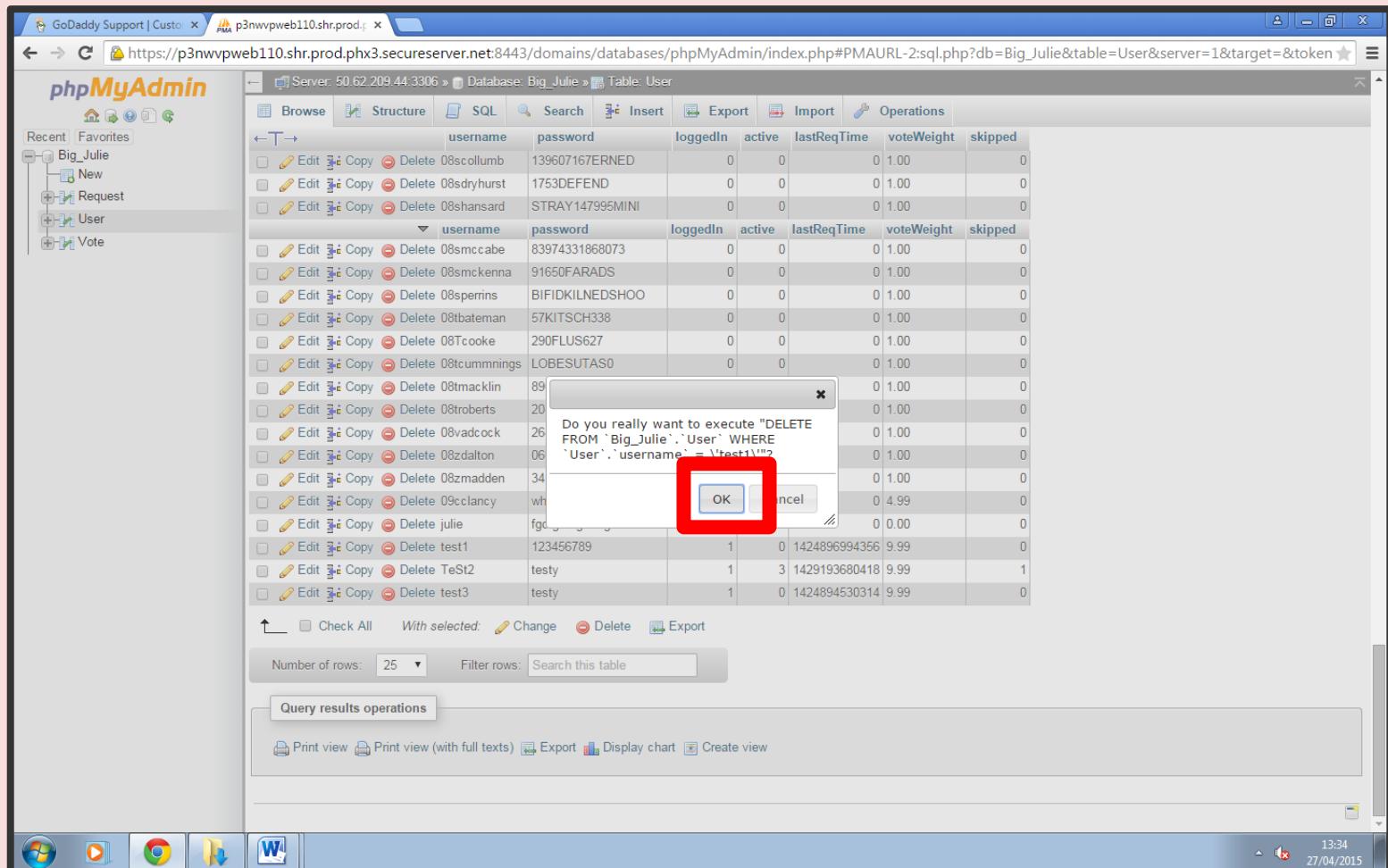


The screenshot shows the phpMyAdmin interface for a database named 'Big_Julie'. The left sidebar shows a tree structure with 'Big_Julie' selected. The main area displays the 'User' table with the following columns: username, password, loggedIn, active, lastReqTime, voteWeight, and skipped. A red box highlights the 'Delete' link for the first row, which has a 'username' of 'test1' and a 'password' of '123456789'. Below the table, there are buttons for 'Check All', 'With selected:', 'Change', 'Delete', and 'Export'. At the bottom, there's a 'Query results operations' section with links for 'Print view', 'Print view (with full texts)', 'Export', 'Display chart', and 'Create view'. The status bar at the bottom right shows the time as 13:32 and the date as 27/04/2015.

username	password	loggedin	active	lastReqTime	voteWeight	skipped
08scolumnb	139607167ERNED	0	0	0	1.00	0
08sdyhurst	1753DEFEND	0	0	0	1.00	0
08shansard	STRAY147995MINI	0	0	0	1.00	0
08smccabe	83974331868073	0	0	0	1.00	0
08smckenna	91650FARADS	0	0	0	1.00	0
08sperrins	BIFIDKILNEDSHOO	0	0	0	1.00	0
08tbateman	57KITSCH338	0	0	0	1.00	0
08tcooke	290FLUS627	0	0	0	1.00	0
08tcummings	LOBESUTAS0	0	0	0	1.00	0
08tmacklin	89074SEARCH	0	3	0	1.00	0
08troberts	2049	1	10	0	1.00	0
08vadcock	26822927	0	2	0	1.00	0
08zdalton	060236594480	0	0	0	1.00	0
08zmadden	34534634	0	0	0	1.00	0
09cclancy	what	1	0	0	4.99	0
	fdfgsdfgdfhgh	0	0	0	0.00	0
test1	123456789	1	0	1424896994356	9.99	0
test2	testy	1	3	1429193680418	9.99	1
test3	testy	1	0	1424894530314	9.99	0

Removing Users

5. Click the OK button in the alert that appears.



Turning Player On / Off

1. To turn off the Big Julie player, simply exit the browser window with www.johnjennings.net/play.html. The system will finish playing the last song and will resume once the page is visited again.



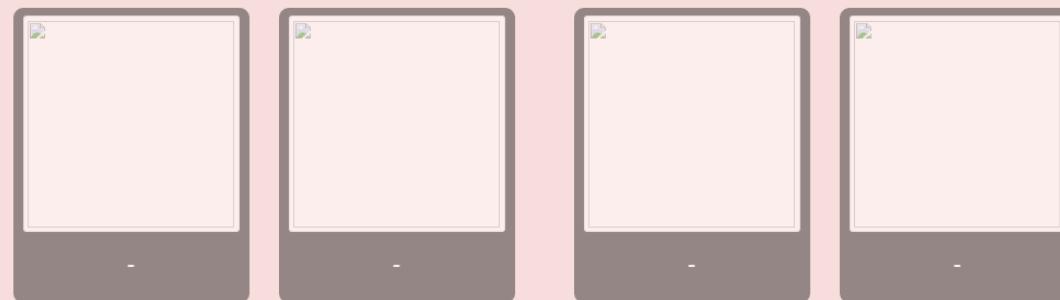
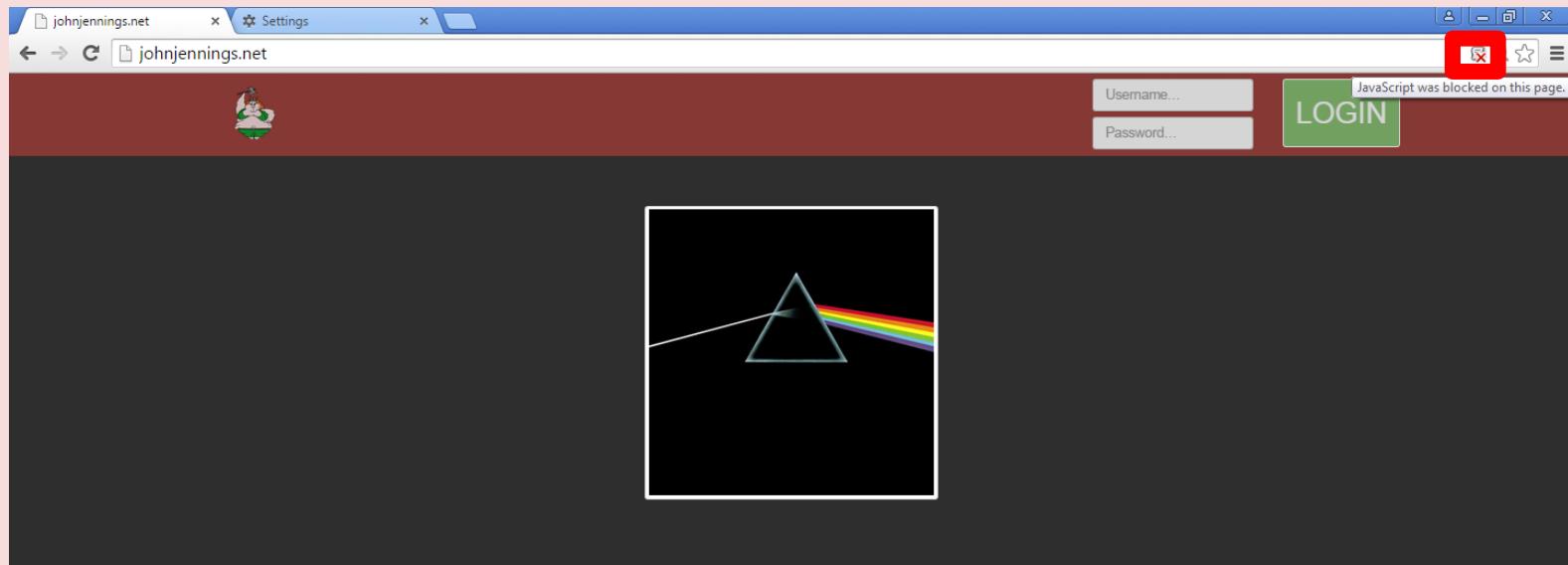
Troubleshooting

This section aims to solve some of the common problems that you may experience when using Big Julie:

- Enabling JavaScript (37-39)
- No songs playing (40-41)

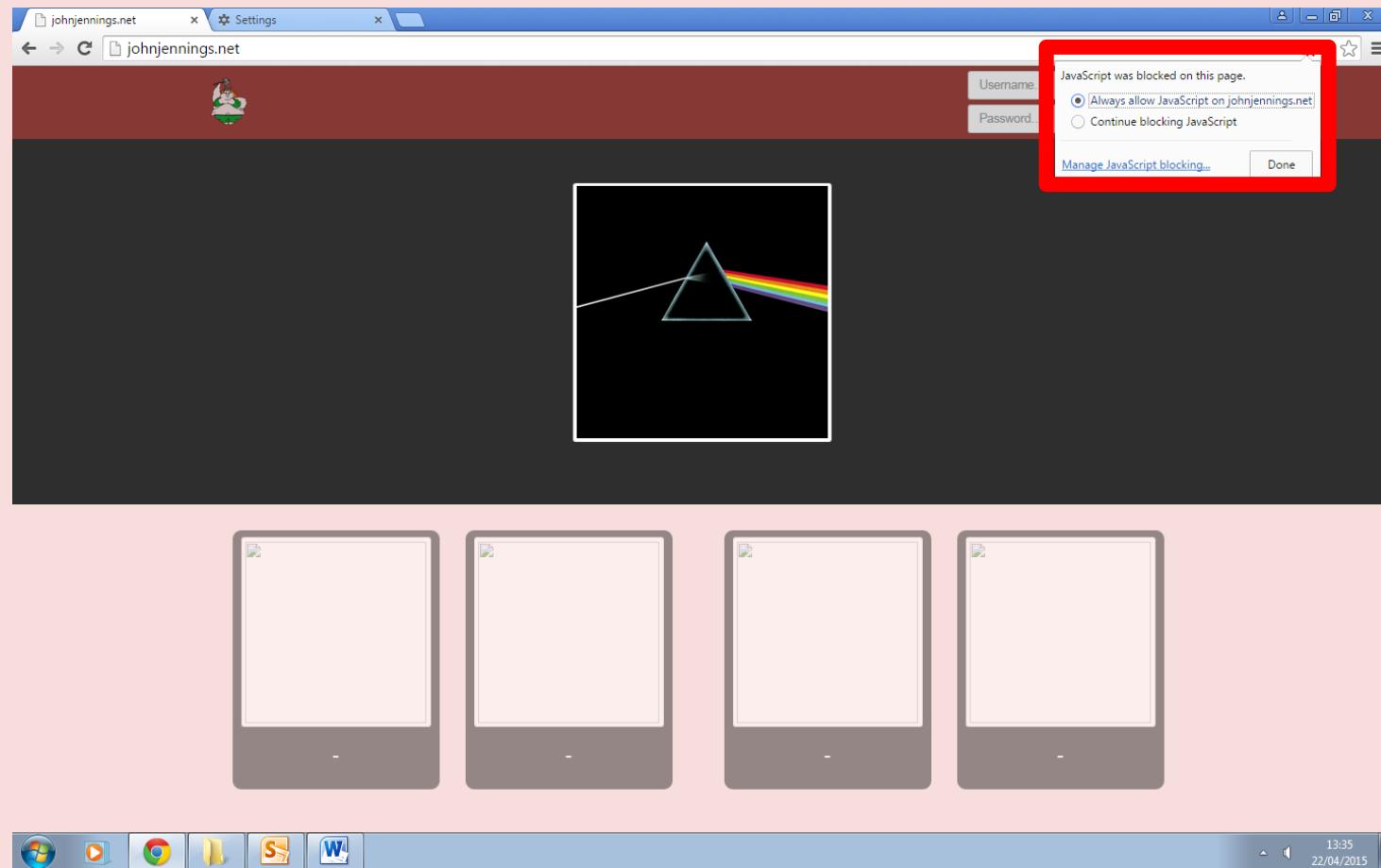
Enabling JavaScript

1. Click the **JavaScript** icon to the left of the magnifying glass in the address bar.



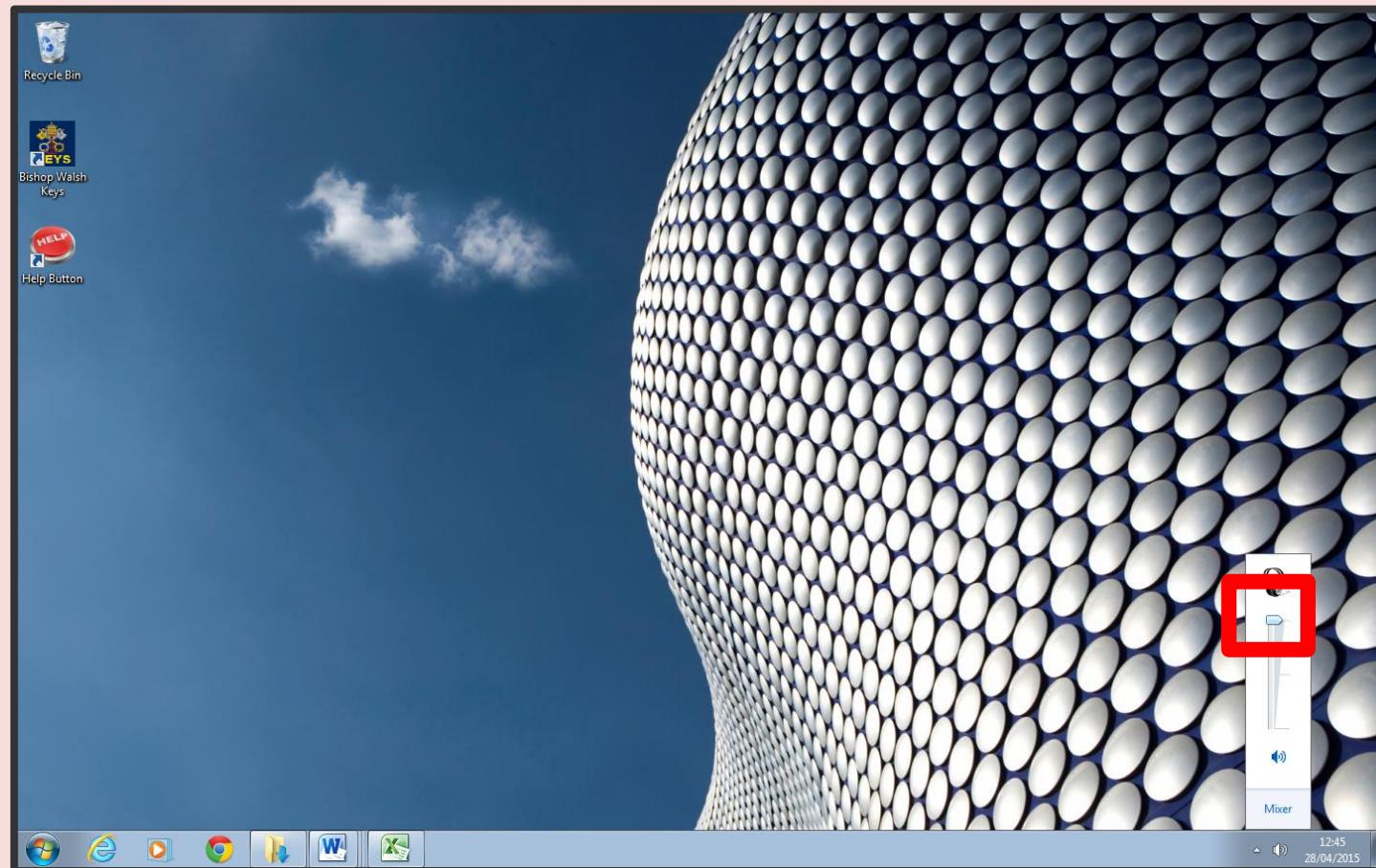
Enabling JavaScript

2. Click the radio button at the top of the alert then click the done button in the bottom right corner of the **alert**. Press the F5 key to refresh the page.



No songs playing

1. Ensure that speakers are connected to the player and the sound output is not muted.



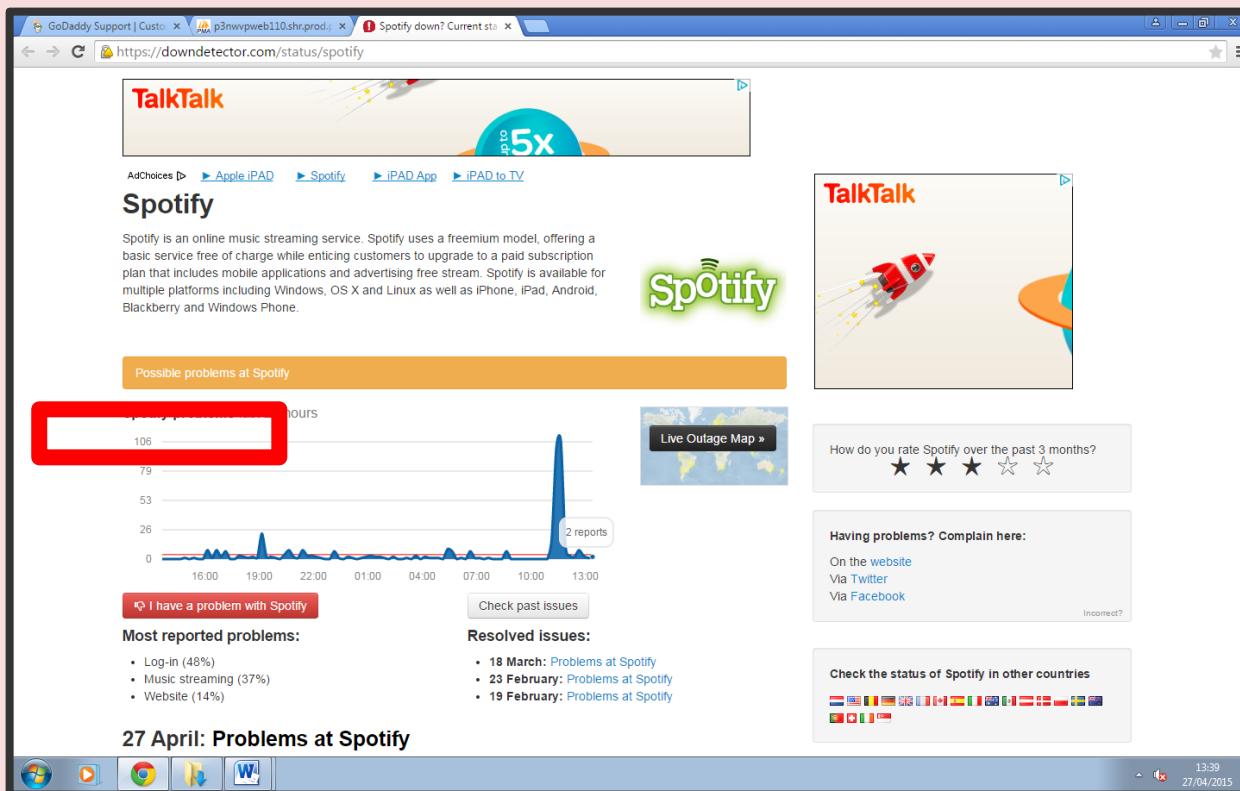
No songs playing

2. Visit www.johnjennings.net/play.html in your browser. If the Big Julie logo is not displayed then the Big Julie website must be down. Ensure that your PC is connected to the internet and call **020 7084 1810** for assistance from **GoDaddy**. You will be put in contact with a technician that will instruct you on how to get the website running again.



No songs playing

3. If the Big Julie logo is displayed then the website is working. Check that the Spotify service is running by visiting <https://downdetector.com/status/spotify>. If Spotify is down then turn Big Julie off and wait until Spotify is up and running again.



Glossary



Glossary

Backup Strategy – copying files or databases so that they will be preserved in case of equipment failure or other catastrophes.

Browser – a software application used to locate, retrieve and display web content e.g. Google Chrome and Internet Explorer.

CSV File – a text file that has a specific format which allows for the saving of textual information/data in an organized fashion.

GoDaddy – The hosting company that runs the servers and website for Big Julie.

JavaScript – a programming language commonly used to create interactive effects within web browsers. Must be installed for Big Julie to operate correctly

JuliePassGen.exe – a program provided with Big Julie that generates a text file containing random passwords,

Spotify – a music streaming service used by Big Julie to play tracks

Spotify Client – the program installed on your computer that is used by Big Julie to play Spotify songs.

System Requirements

The machine that runs the player must meet and preferably exceed these system specifications.

256 MB RAM

2GB free disk space

Windows XP with Service Pack 3

Connection to 2.1 speaker system

Spotify 2.1.1 client

On Screen Help

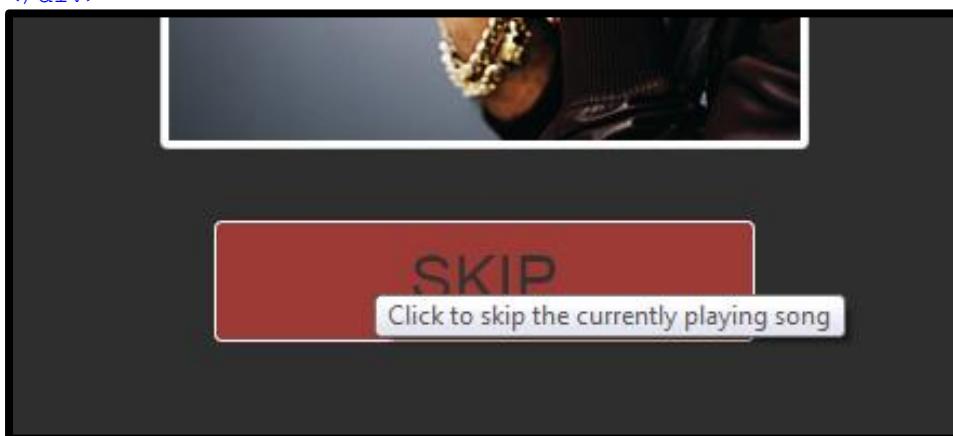
Tooltips

After my website had been created, I added title attributes to some of my HTML elements. When the user hovered over certain objects on the website, a message would be displayed that gave a simple explanation of how to interact with that aspect of the system. This would allow helpful hints to be displayed only when the user required them.

```
<div class="form-group" title="Enter the track name you wish to request here">
    <!-- create textbox with default text "Track name..." -->
    <!-- main.js gets text entered when songReq function called, calls
        songReq function when enter key pressed while in textbox -->
    <input id="tbSong" type="text" class="toSongReqDisable form-control
        toToggle searchBox" placeholder="Track name..." style="display: none;">
</div>
```



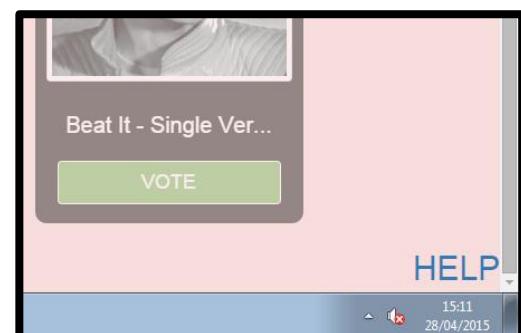
```
<div class ="btnSkip" title="Click to skip the currently playing song">
    <!-- create button with text "SKIP", not displayed until user logged in --
->
    <!-- main.js calls skip function when pressed -->
    <button id="btnSkip" type="button" class="btn btn-default navbar-btn
        toToggle locked" style="display: none;">SKIP</button>
</div>
```



Help link

I also added a link at the bottom of the page that when pressed, would take you to an electronic copy of the Student User Guide for more in depth help.

```
<div class = "help" >
    <a href="SUG.pdf">HELP</a>
</div>
```



Input Requirements

ID	Requirement	How Requirement Was Met	Evidence In	Degree of Success
i1	Must be controllable with an iPhone	Designing layout of website with specific settings for dimensions of iPhone.	C2 Beta Testing Questionnaire Results	Beta testers were asked in a questionnaire if they felt the website scaled well to their device. Users with iPhones indicated that the page adjusted correctly to the dimensions of their device and provided a dynamic and responsive interface.
i2	Users can control the system through a web interface	Users can send song, vote and skip requests from a website to a server hosted by GoDaddy. This server performs calculations based on user input to determine the behaviour of the player e.g. skipping the current song or selecting the next song to play.	C1	The website allowed multiple users to control the system at the same time with very little latency. To improve, I would change my hosting provider as GoDaddy was unreliable and the website was occasionally unavailable as a result. This would have a negative effect on the user experience and is a problem that I cannot mitigate without changing providers.
i3	Allow users to input their song requests as text	Two textboxes on the client page for track name and artist. A query to the Spotify database is created from the information input to these textboxes and the songs returned from the search query can then be added to the play queue.	C1 Process 3 (Song) / Markup 4 (Search)	Adding the ability to search for artist as well as track name allowed users to reliably find any song as long as it was available on Spotify. To improve, I would add a more robust search system with options for genre, year and album etc.
i4	Allow users to vote on songs in queue with buttons	Vote buttons in each next song module that when pressed, will add a vote for the corresponding song to the vote database.	C1 Process 4 (Vote) / Markup 6 (Next Songs)	The logic for voting worked perfectly although it was difficult to find a good value for how much a single vote should move a song forward in the queue. With more users interacting with the system, I could analyse the typical usage and find a more appropriate value or even create a function for dynamically changing the value depending on factors such as number of active users or difference in song request time.
i5	Allow users to skip unfavourable songs with buttons	Skip button in the middle of the page that when pressed, will flag the user as having voted to skip in the user database.	C1 Process 5 (Skip) / Markup 3 (Jumbotron)	The skip functionality added greatly to the user experience. It ensured that the majority of users would always enjoy the music that is playing. To improve, I would allow users to vote to skip the next songs in the queue before they begin playing.

i6	Every Sixth Form student and teacher will have a unique username and password	All Sixth Form usernames in the school's active directory were added to the user database. Passwords for each user were randomly generated by JuliePassGen.exe and then added to the database.	C1 Process 1 (Login) Visual Basic .NET	Importing the usernames from the school database allowed me to ensure that all pupils in the Sixth Form had a username. This was also good for usability since each pupil already knows their school username. The randomly generated passwords were of a good level of complexity and were fairly easy to remember. To improve, I would allow users to change their passwords through the web interface.
i7	Allow users to login with username and password	All usernames and corresponding passwords were printed out and given to Mrs Hayes. Users were instructed to get their username and password from Mrs Hayes and then enter them into textboxes on the client to login.	Student User Guide Typical Use Logging in C1 Process 1 (Login) / Markup 2 (Navbar)	The login system was reliable and accurate. In testing, no faults were found where correct credentials were rejected or incorrect credentials accepted.

Processing Requirements

ID	Requirement	How Requirement Was Met	Evidence In	Degree of Success
p1	Log user in if username and password are correct	When login button is pressed, contents of username and password textbox are sent to the server. Server validates that matching username and password exist in user database and flags the user as logged in if true.	C1 Process 1 (Login) JavaScript / PHP	The login system was reliable and accurate. In testing, no faults were found where correct credentials were rejected or incorrect credentials accepted.
p2	Allow users to send requests only when logged in	Before each request is processed, database is queried for value of loggedIn field for that user. If loggedIn is TRUE then server continues processing request, else outputs error message stating that login is required.	C1 Process 2 (Request) JavaScript / PHP	Besides the small oversight in the vote request validation that was fixed during beta testing, the server consistently processed only requests from users that are logged in.
p3	Add requested songs to the end of a queue	When user presses yes button in search results module, information for that song is sent to server as a song request. If request passes all validation then new record is created in request database with that track information. Time of request is also included in record and table is sorted by request time in ascending order.	C1 Process 3 (Song) JavaScript / PHP	Using the time that the request was sent, the queue could easily be sorted so that the earliest request is at the top and the latest at the bottom. The queue would remain in this order and change only when the votes were included, exactly as planned.
p4	Move song request forward in queue based on number of votes received.	At end of each song, player signals to server that song has ended. On receipt of that signal, server decrements request time of each track by the number of votes present in vote database for that track's URL.	C1 Process 6 (Next Song) JavaScript / PHP	The logic for voting worked perfectly although it was difficult to find a good value for how much a single vote should move a song forward in the queue. With more users interacting with the system, I could analyse the typical usage and find a more appropriate value or even create a function for dynamically changing the value depending on factors such as number of active users or difference in song request time.

p5	Skip the currently playing song if > 60% of active users request skip	<p>Every second, player signals to server to check if song has been skipped. On receipt of signal, server will evaluate if (count of records in user database where skipped field is TRUE) > (count of records in user database where active field is TRUE)*0.6. If evaluation returns TRUE then server signals back that song has been skipped. On receipt of that signal, player will end playback of current song and call functions to determine and play the next song in queue.</p>	C1 Process 6 (Next Song) JavaScript / PHP	<p>The skipping functionality worked very well. During alpha testing, the song would be skipped at almost the same instant that the skip request was sent. During beta testing, users indicated that the threshold for skipping was too low so it was changed from 40% to 60%. This change is reflected in the requirements.</p>
p6	Allow certain users to have more influence in the voting algorithm	<p>Each record in user database has a vote weight value which can be modified by an admin e.g. Mrs Hayes. When determining the new queue order in p4, server will decrement a track's request time by the vote weights of the users that sent each vote in the vote database.</p>	C1 Process 6 (Next Song) PHP	<p>The vote weight system was easy to implement and worked flawlessly. A user with a vote weight of 6 has exactly twice as much influence as a user with a vote weight of 3.</p>
p7	Filter song requests for explicit content	<p>When user searches for a song, client will receive an object from Spotify containing information of all tracks that match the search query that was sent. Client will then iterate through each track in that object, and construct a new object consisting only of tracks where the explicit value is FALSE. The tracks in the new object will then be displayed to the user in the search results module.</p>	C1 Process 3 (Song) JavaScript Search	<p>The solution I developed should in theory work without any flaws. All songs tagged as explicit by Spotify are rejected and not displayed to the user. However, Spotify does not do a very thorough job of tagging songs as explicit so many songs that I would deem as explicit can still be requested and played. This problem is intrinsic to Spotify and the most feasible solution would be to use a different service but even that would require a complete redesign.</p>

p8	Only play song requests that are under 7 minutes long	When user searches for a song, client will receive an object from Spotify containing information of all tracks that match the search query that was sent. Client will then iterate through each track in that object, and construct a new object consisting only of tracks where the track length value is less than 420000. The tracks in the new object will then be displayed to the user in the search results module.	C1 Process 3 (Song) JavaScript Search	Unlike p7, this filter worked exactly as planned. All songs over 7 minutes in length were rejected by the system and not displayed to the user. Some users expressed minor annoyance at the fact that they could not request a particular song but they were usually presented with a shorter version of that song. I feel that this is an acceptable trade-off as I think more users would be annoyed if the songs played were very long.
p9	Return to start of queue when final song ends	When player detects that song has ended, player will signal to server to move song to end of queue. On receipt of signal, server will add 1000000 to request time value of that song in request database. Request table will be sorted by request time in ascending order so order of queue will be preserved but new requests will be played immediately.	C1 Process 6 (Next Song) JavaScript / PHP	Once the code was redesigned so that there were no memory leaks, the player could be left to run and would always cycle through the queue and return to the start with very little latency.
p10	Allow one song request per user in a certain amount of time dependent on number of active users	When server receives song request from a user, server will get last request time of that user from user database. Server will then get number of users with active field TRUE and calculate a cooldown time with the formula $20000 * (\text{activeUsers})^{0.50}$. If difference between last request time and current time is greater than the cooldown time then the song request is processed, else an error message is output stating that the user is still in their cooldown period.	C1 Process 3 (Song) JavaScript / PHP	From testing, I know that the system calculated the cooldown times correctly and applied the correct logic. I feel that in theory, this functionality is essential to a positive user experience as it prevents a malicious user from spamming the system. However during beta testing, some users fed back that being locked out of the system for a certain amount of time was frustrating. I think that a good compromise would be to process every request but increase the request time value of that track by the cooldown time. Meaning that the user would never be locked out of the system but their requests would be spread throughout the queue.

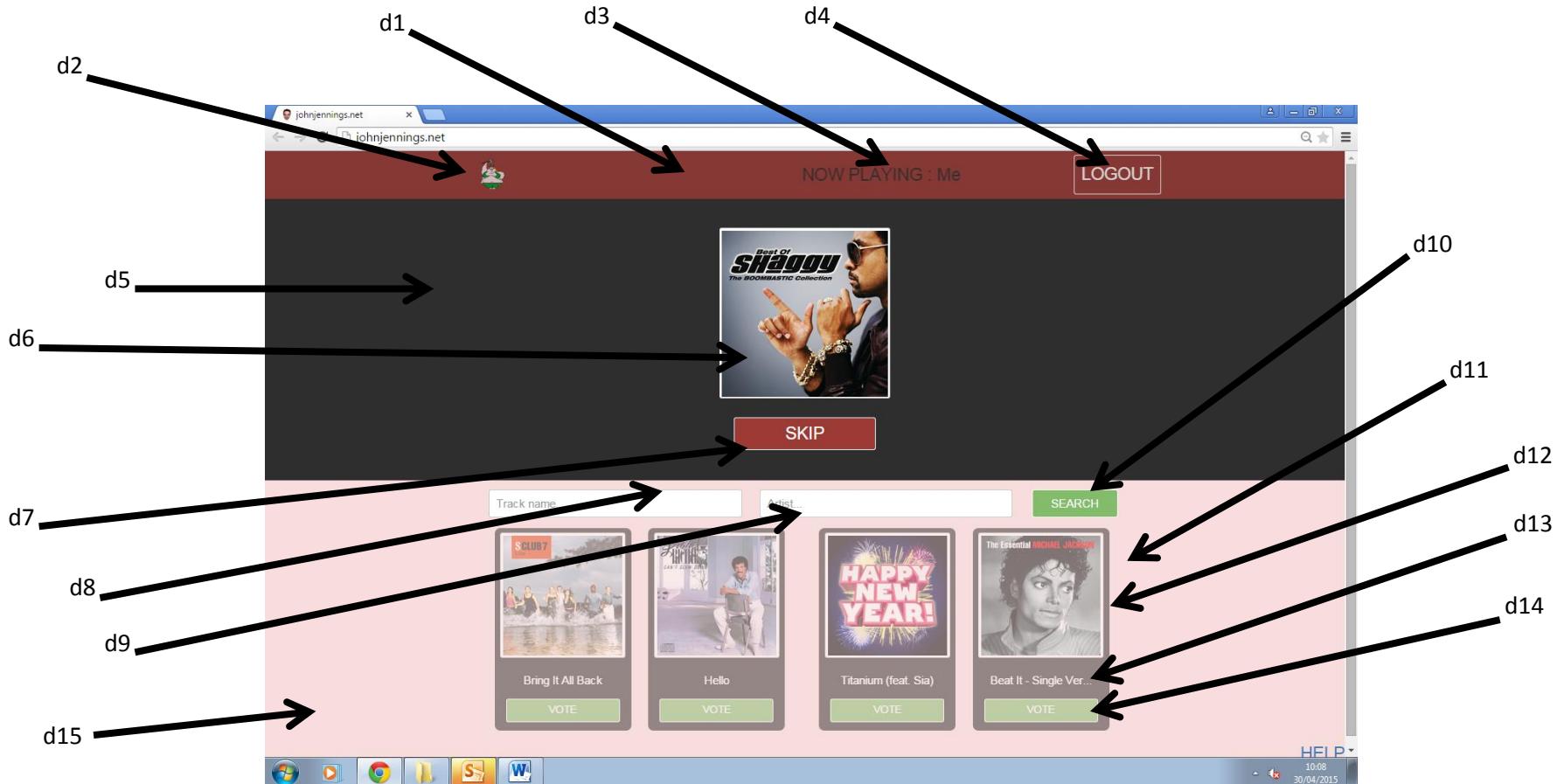
Output Requirements

ID	Requirement	How Requirement Was Met	Evidence In	Degree of Success
o1	Stream requested songs from Spotify	In request database, get URL and length of song with lowest request time. Play song in Spotify and repeat after length of song or if song is flagged as skipped.	C1 Process 6 (Next Song) JavaScript / PHP	In beta testing, the majority of users indicated that the music was of a high quality.
o2	Display currently playing track name and cover art	In request database, get track name and cover art URL of song with lowest request time. Update HTML elements with this new data.	C1 Refreshing JavaScript / PHP	In beta testing, the client updated with very little latency. The testers indicated that they felt the website was responsive and dynamic.
o3	Display track name and cover art of next songs in queue	In request database, get track name and cover art URL of songs with 2 nd -> 5 th lowest request times. Update HTML elements with this new data.	C1 Refreshing JavaScript / PHP	In beta testing, the client updated with very little latency. The testers indicated that they felt the website was responsive and dynamic.
o4	Display all outputted information without refreshing or changing page	Use AJAX to call PHP functions and update HTML asynchronously without needed to refresh page. Have all elements of system available on one HTML document to be loaded only once.	C1 Refreshing JavaScript / PHP	In beta testing, the clients updated with very little latency. The client would only have to be refreshed if an error occurred.
o5	Display all information in an appropriate format on desktop and mobile devices	Detect whether user is on desktop or mobile device and apply separate style sheets accordingly.	C1 Media Query C2 Beta Testing Questionnaire Results	In beta testing, users indicated that they felt the page scaled well to their devices. Some devices with obscure resolutions did not display as planned but were still usable.

Design Requirements

<u>ID</u>	<u>Requirement</u>	<u>Evidence In</u>	<u>Degree of Success</u>
d1	Dark red horizontal navbar at top of page containing d2,d3,d4	C1 Markup 2 (Navbar)	This element displayed as designed on most devices.
d2	Big Julie logo in first 30% of navbar	C1 Markup 2 (Navbar)	This element displayed as designed on most devices.
d3	Currently playing song name as horizontal marquee in next 40% of navbar, black font colour	C1 Markup 2 (Navbar)	This element displayed as designed on most devices.
d4	Green login button in final 30% of navbar	C1 Markup 2 (Navbar)	This element displayed as designed on most devices. I also improved the design by making the login button turn into a red logout button when the user was logged in.
d5	Dark grey jumbotron below navbar, 50% page height containing d6,d7	C1 Markup 3 (Jumbotron)	This element displayed as designed on most devices.
d6	Thumbnail of currently playing song art, 25% page height, horizontally centred	C1 Markup 3 (Jumbotron)	This element displayed as designed on most devices.
d7	Red skip button below thumbnail, font colour white, 10% page width	C1 Markup 3 (Jumbotron)	This element displayed as designed on most devices.

d8	Search module, 50% page width horizontally centred below jumbotron. Containing d9, d10.	C1 Markup 4 (Search)	This element displayed as designed on most devices.
d9	Text box for requesting song, black font colour, 40% page width	C1 Markup 4 (Search)	This element displayed as designed on most devices. Improved design by adding a text box for artist as well as track name.
d10	Green search button adjacent to search box, white font colour, 10% page width	C1 Markup 4 (Search)	This element displayed as designed on most devices.
d11	Four next song modules, 25% page width, containing d12, d13, d14, d15. Arranged in equally spaced row.	C1 Markup 6 (Next Songs)	This element displayed as designed on most devices.
d12	Thumbnail of next song to play, at top of next song module, 90% module width.	C1 Markup 6 (Next Songs)	This element displayed as designed on most devices.
d13	Name of next song to play, below thumbnail.	C1 Markup 6 (Next Songs)	This element displayed as designed on most devices. Improved design by truncating song names that are too long for module.
d14	Green vote button, below song name, first 50% of module width, font colour white.	C1 Markup 6 (Next Songs)	Because d15 was removed, vote button was changed to take up 100% of module.
d15	Red skip button, below song name, last 50% of module width, font colour white.		Skip button was removed from next song modules since a song could not be skipped if it was not already playing.
d16	Light pink page background	C1 Markup 2 (Navbar)	This element displayed as designed on most devices.



Overall Degree of Success

I feel that in general, I have met the objectives that were outlined in the design. In some situations, I extended my product past what was required in order to create a better user experience. An example of this would be when I added the search results module so that users could verify that the song returned by Spotify was correct. This addition was not necessary but I felt that it greatly increased usability.

At some stages in the development, requirements were modified to fit the changing opinions of the users, particularly during beta testing. An example of this would be modifying the skip threshold from 40% of active users to 60% because of user feedback indicating that songs were being skipped too easily. The only requirement that was not met in either the original or modified form was **d15** (Red skip button, below song name, last 50% of module width, font colour white). Due to the nature of how the skip functionality was designed, a song can only be skipped if it is currently playing. This would mean that the songs in each next song module could not be skipped and as a result, placing a skip button in each next song module would be unnecessary. The skipping functionality still works exactly as designed and agreed by the clients. I have simply removed redundant features from the layout of the web page in order to not confuse the user.

Questionnaire

In order to discern the users' opinion of the system, I distributed a questionnaire to a group of 20 pupils during a free period where the system was being used. The feedback was very positive. Users felt that the user guide, on screen help and webpage layout combined to create a comprehensive and user-friendly system. They found it easy to place their requests from their devices and felt that the system was generally easy to use.

1. I feel that the webpage is clear and easy to use

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

2. I feel that the on screen help is clear and easy to follow

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

3. I feel that it is easy for me to make requests

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

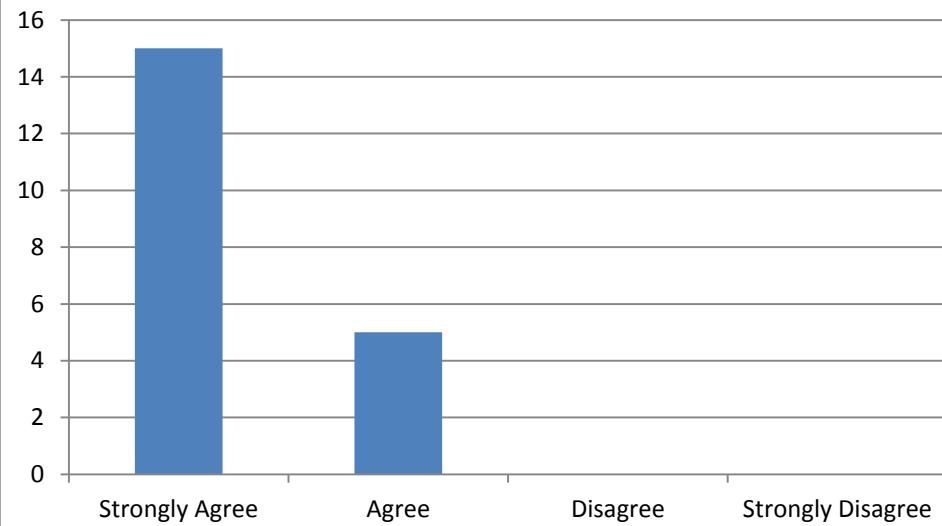
4. I feel that the user guide is comprehensive and easy to follow

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

5. I feel that the system in general is easy to use

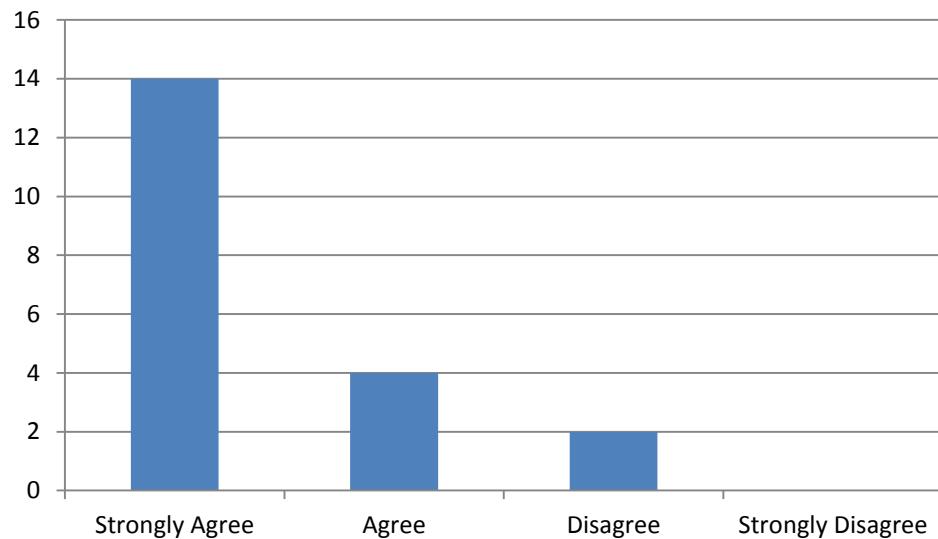
a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

I feel that the webpage is clear and easy to use



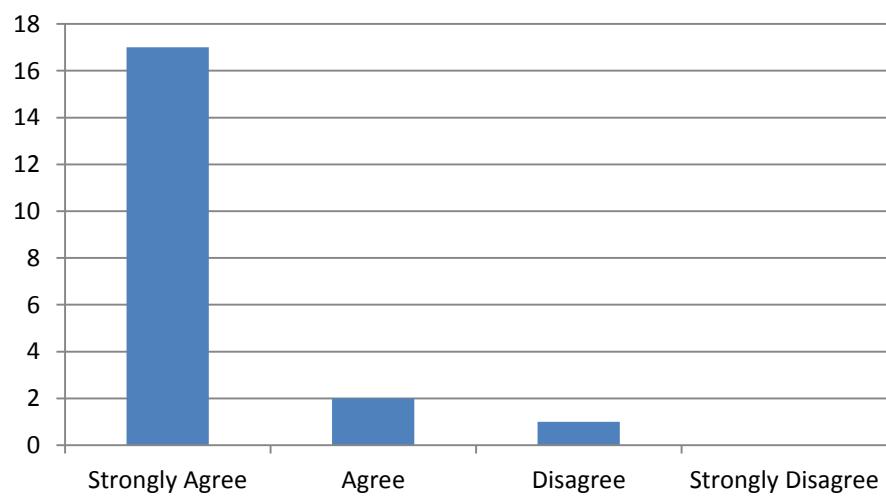
All users feel that the webpage is clear and easy to use

I feel that the on screen help is clear and easy to follow



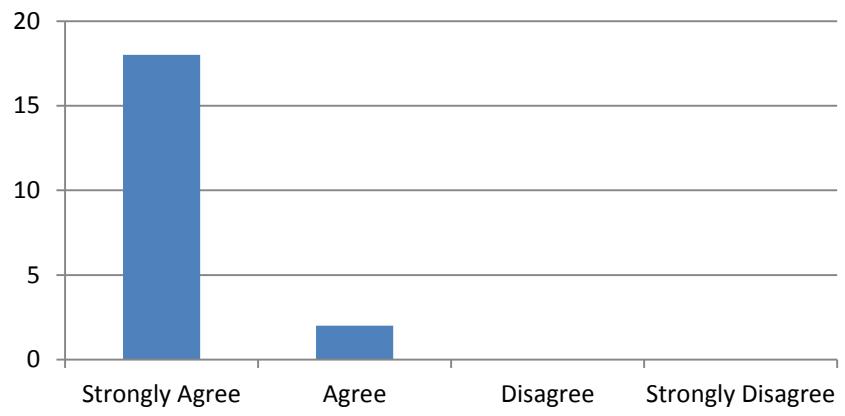
The majority of users feel that the on screen help is clear and easy to follow

I feel that it is easy for me to make requests



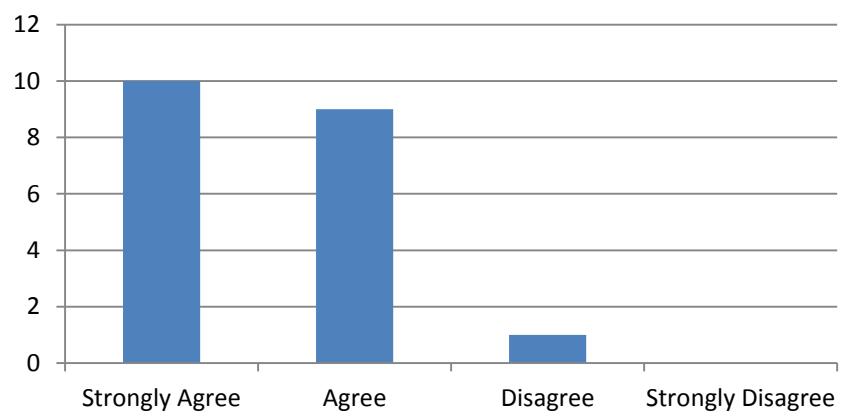
The majority of users feel that it is easy for them to make requests

I feel that the user guide is comprehensive and easy to follow



All users feel that the user guide is comprehensive and easy to follow

I feel that the system in general is easy to use



The majority of users feel that the system is generally easy to use

Handover to Mrs Hayes

Mrs Hayes will be the primary teacher that will be maintaining the system. As a result, I have created a document to gather her feedback on the handover process. It is essential that she is comfortable with all aspects of the system's operation, including setup and troubleshooting.

1. Big Julie fulfils all of the requirements that I have previously specified.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

2. Big Julie performs its required functions reliably and without fault.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

3. Big Julie is easy to use with a clear and responsive interface.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

4. The user guide has instructed me clearly on how to set up and operate Big Julie.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

5. The user guide will allow me to solve any common problems that I may face when Big Julie is in use.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

6. The service offered by Big Julie will be useful to the Sixth Form.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

7. Big Julie is overall a high quality product.

a. Strongly Agree b. Agree c. Disagree d. Strongly Disagree

Additional Comments:

Signed: _____

<u>Item</u>	<u>Received</u>
1x Big Julie Spotify account	
1x Big Julie GoDaddy account	
1x Logitech 2.1 speaker system	
2x Big Julie Teacher User Guide	
20x Big Julie Student User Guide	
1x Big Julie Website	
1x Big Julie Player	

Signed: _____

Conclusion and Extensions

The product I have created vastly exceeds any expectations that I had for it. At the start of the project, I had only ever made a simple static website and had never coded in SQL, PHP or JavaScript. I worried that the scope of the problem was far beyond my skills and that I would not even be able to create a functioning prototype, let alone a finished product. As a result, I am incredibly proud of Big Julie. I have singlehandedly developed my own multi-user streaming service that not only exists as a proof of concept, but as a polished and efficient system that is used almost every day.

The web interface updates dynamically through smooth transitions, ensuring that each client has the latest information. It is scalable and responsive with a layout that adjusts to fit devices of vastly varying dimensions. The design of the page is intuitive, utilising a simplified layout that naturally guides the user through the use of colours and animations. To alleviate the client of any complex processing, a robust and efficient backend has been developed. The server-side code is fault tolerant and modular, communicating asynchronously with clients and external systems. It manipulates a streamlined database with a logical structure that allows simple queries to make complex changes to the system's operation, greatly decreasing the amount of code required. I feel that Big Julie is a high quality product that will be of great use to the Sixth Form for many years to come.

Throughout the development, my programming skills increased greatly. I grew from a complete novice in JavaScript and PHP to a confident and able problem solver with a good understanding of the nuances of each language. With this newfound experience, there are many areas of code that in retrospect, could be made considerably more efficient and elegant. If I were to start the project again, I would decrease my use of global variables as they were very hard to trace during debugging. I would also abstract my logic and code from an object-oriented perspective as it lends itself well to the nature of my system. The interaction between the server, client, player and requests would be much easier to develop when object oriented since this is way in which I naturally think about the system working.

There were also many features that if given more time, I would wish to implement or improve upon.

For song requests, I would like to add a more comprehensive search function so that users can be more specific in their queries to Spotify. It would be fairly trivial to add input forms for parameters such as genre, album name or year of release. The extended search functionality could even be hidden with jQuery until the user presses a button, thus retaining the simplicity of the design.

Currently, users can only vote for the next four songs in the queue and skip the currently playing song. A modification to the refresh function and the next song modules would cause the client to offset the position in the queue that the modules are updated from. Adding buttons for 'left' and 'right' would allow users to cycle through the entire queue and vote for any song that has been requested. The skip functionality could also be redesigned so that it is treated more similarly to a vote request. Although a fairly large amount of code would have to be rewritten, this would allow users to vote to remove any song from the queue before it is played. This would increase the quality of music that does get played and as a result, improve the user experience.

Beyond improving on current features, I wish to extend Big Julie so that multiple 'rooms' can be created with separate queues. The system could then not only be used in the Sixth Form Centre, but separate instances of Big Julie could run concurrently in other rooms of the school or even in people's homes. I could also integrate the player into the client so that users could connect to different 'rooms' and listen on their headphones to what is essentially a user generated radio station.

```
<!DOCTYPE html>
<html>
<!-- load external resources -->
<head>
    <!-- load twitter bootstrap css template, used to create dynamic components e.g. navbar, jumbotron etc -->
    <link rel="stylesheet" href="dist/css/bootstrap.css">
    <!-- load css style for page -->
    <link rel="stylesheet" href="main.css">
    <!-- load jQuery library, used to add functionality to and fix bugs in vanilla javascript e.g. animation and DOM object selection -->
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
    <!-- load javascript for page -->
    <script src="main.js"></script>
</head>

<body id="page">
    <!-- create navigation bar at top of screen, bootstrap.css handles basic layout -->
    <nav class="navbar navbar-default navbar-fixed-top" role="navigation">
        <div class="container">
            <!-- create bootstrap row with width 12, used to arrange elements into dynamic columns -->
            <div class="row">
                <!-- create column with width 3, xs ensures that display is consistent on all devices -->
                <div class="col-xs-3">
                    <!-- display Big Julie logo in column -->
                    <div class="BJ">
                        
                    </div>
                </div>
                <!-- create column with width 6 -->
                <div class="col-xs-6">
                    <!-- display marquee moving right to left with currently playing song name -->
                    <!-- main.js updates text on page load and refresh function called -->
                    <p id="nowPlaying" class="nowPlaying toRefresh"><marquee>
</marquee></p>
                </div>
                <!-- create column with width 2 to contain login textboxes -->
                <div class="col-xs-2">
                    <div class="login form-group toToggle">
                        <div class="username">
                            <!-- create textbox with default text "Username..." -->
                            <!-- main.js gets text entered when login function called, fades textbox out if login successful, calls login function when enter key pressed while in textbox -->
                            <input id="username" type="text" class="form-control " placeholder="Username..."/>
                        </div>
                        <div class="password">
                            <!-- create textbox with default text "Password..." -->
```

```
<!-- main.js performs same logic as to username -->
<input id="password" type="password" class="form-control"
placeholder="Password...">
    </div>
</div>
</div>
<!-- create column with width 6 -->
<div class="col-xs-1">
    <div class="btnLogin">
        <!-- create button with text "LOGIN" -->
        <!-- main.js calls login function when pressed and adds class
"logged", changes text to LOGOUT if login successful, main.css changes display
accordingly-->
        <button id="btnLogin" type="button" class="btn btn-default
navbar-btn">LOGIN</button>
    </div>
</div>
</div>
</div>
</div>
</nav>
<!-- create jumbotron in centre of page, bootstrap.css handles basic layout -->
<div class="jumbotron">
    <div class="container">
        <!-- create thumbnail in centre of jumbotron, bootstrap.css handles basic
layout -->
        <div class="thumbnail">
            <!-- display cover art of currently playing song -->
            <!-- main.js updates image on page load and refresh function called -->
            
        </div>
        <div class="btnSkip">
            <!-- create button with text "SKIP", not displayed until user logged in
-->
            <!-- main.js calls skip function when pressed -->
            <button id="btnSkip" type="button" class="btn btn-default navbar-btn
toToggle locked" style="display: none;">SKIP</button>
        </div>
    </div>
</div>
<!-- contains all elements related to searching and requesting songs -->
<div class="search">
    <div class="container">
        <!-- all contained elements not displayed until user logged in successfully
-->
        <!-- main.js temporarily disables elements when song successfully requested
-->
        <div class="initSearch">
            <div class="row-fluid">
                <div class="col-sm-5">
                    <div class="form-group">
                        <!-- create textbox with default text "Track name..." -->
                        <!-- main.js gets text entered when songReq function called, calls
songReq function when enter key pressed while in textbox -->
                        <input id="tbSong" type="text" class="toSongReqDisable
form-control toToggle searchBox" placeholder="Track name..." style="display:
```

```
none;">>
    </div>
</div>
<div class="col-sm-5">
    <div class="form-group">
        <!-- create textbox with default text "Artist..." -->
        <!-- main.js performs same logic as to tbSong -->
        <input id="tbArtist" type="text" class="toSongReqDisable
form-control toToggle searchBox" placeholder="Artist..." style="display: none;">
    </div>
</div>
<div class="col-sm-2">
    <!-- create button with text "SEARCH" -->
    <!-- main.js calls songReq function when pressed -->
    <button id="btnSearch" type="button" class="toSongReqDisable
btnSearch btn btn-default navbar-btn toToggle" style="display:
none;">SEARCH</button>
</div>
</div>
</div>
<!-- all contained elements not displayed until song request successfully
made -->
<!-- main.js animates fade in of contained elements and moves next song
modules to fit when searchOpen function called -->
<div class="searchResult" >
    <div class="searchToggle toSearchToggle" style="display: none;">
        <div class="row-fluid">
            <!-- create column with width 10 to contain search result information
-->
            <div class="col-sm-10">
                <div class="row-fluid">
                    <div class="col-sm-4">
                        <!-- create thumbnail to left of page, bootstrap.css handles
basic layout -->
                        <div class="thumbnail toUpdate">
                            <!-- display cover art of search result song -->
                            <!-- main.js updates image when new search query made -->
                            
                        </div>
                    </div>
                    <!-- contains information on search result song -->
                    <!-- main.js updates text when new search query made -->
                    <div class="col-sm-8">
                        <div class="trackInfo ">
                            <p id="titleResult" class="toUpdate">Title</p>
                            <p id="artistResult" class="toUpdate">Artist</p>
                            <!-- elements are never displayed, used for song validation
process -->
                            <p id="lengthResult" style="display: none;">Length</p>
                            <p id="idResult" style="display: none;">Explicit</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
<!-- contains buttons for selecting whether search result is correct
```

```
song -->
    <!-- main.js disables buttons temporarily when song request
successfully made -->
    <div class="col-sm-2">
        <div class="reqConfBtns">
            <!-- main.js calls songReq function when pressed, closes search
result module -->
            <button id="btnYes" type="button" class="toSongReqDisable btn
btn-default navbar-btn reqConfBtn btnYes" disabled>YES</button>
            <!-- main.js calls search function again, gets information of next
search result and updates accordingly -->
            <button id="btnNo" type="button" class="toSongReqDisable btn
btn-default navbar-btn reqConfBtn btnNo" disabled>NO</button>
        </div>
    </div>
</div>
<!-- contains all elements related to upcoming songs -->
<div class="nextSongs">
    <div class="container">
        <div class="row-fluid">
            <!-- md column causes modules to display horizontally on desktop, sm column
creates grid layout on mobile -->
            <div class="col-sm-6">
                <div class="col-md-6">
                    <!-- contains all elements for first next song module -->
                    <div class="nextSong">
                        <!-- create thumbnail at top of module, bootstrap.css handles basic
layout -->
                        <div class="thumbnail">
                            <!-- display cover art of next song -->
                            <!-- main.js updates image when getNextSongs function called if
next song has changed -->
                            
                        </div>
                        <!-- display title of next song -->
                        <!-- main.js updates text when getNextSong function called if next
song has changed, if text will not fit in module, text is truncated to 20 characters
and ellipses appended -->
                        <p id="pNext1" style="white-space: nowrap; text-overflow:
ellipsis;" class="pNext toRefresh"> - </p>
                        <span class="btnVote">
                            <!-- create button with text "VOTE" -->
                            <!-- main.js calls vote function when pressed-->
                            <button id="btnVote1" type="button" class="voteBtn locked btn
btn-default navbar-btn toToggle" style="display: none;">VOTE</button>
                        </span>
                    </div>
                </div>
            <!-- contains all elements for second next song module -->
            <div class="col-md-6">
                <div class="nextSong">
```

```
<div class="thumbnail">
    
</div>
<p id="pNext2" style="white-space: nowrap; text-overflow: ellipsis;" class="pNext toRefresh"> - </p>
    <span class="btnVote">
        <button id="btnVote2" type="button" class="voteBtn locked btn btn-default navbar-btn toToggle" style="display: none;">VOTE</button>
    </span>
</div>
</div>
<div class="col-sm-6">
    <div class="col-md-6">
        <!-- contains all elements for third next song module -->
        <div class="nextSong">
            <div class="thumbnail">
                
            </div>
            <p id="pNext3" style="white-space: nowrap; text-overflow: ellipsis;" class="pNext toRefresh"> - </p>
            <span class="btnVote">
                <button id="btnVote3" type="button" class="voteBtn locked btn btn-default navbar-btn toToggle" style="display: none;">VOTE</button>
            </span>
        </div>
        <!-- contains all elements for fourth next song module -->
        <div class="col-md-6">
            <div class="nextSong">
                <div class="thumbnail">
                    
                </div>
                <p id="pNext4" style="white-space: nowrap; text-overflow: ellipsis;" class="pNext toRefresh"> - </p>
                <span class="btnVote">
                    <button id="btnVote4" type="button" class="voteBtn locked btn btn-default navbar-btn toToggle" style="display: none;">VOTE</button>
                </span>
            </div>
        </div>
    </div>
</div>
</body>

</html>
```

```
/* set style of desktop page, will be overridden for mobile where applicable */
/* for entire page */
body{
    /* set background colour to light pink */
    background-color: #F7DDEC;
}
/* for every button */
button{
    /* set border colour to always be white */
    border-color: #FFF !important;
}
/* for entire navbar */
.navbar{
    /* set background colour to dark red */
    background: #9C3B35;
    /* match bottom border with jumbotron background */
    border-bottom-color: #2E2E2E;
    /* set navbar to be slightly transparent */
    opacity: 0.8;
    /* size adjustments for correct layout on desktop */
    padding-top: 5px;
    height: 85px;
}
/* for div containing Big Julie logo */
.BJ {
    /* size adjustments for correct layout on desktop */
    margin-top:-20px;
    font-size: 70px;
}
/* for Big Julie logo */
.BJ img{
    /* set consistent size to fit in navbar */
    height: 70px;
}
/* for marquee displaying currently playing song title */
.nowPlaying {
    /* position in middle of navbar */
    text-align: center;
    /* set text colour to dark grey */
    color: #2E2E2E;
    /* size adjustments for correct layout on desktop */
    font-size:30px;
    height: 70px;
    line-height: 75px;
}
/* for div containing login button */
.btnLogin {
    /* align button to far right of navbar, vertically centre */
    padding-top: 1px;
    text-align: right;
}
/* for login button */
.btnLogin button{
    /* set text colour to white, background colour to green */
    color: #FFF;
```

```
background: #83BD6C ;
/* size adjustments for correct layout on desktop */
margin-top:0px;
font-size: 30px;
height: 70px;
}

/* for login button when logged in */
.logged {
    /* override background to match navbar background */
background: #9C3B35 !important;
/* size adjustments for longer text */
margin-left: -30px;
}

/* for password textbox */
.password{
    /* size adjustments for correct layout on desktop */
padding-top: 5px;
}

/* for entire jumbotron */
.jumbotron {
    /* set background to dark grey */
background: #2E2E2E;
/* align elements in centre */
text-align: center;
/* size adjustments for correct layout on desktop */
padding-top: 135px;
height: 508px;
margin-bottom: 15px;
}

/* for thumbnail containing currently playing song cover art */
.jumbotron .thumbnail {
    /* size adjustments for correct layout on desktop */
display: inline-block;
height:300px;
width:300px;
padding:3px;
}

/* for currently playing song cover art */
.thumbNow{
    /* size adjustments for correct layout on desktop */
height:292px !important;
width:300px;
}

/* for skip button */
.btnSkip button {
    /* set text to white, background to match navbar background */
background-color: #9C3B35;
color: #FFF;
/* size adjustments for correct layout on desktop */
font-size:30px;
width: 250px;
}

/* for skip button when skipped */
.skipped {
    /* override background colour to grey */
```

```
background-color: #BDBDBD !important;
}
/* for div containing all search elements */
.search .container{
    /* size adjustments for correct layout on desktop */
    text-align: center;
    padding-bottom: 10px;
}
/* for track and artist textboxes */
.search input{
    /* size adjustments for correct layout on desktop */
    font-size:20px;
    height:50px;
}
/* for search button */
.btnSearch{
    /* set text to white, background to green */
    color: #FFF;
    background: #83BD6C ;
    /* size adjustments for correct layout on desktop */
    font-size:20px;
    height:50px;
    width:150px;
    margin-top:0px;
}
/* for div containing all search result elements */
.searchToggle {
    /* set background to match jumbotron background */
    background: #2E2E2E;
    /* round corners */
    border-radius: 10px;
    /* set elements to be slightly transparent*/
    opacity: 0.8;
    /* size adjustments for correct layout on desktop */
    margin-bottom: 15px !important;
    margin-top: 65px;
    height:330px;
}
/* when search results hovered over */
.searchToggle:hover{
    /* disable transparency */
    opacity:1;
}
/* for thumbnail containing search result album art */
.searchResult .thumbnail {
    /* ensure that thumbnail is constant size */
    height: 300px !important;
    width: 300px !important;
    /* size adjustments for correct layout on desktop */
    margin-left:-5px;
    margin-top:15px;
    margin-bottom:10px;
}
/* for search result album art */
.thumbResult{
```

```
/* ensure that image fits in thumbnail */
height: 290px !important;
width: 300px !important;
}
/* for div containing search result information */
.trackInfo{
    /* size adjustments for correct layout on desktop */
    margin-top: 25px;
    margin-left: 60px;
}
/* for text displaying search result information*/
.trackInfo p{
    /* set colour to white */
    color: #fff;
    /* size adjustments for correct layout on desktop */
    font-size: 20px;
    text-align: left;
}
/* for div containing YES and NO buttons */
.reqConfBtns{
    /* size adjustments for correct layout on desktop */
    margin-top:10px;
}
/* for YES and NO buttons */
.reqConfBtn{
    /* set text to white */
    color: #FFF;
    /* size adjustments for correct layout on desktop */
    font-size:60px;
    height:150px;
    width:150px;
    margin-top:0px;
}
/* for YES button */
.btnYes{
    /* set colour to green */
    background: #83BD6C ;
}
/* for NO button */
.btnNo{
    /* set colour to red */
    background: #9C3B35;
}
/* for div containing next song modules */
.nextSongs{
    /* ensure that mouse hover targets correct area */
    pointer-events: none;
    /* size adjustments for correct layout on desktop */
    margin-left: -10px;
}
/* for each next song module */
.nextSong {
    /* align elements in centre of module */
    text-align: center;
    /* set background to dark grey */
```

```
background: #2E2E2E;
/* round corners */
border-radius: 10px;
/* set module to 50% transparency */
opacity: 0.5;
/* ensure that mouse hover targets correct area */
pointer-events: fill;
/* ensure that modules align correctly when search result module is open */
position: relative !important;
/* size adjustments for correct layout on desktop */
padding: 5px;
height: 300px;
margin-top: -10px;
}

/* when mouse hovers over specific module */
.nextSong:hover {
/* disable transparency */
opacity: 1;
}
/* for each thumbnail containing next song cover art */
.nextSong .thumbnail {
/* ensure that thumbnail is constant size */
height: 220px !important;
width: 220px !important;
/* size adjustments for correct layout on desktop */
display: inline-block;
margin-top: 3px;
}
/* for each next song cover art */
.nextSong img {
/* ensure that image fits in thumbnail */
height: 210px !important;
width: 220px !important;
}
/* for each next song title */
.nextSong p {
/* set colour to white */
color: #FFF;
/* size adjustments for correct layout on desktop */
font-size: 20px;
}
/* for each vote button */
.btnVote button {
/* set text colour to white, background to green */
color: #FFF;
background: #83BD6C;
/* size adjustments for correct layout on desktop */
width: 100px;
font-size: 18px;
width: 200px;
}
/* for vote button when corresponding song already voted for */
.voted {
/* override background colour to grey */
background-color: #BDBDBD !important;
```

```
}

/* for each next song module on desktop */
.col-md-6 {
    /* size adjustments for correct layout on desktop */
    padding-bottom: 20px;
}
/* apply following CSS only when media query met, i.e. when page loaded from mobile device */
@media="only screen and (max-device-width: 480px) , only screen and (-webkit-min-device-pixel-ratio: 2) , screen and (-webkit-device-pixel-ratio:1.5){
    /* override previous layout for correct display on mobile */
.navbar{
    padding-top: 10px;
    height: 95px;
}
.BJ {
    padding-top: 25px;
    font-size: 35px;
}
.nowPlaying {
    font-size:30px;
    height: 80px;
    line-height: 75px;
}
.btnLogin {
    margin-left:-10px;
}
.btnLogin button{
    font-size: 30px;
    width: 150px;
}
.jumbotron {
    padding-top: 135px;
}
.jumbotron .thumbnail {
    width: 300px;
    padding:3px;
}
.btnSkip button{
    font-size: 30px;
    width: 250px;
}
.search .container{
    padding-bottom: 15px;
}
.initSearch{
    margin-left: -50px;
}
.search input{
    font-size:20px;
    height:50px;
}
.searchResult {
    margin-left:-100px;
}
```

```
    margin-right:-100px;
}
.search .thumbnail{
  padding:3px;
}
.trackInfo{
  margin-top: 25px;
  margin-left: 100px;
}
.reqConfBtns{
  margin-left: -40px;
}
.nextSongs{
  margin-left:-5px;
}
.nextSong {
  padding:5px;
  border-radius: 10px;
  height: 370px;
}
.nextSong .thumbnail {
  height: 280px !important;
  width: 280px !important;
}
.nextSong img {
  height: 270px !important;
  width: 280px !important;
}
.nextSong p {
  font-size: 20px;
}
.btnVote button{
  font-size: 30px;
  width: 250px;
}
.col-md-6 {
  padding-bottom: 20px;
}
.logged {
  background: #9C3B35 !important;
  margin-left: 0px;
}
```

```
function init(){
    // ensure that first search result will be displayed
    localStorage.setItem("searchResCount",0);
    // if user is logged in on page load
    if (localStorage.getItem("loggedIn") == "TRUE"){
        // update page to logged in mode
        toggleLogged();
    }
    else{
        localStorage.setItem("loggedIn","FALSE")
    }
    // for each vote button
    $('.btnVote').each(function(currentElement, index) {
        // assign button with offset index
        $(this).data( "offset", currentElement );
    });

    // use difference in height from CSS media query to determine whether user is
    on mobile or desktop
    if ($.nextSong").height() == 360){
        localStorage.setItem("mobile", "TRUE");
    }
    else {
        localStorage.setItem("mobile", "FALSE");
    }
    refresh();
};

function refresh(){
    getNowPlaying();
    getNextSongs();
}

function getNowPlaying(){
    // get album cover of currently playing song
    $.ajax(
    {
        type: "POST",
        url: "init.php",
        dataType: "text",
        data: {action:"getNextSongCover", PHPindex:0},
        success: function (cover) {
            var thumb = $('#thumbNow');
            // if thumbnail has changed
            if (($.thumb).prop('src')) != $.trim(cover)){
                // enable skip button if disabled
                if ($("#btnSkip").prop('disabled') == true ){
                    $("#btnSkip").toggleClass("skipped");
                    $("#btnSkip").prop('disabled', function(i, v) { return !v; });
                }
                // enable vote buttons if disabled
                $('.voteBtn').each(function(currentElement, index) {
                    if ($(this).prop('disabled') == true ){
                        $(this).toggleClass("voted");
                        $(this).prop('disabled', function(i, v) { return !v; })
                    }
                });
            }
        }
    });
}
```

```
        }
    });
    // get name of currently playing song
    $.ajax(
    {
        type: "POST",
        url: "init.php",
        dataType: "text",
        data: {action:"getNextSongName", PHPindex:0},
        success: function (name) {
            // update marquee with new name
            $('#nowPlaying').fadeToggle("slow",function(){
                $('#nowPlaying').html('<marquee>NOW PLAYING : '+name+'</marquee>');
            });
            $('#nowPlaying').fadeToggle("slow");
        }
    });
}

function getNextSongs(){
// for each thumbnail with class 'thumbNext'
$('.thumbNext').each(function(currentElement, index) {
    // get nth album cover from request database
    $.ajax(
    {
        type: "POST",
        url: "init.php",
        dataType: "text",
        data: {action:"getNextSongCover", PHPindex:(currentElement+1)},
        success: function (cover) {
            // set thumbnail to new cover
            var thumb = $('#thumbNext'+String(currentElement+1));
            setThumb(thumb,$.trim(cover));
        }
    });
});
// for each paragraph with class 'pNext'
$('.pNext').each(function(currentElement, index) {
    // get nth name from request database
    $.ajax(
    {
        type: "POST",
        url: "init.php",
        dataType: "text",
        data: {action:"getNextSongName", PHPindex:(currentElement+1)},
        success: function (name) {
            name = $.trim(name);
            // if track name will overflow module
            if (name.length > 20) {
```

```
// set text to first twenty characters and append ellipses
name = (name.substr(0,20) + '...');

}

// update page with new song name
var para = $('#pNext' + String(currentElement+1));
setName(para, name);

}

});

});

}

function setThumb(thumb, art){
if ($(thumb).prop('src') != art){
$(thumb).fadeToggle("slow", function(){
$(thumb).attr("src", art);
});
$(thumb).fadeToggle("slow");
}
}

function setName(para, name){
if ($(para).text() != name){
$(para).fadeToggle("slow", function(){
$(para).html(name);
});
$(para).fadeToggle("slow");
}
}

function toggleLogged(){
// if search box is open then close
if ($('#btnYes').prop('disabled') == false){
searchClose();
}
// toggle vote buttons and search box visible
if ($.nextSong").height() == 360 || $.nextSong").height() == 290{
$('.toToggle").delay(200).fadeToggle("slow");
}
else{
$('.toToggle").delay(0).fadeToggle("slow");
}
// animate resize for next song module
if (localStorage.getItem("mobile") == 'TRUE'){
// expand module to fit vote button
if ($.nextSong").height() == 360{
$('.nextSong').animate({height:450},500);
}
// retract module to initial height
else{
$('.nextSong').animate({height:370},500);
}
}
// if user on desktop
else{
}

}

// if search box is open then close
function searchClose(){
$( '#searchForm' ).slideToggle( "slow" );
}

```

```
if ($.nextSong").height() == 290){
    $('.nextSong').animate({height:355},500);
}
else{
    $('.nextSong').animate({height:300},500);
}
// animate resize for jumbotron and search box
// expand to fit search box and skip button
if ('.jumbotron').height() == 325{
    $('.jumbotron').animate({height:579},500);
    $('.search .container').animate({height:79},500);
}
// retract to initial height
else{
    $('.jumbotron').delay(100).animate({height:508},500);
    $('.search .container').delay(100).animate({height:25},500);
}
// toggle button between login and logout
$("#btnLogin").fadeToggle("slow",function(){
    // toggle button text and CSS
    if ($("#btnLogin").text() == "LOGIN") {
        $("#btnLogin").text("LOGOUT");
    }
    else {
        $("#btnLogin").text("LOGIN");
    };
    $("#btnLogin").toggleClass("logged");
});
$("#btnLogin").fadeToggle("slow");
};

function login(){
    // get login values from textboxes
    var username = $('#username').val();
    var password = $('#password').val();
    // if user not logged in
    if (localStorage.getItem("loggedIn") == 'FALSE') {
        // send login request to login.php with username and password
        $.ajax(
        {
            type: "POST",
            url: "login.php",
            dataType: "text",
            data: {action:"login", PHPusername: username, PHPpassword: password},
            // return if login was successful
            success: function (loginValid) {
                // if login successful
                if (loginValid == 1) {
                    // update page to logged in mode
                    toggleLogged();
                    // set user as logged in
                    localStorage.setItem("loggedIn", "TRUE");
                    // save username in localstorage for use in requests
                    localStorage.setItem("username", username);
                }
            }
        });
    }
}
```

```
        }
        // if login unsuccessful
    else {
        // output login failed message
        alert("Login unsuccessful, check username and password are correct");
    }
}
// if user logged in
else {
    // send logout request to login.php with username
    $.ajax(
    {
        type: "POST",
        url: "login.php",
        dataType: "text",
        data: {action:"logout", PHPusername: username},
    });
    // update page to logged out mode
    toggleLogged();
    // set user as logged out
    localStorage.setItem("loggedIn", "FALSE");
}
}

function songSearch(){
    // get search result index
    var i = (localStorage.getItem("searchResCount"));
    // get track and artist to search, convert string to valid format for spotify
    var searchTrack = ($("#tbSong").val()).replace(/ /g,"%20");
    var searchArtist = ($("#tbArtist").val()).replace(/ /g,"%20");
    // construct query depending on which data was sent by user
    var query = "https://api.spotify.com/v1/search?q=";
    // if track and artist specified
    if (searchTrack != "" && searchArtist != ""){
        // construct request with track and artist
        query = query + "track:" + searchTrack + "+artist:" + searchArtist;
    }
    // if only track specified
    else if (searchTrack != ""){
        // construct request with only track
        query = query + "track:" + searchTrack;
    }
    // if only artist specified
    else if (searchArtist != ""){
        // construct request with only artist
        query = query + "artist:" + searchArtist;
    }
    query = query + "&type=track&market=GB";
    // query spotify with constructed string
    $.ajax(
    {
        type: "GET",
        url: query,
```

```
success: function (trackObject) {
    var validTracks = [];
    // get array of valid indices for trackObject
    validTracks = validateTrack(trackObject);
    var j = validTracks[i];
    $(".toUpdate").fadeToggle("1200",function(){
        // if object with index j present in trackObject
        if (typeof(trackObject.tracks.items[j]) != "undefined"){
            // assign web elements with corresponding values from trackObject

    $('#thumbResult').attr("src",trackObject.tracks.items[j].album.images[0].url)
    ;
    $('#titleResult').html(trackObject.tracks.items[j].name);

    $('#artistResult').html(trackObject.tracks.items[j].artists[0].name);

    $('#lengthResult').html(parseInt(trackObject.tracks.items[j].duration_ms));
        $('#idResult').html(trackObject.tracks.items[j].id);
    }
    });
    // update display on page
    $(".toUpdate").fadeToggle("1200");
}
});

function validateTrack(trackObject){
    var returnTracks = [];
    var i = 0;
    // iterate through each item in trackObject
    while (typeof(trackObject.tracks.items[i]) != "undefined") {
        // if song is under 7 minutes long and not explicit
        if (((trackObject.tracks.items[i].duration_ms) < 420000) &&
((trackObject.tracks.items[i].explicit) == false)){
            // push corresponding index to returnTracks
            returnTracks.push(i);
        }
        i = i + 1;
    }
    // return array of indices for valid songs in trackObject
    return returnTracks;
}

function searchOpen(){
    // if search results not already displayed
    if ($('#btnYes').prop('disabled') == true){
        // animate opening of search results
        $(".reqConfBtn").prop('disabled', function(i, v) { return !v; });
        $('.nextSong').animate({top:345},500);
        $(".toSearchToggle").fadeToggle("slow");
    }
}

function searchClose(){
    // empty search boxes
```

```
$('#tbSong').val("");
$('#tbArtist').val("");
// animate closing of search results
$(".reqConfBtn").prop('disabled', function(i, v) { return !v; });
    $('.nextSong').animate({top:0},500);
    $(".toSearchToggle").fadeToggle("slow");
};

function songReq(){
    // get request data from html
    var username = (localStorage.getItem("username"));
    var currentTime = getCurrentTime();
    var id = $('#idResult').text();
    // send request data to songReq.php for validation
    $.ajax(
        {
            type: "POST",
            url: "songReq.php",
            dataType: "text",
            data: {action:"validateSongReq", PHPusername: username, PHPcurrentTime:
currentTime, PHPURL: id},
            success: function (songReqValid) {
                // if request has passed all validation
                if ($.trim(songReqValid).charAt(0) == "p"){
                    // get cooldown time from returned data
                    var cooldown = parseInt($.trim(songReqValid).substring(1));
                    // disable song request button
                    toggleSongReqDisabled();
                    // enable song request button after cooldown
                    setTimeout(toggleSongReqDisabled,cooldown);
                    // send song request
                    pushSongReq(id,username);
                }
                // if request has failed validation
                else {
                    // display error message
                    alert(songReqValid);
                }
            }
        });
}

function getCurrentTime(){
    // return current UNIX timestamp in milliseconds
    return (Math.floor(Date.now()));
}

function toggleSongReqDisabled(){
    $('.toSongReqDisable').each(function(currentElement, index) {
        this.prop('disabled', function(i, v) { return !v; });
    });
}

function pushSongReq(id,username){
```

```
// get request data from html
var name = $('#titleResult').text();
var artist = $('#artistResult').text();
var art = $('#thumbResult').attr("src");
var length = parseInt($('#lengthResult').text());
var reqTime = getCurrentTime();
var skipCount = 0;
// send track information to songReq.php to add song to queue
$.ajax(
{
    type: "POST",
    url: "songReq.php",
    dataType: "text",
    data: {action:"pushSongReq", PHPURL: id, PHPusername: username, PHPname: name, PHPArtist: artist, PHPArt: art, PHPlength: length, PHPreqTime: reqTime, PHPskipCount: skipCount},
    success: function () {

    }
});

function skipReq(){
    var username = (localStorage.getItem("username"));
    $.ajax(
    {
        type: "POST",
        url: "skip.php",
        dataType: "text",
        data: {action:"skipReq", PHPusername:username},
        success: function (skipped) {
            // if skip request not made
            if ($.trim(skipped) != "1"){
                // output error message
                alert(skipped);
            }
            // if skip request success
            else{
                // change skip button colour to grey
                $("#btnSkip").toggleClass("skipped");
                // disable skip button
                $("#btnSkip").prop('disabled', function(i, v) { return !v; });
                setTimeout(refresh,1000);
            }
        }
    });
}

function voteReq(offset){
var username = (localStorage.getItem("username"));
$.ajax(
{
    type: "POST",
    url: "vote.php",
```

```
dataType: "text",
data: {action:"voteReq",PHPusername:username,PHPoffset:offset},
success: function (voted) {
    // if vote request not successful
    if ($.trim(voted) != "1"){
        // display error message
        alert(voted);
    }
    else {
        // disable vote button
        $('#btnVote'+String(offset+1)).toggleClass("voted");
        $('#btnVote'+String(offset+1)).prop('disabled', function(i, v) {
return !v; });
    }
},
});
}

$(document).ready(function(){
init();
// if login button pressed
$('#btnLogin').click(function(){
    login();
});
// if enter key pressed in password box
$("#password").keypress(function(e) {
    if(e.which == 13) {
        login();
    }
});
// if search button pressed
$('#btnSearch').click(function(){
    // reset search counter so 1st result is shown
    localStorage.setItem("searchResCount",0)
    // open search results module
    searchOpen();
    // send search request
    songSearch();
});
// if enter key pressed in track or artist box
$(".searchBox").keypress(function(e) {
    if(e.which == 13) {
        localStorage.setItem("searchResCount",0)
        searchOpen();
        songSearch();
    }
});
$('.btnYes').click(function{
    songReq();
    // update page after request sent
    setTimeout(refresh,1000);
    // close search results
    searchClose();
});
$('.btnNo').click(function{
```

```
// increment search result index and update search result

localStorage.setItem("searchResCount", (parseInt(localStorage.getItem("searchResCount"))+1));
    songSearch();
});
// if skip button pressed
$('#btnSkip').click(function() {
    skipReq();
});
// if vote button pressed
$('.btnVote').click(function() {
    // get position of song voted for as offset from left side
    var offset = $(this).data('offset');
    voteReq(offset);
});
window.setInterval(function() {
    // update page with latest data every 5s
    refresh();
}, 5000);
});
```

```
<?php

function getNextSongCover($index) {
    return query('SELECT art FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT
1 OFFSET '.mysql_real_escape_string($index), 'art');
}

function getNextSongName($index) {
    return query('SELECT name FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT
1 OFFSET '.mysql_real_escape_string($index), 'name');
}

include 'main.php';
// user 'action' from login.js to branch to different logic
switch ($_POST['action']) {
    // validate login details, set user.loggedIn as 1 if validation passed
    case "getNextSongCover":
        // establish connection with mySQL
        connect();
        echo getNextSongCover($_POST['PHPindex']);
        break;
    case "getNextSongName":
        connect();
        echo getNextSongName($_POST['PHPindex']);
        break;
}
?>
```

```
<?php
// establishes connection with mySQL server
function connect(){
    $con = mysql_connect('censored', 'censored', 'censored');
    mysql_select_db('Big_Julie', $con);
}
// generalised mySQL query function, unwraps data from mySQL array before returning
function query($query,$returnAs) {
    return (mysql_fetch_array(mysql_query($query))[$returnAs]);
}
// update active counter of specific user
function updateActive($username,$active){
    mysql_query('UPDATE User SET active = '.$active.' WHERE username =
\''.mysql_real_escape_string($username).'\'');
}
// return loggedIn flag of specific user
function userLogged($username){
    return(query('SELECT loggedIn FROM User WHERE username =
\''.mysql_real_escape_string($username).'\'','loggedIn'));
}
// return number of active users
function getActiveUsers(){
    return (query('SELECT COUNT(active) AS activeUsers FROM User WHERE active >
0','activeUsers'));
}
?>
```

```
<?php
include 'main.php';
// return whether username and password match in user database
function loginValid($username,$password) {
    return (query('SELECT COUNT(password) AS loginValid FROM User WHERE username = \'' .mysql_real_escape_string($username).'\' AND password = \'' .mysql_real_escape_string($password). '\'','loginValid'));
}
// set value of loggedIn in user database
function toggleLoggedIn($username,$loggedIn) {
    mysql_query('UPDATE User SET loggedIn = \'' .($loggedIn). '\' WHERE username = \'' .mysql_real_escape_string($username). '\'');
}
// use 'action' from login.js to branch to different logic
switch ($_POST['action']) {
    // login request
    case "login":
        // establish connection with mySQL
        connect();
        // validate login details
        $loginvalid = loginValid($_POST['PHPusername'],$_POST['PHPpassword']);
        // if validation passed
        if( $loginvalid == 1){
            // flag user as logged in and active
            toggleLoggedIn($_POST['PHPusername'],1);
            updateActive($_POST['PHPusername'],10);
        }
        // return if user login is valid to login.js
        echo $loginvalid;
        break;
    // logout request
    case "logout":
        connect();
        // flag user as logged in
        toggleLoggedIn($_POST['PHPusername'],0);
        break;
}
?>
```

```
<?php
function validateSongReq($username,$currentTime,$URL) {
    // get if user logged in
    $loggedIn = userLogged($username);
    //if user logged in
    if( $loggedIn == 1){
        // get time of last request by user
        $lastReqTime = getLastReqTime($username);
        // get number of active users
        $activeUsers = (getActiveUsers());
        // calculate cooldown time for request
        $reqCooldown = 20000*($activeUsers)^0.50;
        // if cooldown is over
        if ((($currentTime - $lastReqTime) > $reqCooldown) {
            // if song not already requested
            if ((checkPresent($URL)) == 0 ){
                // signal to push request to database
                echo ("p".$reqCooldown);
            }
            // if song already requested
            else{
                echo "already requested";
            }
        }
        // if cooldown still active
        else{
            echo ($reqCooldown);
        }
    }
    // if user not logged in
    else{
        echo "not logged in";
    }
}

function getLastReqTime($username) {
    return(query('SELECT lastReqTime FROM User WHERE username =
\'' .mysql_real_escape_string($username). '\'', 'lastReqTime'));
}

function checkPresent ($URL) {
    return (query('SELECT COUNT(URL) AS present FROM Request WHERE URL =
\'' .mysql_real_escape_string($URL). '\'', 'present'));
}

function pushSongReq ($URL,$name,$artist,$art,$length,$reqTime,$skipCount) {
    // prevents request for default song values
    if ($name != "Title"){
        mysql_query('INSERT INTO `Big_Julie`.`Request` (`URL`, `name`, `artist`,
`art`, `length`, `reqTime`, `skipCount`) VALUES
(\'' .mysql_real_escape_string($URL). '\',
\' .mysql_real_escape_string($name). '\',
\' .mysql_real_escape_string($artist). '\',
\' .mysql_real_escape_string($art). '\',
\' .mysql_real_escape_string($length). '\',
\' .mysql_real_escape_string($skipCount). '\')');
    }
}
```

```
\'\''.mysql_real_escape_string($reqTime).\'\",
\'\''.mysql_real_escape_string($skipCount).\'\')');
mysql_query('INSERT INTO `Big_Julie`.`Vote` (`URL`, `username`) VALUES
(\'\''.mysql_real_escape_string($URL).\'\', \'julie\')');
}

function updateLastReqTime($username,$currentTime){
    mysql_query('UPDATE User SET lastReqTime = \'\''].$currentTime.'\'WHERE
username = \'\''.mysql_real_escape_string($username).\'\'');
}

include 'main.php';
// user 'action' from login.js to branch to different logic
switch ($_POST['action']) {
    // validate login details, set user.loggedIn as 1 if validation passed
    case "validateSongReq":
        // establish connection with mySQL
        connect();
        // reset user active counter
        updateActive($_POST['PHPusername'],10);
        echo
validateSongReq($_POST['PHPusername'],$_POST['PHPcurrentTime'],$_POST['PHPURL']);
        break;
    case "pushSongReq":
        connect();
        // add song request to database
        pushSongReq($_POST['PHPURL'],$_POST['PHPname'],$_POST['PHPartist'],$_POST
['PHPart'],$_POST['PHPlength'],$_POST['PHPreqTime'],$_POST['PHPskipCount']);

        // update last request time of user to current time and reset active
        countdown
        updateLastReqTime($_POST['PHPusername'],$_POST['PHPreqTime']);

        break;
}
?>
```

```
<?php
function getURL($offset) {
    return query('SELECT URL FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT
1 OFFSET '.($offset+1), 'URL');
}

function voteReq($username,$offset) {
    // get whether user is logged in
    $loggedIn = userLogged($username);
    // if user logged in
    if ($loggedIn == TRUE) {
        $URL = getURL($offset);
        pushVote($username,$URL);
        return 1;
    }
    // if user not logged in
    else{
        echo "not logged in";
    }
}

function pushVote($username,$URL) {
    mysql_query('INSERT INTO `Big_Julie`.`Vote`(`username`, `URL`) VALUES
(\'' .mysql_real_escape_string($username). '\',
\' .mysql_real_escape_string($URL). '\') ');
}

include 'main.php';
switch ($_POST['action']) {
    case "voteReq":
        connect();
        updateActive($_POST['PHPusername'],10);
        echo voteReq($_POST['PHPusername'],$_POST['PHPoffset']);
        break;
}
?>
```

```
<?php
function skipReq($username) {
    // get whether user is logged in
    $loggedIn = userLogged($username);
    // if user logged in
    if ($loggedIn == TRUE) {
        // if user not already skipped song
        if (getSkipped($username) == 0) {
            // set user to have skipped song
            updateActive($username, 10);
            setSkipped($username);
            return 1;
        }
        else {
            return ("already skipped song");
        }
    }
    else {
        return ("not logged in");
    }
}

function getSkipped($username) {
    return query('SELECT skipped FROM User WHERE username =
\'' .mysql_real_escape_string($username) . '\'', 'skipped');
}

function setSkipped($username) {
    mysql_query('UPDATE User SET skipped = 1 WHERE username
=\'' .mysql_real_escape_string($username) . '\'');
}

include 'main.php';
switch ($_POST['action']) {
    case "skipReq":
        connect();
        // reset active counter for user
        updateActive($_POST['PHPusername'], 10);
        echo skipReq($_POST['PHPusername']);
        break;
}
?>
```

```
<!DOCTYPE html>
<html>

<head>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
    <script src="play.js"></script>

</head>

<body>
    <img src = "julie.png"></img>
</body>

</html>
```

```
var Spotify = {
// http://khlo.co.uk/stuff/spotify/
  currentSong: '',
  play: function(songId) {
    if (songId != this.currentSong) {
      if (!document.getElementById('spotifyPlayer')) {
        spotPlayer = document.createElement('div');
        spotPlayer.id = 'spotifyPlayer';
        document.body.appendChild(spotPlayer);
      }
      document.getElementById('spotifyPlayer').innerHTML = '<iframe src="spotify:track:' + songId + '" height="1" width="1" style="visibility: hidden;"></iframe>';
      this.currentSong = songId;
    }
  }
}

function init(){
  // ensure that first song in queue will play on startup
  localStorage.setItem("playing","false")
}

function play(){
  // set song playing flag as true
  localStorage.setItem("playing","true")
  // get URL of song to play
  getURL();
}

function getURL(){
  // get URL of song to play from request database
  $.ajax(
  {
    type: "POST",
    url: "play.php",
    dataType: "text",
    data: {action:"getURL"},
    success: function (song) {
      // get length of song to play
      getLength($.trim(song));
    }
  });
}

function getLength(URL){
  // get Length of song to play from request database
  $.ajax(
  {
    type: "POST",
    url: "play.php",
    dataType: "text",
    data: {action:"getLength", PHPURL:URL},
    success: function (length) {
      // reduce length of song by 1 second to allow for processing time
    }
  });
}
```

```
between song changes
    length = length - 1000;
    playSong(URL,length);
}
});
}

function playSong(URL,length){
    Spotify.play(URL);
    // set timer to execute end of song logic when song ends
    var timer = setTimeout(function(){songEnd(URL)},length);
    localStorage.setItem("URL",URL);
    localStorage.setItem("timer",timer);
}

function songEnd(URL){
    // push song to end of queue
    var currentTime = getCurrentTime() + 1000000000000000 ;
    $.ajax(
    {
        type: "POST",
        url: "play.php",
        dataType: "text",
        data: {action:"nextSong", PHPURL:URL, PHPtime:currentTime},
        success: function () {
            // flag that no song is playing
            localStorage.setItem("playing","false");
        }
    });
}

function getCurrentTime(){
    // return current UNIX timestamp in seconds
    return(Math.floor(Date.now()));
}

function getSkipped(){
    var URL = localStorage.getItem("URL");
    // get whether song has met criteria for skipping
    $.ajax(
    {
        type: "POST",
        url: "play.php",
        dataType: "text",
        data: {action:"getSkipped", PHPURL:URL},
        success: function (skipped) {
            // if song has met skipping criteria
            if ($.trim(skipped) == "true"){
                // disable countdown timer for song end
                var timer = parseInt(localStorage.getItem("timer"));
                clearTimeout(timer);
                // run logic as if song had ended
                songEnd(localStorage.getItem("URL"));
                dropSong();
            }
        }
    });
}
```

```
        }
    });
}

function dropSong(){
    var URL = localStorage.getItem("URL");
    $.ajax(
        {
            type: "POST",
            url: "play.php",
            dataType: "text",
            data: {action:"dropSong", PHPURL:URL},
            success: function () {

            }
        });
}

$(document).ready(function(){
    init();
    // check for song playing every second
    window.setInterval(function(){
        // if song not currently playing
        if (localStorage.getItem("playing") == "false"){
            // play song
            play();
        }
    }, 1000);
    // check for song skipped every second
    window.setInterval(function(){
        getSkipped();
    }, 1000);
});
```

```
<?php
function getURL($offset) {
    return query('SELECT URL FROM Request ORDER BY `Request`.`reqTime` ASC LIMIT
1 OFFSET '.mysql_real_escape_string($offset), 'URL');
}

function getLength($URL) {
    return query('SELECT length FROM Request WHERE URL =
\'' .mysql_real_escape_string($URL) . '\'', 'length');
}

function nextSong($URL, $time) {
    mysql_query('UPDATE Request SET ReqTime =
.mysql_real_escape_string($time). WHERE URL =
\'' .mysql_real_escape_string($URL) . '\'');
}

function votes() {
    for ($i = 0; $i <=3 ; $i++) {
        $URL = getURL($i);
        mysql_query('UPDATE Request SET reqTime = reqTime - ((SELECT
SUM(voteWeight) FROM User WHERE username IN ( SELECT username FROM Vote WHERE URL
= \'' .mysql_real_escape_string($URL) . '\' ) )*60000) WHERE URL =
\'' .mysql_real_escape_string($URL) . '\'');
        mysql_query('DELETE FROM Vote WHERE URL =
\'' .mysql_real_escape_string($URL) . '\' AND username != \'julie\'');
    }
}

function activeDec() {
    mysql_query('UPDATE User SET active = active - 1 WHERE active > 0');
}

function setSkipped() {
    mysql_query('UPDATE User SET skipped = 0');
}

function getSkipped($URL) {
    if (getNewReq($URL) == 1) {
        return "true";
    }
    // get number of votes to skip current song
    $skipCount = getSkipCount();
    //get number of active users
    $activeUsers = getActiveUsers();
    // if more than 60% of users have voted to skip
    if ($skipCount > ($activeUsers*0.6)) {
        return "true";
    }
    else{
        return "false";
    }
}

function getNewReq($URL) {
```

```
$newURL = getURL(0);
if ($URL != $newURL) {
    return 1;
}
else {
    return 0;
}

function getSkipCount() {
    return query('SELECT COUNT(skipped) AS skipCount FROM User WHERE skipped = 1', 'skipCount');
}

function dropSong($URL) {
    mysql_query('DELETE FROM Request WHERE URL =
\'' .mysql_real_escape_string($URL) . '\'');
}

include 'main.php';
switch ($_POST['action']) {
    case "getURL":
        // establish connection with mySQL
        connect();
        // get URL of next song to play
        echo getURL(0);
        break;
    case "getLength":
        connect();
        //get length of song
        echo getLength($_POST['PHPURL']);
        break;
    case "nextSong":
        connect();
        // update request table since song has ended
        nextSong($_POST['PHPURL'], $_POST['PHPtime']);
        votes();
        // decrement active counter of every user
        activeDec();
        // set each user to have not voted to skip
        setSkipped();
        break;
    case "getSkipped":
        connect();
        // get whether song has met all skip criteria
        echo getSkipped($_POST['PHPURL']);
        break;
    case "dropSong":
        connect();
        dropSong($_POST['PHPURL']);
        break;
}
?>
```