A

Project Report ON

Online Music player

For the Partial fulfillment of the award of

Degree of

Master of Computer Application (MCA)

Under the Guidance of:

Dr. SC Dimri

Submitted by:

Name:- Abhishek kumar
MCA- 4th semester
Roll No:-1102158
Student ID:-21271013

GRAPHIC ERA DEEMED TO BE UNIVERSITY, DEHRADUN (UTTRAKHAND)

GraphicEra
(Deemed to be University)
Accredited by NAAC with Grade A

Candidate's Declaration

I hereby declare that the project work, which is being presented in the Project Report, entitled **"Online Music player"** in partial fulfillment for the award of Degreeof'MasterofComputerApplication"inDep't.OfInformationTechnology, **Graphic Era Deemed to be University** is a record of my own investigations carried under the Guidance of **Dr. SC Dimri**.

I have not submitted the matter presented in this Project Report anywhere for the award of any other Degree.

Signature of Supervisor **(Name and Signature of Candidate)**

Enrolment No:-……..............

Acknowledgement

First of all, my sincere and wholehearted gratitude to **Dr. varsha mittal** (Class Coordinator) of computer application department, Graphic Era Deemed to be University, Dehradun to encourage me and to provide me opportunity to prepare the project. I am also thankful to my friends for their encouraging guidance and supervision in project.

I also express my sincere and gratitude to my guide **Dr. SC dimri** whose guidance and supervision given to me throughout the project.

Table of Contents

Abstract:-

The continuous growing of people’s music library requires more advanced ways of computing playlists through algorithms that match tracks to the user’s preferences. Several approaches have been made to enhance the user’s listening experience; while most of them rely on the music content provided by the user, this project presents an online application that sources the audio content from publicly available resources (YouTube). A playlist generation algorithm is developed that uses only one seed track to compute a playlist of arbitrary length. For sourcing the audio content, YouTube’s track coverage is analyzed and statistics show that, in a real-life usage scenario, almost 80% of the tracks are available while the rest have rather lower popularity. The resulting application is a fully functional but feature limited online music player that can also serve as a framework for future playlist generating algorithms or other content sources.

PREFACE:-

vital for the development of economy Effective management of projects is because development itself is the effect of series of successful managed projects. This makes project management extremely important problem area for developing economy such as ours. Unfortunately many projects experience schedule slippage and cost overruns due to variety of reasons. To remedy the situation, a project has to be meticulously planned, effectively implemented and professionally managed to achieve the objective of the time, cost and performance. Computerization of the project management can play a major role in streamlining the management of project. Thus we see the computer becoming necessity in the day to day life. The use of computer also involves the feeling of healthy competition with the organization receiving much attention these days. Almost everyday uses of paper carry advertisements asking for project managers. The scenario was not so bright a few years back. For that matter even today though lots of seminars are held on project management, name of the universities in India offer any course to students to formally qualify as project managers. Thus, while there is very little supply. This has created all sorts of problems. Project management, unfortunately, is not but project manager practice, our attempts to demonstrate how project management, as it is concerned, can be put into practice.

INTRODUCTION

Music has always been a means of entertaining people even from the earliestages of the civilization. Historically it was produced by musicians and only available during life concerts. The technological evolution made it possible to save the music on vinyl plates, later electromagnetic charged stripes, CDs until the technology brought us to saving tracks digitally. When dealing with a huge collection of tracks, people encounter management problems they did not have before. So they have to develop new ways of using the music collection for their entertainment. Playlists are a good approach for saving successions of tracks that one likes. The most dominant problem of existing playlist generation mechanisms is, however, their lack of flexibility: new tracks are not automatically added, they don't adapt to the user's current mood etc.

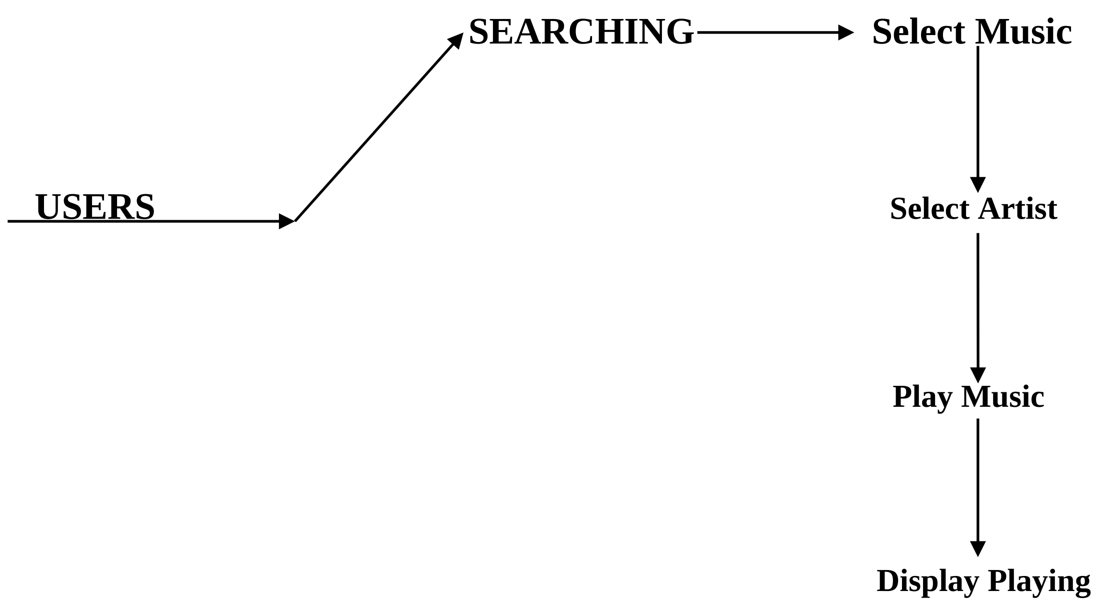
A new approach in dynamically organizing tracks into playlists is on its way: companies like last.fm already suggest an algorithm of mapping songs one toeach other based on their "similarities"; but how to compute these similarities? One way, that did not prove to be very productive, is to analyze the audio contentof the track – its audio frequencies. This way, tracks are split in categories like "Heavy Metal" and "Blues", but people do not like all tracks of a certain gender and these genders might be inaccurate. Another way, which is given more and more attention by researchers and companies worldwide, is computing similarities between tracks based on user input. As an example: if two users add the same two tracks to their playlists, one can deduce that these tracks aresimilar and so, also other people that pick one of them are likely to enjoy the other one as well.

Lorenzi (Lorenzi, 2007) proposes a way of representing the similarity between tracks in a 10-dimensional Euclidian space (further called music space), where the closeness of tracks is approximately proportional to their similarity. 7M songs currently appear in the database, but only 500K of them have enough user statistics to be mapped in the graph. Using this simplified and computationally efficient way of finding similar tracks, several applications can explore new ways of computing playlists. Most of them offer support in playlist generation but none also provides the tracks to be played. This could be seen as a disadvantagebecause not all people possess all tracks that are suggested by the space.

This report sets its goal in developing an application that uses the space of music for computing intelligent playlists and, more important, trying to deliver the required songs to the user through publicly available tracks on YouTube.com. For promotion purposes and also user data gathering, it was chosen to make the application available on Facebook[1], which ensures an easy referral to friends.

The development process is not an easy one as many technologies are requiredto work together to provide the user with a good experience. The huge amount ofdata that has to be processed is demanding with respect to both algorithms and underlying hardware. In the end of this report, a fully functional, but feature- limited, version of an online music player is presented. This application could be the step stone of future, more detail-oriented applications because it sets the basis of playground: an online music player.

Manual Process:-



PROJECT OVERVIEW

Overview

1 Introduction

Introducing the current report and placing it aside of similar ETH internal and other works. Some introductory words about the music space, the goal and the approach used in this report.

2 Problem Statement

This chapter debates the necessity of a new solution to make the world of intelligent track comparison even more accessible to the end user, followed by an approach sketch.

3 Related Work

When a new, revolutionary, product comes to market it usually has a mixed impact on the audience. This chapter analyzes some of the achievements but also flaws of existing solutions and therefore represents a base for the discussion in the Vision Chapter.

4 Vision

This chapter sets the ground rules for developing the application, mostly based on experience gathered from already existing implementations. Important decisions about audio content sourcing and application promotion are discussed and reasoned in this and also the next chapter.

5 Audio Content Provider

This chapter analyzes advantages and disadvantages of the different content providers and presents a coverage analysis for YouTube.

6 Architecture

After discussing the application's approach to serving its goal, it's time to describe the technical details of the implementation. What functions should the client and server side implement? How modularized can and should the application be build? What's the best approach in dealing with the huge data amount? What advantages do the different APIs bring and how can they be used?

7 Analysis

Musical Analysis is a study of how the composers use the notes together to compose music. Those studying music will find differences with each composer's musical analysis, which differs depending on the culture and history of music studied. An analysis of music is meant to simplify the music for you

8 Conclusions

The conclusions section summarizes the projects final state as well as reconsiders the main ideas and results.

9 Discussion and Future Work

The Section deals with the ideas that didn’t make it to the development process mainly because of the tight time scheduler, but that are important to mention and at some point or another in a future development they have to be addressed.

enterprising individuals who have an ambition of revolutionizing the Singapore movie scene. The website www.cinema.com.my first saw the light of a screen in October 1998. Since then it has grown to include thousands of pages worth of content. You can browse and find all the latest happenings of movies, be they local or international. News, reviews, previews, and contests are amongst its highlights. All this from the humble beginnings of Cinema

Show times, now encompassing almost ALL Malaysian cinemas, and as always, updated daily for your convenience. It includes home, movies, charts, features, trailers and show times.

So while doing research on these systems it has been observed that there are lots of bugs in the online movie system. The major issue is speed, and it is almost with all the systems. Due to increasing traffic and demand of online system the system is sometimes not able to bear the load. Often at the time of release of some famous and highly rated movies the server gets down or payment portal fails. Most of the time the amount gets reduced from the users account and payment is not successful, also apart from that there is also a bigger issue of fake review. There are agents of the entire production house which are in existence. They have their social networking cell which works to boost up the review and rating of a particular movie. Fake coupon code and café booking of tickets has also increased black marketing in this field. One more pattern that has been observed I that systems from foreign countries like Dubai, Singapore and USA are much more reliable that the system of Asian countries. To remove this, proposed alternative is using NODE JS and Mongo DB connectivity which is able to increase the speed of system by almost thirty times. Detailed class diagram and software information has been discussed in the proposed methodology section of the research paper. To reduce any kind of fake reviews and rating authorized users will only be able to post the rating and review of any movie will be first checked by the editorial team and then it will be posted on the web site

1.Problem Statement

This chapter debates the necessity of a new solution to make the world of intelligent track comparison even more accessible to the end user, followed by an approach sketch.

SYSTEM REQURIMENTS SPECIFICATION

Software requirements

OPERATING PLATFORM: WINDOWS 2000/NT/XP

RDBMS: SQLSERVER 2000

SOFTWARE: VS.NET 2003

FRONT END TOOL: HTML,CSS,JAVA SCRIPT,PHP

Hardware requirements

RAM: 128MB RAM

HARD DISK: PC with 20 GB hard-disk

Motivation

Several solutions already use intelligent playlists embedded in music players installed on computers. There are also online solutions, the most popular of which is last.fm, which acts as a personalized radio station that plays preferred music. On the other hand it does not allow playback of a certain track. There are also other solutions, like the genius function of iTunes or the Music Explorer; both use the user's music collection to generate playlists. The biggest disadvantage of the latter solution is that the user can use only tracks that he/she already has on his/her PC to generate playlists. Of course this limits the power or the algorithm very much.

There are already services that provide the music content (like last.fm or YouTube to name a few) so it's a natural conclusion to try to use these services in connection with the playlist-generating algorithm.

In order to understand the utility of such an application, just imagine the following scenario: one enjoys listening to music while working. It is not common to store music on the company's computer so one rather has a personal mp3 player with himself during office time. If one takes enough time to prepare ones playlists in order to fit ones current mood, it is a pretty decent solution. But what if new tracks appear that one might like? One first has to do serious research in order to find them and then go through buying them, downloading them to his/her mp3 player, updating the playlists… it already sounds very difficult, right? Now the suggested scenario is the following: one opens a website, types in a track that reflects ones current mood and hits "play". That's it! The player chooses tracks that one likes, also plays new tracks that one did not hear before, and can go like this for hours and hours without repetition. One can go on with one's work and in order to stop the music, one only has to hit stop or close the browser. The simplicity of the solution speaks for itself.

The goal of this thesis is to analyze and implement an approach of building such a web-based music player. The questions it has to find answers for are: How should the user interaction be designed to maximize the user satisfaction? Where to source the audio (and video) data from, while ensuring a maximum coverage? And finally, how to promote the application in order to attract as many users as possible?

The different implementation possibilities are evaluated and the best solution is implemented. The logic behind the web-based music player computes a

sequence of tracks based on their similarity. At the same time, user behavioral data is gathered that helps further releases to be even more user friendly. Another important aspect of the application is its extensibility. Modularity and code reusing are very important parameters of this application, as it acts as a version 1.0 for future releases. These future releases will be able to interact with the user for finding the best track video on YouTube or to determine the music preferences of users and even adapt the space to the new usage statistics.

Approach

The analysis of the currently available tools to accomplish the task is one of the most important steps because the ground concepts of the application should never change, regardless of its future complexity.

The several possible implementations of the web service together with the balancing of computing tasks between server and client are the first parameters that have to be defined for a solid base. Also the programming language plays a crucial part in the development process, as it is shown later. The amount of callbacks to the database in favor of less memory usage is also an important aspect that is difficult to estimate from the start. In order to allow a high flexibility while still maintaining a small dataflow, the implementation of the logic is mainly on the server. The UI responsibility is fully retained by the client side as well as servicing UI requests and only notify the server of such activity.

In order to achieve the high goals that were set, the structure of the application is important to be highly modularized to allow interchanging the modules with better, more complex implementations. It is important to determine which components are possible and also easy to modularize, without introducing too much communication overhead in the interfaces. It turns out that the music content related jobs can easily be modularized, as well as the DB related jobs andthe playlist computing tasks. The core of the application only needs to handle these modules and the logging task. Also the communication with the client is modularized, making it particularly easy to implement new clients running on the same service or new services to serve the same client.

One of the hardest tasks is determining how to provide the audio content to the user. The playlist computing is based on a music space implementation, described later. For the audio content there is anyway no guarantee because it is an externalized task. Several providers of audio content are analyzed and thebest one implemented in such a way that it is reliable for the users.

For the client side, JavaScript and Flash are the only two solutions to provide a good user experience because there is no need to install any software. When working with JavaScript, AJAX (Wikipedia, AJAX (programming), 2008) is needed for the seamless communication between client and server – thus enabling a richer user experience – and also for avoiding communication overhead. GWT (Google, 2008) dramatically eases the interaction and object exchange between the client side and the web service because this tool enables the programming of the client side in java and converting it to JavaScript.

Finally, one of the most important steps in the future extension of this application is the logging of user activity. The activity of the users has to be

logged in such a manner that constructive information can be extracted from the logs. It’s important to build the logs right such that future ideas can find the data to base their research on. For this step no resource is too expensive. One has to differentiate between user activity and user feedback. User activity can be represented through actions like: jumping to a certain track, rearranging the generated playlist; user feedback is the action of notifying the application of a bad chosen track or of a track that the user doesn’t like. In the end the logs are saved in the database and their exact structure discussed.

In the end of the report, an implementation is presented showing a web based, personalized, music player. It is an interactive web site where the user selects a favorite track and generates a playlist based on this. The playlist has the following properties:

- Each track in the playlist is similar to the one before;
- A track is only allowed to be repeated if it has not been in the last 24 tracks; this should guarantee about two hours of non-repeating tracks;
- Tracks from the same artist do not appear more than at least 4 tracks apart;
- The user has the possibility to reorder the tracks, to remove them or toplay them directly, without waiting for their turn to come;
- The music video of the currently playing track appears on the right;
- The user has the possibility to open the playing track in YouTube (with possibility to switch to full screen).
- The user has the possibility to give feedback about the quality of the recording or weather he/she likes the track or not;
- All user activity like reordering playlist, searching for tracks, trackremoval – but also his/her direct feedback – is logged for future analysis.

After describing some of the most important features the application has to cover, an analysis of already existing solutions is made in the Related Work Chapter.

2 Related Work

When a new, revolutionary, product comes to market it usually has a mixed impact on the audience. This chapter analyzes some of the achievements but also flaws of existing solutions and therefore represents a base for the discussion in the Vision Chapter.

ETH Projects

Several projects belonging to the Distributed Computing Group, part of the Electrical Engineering Institute of the ETH Zürich, have presented different approaches as a playlist generating tool based on a specially created music space.

The first and most important was the creation of the music space itself (Lorenzi, 2007). Based on user statistics provided by Last.fm, Lorenzi presented a way of representing the data such that the similarity between two tracks can be quantified. By mapping tracks to points in a 10-dimensinoal space, the similarity of two tracks can be easily computed as the Euclidian distance between them. Through this, computationally hard problems become easy accessible for other applications, such as: comparing similarities between several track pairs, computing the nearest neighbor, finding tracks along a certain path or "negating" a track – meaning finding another track that is far away from this track. A furtherdiscussion of the correctness of this mapping is out of the scope of this report; therefore this report considers the provided music space as efficient and uses it as it is.

Based on the previously described music space, several applications were developed to bring the advantages of the music space to the end user. The musicexplorer web application (Gonukula, Kuederli, & Pasquier, 2008) presents an approach of organizing the user's music collection based on an online service. The user has the possibility to upload the titles of its music collection, select two of them and the algorithm computes a playlist that is a smooth crossover from one track to the other. Both, the musicexplorer and the application presented in this report, allow the user to generate intelligent playlists. The downside of the musicexplorer is the requirement that the user already possesses the tracks to be played. If his/her collection is limited, so is the playlists at the output. Using publicly available tracks not only make it easier for the user but also the results are be better.

Bossard presents a new way of exploring the Space of Music with a visual approach that should allow the user to visually select tracks from the music space (Bossard L. , 2008). The ideas are implemented on the android platform through a customizable music player.

External Projects

Several companies have recently discovered the advantages of user-personalized playlists and are currently building their business model based on providing high quality but simple to use solutions to their customers. Last.fm is one of the first ones on the market to offer personalized music playback; it achieves this through an online "radio station" that is supposed to play music that the user likes. The Last.fm approach is very similar to the one proposed by this report: it is an online music player that usually plays the user's favorite tracks and also introduce new tracks if available. However, because the personalization of the player is done on a per-user basis (opposed to the per-session proposal of this report) it could happen that it does not react so fast to mood changes in the user's behavior. The biggest drawback is still the inability of the user to select certain tracks to play; even if he/she has a certain favorite track in mind, he/she has to stick to the automatically generated playlist of the player. This limitation is probably more license-based than technological, but it is a field where improvement is desired.

Another approach to delivering similar tracks to a user is the Genius function of the Apple iTunes music player. The principle is fairly easy: iTunes analyzes your music library, submits the tracks to an iTunes server, which returns popularity measurements personalized for your tracks. During playback, the user has the possibility to select a track and let the application generate a playlist containing tracks similar to the selected one. The biggest drawback of this approach is the limitation to the user's personal computer, where he/she holds his/her private music collection.

During the development process, two new companies launched an implementation that is very similar to the presented approach. One of them is Songza (www.songza.com), an online music player where users can search for a track and play it. The audio (and sometimes also video) content is sourced from YouTube or Last.fm. The user has the possibility to manually create a playlist. However no mechanism of automatically generating playlists is yet available. The second, and much more similar to this report's implementation, is DropPlay (www.dropplay.com). It allows the user to search for songs, generate playlists based on similarity and also share these songs with friends on Facebook. Only time will show which of the two implementations will be better for the users.

The discussion of external projects is continued in the Vision chapter, where the pros and cons of the mentioned applications are further analyzed and weighted.

3 Vision

This chapter sets the ground rules for developing the application, mostly based on experience gathered from already existing implementations. Important decisions about audio content sourcing and application promotion are discussed and reasoned in this and also the next chapter.

Requirement Specifications

As presented in the related work section, there already are several implementations that are more or less similar to this report's solution. All of them have their strengths and weaknesses so they are going to be analyzed and combined to work in the targeted scenario. The goal is to create a web based application that is simple to use and that can function independent of the users resources. What can be learned from other approaches and what can be used to increase the value of this implementation?

Compared to the current music explorer web application, the designed application is also a web-based service and targets the same audience: people who want to have their music experience enhanced without a lot of effort on their side. The users interact with the existing music explorer application bysubmitting all the titles of their music library and then selecting two of those tracks that are used as endpoints of a playlist. All the tracks in between represent a smooth progress from one track to the other. This implementation faces the problem that it completely depends on the user to provide the audio content. The quality of the offered result is only as good as the number and variety of submitted tracks; even the best implementation performs poorly if it is fed with low quality resources. Also the mapping of tracks from the user's tags[2] to the applications tags is a problem to take into account. The lack of own music content is also a problem for people that do not own a big music collection or do not have it at hand. Hence, the current application sets a high priority to providing the intelligent playlists together with the audio content. This ensures both a greater QoS[3] and ease of use.

Generating an intelligent playlist and then playing the tracks is similar to Last.fm. In the case of Last.fm however, license limitations prevent the user to search for specific tracks and play them. As long as the designed application has no restriction in this matter, it is a high priority to implement such a popular feature. A search bar allows the user to browse through the library and select individual tracks to play. Last.fm also avoids displaying an actual playlist, although most users like to at least see the tracks that follow, and possibly even edit this list. To honor the playlist generating feature of the application, a playlist is displayed to the user together with the possibility to add, remove and rearrange tracks from the list. This makes the application very similar to the music players the user is already used to.

As discussed in the last paragraph, the UI[4] imitates a common music player such that the users do not have to get used to something completely new. This brings the question wheater to implement the application as an installable – standalone application or as a web service running in a browser. Modern technologies like JavaScript and AJAX allow a simple development of a solution running in the web browser. Also encouraging for using this approach is the fact that the application most likely provides online-based streamed audio content. Online applications provide decent results if they don't have resource-consuming client side tasks; also they are easy to update and maintain. In respect to all the mentioned pros, it makes sense to develop the application as a JavaScript based implementation that runs directly in the web browser, without the need to download or install anything.

User Perspective

As one of the important goals is to have a user-friendly application, it is necessary to provide good results with a minimum of information requested from the user. As a second, but also very important feature, the user actions have to be logged for understanding and improving the application's behavior. The logging should be made optional, to avoid privacy issues.

Understanding the resources that are at disposal is necessary to find the minimum information required to return satisfactory results. In order to provide tracks that the user likes to listen to, the application has to know which are the preferences of the user. Last.fm does that by tracking the user's preferred tracks over several sessions and probably computes a user profile based on this behavior. This approach is an elaborate one but introduces a huge overhead in the user-management algorithms. And if the user decides at some point to listen to something else, the inertia of the algorithm makes this very difficult. The best approach that also ensures a minimum user interaction is no user tracking at all. The application should work well for each user, independent on his/her history with the application.

[3] Quality of Service

[4] User interface

Choosing not to implement any user management requires obtaining as much information as possible at the start of each session. In order to make it very flexible, the application can ask for “guidelines” (favorite tracks of the user) each time it’s requested to compute a new playlist. The approach of another ETH project was to ask the user for several seed[5] tracks and computing a virtual path in the music space that connected all the points. Of course the seed list was also rearranged such that the resulting path is minimal. Using several seeds is a good approach because it gives more details about the user’s current music preference. But on the other side, the user might feel overwhelmed by the information he/she has to provide to create a simple playlist. In respect to that, the first approach is to ask the user for only one seed track. This track is used to create a new playlist, and if the user doesn’t like the offered tracks he/she can insert a new seed or just reshuffle. Tests will show if this minimal user input approach is efficient enough in providing good tracks.

User feedback is very important for improving this application but it is well known that the average user does not usually bother to give feedback; and if he/she does, he/she usually reports features that are not working. To adapt to this attitude, it is good to implement a way of monitoring the user’s actions, like when he/she changed the track, when he/she removed a track from the proposed ones, etc. Also the user has to be given the possibility to report faulty features. Because the music content provider is YouTube and no guarantee canbe given for the played content, the user has the possibility to report tracks that don’t fulfill his/her expectations. For example, if a track is a low quality, concert recorded version, the user can notify the application and other users do not receive it anymore. Another complaint possibility is about the playlist-generating algorithm itself. The user is able to report a track that he/she doesn’t like so much, indicating an inconsistency in the playlist generating algorithm or the underlying music space.

Implementation as Facebook App

Promoting the application to many users is important – among other reasonsalso because user’s behavior is recorded and eventually leads to a better application. Applications usually get promoted through commercials, referralprograms etc. As the application is one of a kind, word of mouth is the best way to promote it. Community networks like Facebook and MySpace[6] are perfect environments for deploying and promoting such community-targeted applications. This networking platforms already have thousands of daily returning users signed up and also facilitate the integration of applications through specially designed APIs. Facebook APIs also offer the possibility to read user data; this could come in handy at a later development stage. Because the possibility of recommending applications between the users is provided by the Facebook platform, the application is basically promoting itself – provided it proves it’s usefulness. Also appearing on a Facebook official site, the application

[5] Seeds or seed tracks refer to the track provided by the user and used by theplaylist generating algorithm to find similar tracks.
[6] www.myspace.com - community networking site

is granted more trust from the users, making it possible to integrate it evenbetter with the community.

Facebook offers two possibilities of designing an application: either the app is written in a Facebook-specific markup language or any other web site. If the latter is chosen, the application runs in an iFrame, of maximum 760 visible pixels in width. Because the application is intended to run also as a stand-alone web site, the deployment in an iFrame is chosen.

Choosing the promotion through Facebook doesn't limit the application in any way. It still runs in a web browser without any restrictions and independent of Facebook. One can see this deployment as an add-on to the functionality.

Also the DropPlay application, presented in the Related Works Chapter, integrates with Facebook and uses Facebook to recommend tracks to friends (and through this indirectly refer the site to the friends). However, the DropPlay application is not deployed as a Facebook app; not doing so neglects the trusty environment for the users. To speculate upon the reasons, the width of the iFrame might be too restrictive for DropPlay's UI.

Having discussed the ground rules of functionality and also reasoned the decisions through arguments mainly from the users perspective, the spotlight continues towards the sourcing of the audio content.

4 Audio Content Provider

This chapter analyzes advantages and disadvantages of the different content providers and presents a coverage analysis for YouTube.

Content Provider Analysis

The necessity of providing public music content was analyzed earlier in this document, concluding that it is important for a good user experience. To be taken into consideration, a music provider has to fulfill the condition that it offers a complete (or high) coverage of the track database. With respect to this requirement, only YouTube and Last.fm can be considered good candidates at this moment.

YouTube's pros are: an easy to use API for implementing the video content in the web browser; a java library for making the search for tracks easy and fast; the tracks are uploaded by users, which means the data is always up to date with the latest tracks. On the downside, because the content is user-generated, the quality of some content is so bad, a results post-processing algorithm is necessary. Also not all tracks are found on YouTube, but popular tracks are usually available.

Last.fm has a better coverage of the tracks in the music space and the quality of the content is also very high. The biggest downside of Last.fm is that the majority of tracks are only 30 seconds snippets of the actual tracks. This is something unacceptable for a music player, thus turning the decision in favor of YouTube.

The developers of Songza couldn't apparently decide for one of the providers and implemented both. It is a good approach because the songs on Last.fm have a

guaranteed quality and the application only switches to YouTube if there is no unclipped track on Last.fm. Implementing both alternatives however, brings problems especially in the UI part of the application. Also indirect user feedback is much harder to trace. This is why the application is relying on YouTube as it's content provider.

5 Architecture

After discussing the application's approach to serving its goal, it's time to describe the technical details of the implementation. What functions should the client and server side implement? How modularized can and should the application be built? What is the best approach in dealing with the huge data amount? What advantages do the different APIs bring and how can they be used?

Client and Server side Functions

The goal of the application is set and also the ground rules – also for implementation – have also been set in the Vision Chapter. The application is designed as a two-sided application: client and server side.

The Client Side

The client side must implement the UI of the application. It must be a stand-alone program that only requests and delivers parameterized data from and to the server. It is important to have a strict and simple communication between the two applications (client and server) such that an extension or even replacement of each one of them is easy to understand by the developer. The client application is responsible for delivering an intuitive UI and handling all requests of the user that are related to the UI. Basically the client implements the whole music player UI, with play, pause, repeat functions as well as playlist generation and modification. The list of tracks actually present in a playlist is generated by the server and returned to the client with enough parameters (video id) suchthat the client itself can download the music content from YouTube. No music content is available from or routed through the application's service.

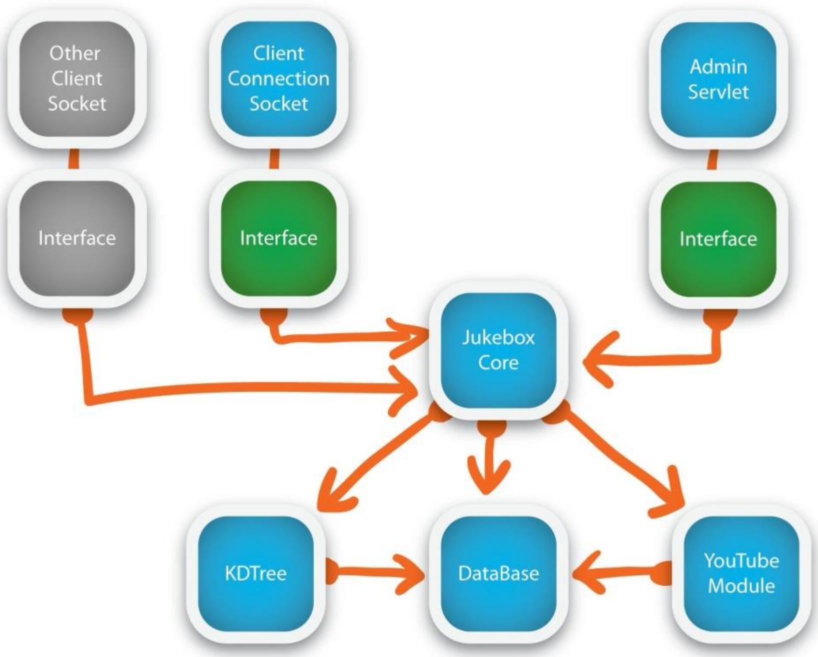
The seed to be used for generating a new playlist is not selected in a traditional 'search and select candidate' manner. A type sensitive field responds to each keystroke with a suggestion list for tracks that contain the words either as artist or title. The user has to select a track from this list and use it as a seed for the playlist he/she wants to generate. The client does not fetch results from the server for each letter the user types. A delay is set such that a suggestion request is sent to the user only if the user has stopped writing for more than 'delay' time ago.

The playlist is fully customizable and offers the user the possibility to rearrange the tracks in order to influence the playing order, to remove and to jump to tracks. No restrictions are made on the rearranging of the playlist; for example if a user whishes to replay a track that he/she already heard, he/she is able to do that. Also the playlist is responsible of always having a minimal number of tracks loaded after the currently playing one. If this number decreases because of playing track advancement, track removal or reordering, the playlist automatically loads new tracks until the minimum size is reached again.

The UI has an area used to display track specific information like title, artist and other. The user uses this area also to directly give feedback about the current track (low quality or bad track choice). A direct link to the implementation on YouTube is also available.

The Server Side

The server must provide the following two services for the client application: playlist generation and suggestion generation. The core of the server is the only non-interchangeable part and it acts as a bridge between the modules to provide the services requested by the client. The server side application consists of modules that handle: the database, the YouTube related data, the music space data and the client connection socket. The client connection socket is distinct from the core module, such that it can also be interchangeable if a different type of client is used to connect to the same server. In this case however, the core has to be modified as well to provide the new functionalities. Also the socket has responsibilities like maintaining the session and converting the data from the structures (Objects) used in the server to the ones used in the communication to the client. Figure 6-1 - Modular Structure of the Server Side Application offers an overview of the modular structure.



The suggestion service[7], uses an algorithm that executes an indexed crawl in the database for the words currently entered by the user. Words shorter than 3 letters are not included to avoid a high number of results. Also the results returned by the suggestion algorithm are sorted by popularity, such that the most popular tracks come up in the limited number of suggestions at an early stage of typing.

The server also offers the possibility to monitor and change its state by other means than the client application or the console. A servlet connects to the service and enquires status information like: state of the music space, state of the db connection or errors that were thrown upon client requests; this information is printed out in a web site so it can be accessed via a web browser. The

[7] Service that proposes tracks as result of user inserted words. Described later.

administration tool can be extended to implement functionality to reinitialize the server, to read and output user statistics etc.

The Communication

Using only standardized communication between the client and server helps future developers to interchange the components seamlessly. It's important for calls to be serviced directly, to return the requested values and to finish upon returning. Otherwise, if calls depend on local variables and states, the communication becomes very complex. The only two states the service can be in are: initializing and running. The initializing state is only at startup so it is assumed no request ever finds the server in this state (except for the administration interface).

Well-defined interfaces are at the connection of the client application to the client socket of the server; between the client socket and the core of the server and between the core and each of the modules of the server application. Several objects have been specially designed for – and are used only in – the communication between server and client. Each request is executed immediately, sequentially without (many) threads running concurrent to compute the results. Responses to requests are synchronous - always returned to the call and not sent back at a later time (when done).

Music Space (KDTree)

The music space is built of tracks that are mapped to points in a 10 dimensional field. To be able to find nodes in this space and get their neighbors, it is necessary to find a method of storing the data to be accessible in a fast way. A KD Tree (Wikipedia, KD Trees, 2009) is a right choice for this, as it is very fast at computing the nearest neighbor – O(log n) – and also nearest neighborhood – basically the two functions needed for the proposed playlist generation algorithm.

There are only few libraries that offer solutions for KD Tree implementations. One of them is Perst[8], a database that can store data in a KD Tree. As Perst is still under development, the KD Tree algorithms are not complete and there is no efficient implementation for retrieving the nearest neighbor or nearest neighborhood. The advantage of Perst was that the whole music space wouldn't have been stored into the memory, but rather in a database.

The second option that is also used because of the shortcomings of Perst, is a KD Tree fully loaded into the memory. KDTree.jar is a KD Tree implementing library for java. The disadvantage of using KDTree.jar is the huge amount of used system memory, occupied by the music space, but also the reloading process that has to run every time the server is restarted. However there is also an advantage when saving the tree in memory: high access speed. Only the coordinates of a track are saved in the tree, together with the trackID and the track's popularity. The track's title and artist together with other information is crawled from the database at run time as this occupies too much memory.

Playlist Generation Algorithm

Existing Solutions

The base of the playlist generation algorithm is the music space containing data about similarity between tracks. The music space basically states that the smaller the distance between two tracks, the more similar they are. The topic of creating algorithms that use this property of the music space to generate playlists has been addressed by several ETH internal projects discussed in the Related Work Section.

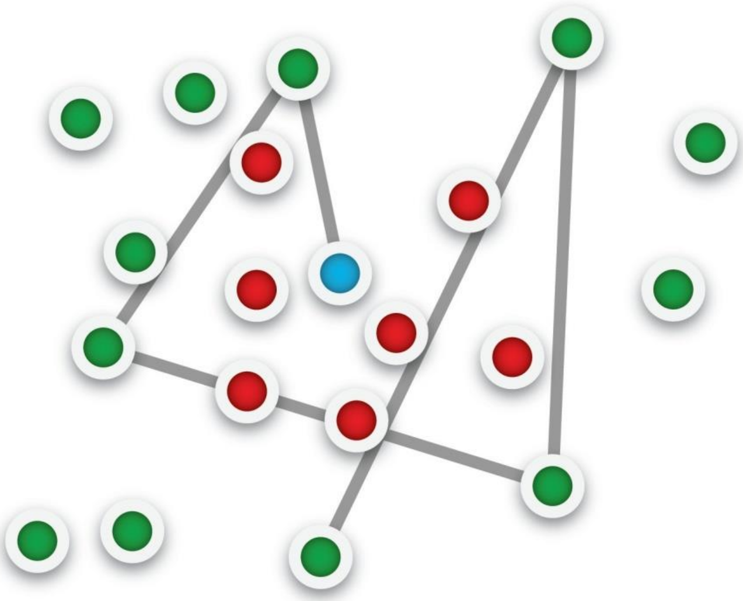
The existing musicexplorer web application allows the user, in a visual 2D way, to select two tracks representing the beginning and the end of his/her playlist. The algorithm then draws a line in the music space connecting the two tracks and adds tracks found in the vicinity of this line to the playlist such that in the end the user has a playlist that represents a smooth transition between the chosen tracks. Some problems arise from using this approach, because the algorithm never knows what two tracks the user selects. If he/she just states two of his/her favorite tracks, the distance between these two is very short in music space metric, not allowing the algorithm to find optimal transition tracks. A very high distance might make the transition not smooth enough. Optimizing in respect to this variable – distance – is a difficult task and it's even more difficult to prove its correctness. Also another downside is that the generated playlist has a fixed size and cannot be extended.

Also the Amarok2 plug-in (Känel, 2008) uses a similar way of computing playlists. The difference is that it does not take only two inputs but an indefinite number, such that the user can state several of his/her favorite tracks. First, these are reordered such that a minimal path is used to reach all of them. Next, the algorithm walks along the paths with a predefined step size and after each step it chooses the nearest track to that point in the space. After reaching the end destination, it randomly chooses a point in the graph and walks to that one such that the user can explore other types of music. The algorithm is intuitive and has several advantages; the user can insert several favorite tracks and implicitly helps the application to understand his/her music taste better. The random chosen point at the end of the walk encourages the user to explore new tracks but even more, it allows the generation of an 'infinitely long' playlist.

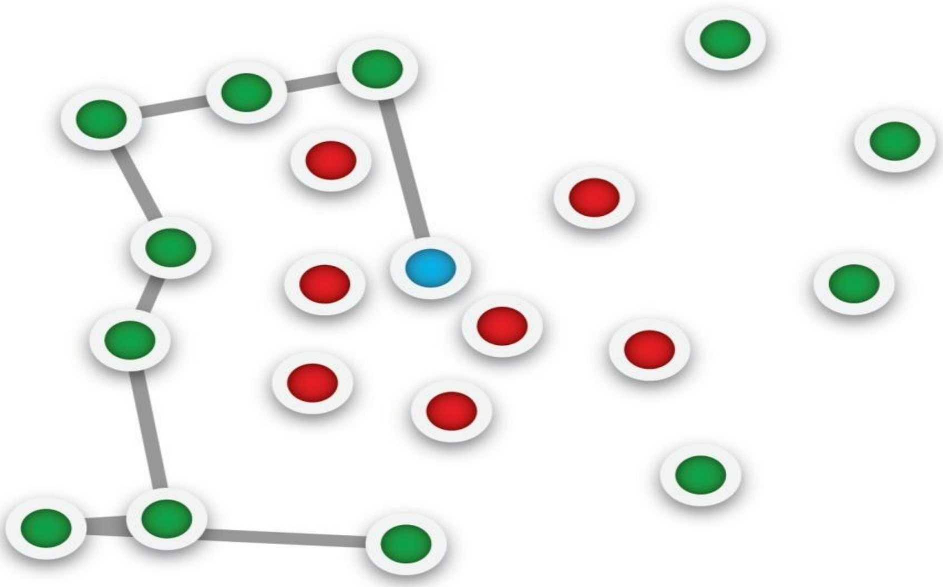
First Approach

Both the approaches presented in the previous section tend to exceed the simplicity that's set as a goal of this application. Also the user might be overwhelmed when asked to insert several tracks; he/she might just have one track in mind that he/she likes to listen to.

One first approach when dealing with only one seed track is to simply take the *k* closest tracks to the seed. Through this, the user is likely to like the generated list of tracks. An immediate problem is when trying to generate an 'infinitely long' playlist – the transition between the tracks is not very smooth after a while. Figure 6-2 shows succession of tracks in a playlist after a certain number of tracks have been already played. You can see how the algorithm makes very big jumps to get the track that's nearest to the seed. Note that red tracks are ones that have already been picked by the algorithm previously and cannot be chosen anymore.



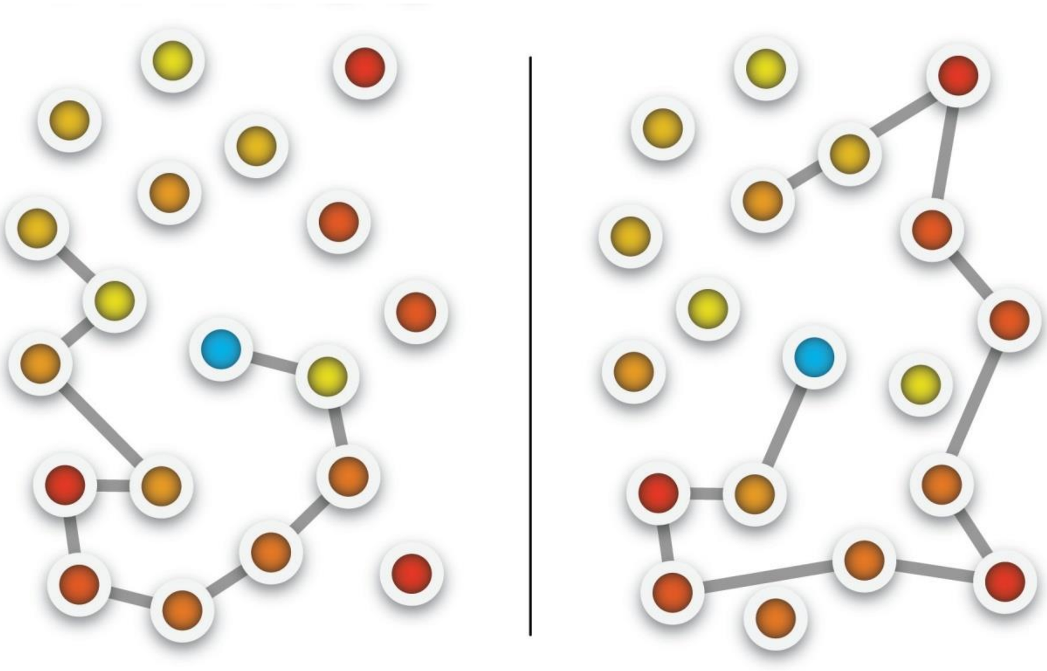
A modification of this algorithm might bring the solution to this problem: reassigning the seed for each iteration (Figure 6-3). The playlist generating algorithm is a successive iteration of an algorithm that chooses the nearest track to the seed, followed by setting the newly found track as seed for the next iteration. In this way the application avoids as much as possible jumps over long distances in music space metric. In the same scenario as the last example, the modified algorithm performs without big jumps.



The modified algorithm also has other advantages: the distance between the chosen tracks is always very short, guaranteeing a smooth transition. Not only can the algorithm provide an infinite number of tracks, but also each track is guaranteed to be very similar to the previous one. The random walking behavior of the Amarok algorithm (Känel, 2008) is also simulated, allowing the user to randomly explore unknown regions of the music space.

Improvement

Choosing the nearest track to the current seed provides results equally distributed regarding the popularity of the tracks. But the probability of users enjoying the returned tracks is greater if the popularity of the returned track is maximized. An improvement suggestion is therefore not to return the track nearest to the seed, but the track in the vicinity of the node that has the highest popularity. To find it, the algorithm queries for the first 10 tracks in its vicinity and takes the one with the maximum popularity. Figure 6-4 shows a playlist- computing example with and without this enhancement.



The example in Figure 6-4 shows the algorithm performing with or without vicinity popularity search (the vicinity consists of three tracks). On the left the algorithm selects much less popular tracks in the first 10 than the one on the right; but all this comes at a cost: the step size of the right algorithm is no more than three times greater (on average) than the one on the left.

Because songs of the same artist are usually very close together in the music space, the application tends to select several songs of the same artist and place them successively in the playlist. This is an undesired behavior. To avoid it the algorithm can be taught to also avoid tracks whose artist has appeared in the last 4 tracks. By applying this additional constraint, the algorithm avoids placing tracks of the same artist less then 4 tracks apart.

When searching for tracks in the vicinity of a seed, tracks that have already been added to this playlist in the past are omitted – as a rule. Doing so, if the algorithm was not able to ‘move’ to a different region in the music space it starts picking the lower popularity tracks. So after 24 tracks the algorithm can start adding tracks that were present 24 tracks ago. Why 24? 24 tracks with an average of 5 min/track are almost two hours of music – enough time to re-enjoy the tracks.

User Feedback

The importance of user feedback (both indirect and direct) has been discussed throughout several sections of this report. Most important when designing the feedback algorithm was the decision what data to store so that it is of use to any future evaluation.

Direct and Indirect User Feedback

Direct user feedback is represented by the feedback the user chooses to submitto the application in order to help the community. Currently two types of direct feedback are implemented: ‘Quality is bad’ and ‘I don’t like the track’. Because

the application doesn’t directly react to feedback (yet) this feedback is stored for a later processing. This direct feedback shows for example learn how many users actually used the feedback algorithm.

The users should hit ‘Quality is bad’ to signal that a YouTube video of a track has bad quality or doesn’t show the track at all. In a future version, the application immediately reacts to this feedback and propose a different track or, even better, let the user propose a better track by presenting a list of alternatives.

The ‘I don’t like this track’ is intended for users notifying the algorithm that the chosen track was not chosen right by the playlist generation algorithm, because this algorithm should only generate tracks that the user likes.

Indirect user feedback is feedback recorded and reported by the client application and without the knowledge of the user. It’s important to log also the behavior and commands of the users to be able to figure out algorithm and application flaws. For example, if a user listens to the first 5 seconds of the first three tracks, but listens the whole fourth track, it means that the first three tracks were not a good choice for this user.

Activity and Feedback Log Structure

Because most computations of the application are done per session basis, it is good to keep track of the user's sessions and their activity in these sessions. That is why the application uses a table called tlbFeedbackSessions to save information about the session. This information includes the sessionID of the session – which at the same time is the date and time the session was first started stated in milliseconds since Jan 1st, 1970 – and the IP address of the user to compute a user-country statistic and to help user-tracking over several sessions.

All commands issued by the users are saved in a table called tblFeedbackCommands. Most of the commands issued by the user (as direct or indirect feedback) but also commands issued automatically by the application (like a track finishing playback) are recorded together with their time of issue – at the same time being a commandID, the associated sessionID, the operation code and the trackID upon which the command was executed. The list of operation codes with their explanation is found in Table 5-1 .

Operation Code	Command	Track ID
-1*	Report bad video	
-2*	Report bad choice	
0	A track reached the end and the next is played	The next trackID
1	Manually select a track to play	
2	Use one of the tracks as seed	
3	Manually remove a track	
4	Track lifted to be moved**	
5	Track put down after move**	trackID before which the dragged track was put down, or -1 if it was put down as last track in the playlist
6	Toggle repeat off	
7	Toggle repeat on	

* Negative operation codes show direct user feedback, while positive codes show indirect user feedback.

** These two commands are supported but not implemented on the client side application. Hence, if a track is moved no command trigger is issued to theserver.

*** Commands having codes of the form 10* are not issued by the client side application, but by the service itself, when the user requests new tracks forexample

The table tblFeedbackTracks holds some special case data. When the user issues a generate playlist command, the service logs that command in tblFeedbackCommands but also saves the tracks returned to the user as result of his/her request in the tblFeedbackTracks table. Each track is saved with the following fields:

- commandsID – representing the commandID whose result generated this trackID;
- the trackID;
- the position, representing the position of the track in the returned array + videoID.

The latter is important for example for evaluating commands like 1 – Manually select a track to play. If the user did manually select a track to play, it could mean this track is one of the users favorites so the algorithm can see where in the list it was and how the algorithm could be improved to have this track appear earlier in the playlist.

`

FEASIBILILTY ANALYSIS

Feasibility study is done so that an ill-conceived system is recognized early in definition phase. During system engineering, however, we concentrate our attention on four primary areas of interest. This phase is really important as before starting with the real work of building the system it was very important to find out whether the idea thought is possible or not.

- Economic Feasibility: An evaluation of development cost weighted against the ultimate income or benefit derived from the developed system.

- Technical Feasibility: A study of function, performance and constraints that may affect the ability to achieve an acceptable system.

- Operational Feasibility: A study about the operational aspects of the system.

`

ECONOMIC ANALYSIS

Among the most important information contained in feasibility study is Cost Benefit Analysis and assessment of the economic justification for a computer based system project. Cost Benefit Analysis delineates costs for the project development and weighs them against tangible and intangible benefits of a system. Cost Benefits Analysis is complicated by the criteria that vary with the characteristics of the system to be developed, the relative size of the project and the expected return on investment desired as part of company’s strategic plan. In addition, many benefits derived from a computer-based system are intangible (e.g. better design quality through iterative optimization, increased customer satisfaction through programmable control etc.)As this is an in-house project for the company, to be used for its own convenience and also it is not that big a project. So neither it requires a huge amount of money nor any costly tools or infrastructure need to be set up for it.

According to the computerized system we propose, the costs can be broken down to two categories.

1. Costs associated with the development of the system.
2. Costs associated with operating the system.

`

TECHNICAL ANALYSIS

During technical analysis, the technical merits of the system are studied and at the same time collecting additional information about performance, reliability, maintainability and predictability.

Technical analysis begins with an assessment of the technical viability of the proposed system.

- What technologies are required to accomplished system function and performance?
- What new materials, methods, algorithms or processes are required and what is their development risk?
- How will these obtained from technical analysis form the basis for another go/no-go decision on the test system.

As the software is vary much economically feasible, then it is really important for it to be technically sound. The software will be build among:

- MS SQL SERVER as Back End
- ASP.NET as Front End

Operational Feasibility

The project is operationally feasible. This project is being made for the convenience of the patients and doctors only. This system will greatly reduce a huge burden of doctors. So because of the above stated advantages the users of the system will not be reluctant at all. A proposed system is beneficial only if it can be turned into an information system that will meet the operational requirements of an organization. A system often fails if it does not fit within existing operations and if users resist the change.

`

SYSTEM ANALYSIS

INTRODUCTION:

System analysis is the process of studying the business processors and procedures, generally referred to as business systems, to see how they can operate and whether improvement is needed.
This may involve examining data movement and storage, machines and technology used in the system, programs that control the machines, people providing inputs, doing the processing and receiving the outputs.

INVESTIGATION PHASE

The investigation phase is also known as the fact-finding stage or the analysis of the current system. This is a detailed study conducted with the purpose of wanting to fully understand the existing system and to identify the basic information requirements. Various techniques may be used in fact-finding and all fact obtained must be recorded.
A thorough investigation was done in every effected aspect when determining whether the purposed system is feasible enough to be implemented.

`

INVESTIGATION

As it was essential for us to find out more about the present system, we used the following methods to gather the information: -

1. Observation: - Necessary to see the way the system works first hand.

2 Document sampling: - These are all the documents that are used in the system. They are necessary to check all the data that enters and leaves the system.

3 Questionnaires: - These were conducted to get views of the other employees who are currently employed in the system.

ANALYSIS OF THE INVESTIGATION

Strengths of the System

1. No complex equipment: - The equipment that is used is very simple and no special skills have to be mastered to be able to operate the system. Therefore no training is required for the employees.

2. Low cost: - There is little money spent in maintaining the present system other than buying the necessary office equipment and the ledgers.

`

SELECTED SOFTWARE

WHAT IS HTML?

To publish information for global distribution, one needs a university-understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML (Hyper Text Markup Language)

HTML Gives Authors the Means To

1. Publish online documents with headings, text, tables, list, photos etc.
2. Retrieve online information via hypertext links, at the click of a button
3. Design forms for conducting transactions with remote services, for use in searching information, making reservation, ordering products etc.;
4. Includes spreadsheets, video clips, sound clips, and other applications directly in the documents.

HTML 4.0 extends with mechanisms for style sheets, scripting, frames embedding objects, improved support for right to left and mixed direction texts, richer tables and enhancements to form, offering improved accessibilities for people with disability

WHAT IS JAVA SCRIPT?

JavaScript, originally supported by Netscape Navigator, is the most popular Web scripting language today. JavaScript lets you embed programs right in your Web pages and run these programs using the Web browser. You place these programs in a <SCRIPT> element. If you want the script to write directly to the Web page, place it in the <BODY> element.

JavaScript is an object-oriented language. JavaScript comes with a number of predefined objects.

`

Objects of the JavaScript

1. Document: Corresponds to the current Web page’s body. Using this object, you have access to the HTML of the page itself, including the all links, images and anchors in it.
2. Form: Holds information about HTML forms in the current page.
3. Frame: Refers to a frame in the browser’s window.
4. History: Holds the records of sites the Web browser has visited before reaching the current page.
5. Location: Holds information about the location of the current web page.
6. Navigator: Refers to the browser itself, letting you determine what browser the user has.
7. Window: Refers to the current browser window.

The .NET Framework is a new computing platform that simplifies application development in the highly distributed environment of the Internet. The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. ASP.NET is a programming framework built on the common language runtime that can be used on a server to build powerful Web applications. The ASP.NET Web Forms page framework is a scalable common language runtime programming model that can be used on the server to dynamically generate Web pages.

DOT NET DATABASE CONNECTIVITY (ADO.NET)

- ADO.NET uses a multilayered architecture that revolves around a few key concepts, such as Connection, Command, and Dataset objects. However, the ADO.NET architecture is quite a bit different from classic ADO.
- One of the key differences between ADO and ADO.NET is how they deal with the challenge of different data sources. In ADO, programmers always use a generic set of objects, no matter what the underlying data source is.

`

.RDBMS CONCEPTS

1. DATA ABSTRACTION

A major purpose of a database system is to provide users with an abstract view of the data. This system hides certain details of how the data is stored and maintained. However in order for the system to be usable, data must be retrieved efficiently. The efficiency lead to the design of complex data structure for the representation of data in the database. Certain complexity must be hidden from the database system users. This accomplished by defining several levels of abstraction at which the database may be viewed.

2. CLASSIFICATION OF DATABASE

There are 3 types of database approaches given below,

a) Hierarchical Database:

In this type of model data is represented in simple tree structured. The record at the top of three is known as root, the root may have any number of dependents

b) Network Database:

In a Network database, data is represented by Network structure. In this approach record occurrence can have any number of superiors as well as any number of immediate dependents thus allow many to many correspondence directly than an hierarchical approach.

c) Relational Database:

The Relational model represents data and relationships among data by a collection of tables each of which has a number of columns with unique names.

`

THE SQL LANGUAGE

SQL is a language for relational database. SQL is a non-procedural i.e., when we use SQL we specify what we want to be done not how to do it.

Features Of SQL

1. SQL is an interactive query language.
2. SQL is a database administration language.
3. SQL is a database programming language.
4. SQL is a client/server language.
5. SQL is a distributed database language.
6. SQL is a database gateway language.

What is PHP?

PHP is used in the first form to generate web page content dynamically. For example, if you have a blog website, you might write some PHP scripts to retrieve your blog posts from a database and display them. Other uses for PHP scripts include: Processing and saving user input from form data.

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

`

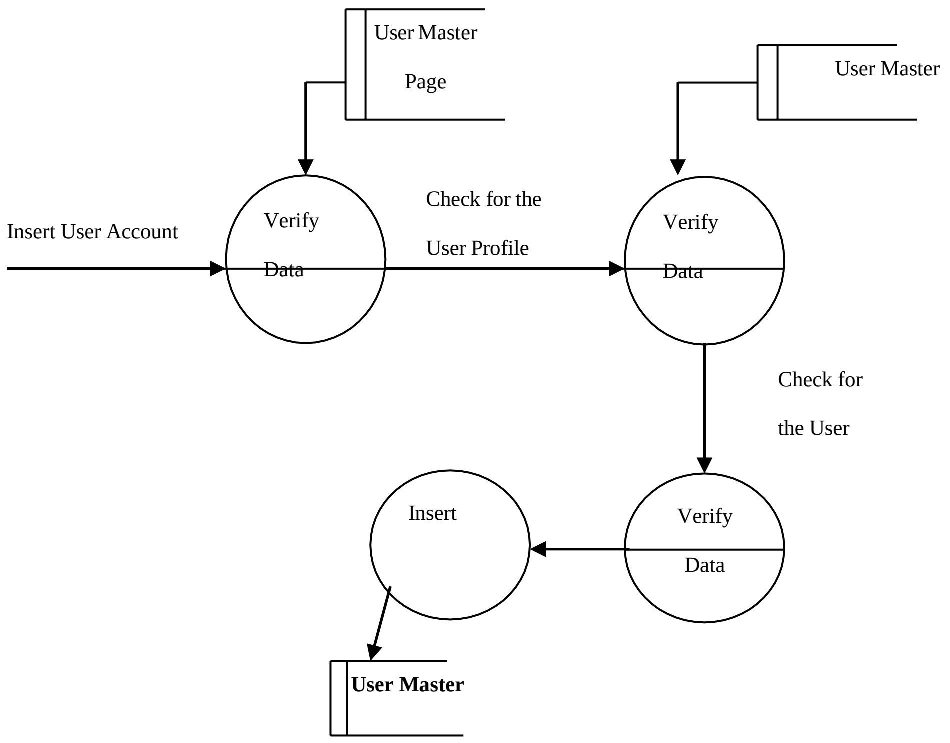
DATA FLOW DIAGRAM

Data flows are data structures in motion, while data stores are data structures. Data flows are paths or 'pipe lines', along which data structures travel, where as the data stores are place where data structures are kept until needed.

Data flows are data structures in motion, while data stores are data structures at rest. Hence it is possible that the data flow and the data store would be made up of the same data structure.

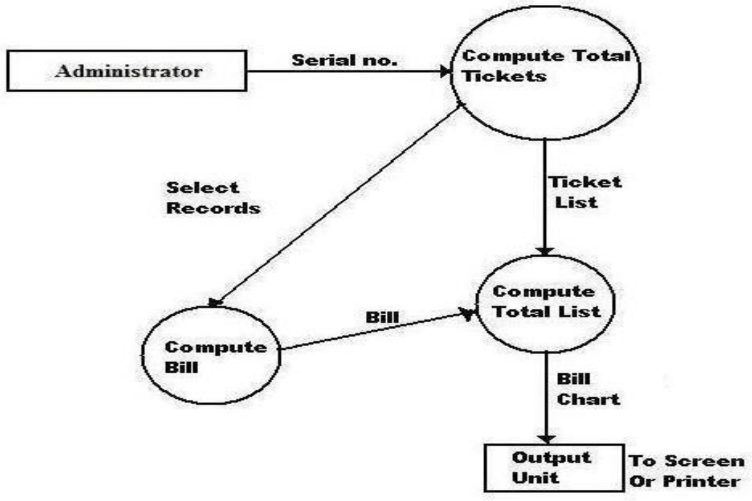
Data flow diagrams is a very handy tool for the system analyst because it gives the analyst the overall picture of the system, it is a diagrammatic approach.

DFD for User Account

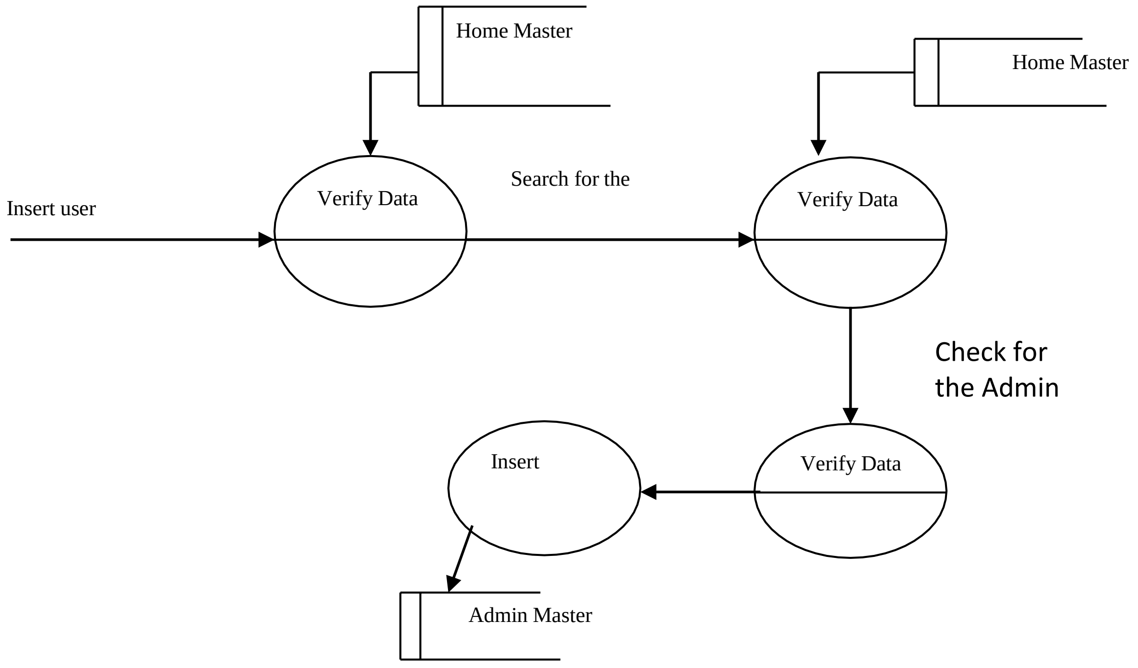


`

DFD for Playing



DFD of Modification:-



`

Codeing:-

Register page

```
<?php
	include("includes/config.php");
	include("includes/classes/Account.php");
	include("includes/classes/Constants.php");

	$account = new Account($con);

	include("includes/handlers/register-handler.php");
	include("includes/handlers/login-handler.php");

	function getInputValue($name) {
		if(isset($_POST[$name])) {
			echo $_POST[$name];
		}
	}
?>

<html>
<head>
	<title>Welcome to Spotify Clone!</title>

	<link rel="stylesheet" type="text/css" href="assets/css/register.css">

	<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
	<script src="assets/js/register.js"></script>
</head>
<body>
	<?php

	if(isset($_POST['registerButton'])) {
		echo '<script>
				$(document).ready(function() {
					$("#loginForm").hide();
					$("#registerForm").show();
				});
			</script>';
	}
	else {
		echo '<script>
				$(document).ready(function() {
					$("#loginForm").show();
					$("#registerForm").hide();
				});
			</script>';
	}

	?>
```

```
`

	<div id="background">

		<div id="loginContainer">

			<div id="inputContainer">
				<form id="loginForm" action="register.php" method="POST">
					<h2>Login to your account</h2>
					<p>
						<?php echo $account->getError(Constants::$loginFailed); ?>
						<label for="loginUsername">Username</label>
						<input id="loginUsername" name="loginUsername" type="text"
placeholder="e.g. bartSimpson" value="<?php getInputValue('loginUsername') ?>" required autocomplete="off">
					</p>
					<p>
						<label for="loginPassword">Password</label>
						<input id="loginPassword" name="loginPassword" type="password"
placeholder="Your password" required>
					</p>

					<button type="submit" name="loginButton">LOG IN</button>

					<div class="hasAccountText">
						<span id="hideLogin">Don't have an account yet? Signup here.</span>
					</div>

				</form>



				<form id="registerForm" action="register.php" method="POST">
					<h2>Create your free account</h2>
					<p>
						<?php echo $account->getError(Constants::$usernameCharacters); ?>
						<?php echo $account->getError(Constants::$usernameTaken); ?>
						<label for="username">Username</label>
						<input id="username" name="username" type="text" placeholder="e.g.
bartSimpson" value="<?php getInputValue('username') ?>" required>
					</p>

					<p>
						<?php echo $account->getError(Constants::$firstNameCharacters); ?>
						<label for="firstName">First name</label>
						<input id="firstName" name="firstName" type="text" placeholder="e.g.
Bart" value="<?php getInputValue('firstName') ?>" required>
					</p>

					<p>
						<?php echo $account->getError(Constants::$lastNameCharacters); ?>
						<label for="lastName">Last name</label>
						<input id="lastName" name="lastName" type="text" placeholder="e.g.
Simpson" value="<?php getInputValue('lastName') ?>" required>
```

```
`
                                        </p>

                                        <p>
                                                <?php echo $account->getError(Constants::$emailsDoNotMatch); ?>
                                                <?php echo $account->getError(Constants::$emailInvalid); ?>
                                                <?php echo $account->getError(Constants::$emailTaken); ?>
                                                <label for="email">Email</label>
                                                <input id="email" name="email" type="email" placeholder="e.g.
bart@gmail.com" value="<?php getInputValue('email') ?>" required>
                                        </p>

                                        <p>
                                                <label for="email2">Confirm email</label>
                                                <input id="email2" name="email2" type="email" placeholder="e.g.
bart@gmail.com" value="<?php getInputValue('email2') ?>" required>
                                        </p>

                                        <p>
                                                <?php echo $account->getError(Constants::$passwordsDoNoMatch); ?>
                                                <?php echo $account-
>getError(Constants::$passwordNotAlphanumeric); ?>
                                                <?php echo $account->getError(Constants::$passwordCharacters); ?>
                                                <label for="password">Password</label>
                                                <input id="password" name="password" type="password"
placeholder="Your password" required>
                                        </p>

                                        <p>
                                                <label for="password2">Confirm password</label>
                                                <input id="password2" name="password2" type="password"
placeholder="Your password" required>
                                        </p>

                                        <button type="submit" name="registerButton">SIGN UP</button>

                                        <div class="hasAccountText">
                                                <span id="hideRegister">Already have an account? Log in here.</span>
                                        </div>

                                </form>


                        </div>

                        <div id="loginText">
                                <h1>Get great music, right now</h1>
                                <h2>Listen to loads of songs for free</h2>
                                <ul>
                                        <li>Discover music you'll fall in love with</li>
                                        <li>Create your own playlists</li>
                                        <li>Follow artists to keep up to date</li>
                                </ul>
```

```
   `
                              </div>

                    </div>
          </div>

</body>
</html>
```

Account page:-

```
<?php
          class Account {

                    private $con;
                    private $errorArray;

                    public function __construct($con) {
                              $this->con = $con;
                              $this->errorArray = array();
                    }

                    public function login($un, $pw) {

                              $pw = md5($pw);

                              $query = mysqli_query($this->con, "SELECT * FROM users WHERE username='$un' AND
password='$pw'");

                              if(mysqli_num_rows($query) == 1) {
                                        return true;
                              }
                              else {
                                        array_push($this->errorArray, Constants::$loginFailed);
                                        return false;
                              }

                    }

                    public function register($un, $fn, $ln, $em, $em2, $pw, $pw2) {
                              $this->validateUsername($un);
                              $this->validateFirstName($fn);
                              $this->validateLastName($ln);
                              $this->validateEmails($em, $em2);
                              $this->validatePasswords($pw, $pw2);

                              if(empty($this->errorArray) == true) {
                                        //Insert into db
                                        return $this->insertUserDetails($un, $fn, $ln, $em, $pw);
                              }
                              else {
                                        return false;
                              }
```

```
`

                }

                public function getError($error) {
                        if(!in_array($error, $this->errorArray)) {
                                $error = "";
                        }
                        return "<span class='errorMessage'>$error</span>";
                }

                private function insertUserDetails($un, $fn, $ln, $em, $pw) {
                        $encryptedPw = md5($pw);
                        $profilePic = "assets/images/profile-pics/head_emerald.png";
                        $date = date("Y-m-d");

                        $result = mysqli_query($this->con, "INSERT INTO users VALUES ('', '$un', '$fn', '$ln', '$em',
'$encryptedPw', '$date', '$profilePic')");

                        return $result;
                }

                private function validateUsername($un) {

                        if(strlen($un) > 25 || strlen($un) < 5) {
                                array_push($this->errorArray, Constants::$usernameCharacters);
                                return;
                        }

                        $checkUsernameQuery = mysqli_query($this->con, "SELECT username FROM users
WHERE username='$un'");
                        if(mysqli_num_rows($checkUsernameQuery) != 0) {
                                array_push($this->errorArray, Constants::$usernameTaken);
                                return;
                        }

                }

                private function validateFirstName($fn) {
                        if(strlen($fn) > 25 || strlen($fn) < 2) {
                                array_push($this->errorArray, Constants::$firstNameCharacters);
                                return;
                        }
                }

                private function validateLastName($ln) {
                        if(strlen($ln) > 25 || strlen($ln) < 2) {
                                array_push($this->errorArray, Constants::$lastNameCharacters);
                                return;
                        }
                }

                private function validateEmails($em, $em2) {
```

```
   `
                    if($em != $em2) {
                          array_push($this->errorArray, Constants::$emailsDoNotMatch);
                          return;
                    }

                    if(!filter_var($em, FILTER_VALIDATE_EMAIL)) {
                          array_push($this->errorArray, Constants::$emailInvalid);
                          return;
                    }

                    $checkEmailQuery = mysqli_query($this->con, "SELECT email FROM users WHERE
email='$em'");
                    if(mysqli_num_rows($checkEmailQuery) != 0) {
                          array_push($this->errorArray, Constants::$emailTaken);
                          return;
                    }

              }

              private function validatePasswords($pw, $pw2) {

                    if($pw != $pw2) {
                          array_push($this->errorArray, Constants::$passwordsDoNoMatch);
                          return;
                    }

                    if(preg_match('/[^A-Za-z0-9]/', $pw)) {
                          array_push($this->errorArray, Constants::$passwordNotAlphanumeric);
                          return;
                    }

                    if(strlen($pw) > 30 || strlen($pw) < 5) {
                          array_push($this->errorArray, Constants::$passwordCharacters);
                          return;
                    }

              }


        }
```

Playing page:-

```
<?php
$songQuery = mysqli_query($con, "SELECT id FROM songs ORDER BY RAND() LIMIT 10");

$resultArray = array();

while($row = mysqli_fetch_array($songQuery)) {
      array_push($resultArray, $row['id']);
}

$jsonArray = json_encode($resultArray);
```

```
    `
?>

<script>

$(document).ready(function() {
	var newPlaylist = <?php echo $jsonArray; ?>;
	audioElement = new Audio();
	setTrack(newPlaylist[0], newPlaylist, false);
	updateVolumeProgressBar(audioElement.audio);


	$("#nowPlayingBarContainer").on("mousedown touchstart mousemove touchmove", function(e) {
		e.preventDefault();
	});


	$(".playbackBar .progressBar").mousedown(function() {
		mouseDown = true;
	});

	$(".playbackBar .progressBar").mousemove(function(e) {
		if(mouseDown == true) {
			//Set time of song, depending on position of mouse
			timeFromOffset(e, this);
		}
	});

	$(".playbackBar .progressBar").mouseup(function(e) {
		timeFromOffset(e, this);
	});


	$(".volumeBar .progressBar").mousedown(function() {
		mouseDown = true;
	});

	$(".volumeBar .progressBar").mousemove(function(e) {
		if(mouseDown == true) {

			var percentage = e.offsetX / $(this).width();

			if(percentage >= 0 && percentage <= 1) {
				audioElement.audio.volume = percentage;
			}
		}
	});

	$(".volumeBar .progressBar").mouseup(function(e) {
		var percentage = e.offsetX / $(this).width();

		if(percentage >= 0 && percentage <= 1) {
			audioElement.audio.volume = percentage;
`
                }
        });

        $(document).mouseup(function() {
                mouseDown = false;
        });




});

function timeFromOffset(mouse, progressBar) {
        var percentage = mouse.offsetX / $(progressBar).width() * 100;
        var seconds = audioElement.audio.duration * (percentage / 100);
        audioElement.setTime(seconds);
}

function prevSong() {
        if(audioElement.audio.currentTime >= 3 || currentIndex == 0) {
                audioElement.setTime(0);
        }
        else {
                currentIndex = currentIndex - 1;
                setTrack(currentPlaylist[currentIndex], currentPlaylist, true);
        }
}

function nextSong() {
        if(repeat == true) {
                audioElement.setTime(0);
                playSong();
                return;
        }

        if(currentIndex == currentPlaylist.length - 1) {
                currentIndex = 0;
        }
        else {
                currentIndex++;
        }

        var trackToPlay = shuffle ? shufflePlaylist[currentIndex] : currentPlaylist[currentIndex];
        setTrack(trackToPlay, currentPlaylist, true);
}

function setRepeat() {
        repeat = !repeat;
        var imageName = repeat ? "repeat-active.png" : "repeat.png";
        $(".controlButton.repeat img").attr("src", "assets/images/icons/" + imageName);
}
```

```
   `
function setMute() {
	audioElement.audio.muted = !audioElement.audio.muted;
	var imageName = audioElement.audio.muted ? "volume-mute.png" : "volume.png";
	$(".controlButton.volume img").attr("src", "assets/images/icons/" + imageName);
}

function setShuffle() {
	shuffle = !shuffle;
	var imageName = shuffle ? "shuffle-active.png" : "shuffle.png";
	$(".controlButton.shuffle img").attr("src", "assets/images/icons/" + imageName);

	if(shuffle == true) {
		//Randomize playlist
		shuffleArray(shufflePlaylist);
		currentIndex = shufflePlaylist.indexOf(audioElement.currentlyPlaying.id);
	}
	else {
		//shuffle has been deactivated
		//go back to regular playlist
		currentIndex = currentPlaylist.indexOf(audioElement.currentlyPlaying.id);
	}

}

function shuffleArray(a) {
    var j, x, i;
    for (i = a.length; i; i--) {
        j = Math.floor(Math.random() * i);
        x = a[i - 1];
        a[i - 1] = a[j];
        a[j] = x;
    }
}


function setTrack(trackId, newPlaylist, play) {

	if(newPlaylist != currentPlaylist) {
		currentPlaylist = newPlaylist;
		shufflePlaylist = currentPlaylist.slice();
		shuffleArray(shufflePlaylist);
	}

	if(shuffle == true) {
		currentIndex = shufflePlaylist.indexOf(trackId);
	}
	else {
		currentIndex = currentPlaylist.indexOf(trackId);
	}
	pauseSong();

	$.post("includes/handlers/ajax/getSongJson.php", { songId: trackId }, function(data) {
```

```
`

                var track = JSON.parse(data);
                $(".trackName span").text(track.title);

                $.post("includes/handlers/ajax/getArtistJson.php", { artistId: track.artist }, function(data) {
                        var artist = JSON.parse(data);
                        $(".trackInfo .artistName span").text(artist.name);
                        $(".trackInfo .artistName span").attr("onclick", "openPage('artist.php?id=" + artist.id + "')");
                });

                $.post("includes/handlers/ajax/getAlbumJson.php", { albumId: track.album }, function(data) {
                        var album = JSON.parse(data);
                        $(".content .albumLink img").attr("src", album.artworkPath);
                        $(".content .albumLink img").attr("onclick", "openPage('album.php?id=" + album.id + "')");
                        $(".trackInfo .trackName span").attr("onclick", "openPage('album.php?id=" + album.id + "')");
                });


                audioElement.setTrack(track);

                if(play == true) {
                        playSong();
                }
        });

}

function playSong() {

        if(audioElement.audio.currentTime == 0) {
                $.post("includes/handlers/ajax/updatePlays.php", { songId: audioElement.currentlyPlaying.id });
        }

        $(".controlButton.play").hide();
        $(".controlButton.pause").show();
        audioElement.play();
}

function pauseSong() {
        $(".controlButton.play").show();
        $(".controlButton.pause").hide();
        audioElement.pause();
}
</script>


<div id="nowPlayingBarContainer">

        <div id="nowPlayingBar">

                <div id="nowPlayingLeft">
                        <div class="content">
```

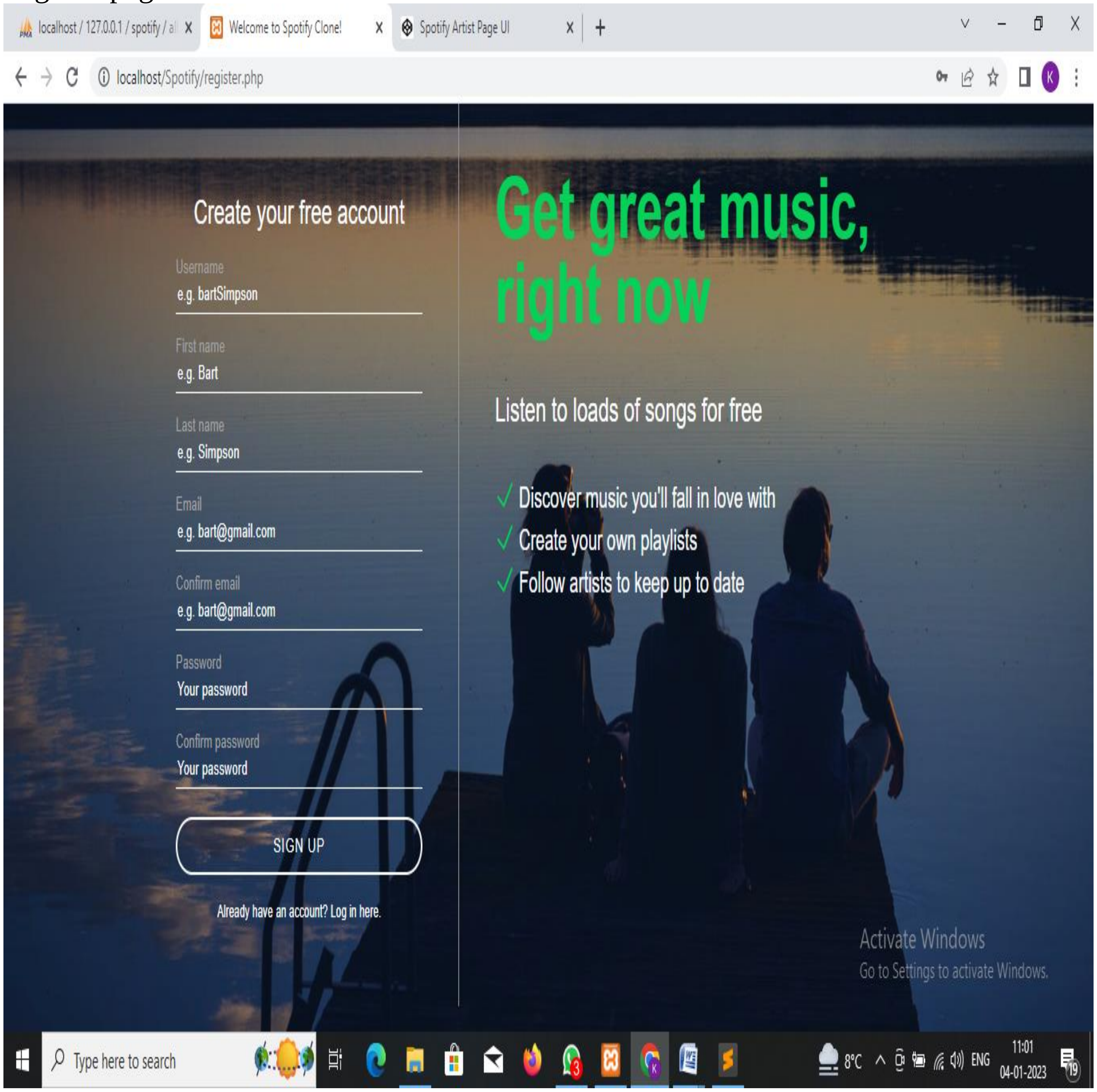
```
`
                                        <span class="albumLink">
                                                  <img role="link" tabindex="0" src="" class="albumArtwork">
                                        </span>

                                        <div class="trackInfo">

                                                  <span class="trackName">
                                                            <span role="link" tabindex="0"></span>
                                                  </span>

                                                  <span class="artistName">
                                                            <span role="link" tabindex="0"></span>
                                                  </span>

                                        </div>



                              </div>
                    </div>

                    <div id="nowPlayingCenter">

                              <div class="content playerControls">

                                        <div class="buttons">

                                                  <button class="controlButton shuffle" title="Shuffle button"
onclick="setShuffle()">
                                                            <img src="assets/images/icons/shuffle.png" alt="Shuffle">
                                                  </button>

                                                  <button class="controlButton previous" title="Previous button"
onclick="prevSong()">
                                                            <img src="assets/images/icons/previous.png" alt="Previous">
                                                  </button>

                                                  <button class="controlButton play" title="Play button" onclick="playSong()">
                                                            <img src="assets/images/icons/play.png" alt="Play">
                                                  </button>

                                                  <button class="controlButton pause" title="Pause button" style="display: none;"
onclick="pauseSong()">
                                                            <img src="assets/images/icons/pause.png" alt="Pause">
                                                  </button>

                                                  <button class="controlButton next" title="Next button" onclick="nextSong()">
                                                            <img src="assets/images/icons/next.png" alt="Next">
                                                  </button>

                                                  <button class="controlButton repeat" title="Repeat button"
onclick="setRepeat()">
```

```
  `
                        <img src="assets/images/icons/repeat.png" alt="Repeat">
                    </button>

                </div>


                <div class="playbackBar">

                    <span class="progressTime current">0.00</span>

                    <div class="progressBar">
                        <div class="progressBarBg">
                            <div class="progress"></div>
                        </div>
                    </div>

                    <span class="progressTime remaining">0.00</span>


                </div>


            </div>


        </div>

        <div id="nowPlayingRight">
            <div class="volumeBar">

                <button class="controlButton volume" title="Volume button" onclick="setMute()">
                    <img src="assets/images/icons/volume.png" alt="Volume">
                </button>

                <div class="progressBar">
                    <div class="progressBarBg">
                        <div class="progress"></div>
                    </div>
                </div>

            </div>
        </div>




    </div>

</div>
```

`

Output:-

Register page:-


localhost / 127.0.0.1 / spotify / al
Welcome to Spotify Clone!
Spotify Artist Page UI
localhost/Spotify/register.php
Type here to search
8°C
ENG
11:01
04-01-2023
Activate Windows
Go to Settings to activate Windows.

Create your free account

Username
e.g. bartSimpson

First name
e.g. Bart

Last name
e.g. Simpson

Email
e.g. bart@gmail.com

Confirm email
e.g. bart@gmail.com

Password
Your password

Confirm password
Your password

SIGN UP

Already have an account? Log in here.

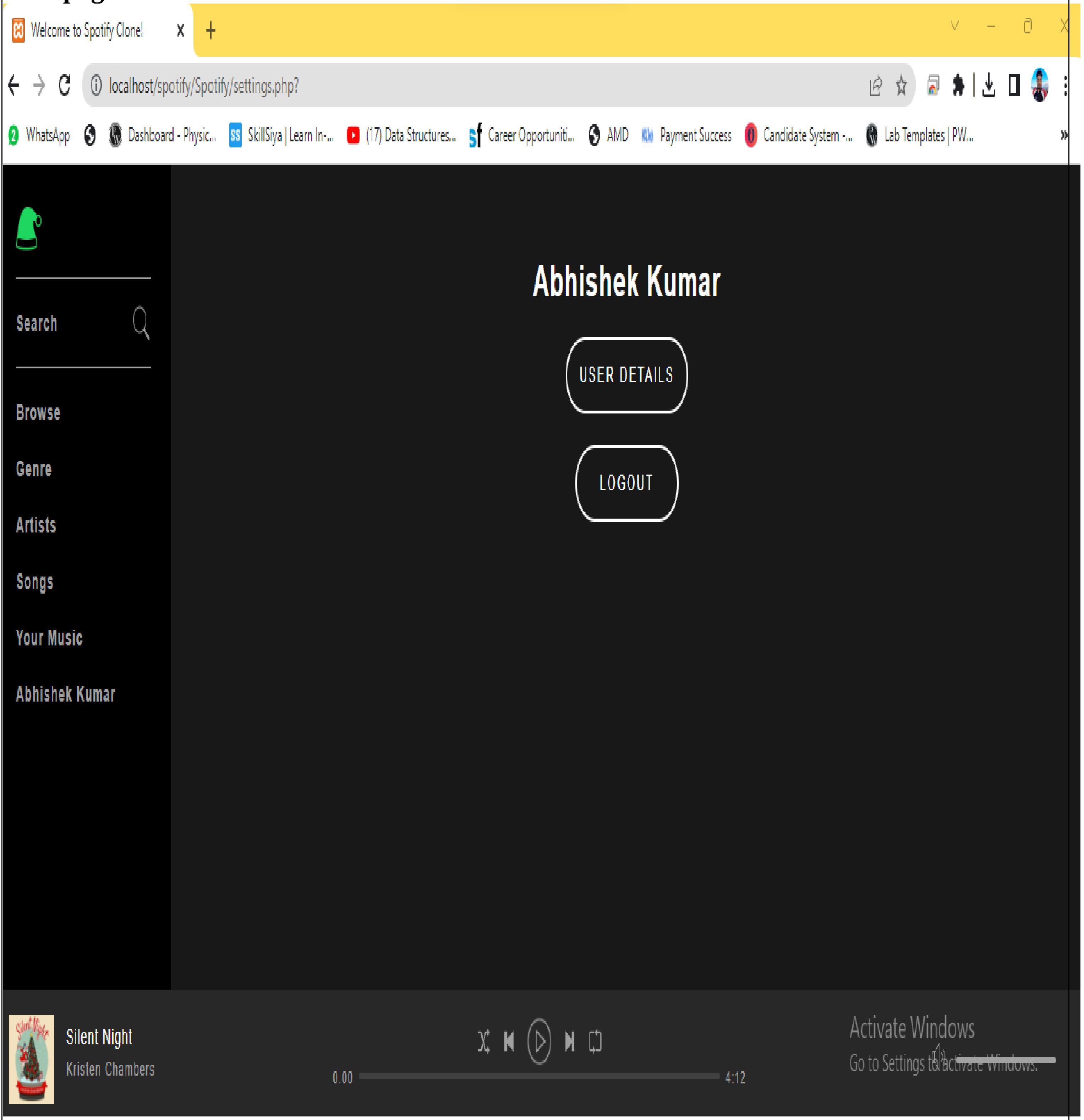
Get great music, right now

Listen to loads of songs for free

✓ Discover music you'll fall in love with
✓ Create your own playlists
✓ Follow artists to keep up to date

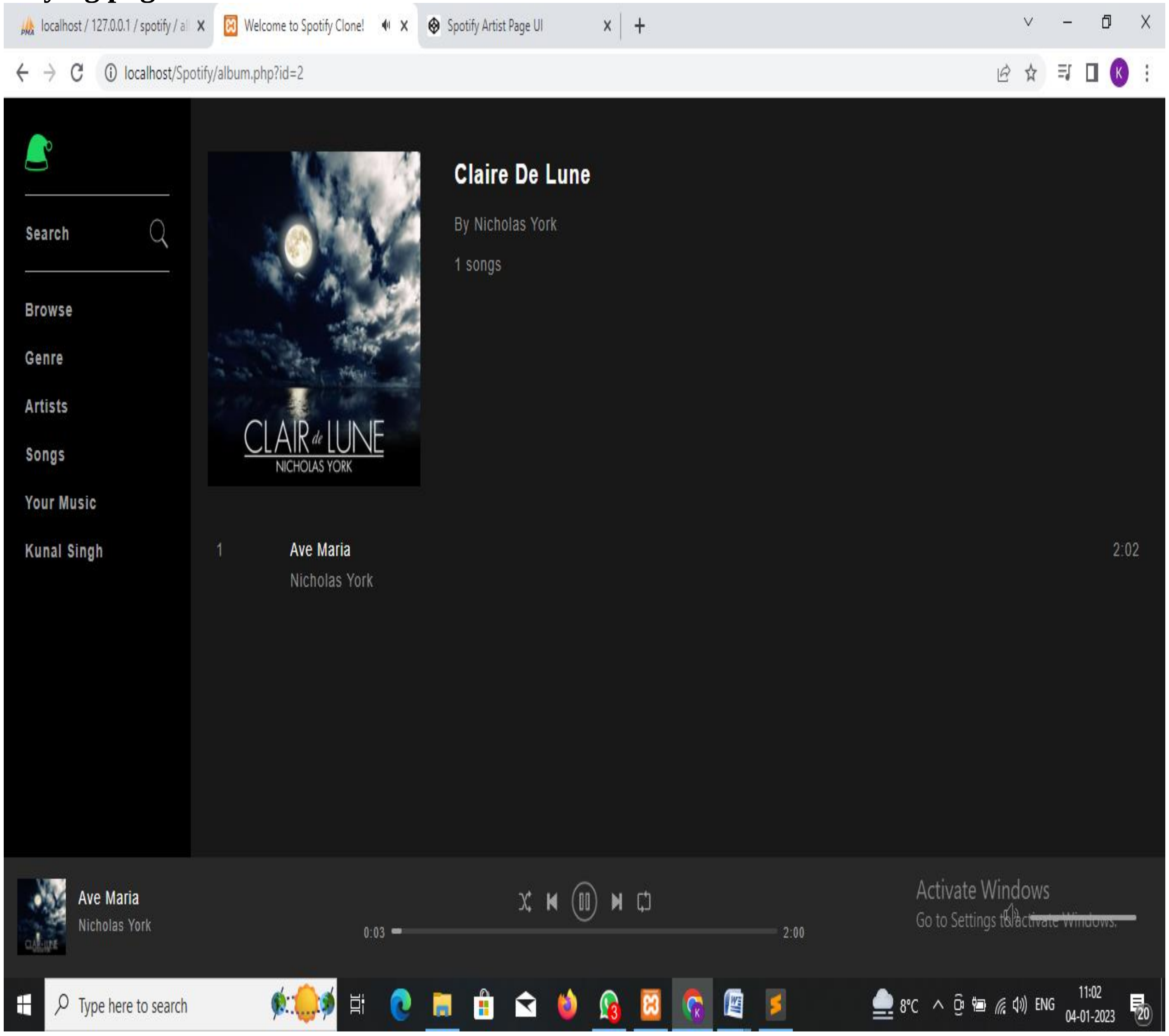
`

User page:-


Welcome to Spotify Clone!
localhost/spotify/Spotify/settings.php?
WhatsApp
Dashboard - Physic...
SkillSiya | Learn In-...
(17) Data Structures...
Career Opportuniti...
AMD
Payment Success
Candidate System -...
Lab Templates | PW...
Search
Browse
Genre
Artists
Songs
Your Music
Abhishek Kumar
Abhishek Kumar
USER DETAILS
LOGOUT
Silent Night
Kristen Chambers
0.00
4:12
Activate Windows
Go to Settings to activate Windows.

`

Playing page:-



localhost / 127.0.0.1 / spotify / al

Welcome to Spotify Clone!

Spotify Artist Page UI

localhost/Spotify/album.php?id=2

Search

Browse

Genre

Artists

Songs

Your Music

Kunal Singh

Claire De Lune

By Nicholas York

1 songs

1	Ave Maria Nicholas York	2:02

Ave Maria
Nicholas York

0:03 2:00

Activate Windows
Go to Settings to activate Windows.

Type here to search

8°C ENG 11:02 04-01-2023