Spotify Clone – A Music Player

A Project Report

submitted in partial fulfillment of the requirements

of

Applied Cloud Computing for

Software Development

by

ANKIT TRIPATHI, 2101610109005

SONAM SRIVASTAVA, 2101610109025

REETIKA SRIVASTAVA, 2001610100165

EKATA RAI, 2001610100082

Under the Esteemed Guidance of

Mr. Hrishikesh Mahure

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my training course Mentor, **Mr. Hrishikesh Mahure**, for their invaluable mentorship and guidance throughout my training of Applied Cloud Computing for Software Development.Their expertise and support played a significant role in shaping my professional growth and development during this period. I am truly grateful for their continuous guidance, encouragement, and willingness to share their knowledge and expertise.

Their insightful feedback and constructive criticism have helped me refine my skills and enhance my understanding of the industry. Their mentorship went beyond the technical aspects of the training and encompassed valuable advice on career advancement, networking, and personal development. Their patience, approachability, and commitment to my growth have made a lasting impact on my professional journey.

I would also like to extend my gratitude to the entire team at EDUNET Foundation for creating a conducive learning environment. The collaborative and supportive culture within the organization fostered a sense of belonging and allowed me to thrive in my role. The collective knowledge and experiences shared by my colleagues have been instrumental in my professional development. I am grateful for their willingness to answer my questions, offer guidance, and provide valuable insights throughout my internship.

Thank You!!

Regards

Ankit Tripathi

Sonam Srivastava

Reetika Srivastava

Ekta Rai

ABSTRACT

The Spotify clone is a web-based music player created using HTML, CSS, and JavaScript. The user interface replicates the sleek design of Spotify, offering an immersive experience for music enthusiasts. The HTML structure defines the layout, while CSS stylizes the elements to mirror Spotify's aesthetic. JavaScript handles dynamic functionalities, allowing users to play, pause, skip tracks, and adjust volume seamlessly. The clone utilizes APIs for fetching and displaying music data, ensuring a diverse and up-to-date library. Overall, this project aims to provide a user-friendly and visually appealing platform for music streaming enthusiasts, emulating the core features of the popular Spotify application. Java, extends its functionality to include user authentication through a robust login and registration system. The project integrates Java for server-side processing, enabling secure and seamless user interactions. The login and registration features provide users with personalized experiences, allowing them to create accounts, log in securely, and access personalized playlists. Java facilitates user authentication, ensuring the protection of sensitive information. The website, combining the aesthetic appeal of Spotify with Java-powered authentication, aims to deliver comprehensive music streaming experience, prioritizing both user convenience and security.

Chapter-1

1. INTRODUCTION

Welcome to “Spotify Clone - A Music Player”. This is the first module in the series we will see “What is Music Player and how does it work”. Music Player is a digital music, podcast and video streaming services that gives you access to millions of songs from artists all over the world, like other music streaming platform for e.g. Youtube Music, Jio Savaan, Music Mania, Retro music, etc.

Music Player is immediately appealing because you can access content for free by simply signing up using an Email address or by connecting with Facebook, Gmail Account. If you’re not keen on monthly subscription fees for Music Mania Premium, or just want to dip your toe in and test it out, it’s out, it’s easy to get started and there’s no commitment.

You can find out the main differences between Music Player Free and Premium in our separate feature but as a quick summary, the free version is ad-supported, much like radio stations. The free version of Music Mania can be accessed on PC, laptop and mobile phone, but the full service needs a Music Mania Premium subscription.

1.1 BACKGROUND

Sounds are all around us, from birds chirping and waves lapping against a coastline to cars honking in traffic. But sometimes sounds are put together in purposeful ways to create a specific atmosphere or to express ideas or emotions. Such organized sounds are called music.

Music is a collection of coordinated sound or sounds. Making music is the process of putting sounds and tones in an order, often combining them to create a unified composition. People who make music creatively organize sounds for a desired result, like a Beethoven symphony or one of Duke Ellington's jazz songs. Music is made of sounds, vibrations, and silent moments, and it doesn't always have to be pleasant or pretty. It can be used to convey a whole range of experiences, environments, and emotions.

Almost every human culture has a tradition of making music. Examples of early instruments like flutes and drums have been found dating back thousands of years. Ancient Egyptians used music in religious ceremonies. Many other African cultures have traditions related to drumming for important rituals. Today, rock and pop musicians tour and perform around the world, singing the songs that made them famous. All of these are examples of music.

1.2 OBJECTIVE

When you have completed this module you will be able to-:

Basic functions such as playing music are totally free, but you can also choose to upgrade to Music Mania Premium. Either way, you can:

- Choose what you want to listen to with Browse and Search
  - Find what you're looking for with Search, including:

1.Songs
2.Albums
3.Artists
4.Playlist
5.Podcast shows and episodes

On mobile and tablet, you can also use Search to browse categories such as genres, moods charts, and new releases.

- Get recommendations from personalized features, such as Discover Weekly, Release Radar, and Daily Mix.
  - Find Made for you playlist in Home  .
  - Or Search the name of any playlist made for you.

- Build collections of music.
  - When you like  a song, playlist, album, or follow an artist or podcast, you can find it in Your Library.

- See what friends, artists, and celebrities listen to
  - Follow artists to receive notifications and never miss a new release.
    1. Go to the artist's profile. 2.Select Follow.
  - Follow friends to see what they're listening to in Friend Activity.

- ☐ Create your own Radio station.

- Keep the mood going. Music Mania Radio creates a collection of songs based on any artist, album, playlist, or song of your choice. It even updates over time to keep fresh.

    1. Go to any artist, album, playlist, or song.
    2. Select  or  .
    3. Select Go to radio.

1.3 PURPOSE

The purpose of this document is to inform user Music Mania is a digital streaming services that gives you access to millions of songs from artists all over the world.

Our mission to unlock the potential of human creativity by giving a million creative artists the opportunity to live off their art and billions of fans the opportunity to enjoy and be inspired by it.

Music Mania manage and share tracks, including podcast titles, for free, or upgrade to Music Mania Premium to access exclusive features for music including improved sound quality and an on-demand, offline listening experience.

1.4. SCOPE

With Music Mania, it’s easy to find the right music or podcast for every moment – on your phone, your computer, your tablet and more.

There are millions of tracks and episodes on Music Mania. So whether you’re behind the wheel, working out, partying or relaxing, the right music or podcast is always at your fingertips. Choose what you want to listen to, or let Music Mania surprise you.

You can also browse through the collections of friends, artists, and celebrities, or create a radio station and just sit back.

1.5 APPLICABILITY

The functions of playing music and multimedia have become essential in one device as a smart phone since the smart phone appeared.

It is very convenient, but it contains controversial arguments about sound quality, so many smart phone users use the music player application. By using these music applications, people start to think about the relationship between music playing and sound quality. However, those applications are not perfect, so it is hard to choose a good application.

This thesis is about the advantages of the sound quality of music player applications that are currently sold in Android Market through Right Mark Audio Analyzer program, and plans to suggest android music player application system design by analyzing applications by covering disadvantages of these applications.

Chapter-2

2.System Planning

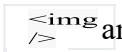
2.1.Survey of technologies

1.Front End-:

HTML

Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as  and  directly introduce content into the page. Other tags such as  surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name *cascading* comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents

React.JS-

ReactJS is JavaScript library used for building reusable UI components. According to React official documentation, following is the definition −

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

✓ React Features

- JSX − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
- Components − React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.
- Unidirectional data flow and Flux − React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
- License − React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

✓ React Advantages

- Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
- Can be used on client and server side as well as with other frameworks.
- Component and data patterns improve readability, which helps to maintain larger apps.

- **React Limitations**

- Covers only the view layer of the app, hence you still need to choose other technologies to get a complete tooling set for development.
- Uses inline templating and JSX, which might seem awkward to some developers.

Language used JavaScript-

JavaScript **(**JS**)** is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) style.

JavaScript runs on the client side of the web, which can be used to design / program how the web pages behave on the occurrence of an event. JavaScript is an easy to learn and also powerful scripting language, widely used for controlling web page behavior.

JavaScript is *not* **"Interpreted Java"**. In a nutshell, JavaScript is a dynamic scripting language supporting prototype based object construction. The basic syntax is intentionally similar to both Java and C++ to reduce the number of new concepts required to learn the language. Language constructs, such as if statements, for and while loops, and switch and try ... catch blocks function the same as in these languages (or nearly so).

JavaScript can function as both a procedural and an object oriented language. Objects are created programmatically in JavaScript, by attaching methods and properties to otherwise empty objects at run time, as opposed to the syntactic class definitions common in compiled languages like C++ and Java. Once an object has been constructed it can be used as a blueprint (or prototype) for creating similar objects.

Back End:

Node.js

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Npx-:

NPX (NPM Package Runner) Commands

List of useful npx (NPM Package Runner) commands.

What is NPX?

Using NPX we can execute/run node binaries without the need to install it locally or globally.

Commands

Create local server

npx server <folder-name>

Format using prettier

npx pretty-quick --check # to check the no of files to be affected
npx pretty-quick # format all files shown in check

Show system info, browsers installed, binaries like node, npm, yarn and npm packages installed in local & globally

npx envinfo

To show system info and show specific npm packages info

npx envinfo --preset <package-name>

Deploy to now.sh

npx now --public

Create react app

npx create-react-app <app-name>

yarn-:

Creates new projects from any `create-*` starter kits.

```
yarn create <starter-kit-package> [<args>]
```

This command is a shorthand that helps you do two things at once:

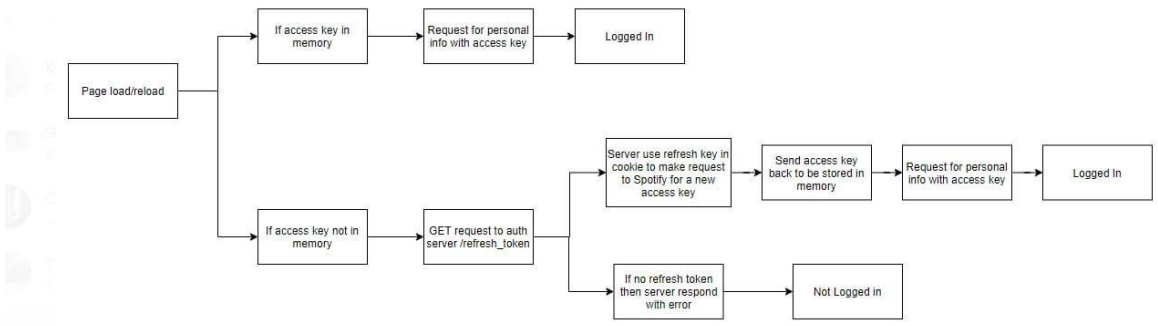
- Install `create-<starter-kit-package>` globally, or update the package to the latest version if it already exists
- Run the executable located in the `bin` field of the starter kit's `package.json`, forwarding any `<args>` to it

For example, `yarn create react-app my-app` is equivalent to:

```
```

API-:

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.



A clone web application using the create-react-app. The app comsumes data from the Spotify API and tries to mimic the UI and front-end behaviours of the official Spotify web player as much as possible

In order to be logged in, the app must have 2 things: a refresh_key stored in cookie and an access_key stored in memory. When there these values are present, the user is effectively "logged in" and therefore the app will render the "logged in" version with the user's personal info.

2.2 Fact Finding Techniques:

1.Onsite Observation's

We observed various website for online websites for this project for e.g.

1. https://www.apptunix.com/blog/spotify-like-app-development-guide/

2. https://www.pocket-lint.com/apps/news/spotify/139236-what-is-spotify-and- how-does-it-work

3. https://www.youtube.com/watch?v=pnkuI8KXW_8

2.Review the existing documents

We reviewed various websites and application that people use to see website related to Music apps and how they works.

3.Interviews

We mainly Interviewed the various owners of this business and asked how thye feel about making websites related to their business.

The information provided by both interviewees was very important because it helped us in the analysis of the current system and the constructs of the data flow diagrams. In the interview some objectives are considered such as; Determining the areas to be discovered.\

4. Questionnaire

A series of questions to be asked is called questionnaires.

1.What is Music Mania and how does it work?

2.Can you download music from Music Mania?

3.How much data does Music Mania use?

4.How to find people and friends on Music Mania?

5.What is Music Mania Connect?

6.Is there Music Mania Connectivity with Amazon and Google Home?

7.What is Music Mania Pets?

8.What is Music Mania kids?

9.How much is Music Mania?

10.What do you get with Music Mania Free?

11.What is Music Mania time capsule?

12.What are Daily Mix Playlists?

2.3 Feasibility Study

1.Technical Feasibility

A technical feasibility study assesses the details of how you intend to deliver a product or service to customers. Think materials, labor, transportation, where your business will be located, and the technology that will be necessary to bring all this together.

2.Economical Feasibility

The degree to which the economic advantages of something to be made, done, or achieved are greater than the economic costs.

3.Financial Feasibility

Financial feasibility focuses specifically on the financial aspects of the study. It assesses the economic viability of a proposed venture by evaluating the startup costs, operating expenses, cash flow and making a forecast of future performance.

4.Operational Feasibility

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

2.4 Stockholders-:

1. App Onboarding

This feature includes registration, authentication and user profile creation.

In the user profile section, the user should have his own page containing all the essential information like his name, age, gender, date of birth, music preferences, and so on. In this section, you should take information from the user regarding his preferences to envision and provide him content based on his/her desires

2. Music-Streaming

This is the main feature of a music streaming app. The streaming method doesn't require the downloading of the entire file. The audio that the user requests is delivered to him in small packets to play the music instantly.

3. Search

The entire idea of a music streaming application is to give the listeners the opportunity to search for the type of music they want to listen to as per their mood.

4. Playlists

What could be a better option than giving your users a platform where they can create a list of all their preferred tracks in a single spot, classified according to their mood.

5. Social-Sharing

It is a well-known saying that the success your application gets is directly related to the promotions it gets on social-networking websites.

6. Offline-Mode

This feature permits users to listen to their favorite music even without the internet connection. It utilizes the local storage of the device to cache the audio information.

7. Push Notifications

Push-Notifications are not just a must-have feature, however, the most helpful feature through which you can stun your audience by giving them astonishing offers, notifications about recently added songs, discounts, and more.

8. Payment

The integration of this feature relies upon your spotify like application's business model. If your application is having a freemium business model like Spotify, it is important to incorporate this feature in your application so that users can pay hassle-free for all that they want.

CHAPTER-3

3. Requirement and analysis Problem Definition-

The biggest drawback is the low audio quality, MP3 uses the lossy algorithm which deletes the lesser audible music content to reduce the file size, thus compromising on the music quality, Music piracy increased to a greater extent, Cheaper or free duplicate versions of the original music files are available on the Internet for download .

There are some disadvantages of the existing system.

- The sound quality of the MP3 format is not as good as that of the CD , So , CD players provide clearer audio than do MP3 players , Although MP3s can be compressed at a higher bit rate , Most are encoded at 128 kilobits per second , compared with CDs , on which the listener receives sound at 196 kilobits per second , about 50 per cent higher .

- The data is susceptible to losses due to the malware or virus attacks , The people who used the file-sharing service , They had their computers accessed by the hackers , MP3 players are generally more expensive than CD players .

- MP3 compression may discard as much as 90 percent of the data from the original recording without a significant drop in sound quality , Nevertheless , The listeners with the exceptional hearing or high-end earphones may detect slight differences between the MP3 file & the original uncompressed CD recording .

- Unlike CDs , The albums on MP3s cannot be resold , When the people purchase the song from iTunes or another online MP3 store , They are not so much buying the song as indefinitely leasing it , This may limit the ability of the owners of MP3 players to refresh their libraries frequently , unlike owners of CD players , they cannot legally trade their songs for new ones .

Requirement Specification-

Music Mania is immediately appealing because you can access content for free by simply signing up using an Email address or by connecting with Facebook, Gmail Account. If you're not keen on monthly subscription fees for Music Mania Premium, or just want to dip your toe in and test it out, it's out, it's easy to get started and there's no commitment.

You can find out the main differences between Music Mania Free and Premium in our separate feature but as a quick summary, the free version is ad-supported, much like radio stations. The free version of Music Mania can be accessed on PC, laptop and mobile phone, but the full service needs a Music Mania Premium subscription.

MODULES OF PROPOSE SYSTEM-

1. Registration

Using this module customer can register or login into the system in order to use that system. User can search the for Music and create it's own playlist.

2. Music-Streaming

The streaming method doesn't require the downloading of the entire file. The audio that the user requests is delivered to him in small packets to play the music instantly.

3.Search

The entire idea of a music streaming application is to give the listeners the opportunity to search for the type of music they want to listen to as per their mood.

4,Playlists

What could be a better option that giving your users a platform where they can create a list of all their preferred tracks in a single spot, classified according to their mood.

5.Social-Sharing

It is a well- known saying that the success your application gets is directly related to the promotions it gets on social networking websites.

6.Offline-Mode

This feature permits users to listen to their favorite music even without the internet connection. It utilizes the local storage of the device to cache the audio information.

7.Push Notifications

Push-Notifications are not just a must-have feature, however, the most helpful feature through which you can stun your audience by giving them astonishing offers.

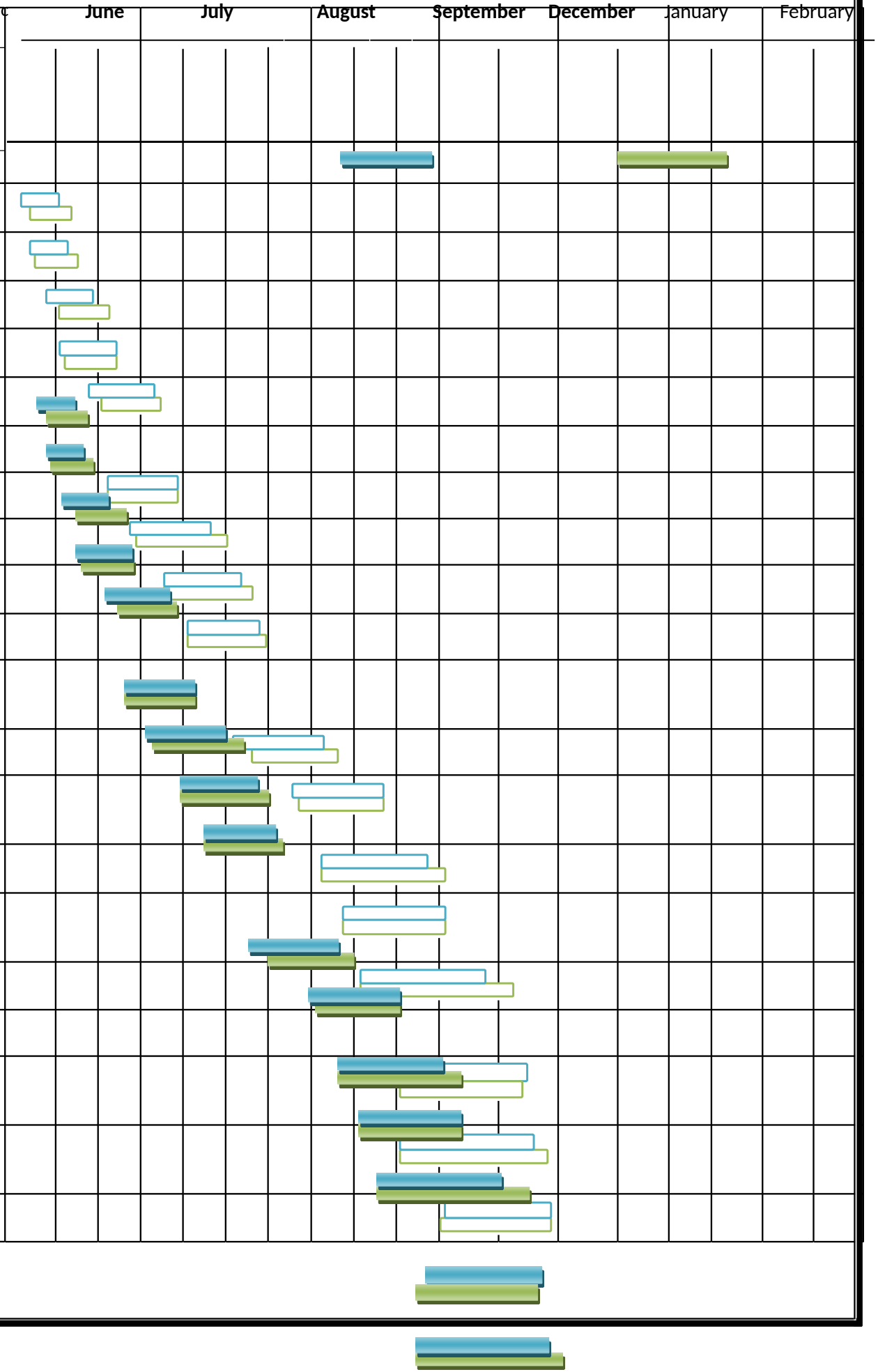
8.Payment

Payment can be done by using a credit card, debit card, internet banking, online. Payment entry is highly secure and trusted.

GANTT CHART

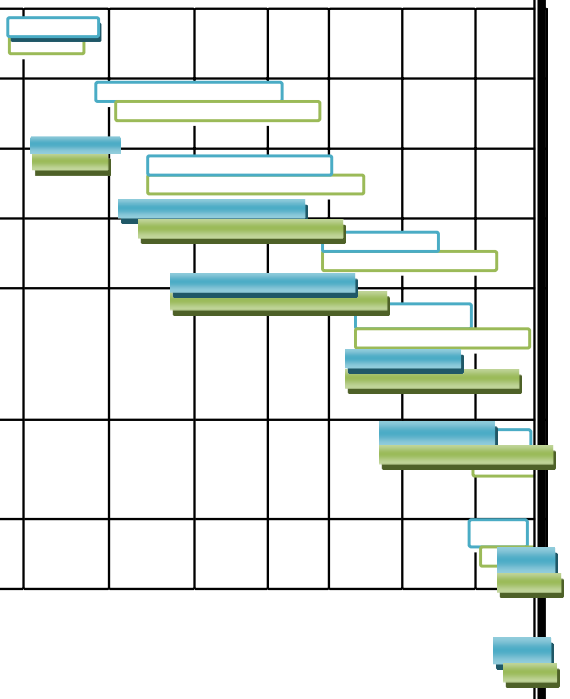
Targ et Time

Time Taken



SR. NO	Topic
1.	1. Background
1.1	Objective
1.2	Purpose
1.3	Scope
1.4	Applicability
2	System Planning
2.1	Survey of Technology
2.2	Fact Find techniques
2.3	Feasibility Study
2.4	Stakeholders
3	Requirement and Analysis
3.1	Problem Definition
3.2	Requirement and Specification
3.3	Planning and Scheduling
3.4	Software and Hardware
3.5	Conceptual Models
4	System Design
4.1	Data Integrity and
4.2	Constraints User Interface and Design
	Security Issues

4.3	Test Cases																	
5.1	Codin g																	
5.2	Details Code																	
5.3	Efficiency Testing																	
5.4	Approach Modification s and Improvemen																	
6	ts Conclusion And Future																	
7	Work References																	



SOFTWARE AND HARDWARE REQUIREMENT-

➢ Hardware Requirement

Hardware requirement for this system are as Follows:

	Processor	RAM	Disk Space
Client side	Intel P4 or equivalent	512MB	2GB
	Intel P4 or equivalent	512MB	1GB
Server side	Server Environment Capable H/w	2GB	As per the size of requirements DBMS

➢ Software Requirement

Software Requirement for this system are as follows:

FRONT END	Html5,css,JS,React.JS
BACK END	Node.js,npx,yarn
OPERATING SYSTEM	Windows 10

Conceptual Models

ER Diagram-:

An entity-relationship diagram, or ERD, is a chart that visually represents the relationship between database entities. ERDs model an organization's data storage requirements with three main components: entities, attributes, and relationships.

ENTITIES: Entities are objects or concepts that represent important data. They are typically nouns, e.g. custo er, supervisor, location, or promotion.

Strong entities: exist independently from other entity types. They always possess one or more attributes that uniquely distinguish each occurrence of t e entity.

Weak entities: depend on some other entity type. They don't possess unique attributes (also known as a primary key) and have no meaning in the diagram without depending on another entit . This other entity is known as the owner.

Associative entities: are entities that associate the instances of one or more entity types. They also contain attributes that are unique to the relationship between those entity instances.

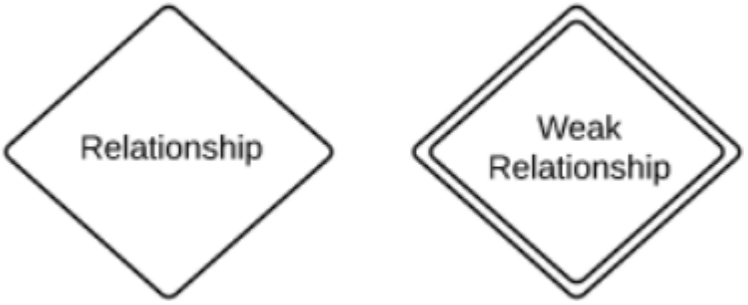
13/6 - 17/6 | 18/6 - 22/6 | 23/6 - 28/6 | 29/6-3/7 | 4/7-9/7 | 10/7-20/7 | 21/7-31/7 | 1/8-10/8 | 11/8-21/8 | 22/8-30/8 | 1/9-10/9 | 11/9-19/9 | 7/12-17/12 | 17/12-29/12 | 2/1-12/1 | 13/1-23/1 | 24/1-5/2 | 6/2-16/2

ntroduction

RELATIONSHIPS

Relationships are meaningful associations between or among entities. They are usually verbs, e.g. assign, associate, or track. A relationship provides useful information that could not be discerned with just the entity types.

Weak relationships, or identifying relationships, are connections that exist between a weak entity type and its owner

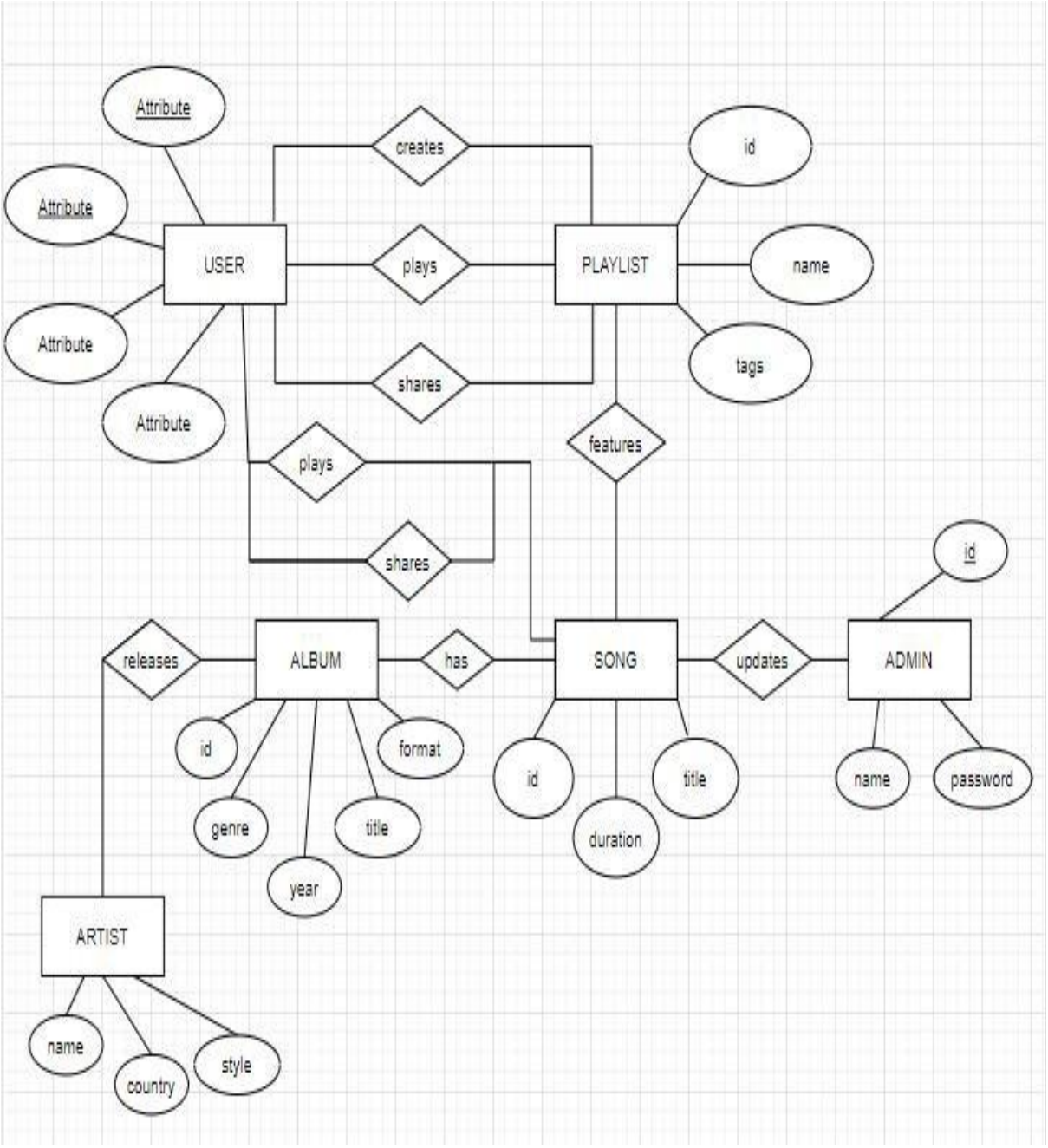


ATTRIBUTES

Attributes are characteristics of an entity, a many-to-many relationship, or a one-toone relationship.

Multivalve attributes are those that are capable of taking on more than one value.

Derived attributes are attributes whose value can be calculated from Related attribute

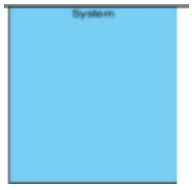

Attribute
Attribute
Attribute
Attribute
creates
USER
plays
shares
PLAYLIST
id
name
tags
plays
shares
features
releases
ALBUM
has
SONG
updates
ADMIN
id
id
format
genre
title
year
id
duration
title
name
password
ARTIST
name
country
style

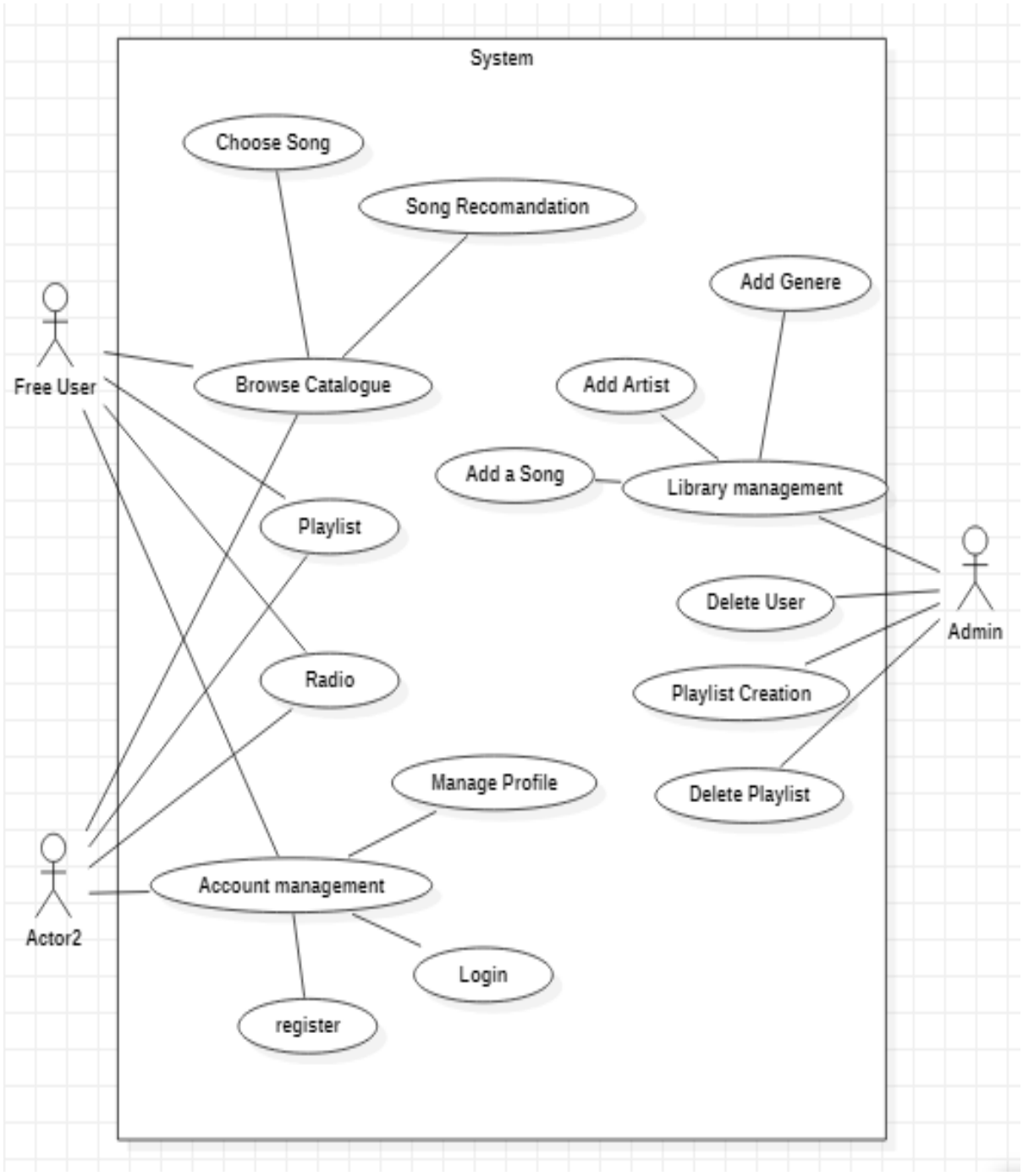
(ER DIAGRAM)

Use case Diagram-

USE CASE DIAGRAM is an expression of relations between the use cases in a specific system or object and the external actors. Use Case expresses the functions of the system and how the system functions interact with the external actors.

Symbol and Description-:

Symbol	Description
	Actor specifies a role played by a user or any other system that interacts with the subject
	Use case is a list of steps, typically defining interactions between an actor and a system, to achieve a goal.
	Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.
	An association is the relationship between an actor and a business use case. It indicates that an actor can use a certain functionality of the business system.



(USE CASE DIAGRAM)

Class Diagram-:

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

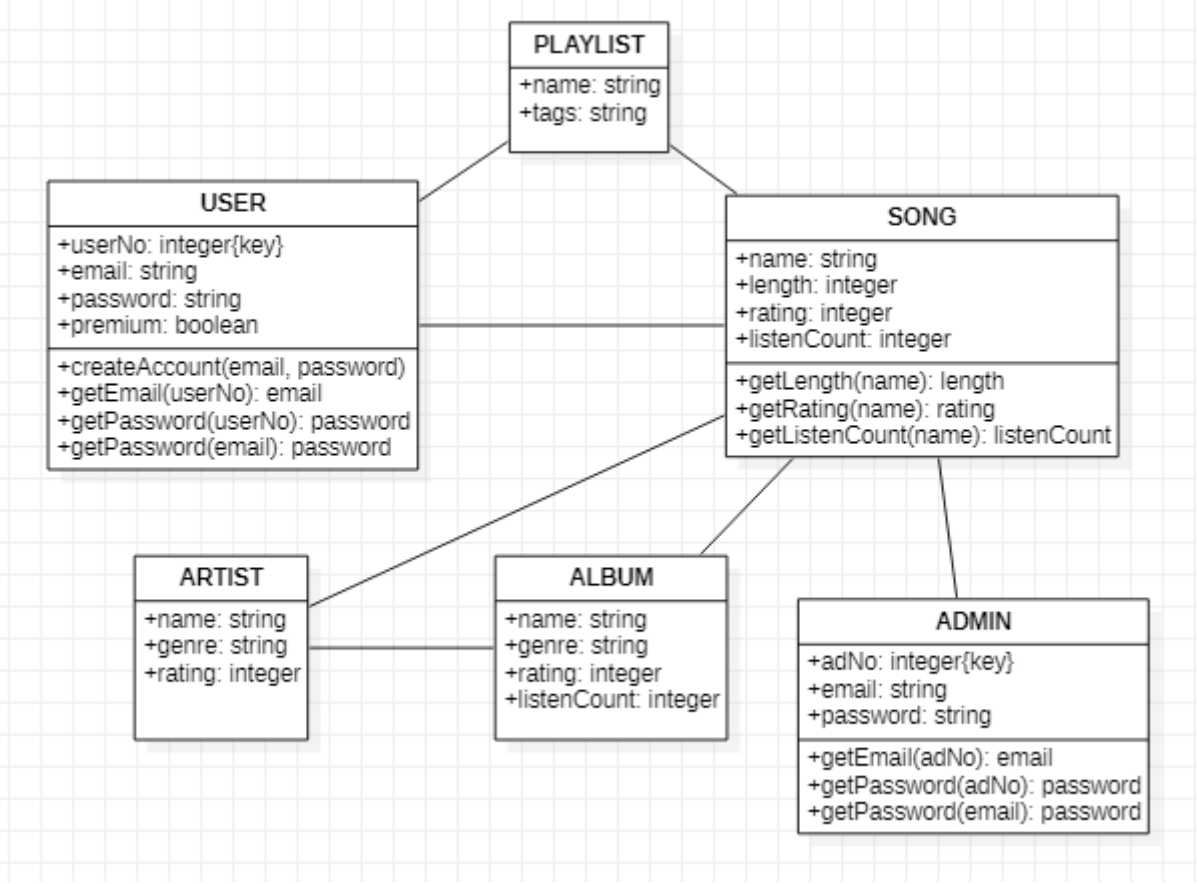
In the diagram, classes are represented with boxes that contain three compartments:

The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.

The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.

The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

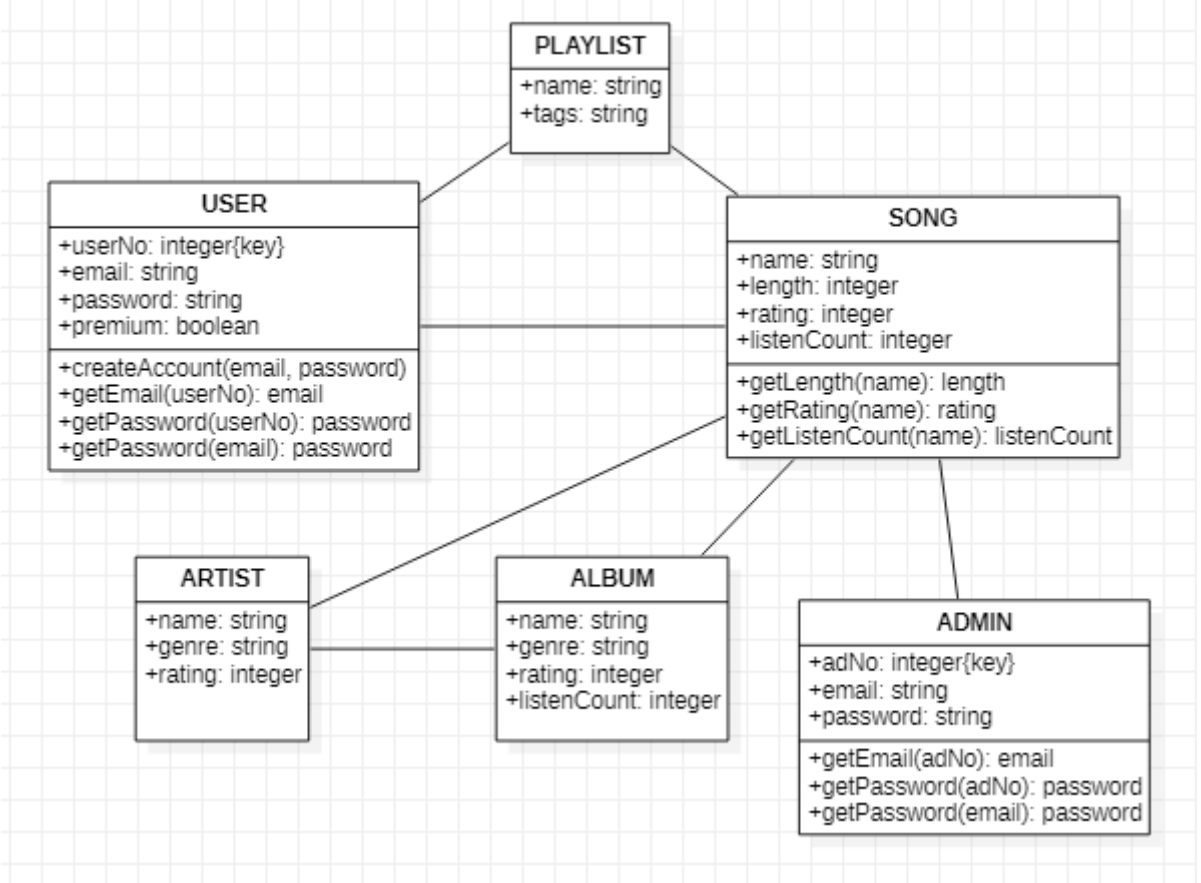
In the design of a system, a number of classes are identified and grouped together in a class diagram that helps to determine the static relations between them. With detailed modeling, the classes of the conceptual design are often split into a number of subclasses.



(CLASS DIAGRAM)

Object Diagram

An object diagram is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The use of object diagrams is fair ly limited, namely to show examples of data structure.



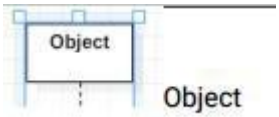
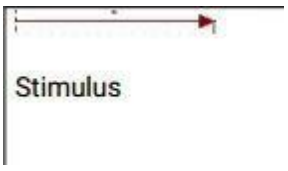
(OBJECT DIAGRAM)

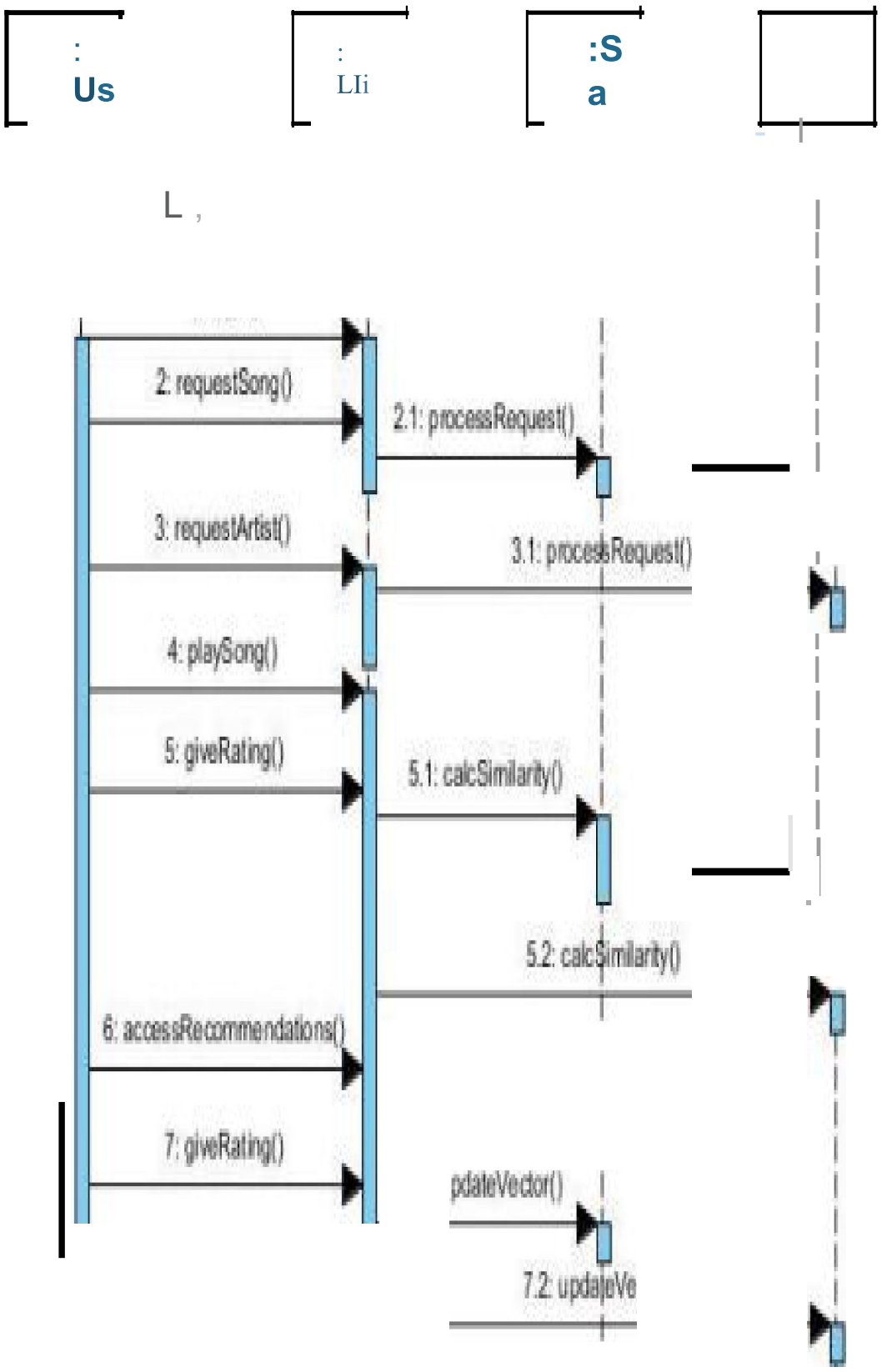
Sequence Diagram-

A sequence diagram shows object connections arranged in time chain. It depicts the objects and classes implicated in the situation and the series of messages exchange between the objects needed to carry out the functionality of the scenario. Sequence diagrams are usually associated with use case realizations in the Logical View of the system under improvement. Sequence diagrams are sometimes called event diagrams or event scenarios.A sequence diagram shows,
as parallel vertical lines (lifelines), different processes or objects that
live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Sequence Diagram expresses the interactions of instances. It is a direct expression of the Interaction Instance Set, which is a set of the stimuli exchanged between the instances within a Collaboration Instance Set

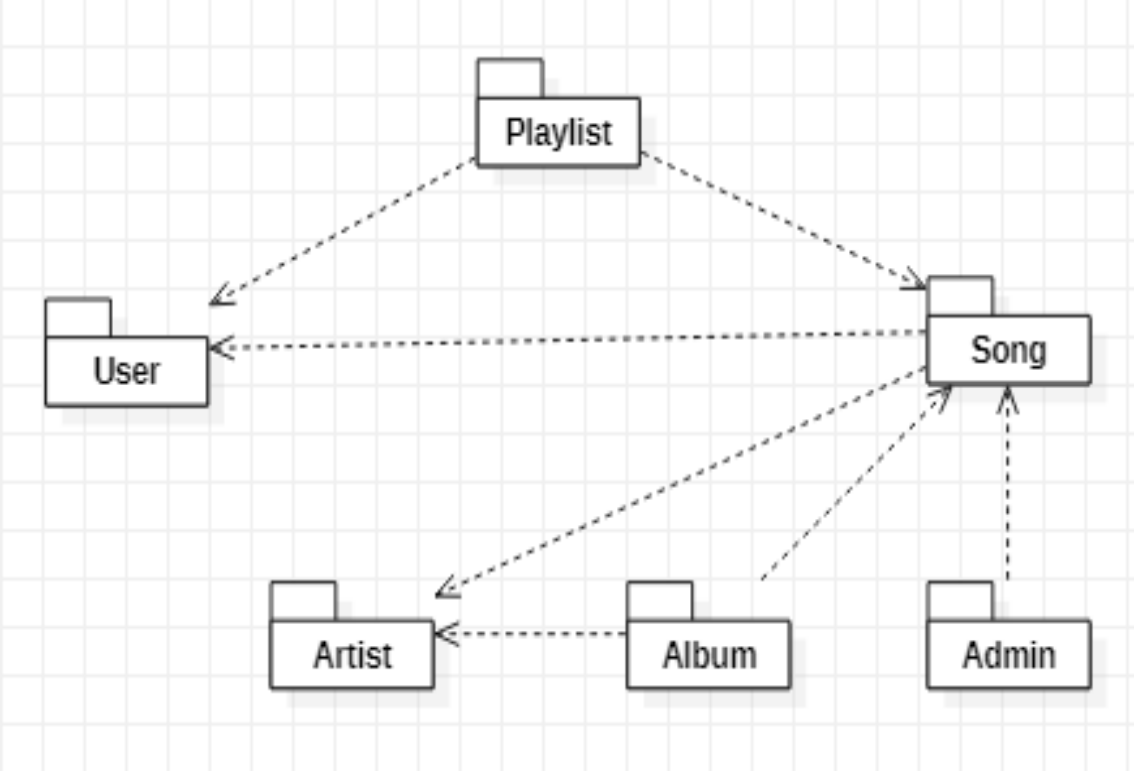
Symbols and Description:

Symbol	Description
	Object are model element that represent instances of a class or a class
	Message is a element that defines a specifies kind of communication between the instances in an interaction


:
Us
:
LIi
:S
a
2: requestSong()
2.1: processRequest()
3: requestArtist()
3.1: processRequest()
4: playSong()
5: giveRating()
5.1: calcSimilarity()
5.2: calcSimilarity()
6: accessRecommendations()
7: giveRating()

Package Diagram-:

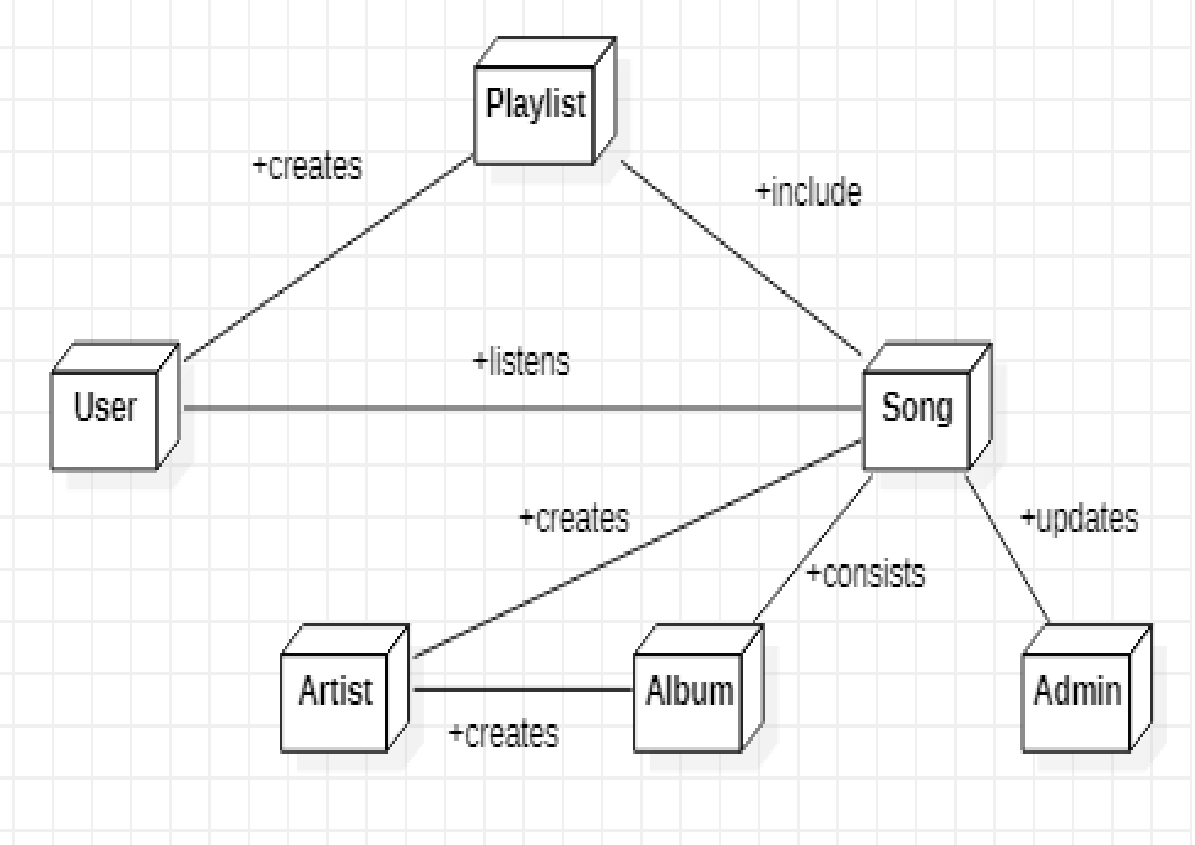
Package diagram is used to simplify complex class diagrams, you can group classes into packages. A package is a collection of logically related UML elements. The diagram below is a business model in which the classes are grouped into packages: Packages appear as rectangles with small tabs at the top.



(PACKAGE DIAGRAM)

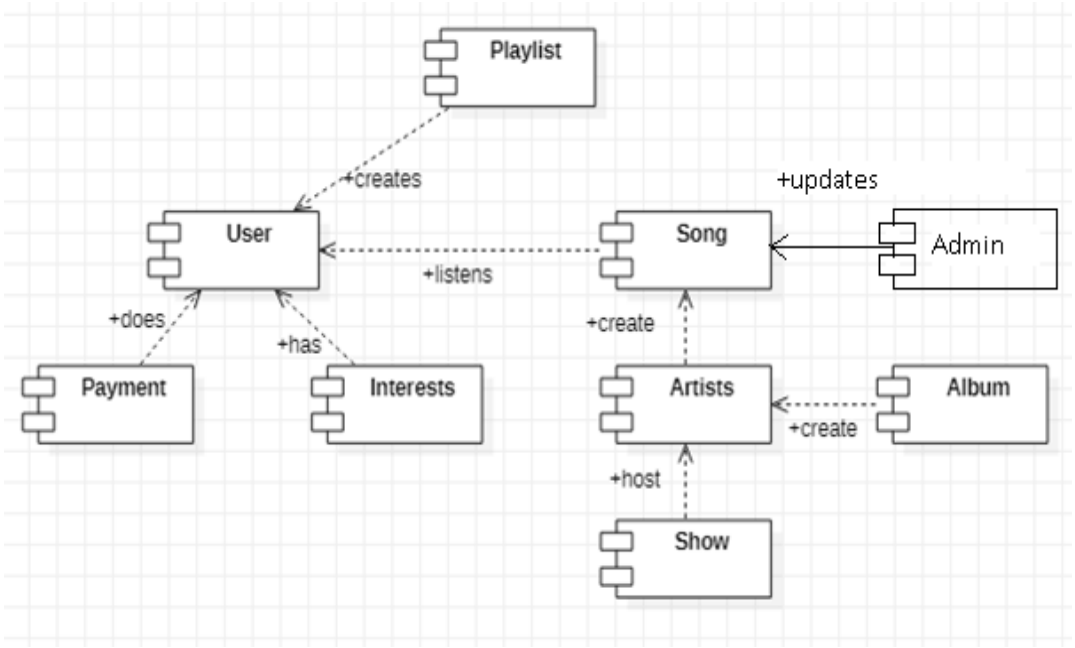
Deployment Diagram-:

A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components exist, what software components run on each node, and how the different pieces are connected.



Component Diagram-:

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development**.**



CHAPTER-4

System Design

Data Design-:

Registration table-:

Login Table-:

ATTRIBUTE
User name
Password
Forget Password

PLAYLIST TABLE-:

ATTRIBUTE
Create Playlist
Choose Categries
Add Music
List Music

PAYMENT TABLE-:

ATTRIBUTE
First Name
Last Name
Total Amount
Card Number
Cvv Number
Id number
Address

FEEDBACK TABLE-:

ATTRIBUTE
NAME
EMAIL ID
PHONE NO
DESCRIPTION

ADMIN TABLE-:

ATTRIBUTE
NAME
PASSWORD

Data Integrity and Constraints:

REGISTRATION TABLE

ATTRIBUTE	DATA TYPE	CONSTRAINT
First Name	Varchar(30)	Not Null
Last Name	Varchar(30)	Not Null
Phone No	Long(10)	Primary key
Email Id	Varchar(50)	Not null
City	Varchar(30)	Not Null
Password	Varchar(20)	Not Null
Confirm Password	Varchar(20)	Not Null

LOGIN TABLE:-

ATTRIBUTE	DATA TYPE	CONSTRAINT
User name	Varchar(30)	Primary key
Password	Varchar(20)	Not null

ATTRIBUTE
First Name
Last Name
Phone no.
Email ID
Password
Conform Password

PAYMENT TABLE:-

ATTRIBUTE	DATA TYPE	CONSTRAINT
Total Amount	Int(10)	Not null

FEEDBACK TABLE:-

ATTRIBUTE	DATA TYPE	CONSTRAIN T
Name	Varchar (30)	Not n ull
Em ail l!d	Varchar (20)	Not nul!l
Phone no	Int(10)	Pr im ary key
Descr ipt iion	Varchar (90)	!Not nulll

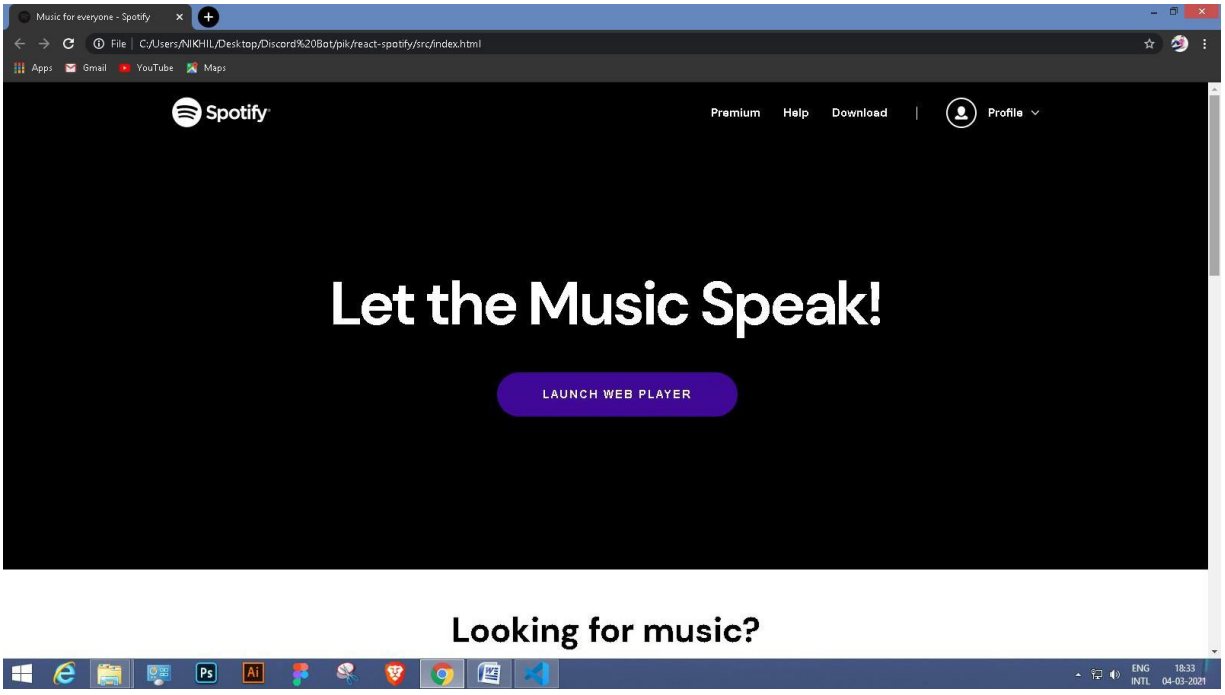
ADMIN TABLE:-

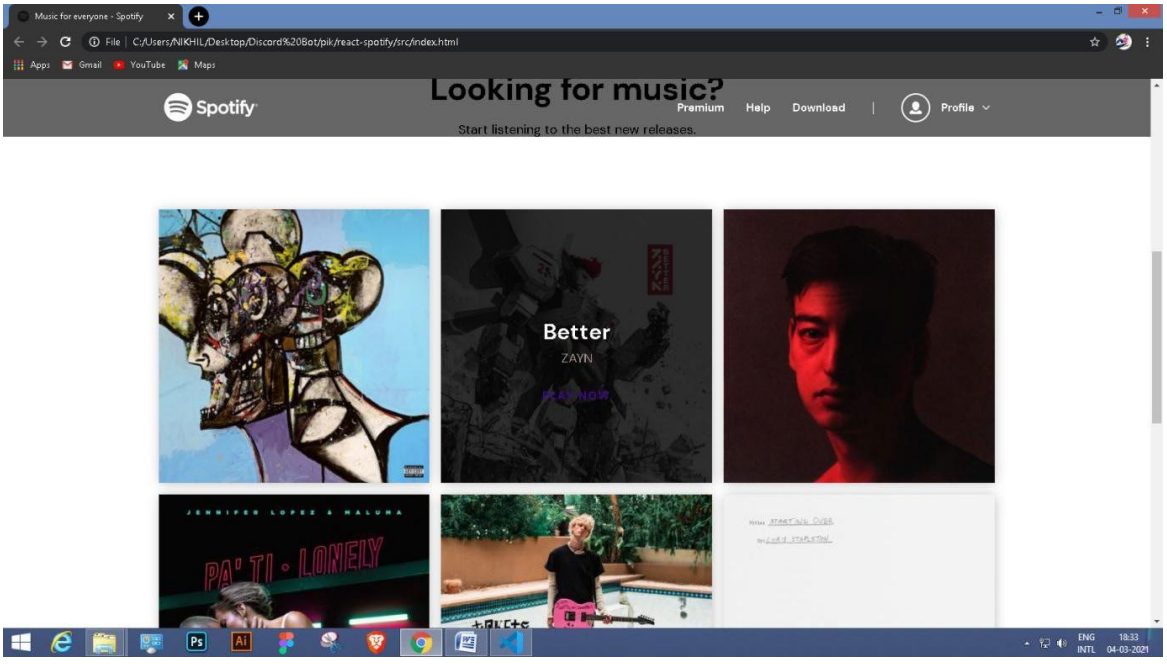
ATTRI BUTE	DATA TYPE	CONSTRAI NT
User name	Varchar(30)	Primary key
Password	Varchar(30)	Not nul!ll

User interface and Design:

User interface (UI) design is the process of making interfaces in software or computerized devices with a focus on looks or style. Designers aim to create designs users will find easy to use and pleasurable. UI design typically refers to graphical user interfaces but also includes others, such as voice-controlled ones.

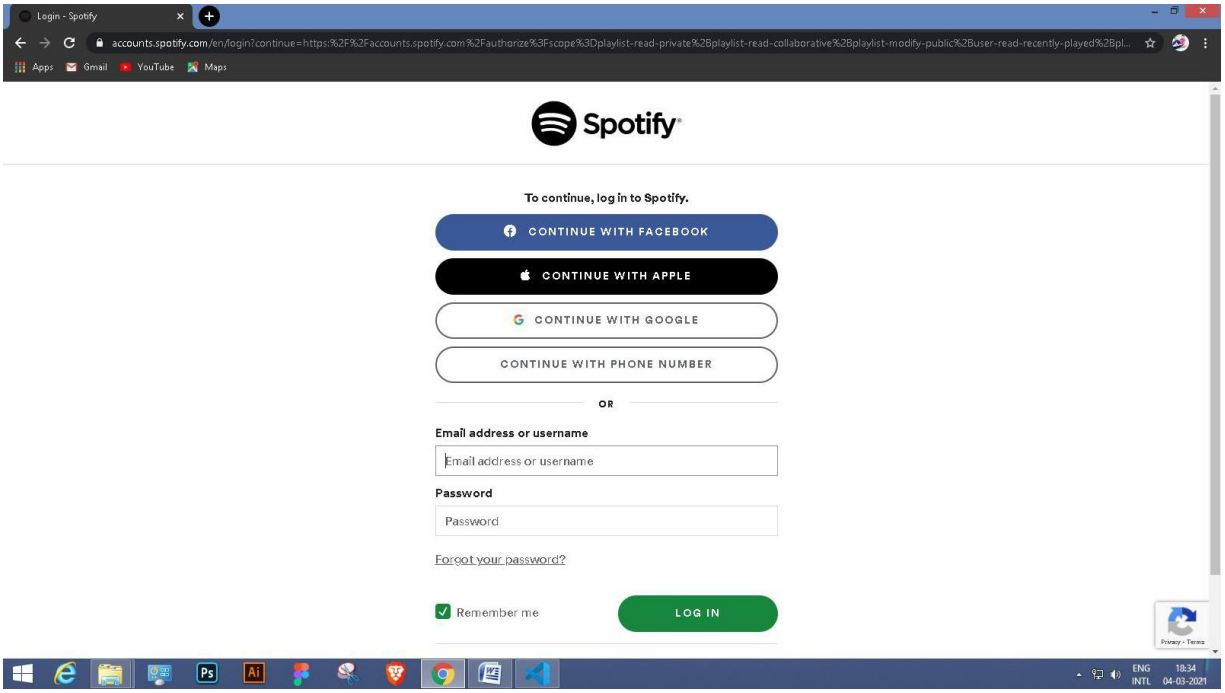
Home page-:



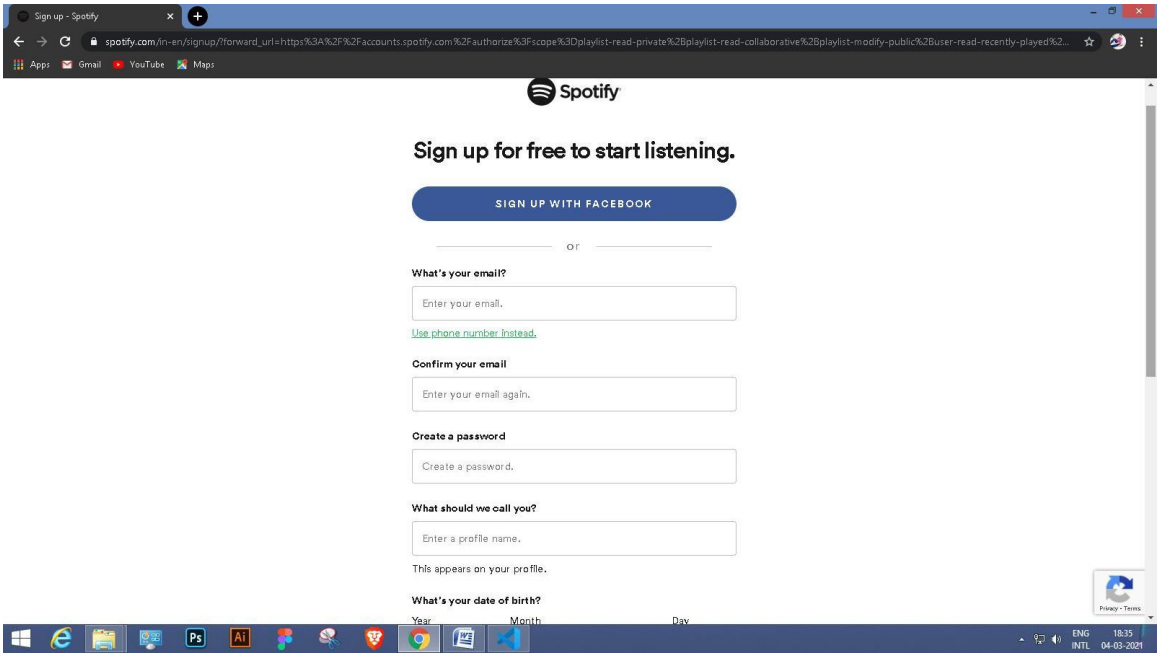



Spotify
COMPANY
About
Contact
For the Record
COMMUNITIES
For artists
Developers
Advertising
Investors
Vendors
USEFUL LINKS
Help
Web Player
Free Mobile App
Legal
Privacy Center
Privacy Policy
Cookies
Terms and Conditions
About Ads
Additional CA Privacy Disclosures
© Copyright
Nikhil Arjun
Shegun Deoghare
ENG
INTL
10:34
04-08-2021

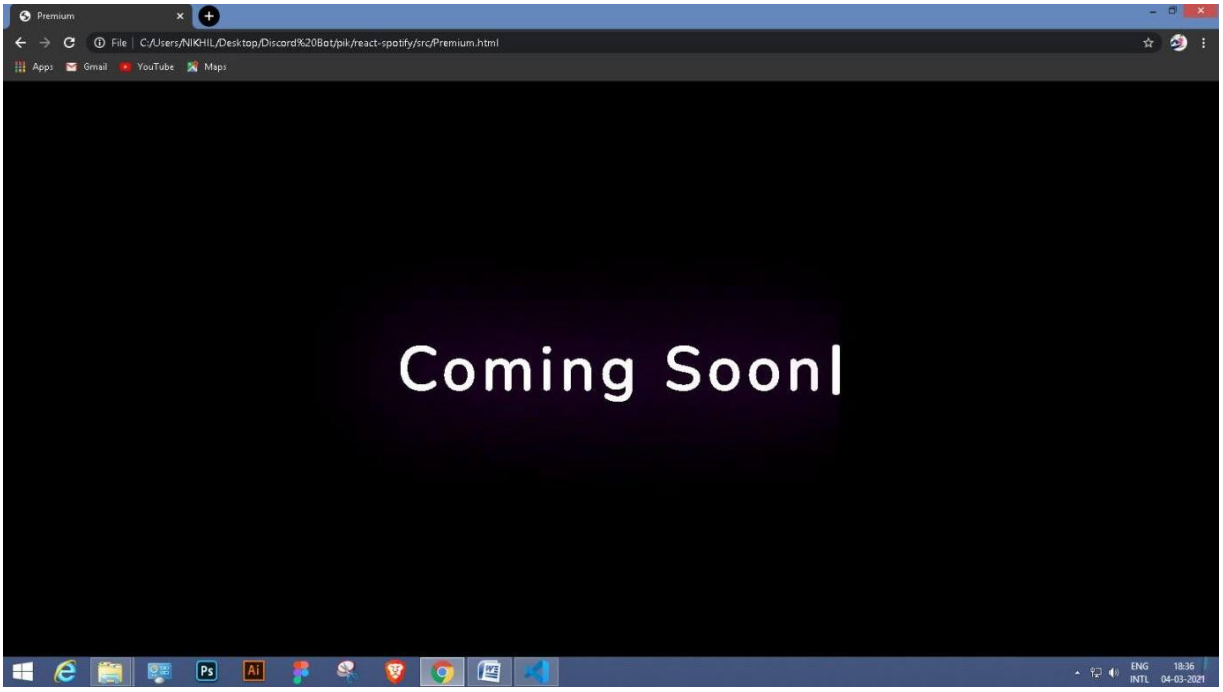
LOGIN FORM-:



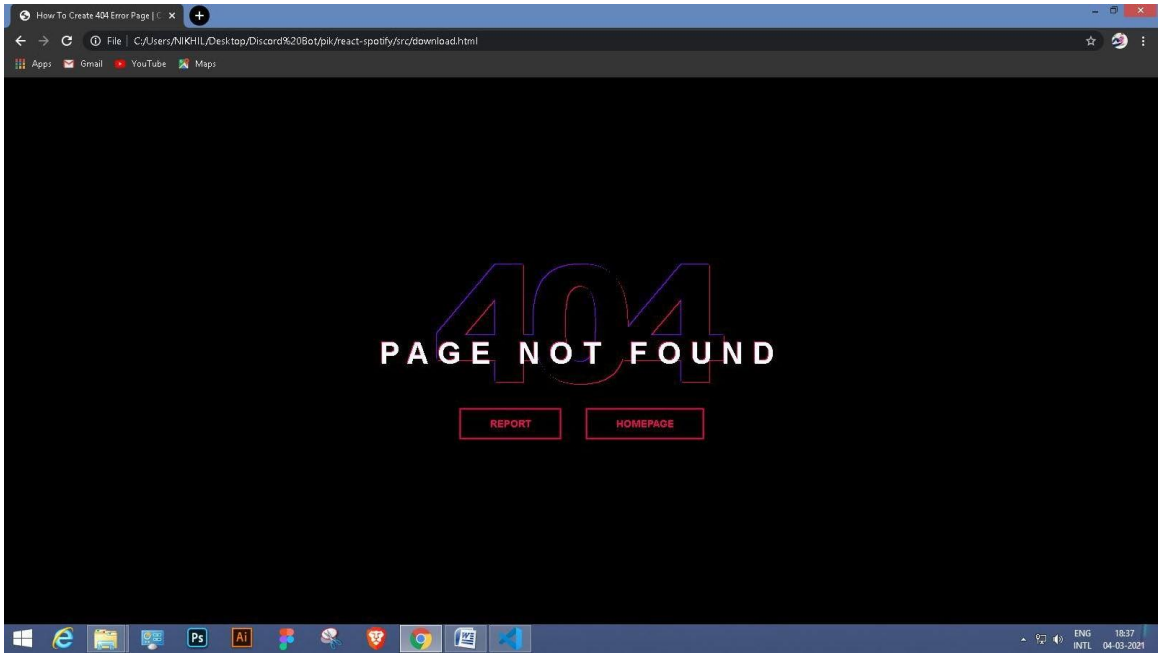
Registration Page-:



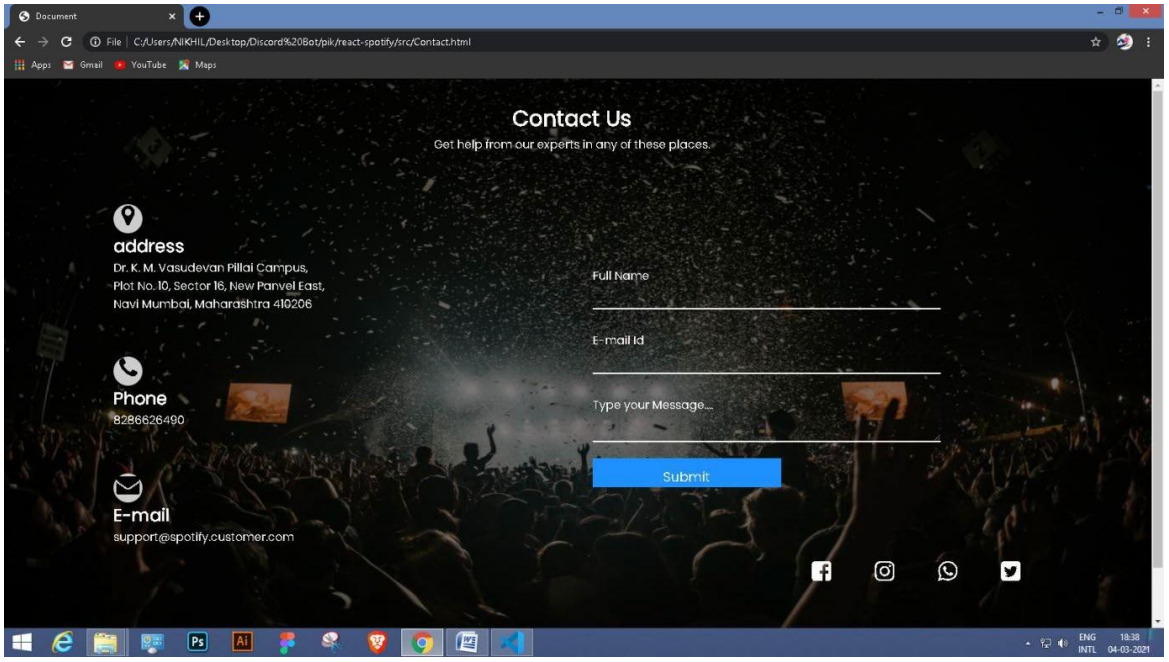
Premium Page(Upcoming)-:



Error Page-:

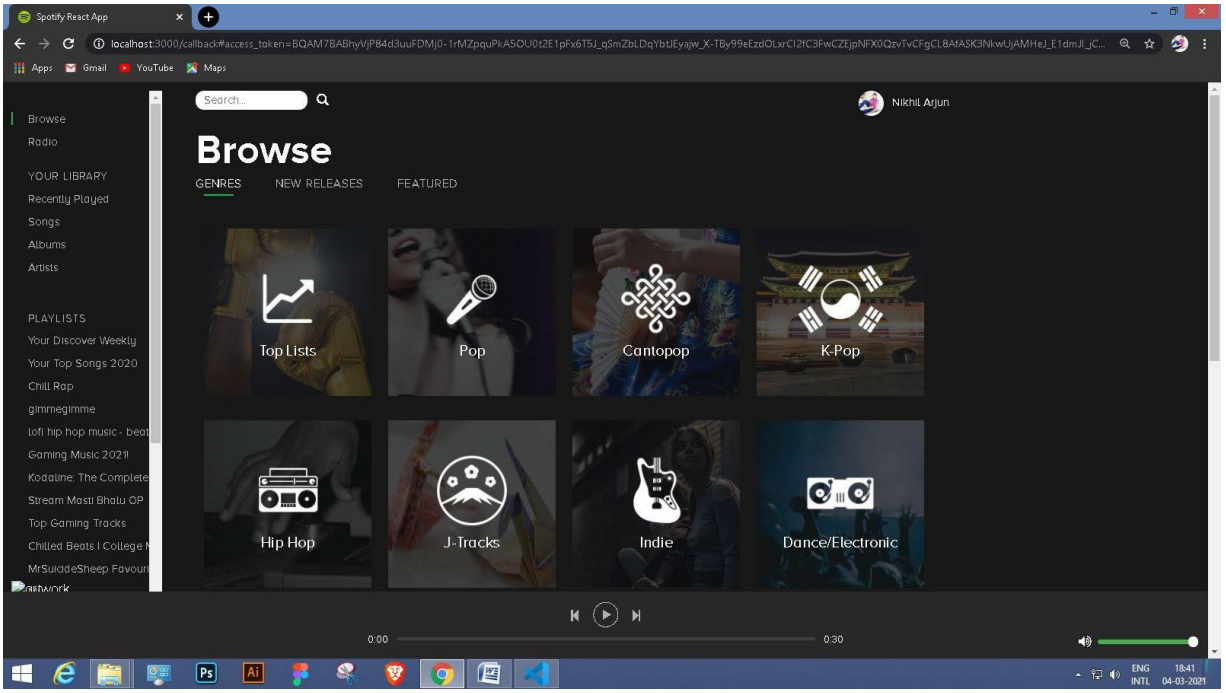


Contact Us Page-:

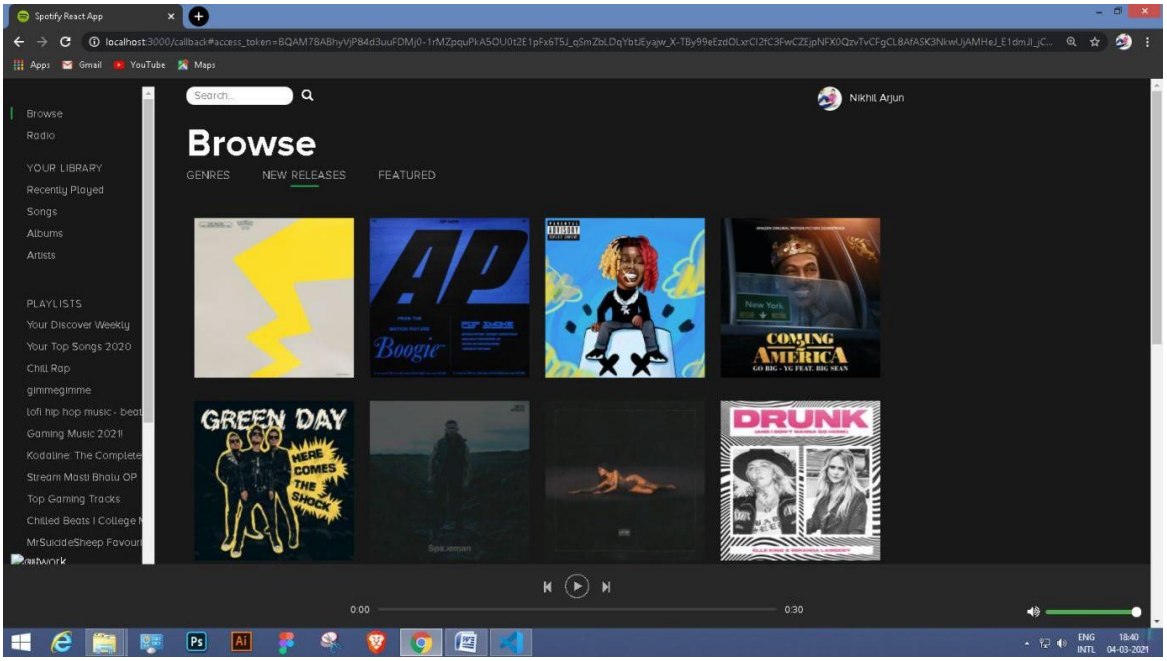


Player Page-:

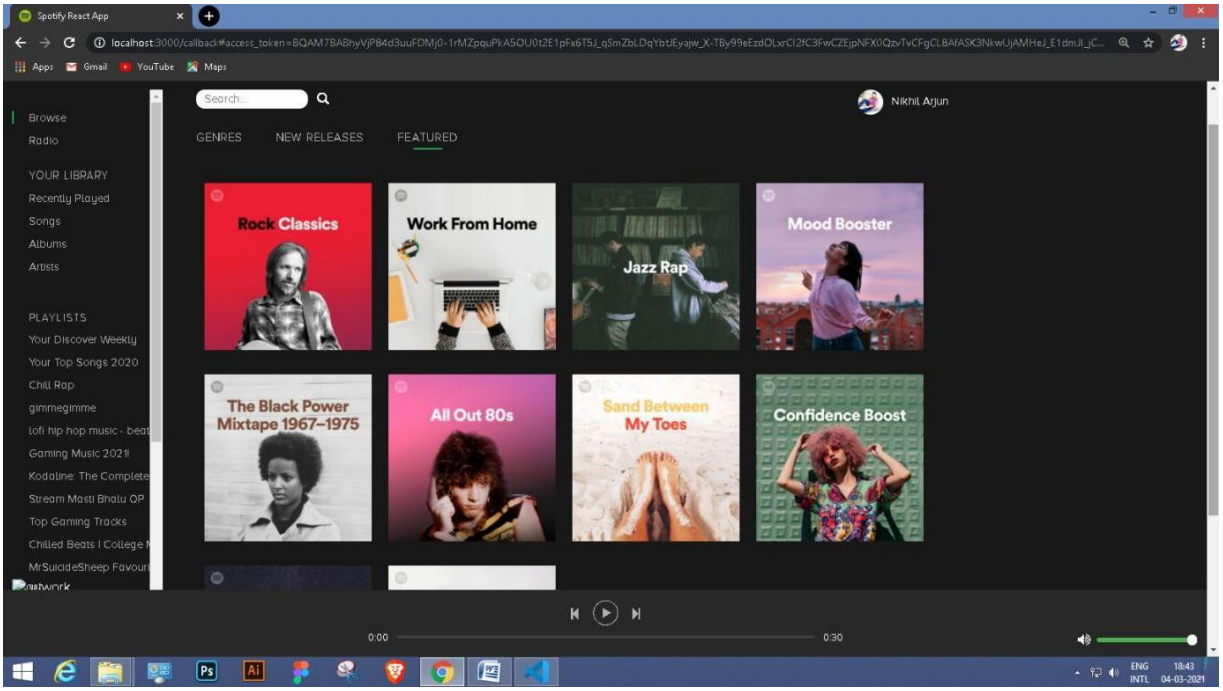
GENRES



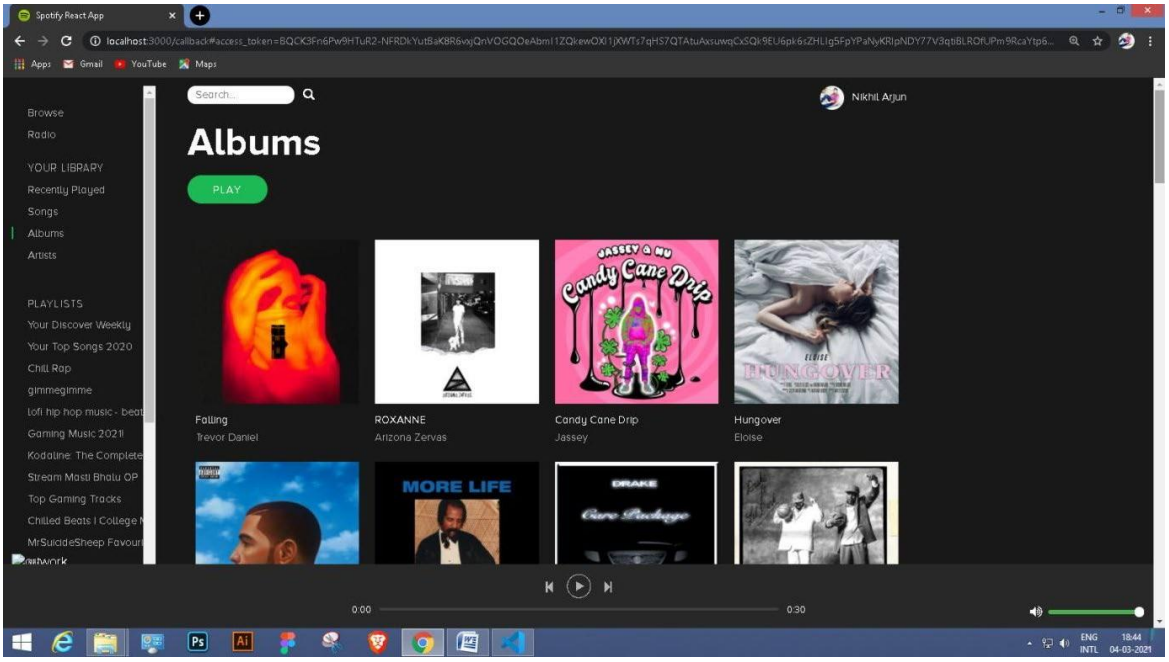
New releases-:



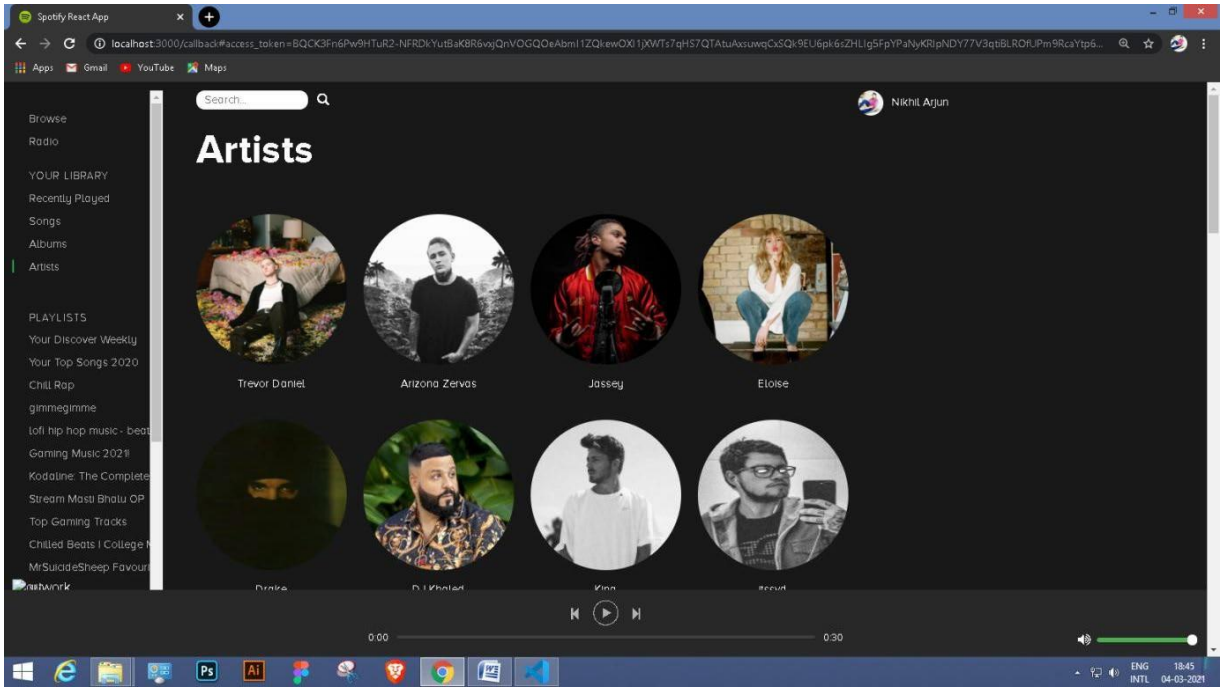
Featured-:



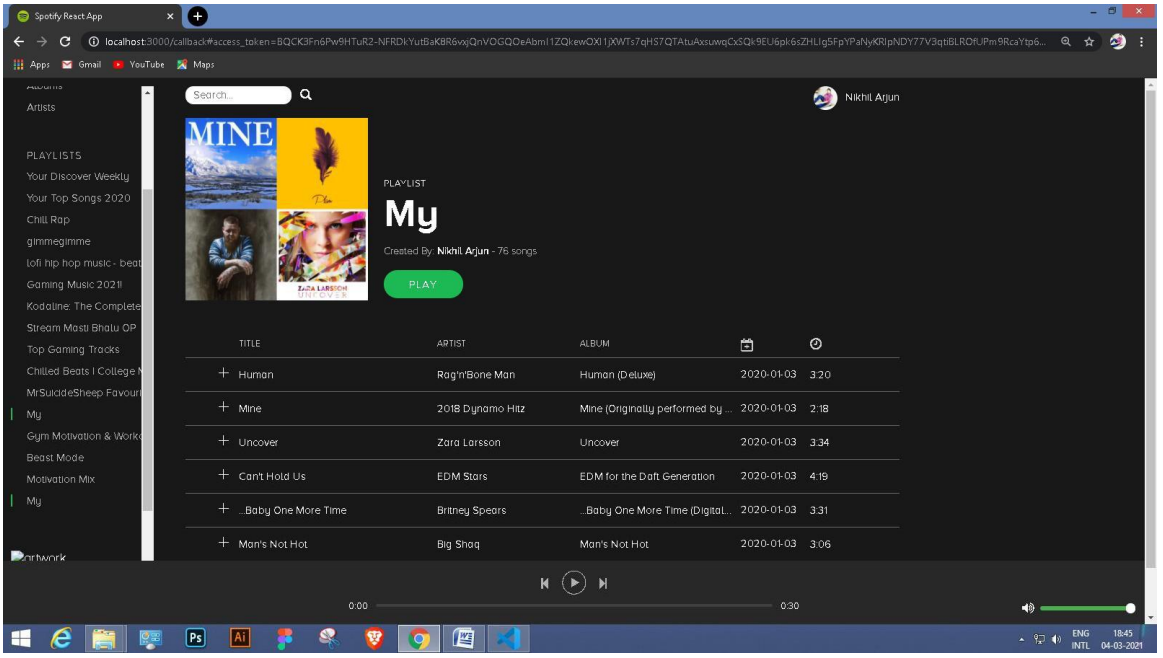
Albums-:



Artists-:



Playlist-:



Security Issues:

Most Common Website Security Issues

1 . SQL INJECTIONS

SQL injection is a type of web application security vulnerability in which an attacker attempts to use application code to access or corrupt database content. If successful, this allows the attacker to create, read, update, alter, or delete data stored in the back-end database. SQL injection is one of the most prevalent types of web application security vulnerabilities.

2. CROSS SITE SCRIPTING (XSS)

Cross-site scripting (XSS) targets an application's users by injecting code, usually a client-side script such as JavaScript, into a web application's output. The concept of XSS is to manipulate client-side scripts of a web application to execute in the manner desired by the attacker. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface websites or redirect the user to malicious sites.

3. BROKEN AUTHENTICATION & SESSION MANAGEMENT

Broken authentication and session management encompass several security issues, all of them having to do with maintaining the identity of a user. If authentication credentials and session identifiers are not protected at all times, an attacker can hijack an active session and assume the identity of a user.

4. INSECURE DIRECT OBJECT REFERENCES

Insecure direct object reference is when a web application exposes a reference to an internal implementation object. Internal implementation objects include files, database records, directories and database keys. When an application exposes a reference to one of these objects in a URL, hackers can manipulate it to gain access to a user's personal data.

5. SECURITY MISCONFIGURATION

Security misconfiguration encompasses several types of vulnerabilities all centered on a lack of maintenance or a lack of attention to the web application configuration. A secure configuration must be defined and deployed for the application, frameworks, application server, web server, database server and platform. Security misconfiguration gives hackers access to private data or features and can result in a complete system compromise.

6. CROSS-SITE REQUEST FORGERY (CSRF)

Cross-Site Request Forgery (CSRF) is a malicious attack where a user is tricked into performing an action he or she didn't intend to do. A third-party website will send a request to a web application that a user is already authenticated against (e.g. their bank). The attacker can then access functionality via the victim's already authenticated browser. Targets include web applications like social media, in browser email clients, online banking, and web interfaces for network devices. Don't get caught with your guard down. Practice safe website security measures and always be ready to protect yourself, and your company's future, from an attack that you might never recover from. The best way to tell if your website or server is vulnerable is to conduct regular security audits.

Encryption Algorithms

Encryption algorithms are commonly used in computer communications, including FTP transfers. Usually they are used to provide secure transfers. If an algorithm is used in a transfer, the file is first translated into a seemingly meaningless cipher text and then transferred in this configuration; the receiving computer uses a key to translate the cipher into its original form. So if the message or file is intercepted before it reaches the receiving computer it is in an unusable (or encrypted) form.

Here are some commonly used algorithms:

DES/3DES or TripleDES

This is an encryption algorithm called Data Encryption Standard that was first used by the U.S. Government in the late 70's. It is commonly used in ATM machines (to encrypt PINs) and is utilized in UNIX password encryption. Triple DES or 3DES has replaced the older versions as a more secure method of encryption, as it encrypts data three times and uses a different key for at least one of the versions.

Blowfish

Blowfish is a symmetric block cipher that is unpatented and free to use. It was developed by Bruce Schneier and introduced in 1993.

AES

Advanced Encryption Standard or Rijndael; it uses the Rijndael block cipher approved by the National Institute of Standards and Technology (NIST). AES was originated by cryptographers Joan Daemen and Vincent Rijmen and replaced DES as the U.S. Government encryption technique in 2000.

Twofish

Twofish is a block cipher designed by Counterpane Labs. It was one of the five Advanced Encryption Standard (AES) finalists and is unpatented and open source.

IDEA

This encryption algorithm was used in Pretty Good Privacy (PGP) Version 2 and is an optional algorithm in OpenPGP. IDEA features 64 bit blocks with a 1 28 bit key.

MD5

MD5 was developed by Professor Ronald Riverst and was used to create digital signatures. It is a one way hash function and intended for 32 bit machines. It replaced the MD4 algorithm.

SHA 1

SHA 1 is a hashing algorithm similar to MD5, yet SHA 1 may replace MD5 since it offers more security

HMAC

This is a hashing method similar to MD5 and SHA 1 , sometimes referred to as HMAC MD5 and HMAC SHA1 .

RSA Security

- **RC4** RC4 is a variable key size stream cipher based on the use of a random permutation.

- **RC5** This is a parameterized algorithm with a variable block, key size and number of rounds.

- **RC6** This an evolution of RC5, it is also a parameterized algorithm that has variable block, key and a number of rounds. This algorithm has integer multiplication and 4 bit working registers

TEST CASES

SR NO	Form Name	Test Condition	Step or Procedure	Input Test Data	Expected Result	Actual Output	Pass/ Fail
1	Login	Check login with valid input	Wrong username with correct password	User name: admin PASS:admin	Display Message: “Invalid Username or Password	Display Message “Invalid Username or Password	PASS
2	Login	Check login with valid input	If Numbers are inserted	User name: admin PASS:admin	Display Message: “Invalid Username or Password	Display Message: “Invalid Username or Password “	PASS
3	User	Check Alphabetic Values	If mobile number is more than 10 digit	9653329853	Display Message: “only Characters are allowed”	Display Message: “Only Characters are allowed,	PASS
4	User	Check Email id	Wrong username with correct password	Name: rahul456	Display Message: “Enter 1 0 digit number only”	Display Message: “Enter 1 0 digit number only”.”	PASS
5	User	Check Email id	If mobile number is more than 10 digit	828666425	Display Message: “Phone number cannot be less than 1 o digit”.	Display Message: “Phone number cannot be less than 1 o digit”.”	PASS
6	User	Check Email id	If @mail.com is not specified	nik@gmail.com	Display Message: “Email is expected	Display message : “ Email is Expected	PASS

Chapter 5

System Coding, Implementation and Testing

Index.html(Main page)-

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Music for everyone - Spotify</title>
    <link rel="icon" href="images/favicon.ico" />
    <link
      href="https://fonts.googleapis.com/css2?family=DM+Sans:wght@400;500;700&display=swap
      " rel="stylesheet"
    />
    <link rel="stylesheet" href="css/base.css" />
    <link rel="stylesheet" href="css/header.css" />
    <link rel="stylesheet" href="css/hero.css" />
    <link rel="stylesheet" href="css/albums.css" />
    <link rel="stylesheet" href="css/footer.css" />
    <link rel="stylesheet" href="css/media-queries.css" />
  </head>
  <body>
    <!-- HEADER -->
    <header>
      <div class="header-container">
        <div class="header-logo">
          <img src="images/spotify-logo.svg" />
        </div>
        <!-- This is the menu that will be shown on mobile devices-->
        <nav class="nav-mobile">
          <!-- Profile -->
          <div class="profile">
            <svg viewBox="0 0 1024 1024" class="profile-icon">
              <path
                d="M730.06 679.64q-45.377 53.444-101.84 83.443t-120 29.999q-64.032
0-120.75-30.503t- 102.6-84.451q-40.335 13.109-77.645 29.747t-53.948 26.722l-17.142
10.084Q106.388 763.84 84.96
802.41t-21.428 73.107 25.461 59.242 60.754 24.705h716.95q35.293 0 60.754-24.705t25.461-59.242-
21.428-72.603-51.679-57.225q-6.554-4.033-18.907-10.84t-51.427-24.453-79.409-30.755zm-221.84
25.72q-34.285 0-67.561-14.873t-60.754-40.335-51.175-60.502-40.083-75.124-25.461-84.451-9.075-
87.728q0-64.032 19.915-116.22t54.452-85.964 80.67-51.931 99.072-18.151 99.072 18.151 80.67
51.931 54.452 85.964 19.915 116.22q0 65.04-20.167 130.58t-53.948 116.72-81.426 83.443-98.568
32.268z"
              ></path>
            </svg>
          </div>
          <!-- Hamburger menu -->
          <input type="checkbox" class="toggler" />
          <div class="hamburger-menu"><div></div></div>
          <div class="menu">
            <div class="background-overlay"></div>
```

```
          <div class="menu-overlay">
            <ul>
              <li><a href="Premium.htm">Premium</a></li>
              <li><a href="#">Help</a></li>
              <li><a href="download.html">Download</a></li>
              <li role="separator"></li>
              <li><a href="#">Account</a></li>
              <li><a
href="https://accounts.spotify.com/en/login?continue=https:%2F%2Faccounts.spotify.com%2Fauthor
ize%3Fscope%3Dplaylist-read-private%2Bplaylist-read-collaborative%2Bplaylist-modify-
public%2Buser-read-recently-played%2Bplaylist-modify-private%2Bugc-image-upload%2Buser-
follow-modify%2Buser-follow-read%2Buser-library-read%2Buser-library-modify%2Buser-read-
private%2Buser-read-email%2Buser-top-read%2Buser-read-playback-
state%26response_type%3Dtoken%26redirect_uri%3Dhttp%253A%252F%252Flocalhost%253A300
0%252Fcallback%26client_id%3D230be2f46909426b8b80cac36446b52a">Sign in</a></li>
            </ul>
            <div class="menu-overlay-logo">
              <img src="images/spotify-logo.svg" />
            </div>
          </div>
        </div>
      </nav>
      <!-- This is the menu that will be shown on Desktop -->
      <nav class="nav-desktop">
        <!-- Site menu -->
        <ul>
          <li><a href="Premium.html">Premium</a></li>
          <li><a href="#">Help</a></li>
          <li><a href="download.html">Download</a></li>
          <li role="separator"></li>
        </ul>
        <!-- Profile menu -->
        <div class="profile-container">
          <input type="checkbox" class="dropdown-menu-toggler" />
          <div class="dropdown-menu">
            <ul>
              <li><a href="#">Account</a></li>
              <li><a
href="https://accounts.spotify.com/en/login?continue=https:%2F%2Faccounts.spotify.com%2Fauthor
ize%3Fscope%3Dplaylist-read-private%2Bplaylist-read-collaborative%2Bplaylist-modify-
public%2Buser-read-recently-played%2Bplaylist-modify-private%2Bugc-image-upload%2Buser-
follow-modify%2Buser-follow-read%2Buser-library-read%2Buser-library-modify%2Buser-read-
private%2Buser-read-email%2Buser-top-read%2Buser-read-playback-
state%26response_type%3Dtoken%26redirect_uri%3Dhttp%253A%252F%252Flocalhost%253A300
0%252Fcallback%26client_id%3D230be2f46909426b8b80cac36446b52a">Sign in</a></li>
            </ul>
          </div>
          <div class="profile">
            <svg viewBox="0 0 1024 1024" class="profile-icon">
              <path
                d="M730.06 679.64q-45.377 53.444-101.84 83.443t-120 29.999q-64.032 0-120.75-
30.503t-102.6-84.451q-40.335 13.109-77.645 29.747t-53.948 26.722l-17.142 10.084Q106.388
763.84 84.96 802.41t-21.428 73.107 25.461 59.242 60.754 24.705h716.95q35.293 0 60.754-
24.705t25.461-59.242-21.428-72.603-51.679-57.225q-6.554-4.033-18.907-10.84t-51.427-24.453-
79.409-30.755zm-221.84 25.72q-34.285 0-67.561-14.873t-60.754-40.335-51.175-60.502-40.083-
```

```
75.124-25.461-84.451-9.075-87.728q0-64.032 19.915-116.22t54.452-85.964 80.67-51.931 99.072-18.151 99.072 18.151 80.67 51.931 54.452 85.964 19.915 116.22q0 65.04-20.167 130.58t-53.948 116.72-81.426 83.443-98.568 32.268z"
            ></path>
          </svg>
        </div>
        <ul>
          <li>Profile</li>
          <li>
            <svg viewBox="0 0 1024 1024" class="profile-arrow">
              <path
                d="M476.455 806.696L95.291 425.532Q80.67 410.911 80.67 390.239t14.621-34.789 35.293-14.117 34.789 14.117L508.219 698.8l349.4-349.4q14.621-14.117 35.293-14.117t34.789 14.117 14.117 34.789-14.117 34.789L546.537 800.142q-19.159 19.159-38.318 19.159t-31.764-12.605z"
              ></path>
            </svg>
          </li>
        </ul>
      </div>
    </nav>
  </div>
</header>
<!-- HERO -->
<section class="hero">
  <div class="hero-container">
    <div class="hero-content">
      <h1 class="hero-title">Let the Music Speak!</h1>
      <a
href="https://accounts.spotify.com/en/login?continue=https:%2F%2Faccounts.spotify.com%2Fauthorize%3Fscope%3Dplaylist-read-private%2Bplaylist-read-collaborative%2Bplaylist-modify-public%2Buser-read-recently-played%2Bplaylist-modify-private%2Bugc-image-upload%2Buser-follow-modify%2Buser-follow-read%2Buser-library-read%2Buser-library-modify%2Buser-read-private%2Buser-read-email%2Buser-top-read%2Buser-read-playback-state%26response_type%3Dtoken%26redirect_uri%3Dhttp%253A%252F%252Flocalhost%253A3000%252Fcallback%26client_id%3D230be2f46909426b8b80cac36446b52a">Sign in<button
class="hero-button">LAUNCH WEB PLAYER</button></a>
    </div>
  </div>
</section>
<!-- ALBUMS -->
<section class="albums">
  <div class="albums-container">
    <div class="albums-content">
      <h2>Looking for music?</h2>
      <p>Start listening to the best new releases.</p>
    </div>
    <div class="albums-cards-container">
      <!-- Travis Scott -->
      <div class="album">
        <img src="images/travis-scott.jpg" />
        <div class="album-info">
          <h2>FRANCHISE (feat. Young Thug & M.I.A.)</h2>
          <h4>Travis Scott</h4>
          <a href="#">PLAY NOW</a>
```

```
      </div>
    </div>
    <!-- ZAYN -->
    <div class="album">
      <img src="images/zayn.jpg" />
      <div class="album-info">
        <h2>Better</h2>
        <h4>ZAYN</h4>
        <a href="#">PLAY NOW</a>
      </div>
    </div>
    <!-- Joji -->
    <div class="album">
      <img src="images/joji.jpg" />
      <div class="album-info">
        <h2>Nectar</h2>
        <h4>Joji</h4>
        <a href="#">PLAY NOW</a>
      </div>
    </div>
    <!-- Jennifer Lopez -->
    <div class="album">
      <img src="images/jennifer-lopez.jpg" />
      <div class="album-info">
        <h2>Pa Ti + Lonely</h2>
        <h4>Jennifer Lopez</h4>
        <a href="#">PLAY NOW</a>
      </div>
    </div>
    <!-- Machine Gun Kelly -->
    <div class="album">
      <img src="images/machine-gun-kelly.jpg" />
      <div class="album-info">
        <h2>Tickets To My Downfall</h2>
        <h4>Machine Gun Kelly</h4>
        <a href="#">PLAY NOW</a>
      </div>
    </div>
    <!-- Cold -->
    <div class="album">
      <img src="images/chris-stapleton.jpg" />
      <div class="album-info">
        <h2>Cold</h2>
        <h4>Chris Stapleton</h4>
        <a href="#">PLAY NOW</a>
      </div>
    </div>
  </div>
 </div>
</section>
<!-- FOOTER -->
<footer>
 <nav class="footer-nav">
  <div class="logo-footer">
   <a href="#"><img src="images/spotify-logo.svg" /></a>
```

```
</div>
<div class="top-links">
  <div class="company-links">
    <ul>
      <li>COMPANY</li>
      <li><a href="#">About</a></li>
      <li><a href="contact.html">Contact</a></li>
      <li><a href="#">For the Record</a></li>
    </ul>
  </div>
  <div class="communities-links">
    <ul>
      <li>COMMUNITIES</li>
      <li><a href="#">For artists</a></li>
      <li><a href="#">Developers</a></li>
      <li><a href="#">Advertising</a></li>
      <li><a href="#">Investors</a></li>
      <li><a href="#">Vendors</a></li>
    </ul>
  </div>
  <div class="useful-links">
    <ul>
      <li>USEFUL LINKS</li>
      <li><a href="#">Help</a></li>
      <li><a href="#">Web Player</a></li>
      <li><a href="#">Free Mobile App</a></li>
    </ul>
  </div>
</div>
<div class="social-icons">
  <ul>
    <li>
      <a href="#"><img src="images/instagram-icon.svg" /></a>
    </li>
    <li>
      <a href="#"><img src="images/twitter-icon.svg" /></a>
    </li>
    <li>
      <a href="#"><img src="images/facebook-icon.svg" /></a>
    </li>
  </ul>
</div>
<div class="country">
  <a href="#"><span>India</span><img src="images/India.svg"/></a>

</div>
<div class="bottom-links">
  <ul>
    <li><a href="#">Legal</a></li>
    <li><a href="#">Privacy Center</a></li>
    <li><a href="#">Privacy Policy</a></li>
    <li><a href="#">Cookies</a></li>
    <li><a href="#">Terms and Conditions</a></li>
    <li><a href="#">About Ads</a></li>
    <li><a href="#">Additional CA Privacy Disclosures</a></li>
```

```
        </ul>
        <span>&copy; Copyright<br>
          Shagun Deogharkar<br>
          Shagun Deogharka</span>
       </div>
     </nav>
   </footer>
  </body>
  <script src="./js/script.js"></script>
</html>
```

App.js

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import { fetchUser } from './actions/userActions';
import { setToken } from './actions/tokenActions';
import {
  playSong,
  stopSong,
  pauseSong,
  resumeSong,
} from './actions/songActions';
import Header from './components/Header';
import Footer from './components/Footer';
import UserPlaylists from './components/UserPlaylists';
import MainView from './components/MainView';
import ArtWork from './components/ArtWork';
import MainHeader from './components/MainHeader';
import SideMenu from './components/SideMenu';
import './App.css';

class App extends Component {
  static audio;

  componentDidMount() {
    let hashParams = {};
    let e,
      r = /([^&;=]+)=?([^&;]*)/g,
      q = window.location.hash.substring(1);
    while ((e = r.exec(q))) {
      hashParams[e[1]] = decodeURIComponent(e[2]);
    }

    if (!hashParams.access_token) {
      window.location.href =

'https://accounts.spotify.com/authorize?client_id=230be2f46909426b8b80cac36446b52a&scope=play
list-read-private%20playlist-read-collaborative%20playlist-modify-public%20user-read-recently-
played%20playlist-modify-private%20ugc-image-upload%20user-follow-modify%20user-follow-
read%20user-library-read%20user-library-modify%20user-read-private%20user-read-email%20user-
top-read%20user-read-playback-
state&response_type=token&redirect_uri=http://localhost:3000/callback';
```

```
  } else {
    this.props.setToken(hashParams.access_token);
  }
}

componentWillReceiveProps(nextProps) {
  if (nextProps.token) {
  this.props.fetchUser(nextProps.token);
  }

  if (this.audio !== undefined) {
    this.audio.volume = nextProps.volume / 100;
  }
}

stopSong = () => {
  if (this.audio) {
    this.props.stopSong();
    this.audio.pause();
  }
};

pauseSong = () => {
  if (this.audio) {
    this.props.pauseSong();
    this.audio.pause();
  }
};

resumeSong = () => {
  if (this.audio) {
    this.props.resumeSong();
    this.audio.play();
  }
};

audioControl = (song) => {
  const { playSong, stopSong } = this.props;

  if (this.audio === undefined) {
    playSong(song.track);
    this.audio = new Audio(song.track.preview_url);
    this.audio.play();
  } else {
    stopSong();
    this.audio.pause()
    ;
    playSong(song.track);
    this.audio = new Audio(song.track.preview_url);
    this.audio.play();
  }
};

render() {
  return (
    <div className="App">
```

```
        <div className="app-container">
          <div className="left-side-section">
            <SideMenu />
            <UserPlaylists />
            <ArtWork />
          </div>
          <div className="main-section">
            <Header />
            <div className="main-section-container">
              <MainHeader
                pauseSong={this.pauseSong}
                resumeSong={this.resumeSong
                }
              />{' '}
              <MainView
                pauseSong={this.pauseSong}
                resumeSong={this.resumeSong}
                audioControl={this.audioControl}
              />
            </div>
          </div>
          <Footer
            stopSong={this.stopSong}
            pauseSong={this.pauseSong}
            resumeSong={this.resumeSong
            }
            audioControl={this.audioControl}
          />
        </div>
      </div>
    );
  }
}

App.propTypes = {
  token:
  PropTypes.string,
  fetchUser: PropTypes.func,
  setToken: PropTypes.func,
  pauseSong: PropTypes.func,
  playSong: PropTypes.func,
  stopSong: PropTypes.func,
  resumeSong: PropTypes.func,
  volume: PropTypes.number,
};

const mapStateToProps = (state) => {
  return {
    token: state.tokenReducer.token,
    volume: state.soundReducer.volume,
  };
};

const mapDispatchToProps = (dispatch) => {
  return bindActionCreators(
    {
      fetchUser,
      setToken,
```

```
      playSong,
      stopSong,
      pauseSong,
      resumeSong,
    },
    dispatch
  );
};

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import { createStore, applyMiddleware } from "redux";
import thunk from "redux-thunk";
import { Provider } from "react-redux";
import reducers from "./reducers";
import App from "./App";

//create the redux store
const store = createStore(
reducers,
  window. REDUX_DEVTOOLS_EXTENSION &&
window. REDUX_DEVTOOLS_EXTENSION (),
  applyMiddleware(thunk)
);

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById("root")
);
```

albumAction.js

```
export const fetchAlbumsPending = () => {
  return {
    type: 'FETCH_ALBUMS_PENDING'
  };
};

export const fetchAlbumsSuccess = (albums) => {
  return {
    type: 'FETCH_ALBUMS_SUCCESS',
    albums
  };
};

export const fetchAlbumsError = () => {
  return {
    type: 'FETCH_ALBUMS_ERROR'
  };
};

export const fetchAlbums = (accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/me/albums`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    dispatch(fetchAlbumsPending());

    fetch(request).then(res => {
      return res.json();
    }).then(res => {
      dispatch(fetchAlbumsSuccess(res.items));
    }).catch(err => {
      dispatch(fetchAlbumsError(err));
    });
  };
};
```

artistAction.js

```
export const fetchArtistsPending = () => {
  return {
    type: 'FETCH_ARTISTS_PENDING'
  };
};

export const fetchArtistsSuccess = (artists) => {
  return {
    type: 'FETCH_ARTISTS_SUCCESS',
    artists
  };
};

export const fetchArtistsError = () => {
  return {
    type: 'FETCH_ARTISTS_ERROR'
  };
};

export const fetchArtists = (accessToken, artistIds) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/artists?ids=${artistIds}`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    dispatch(fetchArtistsPending());

    fetch(request).then(res => {
      return res.json();
    }).then(res => {
      dispatch(fetchArtistsSuccess(res));
    }).catch(err => {
      dispatch(fetchArtistsError(err));
    });
  };
};


export const fetchArtistSongsPending = () => {
  return {
    type: 'FETCH_ARTIST_SONGS_PENDING'
  };
};

export const fetchArtistSongsSuccess = (songs) => {
  return {
    type: 'FETCH_ARTIST_SONGS_SUCCESS',
```

```
    songs
  };
};

export const fetchArtistSongsError = () => {
  return {
    type: 'FETCH_ARTIST_SONGS_ERROR'
  };
};

export const fetchArtistSongs = (artistId, accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/artists/${artistId}/top-
tracks?country=US`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    dispatch(fetchArtistSongsPending());

    fetch(request).then(res => {
      if(res.statusText === "Unauthorized") {
      window.location.href = './';
      }
      return res.json();
    }).then(res => {
      // map the response to match that returned from get song request
      res.items = res.tracks.map(item => {
        return {
          track: item
        };
      });

      dispatch(fetchArtistSongsSuccess(res.items));
    }).catch(err => {
      dispatch(fetchArtistSongsError(err));
    });
  };
};


export const setArtistIds = (artistIds) => {
  return {
    type: 'SET_ARTIST_IDS',
    artistIds
  };
};
```

browseAction.js

```
export const fetchCategoriesSuccess = (categories) => {
  return {
    type: 'FETCH_CATEGORIES_SUCCESS',
    categories
  };
};

export const fetchCategoriesError = () => {
  return {
    type: 'FETCH_CATEGORIES_ERROR'
  };
};

export const fetchCategories = (accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/browse/categories`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });
    fetch(request).then(res => {
      return res.json();
    }).then(res => {
      dispatch(fetchCategoriesSuccess(res.categories));
    }).catch(err => {
      dispatch(fetchCategoriesError(err));
    });
  };
};

export const fetchNewReleasesSuccess = (newReleases) => {
  return {
    type: 'FETCH_NEW_RELEASES_SUCCESS',
    newReleases
  };
};

export const fetchNewReleasesError = () => {
  return {
    type: 'FETCH_NEW_RELEASES_ERROR'
  };
};

export const fetchNewReleases = (accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/browse/new-releases`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });
    fetch(request).then(res => {
      return res.json();
```

```
    }).then(res => {
      dispatch(fetchNewReleasesSuccess(res.albums));
    }).catch(err => {
      dispatch(fetchNewReleasesError(err));
    });
  };
};

export const fetchFeaturedSuccess = (featured) => {
  return {
    type: 'FETCH_FEATURED_SUCCESS',
    featured
  };
};

export const fetchFeaturedError = () => {
  return {
    type: 'FETCH_FEATURED_ERROR'
  };
};

export const fetchFeatured = (accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/browse/featured-playlists`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });
    fetch(request).then(res => {
      return res.json();
    }).then(res => {
      dispatch(fetchFeaturedSuccess(res.playlists));
    }).catch(err => {
      dispatch(fetchFeaturedError(err));
    });
  };
};
```

playlistAction,js

```
import uniqBy from 'lodash/uniqBy';

export const fetchPlaylistMenuPending = () => {
  return {
    type: 'FETCH_PLAYLIST_MENU_PENDING'
  };
};

export const fetchPlaylistMenuSuccess = (playlists) => {
  return {
    type: 'FETCH_PLAYLIST_MENU_SUCCESS',
    playlists
  };
};

export const fetchPlaylistMenuError = () => {
  return {
    type: 'FETCH_PLAYLIST_MENU_ERROR'
  };
};

export const addPlaylistItem = (playlist) => {
  return {
    type: 'ADD_PLAYLIST_ITEM',
    playlist
  };
};

export const fetchPlaylistsMenu = (userId, accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/users/${userId}/playlists`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    dispatch(fetchPlaylistMenuPending());

    fetch(request).then(res => {
      if(res.statusText === "Unauthorized") {
      window.location.href = './';
      }
      return res.json();
    }).then(res => {
      dispatch(fetchPlaylistMenuSuccess(res.items));
    }).catch(err => {
      dispatch(fetchPlaylistMenuError(err));
    });
  };
};


export const fetchPlaylistSongsPending = () => {
```

```
  return {
    type: 'FETCH_PLAYLIST_SONGS_PENDING'
  };
};

export const fetchPlaylistSongsSuccess = (songs) => {
  return {
    type: 'FETCH_PLAYLIST_SONGS_SUCCESS',
    songs
  };
};

export const fetchPlaylistSongsError = () => {
  return {
    type: 'FETCH_PLAYLIST_SONGS_ERROR'
  };
};

export const fetchPlaylistSongs = (userId, playlistId, accessToken) => {
  return dispatch => {
    const request = new
Request(`https://api.spotify.com/v1/users/${userId}/playlists/${playlistId}/tracks`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });
    dispatch(fetchPlaylistSongsPending());
    fetch(request).then(res => {
      return res.json();
    }).then(res => {
      //remove duplicate tracks
      res.items = uniqBy(res.items, (item) => {
        return item.track.id;
      });
      dispatch(fetchPlaylistSongsSuccess(res.items));
    }).catch(err => {
      dispatch(fetchPlaylistSongsError(err));
    });
  };
};
```

songAction.js

```
import uniqBy from 'lodash/uniqBy';
import { setArtistIds } from './artistActions';

export const fetchSongsPending = () => {
  return {
    type: 'FETCH_SONGS_PENDING'
  };
};

export const fetchSongsSuccess = (songs) => {
  return {
    type: 'FETCH_SONGS_SUCCESS',
    songs
  };
};

export const fetchSongsError = () => {
  return {
    type: 'FETCH_SONGS_ERROR'
  };
};

export const fetchSongs = (accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/me/tracks?limit=50`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    dispatch(fetchSongsPending());

    fetch(request).then(res => {
      if(res.statusText === "Unauthorized") {
      window.location.href = './';
      }
      return res.json();
    }).then(res => {
      // get all artist ids and remove duplicates
      let artistIds = uniqBy(res.items, (item) => {
        return item.track.artists[0].name;
      }).map(item => {
        return item.track.artists[0].id;
      }).join(',');
      dispatch(setArtistIds(artistIds));
      dispatch(fetchSongsSuccess(res.items));
    }).catch(err => {
```

```
      dispatch(fetchSongsError(err));
    });
  };
};

export const searchSongsPending = () => {
  return {
    type: 'SEARCH_SONGS_PENDING'
  };
};

export const searchSongsSuccess = (songs) => {
  return {
    type: 'SEARCH_SONGS_SUCCESS',
    songs
  };
};

export const searchSongsError = () => {
  return {
    type: 'SEARCH_SONGS_ERROR'
  };
};

export const searchSongs = (searchTerm, accessToken) => {
  return dispatch => {
    const request = new
Request(`https://api.spotify.com/v1/search?q=${searchTerm}&type=track`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken,
        'Accept': 'application/json'
      })
    });

    dispatch(searchSongsPending());

    fetch(request).then(res => {
      if(res.statusText === "Unauthorized") {
      window.location.href = './';
      }
      return res.json();
    }).then(res => {
      res.items = res.tracks.items.map(item => {
        return {
          track: item
        };
      });
      dispatch(searchSongsSuccess(res.items));
    }).catch(err => {
      dispatch(fetchSongsError(err));
```

```
    });
  };
};

export const fetchRecentlyPlayedPending = () => {
  return {
    type: 'FETCH_RECENTLY_PLAYED_PENDING'
  };
};

export const fetchRecentlyPlayedSuccess = (songs) => {
  return {
    type: 'FETCH_RECENTLY_PLAYED_SUCCESS',
    songs
  };
};

export const fetchRecentlyPlayedError = () => {
  return {
    type: 'FETCH_RECENTLY_PLAYED_ERROR'
  };
};

export const fetchRecentlyPlayed = (accessToken) => {
  return dispatch => {
    const request = new Request(`https://api.spotify.com/v1/me/player/recently-played`, {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });
    dispatch(fetchRecentlyPlayedPending());
    fetch(request).then(res => {
      return res.json();
    }).then(res => {
      //remove duplicates from recently played
      res.items = uniqBy(res.items, (item) => {
      return item.track.id;
      });
      dispatch(fetchRecentlyPlayedSuccess(res.items));
    }).catch(err => {
      dispatch(fetchRecentlyPlayedError(err));
    });
  };
};

export const playSong = (song) => {
  return {
    type: 'PLAY_SONG',
```

```
    song
  };
};

export const stopSong = () => {
  return {
    type: 'STOP_SONG'
  };
};

export const pauseSong = () => {
  return {
    type: 'PAUSE_SONG'
  };
};

export const resumeSong = () => {
  return {
    type: 'RESUME_SONG'
  };
};

export const increaseSongTime = (time) => {
  return {
    type: 'INCREASE_SONG_TIME',
    time
  };
};

export const updateViewType = (view) => {
  return {
    type: 'UPDATE_VIEW_TYPE',
    view
  };
};
```

UserAction.js

```
export const fetchUserSuccess = (user) => {
  return {
    type: 'FETCH_USER_SUCCESS',
    user
  };
};

export const fetchUserError = () => {
  return {
    type: 'FETCH_USER_ERROR'
  };
};
export const fetchUser = (accessToken) => {
  return dispatch => {
    const request = new Request('https://api.spotify.com/v1/me', {
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    fetch(request).then(res => {
      // send user back to homepage if no token
      if(res.statusText === "Unauthorized") {
      window.location.href = './';
      }
      return res.json();
    }).then(res => {
      dispatch(fetchUserSuccess(res));
    }).catch(err => {
      dispatch(fetchUserError(err));
    });
  };
};

export const addSongToLibrarySuccess = (songId) => {
  return {
    type: 'ADD_SONG_TO_LIBRARY_SUCCESS',
    songId
  };
};

export const addSongToLibraryError = () => {
  return {
    type: 'ADD_SONG_TO_LIBRARY_ERROR'
  };
};
```

```
export const addSongToLibrary = (accessToken, id) => {

  return dispatch => {

    const request = new Request(`https://api.spotify.com/v1/me/tracks?ids=${id}`, {
      method: 'PUT',
      headers: new Headers({
        'Authorization': 'Bearer ' + accessToken
      })
    });

    fetch(request).then(res => {
      if(res.ok) {
        dispatch(addSongToLibrarySuccess(id));
      }
    }).catch(err => {
      dispatch(addSongToLibraryError(err));
    });
  };
};
```

5.2 TESTING APPROACH

5.2.1 Unit Testing

UNIT TESTING is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing. Unit Testing is the first level of software testing and is performed prior to Integration Testing. It is normally performed by software developers themselves or their peers. In rare cases, it may also be performed by independent software testers.

5.2.2 Integrated Testing

INTEGRATION TESTING is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing. Any of Black Box Testing, White Box Testing and Gray Box Testing methods can be used. Normally, the method depends on your definition of ‘unit’.

TEST RESULT:

Sr. No.	TEST CONDITION	STEPS OR PROCEDURE	INPUT TEST DATA	EXCEPTED RESULT	ACTUAL OUTPUT	PASS/ FAIL
1	Check whether product is added properly to specific category	Add product from admin dashboard and it will display in user dashboard	Admin Dashboard Add product wood guitar in guitar category With all details.	Guitar.jsp page Here guitar images and all details are shown .	Guitar.jsp page Productname: wood guitar Price:4569 Data is displayed in proper layout	Pass
2	Check whether one or more products are added in the cart or not	User can check products and click on add to cart button it will add products in cart, user can add one or more products in cart	User dashboard User check product and add the product in cart and go back for a adding more products	User dashboard (cart details) Here all the products are displayed in cart table	User dashboard (cart details) Here are all the products are displayed which is added from the user and the total amount is displayed	Pass
3	Check whether after user has done payment user can get mail or not	After user has done the payment process user can get mail from admin about the delivery details.	After admin get payment from user admin send the delivery details mail to user.	User get mail from admin about delivery details	User get mail from admin about delivery details	Pass

5.3 MODIFICATION AND IMPROVEMENT

Various changes were made after performing the testing, the agile methodology led the changes to be made easily in the websites, the changes are listed below.

For increasing the security more features like encryption and OTP are added to the website. MD5 encryptions allow storing the password in the encrypted format which allows more security. OTP allows to check whether the user is genuine or not by sending the OTP to the email and verifying it.

Chapter 6

Conclusion And Future Work

The Music Mania website future work is that it will be updated directly proportional with time. The user will have to constantly their work in order to match with trendy designs. Well talking about future, the website will have a feature where the user add their favorite song in playlist without contacting the web developer for it the each time. The users might get a user panel through which they can easily update their playlist and wherever they want which playlist share on cross platforms.

The website will be updated with many other payment options and new wallet and methods will be added for buying the packages instead of email contact

From this i would like to conclude this topic here by thanking all my professors who have always been there for me as my mentor and helped me out in developing of this beautiful music website.