## **MUSIFY**

## A PROJECT REPORT

*Submitted by*

# MITHIL MUDALIYAR [ RA2211003011476]

# KALPESH BONDE [ RA2211003011502]

# GITANSH PISE [ RA2211003011504]

*Under the Guidance of*

# DR. ARUNA M

Assistant Professor, Department of Computing Technologies

*In partial fulfilment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY** in

**COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTING TECHNOLOGY**

# COLLEGE OF ENGINEERING AND TECHNOLOGY

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

**NOV 2023**



### SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR

### (Under Section 3 of UGC Act,1956)

**BONAFIDE CERTIFICATE**

Certified that Mini project titled “**MUSIFY**” is the bonafide work of **MITHIL MUDALIYAR [RA2211003011476], KALPESH BONDE [RA2211003011502] and GITANSH PISE [RA2211003011504]** **who** carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

**SIGNATURE**

Dr. M Pushpalatha

**HEAD OF DEPARTMENT**

Professor & Head

Department of Computing Technologies

**SIGNATURE**

Dr.Aruna M

**GUIDE**

Assistant Professor

Department of Computing Technologies

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| ***C.NO.*** | ***TITLE*** | ***PAGE NO.*** |
|  | Abstract | viii |
| 1. | INTRODUCTION | 9 |
|  | 1.1 General | 9 |
|  | 1.2 Purpose  1.3 Project Scope | 9 |
| 2 | LITERATURE REVIEW | 10 |
| 3 | SYSTEM ARCHITECTURE | 11 |
| 4 | PROPOSED METHODOLOGY | 14 |
|  |  |  |
| 5  6 | IMPLEMENTATION AND TESTING  CODE | 15 |
| 7 | RESULTS | 18 |
| 8  9 | CONCLUSION  FUTURE SCOPE  REFERENCES | 22  23    25 |
|  |  |  |
|  |  |  |

**ABSTRACT**

The In the rapidly evolving digital age, music consumption and interaction have undergone a transformative shift, primarily driven by web-based music applications. "Musify" emerges as a groundbreaking web-based music streaming application, created as a core component of an operating systems project. This application represents a substantial leap forward, redefining the way we experience and engage with music in a digital context.

The core purpose of Musify is to offer users a sophisticated and immersive music listening experience, seamlessly integrating into their digital lives. With the global music streaming landscape teeming with millions of users seeking a platform that seamlessly caters to their music needs, Musify endeavors to fill this niche by providing an elegant, feature-rich solution. Its aim is to provide an exceptional and user-centric music streaming experience that stands out in this dynamic and competitive landscape.

Within the scope of this project, we delve deep into the intricate interplay between Musify and the operating systems that underpin its functionality. Operating systems serve as the fundamental foundation upon which this web-based music application stands, ensuring robust performance, resource efficiency, and airtight security. This project sheds light on the symbiotic relationship between Musify and the operating systems that support it, unveiling the complexities of this synergy.

Musify is more than just an application; it serves as a testament to the convergence of music, technology, and user experience. It exemplifies the boundless possibilities that this fusion brings to the world of digital music consumption. Musify's development journey, as explored in this project, showcases how the seamless integration of music and technology can deliver a remarkable and immersive user experience, shaping the future of music streaming applications.

# CHAPTER 1

# INTRODUCTION

* 1. **General:**

In the digital age, the way we consume and interact with music has evolved dramatically, thanks to web-based music applications. "Musify" represents a significant leap in this evolution. It is a sophisticated web-based music application designed to offer users a seamless and immersive music listening experience.

**1.2 Purpose:**

The global music streaming landscape is vibrant and dynamic, with millions of users seeking a platform that seamlessly integrates into their digital lives. Musify endeavours to fill this niche by providing an elegant and feature-rich solution. It aims to redefine the way people engage with music by delivering a high-quality music streaming experience.

**1.3 Project Scope:**

Within the framework of this project, we delve into the intricate interplay between Musify and operating systems, exploring its design, development, and operational aspects. The realm of operating systems plays a pivotal role in Musify's journey. It is the foundation upon which this web-based music application stands, ensuring robust performance, resource efficiency, and security. As we explore Musify's development, we will underscore the symbiotic relationship between the application and the operating systems that support it, unveiling the intricacies of this synergy. Musify is not just an application; it is a testament to the intersection of music, technology, and user experience, and it exemplifies the boundless possibilities that this fusion holds in the world of digital music consumption.

# CHAPTER 2

# LITERATURE REVIEW

The web-based music app landscape has rapidly evolved, impacting Musify's development. Key areas explored in the literature include web design, user interfaces, audio streaming, and operating system integration.

**Web Application Development:** The foundation of web-based music applications hinges on efficient web development principles. Studies in this area emphasize the importance of adhering to best practices in web application development, including responsive design to ensure a consistent user experience across diverse devices. These principles underscore the need for Musify to provide a seamless and accessible interface to users.

**User Interface Design:** User interface design is pivotal in the success of any web application, and music applications are no exception. Research in this field has elucidated the significance of user-centred design, emphasizing aesthetics, simplicity, and ease of use. Musify's interface is meticulously crafted to cater to these principles, creating an immersive environment for users to explore music effortlessly.

**Operating System Integration**: The integration of operating systems in web applications is a crucial component that sets the foundation for Musify's performance and reliability. It encompasses optimizing resource utilization and ensuring secure interactions with the underlying OS. Research underscores the need for Musify to leverage the strengths of various operating systems, enhancing compatibility and stability.

8

# CHAPTER 3

# SYSTEM ARCHITECTURE

**ARCHITECTURE:**

Designing the system architecture for a music streaming application like "Musify" in an operating system project can be a complex task. It involves multiple components and layers that work together to provide the desired functionality. Here's a high-level overview of a typical system architecture for such an application:

**USER INTERFACE (UI):**

The user interface layer is responsible for presenting the application to users. It includes components like the app interface, web interface, and mobile app interfaces.

The UI layer should provide user-friendly features for browsing, searching, and playing music, managing playlists, and user account management.

**APPLICATION LOGIC:**

The application logic layer handles user requests and processes them.

It manages user authentication, user profiles, playlists, and music recommendations.

It communicates with the database layer to fetch and store user data and music information.

**MUSIC STREAMING AND PLAYBACK:**

This layer is responsible for streaming and playing music.

It should support features like streaming quality control, audio format compatibility, and real-time audio playback.

**BACKEND SERVICES:**

Backend services manage the core functionality of the application.

This layer includes components like user management, playlist management, and music catalog management.

**DATABASE:**

The database layer stores various types of data, including user profiles, playlists, music metadata, and user interactions.

**EXTERNAL APIS AND DATA SOURCES:**

Integration with external APIs, such as music licensing services, music recommendation engines, and payment gateways, is essential for a music streaming application.

These APIs provide access to music libraries and help enhance the user experience.

**CONTENT DELIVERY:**

To ensure low latency and fast music streaming, a content delivery network (CDN) may be used to distribute media files to users across the globe.

**SECURITY:**

Security is paramount to protect user data, payment information, and prevent unauthorized access.

Implement user authentication, encryption, and secure communication protocols.

**SCALABILITY AND LOAD BALANCING:**

To handle a potentially large number of users, the system should be designed for scalability. Load balancing mechanisms can distribute incoming traffic evenly across multiple servers or instances.

**CACHING:**

Implement caching mechanisms to reduce the load on the database and enhance performance. Popular and frequently accessed music files and data can be cached.

**MONITORING AND ANALYTICS:**

Implement monitoring and analytics tools to track system performance, user behavior, and issues in real-time.

**DEPLOYMENT:**

Choose a suitable hosting environment, such as cloud services like AWS, Azure, or Google Cloud, for deploying the application components.

**HIGH AVAILABILITY AND DISASTER RECOVERY:**

Ensure high availability by using redundant servers and disaster recovery plans to minimize downtime.

# CHAPTER 4

# PROPOSED METHODOLOGY

The development of Musify is a comprehensive process that encompasses a series of well-defined steps to ensure its successful implementation and operation within the context of operating systems. Here's a more detailed breakdown of the methodology:

**System Architecture:** The next step involves the design of a robust and scalable system architecture. Musify's architecture is meticulously crafted to support a growing user base and evolving features.

Front-end Development: The user interface is a critical component of Musify, and its development involves creating an intuitive and visually appealing design. Responsive web design principles are applied to ensure that Musify can adapt to various screen sizes and resolutions.

**Back-end Development:** The back-end development phase is where the core functionalities are implemented. This includes creating user authentication and authorization systems, setting up databases to store user profiles and music metadata, and integrating audio streaming capabilities

**Operating System Integration**: To optimize resource management and enhance compatibility across different operating systems, Musify incorporates operating system-specific strategies.

The proposed methodology for Musify ensures a high-quality, cross-compatible web-based music app that offers an exceptional streaming experience with system stability and security.

# CHAPTER 5

# IMPLEMENTAION AND TESTING

**Implementation Phase:**

**Technology Stack Selection:**

Choose the programming languages, frameworks, and tools to build the application. Consider compatibility with the chosen operating systems.

**User Interface (UI) Development:**

Create the user interface for Musify, including features for browsing, searching, playlists, and audio playback.

Ensure a responsive design for web and mobile platforms.

**Application Logic:**

Develop the backend logic for user authentication, user profiles, playlists, and music recommendation algorithms.

Implement logic for streaming music, controlling playback, and managing user interactions.

**Database Setup:**

Design and set up the database to store user data, music metadata, playlists, and user interactions.

Implement efficient database queries to handle the application's data requirements.

**Music Streaming and Playback:**

Implement the music streaming and playback functionality. This includes audio codec support, quality control, buffering, and real-time audio playback.

**Backend Services:**

Develop backend services for user management, playlist management, and music catalog management.

Integrate with external APIs for licensing, rights management, and recommendations.

**Security Implementation:**

Implement security measures, including user authentication, encryption, and secure communication protocols.

**Scalability and Performance Optimization:**

Ensure that the application can handle a large number of users by designing for scalability and implementing load balancing mechanisms.

**Content Delivery:**

Set up a Content Delivery Network (CDN) for distributing media files to users, ensuring low latency and fast music streaming.

**Deployment and Continuous Integration/Continuous Deployment (CI/CD):**

Choose a hosting environment and deploy the application components.

Implement CI/CD pipelines for automated deployment and scaling.

**Testing Phase:**

**Unit Testing:**

Conduct unit tests for individual components, ensuring that they function correctly and handle various scenarios.

**Integration Testing:**

Test the interaction between different modules and services to ensure that they work together seamlessly.

**User Acceptance Testing (UAT):**

Involve end-users or testers to perform UAT to verify if the application meets user expectations and requirements.

**Security Testing:**

Perform security testing to identify and address vulnerabilities, including penetration testing and code reviews.

**Compatibility Testing:**

Test the application on various operating systems, browsers, and devices to ensure cross-platform compatibility.

**Usability Testing:**

Evaluate the user interface and user experience to identify any usability issues and make improvements.

**Regression Testing:**

Continuously test the application when changes are made to ensure that new updates do not introduce new issues.

**Stress Testing:**

Assess the application's ability to handle extreme conditions and peak loads.

**Load Testing:**

Test the application's response times and stability under heavy user loads

.

# CHAPTER 6

# CODE

**Components:**

**1-App.js: -**

import React from 'react'

import { Global, css, jsx } from '@emotion/core'

import MusicPlayer from './index'

/\*\*

\* @function App

\*/

const App = () => (

<>

<Global styles={GlobalCSS} />

<MusicPlayer />

</>

)

const GlobalCSS = css`

\* {

box-sizing: border-box;

font-family: 'Signika', sans-serif;

}

html,

body,

.app {

margin: 0;

height: 100%;

width: 100%;

}

a {

text-decoration: none;

}

ul {

margin: 0;

list-style: none;

padding: 0;

}

`export default App

**2-Content.js :-**

/\* eslint-disable prettier/prettier \*/

/\*\* @jsx jsx \*/

import React, { useCallback, useContext, useState } from 'react'

import { css, jsx } from '@emotion/core'

import { StoreContext } from './index'

//import "./Content.css"

import Modal from './Modal'

import Toast from './Toast'

import sym from '../img/logo10.png'

const Content = () => {

const { state, dispatch } = useContext(StoreContext)

const[searchTerm,setSearchTerm] = useState('');

const [toast, setToast] = useState('')

const [playlistSelect, setPlayListSelect] = useState('')

const [playVisibleId, setPlayVisibleId] = useState(

false

)

//serial number counter

var i=0

const currentPlaylist = state.currentPlaylist

const playlists = Object.keys(state.playlists).filter(

list => !['home', 'favorites'].includes(list)

)

const songIds = Array.from(

state.playlists[currentPlaylist]

)

const handleSelect = useCallback(e => {

setPlayListSelect(e.target.value)

})

const handleSearch = useCallback(e => {

setSearchTerm(e.target.value)

})

return (

<div className="Content" css={CSS}>

<div className="Search"><input type="text" value={searchTerm} placeholder ="Search for a song..." onChange={handleSearch} />

</div>

<div className="playlist-title" style={{marginBottom:40}}>

{currentPlaylist}

</div>

{songIds.length <= 0 ? (

<p style={{ marginTop: 10 }}>

Your playlist is empty. Start by adding some

songs!

</p>

) : (

<table>

<thead>

<tr>

<td className="sl" style={{paddingLeft :30}}>#</td>

<td />

<td>Title</td>

<td>Artist</td>

<td>Duration</td>

<td />

<td/>

</tr>

</thead>

<tbody>

{

songIds.filter((id)=>{

if(searchTerm == ''){

return id

} else if(state.media[id].title.toLowerCase().includes(searchTerm.toLowerCase())){

return id

}

} ).map(id => {

const {

title,

artist,

length

} = state.media[id]

const isFavorite = state.playlists.favorites.has(

id

)

i=i+1

return (

<tr key={id}

style={{ color: state.playing && state.currentSongId == id ? "#D4AF37" : "white", fontWeight: state.playing && state.currentSongId == id ? "bold" : "normal"}}

//mouse hover

onMouseEnter={() =>

setPlayVisibleId(id)

}

onMouseLeave={() =>

setPlayVisibleId('')

}>

<td

style={{ width: 75, paddingLeft: 30 }}

>

<PlayPause

playing={state.playing}

songId={id}

isCurrentSong={

state.currentSongId === id

}

visible={playVisibleId === id}

num={i}

/>

<span style={{ marginRight: 50 }} />

</td>

<td><img style={{width:45,height:45,marginRight:5,borderRadius:70}} src={sym} /></td>

<td >{title}</td>

<td>{artist}</td>

<td>{length}</td>

<td>

<Favorite

isFavorite={isFavorite}

songId={id}

/>

<span style={{ marginRight: 10 }} />

</td>

<td

style={{ width: 75, paddingLeft: 5 }}

>

<AddSong

visible={playVisibleId === id}

songId={id}

/>

</td>

</tr>

)

})}

</tbody>

</table>

)}

<Modal

show={state.addToPlaylistId}

close={() => {

dispatch({ type: 'ABORT\_ADD\_TO\_PLAYLIST' })

}}

>

<div style={{ textAlign: 'center' }}>

<div style={{ fontSize: 18, marginBottom: 20 }}>

Add To Playlist

</div>

{playlists.length < 1 ? (

<>

<p>

You don't have any custom playlists yet.

Start by creating one in the sidebar menu!

</p>

<div style={{ marginTop: 15 }}>

<button onClick={() => {

dispatch({ type: 'ABORT\_ADD\_TO\_PLAYLIST' })

}}>Ok</button>

</div>

</>

) : (

<>

<select

value={playlistSelect}

onChange={handleSelect}

style={{

borderRadius:60,

fontSize: 16,

textTransform: 'capitalize',

width: 115,

height: 25

}}

>

<option value="">Choose</option>

{playlists.map(list => (

<option

key={list}

value={list}

disabled={state.playlists[list].has(

state.addToPlaylistId

)}

>

{list}

</option>

))}

</select>

<div style={{ marginTop: 20 }}>

<button

onClick={() => {

if (playlistSelect === '') return

dispatch({

type: 'SAVE\_TO\_PLAYLIST',

playlist: playlistSelect

})

setToast(

'Successfully added to your playlist.'

)

setPlayListSelect('')

}}

>

Save

</button>

</div>

</>

)}

</div>

</Modal>

<Toast toast={toast} close={() => setToast('')} />

</div>

)

}

const Favorite = ({ isFavorite, songId }) => {

const { dispatch } = useContext(StoreContext)

return isFavorite ? (

<i

className="fa fa-heart"

onClick={() => dispatch({ type: 'REMOVE\_FAVORITE', songId })}

/>

) : (

<i

className="fa fa-heart-o"

onClick={() => dispatch({ type: 'ADD\_FAVORITE', songId })}

/>

)

}

const PlayPause = ({ playing, songId, isCurrentSong, visible, num }) => {

const { dispatch } = useContext(StoreContext)

const style = { visibility: visible ? 'visible' : 'hidden' }

const number = num

if(visible){

if (isCurrentSong && playing) {

return (

<i

className="fa fa-pause"

onClick={() => dispatch({ type: 'PAUSE' })}

style={style}

/>

)

} else {

return (

<i

className="fa fa-play"

onClick={() => dispatch({ type: 'PLAY', songId })}

style={style}

/>

)

}

}else{

return(

number

)

}

}

const AddSong =({visible,songId}) =>{

const { dispatch } = useContext(StoreContext)

const style = { visibility: visible ? 'visible' : 'hidden' }

return(

<i

className="fa fa-plus"

onClick={() => {

dispatch({

type: 'ADD\_TO\_PLAYLIST',

songId

})

}}

style={style}

/>

)

}

const CSS = css`

width: calc(100% - 200px);

height: calc(100% - 75px);

padding: 50px;

background-image:linear-gradient(#000e34,#2C041C); // \*\*

padding-top: 10px;

overflow-y: scroll;

overflow-x:hidden;

color: white;// #00203FFF;/white; \*/ /#00203FFF; / /\*white;/

::-webkit-scrollbar {

width: 15px;

}

::-webkit-scrollbar-thumb {

//background: #00203FFF; // background: #00103f;

background-image:linear-gradient(#000000,#2C041C,#000000);

}

.playlist-title {

font-size: 30px;

font-weight: bold;

text-transform: capitalize;

font-family:Arial;

}

table {

border-collapse: collapse;

width: 100%;

margin-top: 15px;

font-size: 20px;

}

/\*table tr {

border-bottom: 1px solid #00203FFF;

}\*/

table td {

padding: 10px 0;

//top and bottom - 10px

//right and left - 0px

}

table thead {

//border-bottom: 2.5px solid #00203FFF;

border-bottom: 2.5px solid white;

font-weight: 600;

}

tbody:before {

line-height:1.5em;

content:"-";

color:transparent; /\* to hide text \*/

display:block;

}

//nesting

table thead td {

text-transform: uppercase;

padding-bottom:20px;

}

i {

cursor: pointer;

}

button {

background-color: #5d093b ;

color: white;

padding: 12.5px 30px;

border-radius: 25px;

text-transform: uppercase;

font-weight: bold;

font-size: 13px;

border: none;

cursor: pointer;

}

thead tr:hover td{ background: transparent !important;

color: white !important; }

tr:hover td { background: #46072d !important;

color:#D4AF37 !important; }

.Search{

/\* position:absolute;

left: 50%;

top: 5%;

transform: translate(-50%, -50%);

padding: 20px; \*/

position:relative;

height: 100px;

left:87px;

padding: 20px;

}

input {

width: 400px;

padding: 0 20px;

}

input,

input::-webkit-input-placeholder {

font-size: 20px;

line-height: 2;

border-radius: 60px;

}

`

export default Content

**3-index.js:-**

/\* eslint-disable prettier/prettier \*/

/\*\* @jsx jsx \*/

import React, { createContext, useEffect, useReducer, useRef } from 'react'

import { css, jsx } from '@emotion/core'

import { initialState, reducer } from '../reducers'

//import Topbar from './Topbar'

import Sidebar from './Sidebar'

import Content from './Content'

import Playbar from './Playbar'

//import Search from './Search'

export const StoreContext = createContext(null)

const MusicPlayer = () => {

const [state, dispatch] = useReducer(reducer, initialState)

const audioRef = useRef()

useEffect(() => {

if (state.playing) {

audioRef.current.load()

audioRef.current.play()

} else {

audioRef.current.pause()

}

}, [state.playing, state.currentSongId])

useEffect(() => {

audioRef.current.volume = state.volume

}, [state.volume])

useEffect(() => {

audioRef.current.currentTime = state.currentTime

}, [state.currentTime])

//console.log(audioRef);

const song = state.media[state.currentSongId]

return (

<StoreContext.Provider value={{ state, dispatch }}>

<div css={CSS}>

{/\* <Topbar /> \*/}

<Sidebar />

<Content />

<Playbar />

<audio

ref={audioRef}

src={

song && song.title

? `./media/${song.title} - ${song.artist}.mp3`

: ''

}

onLoadedMetadata={() =>

dispatch({

type: 'SET\_DURATION',

duration: audioRef.current.duration

})

}

onTimeUpdate={e =>

dispatch({ type: 'SET\_CURRENT\_TIME', time: e.target.currentTime })

}

onEnded={() =>

dispatch({

type: 'SET\_NEXT'

})

}

/>

</div>

</StoreContext.Provider>

)

}

const CSS = css`

height: 100%;

width: 100%;

display: flex;

position: relative;

color: white;

`

export default MusicPlayer

**4-Modal.js :-**

/\*\* @jsx jsx \*/

import React from 'react'

import { css, jsx } from '@emotion/core'

const Modal = ({ children, show, close }) => {

if (!show) return null

return (

<div className="Modal" css={CSS}>

<div className="modal-content">

<i className="fa fa-times" onClick={close} />

{children}

</div>

</div>

)

}

const CSS = css`

height: 100vh;

width: 100vw;

background: rgba(0, 0, 0, 0.75);

position: absolute;

top: 0;

right: 0;

display: flex;

align-items: center;

justify-content: center;

.modal-content {

position: relative;

width: 400px;

background-image:linear-gradient(#000e34,#2C041C);

border-radius: 4px;

padding: 25px;

}

i {

position: absolute;

right: 15px;

top: 15px;

cursor: pointer;

}

`

export default Modal

**5-Playbar.js :-**

/\*\* @jsx jsx \*/

import React, { useContext, useCallback } from 'react'

import { css, jsx } from '@emotion/core'

import { StoreContext } from './index'

import logo from '../img/logo10.png'

const formatTime = inputSeconds => {

let seconds = Math.floor(inputSeconds % 60)

if (seconds < 10) seconds = `0${seconds}`

const minutes = Math.floor(inputSeconds / 60)

return `${minutes}:${seconds}`

}

const Playbar = () => {

const { state, dispatch } = useContext(StoreContext)

const song = state.media[state.currentSongId]

if (!song) {

return <div className="Playbar" css={CSS} />

}

const playOrPause = (e =>

state.playing ? dispatch({ type: 'PAUSE'}) : dispatch({ type: 'PLAY' })

)

const setVolume = useCallback(e =>

dispatch({ type: 'SET\_VOLUME', volume: e.target.value })

)

const setTime = useCallback(e =>

dispatch({ type: 'SET\_CURRENT\_TIME', time: e.target.value })

)

const setNext = useCallback(e =>

dispatch({ type: 'SET\_NEXT' })

)

const setPrev = useCallback(e =>

dispatch({ type: 'SET\_PREV' })

)

return (

<div className="Playbar" css={CSS}>

<div className="left">

<img src={logo} style={{borderRadius:70}}/>

{song && (

<>

<div>{song.title}</div>

<div className="artist">{song.artist}</div>

</>

)}

</div>

<div className="middle">

<div className="play-icon">

<div className="loop-icon" >

<i className={'fa fa-retweet'} />

</div>

<div className="backward-icon" >

<i className={'fa fa-step-backward'} onClick={setPrev}/>

</div>

<div className="play-pause-circle" onClick={playOrPause}>

<i

className={`fa fa-${state.playing ? 'pause' : 'play'}`}

style={{ transform: state.playing ? '' : 'translateX(1.5px)' }}

/>

</div>

<div className="forward-icon" >

<i className={'fa fa-step-forward'} onClick={setNext}/>

</div>

<div className="shuffle-icon" >

<i className={'fa fa-random'} />

</div>

</div>

<div style={{ marginTop: 2.5, display: "inline-flex" }}>

<span style={{ marginTop: 2.5}}>{formatTime(Math.floor(state.currentTime))}</span>

<div className="progress\_bar">

<input

min="0"

type="range"

max={Math.floor(state.duration)}

value={Math.floor(state.currentTime)}

onChange={setTime}

/>

</div>

<span style={{ marginTop: 2.5}}>{formatTime(state.duration)}</span>

</div>

</div>

<div className="right">

<i className="fa fa-volume-up" />

<input

type="range"

min="0"

max="1"

value={state.volume}

step="0.01"

style={{ marginLeft: 10 }}

onChange={setVolume}

/>

</div>

</div>

)

}

const CSS = css`

position: absolute;

bottom: 0;

left: 0;

width: 100%;

height: 80px;

background:linear-gradient(#000000,#000e34);

z-index: 99;

padding: 0 20px;

display: flex;

align-items: center;

justify-content: space-between;

.left {

width: 300px;

.artist {

font-size: 14px;

color: '#999999';

margin-top: 5px;

}

}

.middle {

display: flex;

flex-direction: column;

align-items: center;

.fa-play,

.fa-pause {

font-size: 14px;

}

.play-icon{

display: flex;

.loop-icon{

font-size: 19px;

width: 55px;

height: 35px;

display: flex;

align-items: center;

justify-content: flex-start;

cursor: pointer;

}

.backward-icon{

font-size: 22px;

width: 35px;

height: 35px;

display: flex;

align-items: center;

justify-content: flex-start;

cursor: pointer;

}

.play-pause-circle {

width: 35px;

height: 35px;

border: 1px solid white;

border-radius: 50%;

display: flex;

align-items: center;

justify-content: center;

cursor: pointer;

}

.forward-icon{

font-size: 20px;

width: 35px;

height: 35px;

.shuffle-icon{

font-size: 19px;

width: 55px;

height: 36px;

display: flex;

align-items: center;

justify-content: flex-end;

cursor: pointer;

}

}

.progress\_bar input{

width:600px;

display:inline-flex;

height: 3px;

position: relative;

margin: 0px 10px 15px 10px;

//padding-bottom:5px;

cursor:pointer;

}

}

.right {

.fa-volume-up,

.fa-volume-off {

font-size: 20px;

}

}

img {

height: 50px;

width:50px;

margin-right: 20px;

margin-top: 0px;

padding-left: 0px;

float: left;

}

`

export default Playbar

**6-Sidebar.js :-**

/\*\* @jsx jsx \*/

import React, { useState, useRef, useContext } from 'react'

import { css, jsx } from '@emotion/core'

import { StoreContext } from './index'

import Modal from './Modal'

import Toast from './Toast'

import logo from '../img/logoNew2.png'

const Sidebar = () => {

const [sidebarState, setState] = useState({

modal: false,

toast: ''

})

//accepts a context object (the value returned from React.createContext) and returns the current context value

const { state, dispatch } = useContext(StoreContext)

const playlistRef = useRef(null)

const playlists = Object.keys(state.playlists)

const addPlaylist = e => {

e.preventDefault()

const list = playlistRef.current.value

dispatch({ type: 'ADD\_PLAYLIST', playlist: list })

setState({

...sidebarState,

modal: false,

toast: 'Playlist was created successfully!'

})

}

const handleModal = () =>

setState({ ...sidebarState, modal: !sidebarState.modal })

return (

<ul className="Sidebar" css={CSS}>

{/\* <div className="container"> \*/}

<div className="logo">

<img src={logo} style={{ borderRadius:80}}/>

<div

className="text">

<h2>Musify</h2>

</div>

</div>

{/\* </div> \*/}

<li className="library">Library</li>

{playlists.map(list => (

<li

key={list}

className={list === state.currentPlaylist ? 'active' : ''}

onClick={() => dispatch({ type: 'SET\_PLAYLIST', playlist: list })}>

<i className={ list=='home' ? ["fa fa-home"] : [list=='favorites' ? "fa fa-heart" : "fa fa-headphones" ]} />

&nbsp;

&nbsp;

{list}

</li>

))}

<li className="new-playlist" onClick={handleModal}>

<i className="fa fa-plus-circle" />

<span>New Playlist</span>

</li>

<Modal show={sidebarState.modal} close={handleModal}>

<form onSubmit={addPlaylist}>

<div className="title">New Playlist</div>

<div className="content-wrap">

<input

type="text"

placeholder="My Playlist"

ref={playlistRef}

required

/>

<br />

<button type="submit">Create</button>

</div>

</form>

</Modal>

<Toast

toast={sidebarState.toast}

close={() => setState({ ...sidebarState, toast: '' })}

/>

</ul>

)

}

const CSS = css`

width: 300px;

height: 100%;

background: linear-gradient(#000000,#000e34,#1a001a);

padding-top: 20px;

img {

height: 60px;

//padding-left: 20px;

margin-left:20px;

margin-bottom: 20px;

}

li {

padding-left: 20px;

text-transform: capitalize;

margin-bottom: 10px;

cursor: pointer;

font-weight: bold;

font-size:20px;

}

.container {

display: flex;

align-items: right;

justify-content: right;

}

li.active {

border-left: 2px solid white;

padding-left: 18px;

}

li.library {

margin-top:30px;

cursor: unset;

color: #999;

text-transform: uppercase;

font-weight: normal;

margin-bottom:30px;

}

.img {

padding-left:0px;

flex-basis: 40%;

float:left;

}

.text {

font-size: 15px;

position:relative;

display: inline flow-root list-item;

right:130px;

float:right;

}

li.new-playlist {

position: relative;

top: 40px;

i {

margin-right: 5px;

transform: translateY(1px);

&:before {

font-size: 20px;

}

}

span

font-weight: 300;

}

}

form {

button {

background-color: #5d093b;

color: white;

padding: 12.5px 30px;

border-radius: 25px;

text-transform: uppercase;

font-weight: bold;

font-size: 13px;

border: none;

cursor: pointer;

}

.title {

margin: 0;

margin-bottom: 35px;

}

input {

margin-bottom: 20px;

height: 35px;

padding-left: 8px;

font-size: 16px;

width: 100%;

color: black;

}

.logo{

display:inline-flex; !important

}

.content-wrap {

margin: 0px auto;

max-width: 250px;

text-align: center;

}

}

`

export default Sidebar

**7-Toast.js :-**

// @ts-nocheck

/\*\* @jsx jsx \*/

import React, { useEffect } from 'react'

import { css, jsx } from '@emotion/core'

const Toast = ({ toast, close }) => {

useEffect(() => {

if (!toast) return

const closeToast = () => {

setTimeout(() => {

close()

}, 2500)

}

closeToast()

return () => clearTimeout(closeToast)

}, [toast])

if (!toast) return null

const CSS = css`

-webkit-animation: toast 0.5s cubic-bezier(0.25, 0.46, 0.45, 0.94) both;

animation: toast 0.5s cubic-bezier(0.25, 0.46, 0.45, 0.94) both;

position: absolute;

top: 0;

left: calc(50% - 175px);

background: white;

color: black;

width: 350px;

padding: 25px;

text-align: center;

@-webkit-keyframes toast {

0% {

-webkit-transform: translateY(0);

transform: translateY(0);

}

100% {

-webkit-transform: translateY(75px);

transform: translateY(75px);

}

}

@keyframes toast {

0% {

-webkit-transform: translateY(0px);

transform: translateY(0px);

}

100% {

-webkit-transform: translateY(75px);

transform: translateY(75px);

}

}

`

export default Toast

**8-Useprevious.js**

import { useRef, useEffect } from 'react'

const usePrevious = value => {

const ref = useRef()

useEffect(() => {

ref.current = value

})

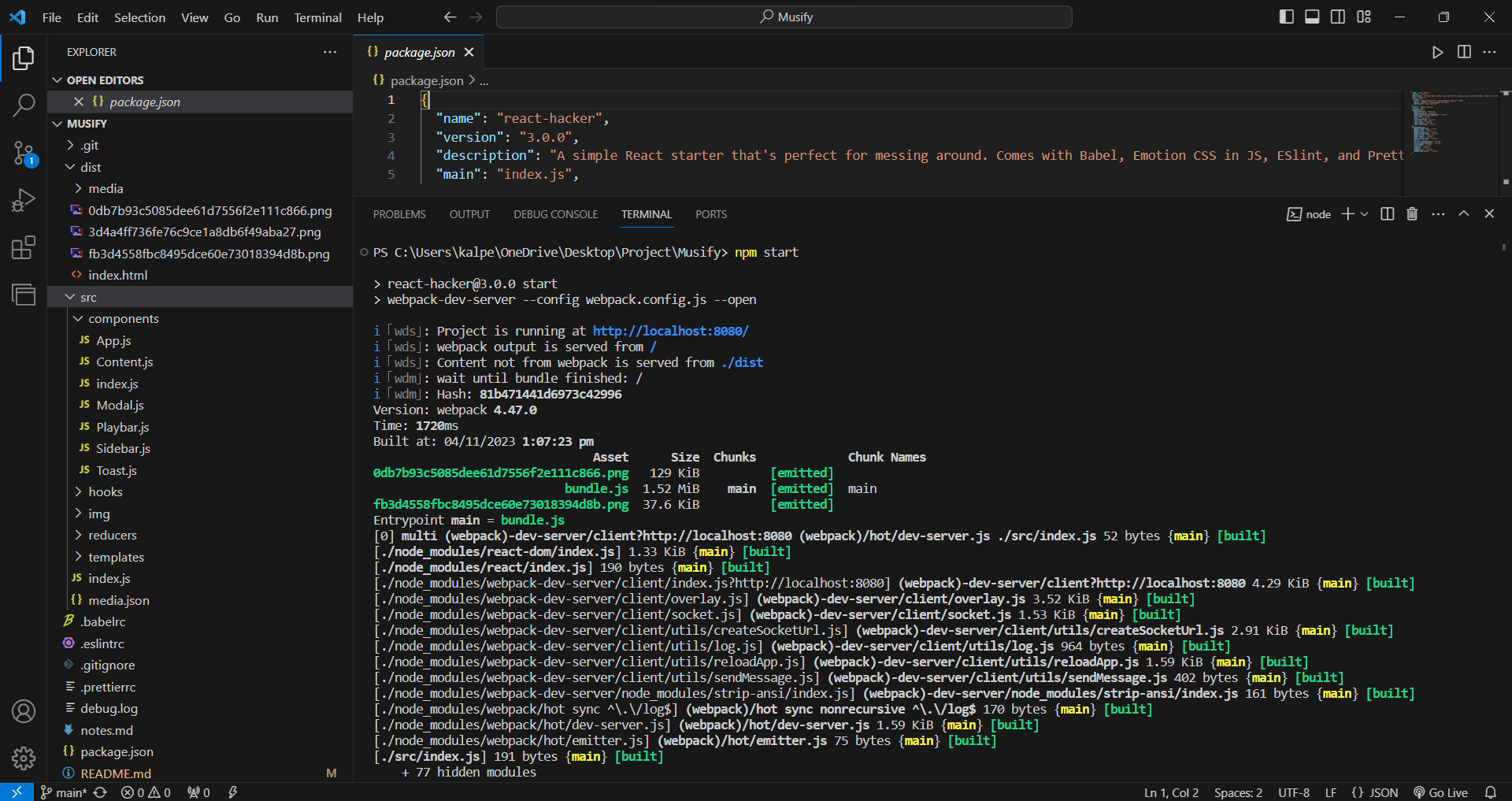
return ref.current

}

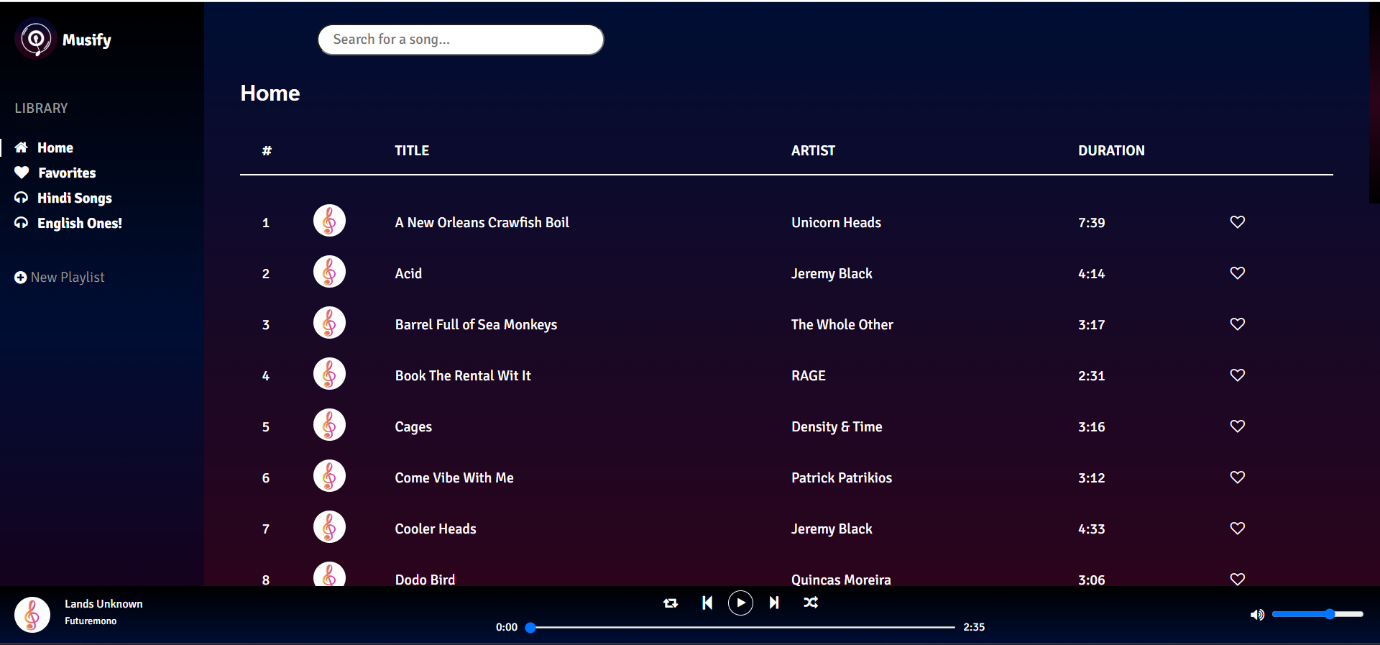
export default usePrevious

# CHAPTER 7

# RESULT



**Figure: 6.1**



**Figure: 6.1**

# CHAPTER 8

# CONCLUSION

In summary, the development of Musify marks a significant milestone in the realm of web-based music applications within the context of operating systems. This project exemplifies the successful fusion of user-centric design, efficient resource management, and robust data security. Musify not only offers users a seamless and engaging music listening experience but also underscores the pivotal role of operating systems in ensuring optimal performance and system reliability.

As we conclude, it is worth emphasizing that Musify's journey is far from over. The path forward is paved with exciting possibilities and potential enhancements. Musify's success is not just a destination but a launching pad for future innovations. With continual dedication to improvement and adaptability, this project is poised to remain at the forefront of the web-based music application market.

# 

# CHAPTER 9

# FUTURE SCOPE

* Expansion of music library and genres.
* Integration of AI-driven recommendation systems.
* Enhanced security measures to protect user data and payments.
* Compatibility with a broader range of operating systems and devices.
* Collaboration with artists and music labels to provide exclusive content.
* Integration with social media platforms for sharing and discovery.

The Musify project has the potential to continually evolve and adapt to changing user needs and technological advancements, making it a frontrunner in the web-based music application market.

# REFERENCES

* W3Schools: <https://www.w3schools.com/>
* <https://code-care.com/blog/music-streaming-app-development/>
* <https://www.contus.com/blog/build-music-streaming-website-application-like-spotify/>
* Node.js Documentation: <https://nodejs.org/en/docs/>
* Design <https://uizard.io/templates/web-app-templates/music-streaming-web-app/>