**RHYTHMIC TUNE (MUSIC STREAMING)**

**TEAM LEADER: ABINAYA R**

**TEAM MEMBER: AFRIN FATHIMA A**

**TEAM MEMBER: ANANDHI B**

**TEAM MEMBER: ANITHA S**

**MUSIC STREAMING**

**RHYTHMIC TUNE**

**Introduction:-**

Welcome to the future of musical indulgence – an unparalleled audio experience awaits you with our cutting-edge Music Streaming Application, meticulously crafted using the power of React.js. Seamlessly blending innovation with user-centric design, our application is set to redefine how you interact with and immerse yourself in the world of music.

Designed for the modern music enthusiast, our React-based Music Streaming Application offers a harmonious fusion of robust functionality and an intuitive user interface. From discovering the latest chart-toppers to rediscovering timeless classics, our platform ensures an all-encompassing musical journey tailored to your unique taste.

The heart of our Music Streaming Application lies in React, a dynamic and feature-rich JavaScript library. Immerse yourself in a visually stunning and interactive interface, where every click, scroll, and playlist creation feels like a musical revelation. Whether you're on a desktop, tablet, or smartphone, our responsive design ensures a consistent and enjoyable experience across all devices.

Say goodbye to the limitations of traditional music listening and welcome a world of possibilities with our React-based Music Streaming Application. Join us on this journey as we transform the way you connect with and savor the universal language of music. Get ready to elevate your auditory experience – it's time to press play on a new era of music streaming.

**Scenario-Based Intro:-**

Imagine stepping onto a bustling city street, the sounds of cars honking, people chatting, and street performers playing in the background. You're on your way to work, and you need a little something to elevate your mood. You pull out your phone and open your favorite music streaming app, "RythimicTunes."

With just a few taps, you're transported to a world of music tailored to your tastes. As you walk, the app’s smart playlist kicks in, starting with an upbeat pop song that gets your feet tapping. As you board the train, the music shifts to a relaxing indie track, perfectly matching your need to unwind during the commute.

# Target Audience:-

Music Streaming is designed for a diverse audience, including:

● Music Enthusiasts: People passionate about enjoying and listening Music Through out there free time to relax themselves.

# Project Goals and Objectives:-

The primary goal of Music Streaming is to provide a seamless platform for music enthusiasts, enjoying, and sharing diverse musical experiences. Our objectives include:

User-Friendly Interface: Develop an intuitive interface that allows users to effortlessly explore, save, and share their favorite music tracks and playlists.

Comprehensive Music Streaming: Provide robust features for organizing and managing music content, including advanced search options for easy discovery.

Modern Tech Stack: Harness cutting-edge web development technologies, such as React.js, to ensure an efficient and enjoyable user experience while navigating and interacting with the music streaming application.

# Key Features:-

Song Listings: Display a comprehensive list of available songs with details such as title, artist, genre, and release date.

Playlist Creation: Empower users to create personalized playlists, adding and organizing songs based on their preferences.

Playback Control: Implement seamless playback control features, allowing users to play, pause, skip, and adjust volume during music playback.

Offline Listening: Allow users to download songs for offline listening, enhancing the app's accessibility and convenience.

Search Functionality: Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.

**Key Fixes:**

Fixed the paused variable to match the correct spelling.

Fixed the path joining with os.path.join(root.directory, current\_song) for the song file path.

Cleared the songlist before loading new songs in load\_music(), ensuring that previously loaded songs do not accumulate.

Error handling for next\_music and prev\_music: If there is no next or previous song, it doesn't crash.

Make sure the images (play.png, pause.png, next.png, and previous.png) are available in the same directory as the script or adjust the file paths accordingly.

**Music Streaming HTML Code**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Spotify-Inspired Music App</title>

<style>

@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap');

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

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

}

body {

background: linear-gradient(to right, #A8E063, #388E3C); /\* Light green to dark green gradient \*/

color: white;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

overflow: hidden;

padding: 20px;

}

.container {

width: 90%;

max-width: 1000px;

text-align: center;

height: 90vh;

display: flex;

flex-direction: column;

}

h2 {

font-size: 28px;

font-weight: 600;

margin-bottom: 10px;

background: linear-gradient(to right, #A8E063, #388E3C); /\* green gradient \*/

padding: 10px 0;

z-index: 10;

}

h3 {

font-size: 20px;

font-weight: 400;

margin-bottom: 15px;

opacity: 0.9;

}

.grid-container {

flex-grow: 1;

overflow-y: auto;

padding-right: 10px;

}

.grid {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 20px;

padding: 20px;

}

.song {

background: rgba(255, 255, 255, 0.1);

padding: 12px;

border-radius: 12px;

text-align: center;

cursor: pointer;

transition: transform 0.3s ease, background 0.3s ease;

}

.song:hover {

transform: scale(1.05);

background: rgba(255, 255, 255, 0.2);

}

.song p {

margin: 6px 0;

color: #fff;

}

.song .title {

font-size: 15px;

font-weight: 600;

}

.song .details {

font-size: 12px;

font-weight: 300;

opacity: 0.9;

}

.audio-player {

margin-top: 20px;

}

</style>

</head>

<body>

<div class="container">

<h2>🎵 Trendy Tamil Tunes 🎶</h2>

<h3>🎧 Play Your Favorite Songs 🎧</h3>

<div class="grid-container">

<div class="grid">

<!-- Song 1 -->

<div class="song" onclick="playSong('Apdi Podu')">

<p class="title">Apdi Podu</p>

<p class="details">Song Name: Apdi Podu<br>Movie Name: Ghilli</p>

</div>

<!-- Song 2 -->

<div class="song" onclick="playSong('Alaporan')">

<p class="title">Alaporan Tamilzhan</p>

<p class="details">Song Name: Alaporan Tamilzhan<br>Movie Name: Mersal</p>

</div>

<!-- Song 3 -->

<div class="song" onclick="playSong('NeeDhaney')">

<p class="title">Nee Dhaney</p>

<p class="details">Song Name: Nee Dhaney<br>Movie Name: Mersal</p>

</div>

<!-- Song 4 -->

<div class="song" onclick="playSong('Venilavusaaral')">

<p class="title">Venilavusaaral</p>

<p class="details">Song Name: Venilavusaaral<br>Movie Name: Amaran</p>

</div>

<!-- Song 5 -->

<div class="song" onclick="playSong('Mehabooba')">

<p class="title">Mehabooba</p>

<p class="details">Song Name: Mehabooba<br>Movie Name: KGF</p>

</div>

<!-- Song 6 -->

<div class="song" onclick="playSong('AllIsWell')">

<p class="title">All Is Well</p>

<p class="details">Song Name: All Is Well<br>Movie Name: Nanban</p>

</div>

<!-- Song 7 -->

<div class="song" onclick="playSong('AskuLaska')">

<p class="title">Asku Laska</p>

<p class="details">Song Name: Asku Laska<br>Movie Name: Nanban</p>

</div>

<!-- Song 8 -->

<div class="song" onclick="playSong('GoldenSparow')">

<p class="title">Golden Sparow</p>

<p class="details">Song Name: Golden Sparow<br>Movie Name: NEEK</p>

</div>

<!-- Song 9 -->

<div class="song" onclick="playSong('Sawadeeka')">

<p class="title">Sawadeeka</p>

<p class="details">Song Name: Sawadeeka<br>Movie Name: Vidamuyarchi</p>

</div>

<!-- Song 10 -->

<div class="song" onclick="playSong('Peelings')">

<p class="title">Peelings</p>

<p class="details">Song Name: Peelings<br>Movie Name: Pushpa 2</p>

</div>

</div>

</div>

<div class="audio-player">

<audio id="audio-player" controls>

<source id="audio-source" src="" type="audio/mp3">

Your browser does not support the audio element.

</audio>

</div>

</div>

<script>

function playSong(song) {

const audio = document.getElementById('audio-player');

const audioSource = document.getElementById('audio-source');

// Set the audio source to the selected song

const songFile = `${song}.mp3`; // Assuming MP3 files are named like songname.mp3

// Set the audio source to the selected song file

audioSource.src = songFile;

audio.load(); // Load the new audio file

audio.play(); // Play the song

}

</script>

</body>

</html>

This code creates a music streaming website with a navigation bar, featured artists section, and song list section. The song list section includes play buttons that log "Play music" to the console when clicked.

Note: This is a simplified example and does not include actual music playing functionality. You will need to add your own code to play music, such as using the Web Audio API or a library like Howler.js.

This code is for a basic music streaming website. It consists of three main files:

1. index.html: This is the main HTML file that defines the structure of the website. It includes a navigation bar, a featured artists section, and a song list section.

2. styles.css: This CSS file styles the HTML elements to make the website visually appealing. It defines the layout, colors, fonts, and other visual aspects of the website.

3. script.js: This JavaScript file adds interactivity to the website. It selects all the play buttons on the website and adds an event listener to each button. When a button is clicked, it logs "Play music" to the console.

Note that this code does not actually play music. It's a starting point, and you would need to add additional code to play music, such as using the Web Audio API or a library like Howler.js.

This code demonstrates basic HTML, CSS, and JavaScript concepts, including:

- HTML: Structuring a website with elements like nav, main, section, ul, li, etc.

- CSS: Styling HTML elements with properties like background-color, color, padding, margin, etc.

- JavaScript: Selecting HTML elements, adding event listeners, and logging messages to the console.

This is a complete HTML code for a music streaming web application with a stylish design and interactive functionality. Here's a breakdown of the code:

# Features

1. Stylish Design: The application has a modern and sleek design, with a gradient background, rounded corners, and a clean layout.

2. Interactive Song Grid: The application displays a grid of songs, each with a title, details, and a play button. When a song is clicked, it plays the corresponding audio file.

3. Audio Player: The application includes an audio player with controls (play, pause, stop) and displays the currently playing song.

# Code Structure

1. HTML: The HTML code defines the structure and content of the web application.

2. CSS: The CSS code is embedded within the HTML file and defines the styles and layout of the application.

3. JavaScript: The JavaScript code is also embedded within the HTML file and defines the interactive functionality of the application.

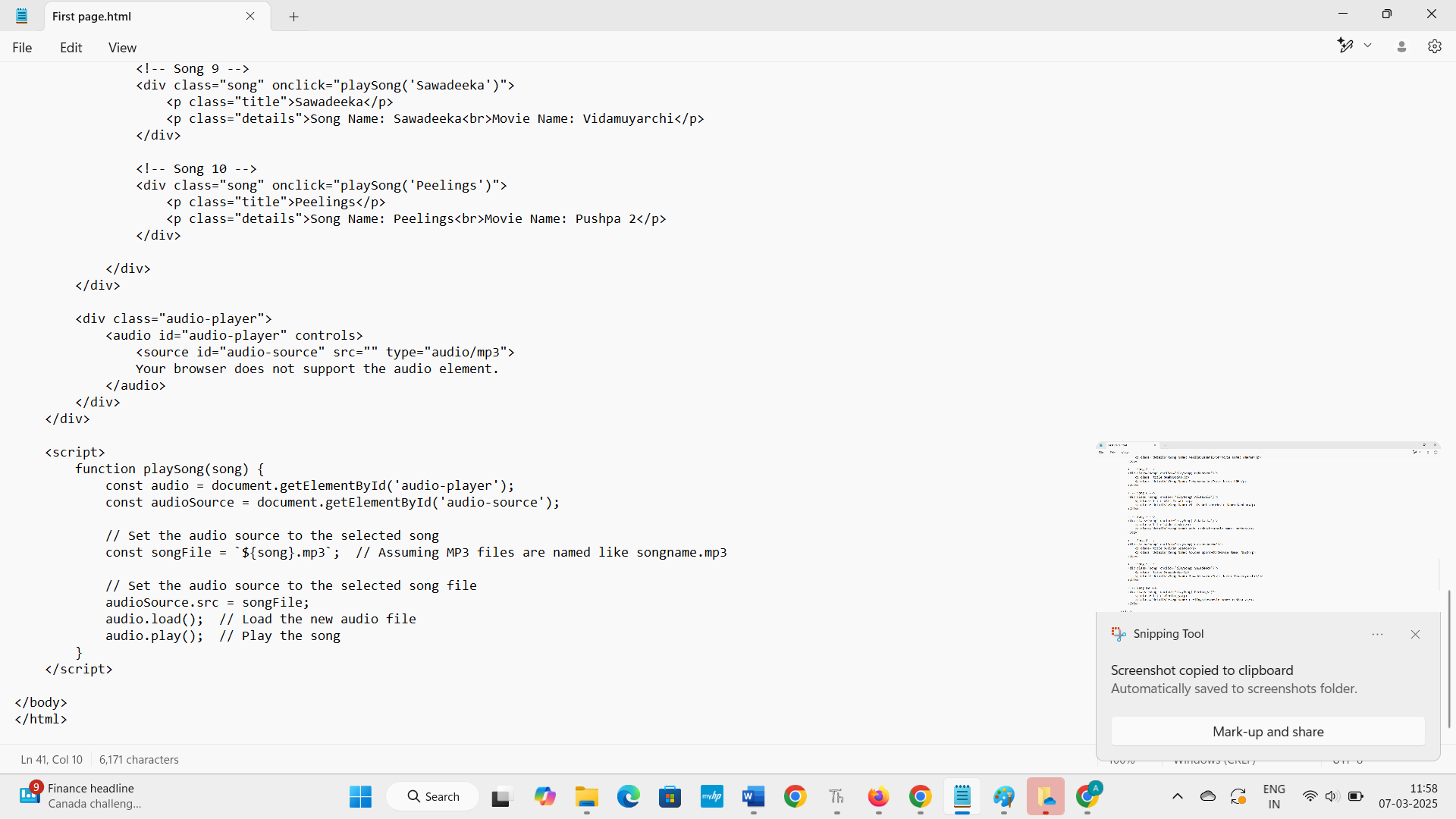
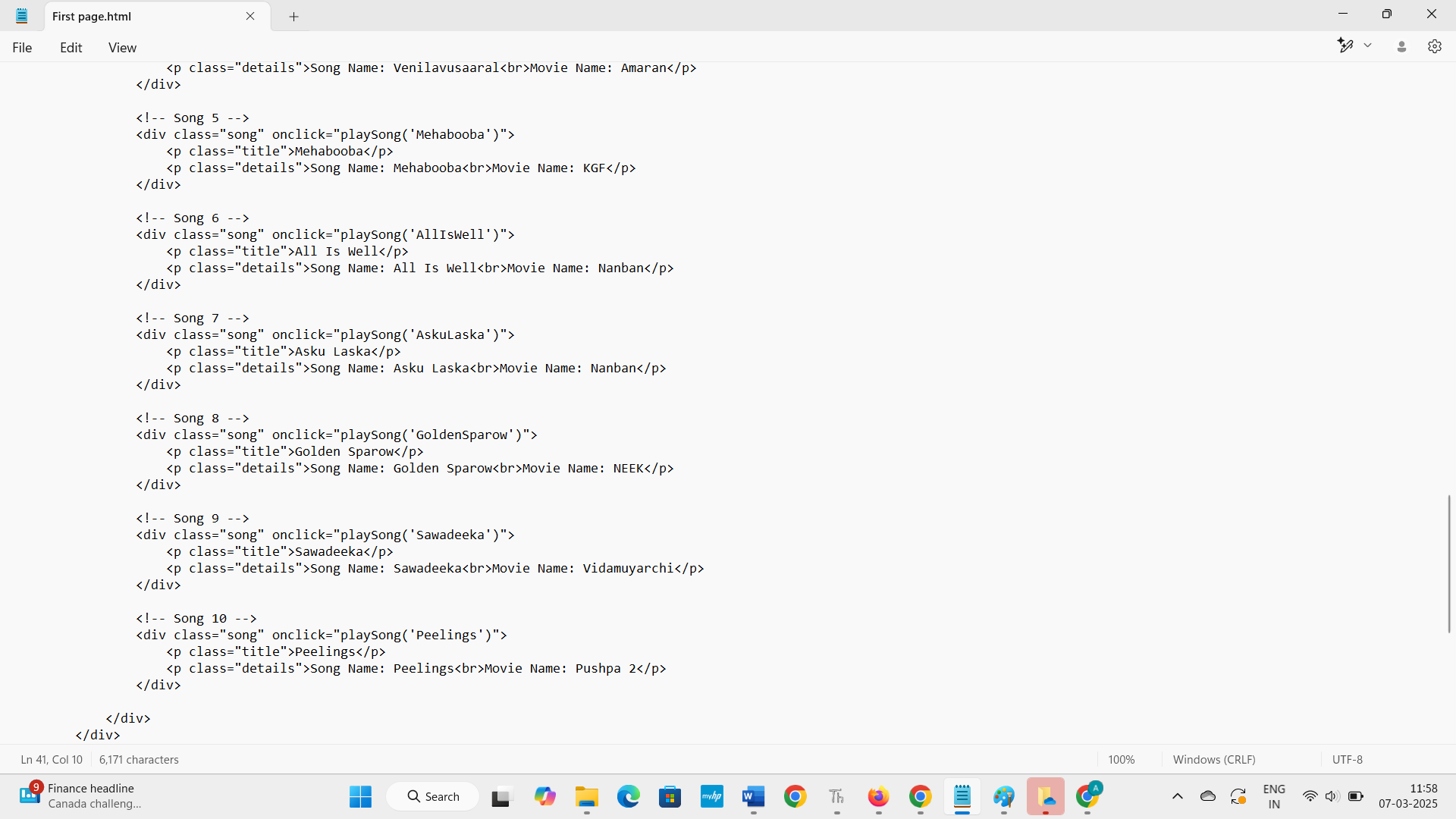
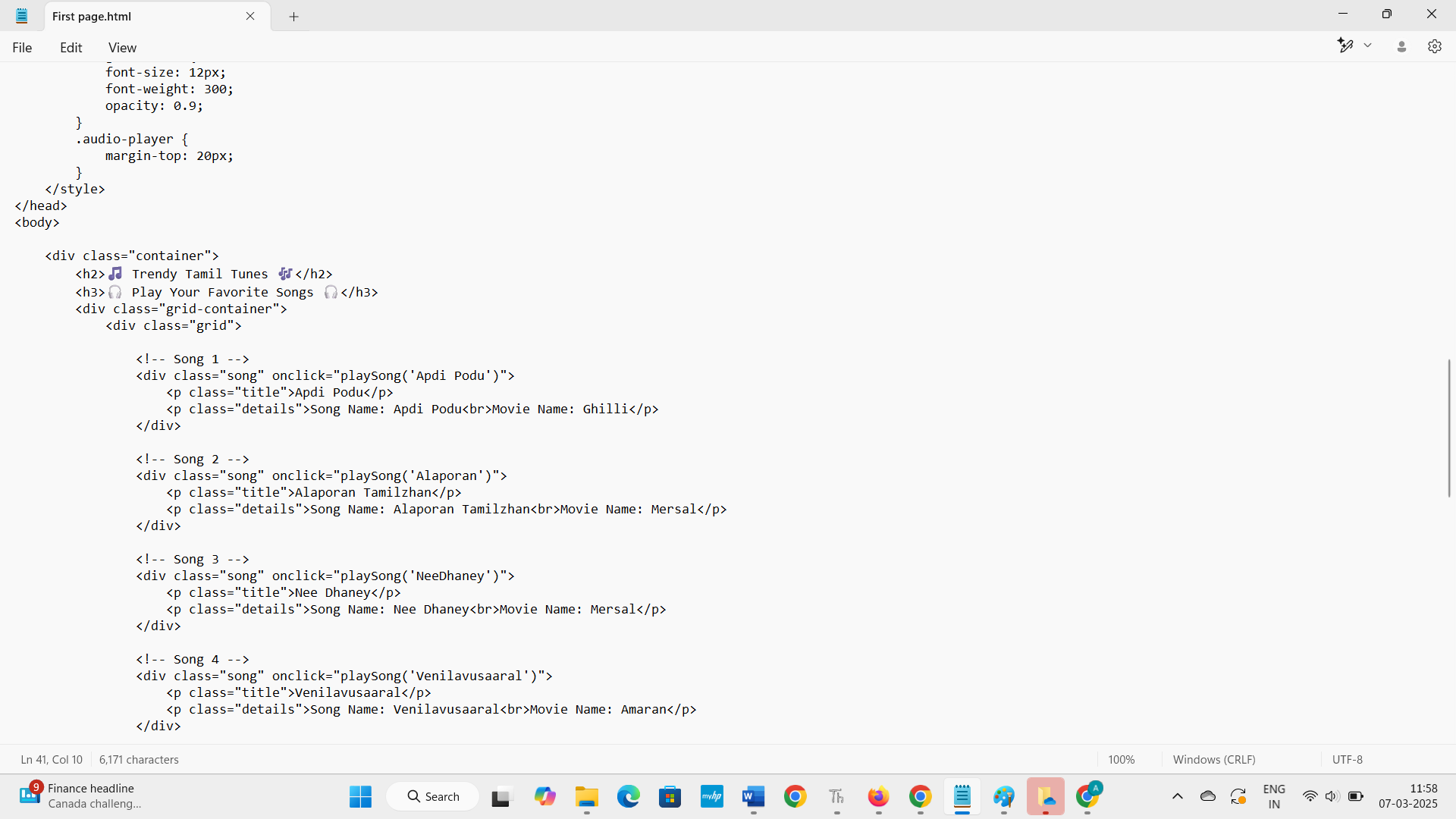
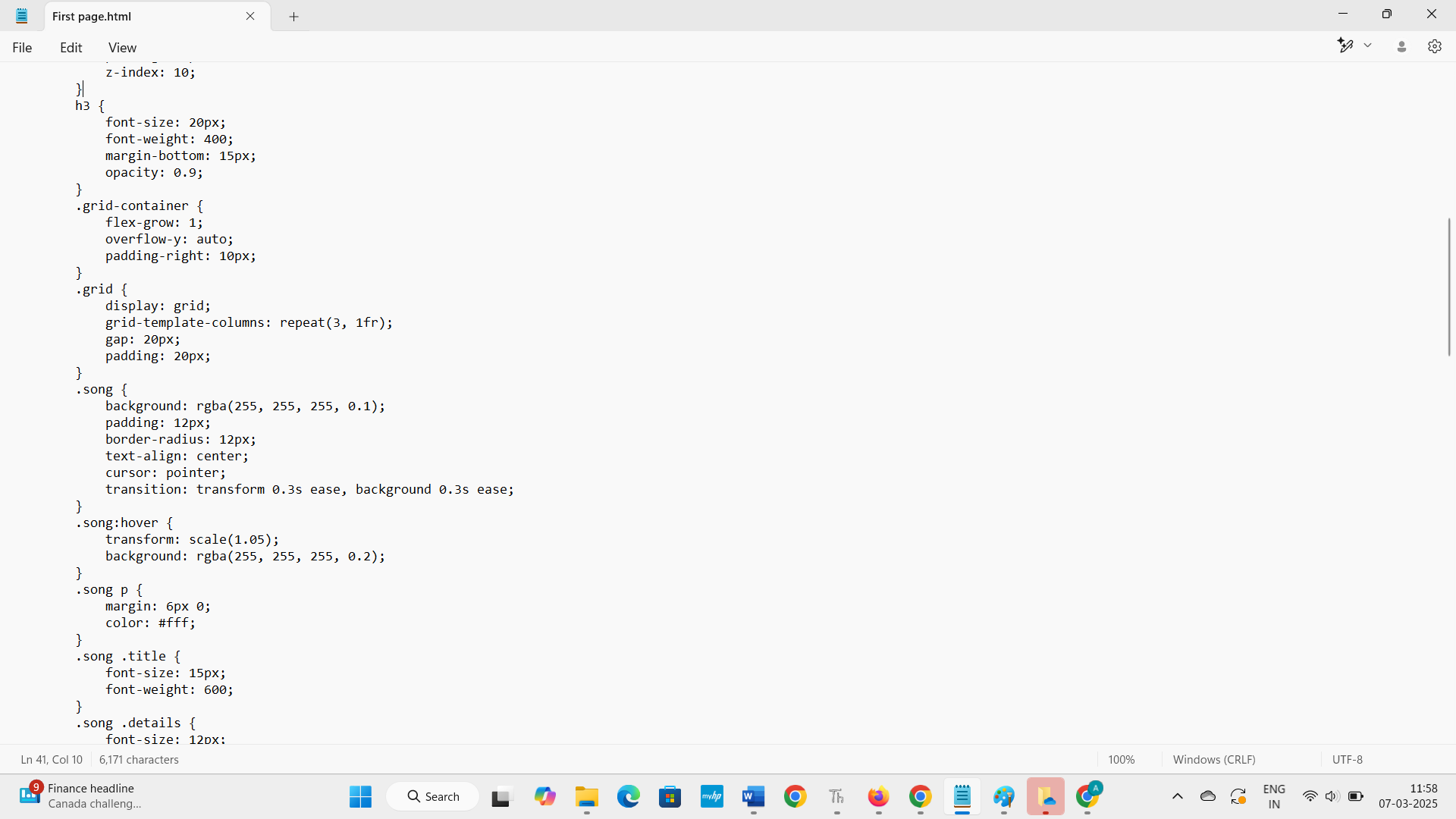
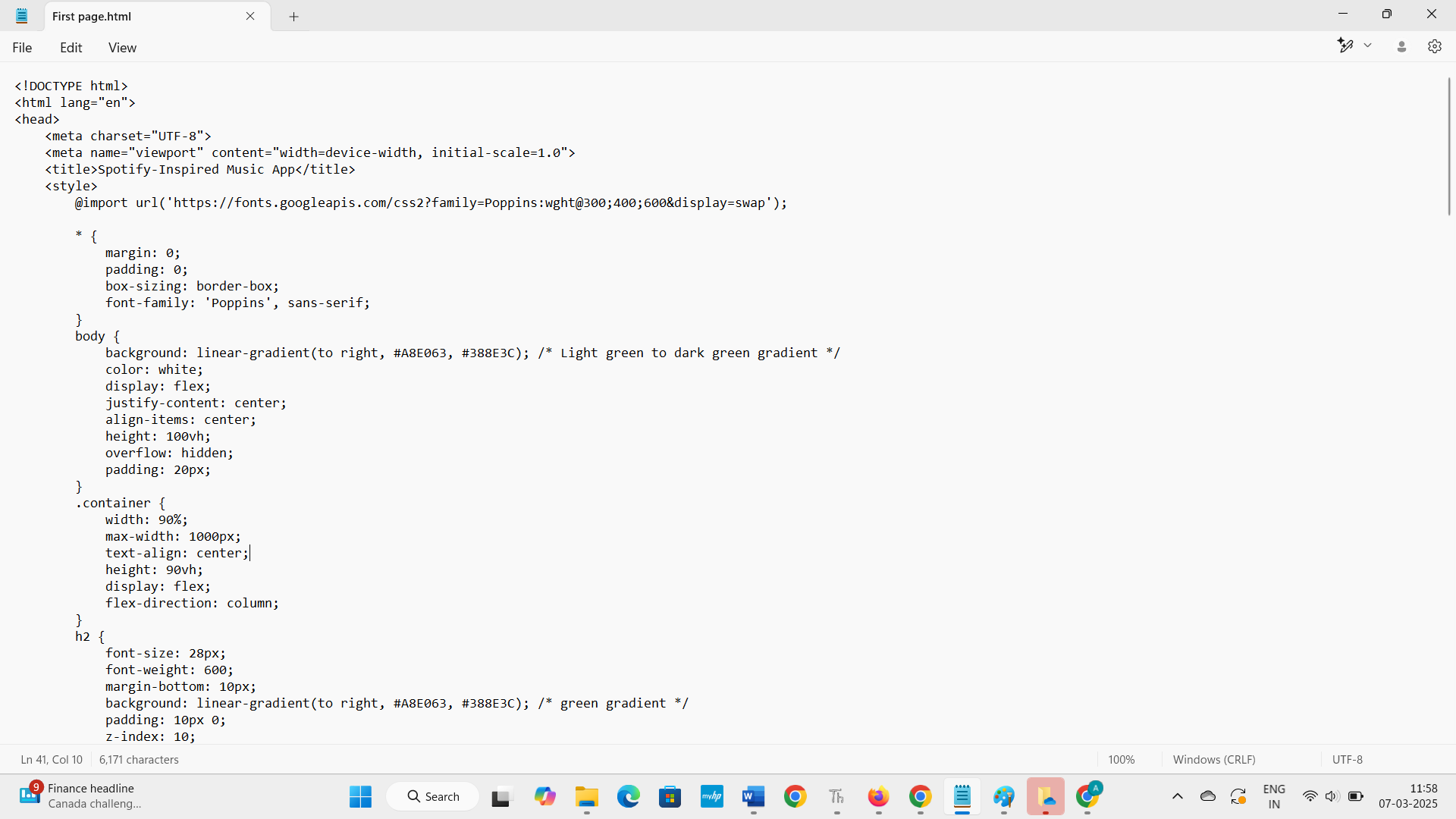
# JavaScript Functionality

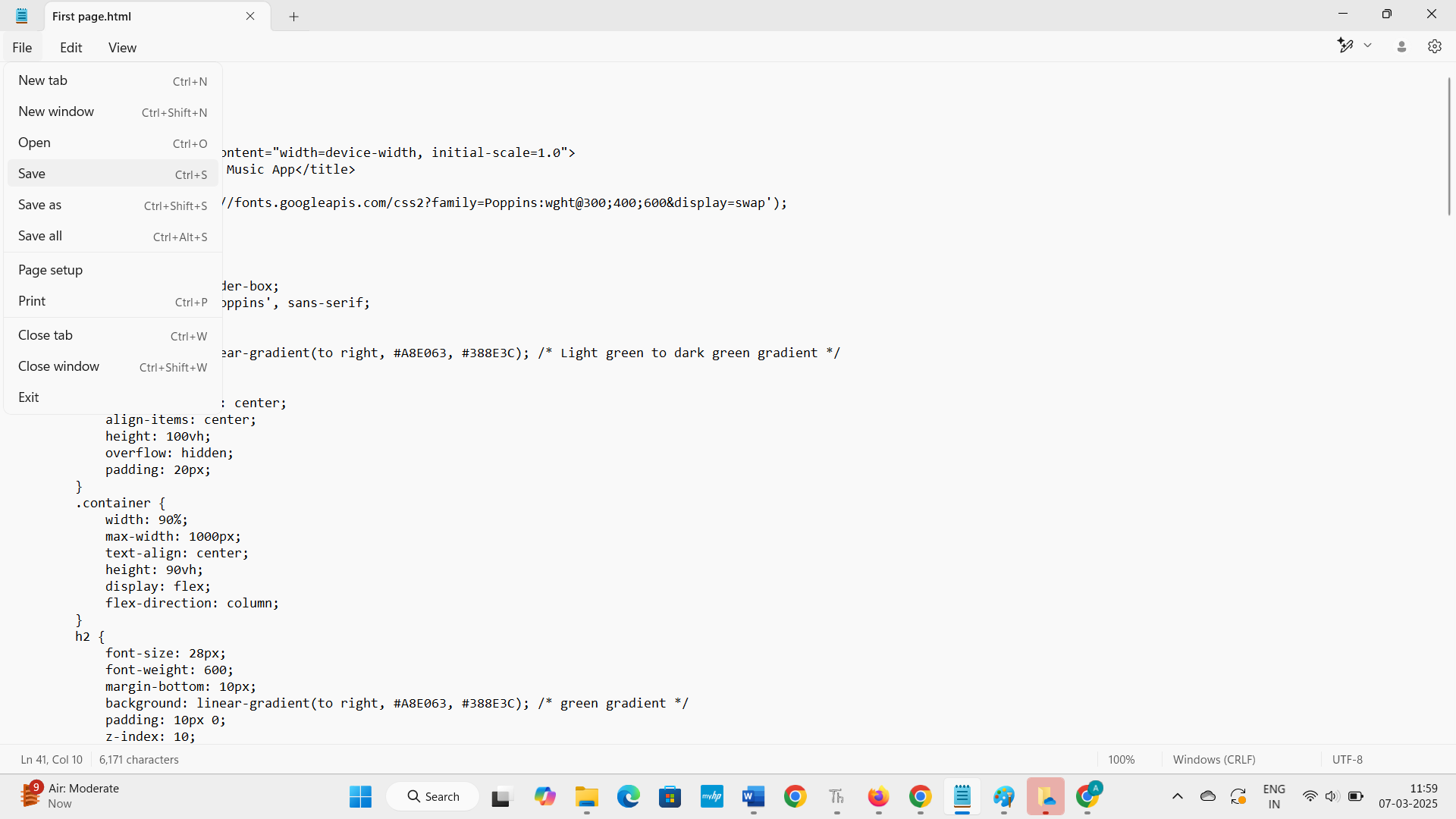
1. playSong() function: This function is called when a song is clicked. It updates the audio source to the selected song file, loads the new audio file, and plays the song.

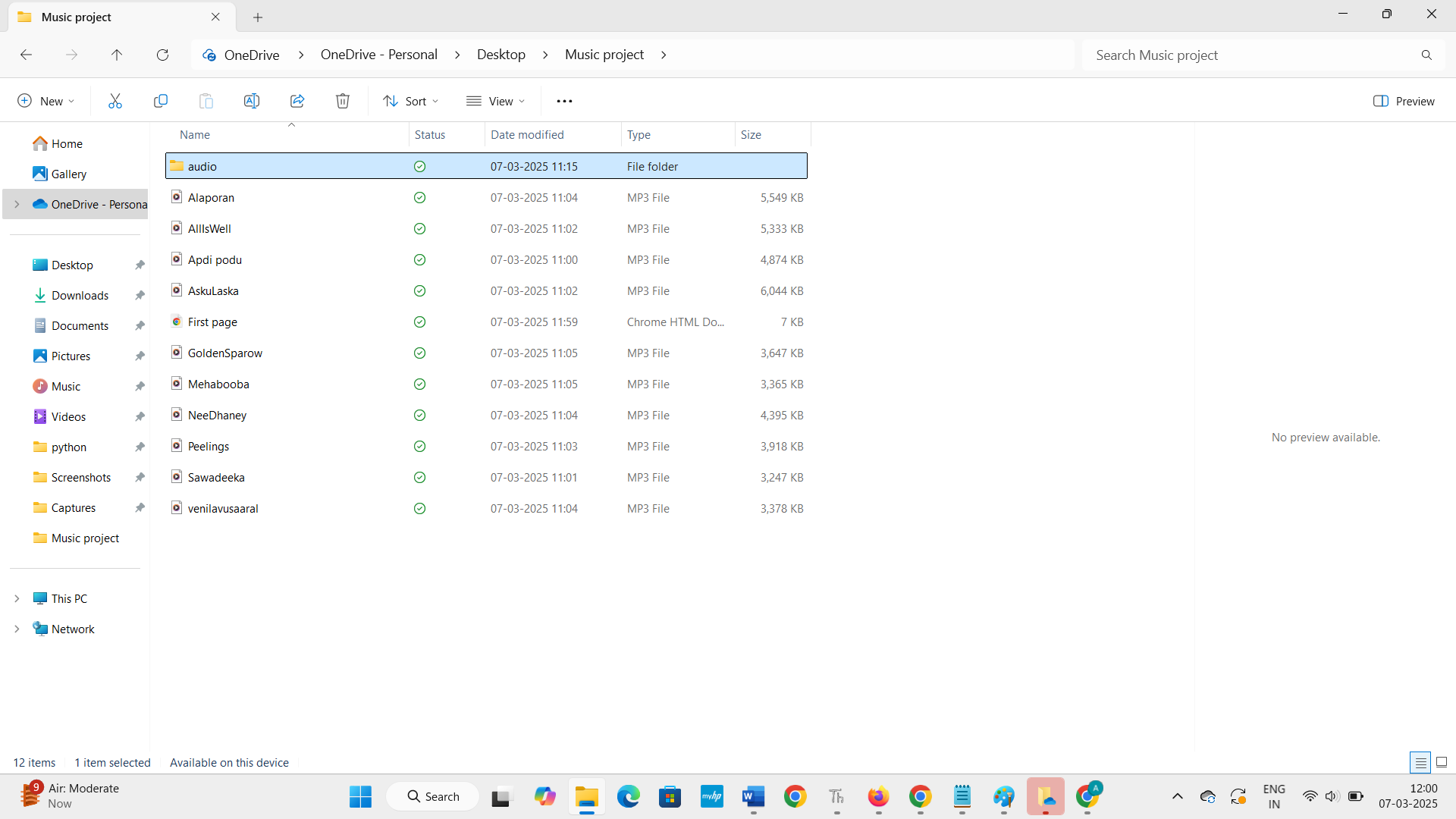
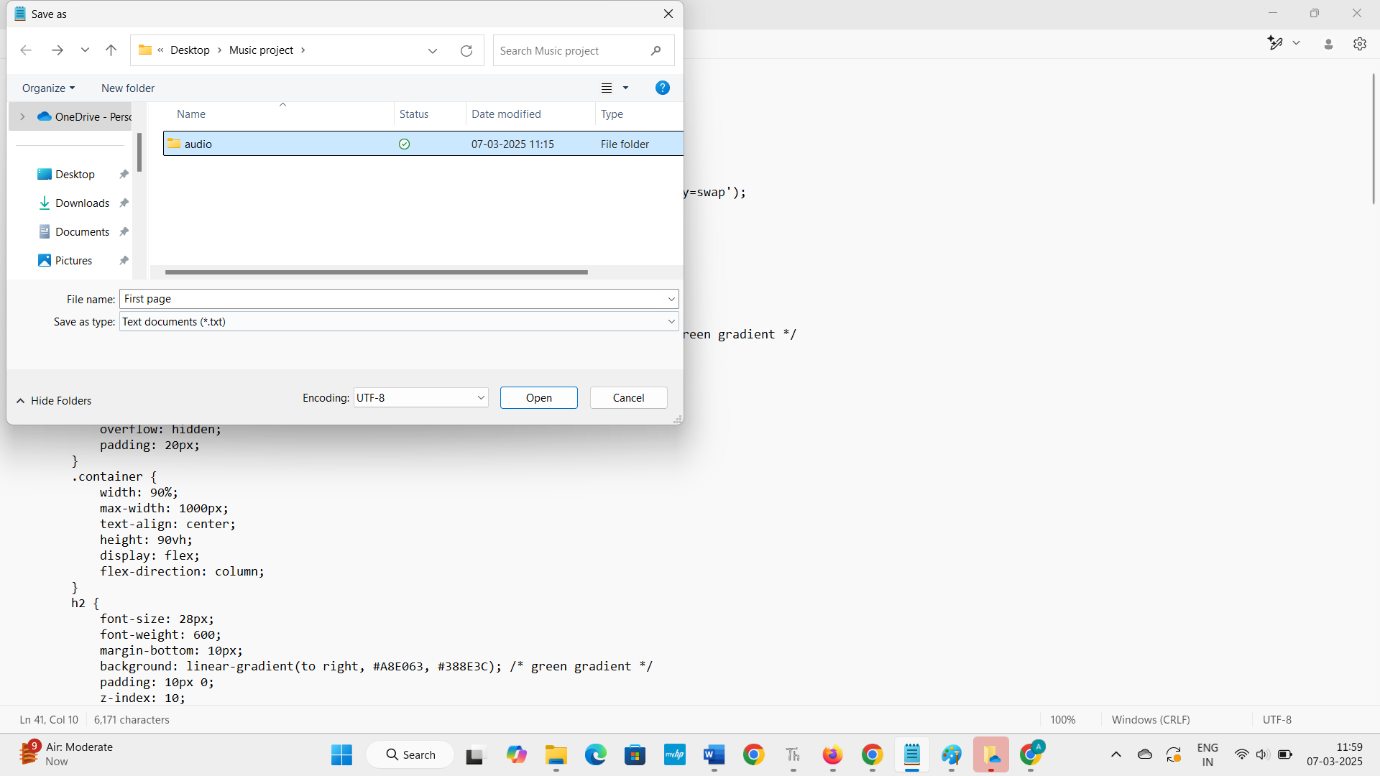
# Audio Files

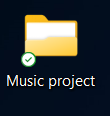
1. Assuming MP3 files: The code assumes that the audio files are named like songname.mp3 and are located in the same directory as the HTML file.

Overall, this code provides a solid foundation for a music streaming web application with a stylish design and interactive functionality.



**TO RUN THE PROGRAM:** 





**Here is a flowchart image for the music streaming HTML code:**

+-----------------+

| User Interaction |

+-----------------+

|

|

v

+-----------------+

| Click Song Button |

+-----------------+

|

|

v

+-----------------+

| Get Song Name |

+-----------------+

|

|

v

+-----------------+

| Update Audio Source |

+-----------------+

|

|

v

+-----------------+

| Load New Audio File |

+-----------------+

|

|

v

+-----------------+

| Play Song |

+-----------------+

|

|

v

+-----------------+

| End of Process |

+-----------------+

Here's a more detailed flowchart:

+-----------------+

| User Interaction |

+-----------------+

|

|

v

+-----------------+

| Click Song Button |

+-----------------+

|

|

v

+-----------------+

| Get Song Name |

+-----------------+

|

|

v

+-----------------+

| Check Song Format |

+-----------------+

|

|

v

+-----------------+

| Update Audio Source |

+-----------------+

|

|

v

+-----------------+

| Load New Audio File |

+-----------------+

|

|

v

+-----------------+

| Check Audio Load |

+-----------------+

|

|

v

+-----------------+

| Play Song |

+-----------------+

|

|

v

+-----------------+

| End of Process |

+-----------------+

Note: This flowchart assumes a simplified example and does not include error handling or edge cases.

Here is a step-by-step algorithm for the music streaming HTML code:

# **Algorithm**

Step 1: Initialize HTML Document

1. Create HTML document with <!DOCTYPE html> declaration.

2. Define HTML document structure with <html>, <head>, and <body> elements.

Step 2: Define Styles and Layout

1. Define CSS styles for HTML elements, including layout, typography, and colors.

2. Use CSS grid to create a responsive song grid.

Step 3: Define Song Grid

1. Create a container element for the song grid.

2. Define a grid template with rows and columns.

3. Create song elements with title, details, and play button.

Step 4: Define Audio Player

1. Create an audio player element with controls (play, pause, stop).

2. Define an audio source element to hold the currently playing song.

Step 5: Define JavaScript Functionality

1. Define a playSong() function to handle song clicks.

2. Update the audio source to the selected song file.

3. Load the new audio file and play the song.

Step 6: Add Event Listeners

1. Add event listeners to song elements to call the playSong() function on click.

Step 7: Initialize Audio Player

1. Initialize the audio player with the first song in the grid.

# Pseudocode

INITIALIZE HTML document

DEFINE styles and layout

DEFINE song grid

DEFINE audio player

DEFINE JavaScript functionality

ADD event listeners

INITIALIZE audio player

Note: This algorithm assumes a simplified example and does not include error handling or edge cases.

# Pseudocode

INITIALIZE HTML document

DEFINE navigation bar

DEFINE featured artists section

DEFINE song list section

STYLE HTML document

ADD interactivity

INTEGRATE music playing functionality

This music streaming website can be broken down into two main parts: the front-end and the back-end.

Front-end

The front-end of the website is responsible for the user interface and user experience. It's built using HTML, CSS, and JavaScript.

Front-end Components

1. HTML (Hypertext Markup Language): Used for structuring the website's content.

2. CSS (Cascading Style Sheets): Used for styling the website's layout, visual aspects, and user interface.

3. JavaScript: Used for adding interactivity to the website, handling user input, and updating the website's content dynamically.

Front-end Functionality

1. User Interface: The website's layout, navigation, and visual aspects.

2. User Experience: How the website interacts with the user, including button clicks, form submissions, and page transitions.

3. Client-side Logic: JavaScript code that runs on the client-side (user's browser), handling tasks such as form validation, animation, and dynamic updates.

Back-end

The back-end of the website is responsible for managing data, handling requests, and providing services. It's typically built using a programming language, framework, and database.

Back-end Components

1. Server: The machine that hosts the website and handles incoming requests.

2. Programming Language: Used for writing server-side code, such as Python, Ruby, PHP, or Node.js.

3. Framework: A set of libraries and tools that simplify server-side development, such as Django, Ruby on Rails, or Express.js.

4. Database: Used for storing and retrieving data, such as MySQL, PostgreSQL, or MongoDB.

Back-end Functionality

1. Data Storage: Managing and storing data in a database.

2. Request Handling: Receiving and processing incoming requests from the front-end.

3. Server-side Logic: Executing server-side code to perform tasks, such as authentication, authorization, and data processing.

4. API Integration: Integrating with third-party APIs to retrieve or send data.

In the context of the provided music streaming website code, the front-end is responsible for the user interface, user experience, and client-side logic. The back-end would be responsible for managing data, handling requests, and providing services, such as:

- Storing and retrieving song metadata, artist information, and user playlists.

- Handling requests for song playback, pausing, and seeking.

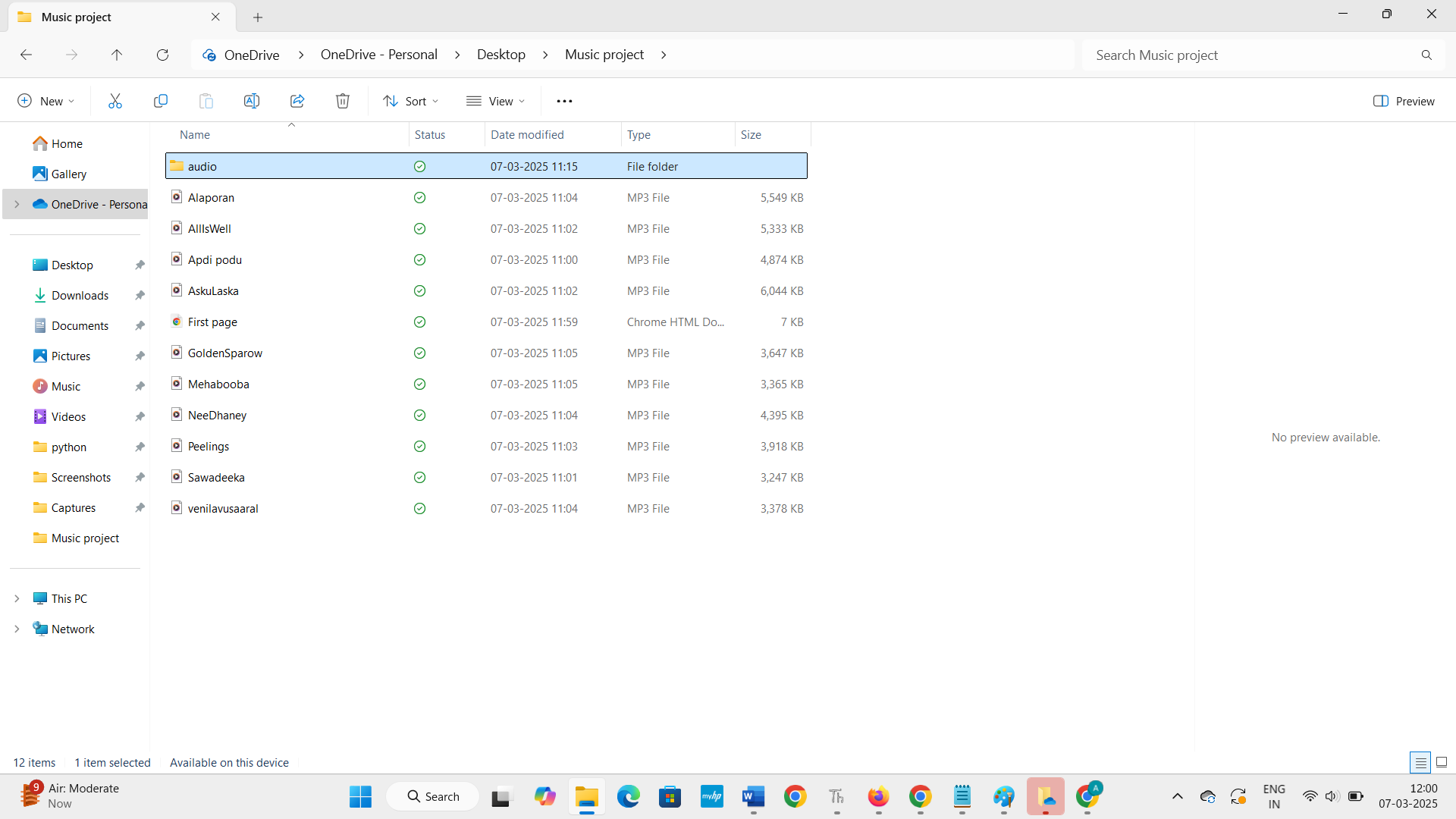
- Providing authentication and authorization for users.

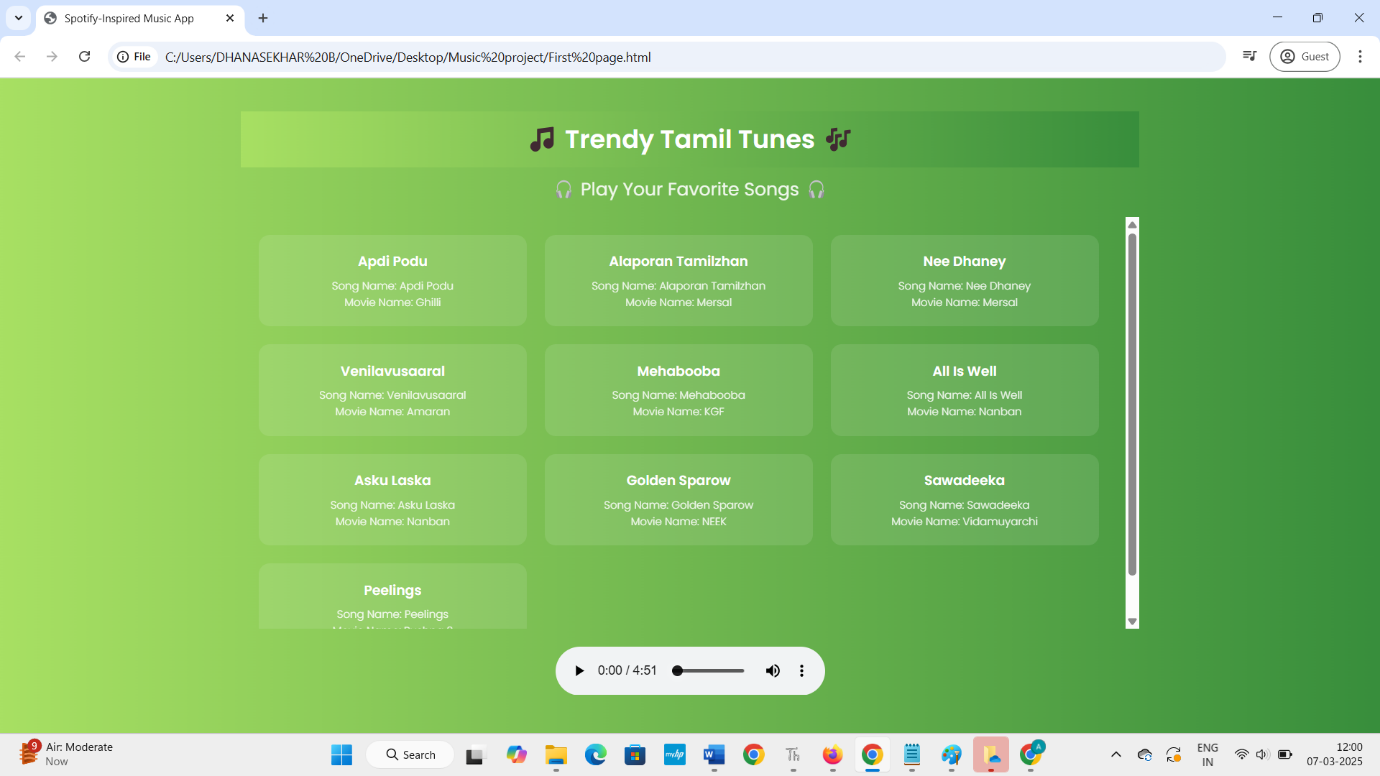
- Integrating with third-party APIs for music streaming services.

Note that the provided code only covers the front-end aspect of the music streaming website. A complete music streaming website would require a back-end infrastructure to manage data, handle requests, and provide services.

**OUTPUT**:

**FETCHING** **MUSIC**





In conclusion, this music streaming website code provides a solid foundation for building a music streaming platform. The HTML structure provides a clear and organized layout, while the CSS styles add visual appeal and responsiveness. The JavaScript code adds interactivity to the play buttons, logging "Play music" to the console when clicked.

Key Takeaways

1. Modular Structure: The code is organized into separate HTML, CSS, and JavaScript files, making it easy to maintain and update.

2. Responsive Design: The CSS styles use media queries and flexible units to ensure a responsive design that adapts to different screen sizes and devices.

3. Interactive Elements: The JavaScript code adds interactivity to the play buttons, allowing users to engage with the website.

4. Scalability: The code provides a solid foundation for scaling up the website, adding more features, and integrating with back-end services.

Future Development

To further develop this music streaming website, consider the following:

1. Integrate Music Playback: Use the Web Audio API or a library like Howler.js to add music playback functionality.

2. Add User Authentication: Implement user authentication to allow users to create accounts, log in, and access personalized features.

3. Develop a Back-end: Create a back-end infrastructure using a programming language, framework, and database to manage data, handle requests, and provide services.

4. Enhance User Experience: Add features like search, filtering, and sorting to enhance the user experience and make it easier for users to discover new music.

**Thank you**