

Step-by-Step Guide to Creating this JS Music Player

Step 1: Setting Up HTML Structure

- An HTML file was created and opened in my preferred code editor (vscode).
- The content needed for the music player was done. This code sets up the basic structure of the music player webpage, including the playlist name, banner image, music content, audio element, progress slider, duration display, and control buttons.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <div class="bodyContainer">
10     <!-- music content here -->
11   </div>
12
13 </body>
14 </html>
```

Step 2: Linked CSS and Boxicons

- Inside the `<head>` section of my HTML file, I ensured that I had linked an external CSS stylesheet and the Boxicons library. These resources will help style the webpage and provide icons for the player controls.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <link rel="stylesheet" href="style.css">
7     <link href='https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css' rel='stylesheet'>
8     <title>JS Music Player</title>
9   </head>
10  <body>
```

Step 3: Styling the music Player and Google Font

- I customized the styles in my CSS file (`style.css`) to achieve the desired look and feel of my music player.


```
1 @import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;700&family=Lexend+Deca&family=Roboto:wght@400;500;700&display=swap');
2
3 *{
4     box-sizing: border-box;
5 }
6
7 body{
8     font-family: 'Roboto', sans-serif;
9     background-color: #B49286;
10 }
11
```

Step 4: Add JavaScript Code

- Created a new JavaScript file (e.g., `main.js`) in the same directory as the HTML file and linked it in the HTML file


```
1 <body>
2     <div class="bodyContainer">
3         <!-- music content here -->
4     </div>
5
6     <!-- link js file -->
7     <script src="main.js"></script>
8 </body>
9 </html>
```

- a. **Get Elements:** Here, Getting references to various HTML elements using their respective IDs and classes were obtained.



```
1 let song = document.getElementById('song');
2 let songProgress = document.getElementById('song-progress');
3 let start = document.getElementById('start');
4 let playControl = document.getElementById('play-control');
5 let fastForward = document.querySelector('.bx-fast-forward');
6 let rewind = document.querySelector('.bx-rewind');
```

- b. **Set Initial Song Progress:** This code sets the initial maximum value of the progress slider (`songProgress.max`) to the total duration of the song (`song.duration`). It also initializes the slider's current value (`songProgress.value`) to the current playback time of the song (`song.currentTime`).



```
1 song.onloadedmetadata = function(){
2     songProgress.max = song.duration;
3     songProgress.value = song.currentTime;
4 }
```

- c. **Defined `playPause()` Function:** This function toggles between playing and pausing the song. It checks the class of the play control button. If it contains the class `'bx-pause'`, it means the song is currently playing, so the function pauses the song and changes the button icon to play. If the button doesn't have the

pause class, it means the song is paused, so the function plays the song and changes the button icon to pause.

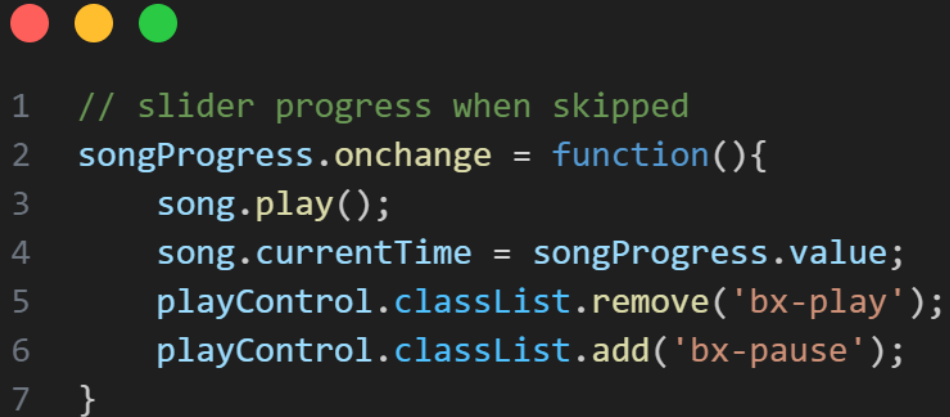
```
1 //playPause function handler
2 function playPause(){
3     if(playControl.classList.contains('bx-pause')){
4         song.pause();
5         playControl.classList.remove('bx-pause');
6         playControl.classList.add('bx-play');
7     }
8     else{
9         song.play();
10        playControl.classList.remove('bx-play');
11        playControl.classList.add('bx-pause');
12    }
13 }
```

- d. **Implement Song Progress Slider Update:** Here, a `setInterval` function is used to repeatedly update the value of the song progress slider. If the song is playing (`!song.paused`), the slider's value is updated to the current playback time of the song (`song.currentTime`) every 500 milliseconds.

```
1 // slider progress when playing
2 if(song.play()){
3     setInterval(()=>{
4         songProgress.value = song.currentTime;
5     }, 500);
6 }
```

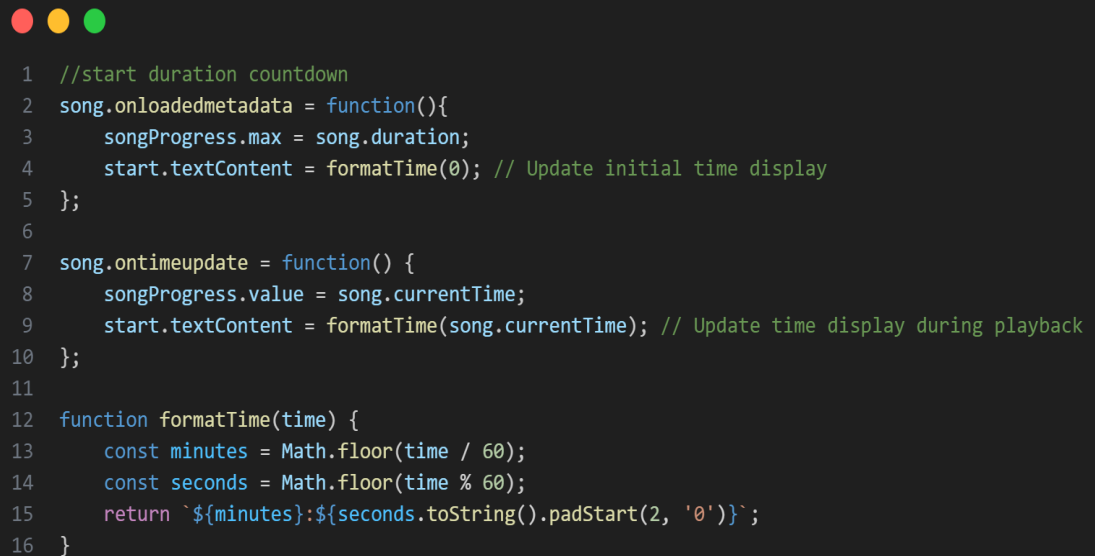
- e. **Handle Manual Song Seek:** This code attaches an `onchange` event handler to the song progress slider. When the user manually changes the slider's value

(seeking), the function sets the song to play, updates the song's current time to match the slider's value, and changes the play control button to the pause icon.



```
1  // slider progress when skipped
2  songProgress.onChange = function(){
3      song.play();
4      song.currentTime = songProgress.value;
5      playControl.classList.remove('bx-play');
6      playControl.classList.add('bx-pause');
7  }
```

- f. **Update Time Display:** This section of code updates the song progress slider and the time display during playback. The `ontimeupdate` event is triggered as the song's playback time changes. Inside the event handler, the slider's value and the time display are updated. The `formatTime(time)` function is used to convert the time in seconds to a formatted `minutes:seconds` string.

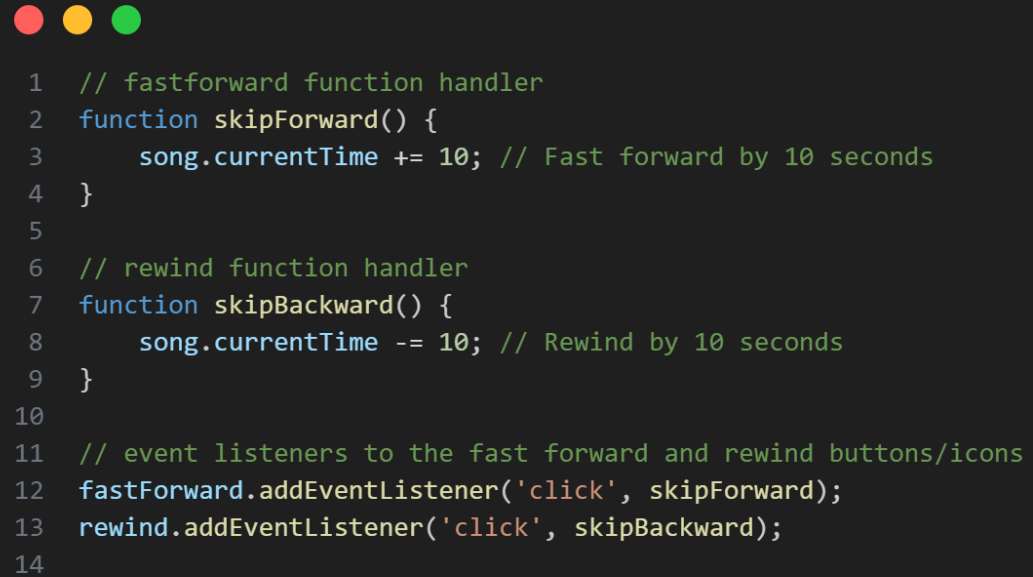


```

1  //start duration countdown
2  song.onloadedmetadata = function(){
3      songProgress.max = song.duration;
4      start.textContent = formatTime(0); // Update initial time display
5  };
6
7  song.ontimeupdate = function() {
8      songProgress.value = song.currentTime;
9      start.textContent = formatTime(song.currentTime); // Update time display during playback
10 };
11
12 function formatTime(time) {
13     const minutes = Math.floor(time / 60);
14     const seconds = Math.floor(time % 60);
15     return `${minutes}:${seconds.toString().padStart(2, '0')}`;
16 }

```

- g. **Implement Fast Forward and Rewind Functionality:** Here, two functions were defined to handle the fast forward and rewind actions. When the fast forward button is clicked, the `skipForward()` function adds 10 seconds to the current playback time. Similarly, when the rewind button is clicked, the `skipBackward()` function subtracts 10 seconds from the current playback time. Event listeners were attached to the fast forward and rewind buttons to trigger these functions when clicked.



```
1 // fastforward function handler
2 function skipForward() {
3     song.currentTime += 10; // Fast forward by 10 seconds
4 }
5
6 // rewind function handler
7 function skipBackward() {
8     song.currentTime -= 10; // Rewind by 10 seconds
9 }
10
11 // event listeners to the fast forward and rewind buttons/icons
12 fastForward.addEventListener('click', skipForward);
13 rewind.addEventListener('click', skipBackward);
14
```

Step 5: Test the Music Player

- This step was carried out for every step done above in order to review any errors or bugs in the code.
- The HTML file was opened in a web browser to test the music player.