HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">


<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>HarmonyHub - Music for everyone</title>
  <link rel="stylesheet" href="css/style.css">
  <link rel="stylesheet" href="css/utility.css">
  <link rel="stylesheet" href="css/responsive.css">
  <link rel="icon" type="png" href="favicon.png">
</head>


<body>
  <div class="container">


    <div class="left">
      <div class="leftNav">
        <div class="logo">
          <img height="35px" src="img/logo.png" alt="logo">
          <img id="cross" src="img/cross.svg" alt="">
        </div>
        <div class="home">
          <img height="25px" class="invert" src="img/home.svg" alt="">
          <p>Home</p>
        </div>
        <div class="search">
          <img class="invert" height="25px" src="img/search.svg" alt="">
          <p>Search</p>
        </div>
      </div>


      <div class="library">
        <div class="yourLibrary">
```

```
    <img height="30px" class="invert" src="img/library.svg" alt="">
    <p id="library">Your Library</p>
   </div>
   <div class="songs">
    <ul>
     <!-- Lists of Songs from Album Appear Here -->
    </ul>


   </div>
  </div>
  <div class="footer">
   <div class="links">
    <div>
     <a href="https://www.spotify.com/in-en/legal/">
      <span>Legal</span>
     </a>
    </div>
    <div>
     <a href="https://www.spotify.com/in-en/privacy/" target="_blank">
      <span>Privacy Center</span>
     </a>
    </div>
    <div>
     <a href="https://www.spotify.com/in-en/legal/privacy-policy/" target="_blank">
      <span>Privacy Policy</span>
     </a>
    </div>
    <div>
     <a href="https://www.spotify.com/in-en/legal/cookies-policy/" target="_blank">
      <span>Cookies</span>
     </a>
    </div>
    <div>
     <a href="https://www.spotify.com/in-en/legal/privacy-policy/#s3" target="_blank">
      <span>About Ads</span>
     </a>
    </div>
    <div>
     <a href="https://www.spotify.com/in-en/accessibility/" target="_blank">
```

```
            <span>Accessibility</span>
          </a>
        </div>


      </div>
      <div class="langBtn">
        <button id="langBtn"><svg class="invert" xmlns="http://www.w3.org/2000/svg"
width="24" height="24"
            viewBox="0 0 24 24" fill="none">
            <circle cx="12" cy="12" r="10" stroke="#000000" stroke-width="1.5" />
            <path d="M8 12C8 18 12 22 12 22C12 22 16 18 16 12C16 6 12 2 12 2C12 2 8
6 8 12Z" stroke="#000000"
              stroke-width="1.5" stroke-linejoin="round" />
            <path d="M21 15H3" stroke="#000000" stroke-width="1.5"
stroke-linecap="round" stroke-linejoin="round" />
            <path d="M21 9H3" stroke="#000000" stroke-width="1.5"
stroke-linecap="round" stroke-linejoin="round" />
          </svg>
          <p>English</p>
        </button>
      </div>
    </div>
  </div>




  <div class="right">
    <div class="header">
      <div class="leftPart">
        <button id="menu"><img src="img/hamburger.svg" alt="menu"></button>
        <button class="prevAlbum pointer">
          <svg class="invert" xmlns="http://www.w3.org/2000/svg" width="35" height="35"
viewBox="0 0 24 24"
            fill="none"
            <path d="M13.5 16C13.5 16 10.5 13.054 10.5 12C10.5 10.9459 13.5 8 13.5 8"
stroke="#000000"
              stroke-width="1.5" stroke-linecap="round" stroke-linejoin="round" />
          </svg></button>
```

```
        <button class="nextAlbum pointer">
          <svg class="invert" xmlns="http://www.w3.org/2000/svg" width="35" height="35"
viewBox="0 0 24 24"
            fill="none">
            <path d="M10.5 16C10.5 16 13.5 13.054 13.5 12C13.5 10.9459 10.5 8 10.5 8"
stroke="#000000"
              stroke-width="1.5" stroke-linecap="round" stroke-linejoin="round" />
            <path d="M13.5 16C13.5 16 10.5 13.054 10.5 12C10.5 10.9459 13.5 8 13.5 8"
              transform="scale(-1, 1) translate(-24, 0)" stroke="#000000" stroke-width="1.5"
stroke-linecap="round"
              stroke-linejoin="round" />
          </svg>
        </button>
      </div>
      <div class="rightPart">
        <button  class="pointer" id="signup">Sign Up</button>
        <button class="pointer" id="login">Log in</button>
      </div>
    </div>
    <div class="playlist">
      <h2 class="">HarmonyHub Playlists</h2>
      <div class="cardContainer">
        <!-- Album Cards Appear Here  -->
        <div class="rightFotter">
          <div class="leftSide">
            <div class="col1">
              <p>Company</p>
              <a href="https://www.spotify.com/in-en/about-us/contact/"
target="_blank"><p>About</p></a>
              <a href="https://www.lifeatspotify.com/" target="_blank"><p>Jobs</p></a>
              <a href="https://newsroom.spotify.com/" target="_blank"><p>For the
Record</p></a>
            </div>
            <div class="col2">
              <p>Communities</p>
              <a href="https://artists.spotify.com/home" target="_blank"><p>For
Artists</p></a>
              <a href="https://developer.spotify.com/"
target="_blank"><p>Developers</p></a>
```

```
            <a href="https://ads.spotify.com/en-IN/"
target="_blank"><p>Advertising</p></a>
            <a href="https://investors.spotify.com/home/default.aspx"
target="_blank"><p>Investors</p></a>
            <a href="https://spotifyforvendors.com/" target="_blank"><p>Vendors</p></a>
          </div>
          <div class="col3">
            <p>Useful links</p>
            <a href="https://support.spotify.com/in-en/"
target="_blank"><p>Support</p></a>
            <a href="https://www.spotify.com/in-en/free/" target="_blank"><p>Free Mobile
App</p></a>
          </div>
        </div>
        <div class="rightSide">
          <div><a href="https://www.instagram.com/allthingsaman/"
target="_blank"><img src="img/insta.svg" alt="instagram"></a></div>
          <div><a href="https://twitter.com/AmanUpa59504263" target="_blank"><img
src="img/x.svg" alt="twitter"></a></div>
          <div><a href="https://linkedin.com/in/allthingsaman" target="_blank"><img
src="img/fb.svg" alt="fscebook"></a></div>
        </div>
      </div>
      <div class="copyright">&copy 2024 HarmonyHub</div>
      <div class="playbar vHid">
        <div class="upperPart">
          <p class="songInfo">Click albums for Harmony</p>
          <div class="playPauseBtn">
           <img id="prev" src="img/previous.svg" alt="prev">
            <img id="play"  src="img/play.svg" alt="play">
            <img id="next"  src="img/next.svg" alt="next">
          </div>
          <div class="volume">
            <p class="timing"></p>

            <div class="vol">
           <img src="img/volume.svg" id="volIcon" alt="volume">
            <input type="range" name="volume" id="volume" min="0" max="100"></div>
          </div>
        </div>
```

```
          <div class="lowerPart">
          <div class="seekbar">
            <input type="range" name="seek" id="seek" min="0" max="100">
          </div></div>
        </div>
      </div>
    </div>
  </div>
  <script src="js/script.js"></script>
</body>


</html>
```

index.html :- Main Structure of my Web Application

This HTML document stands as the cornerstone of the HarmonyHub music player web application, meticulously crafted to usher users into an immersive world of musical delight. Let's embark on a detailed exploration of its intricate architecture and functionalities:

1. **Document Type Declaration (DOCTYPE):**
    - At the outset, we encounter the robust declaration of document type, ensuring strict adherence to HTML standards and compatibility across various browsers and platforms.

2. **HTML Structure:**
    - Nestled within the <html> tags lies the entire structural framework of our musical masterpiece, meticulously organized to ensure seamless user interaction.
    - The <head> section emerges as a bastion of metadata, housing critical directives such as character encoding specifications, viewport configurations for optimal display on diverse devices, and a captivating title that serves as a beacon, drawing users into the captivating world of HarmonyHub.

3. **Metadata Management:**
    - Herein lies the meticulous curation of metadata, an essential aspect ensuring the smooth operation and presentation of our web application.
    - External stylesheets, meticulously referenced, play a pivotal role in shaping the visual aesthetics of our creation, while the favicon serves as a distinctive marker amidst the labyrinth of browser tabs, reinforcing brand identity with each click.

4. **Body Content:**
    - As users traverse the digital landscape, they are greeted by two distinct realms – the "left" and the "right."
    - The "left" realm serves as a bastion of navigation, offering an array of options ranging from a majestic logo symbolizing the gateway to musical bliss, to intuitive navigation options such as home, search, and library, each beckoning users to embark on a voyage of musical discovery.

- Meanwhile, the "right" realm unfolds as the heart of our creation, showcasing a tapestry of headers, buttons, and playlists meticulously woven together to form the very fabric of musical indulgence.
- Not to be overlooked, the footer emerges as a humble yet essential component, housing an array of links guiding users to legal and privacy resources, language selectors facilitating linguistic versatility, and portals to the digital realm's social spheres.

5. **Script Integration:**
    - Ah, the script – the unseen conductor orchestrating the symphony of interactivity and functionality! At the culmination of the user's journey, the JavaScript file awaits, poised to infuse our creation with dynamic behavior, elevating the user experience to new heights with each interaction.

In essence, this HTML document serves as the foundational cornerstone of the HarmonyHub music player web application, embodying meticulous craftsmanship and boundless creativity, beckoning users to immerse themselves in the captivating world of music.

CSS (style.css)

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

* {
  margin: 0;
  padding: 0;
}
body {
  font-family: "Poppins", sans-serif;
  background-color: black;
  color: white;
}

/* Main Body */
.container {
  position: relative;
  display: flex;
  width: 100vw;
  gap: 15px;
  padding-top: 10px;
  padding-left: 5px;
  margin: auto;
}
/* Left Part */
.left {
  width: 30%;
  height: 98vh;
  display: flex;
  flex-direction: column;
  gap: 5px;
}
.leftNav {
  padding: 15px;
  display: flex;
  flex-direction: column;
  gap: 10px;
  height: 18%;
  background-color: #121212;
```

```
  border-radius: 10px;
}
.logo {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.home,
.search {
  display: flex;
  align-items: center;
  gap: 20px;
}
.library {
  height: 55%;
  background-color: #121212;
  border-radius: 10px;
}
.yourLibrary {
  padding: 15px;
  display: flex;
  align-items: center;
  gap: 10px;
}
.songs {
  height: 80%;
  overflow-y: auto;
  scrollbar-color: #353333 #000000;
  scrollbar-width: thin;
}
.gaana {
  cursor: pointer;
  margin: 5px;
  display: flex;
  align-items: center;
  justify-content: space-between;
  font-size: 10px;
  border: 1px solid white;
  color: #999595;
  padding-left: 2px;
```

```
  padding-right: 5px;
  height: 50px;
  border-radius: 10px;
  gap: 5px;
  transition: all 0.3s;
}
.gaana:hover {
  scale: 0.97;
  color: white;
  border: #999595;
}
.gaana > p {
  width: 200px;
  text-align: center;
}
.gaana > button {
  font-size: 12px;
  width: 27;
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #121212;
  border: none;
  color: white;
}
.footer {
  background-color: #121212;
  height: 20%;
  border-radius: 10px;
  padding: 0px 10px 0px 10px;
}
.footer > .links {
  padding: 10px;
  display: flex;
  gap: 5px;
  flex-wrap: wrap;
  justify-content: space-between;
  align-items: end;
}
.footer a {
```

```
  color: grey;
  font-size: 10px;
}
.langBtn {
  display: flex;
  justify-content: flex-start;
  align-items: center;
}
#langBtn {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 5px;
  background-color: black;
  color: rgb(167, 160, 160);
  border-radius: 20px;
  padding: 2px 5px 2px 5px;
  border: 1px solid rgb(189, 182, 182);
}
#langBtn:hover {
  background-color: black;
  color: white;
  border: 1px solid white;
}
/* Right Part */
.right {
  width: 70%;
}
.right > .header {
  background-color: #121212;
  border-radius: 10px 10px 0px 0px;
  display: flex;
  padding: 10px;
  align-items: center;
  justify-content: space-between;
}
.leftPart > button {
  border: none;
  background-color: #121212;
}
```

```
#signup {
  background-color: #121212;
  border: none;
  color: rgb(136, 136, 136);
  font-weight: 700;
  font-size: 14px;
  transition: all 0.1s linear;
}
#signup:hover {
  transform: scale(1.1);
  color: white;
}
#login {
  border: none;
  border-radius: 30px;
  padding: 12px 20px 12px 20px;
  margin-left: 20px;
  margin-right: 20px;
  font-weight: 700;
  font-size: 14px;
  transition: all 0.1s linear;
}
#login:hover {
  transform: scale(1.05);
}

.playlist {
  background-color: #1e1e1e;
  height: 88vh;
}
.playlist > h2 {
  display: inline-block;
  padding: 40px 20px 15px 20px;
}
.cardContainer {
  position: relative;
  height: 74vh;
  display: flex;
  flex-wrap: wrap;
  padding-left: 10px;
```

```
  gap: 15px;
  overflow-y: auto;
  scrollbar-color: #353333 #000000;
  scrollbar-width: thin;
}

.card {
  cursor: pointer;
  position: relative;
  display: flex;
  justify-content: center;
  width: 140px;
  height: 200px;
  flex-direction: column;
  align-items: center;
  padding: 8px;
  background-color: #141414;
  border-radius: 10px;
  font-size: 14px;
  transition: all 0.5s;
}
.card:hover {
  scale: 1.02;
  background-color: #555555;
}
.card:hover #hover {
  bottom: 78px;
  opacity: 1;
}
.card > div > img {
  height: 130px;
  border-radius: 5px;
}
#hover {
  height: 35px;
  position: absolute;
  bottom: 70px;
  right: 14px;
  opacity: 0;
  transition: all 0.3s ease-in-out;
```

```
}

.details {
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
}
.details:nth-child(1) {
  font-size: 14;
}
.details > p:nth-child(2) {
  font-size: 13;
  color: grey;
}

.rightFotter {
  width: 100%;
  display: flex;
  justify-content: space-between;
  padding: 100px 13px 50px 10px;
}
.leftSide {
  display: flex;
  justify-content: space-between;
  width: 70%;
  gap: 10px;
}
.leftSide div {
  display: flex;
  flex-direction: column;
  gap: 10px;
}
.leftSide div p {
  font-weight: 800;
}
.leftSide div a {
  color: grey;
}
.leftSide div a:hover {
  color: rgb(255, 255, 255);
```

```
}
.rightSide {
  display: flex;
  align-items: baseline;
  justify-content: space-between;
  width: 15%;
}
.copyright {
  color: grey;
}

.playbar {
  height: 12vh;
  background-color: rgba(233, 233, 233, 0.966);
  color: rgb(0, 0, 0);
  width: 99%;
  position: sticky;
  bottom: 0;
  border-radius: 8px;
  transition: all 0.3s linear;
}
.playbar:hover {
  background-color: white;
}
.upperPart {
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding-left: 5px;
  padding-right: 5px;
  height: 57px;
}
.songInfo {
  font-size: 12px;
  width: 20%;
  font-weight: 400;
  text-align: center;
}

.playPauseBtn {
```

```
  width: 118px;
  display: flex;
  justify-content: space-between;
}
#prev:hover,
#next:hover {
  background-color: #d1cbcbe7;
  border-radius: 50%;
  cursor: pointer;
}
.volume {
  display: flex;
  justify-content: space-between;
  align-items: center;
  width: 180px;
}
.volume > p {
  display: flex;
  justify-content: space-between;
  align-items: center;
  font-size: 14px;
  font-weight: 500;
  width: 90px;
}
.vol {
  display: flex;
  justify-content: center;
  align-items: center;
}
.vol > img:hover {
  background-color: #d1cbcbe7;
  /* border: 2px solid #797676; */
  border-radius: 50%;
}
.lowerPart {
  padding-top: 5px;
  width: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
```

```
  position: absolute;
  bottom: 0;
}
.seekbar {
  width: 95%;
}
```

style.css :- To Buitify our Web Application

This CSS file serves as the stylistic backbone of the HarmonyHub music player web application, meticulously defining the visual elements and layout that captivate users as they navigate through its enchanting interface. Let's delve into the intricacies of its styling:

1. **Font Import and Body Styling:**
    - The inclusion of the Poppins font family from Google Fonts imbues the application with a modern, sleek aesthetic.
    - The global reset of margin and padding for all elements ensures a consistent spacing and layout foundation across the application.
    - The body styling sets the stage for an immersive experience, with a black background serving as the canvas upon which the musical journey unfolds, complemented by crisp white text for optimal readability.

2. **Container and Layout:**
    - The container class establishes a responsive, flex-based layout, dynamically adjusting to different viewport sizes and orientations.
    - Within this container, the left and right sections are defined, each with distinct roles and styling.

3. **Left Section Styling:**
    - The left section, comprising navigation and library components, features a dark theme with subtle highlights, inviting users to explore its offerings.
    - Navigation elements such as the logo, home, and search icons are arranged with precision, enhancing both functionality and aesthetics.
    - The library component boasts a scrollbar for seamless navigation through the user's music collection, while individual song entries exude a tactile feel with hover effects and border styling.

4. **Footer Styling:**
    - The footer section offers a wealth of resources and social links, all presented in a clean, organized manner consistent with the application's design language.
    - Links are styled for easy identification and interaction, ensuring users can seamlessly access legal, privacy, and support information, as well as connect with the application's social media presence.

5. **Right Section Styling:**
    - The right section, housing playlist and playback controls, exudes a sense of elegance and functionality.
    - Playlists are displayed in a visually appealing grid layout, with each card inviting users to explore further through hover effects and detailed information.
    - Playback controls at the bottom provide users with intuitive access to essential functions such as play, pause, previous, and next, all seamlessly integrated into the application's visual aesthetic.

6. **Playbar Styling:**
    - The playbar section, positioned at the bottom of the interface, features a light background to differentiate it from the main content area.
    - Essential playback information and controls are neatly arranged within this section, ensuring users can effortlessly manage their listening experience.

In essence, this CSS file represents a harmonious blend of form and function, meticulously crafted to enhance user engagement and elevate the HarmonyHub music player web application to new heights of visual sophistication and usability.

CSS (responsive.css)

```
/* Responsive Breakpoints */
@media screen and (max-width: 1025px) {
  /* Extra large devices (large desktops) */
  .leftSide{
    gap: 25px;
  }
}

@media screen and (max-width: 992px) {
  /* Large devices (desktops) */
   .right{
    width: 100%;
  }
  #menu{
    display: inline-block;
  }
  #cross{
    display: inline-block;
}
  .playlist{
    height: 86vh;
  }
  .cardContainer{
    height: 70vh;
  }

  .left{
    display: none;
    position: absolute;
    z-index: 2;
    /* right: 113%; */
    width: 350px;
    background-color: rgb(0, 0, 0);
   }
   .rightFotter{
    flex-direction: column;
   }
   .leftSide{
```

```
    width: 100%;
   }
}

@media screen and (max-width: 576px) {
  /* Small devices (phones) */
  .playbar{
  height: 25vh;
  padding: auto;
  }
  .card{
    justify-content: space-around;
    width: 90%;
    height: 100vh;
    font-size: 190%;
  }
  .card>div>img{
    width: 100% !important;
    height: auto;
  }
  .card>div{
    object-fit: cover;
  }
  .upperPart{
    flex-direction: column;
    height: auto;
  }
  .songInfo{
    font-size: 20px;
    width: 80%;
    height: 12vh;
    display: flex;
    justify-content: center;
    align-items: center;
  }
  .volume{
    display: flex;
    width: 90%;
    justify-content: space-between;
  }
```

```
  #hover{
    height: 100px;
  }

}
```

responsive.css :- Good Look & Feel for all Users

This responsive.css file is crucial for ensuring that the HarmonyHub music player web application maintains optimal functionality and aesthetics across a variety of devices and screen sizes. Let's explore its key features:

1. **Responsive Breakpoints:**
    - The media queries defined in this file target specific device widths, allowing for custom styling adjustments based on screen size
      .
2. **Large Desktops and Desktops:**
    - At widths up to 1025px, adjustments are made to the layout of the leftSide element, increasing the gap between its child elements for improved spacing and readability.
    - For screens up to 992px wide, the right section expands to occupy the full width of the viewport, ensuring a seamless viewing experience on large desktops.
    - Additionally, the menu and cross icons become visible to facilitate navigation, while the playlist and card container heights are adjusted to accommodate varying screen sizes.

3. **Small Devices (Phones):**
    - When the screen width is below 576px, catering to small devices such as phones, extensive styling modifications are implemented to enhance usability.
    - The playbar height is reduced and centered vertically, optimizing its visibility and accessibility on smaller screens.
    - Card elements undergo significant adjustments, with increased font size, spacing, and image dimensions to improve legibility and visual appeal on smaller devices.
    - The upper part of the playbar transitions to a vertical layout, ensuring that playback information remains clear and accessible.
    - Volume controls and hover elements are adjusted to maintain functionality and usability across a range of screen sizes.

In summary, the responsive.css file plays a vital role in ensuring that the HarmonyHub music player adapts seamlessly to different devices and screen sizes, providing users with a consistent and engaging experience regardless of the platform they choose to engage with.

CSS (utility.css)

```
.dNone{
    display: none;
}
.vHid{
    visibility: hidden;
}

.invert{
    filter: invert(1);
}
.border{
    border: 2px solid red;
}
.flex{
    display: flex;
}
.alignItems{
    align-items: center;
}
.cGrey{
  color: grey;
}
.rel{
    position: relative;
}
.pointer{
    cursor: pointer;
}
input[id="seek"] {
    -webkit-appearance: none;
    width: 99%;
    height: 10px;
    background: #838282;
    outline: none;
    opacity: 0.7;
    -webkit-transition: 0.2s;
    cursor: pointer;
    transition: opacity 0.2s;
```

```
    border-radius: 5px;

}

/* Stylish thumb handle */
input[id="seek"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 20px;
    height: 20px;
    background: #52d657;
    cursor: pointer;
    border-radius: 50%;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.2);
    transition: all 0.3s ease;
}

/* Hover effect for better visibility */
input[id="seek"]:hover {
    opacity: 1;
}
input[id="volume"] {
    width: 70px;
    height: 5px;

}

#menu{
    display: none;
}
#cross{
    display: none;
}
.radius{
    border-radius: 50%;
}
```

utility.css :- Consistency, Efficiency, Reusability

The utility.css file contains a collection of utility classes that are used to apply specific styles or behaviors to elements throughout the HarmonyHub music player web application. Here's a breakdown of its key features:

1. **Display Utilities:**
    - .dNone: Hides elements by setting their display property to "none".
    - .vHid: Makes elements invisible by setting their visibility property to "hidden".
    - .flex: Sets the display property to "flex" for flexible layout.
    - .alignItems: Aligns flex items along the cross axis.
    - .rel: Sets the position property to "relative" for relative positioning.

2. **Styling Utilities:**
    - .invert: Applies an invert filter to elements, inverting their colors.
    - .border: Adds a 2px solid red border to elements.
    - .cGrey: Sets the text color to grey.

3. **Interaction Utilities:**
    - .pointer: Sets the cursor property to "pointer" to indicate interactive elements.
    - Custom styling for input elements:
        - input[id="seek"]: Styles the seekbar input element with a customized appearance and thumb handle.
        - input[id="volume"]: Styles the volume input element with a specific width and height.

4. **Layout Utilities:**
    - .radius: Applies a border-radius of 50% to elements, creating a circular shape.

5. **Navigation Controls:**
    - **#menu** and **#cross**: Initially hidden navigation icons that can be displayed when needed, likely for responsive navigation menus.

These utility classes provide developers with a convenient way to apply consistent styling and behavior across various elements within the HarmonyHub music player application, enhancing its usability and aesthetics.

JavaScript (script.js)

```
let currAudio = new Audio();
let currFolder;
let songs;
const songInfo = document.querySelector(`.songInfo`);
const timing = document.querySelector(`.timing`);
const seekCircle = document.getElementById(`seek`);
seekCircle.value = 0;
const volume = document.getElementById(`volume`);
volume.value = 100;
const volIcon = document.querySelector(`#volIcon`);
const playbar = document.querySelector(`.playbar`);
let folderIndex = 5;
let index;

// Step 1: To display album on website by scanning the songs folder
const displayAlbums = async () => {
  let cardContainer = document.querySelector(`.cardContainer`);
  // Get all folders in "Songs" directory
  let responce = await fetch(`/songs/`);
  let text = await responce.text();
  let doc = document.createElement(`div`);
  doc.innerHTML = text;
  let anchors = Array.from(doc.querySelectorAll(`a`));
  let folders = [];
  for (let i = 0; i < anchors.length; i++) {
    const anchor = anchors[i];
    if (anchor.href.includes(`/songs/`)) {
      folders.push(anchor.href.split("/").slice(-2, -1)[0]);
    }
  }
  console.log(folders);

  for (const folder of folders) {
    let card = document.createElement("div");
    let responce = await fetch(`/songs/${folder}/info.json`);
    let folderInfo = await responce.json();
    card.innerHTML = `
        <div><img src="/songs/${folder}/cover.jpg" alt="cover"></div>
```

```
        <div class="details">
          <p>${folderInfo.title}</p>
          <p>${folderInfo.description}</p>
          <div class="hover ">
          <img id="hover" src="img/hover.png" alt="">
        </div>
        </div>`;
    card.classList.add("card");
    card.dataset.folder = folderInfo.name;
    cardContainer.prepend(card);

    card.addEventListener(`click`, (e) => {
      loadMusic(e.currentTarget.dataset.folder);
      folderIndex = folders.indexOf(e.currentTarget.dataset.folder);
    });
  }

  // Album mai agea piche hone ke liye upar jo 2 buttons hai usse index 5 se start hai 0 ki wajah kyuki prepend kiya tha isliye songs folder ke song wale cards ulte order mai lage hai website pe
  let prevAlbum = document.querySelector(`.prevAlbum`);
  prevAlbum.addEventListener(`click`, (e) => {
    console.log(`prevAlbum is clicked`);
    if (folderIndex < 5) {
      console.log(`select ${folderIndex + 1} album`);
      folderIndex++;
      loadMusic(folders[folderIndex]);
    }
  });
  let nextAlbum = document.querySelector(`.nextAlbum`);
  nextAlbum.addEventListener(`click`, (e) => {
    console.log(`nextAlbum is clicked`);
    if (folderIndex > 0) {
      console.log(`select ${folderIndex - 1} album`);
      folderIndex--;
      loadMusic(folders[folderIndex]);
    }
  });
};
```

```
// album pe click karn pe jo puri songs ki list library mai add hogi
const loadMusic = async (folder) => {
  currFolder = folder;
  let songList = document.querySelector(`.songs ul`);
  songList.innerHTML = ``;

  let responce = await fetch(`/songs/${folder}/`);
  let text = await responce.text();
  let doc = document.createElement(`div`);
  doc.innerHTML = text;
  let anchors = Array.from(doc.querySelectorAll(`a`));
  songs = [];
  for (let i = 0; i < anchors.length; i++) {
    const anchor = anchors[i];
    if (anchor.href.endsWith(`.mp3`)) {
      songs.push(decodeURI(anchor.href.split("/").slice(-1)[0]));
    }
  }

  // play first audio of album on clicking and update other things
  playMusic(folder, songs[0]);
  index = 0;
  play.src = "img/pause.svg";
  playbar.classList.remove(`vHid`);
  let songName = decodeURI(currAudio.src.split("/").slice(-1)[0]);
  console.log(songs);
  console.log(songName);
  index = songs.indexOf(songName);
  console.log(index);

  let songLi;
  for (let i = 0; i < songs.length; i++) {
    const song = songs[i];
    songLi = document.createElement(`li`);
    songLi.innerHTML = `
    <svg class="invert" xmlns="http://www.w3.org/2000/svg" width="24" height="44"
viewBox="0 0 24 24"
    fill="none">
    <circle cx="6.5" cy="18.5" r="3.5" stroke="#000000" stroke-width="1.5" />
    <circle cx="18" cy="16" r="3" stroke="#000000" stroke-width="1.5" />
```

```
    <path
      d="M10 18.5L10 7C10 6.07655 10 5.61483 10.2635 5.32794C10.5269 5.04106
11.0175 4.9992 11.9986 4.91549C16.022 4.57222 18.909 3.26005 20.3553
2.40978C20.6508 2.236 20.7986 2.14912 20.8993 2.20672C21 2.26432 21 2.4315 21
2.76587V16"
      stroke="#000000" stroke-width="1.5" stroke-linecap="round" stroke-linejoin="round"
/>
    <path d="M10 10C15.8667 10 19.7778 7.66667 21 7" stroke="#000000"
stroke-width="1.5"
      stroke-linecap="round" stroke-linejoin="round" />
  </svg>
  <p>${song}</p>
  <button class="playNow flex"><a class="download" href="songs/${folder}/${song}"
download><img height="25px" class="invert" src="img/download.svg"
alt="download"></a></button>
      `;
    songLi.classList.add("gaana");
    songList.append(songLi);
    let download = document.querySelectorAll(`.download`);
    download.forEach((e) => {
      e.addEventListener(`click`, (e) => {
        e.stopPropagation();
        console.log(`download`);
      });
    });
  }
  let sList = document.querySelectorAll(`.gaana`);
  sList.forEach((li) => {
    li.addEventListener(`click`, (e) => {
      play.src = "img/pause.svg";
      playMusic(currFolder, e.currentTarget.querySelector(`p`).textContent);
      console.log(
        songs.indexOf(e.currentTarget.querySelector(`p`).textContent)
      );
      index = songs.indexOf(e.currentTarget.querySelector(`p`).textContent);
    });
  });
};

const playMusic = (file, audio) => {
```

```
  currAudio.pause();
  currAudio = new Audio(`/songs/${file}/${audio}`);
  currAudio.play();
  currAudio.volume = volume.value / 100;
  songInfo.textContent = audio;
  //timeupdate event
  currAudio.addEventListener("timeupdate", () => {
    timing.innerHTML = `<p>${timeFormat(
      currAudio.currentTime
    )}</p><p>/${timeFormat(currAudio.duration)}</p>`;
    seekCircle.value = `${(currAudio.currentTime / currAudio.duration) * 100}`;
  });
  seekCircle.addEventListener("input", (event) => {
    // Handle seeking when the user interacts with the seekbar
    const seekPercentage = event.target.value;
    const newTime = (seekPercentage / 100) * currAudio.duration;
    currAudio.currentTime = newTime;
  });

  // Next audio of album automatically playing
  currAudio.addEventListener("ended", () => {
    console.log(`next song playing`);
    if (index + 1 < songs.length) {
      index++;
      playMusic(currFolder, songs[index]);
    } else {
      play.src = "img/play.svg";
    }
  });
};

const timeFormat = (seconds) => {
  if (isNaN(seconds) || seconds < 0) {
    return "00:00";
  }
  const minu = Math.floor(seconds / 60);
  const remainingSec = Math.floor(seconds % 60);
  const formattedMin = String(minu).padStart(2, "0");
  const formattedSec = String(remainingSec).padStart(2, "0");
  return `${formattedMin}:${formattedSec}`;
```

```
};

async function main() {
  displayAlbums();

  play = document.querySelector(`#play`);
  play.addEventListener(`click`, (e) => {
    if (currAudio.paused) {
      currAudio.play();
      play.src = "img/pause.svg";
    } else {
      currAudio.pause();
      play.src = "img/play.svg";
    }
  });

  // Handle volume when the user interacts with the volume baar
  volume.addEventListener("input", (e) => {
    currAudio.volume = e.target.value / 100;
    if (currAudio.volume === 0) {
      volIcon.src = `img/mute.svg`;
    } else {
      volIcon.src = `img/volume.svg`;
    }
  });

  volIcon.addEventListener(`click`, (e) => {
    if (volIcon.src.includes("volume")) {
      volIcon.src = `img/mute.svg`;
      currAudio.volume = 0;
      volume.value = 0;
    } else {
      volIcon.src = `img/volume.svg`;
      currAudio.volume = 0.2;
      volume.value = 20;
    }
  });
  let hamburger = document.querySelector(`#menu`);
  hamburger.addEventListener(`click`, (e) => {
    document.querySelector(`.left`).style.display = `inline-block`;
```

```
  });
  let cross = document.querySelector(`#cross`);
  cross.addEventListener(`click`, (e) => {
    document.querySelector(`.left`).style.display = `none`;
  });

  //  Handling prev next button to go forward and backward in library
  const prev = document.querySelector(`#prev`);
  prev.addEventListener("click", (e) => {
    console.log("Previous button clicked");
    if (index - 1 >= 0) {
      index--; // Update the index
      play.src = "img/pause.svg";
      playMusic(currFolder, songs[index]);
    }
  });
  const next = document.querySelector(`#next`);
  next.addEventListener("click", (e) => {
    console.log("Next button clicked");
    if (index + 1 < songs.length) {
      index++; // Update the index
      play.src = "img/pause.svg";
      playMusic(currFolder, songs[index]);
    }
  });
}
main();
```

script.js :- Soul of our web App

The provided JavaScript code constitutes a comprehensive music player application. Let's delve into its functionality and structure:

1. **Initialization and Variables:**
    - **currAudio:** Represents the current audio element.
    - **currFolder:** Stores the current music folder.
    - **songs:** Holds the list of songs.
    - **songInfo:** References the element displaying song information.
    - **timing:** References the element showing playback timing.
    - **seekCircle:** References the seek bar element.
    - **volume:** References the volume control element.
    - **volIcon:** References the volume icon.
    - **playbar:** References the player's bar.
    - **folderIndex:** Index of the current folder being displayed.
    - **index:** Index of the currently playing song in the songs array.

2. **Display Albums:**
    - displayAlbums: Fetches data about albums from the server and dynamically generates HTML elements to display them on the webpage.
    - Utilizes fetch to retrieve information about albums and their covers.
    - Generates album cards with event listeners for selecting an album.

3. **Load Music:**
    - loadMusic: Loads music from a selected album.
    - Fetches the list of songs from the server and dynamically generates HTML elements to display them.
    - Handles click events on song items to play the selected song.

4. **Play Music:**
    - playMusic: Plays the selected music track.
    - Updates the current audio source and plays the audio.
    - Handles seeking, volume adjustment, and autoplay for the next track.

5. **Helper Functions**:
    - timeFormat: Formats time in seconds to MM:SS format.

6. **Event Listeners:**
    - Play/pause button click event.
    - Volume control input event.
    - Volume icon click event.
    - Previous/next button click events.
    - Hamburger menu and close menu button click events.

7. **Initialization:**
    - main: Entry point of the script.
    - Calls displayAlbums to load albums.
    - Sets up event listeners for player controls.

This JavaScript code demonstrates a robust music player application with features like album selection, song listing, playback control, seeking, and volume adjustment. It effectively integrates server-side data fetching and client-side manipulation to provide an interactive user experience.

Functionality of Functions:-

---

1. playMusic (file,audio):

The playMusic function handles the playback of music tracks. Let's break down its functionality:

```
const playMusic = (file, audio) => {
  currAudio.pause();
  currAudio = new Audio(`/songs/${file}/${audio}`);
  currAudio.play();
  currAudio.volume = volume.value / 100;
  songInfo.textContent = audio;

  currAudio.addEventListener("timeupdate", () => {
    timing.innerHTML = `<p>${timeFormat(
      currAudio.currentTime
    )}</p><p>/${timeFormat(currAudio.duration)}</p>`;
    seekCircle.value = `${(currAudio.currentTime /
currAudio.duration) * 100}`;
  });
  seekCircle.addEventListener("input", (event) => {
    const seekPercentage = event.target.value;
    const newTime = (seekPercentage / 100) *
currAudio.duration;
    currAudio.currentTime = newTime;
  });
  currAudio.addEventListener("ended", () => {
    if (index + 1 < songs.length) {
      index++;
      playMusic(currFolder, songs[index]);
    } else {
      play.src = "img/play.svg";}});};
```

1. **Pause Current Audio:** Initially, it pauses the currently playing audio to avoid overlapping playback.

2. **Set New Audio Source:** It sets the source of the currAudio element to the selected music file (/songs/${file}/${audio}).

3. **Play New Audio:** Once the new audio source is set, it plays the audio.

4. **Set Volume:** It adjusts the volume of the audio player based on the current value of the volume control (volume.value). The volume is set as a percentage of the maximum volume (100).

5. **Update Song Information:** It updates the songInfo element to display the name of the currently playing audio.

6. **Time Update Event:** It listens for the timeupdate event on the audio player. This event is triggered as the playback position of the audio changes. Upon receiving this event, it updates the UI to reflect the current playback time and total duration of the audio track. This information is displayed in the timing element.

7. **Seek Bar Interaction:** It adds an event listener to the seekCircle element, which represents the seek bar. When the user interacts with the seek bar (e.g., dragging the seek handle), it calculates the new playback time based on the percentage of the seek bar's position relative to its total length. It then updates the playback position of the audio accordingly.

8. **Auto Play Next Track:** It listens for the ended event on the audio player. This event is triggered when the current audio track finishes playing. If there are more tracks in the playlist (songs array), it automatically plays the next track by recursively calling the playMusic function with the next track's information (folder and filename). If the current track is the last one in the playlist, it stops playback and updates the play/pause button (play.src) to display the play icon.

Overall, the playMusic function manages the playback of audio tracks, updates UI elements to reflect playback status and provides user interaction for seeking through tracks.

2. loadMusic (folder):

The loadMusic function is responsible for loading and displaying the music tracks associated with a specific folder (album). Let's break down its functionality:

```
const loadMusic = async (folder) => {
  currFolder = folder;
  let songList = document.querySelector(`.songs ul`);
  songList.innerHTML = ``;

  let responce = await fetch(`/songs/${folder}/`);
  let text = await responce.text();
  let doc = document.createElement(`div`);
  doc.innerHTML = text;
  let anchors = Array.from(doc.querySelectorAll(`a`));
  songs = [];
  for (let i = 0; i < anchors.length; i++) {
    const anchor = anchors[i];
    if (anchor.href.endsWith(`.mp3`)) {

songs.push(decodeURI(anchor.href.split("/").slice(-1)[0]));
    }
  }
```

1. **Set Current Folder:** It sets the currFolder variable to the folder passed as an argument to the function.

2. **Clear Song List:** It clears the content of the <ul> element inside the element with the class .songs. This ensures that any previously displayed songs are removed before loading the new ones.

3. **Fetch Songs:** It sends a fetch request to the server to retrieve the list of songs (MP3 files) within the specified folder (/songs/${folder}/).

4. **Extract Song Names:** After receiving the response, it parses the HTML content to extract the names of the MP3 files using a regular expression. These names are then decoded and stored in the songs array.

5. **Play First Audio:** It calls the playMusic function to play the first audio track in the album. This function is also responsible for updating various UI elements such as the play/pause button and the seek bar.

6. **Update Index and UI:** It updates the index variable to reflect the index of the currently playing song in the songs array. Additionally, it updates the UI elements such as the play button icon and shows the playbar by removing the vHid class.

7. **Create Song List:** It dynamically creates the list of songs in the album by iterating over the songs array. For each song, it creates an <li> element containing the song name and a download button.

8. **Download Button:** It adds an event listener to each download button to handle the download functionality. When clicked, it prevents the default action and logs a message to the console.

9. **Song Click Event:** It adds an event listener to each song item (<li> element) in the song list. When clicked, it triggers the playMusic function to play the selected song, updates the play button icon, and sets the index variable to the index of the selected song in the songs array.

Overall, the loadMusic function efficiently loads and displays the songs of a selected album, enabling users to interact with them through the UI.

3. displayAlbum ( ):

The displayAlbums function is responsible for dynamically displaying albums on a website by scanning the songs folder. Let's break down its functionality:

```
const displayAlbums = async () => {
  let cardContainer =
document.querySelector(`.cardContainer`);
  // Get all folders in "Songs" directory
  let responce = await fetch(`/songs/`);
  let text = await responce.text();
  let doc = document.createElement(`div`);
  doc.innerHTML = text;
  let anchors = Array.from(doc.querySelectorAll(`a`));
  let folders = [];
  for (let i = 0; i < anchors.length; i++) {
    const anchor = anchors[i];
    if (anchor.href.includes(`/songs/`)) {
      folders.push(anchor.href.split("/").slice(-2, -1)[0]);
    }
  }

  for (const folder of folders) {
    let card = document.createElement("div");
    let responce = await fetch(`/songs/${folder}/info.json`);
    let folderInfo = await responce.json();
    card.innerHTML = `
        <div><img src="/songs/${folder}/cover.jpg"
alt="cover"></div>
        <div class="details">
          <p>${folderInfo.title}</p>
          <p>${folderInfo.description}</p>
          <div class="hover ">
          <img id="hover" src="img/hover.png" alt="">
```

```
            </div>
            </div>`;
        card.classList.add("card");
        card.dataset.folder = folderInfo.name;
        cardContainer.prepend(card);

        card.addEventListener(`click`, (e) => {
          loadMusic(e.currentTarget.dataset.folder);
          folderIndex =
folders.indexOf(e.currentTarget.dataset.folder);
        });
      }

      let prevAlbum = document.querySelector(`.prevAlbum`);
      prevAlbum.addEventListener(`click`, (e) => {
        if (folderIndex < 5) {
          folderIndex++;
          loadMusic(folders[folderIndex]);
        }
      });
      let nextAlbum = document.querySelector(`.nextAlbum`);
      nextAlbum.addEventListener(`click`, (e) => {
        if (folderIndex > 0) {
          folderIndex--;
          loadMusic(folders[folderIndex]);
        }
      });
};
```

1. **Select Card Container:** It selects the container element where the album cards will be displayed using document.querySelector('.cardContainer').

2. **Fetch Folders:** It sends a fetch request to the server to retrieve the list of folders (albums) within the "Songs" directory (/songs/).

3. **Parse Folder Names:** After receiving the response, it parses the HTML content to extract the names of the folders using a regular expression. These folder names are then stored in the folders array.

4. **Create Album Cards:** It iterates over each folder in the folders array and dynamically creates a card for each album. Each card contains an image, title, description, and a hover effect.

5. **Add Event Listeners:** It adds event listeners to each album card. When clicked, it triggers the loadMusic function to load and display the songs of the selected album. It also updates the folderIndex variable to keep track of the current album index.

6. **Previous and Next Buttons:** It adds event listeners to the previous and next album buttons (prevAlbum and nextAlbum). When clicked, these buttons allow users to navigate between albums by updating the folderIndex and loading the corresponding album using the loadMusic function.

Overall, the displayAlbums function dynamically generates album cards based on the folders present in the "Songs" directory and enables users to navigate between albums using the provided buttons.

4. main ( ):

The main function is the entry point of the application. Here's a breakdown of its functionality:

```
async function main() {
  displayAlbums();

  play = document.querySelector(`#play`);
  play.addEventListener(`click`, (e) => {
    if (currAudio.paused) {
      currAudio.play();
      play.src = "img/pause.svg";
    } else {
      currAudio.pause();
      play.src = "img/play.svg";
    }
  });

  volume.addEventListener("input", (e) => {
    currAudio.volume = e.target.value / 100;
    if (currAudio.volume === 0) {
      volIcon.src = `img/mute.svg`;
    } else {
      volIcon.src = `img/volume.svg`;
    }
  });

  volIcon.addEventListener(`click`, (e) => {
    if (volIcon.src.includes("volume")) {
      volIcon.src = `img/mute.svg`;
      currAudio.volume = 0;
      volume.value = 0;
    } else {
      volIcon.src = `img/volume.svg`;
```

```
      currAudio.volume = 0.2;
      volume.value = 20;
    }
  });

  let hamburger = document.querySelector(`#menu`);
  hamburger.addEventListener(`click`, (e) => {
    document.querySelector(`.left`).style.display =
`inline-block`;
  });
  let cross = document.querySelector(`#cross`);
  cross.addEventListener(`click`, (e) => {
    document.querySelector(`.left`).style.display = `none`;
  });

  const prev = document.querySelector(`#prev`);
  prev.addEventListener("click", (e) => {
    if (index - 1 >= 0) {
      index--;
      play.src = "img/pause.svg";
      playMusic(currFolder, songs[index]);
    }
  });

  const next = document.querySelector(`#next`);
  next.addEventListener("click", (e) => {
    if (index + 1 < songs.length) {
      index++;
      play.src = "img/pause.svg";
      playMusic(currFolder, songs[index]);
    }
  });
}
```

1. **Display Albums**: It calls the displayAlbums function to display the albums on the website.

2. **Play Button Event Listener:** It adds an event listener to the play button **(#play)**. When clicked, it toggles between playing and pausing the current audio. It also updates the play button's icon accordingly.

3. **Volume Input Event Listener:** It adds an event listener to the volume bar **(volume)**. When the user adjusts the volume, it updates the volume level of the current audio. It also changes the volume icon to indicate the mute or unmute state.

4. **Volume Icon Click Event Listener:** It adds an event listener to the volume icon **(volIcon)**. When clicked, it toggles between muting and unmuting the audio. It adjusts the volume level and updates the volume bar accordingly.

5. **Hamburger Menu Click Event Listener:** It adds an event listener to the hamburger menu icon **(#menu)**. When clicked, it displays the left-side menu by setting its display property to inline-block.

6. **Cross Icon Click Event Listener:** It adds an event listener to the cross icon **(#cross).** When clicked, it hides the left-side menu by setting its display property to none.

7. **Previous Button Click Event Listener:** It adds an event listener to the previous button **(#prev)**. When clicked, it plays the previous song in the album by updating the index and calling the playMusic function.

8. **Next Button Click Event Listener:** It adds an event listener to the next button **(#next)**. When clicked, it plays the next song in the album by updating the index and calling the playMusic function.

Overall, the main function orchestrates the main functionalities of the music player application, including playing/pausing audio, adjusting volume, navigating between songs, and displaying the left-side menu.

5. timeFormat (seconds):

The timeFormat function is responsible for converting a given duration in seconds into a formatted time string in the format "MM:SS" (minutes:seconds). Here's how it works:

```
const timeFormat = (seconds) => {
  if (isNaN(seconds) || seconds < 0) {
    return "00:00";
  }
  const minu = Math.floor(seconds / 60);
  const remainingSec = Math.floor(seconds % 60);
  const formattedMin = String(minu).padStart(2, "0");
  const formattedSec = String(remainingSec).padStart(2, "0");
  return `${formattedMin}:${formattedSec}`;
};
```

1. **Input Validation:** It first checks if the input seconds is not a number (NaN) or if it's less than zero. If either condition is true, it returns the default time string "00:00".

2. **Conversion to Minutes and Seconds:** If the input seconds is valid, it proceeds to calculate the number of minutes and remaining seconds. It divides the total seconds by 60 to get the minutes (minu) and calculates the remaining seconds (remainingSec) using the modulo operator %.

3. **Formatting:** It then converts the minutes and remaining seconds into strings (formattedMin and formattedSec) and pads them with leading zeros if necessary to ensure they are two digits long.

4. **Return:** Finally, it combines the formatted minutes and seconds into a string in the format "MM:SS" and returns the result.

Overall, the timeFormat function ensures that the duration in seconds is correctly formatted into a human-readable time string.