JORNADA FULL STACK IMPRESSIONADOR

Aula 3

Aula 3
Aula 3

Aula 3 – Tornando os Componentes Mais Dinâmicos

No código anterior, o componente Artist era muito simples: ele apenas renderizava um com o texto "Artist". Ou seja, não havia nenhuma lógica, estilização ou reaproveitamento de estrutura — apenas uma marcação fixa.

```jsx
import React from "react";

const Artist = () => {
  return <div>Artist</div>;
};

export default Artist;
```

Vamos alterar para:

```jsx
import React from "react";
import Main from "../components/Main";

const Artists = () => {
  return <Main type="artists" />;
};

export default Artists;
```

- Foi criado um componente mais dinâmico, utilizando o componente Main.
- Em vez de escrever o conteúdo diretamente no Artists, agora o componente Main é chamado e recebe uma **propriedade (prop)** chamada type, com valor "artists".
- Com isso, o comportamento e o conteúdo do Artists passam a depender da forma como o Main utiliza essa prop (type).

Essa mudança traz **várias melhorias**:

- **Reaproveitamento de código**: Main pode ser usado para renderizar diferentes páginas apenas mudando o valor da prop type.
- **Organização**: Cada página fica mais limpa e fácil de entender.
- **Escalabilidade**: Se precisarmos adicionar novos tipos (por exemplo, "albums", "songs", etc.), podemos usar o mesmo Main passando diferentes types, sem precisar criar novos componentes do zero.

Aula 3 – Tornando os Componentes Mais Dinâmicos

Seguindo o mesmo raciocínio da mudança anterior, antes, o componente Songs simplesmente retornava um com o texto "Songs", sem reaproveitar nenhuma estrutura ou lógica. Era um componente totalmente **estático** e **limitado**.

```jsx
import React from "react";

const Songs = () => {
  return <div>Songs</div>;
};

export default Songs;
```

Vamos alterar para:

```jsx
import React from "react";
import Main from "../components/Main";

const Songs = () => {
  return <Main type="songs" />;
};

export default Songs;
```

Agora:

- O componente Songs utiliza o componente Main.
- O type passado como prop ("songs") indica para o Main qual tipo de conteúdo ele deve exibir.
- A lógica de apresentação (layout, estilização, listagem, etc.) está centralizada dentro do Main, tornando o Songs um componente muito mais **enxuto** e **fácil de manter**.

Principais vantagens dessa refatoração:

- **Reaproveitamento de estrutura**: O mesmo Main atende várias páginas (como Artists, Songs, etc.).
- **Facilidade de manutenção**: Alterações na estrutura geral são feitas apenas no Main.
- **Facilidade para escalar**: Adicionar novas páginas (Albums, Playlists, etc.) exige mínimo esforço, apenas alterando a prop type.
- **Organização do projeto**: Menos duplicação de código e mais consistência entre as páginas.

```jsx
const Main = () => {
  return (
    <div className="main">
      {/* Item List de Artistas */}
      <ItemList
        title="Artistas"
        items={10}
        itemsArray={artistArray}
        path="/artists"
        idPath="/artist"
      />

      {/* Item List de Músicas */}
      <ItemList
        title="Músicas"
        items={20}
        itemsArray={songsArray}
        path="/songs"
        idPath="/song"
      />
    </div>
  );
};
```

Agora que já entendemos como criar componentes no React, vamos dar mais um passo importante:

Vamos aplicar uma lógica dentro do nosso componente Main para que ele consiga se adaptar dependendo da informação que receber.
Ou seja, o Main vai deixar de ser fixo, e vai passar a ser **dinâmico**: ele vai identificar se deve exibir **artistas**, **músicas** ou até outros tipos no futuro.

O problema do código antigo

Antes, o Main exibia **sempre** duas listas:
- uma lista de artistas
- uma lista de músicas

Mesmo que a página que o usuário estivesse acessando fosse apenas de artistas, a lista de músicas ainda era carregada na tela (mesmo que escondida depois com CSS, era processada).

Isso deixava o projeto:
- menos performático
- mais difícil de escalar
- com código menos flexível

Aula 3 – Tornando os Componentes Mais Dinâmicos

A solução: usar **props** e **renderização condicional**

A ideia agora é:

- passar uma **prop** chamada type para o Main
- dentro do Main, verificar qual o valor de type
- renderizar apenas o conteúdo que corresponde a esse tipo

Se o type for "artists", mostramos só os artistas.
Se for "songs", mostramos só as músicas.

O que acontece aqui:

- **({ type })** → Estamos recebendo a prop type no componente.
- **{type === "artists" ? (...) : null}** →Se o type for "artist", renderiza a lista de artistas.
- Se não for, não renderiza nada (null).
- **{type === "songs" ? (...) : null}** → Se o type for "songs", renderiza a lista de músicas.

Assim, o Main se adapta automaticamente ao tipo de conteúdo que queremos exibir!

Benefícios dessa abordagem

- Deixamos o componente **inteligente** e **reutilizável**.
- Evitamos carregamento desnecessário de dados.
- Facilitamos a manutenção: agora, se quisermos criar páginas para **álbuns**, **podcasts**, **playlists**, etc., basta enviar um novo type e configurar o que será renderizado.

```
const Main = ({ type }) => {
  return (
    <div className="main">

      {/* Se o type for artists ou não for informado, mostramos a lista de artistas */}
      {type === "artists" || type === undefined ? (
        <ItemList
          title="Artistas"
          items={10}
          itemsArray={artistArray}
          path="/artists"
          idPath="/artist"
        />
      ) : null}

      {/* Se o type for songs ou não for informado, mostramos a lista de músicas */}
      {type === "songs" || type === undefined ? (
        <ItemList
          title="Músicas"
          items={20}
          itemsArray={songsArray}
          path="/songs"
          idPath="/song"
        />
      ) : null}
```

Agora que já configuramos o nosso componente Main para ser dinâmico,
vamos também **melhorar o comportamento** do nosso componente ItemList.
A ideia agora é fazer o ItemList **entender em qual página o usuário está** e **adaptar o que ele exibe** dependendo disso.

O problema do código antigo
Antes, o ItemList sempre:

- Exibia um **botão "Mostrar tudo"** (link para outra página), mesmo quando o usuário já estava vendo tudo.
- Limitava a quantidade de itens exibidos, mesmo que o usuário estivesse numa página onde deveria ver a lista completa.

Isso criava alguns problemas de experiência para o usuário e deixava o componente menos flexível.

A solução: usar useLocation para detectar a página atual
O que vamos fazer é:

- Usar o **hook useLocation** do react-router-dom para descobrir qual é a URL atual.
- Se estivermos na página inicial (/), exibimos só uma prévia dos itens e mostramos o botão "Mostrar tudo".
- Se estivermos em outra página (por exemplo, /artists ou /songs), exibimos **todos** os itens e **escondemos o botão**.

Vamos importar o useLocation, um hook que nos dá informações sobre a rota atual.

```
import { Link, useLocation } from "react-router-dom";
```

Pegar o **pathname** (exemplo: /, /artists, /songs) da URL atual.

```
const { pathname } = useLocation();
```

Criamos uma variável isHome que é verdadeira apenas se estivermos na home (/).

```
const isHome = pathname === "/";
```

- Se estivermos na home, mostramos apenas uma quantidade limitada (items).
- Se estivermos em outra página, mostramos todos (Infinity faz o .filter não limitar nada).

```
const finalItems = isHome ? items : Infinity;
```

Ajustando o botão "Mostrar tudo"

```
{isHome ? (
  <Link to={path} className="item-list__link">
    Mostrar tudo
  </Link>
) : (
  <></>
)}
```

- Se estivermos na home, mostramos o link "Mostrar tudo".
- Se estivermos em outra página, **não mostramos nada**.

Benefícios dessa mudança

- Tornamos o ItemList **inteligente**, adaptando o conteúdo de acordo com a página.
- Melhoramos a **experiência do usuário**: agora ele vê uma prévia na home e a lista completa na página dedicada.
- Evitamos links desnecessários ("Mostrar tudo" quando já estamos vendo tudo).
- Deixamos o projeto mais **escalável**: no futuro, podemos adicionar outras páginas e o comportamento continuará correto.

Aula 3 – Deixando o Componente ItemList mais inteligente com useLocation

O problema do código antigo
Antes, o ItemList sempre:

- Exibia um **botão "Mostrar tudo"** (link para outra página), mesmo quando o usuário já estava vendo tudo.
- Limitava a quantidade de itens exibidos, mesmo que o usuário estivesse numa página onde deveria ver a lista completa.

Isso criava alguns problemas de experiência para o usuário e deixava o componente menos flexível.

A solução: usar useLocation para detectar a página atual
O que vamos fazer é:

- Usar o **hook useLocation** do react-router-dom para descobrir qual é a URL atual.
- Se estivermos na página inicial (/), exibimos só uma prévia dos itens e mostramos o botão "Mostrar tudo".
- Se estivermos em outra página (por exemplo, /artists ou /songs), exibimos **todos** os itens e **escondemos o botão**.

Vamos começar refarotando o componente, importando o useLocation, um hook que nos dá informações sobre a rota atual.

```
import { Link, useLocation } from "react-router-dom";
```

Aula 3 – Deixando o Componente ItemList mais inteligente com useLocation

Pegamos o **pathname** (exemplo: /, /artists, /songs) da URL atual.

```
const { pathname } = useLocation();
```

Criamos uma variável **isHome** que é verdadeira apenas se estivermos na home (/).

```
const isHome = pathname === "/";
```

- Se estivermos na home, mostramos apenas uma quantidade limitada (items).
- Se estivermos em outra página, mostramos todos (Infinity faz o .filter não limitar nada).

```
const finalItems = isHome ? items : Infinity;
```

Ajustando o botão "Mostrar tudo"

```
{isHome ? (
  <Link to={path} className="item-list__link">
    Mostrar tudo
  </Link>
) : (
  <></>
)}
```

Aula 3 – Deixando o Componente ItemList mais inteligente com useLocation

Benefícios dessa mudança

- Tornamos o ItemList **inteligente**, adaptando o conteúdo de acordo com a página.
- Melhoramos a **experiência do usuário**: agora ele vê uma prévia na home e a lista completa na página dedicada.
- Evitamos links desnecessários ("Mostrar tudo" quando já estamos vendo tudo).
- Deixamos o projeto mais **escalável**: no futuro, podemos adicionar outras páginas e o comportamento continuará correto.

Código Completo:

```jsx
import React from "react";
import SingleItem from "./SingleItem";
import { Link, useLocation } from "react-router-dom";

const ItemList = ({ title, items, itemsArray, path, idPath }) => {

  const { pathname } = useLocation();
  const isHome = pathname === "/";
  const finalItems = isHome ? items : Infinity;

  return (
    <div className="item-list">
      <div className="item-list__header">
        <h2>{title} populares</h2>

        {isHome ? (
          <Link to={path} className="item-list__link">
            Mostrar tudo
          </Link>
        ) : (
          <></>
        )}
      </div>
      <div className="item-list__container">
        {itemsArray
          .filter((currentValue, index) => index < finalItems)
          .map((currObj, index) => (
            <SingleItem
              // id={currObj.id}
              // name={currObj.name}
              // image={currObj.image}
              // banner={currObj.banner}
              {...currObj}
              idPath={idPath}
              key={`${title}-${index}`}
            />
          ))}
      </div>
    </div>
  );
};

export default ItemList;
```

Aula 3 – Criando a página do Artista com Detalhes

Começamos com um componente extremamente simples:

```
import React from "react";

const Artist = () => {
  return <div>Artist</div>;
};

export default Artist;
```

Esse componente exibia apenas um texto fixo — **sem lógica, sem dados, sem conexão com o restante da aplicação**.
Agora, vamos transformá-lo em uma página dinâmica e funcional que:

- Exibe os dados de um artista específico;
- Mostra a lista de músicas populares desse artista;
- Usa rotas para identificar qual artista deve ser exibido;
- Mostra um botão para tocar uma música aleatória daquele artista.

🧱 Estrutura e importações
No novo código, o componente Artist importa várias dependências:

```
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCirclePlay } from "@fortawesome/free-solid-svg-icons";
import { Link, useParams } from "react-router-dom";
import SongList from "../components/SongList";
import { artistArray } from "../assets/database/artists";
import { songsArray } from "../assets/database/songs";
```

Essas importações trazem:

- Ícones (para exibir o botão de play),
- Funções de roteamento (useParams, Link),
- Componentes reutilizáveis (SongList),
- E os dados dos artistas e músicas.

🔎 Pegando o ID da URL

Com o hook useParams, acessamos o **parâmetro de rota**, ou seja, o id do artista passado pela URL:

```
const { id } = useParams();
```

Se a URL for algo como /artist/3, esse id será "3".

🔍 Buscando os dados do artista

Agora que temos o id, usamos o array de artistas (artistArray) para encontrar o artista correspondente:
Aqui:

```
const { name, banner } = artistArray.filter(
  (currentArtistObj) => currentArtistObj.id === Number(id)
)[0];
```

- filter() percorre a lista de artistas,
- Verificamos se o id do artista bate com o id da URL,
- E pegamos as propriedades name e banner do resultado.

🎶 Buscando as músicas do artista

Agora usamos o nome do artista para filtrar o array de músicas (songsArray), retornando apenas as que pertencem a ele:

```
const songsArrayFromArtist = songsArray.filter(
  (currentSongObj) => currentSongObj.artist === name
);
```

🎲 Selecionando uma música aleatória

Queremos criar um botão que leve para uma música aleatória do artista.
Para isso:

- Calculamos um índice aleatório dentro do array de músicas do artista,
- Pegamos o id dessa música.

```
const randomIndex = Math.floor(
  Math.random() * (songsArrayFromArtist.length - 1)
);
const randomIdFromArtist = songsArrayFromArtist[randomIndex].id;
```

🧱 **Estrutura visual da página**

Agora montamos a parte visual com JSX:

Cabeçalho com o nome e o banner do artista:

```
<div
  className="artist__header"
  style={{
    backgroundImage: `linear-gradient(to bottom, var(--_shade), var(--_shade)),url(${banner})`
  }}
>
  <h2 className="artist__title">{name}</h2>
</div>
```

- A imagem de fundo é o banner do artista,
- E aplicamos um degradê sobre ela para melhorar o contraste do texto.

Corpo com músicas:

```
<div className="artist__body">
  <h2>Populares</h2>
  <SongList songsArray={songsArrayFromArtist} />
</div>
```

- Exibimos o título “Populares”,
- E renderizamos a lista de músicas com o componente SongList, passando as músicas do artista.

Botão de tocar música aleatória:

```
<Link to={`/song/${randomIdFromArtist}`}>
  <FontAwesomeIcon
    className="single-item__icon single-item__icon--artist"
    icon={faCirclePlay}
  />
</Link>
```

Esse link leva para a página de uma música aleatória daquele artista e mostra o ícone de play.

✅ O que aprendemos
Com essa construção, vimos na prática:

- Como acessar parâmetros da rota com useParams;
- Como filtrar arrays de dados com base em condições;
- Como criar uma página personalizada para cada artista usando um ID da URL;
- Como exibir dinamicamente informações visuais (nome, banner e músicas);
- E como adicionar interatividade com um botão que toca uma música aleatória.

Código Completo:

```jsx
import React from "react";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCirclePlay } from "@fortawesome/free-solid-svg-icons";
import { Link, useParams } from "react-router-dom";
import SongList from "../components/SongList";
import { artistArray } from "../assets/database/artists";
import { songsArray } from "../assets/database/songs";

const Artist = () => {
  const { id } = useParams();

  const { name, banner } = artistArray.filter(
    (currentArtistObj) => currentArtistObj.id === Number(id)
  )[0];

  const songsArrayFromArtist = songsArray.filter(
    (currentSongObj) => currentSongObj.artist === name
  );

  const randomIndex = Math.floor(
    Math.random() * (songsArrayFromArtist.length - 1)
  );
  const randomIdFromArtist = songsArrayFromArtist[randomIndex].id;

  return (
    <div className="artist">
      <div
        className="artist__header"
        style={{
          backgroundImage: `linear-gradient(to bottom, var(--_shade), var(--_shade)),url(${banner})`,
        }}
      >
        <h2 className="artist__title">{name}</h2>
      </div>
      <div className="artist__body">
        <h2>Populares</h2>
        <SongList songsArray={songsArrayFromArtist} />
      </div>

      <Link to={`/song/${randomIdFromArtist}`}>
        <FontAwesomeIcon
          className="single-item__icon single-item__icon--artist"
          icon={faCirclePlay}
        />
      </Link>
    </div>
  );
};

export default Artist;
```

Aula 3 – Criando o componente SingleItem passo a passo

✅ **Passo 1: Criando a estrutura inicial do componente**

Vamos começar criando um componente React funcional básico chamado SingleItem. Ele ainda não faz nada, mas já está preparado para ser desenvolvido.

```
import React from "react";

const SingleItem = () => {
  return <div>SingleItem</div>;
};

export default SingleItem;
```

📌 **Explicação**:

- Importamos o React.
- Criamos um componente funcional chamado SingleItem.
- Por enquanto ele só retorna uma com o texto "SingleItem".
- Exportamos esse componente como padrão para poder usá-lo em outros arquivos.

✅ **Passo 2: Tornando o componente reutilizável com props**

Agora vamos permitir que esse componente receba dados dinâmicos por meio de props. Isso nos permite reaproveitá-lo para exibir diferentes artistas ou músicas.

```
const SingleItem = ({ id, name, image, banner, artist, idPath }) => {
  return <div>SingleItem - {name}</div>;
};
```

📌 **Explicação**:

- Estamos esperando receber várias propriedades:
  - id: o identificador único do item.
  - name: o nome do artista ou da música.
  - image: a imagem a ser exibida.
  - banner: (não será usado neste componente, mas pode vir de fora).
  - artist: usado apenas quando o item for uma música.
  - idPath: o caminho base para a navegação.
- Renderizamos o name apenas para testar se está chegando corretamente.

✅ **Passo 3: Envolvendo tudo com um <Link> para navegação**

Queremos que, ao clicar no item, o usuário seja redirecionado para uma nova rota. Para isso, vamos usar o componente <Link> do react-router-dom.

```jsx
import { Link } from "react-router-dom";

const SingleItem = ({ id, name, image, banner, artist, idPath }) => {
  return (
    <Link to={`${idPath}/${id}`} className="single-item">
      <div>{name}</div>
    </Link>
  );
};
```

📌 **Explicação**:

- Importamos o Link do react-router-dom.
- Usamos to={${idPath}/${id}} para construir o caminho dinâmico.
  - Exemplo: se idPath for "artist" e id for 1, o link será artist/1.
- Adicionamos a classe single-item para estilizar depois com CSS.

✅ **Passo 4: Adicionando a imagem com estrutura de layout**

Agora vamos adicionar a imagem principal do item, que pode ser um artista ou uma música. Envolvemos a imagem dentro de uma div para facilitar a estilização.

```
<div className="single-item__div-image-button">
  <div className="single-item__div-image">
    <img
      className="single-item__image"
      src={image}
      alt={`Imagem do Artista ${name}`}
    />
  </div>
</div>
```

📌 **Explicação**:

- A imagem fica dentro de duas divs para termos maior controle de layout.
- A className ajuda na organização do CSS.
- O alt foi pensado para acessibilidade e SEO, usando o name dinamicamente.

✅ Passo 5: Adicionando o ícone de play com Font Awesome

Queremos um ícone de "play" sobre a imagem para indicar que o item é interativo.

```
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCirclePlay } from "@fortawesome/free-solid-svg-icons";

<FontAwesomeIcon className="single-item__icon" icon={faCirclePlay} />
```

📌 **Explicação**:

- Importamos o ícone faCirclePlay da biblioteca Font Awesome.
- Usamos o componente <FontAwesomeIcon /> para exibir esse ícone.
- Ele fica posicionado sobre a imagem para dar a ideia de ação (play).

✅ Passo 6: Adicionando os textos: nome e tipo

Abaixo da imagem, vamos exibir o nome do item e o tipo (nome do artista ou a palavra "Artista", se for uma página de artista).

```
<div className="single-item__texts">
  <div className="single-item__2lines">
    <p className="single-item__title">{name}</p>
  </div>

  <p className="single-item__type">{artist ?? "Artista"}</p>
</div>
```

📌 **Explicação**:

- Mostramos o name com destaque.
- Embaixo, usamos o operador ?? para verificar se a propriedade artist existe:
  - Se existir, exibimos o nome do artista.
  - Se não, exibimos a palavra “Artista”.

Aula 3 – Criando o componente SingleItem passo a passo

Código Completo:

```jsx
import React from "react";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCirclePlay } from "@fortawesome/free-solid-svg-icons";
import { Link } from "react-router-dom";

const SingleItem = ({ id, name, image, banner, artist, idPath }) => {
  return (
    <Link to={`${idPath}/${id}`} className="single-item">
      <div className="single-item__div-image-button">
        <div className="single-item__div-image">
          <img
            className="single-item__image"
            src={image}
            alt={`Imagem do Artista ${name}`}
          />
        </div>
        <FontAwesomeIcon className="single-item__icon" icon={faCirclePlay} />
      </div>

      <div className="single-item__texts">
        <div className="single-item__2lines">
          <p className="single-item__title">{name}</p>
        </div>

        <p className="single-item__type">{artist ?? "Artista"}</p>
      </div>
    </Link>
  );
};

export default SingleItem;
```

Aula 3 – Criando o componente SongItem passo a passo

Criar um componente chamado **SongItem** que:

- Recebe os dados de uma música individual como props.
- Exibe:
  - O número da música na lista.
  - A imagem e o nome da música.
  - A duração da faixa.
- Transforma cada item da lista em um **link clicável**, levando para a página da música (/song/{id}).

🧱 1. Estrutura inicial do componente

Começamos com o básico:

```jsx
import React from "react";

const SongItem = () => {
  return <div>SongItem</div>;
};

export default SongItem;
```

🔗 2. Importando o Link do react-router-dom

Vamos envolver todo o item da música em um <Link> para tornar clicável e permitir a navegação para a página da música:

```
import { Link } from "react-router-dom";
```

📦 3. Recebendo os dados da música via props

Recebemos todos os dados necessários para montar o item:

```
const SongItem = ({ image, name, duration, artist, audio, id, index }) => {
```

- image: imagem do álbum.
- name: nome da música.
- duration: duração da faixa.
- id: usado para montar o link.
- index: posição da música na lista (ex: 0, 1, 2...).

🖼4. Estrutura visual do componente

Transformamos a música em um **item clicável** com aparência de lista musical:

🎨 Explicação das classes CSS

Você pode definir as seguintes classes no seu CSS para organizar o visual:

- song-item: estilo do bloco completo.
- song-item__number-album: agrupa o número e o álbum.
- song-item__album: agrupa a imagem e o nome da música.
- song-item__image: controla o tamanho da imagem.
- song-item__name: define a tipografia do nome da música.

```
return (
  <Link to={`/song/${id}`} className="song-item">
    <div className="song-item__number-album">
      <p>{index + 1}</p> {/* Posição da música na lista */}

      <div className="song-item__album">
        <img
          src={image}
          alt={`Imagem da Música ${name}`}
          className="song-item__image"
        />
        <p className="song-item__name">{name}</p>
      </div>
    </div>

    <p>{duration}</p> {/* Tempo da faixa */}
  </Link>
);
```

Código Completo:

```jsx
import React from "react";
import { Link } from "react-router-dom";

const SongItem = ({ image, name, duration, artist, audio, id, index }) => {
  return (
    <Link to={`/song/${id}`} className="song-item">
      <div className="song-item__number-album">
        <p>{index + 1}</p>

        <div className="song-item__album">
          <img
            src={image}
            alt={`Imagem da Música ${name}`}
            className="song-item__image"
          />

          <p className="song-item__name">{name}</p>
        </div>
      </div>

      <p>{duration}</p>
    </Link>
  );
};

export default SongItem;
```

Aula 3 – Criando a página Song passo a passo

Este componente será responsável por exibir a **página de uma música específica**, com:

- Imagem da música
- Nome do artista
- Um player para tocar o som
- Link para o perfil do artista
- Sugestões de outras músicas do mesmo artista

✅ Passo 1: Estrutura base do componente

Vamos começar com o básico: o nosso componente React está com uma de teste.

```jsx
import React from "react";

const Song = () => {
  return <div>Song</div>;
};

export default Song;
```

📌 **Explicação**:

- Importamos o React.
- Criamos o componente Song.
- Exportamos ele para ser usado em uma rota específica.

✅ **Passo 2: Importando os dados e o useParams para pegar o ID da URL**

Para saber qual música será exibida, precisamos capturar o ID vindo da URL com o useParams. Também importamos o banco de dados de músicas e artistas.

```jsx
import React from "react";
import { useParams } from "react-router-dom";
import { songsArray } from "../assets/database/songs";
import { artistArray } from "../assets/database/artists";

const Song = () => {
  const { id } = useParams(); // pega o ID da URL

  return <div>Song ID: {id}</div>;
};

export default Song;
```

📌 **Explicação**:

- useParams() nos dá acesso ao ID passado pela rota (ex: /song/3 → id = 3).
- Já conectamos aos arrays songsArray e artistArray, que são nossos "bancos de dados" locais.

✅ **Passo 3: Buscando a música correta usando o ID**
Agora vamos buscar no songsArray a música que tem esse ID.

```
const songObj = songsArray.filter(
  (currentSongObj) => currentSongObj.id === Number(id)
)[0];

const { image, name, duration, artist, audio } = songObj;
```

📌 **Explicação**:
- Usamos filter() para encontrar a música com o id que veio da URL.
- Usamos [0] no final para pegar o **primeiro item do resultado**.
- Usamos **desestruturação** para extrair os dados da música: image, name, duration, etc.

✅ **Passo 4: Buscando os dados do artista da música**

Agora vamos encontrar o objeto do artista que cantou essa música.

```
const artistObj = artistArray.filter(
  (currentArtistObj) => currentArtistObj.name === artist
)[0];
```

📌 **Explicação**:

- Usamos o nome do artista da música (artist) para encontrar o objeto completo do artista no artistArray.

✅ **Passo 5: Buscando outras músicas do mesmo artista**

Vamos filtrar o array de músicas para encontrar **todas as músicas desse mesmo artista**, e selecionar **duas aleatórias** para recomendação.

📌 **Explicação**:

- filter() retorna todas as músicas do artista.
- Math.floor(Math.random() * length) gera índices aleatórios.
- Pegamos os IDs dessas músicas para enviar ao componente Player.

```
const songsArrayFromArtist = songsArray.filter(
  (currentSongObj) => currentSongObj.artist === artist
);

const randomIndex = Math.floor(Math.random() * (songsArrayFromArtist.length - 1));
const randomIndex2 = Math.floor(Math.random() * (songsArrayFromArtist.length - 1));

const randomIdFromArtist = songsArrayFromArtist[randomIndex].id;
const randomId2FromArtist = songsArrayFromArtist[randomIndex2].id;
```

✅ **Passo 6: Estrutura de layout com imagem da música e player**
Agora montamos o layout principal da página da música.

📌 **Explicação**:
- A imagem da música aparece em destaque.
- Usamos o componente <Player /> (que já deve estar pronto em outro arquivo).
- Exibimos o nome da música e do artista.
- A imagem do artista funciona como link para a página dele (/artist/{id}).

```
import Player from "../components/Player";
import { Link } from "react-router-dom";

return (
  <div className="song">
    <div className="song__container">
      <div className="song__image-container">
        <img src={image} alt={`Imagem da música ${name}`} />
      </div>
    </div>

    <div className="song__bar">
      <Link to={`/artist/${artistObj.id}`} className="song__artist-image">
        <img
          width={75}
          height={75}
          src={artistObj.image}
          alt={`Imagem do Artista ${artist}`}
        />
      </Link>
```

```
        <Player
          duration={duration}
          randomIdFromArtist={randomIdFromArtist}
          randomId2FromArtist={randomId2FromArtist}
        />

        <div>
          <p className="song__name">{name}</p>
          <p>{artist}</p>
        </div>
      </div>
    </div>
  );
```

Aula 3 – Criando a página Song passo a passo

Código Completo:

```jsx
import React from "react";
import Player from "../components/Player";
import { Link, useParams } from "react-router-dom";
import { songsArray } from "../assets/database/songs";
import { artistArray } from "../assets/database/artists";

const Song = () => {
  const { id } = useParams();

  const { image, name, duration, artist, audio } = songsArray.filter(
    (currentSongObj) => currentSongObj.id === Number(id)
  )[0];

  const artistObj = artistArray.filter(
    (currentArtistObj) => currentArtistObj.name === artist
  )[0];

  const songsArrayFromArtist = songsArray.filter(
    (currentSongObj) => currentSongObj.artist === artist
  );

  const randomIndex = Math.floor(
    Math.random() * (songsArrayFromArtist.length - 1)
  );

  const randomIndex2 = Math.floor(
    Math.random() * (songsArrayFromArtist.length - 1)
  );
```

```jsx
  const randomIdFromArtist = songsArrayFromArtist[randomIndex].id;
  const randomId2FromArtist = songsArrayFromArtist[randomIndex2].id;

  return (
    <div className="song">
      <div className="song__container">
        <div className="song__image-container">
          <img src={image} alt={`Imagem da música ${name}`} />
        </div>
      </div>

      <div className="song__bar">
        <Link to={`/artist/${artistObj.id}`} className="song__artist-image">
          <img
            width={75}
            height={75}
            src={artistObj.image}
            alt={`Imagem do Artista ${artist}`}
          />
        </Link>

        <Player
          duration={duration}
          randomIdFromArtist={randomIdFromArtist}
          randomId2FromArtist={randomId2FromArtist}
        />

        <div>
          <p className="song__name">{name}</p>
          <p>{artist}</p>
        </div>
      </div>
    </div>
  );
};

export default Song;
```

Aula 3 – Criando o componente Player passo a passo

📁 1. Estrutura inicial do componente

Vamos começar com a estrutura básica de um componente funcional em React:

```
import React from "react";

const Player = () => {
  return <div>Player</div>;
};

export default Player;
```

🔧 2. Importações necessárias

Para adicionar os ícones do player, precisamos importar os ícones do FontAwesome e também o Link do react-router-dom, pois vamos usar navegação entre músicas:

```
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import {
  faCirclePlay,
  faBackwardStep,
  faForwardStep,
} from "@fortawesome/free-solid-svg-icons";
import { Link } from "react-router-dom";
```

🪝 3. Recebendo as props

Esse componente vai receber três informações como **props**:

- duration: duração da música atual.
- randomIdFromArtist: ID de uma música aleatória do mesmo artista (para o botão "anterior").
- randomId2FromArtist: outro ID aleatório do mesmo artista (para o botão "próxima").

Vamos declarar essas props na função:

```
const Player = ({ duration, randomIdFromArtist, randomId2FromArtist }) => {
```

🧱 4. Estrutura HTML do componente

Agora vamos montar a estrutura JSX. O componente será dividido em duas partes principais:

A. Controles do player

- O botão de **voltar** é um Link que leva à música anterior.
- O **play** é apenas visual (por enquanto).
- O botão de **próxima música** também é um Link.

```
<div className="player__controllers">
  <Link to={`/song/${randomIdFromArtist}`}>
    <FontAwesomeIcon className="player__icon" icon={faBackwardStep} />
  </Link>

  <FontAwesomeIcon
    className="player__icon player__icon--play"
    icon={faCirclePlay}
  />

  <Link to={`/song/${randomId2FromArtist}`}>
    <FontAwesomeIcon className="player__icon" icon={faForwardStep} />
  </Link>
</div>
```

B. Barra de progresso
Abaixo dos botões, adicionamos a barra de progresso:

- 00:00 representa o início da música.
- {duration} exibe o tempo total (recebido por props).
- A barra de progresso é apenas visual (pode ser funcional depois).

```
<div className="player__progress">
  <p>00:00</p>

  <div className="player__bar">
    <div className="player__bar-progress"></div>
  </div>

  <p>{duration}</p>
</div>
```

Aula 3 – Criando o componente Player passo a passo

Código Completo:

```jsx
import React from "react";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import {
  faCirclePlay,
  faBackwardStep,
  faForwardStep,
} from "@fortawesome/free-solid-svg-icons";
import { Link } from "react-router-dom";

const Player = ({ duration, randomIdFromArtist, randomId2FromArtist }) => {
  return (
    <div className="player">
      <div className="player__controllers">
        <Link to={`/song/${randomIdFromArtist}`}>
          <FontAwesomeIcon className="player__icon" icon={faBackwardStep} />
        </Link>

        <FontAwesomeIcon
          className="player__icon player__icon--play"
          icon={faCirclePlay}
        />
        <Link to={`/song/${randomId2FromArtist}`}>
          <FontAwesomeIcon className="player__icon" icon={faForwardStep} />
        </Link>
      </div>

      <div className="player__progress">
        <p>00:00</p>

        <div className="player__bar">
          <div className="player__bar-progress"></div>
        </div>

        <p>{duration}</p>
      </div>
    </div>
  );
};

export default Player;
```

Aula 3 – Criando o componente SongList passo a passo

Criar um componente chamado SongList que:

- Exibe uma lista inicial de músicas (5 por padrão).
- Ao clicar em "Ver mais", adiciona mais 5 músicas à lista.
- Usa o componente SongItem para renderizar cada música individualmente.

🧱 1. Estrutura inicial do componente

Começamos com a base de um componente funcional que recebe uma **lista de músicas** como props:

```jsx
import React from "react";

const SongList = ({ songsArray }) => {
  return <div>SongList</div>;
};

export default SongList;
```

📦 **2. Importando dependências**

- Vamos usar o useState para controlar quantas músicas estão sendo exibidas.
- Também importamos o componente SongItem, responsável por mostrar cada música.

```
import { useState } from "react";
import SongItem from "./SongItem";
```

⚙**3. Estado inicial – número de músicas exibidas**

Usamos useState para definir quantos itens (músicas) serão mostrados inicialmente:

Isso significa que, ao abrir a página, o componente mostra as **5 primeiras músicas** da lista.

```
const [items, setItems] = useState(5);
```

✍4. Renderização das músicas com .filter() e .map()

Agora, vamos percorrer o array songsArray, **filtrar os primeiros items elementos** e renderizar cada um usando o SongItem.

```
{songsArray
  .filter((currentValue, index) => index < items)
  .map((currentSongObj, index) => (
    <SongItem {...currentSongObj} index={index} key={index} />
  ))}
```

Explicação:

- .filter(...): filtra a lista para exibir apenas os primeiros items elementos.
- .map(...): percorre os itens filtrados e retorna o componente SongItem para cada música.
- key={index}: define uma chave única para cada item renderizado (usamos index por simplicidade).
- {...currentSongObj}: envia todas as informações da música como props.

➕ 5. Botão “Ver mais”

Abaixo da lista de músicas, criamos um parágrafo com o texto "Ver mais". Ele funciona como botão para **mostrar mais 5 músicas** ao clicar:

```
<p
  className="song-list__see-more"
  onClick={() => {
    setItems(items + 5);
  }}
>
  Ver mais
</p>
```

- Ao clicar, a função setItems(items + 5) atualiza o estado.
- Isso **re-renderiza** o componente com mais músicas na tela.
- Se houver menos de 5 músicas restantes, ele renderiza apenas o que falta.

Código Completo:

```jsx
import React from "react";
import SongItem from "./SongItem";
import { useState } from "react";

const SongList = ({ songsArray }) => {
  const [items, setItems] = useState(5);

  return (
    <div className="song-list">
      {songsArray
        .filter((currentValue, index) => index < items)
        .map((currentSongObj, index) => (
          <SongItem {...currentSongObj} index={index} key={index} />
        ))}

      <p
        className="song-list__see-more"
        onClick={() => {
          setItems(items + 5);
        }}
      >
        Ver mais
      </p>
    </div>
  );
};

export default SongList;
```