

JORNADA FULL STACK IMPRESSIONADOR



JORNADA-FULL STACK IMPRESSIONADOR | HASHTAG PROGRAMAÇÃO



Aula 1

Aula 1

Aula 1

Bem-vindo à **Jornada Full Stack da Hashtag!** 🚀

Ao longo das aulas, vamos mergulhar no universo do desenvolvimento web e aprender as principais tecnologias utilizadas no mercado. Você terá contato com ferramentas essenciais como **React, HTML, CSS, MongoDB, Express.js, JavaScript, Vite, Git e GitHub, Node.js**, e muito mais!

E o melhor: colocaremos tudo isso em prática construindo **uma réplica do Spotify!** 🎵💻



Essa jornada é para quem quer desenvolver habilidades Full Stack, dominando tanto o **Front-end** quanto o **Back-end**, criando aplicações modernas e escaláveis.

Prepare-se para muito aprendizado e prática. Vamos juntos transformar código em realidade! 🎉🔥

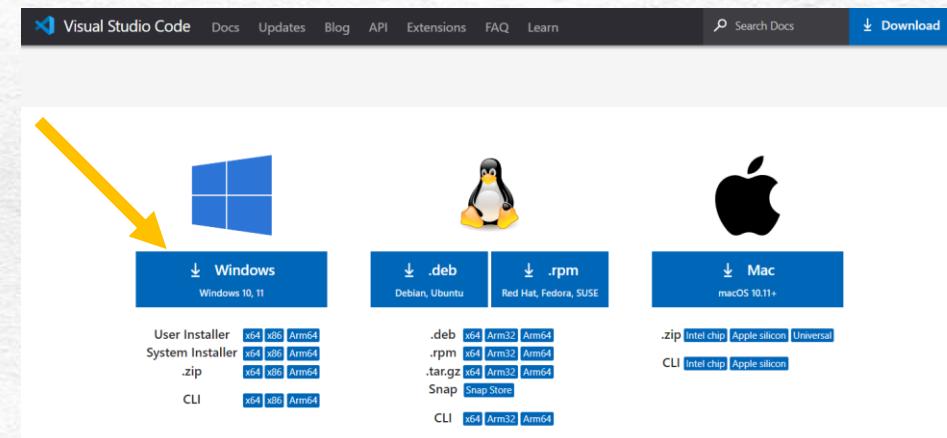
Aula 1 – Baixando e Configurando nosso editor de códigos (1 / 10)

Visual Studio Code

O Visual Studio Code (VS Code) é um editor de código amplamente utilizado para desenvolver sites e aplicativos web usando HTML e CSS. Ele oferece uma interface amigável e funcionalidades que facilitam a escrita e edição de código, como realce de sintaxe, sugestões de código e integração com extensões que agilizam o desenvolvimento web. O VS Code é altamente personalizável e é uma escolha popular entre desenvolvedores front-end devido à sua eficiência e suporte à comunidade.

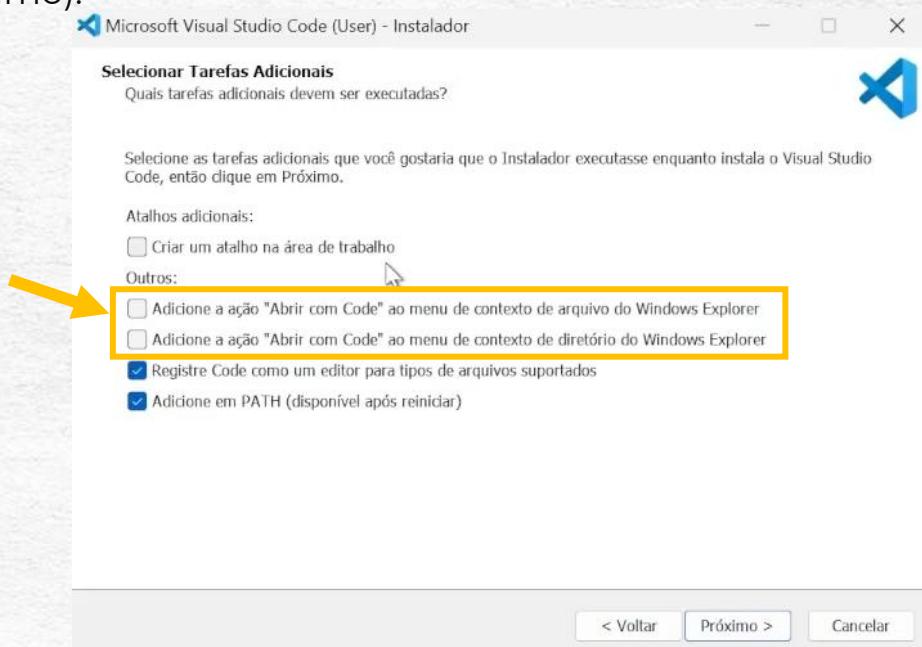
Baixando o VS Code

- 1. Acesse o Site Oficial:** Abra seu navegador da web e vá para o site oficial do Visual Studio Code em <https://code.visualstudio.com/>.
- 2. Download:** Clique no botão "Download" de acordo com o sistema operacional do seu computador na página inicial do site. No nosso caso estaremos instalando para Windows.



- 3. Baixe o Arquivo de Instalação:** O navegador irá baixar um arquivo de instalação com extensão ".exe". Aguarde até que o download seja concluído.
- 4. Inicie o Instalador:** Após o download, clique duas vezes no arquivo ".exe" baixado para iniciar o instalador do VS Code.
- 5. Aceite os Termos de Uso:** Você verá uma tela inicial do instalador. Clique em "Next" (Próximo) para continuar.
- 6. Selecione as Opções de Instalação:** Escolha o local onde deseja instalar o VS Code e selecione as opções adicionais, como adicionar ícones à área de trabalho e associar arquivos ".code" ao VS Code. Em seguida, clique em "Next" (Próximo).
- 7. Selecione Componentes Adicionais (Opcional):** O instalador pode oferecer a opção de instalar componentes adicionais, como o Git. Se desejar, selecione as opções desejadas e clique em "Next" (Próximo).

Ao chegar na tela ao lado, você poderá adicionar a opção de adicionar a ação "Abrir com Code" que permite que você abra pastas ou arquivos diretamente no Visual Studio Code (VS Code) a partir do menu de contexto (clique com o botão direito do mouse) do Windows Explorer ou do gerenciador de arquivos do seu sistema operacional. Isso é particularmente útil quando você está trabalhando em projetos e deseja abrir pastas ou arquivos específicos diretamente do explorador de arquivos do seu sistema operacional, economizando tempo e simplificando o processo de desenvolvimento.

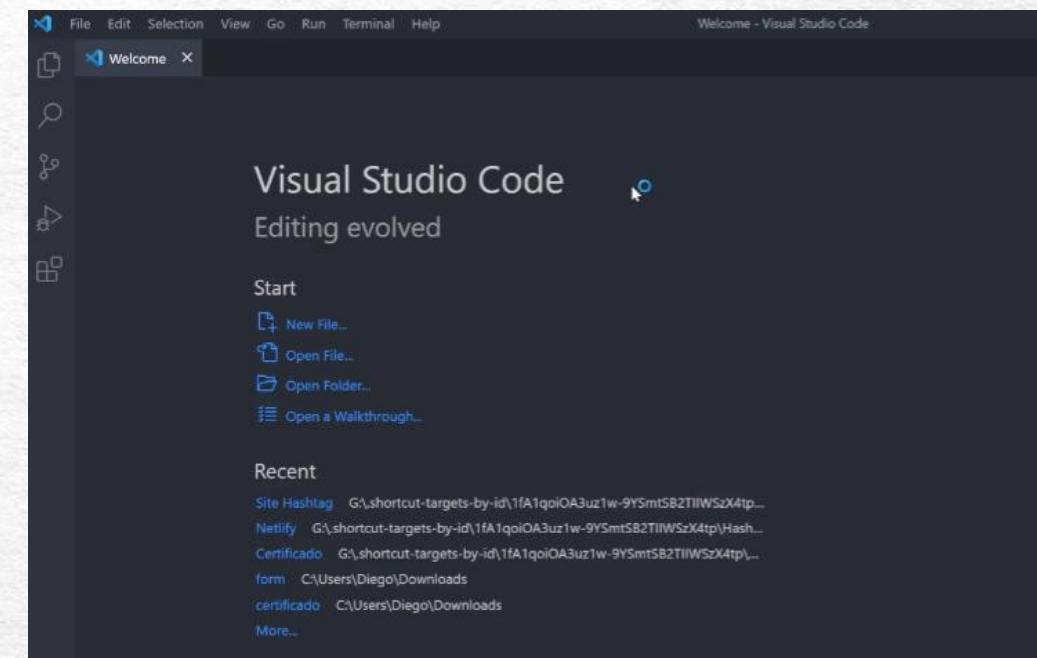
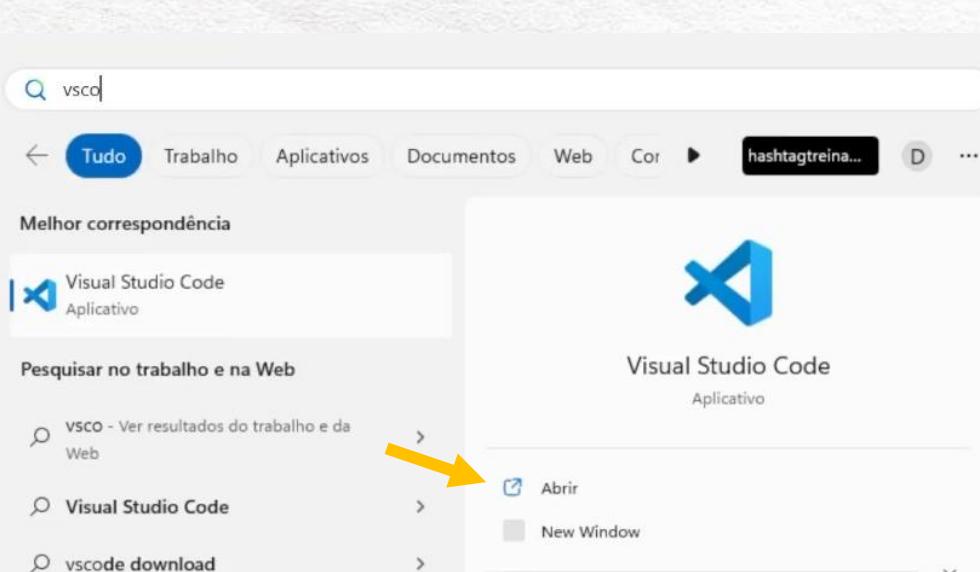


8. Inicie a Instalação: Clique em "Install" (Instalar) para iniciar o processo de instalação.

9. Conclua a Instalação: Após a conclusão da instalação, clique em "Finish" (Concluir) para fechar o instalador.

Agora você tem o Visual Studio Code instalado em seu computador Windows e pode começar a usá-lo para desenvolver projetos HTML e CSS.

Agora você irá abrir o menu iniciar do seu computador, pesquisar por VS Code e inicializar o programa.

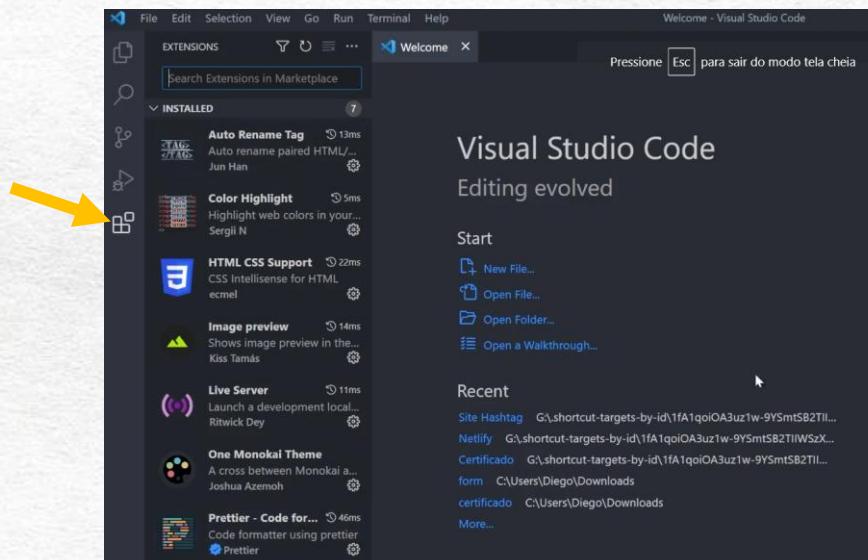


As extensões no Visual Studio Code (VS Code) são módulos de software adicionais que você pode instalar para estender a funcionalidade padrão do editor.

Para instalar extensões no VS Code, você pode seguir estes passos:

1. Abra o VS Code.
2. Vá para a seção "Extensões" no menu lateral esquerdo (ícone de quebra-cabeça).
3. Pesquise a extensão desejada na barra de pesquisa.
4. Clique em "Instalar" na extensão que deseja adicionar.
5. Depois de instalada, você pode configurar e usar a extensão conforme necessário.

As extensões tornam o VS Code altamente flexível e adaptável às suas necessidades de desenvolvimento, permitindo que você crie um ambiente de desenvolvimento personalizado e eficiente.



As extensões no Visual Studio Code (VS Code) são módulos de software adicionais que você pode instalar para estender a funcionalidade padrão do editor.

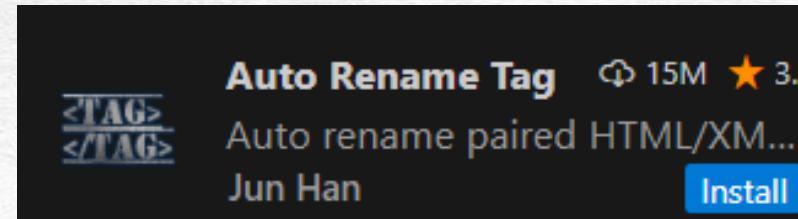
Para instalar extensões no VS Code, você pode seguir estes passos:

1. Abra o VS Code.
2. Vá para a seção "Extensões" no menu lateral esquerdo (ícone de quebra-cabeça).
3. Pesquise a extensão desejada na barra de pesquisa.
4. Clique em "Instalar" na extensão que deseja adicionar.
5. Depois de instalada, você pode configurar e usar a extensão conforme necessário.

As extensões tornam o VS Code altamente flexível e adaptável às suas necessidades de desenvolvimento, permitindo que você crie um ambiente de desenvolvimento personalizado e eficiente.

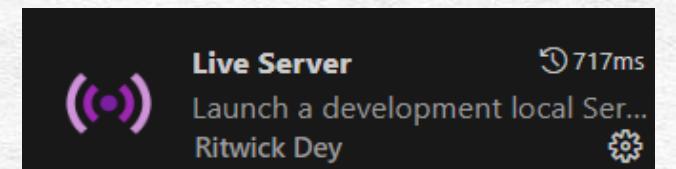
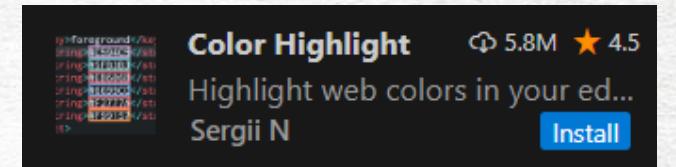
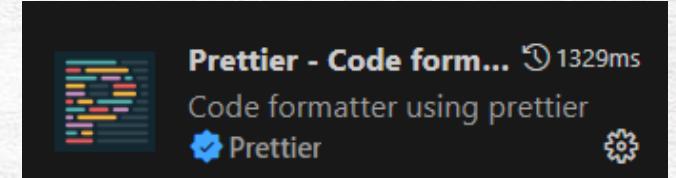
Extensões

- **Auto Rename Tag**: Esta extensão é útil ao trabalhar com HTML/XML. Ela permite que você renomeie uma tag de abertura e a tag de fechamento correspondente automaticamente, economizando tempo e evitando erros de digitação. Exemplo: Ao renomear a tag `<div>` para `<section>`, a extensão também renomeará automaticamente a tag de fechamento `</div>` para `</section>`.

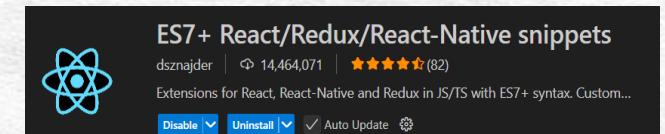
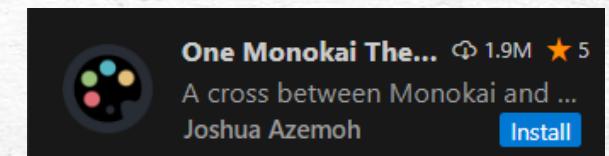
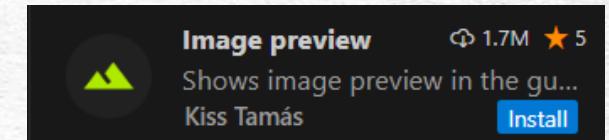
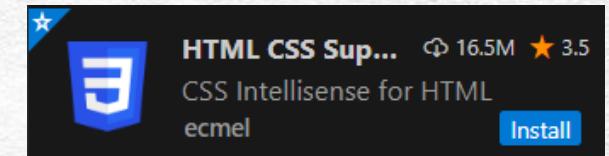


Módulo 1 – Baixando e Configurando nosso editor de códigos (6 / 10)

- **Prettier:** O Prettier é uma extensão de formatação de código que ajuda a manter seu código HTML, CSS e JavaScript bem formatado e consistente. Ele reformata automaticamente seu código de acordo com as convenções de estilo configuradas, facilitando a legibilidade e a colaboração em equipes.
- **Color Highlight:** Com esta extensão, os códigos de cores (por exemplo, #FF0000 para vermelho) no seu CSS ou HTML são realçados com a cor correspondente, permitindo uma rápida visualização das cores usadas em seu código.
- **Live Server:** O Live Server é uma extensão que cria um servidor web local diretamente do VS Code. Isso permite que você visualize e teste seu site ou aplicativo web em tempo real enquanto faz alterações no código, facilitando o desenvolvimento e a depuração. Basta clicar com o botão direito do mouse em um arquivo HTML e selecionar "Open with Live Server" para abrir o site no navegador. As alterações no código HTML são refletidas automaticamente no navegador sem a necessidade de atualizações manuais.



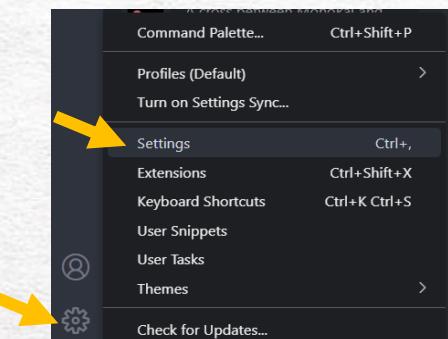
- **HTML CSS SUPPORT:** Esta extensão aprimora o suporte do VS Code para HTML e CSS, fornecendo sugestões de código, realce de sintaxe e atalhos para facilitar a escrita e a edição de código HTML e CSS. Ao digitar um seletor CSS em uma regra de estilo dentro de uma tag HTML, a extensão oferece sugestões automáticas de classes e IDs existentes.
- **Image Preview:** Com essa extensão, você pode visualizar imagens diretamente no Visual Studio Code sem a necessidade de abrir um visualizador de imagens separado. Basta clicar com o botão direito do mouse em um arquivo de imagem e selecionar "Image Preview" para ver a imagem.
- **One Monokai Theme:** Esta é uma extensão de tema que altera a aparência do VS Code. O "One Monokai Theme" aplica um esquema de cores específico ao VS Code, proporcionando uma experiência de desenvolvimento visualmente agradável.
- **ES7+ React/Redux/React-Native snippets:** Esta extensão adiciona atalhos e snippets para facilitar a escrita de código em **React, React Native e Redux** no Visual Studio Code. Com ela, você pode gerar rapidamente componentes, hooks e outras estruturas comuns do React usando comandos curtos, agilizando o desenvolvimento e melhorando a produtividade.



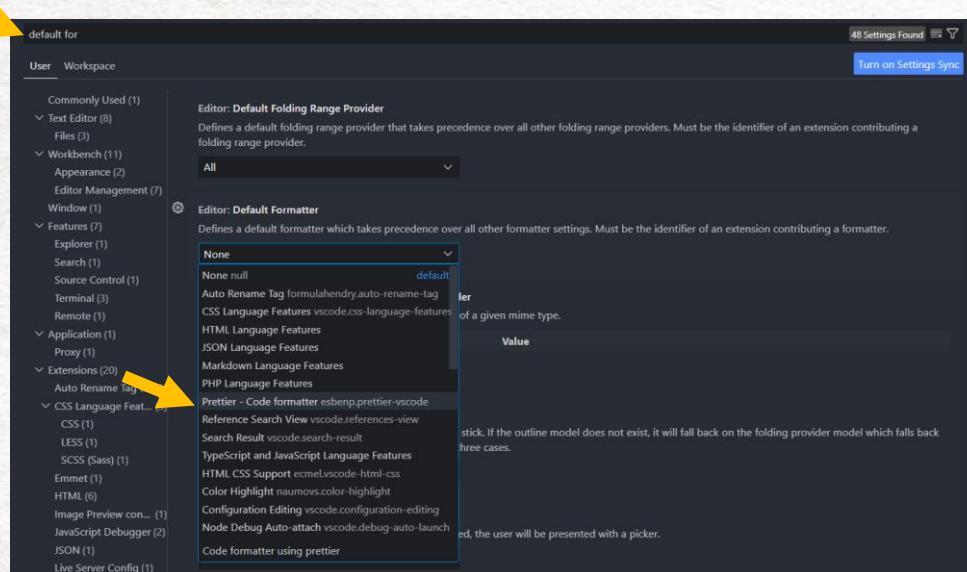
Módulo 1 – Baixando e Configurando nosso editor de códigos (8 / 10)

11

Agora vamos precisar modificar algumas configurações das extensões e para isso clicaremos no ícone de engrenagem:



Vamos perquisar por **Default for** e adicionar o Prettier como formatação padrão do nosso VS Code.

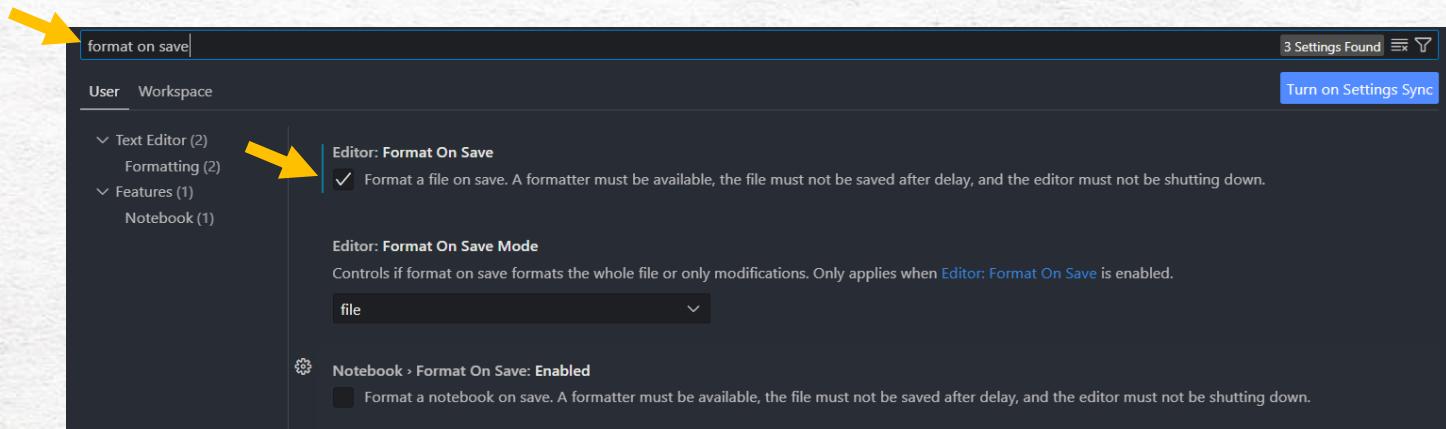


A configuração padrão do Prettier envolve as opções de formatação que são aplicadas automaticamente quando você executa o Prettier em seu código sem personalizar essas opções. As configurações padrão incluem coisas como a quantidade de indentação, estilo de quebra de linha e estilo de citação.



JORNADA FULL STACK IMPRESSIONADOR I HASHTAG PROGRAMAÇÃO

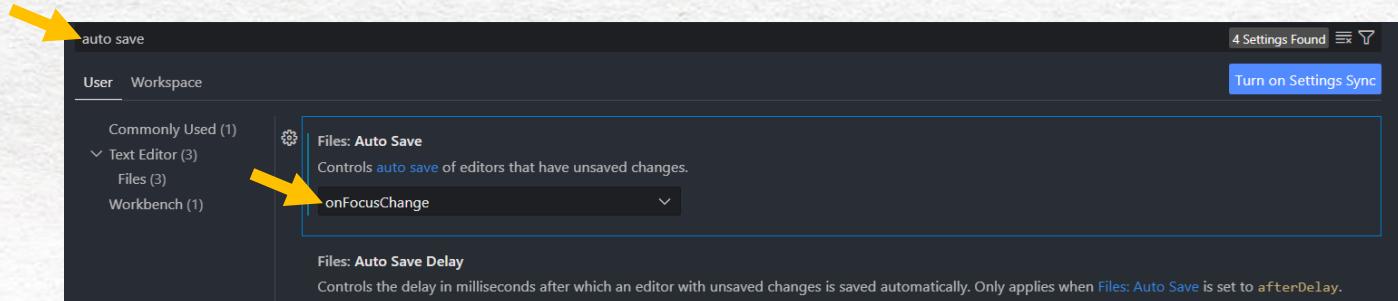
A próxima configuração é a **format on save**, Essa opção permite que o editor formate automaticamente o código no arquivo atual sempre que você o salva:



Auto on save:

Nas configurações padrão do VS Code, a ação "Salvar" ocorre automaticamente quando você:

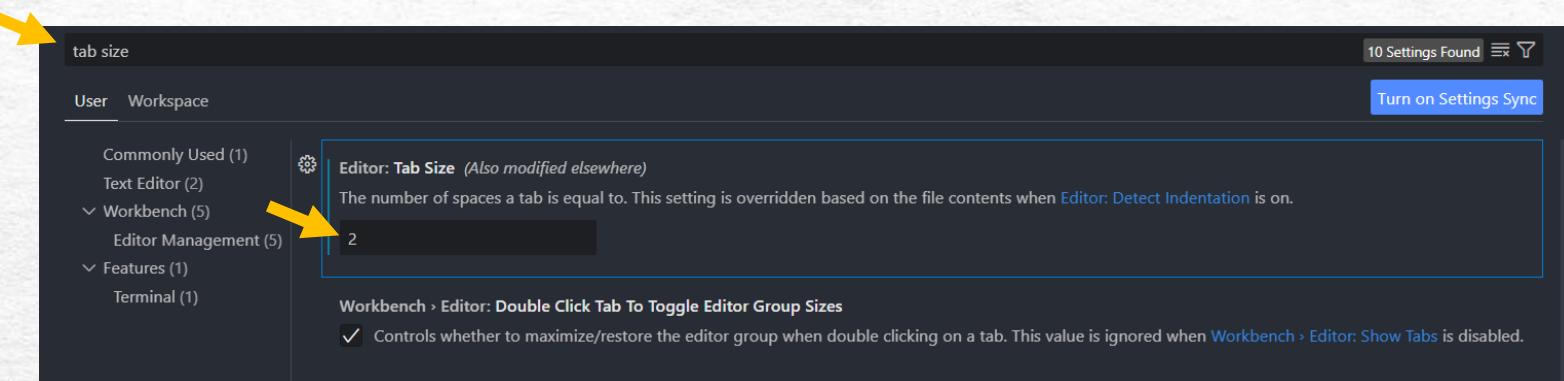
- Altera o foco para outra janela ou aplicativo, ou seja, quando você muda para outra janela ou programa fora do VS Code.
- Fecha o arquivo ou o editor no VS Code.



Tab size (tamanho de tabulação) refere-se ao número de espaços em branco que uma tecla "Tab" representa em um editor de código. Isso afeta a quantidade de espaço que é inserida quando você pressiona a tecla "Tab" ou quando o editor de código converte uma tabulação em espaços em branco.

A configuração do "tab size" é importante porque afeta a formatação e a legibilidade do seu código, bem como a consistência entre diferentes desenvolvedores que trabalham no mesmo projeto. A maioria dos editores de código modernos, incluindo o Visual Studio Code (VS Code), permite que você configure o tamanho da tabulação de acordo com suas preferências.

Por padrão, o tamanho de tabulação no VS Code é geralmente definido como 4 espaços, mas você pode personalizá-lo



NODE.JS

O Node.js é um ambiente de execução de código JavaScript do lado do servidor. Ele permite que você execute código JavaScript fora do navegador, o que significa que você pode criar aplicativos de servidor, scripts de linha de comando e muito mais usando JavaScript.

Para instalar o Node.js, você pode seguir os seguintes passos:

- Acesse o site oficial do Node.js em <https://nodejs.org>.
- Na página inicial, você verá duas versões para download: LTS (Long Term Support) e Current. A versão LTS é recomendada para a maioria dos usuários, pois é mais estável e possui suporte a longo prazo. Selecione a versão LTS ou a versão mais recente, se preferir.
- Após selecionar a versão desejada, você será redirecionado para a página de download. Escolha o instalador adequado para o seu sistema operacional (Windows, macOS ou Linux) e clique no link para iniciar o download.
- Após o download ser concluído, execute o instalador e siga as instruções na tela para concluir a instalação.
- Após a instalação ser concluída, você pode verificar se o Node.js foi instalado corretamente abrindo o terminal ou prompt de comando e digitando o comando node -v. Se a versão do Node.js for exibida, significa que a instalação foi bem-sucedida.



O que é o DevTools?

O **DevTools** (Developer Tools) é um conjunto de ferramentas embutido nos navegadores modernos, como **Google Chrome, Firefox e Edge**, que permite inspecionar, depurar e otimizar páginas da web em tempo real.

Com ele, você pode:

- Visualizar e editar o HTML e CSS** da página.
- Monitorar o console JavaScript** para encontrar erros e testar códigos.
- Analizar requisições de rede** para ver como os arquivos são carregados.
- Depurar código JavaScript** e entender o fluxo da aplicação.
- Testar responsividade** para diferentes tamanhos de tela.

Acessando o DevTools

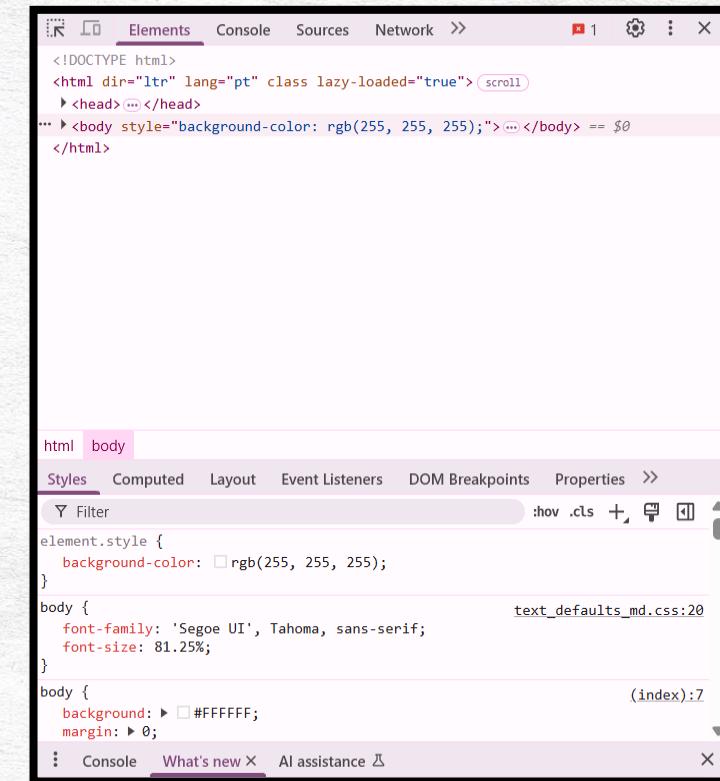
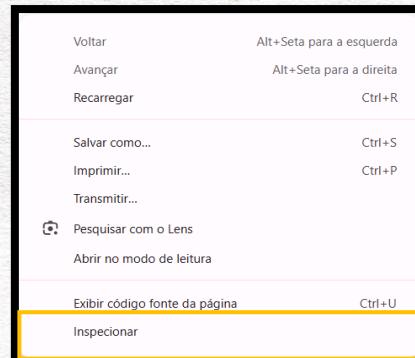
Existem três formas principais de abrir o DevTools:

- ◆ **Atalho no teclado:**

- **Windows/Linux:** F12 ou Ctrl + Shift + I
- **Mac:** Cmd + Option + I

- ◆ **Menu do navegador:**

Clique com o botão direito na página e selecione "Inspecionar".



Principais Guias do DevTools

1 Elements (Elementos)

- Permite visualizar e editar o código **HTML** e **CSS** em tempo real.
- Você pode modificar a estrutura da página e ver as alterações instantaneamente.

2 Console

- Exibe mensagens, erros e logs do JavaScript.
- Aqui, você pode testar códigos diretamente no navegador.

3 Network (Rede)

- Mostra todas as requisições feitas pela página (imagens, arquivos CSS, JavaScript, APIs).
- Ajuda a identificar **arquivos que demoram para carregar** ou **erros de requisição**.

4 Sources (Fontes)

- Usado para depuração de código JavaScript.
- Permite **adicionar breakpoints** para interromper a execução do código e analisá-lo passo a passo.

5 Application (Aplicação)

- Mostra informações sobre **armazenamento local**, **cookies**, **cache**, e **bancos de dados** da página.

6 Lighthouse

- Ferramenta para analisar a performance e acessibilidade da página.
- Ajuda a melhorar o SEO e o desempenho do site.

O que é o Terminal do VS Code?

O **terminal integrado** do Visual Studio Code permite rodar comandos diretamente dentro do editor, sem precisar abrir outra janela. Ele é útil para rodar scripts, gerenciar dependências e executar aplicações.

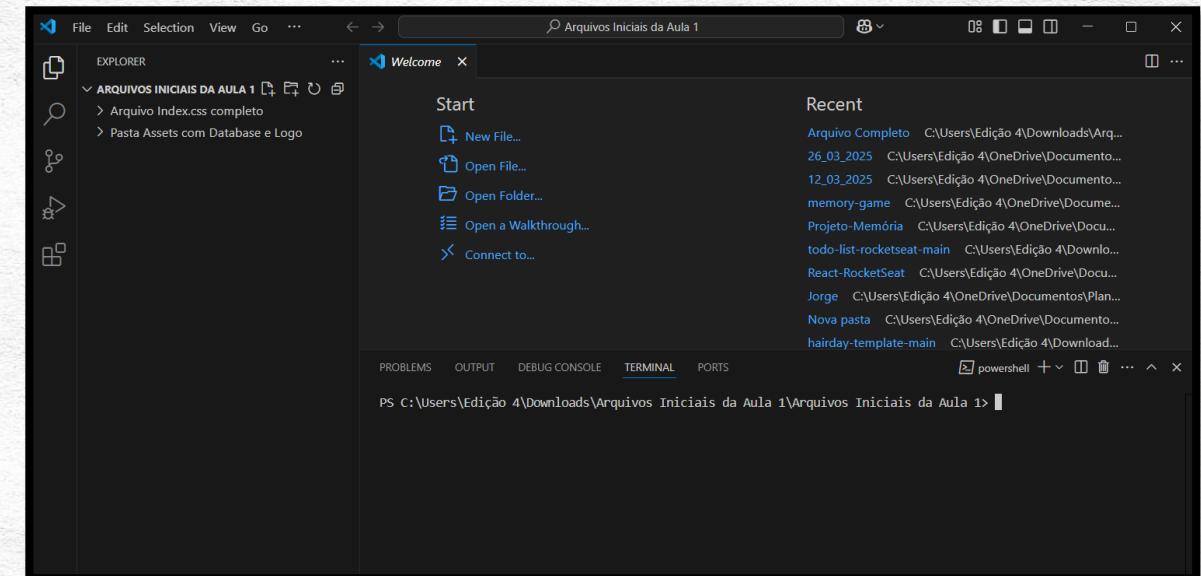
Como abrir o terminal?

Existem três formas principais:

- **Atalho de teclado:**
 - Windows/Linux: Ctrl + J
 - Mac: Cmd + J
- Pelo menu: Vá em **Exibir > Terminal**.
- Clicando no botão "**Terminal**" na parte inferior do VS Code.

Comandos básicos no terminal

- cd nome-da-pasta → Acessa uma pasta específica.
- ls (Linux/Mac) ou dir (Windows) → Lista os arquivos da pasta.
- clear → Limpa o terminal.



O que é um Pacote no Node.js?

Pacotes são conjuntos de códigos prontos que podemos instalar para adicionar funcionalidades ao nosso projeto.

O que é o NPM?

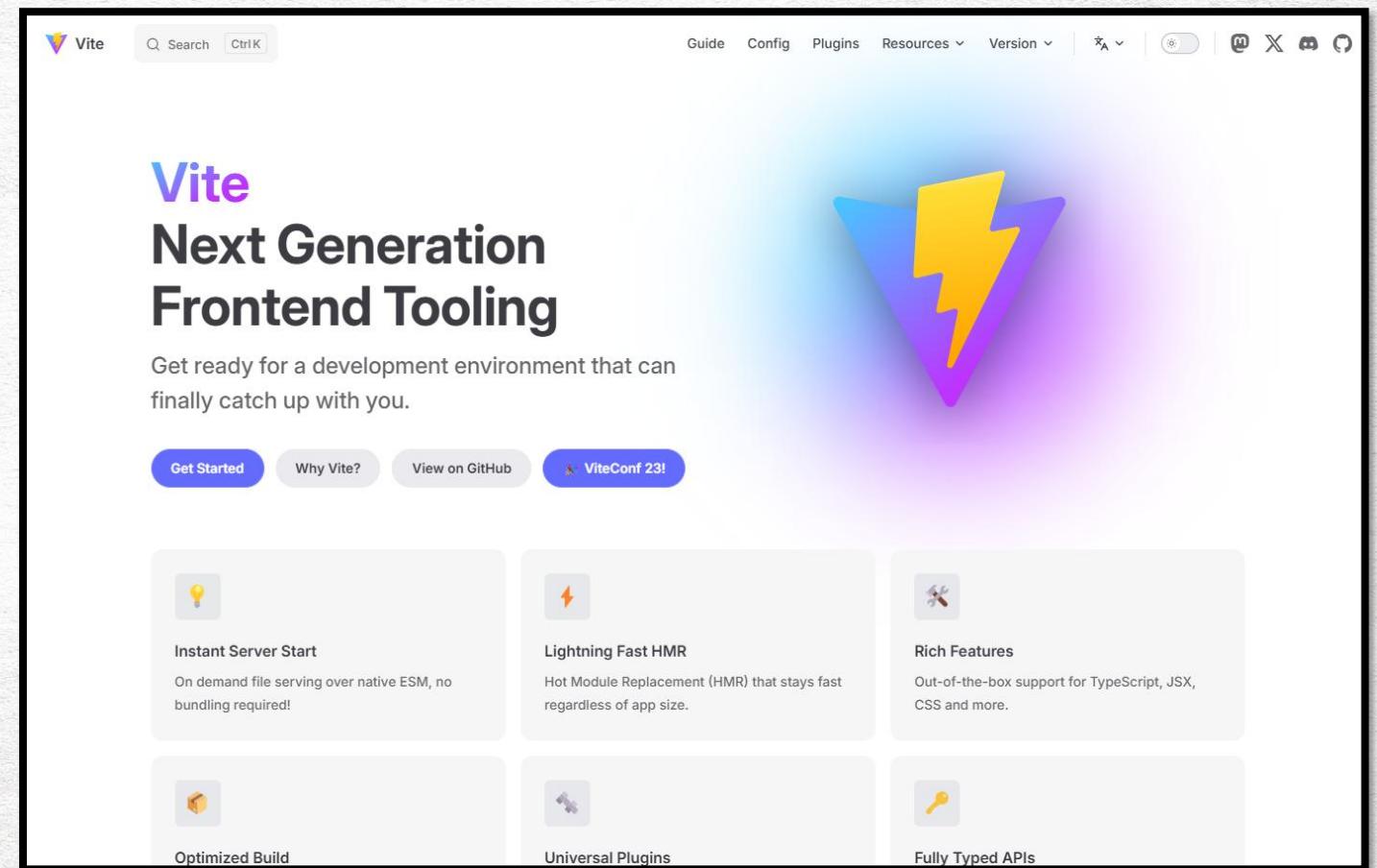
O **NPM (Node Package Manager)** é o gerenciador de pacotes do Node.js. Com ele, podemos instalar, atualizar e remover pacotes de forma rápida.

Vite.js é um ambiente de desenvolvimento rápido para aplicações web modernas. Ele é construído em torno do conceito de "blocos de construção" para construir aplicações web de forma eficiente e escalável. O principal objetivo do Vite.js é oferecer um tempo de inicialização instantâneo durante o desenvolvimento, utilizando a funcionalidade de módulos do JavaScript e a ferramenta de bundling (agrupamento de arquivos) apenas para produção.

O Vite.js é conhecido por sua velocidade, graças ao uso de tecnologias como ES Modules nativos do navegador e um servidor de desenvolvimento otimizado. Ele suporta várias estruturas e bibliotecas populares, como Vue.js, React e Preact.

Com o Vite.js, os desenvolvedores podem criar aplicações web modernas de forma eficiente, aproveitando ao máximo as capacidades das últimas tecnologias da web, enquanto mantêm um fluxo de desenvolvimento ágil e responsivo.

[Vite | Next Generation Frontend Tooling \(vitejs.dev\)](https://vitejs.dev)



Vamos iniciar nosso primeiro projeto no Vite, com o VS Code aberto dentro de uma pasta, vamos executar o comando no terminal:

```
npm create vite@latest
```

Quando você executa **npm create vite@latest**, ele realiza os seguintes passos:

- **Download e Execução do Script de Criação:** O NPM baixa a ferramenta de criação do Vite.js (neste caso, a última versão) e a executa. Isso geralmente envolve o download de uma versão temporária do pacote para configurar o projeto.
- **Processo Interativo:** O comando inicia um assistente interativo que faz algumas perguntas para personalizar a criação do seu novo projeto Vite.js. As perguntas típicas incluem:
 - **Nome do projeto:** O nome da pasta/diretório onde o projeto será criado.
 - **Framework:** Qual framework ou biblioteca você deseja usar (por exemplo, Vue, React, Preact, Svelte, Lit, ou vanilla JavaScript).
 - **Variante do framework:** Se você prefere usar TypeScript ou JavaScript.
- **Geração da Estrutura do Projeto:** Com base nas suas respostas, o Vite cria a estrutura inicial do projeto, incluindo arquivos de configuração, dependências, e um conjunto básico de arquivos de exemplo.

Agora vamos executar as linhas de comando que o Vite nos traz após sua instalação, lembrando que o nome do diretório será o que você terá criado :

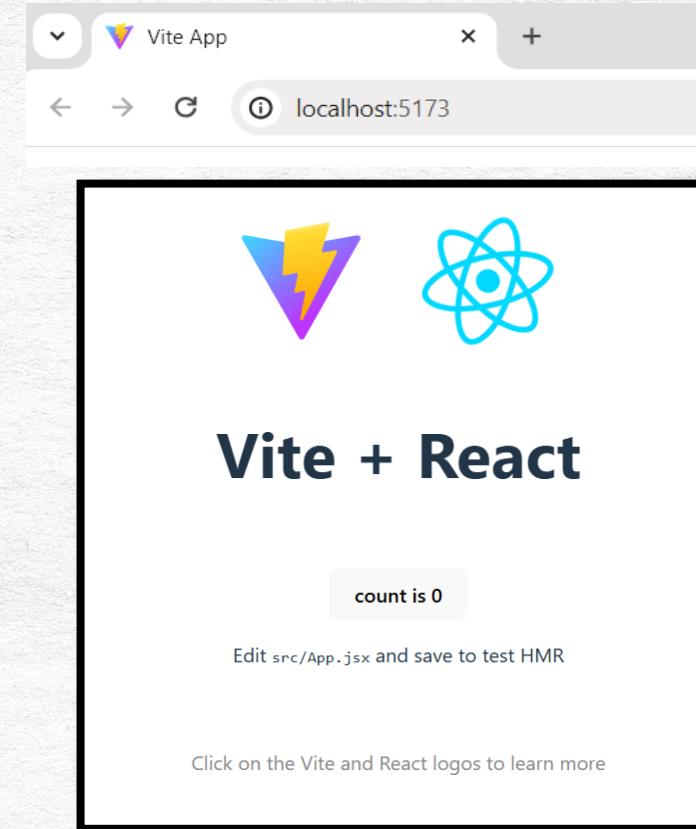
```
PS C:\Users\Edição 4\Downloads\Arquivos Iniciais da Aula 1\Arquivos Iniciais da Aula 1> npm create vite@latest
Project name: replica-spotify
Select a framework:
  React
Select a variant:
  TypeScript
  TypeScript + SWC
  ● JavaScript
  JavaScript + SWC
  React Router v7 >
```

```
PS C:\Users\Edição 4\Documents\Aulas\Vite> cd .\vite-project\
PS C:\Users\Edição 4\Documents\Aulas\Vite\vite-project> npm install
[#####.....] | idealTree:vite-project: timing idealTree:#root Completed in 3289ms
```

```
Done. Now run:

cd vite-project
npm install
npm run dev
```

```
VITE v5.2.11 ready in 172 ms
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```



O que é o npm install no Vite?

Quando criamos um projeto com o Vite, ele gera toda a estrutura inicial do código, mas as dependências do projeto ainda não estão instaladas. Para isso, usamos o comando:

```
npm install
```

Esse comando tem três funções principais:

- **Baixar todas as dependências listadas no projeto**, garantindo que ele funcione corretamente.
- **Criar a pasta node_modules**, onde os pacotes instalados são armazenados.
- **Preparar o ambiente para rodar a aplicação** com todas as bibliotecas necessárias.

Depois de rodar o npm install, podemos iniciar o servidor do projeto com:

```
npm run dev
```

O que é o package.json?

O package.json é um arquivo essencial para projetos que utilizam o Node.js e o NPM. Ele funciona como um **documento de configuração**, registrando informações sobre o projeto, como:

- ✓ O nome e a versão do projeto.
- ✓ As dependências instaladas.
- ✓ Os scripts para rodar o projeto.

Um exemplo de package.json gerado pelo Vite seria:

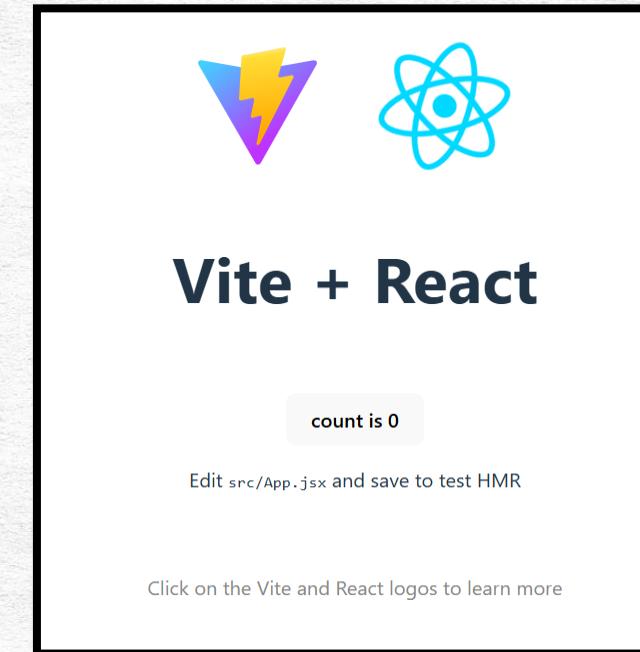
```
{
  "name": "meu-projeto",
  "version": "0.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.0.0",
    "react-dom": "^18.0.0"
  },
  "devDependencies": {
    "vite": "^4.0.0"
  }
}
```



O que o Vite cria no projeto?

Ao rodar o comando `npm create vite@latest`, o Vite gera automaticamente uma estrutura organizada do projeto, que inclui os seguintes arquivos e pastas:

```
/meu-projeto
  ├── node_modules/ <- Pasta onde ficam todas as dependências instaladas.
  ├── public/       <- Arquivos públicos, como favicon e imagens.
  └── src/
      ├── App.jsx    <- Componente principal do React.
      ├── main.jsx   <- Arquivo de entrada do projeto.
      ├── index.css  <- Arquivo de estilos globais.
  └── .gitignore    <- Lista de arquivos que não devem ser enviados para o Git.
  └── package.json <- Configuração do projeto e suas dependências.
  └── vite.config.js <- Arquivo de configuração do Vite.
```



Com essa estrutura pronta, podemos começar a desenvolver nosso projeto com React e Vite de forma rápida e otimizada.

HTML

HTML (HyperText Markup Language) é a linguagem de marcação utilizada para criar a estrutura e o conteúdo de uma página web. Com o HTML, é possível definir elementos como títulos, parágrafos, imagens, links e tabelas.

Esses elementos são organizados em tags, que são inseridas no código HTML para indicar a função de cada elemento. Por exemplo, a tag `<h1>` é utilizada para definir um título de nível 1, a tag `<p>` é utilizada para criar um parágrafo, e assim por diante.

Além dos elementos básicos, o HTML também permite o uso de atributos. Os atributos HTML são palavras especiais que fornecem informações adicionais sobre os elementos e definem características ou propriedades adicionais desses elementos.

Eles são especificados dentro da tag de abertura do elemento, com um nome e um valor. Por exemplo, o atributo `src` é utilizado na tag `` para indicar o caminho ou URL da imagem a ser exibida no documento.

Outros atributos comuns incluem `width` e `height`, que definem a largura e altura de um elemento, respectivamente. Existem também algumas tags especiais no HTML. A tag `
` é utilizada para quebrar uma linha de texto, sendo mais comum o uso da tag fechada `
`, que é suportada tanto em HTML quanto em XHTML. A tag `` é utilizada para destacar uma parte do texto, fazendo com que ele fique mais forte no navegador web.



Quando criamos um projeto React com o Vite, ele gera um arquivo index.html dentro da pasta principal do projeto. Esse arquivo tem uma estrutura um pouco diferente do index.html tradicional, porque o Vite o utiliza para carregar os arquivos do React de forma otimizada.

Vamos entender cada parte desse arquivo gerado pelo Vite!

Estrutura do index.html

Um exemplo de index.html gerado pelo Vite para um projeto React seria:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
    <script type="module" src="/src/main.jsx"></script>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Explicação do código

- ◆ <!DOCTYPE html>

Define que o documento está usando HTML5.

- ◆ <html lang="en">

Define o idioma da página como inglês. Podemos mudar para pt-BR caso nosso projeto seja em português.

- ◆ <head>

Nesta seção, configuramos metadados do documento, como:

- <meta charset="UTF-8" /> → Garante que caracteres especiais (como acentos) sejam exibidos corretamente.
- <meta name="viewport" content="width=device-width, initial-scale=1.0" /> → Faz com que a página se ajuste corretamente em telas de diferentes tamanhos.
- <title>Vite + React</title> → Define o título da aba do navegador.
- <script type="module" src="/src/main.jsx"></script>

Esse é o ponto chave! Diferente de um HTML tradicional, o Vite carrega o React de forma otimizada utilizando módulos ES6. O arquivo main.jsx dentro da pasta src/ é o ponto de entrada do nosso aplicativo React.

- ◆ <body>

- <div id="root"></div> → Essa div vazia é onde o React será injetado.

- O React, através do main.jsx, renderiza os componentes dentro dessa div, transformando o HTML da página dinamicamente.

Como o React interage com esse HTML?

O Vite não usa um HTML tradicional que renderiza diretamente o conteúdo na página. Em vez disso, ele carrega o main.jsx, que contém o código React responsável por manipular e renderizar os elementos dentro do <div id="root"></div>. Dentro do main.jsx, normalmente temos algo assim:

Aqui, o React está pegando a div id="root" e injetando dentro dela o componente <App />, que é onde criamos nossa interface gráfica.

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

Resumo

- O index.html do Vite é diferente do tradicional porque ele não contém código HTML dentro do <body> (somente uma div vazia com id="root").
- O React é carregado através do <script type="module" src="/src/main.jsx"></script>.
- O arquivo main.jsx injeta os componentes React dentro da div id="root", tornando a página interativa.
- O Vite usa esse formato para otimizar o carregamento do React e melhorar a performance do projeto.

Agora que entendemos a estrutura do index.html, podemos avançar na construção da aplicação! 🚀



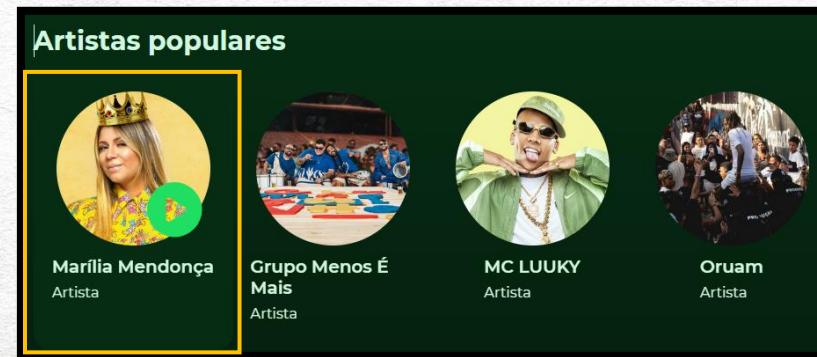
O que são componentes no React?

No React, **componentes** são partes reutilizáveis da interface de usuário. Eles funcionam como blocos independentes que podem ser combinados para criar aplicações complexas. Cada componente pode conter **HTML, CSS e JavaScript**, permitindo que sejam reutilizados e organizados de forma eficiente.

Componentes na prática

Se observarmos a imagem da aplicação, podemos perceber que **cada card de artista** (como o da Marília Mendonça, Grupo Menos é Mais, MC Luuky e Oruam) pode ser um **componente** separado.

Ou seja, ao invés de escrever o código desses cards repetidamente no HTML, criamos um **componente** chamado, por exemplo, ArtistCard. Assim, podemos reutilizá-lo várias vezes, apenas passando as informações de cada artista.



Vantagens de usar componentes

- ✓ **Reutilização:** Criamos um único código e usamos várias vezes.
- ✓ **Organização:** O código fica mais limpo e fácil de manter.
- ✓ **Escalabilidade:** Podemos adicionar novos artistas facilmente sem repetir código.

O React funciona dessa forma modular, separando partes da aplicação em **componentes reutilizáveis**, tornando o desenvolvimento mais eficiente e organizado. 

O que é o JSX?

JSX (**JavaScript XML**) é uma **sintaxe** utilizada no React que permite escrever **HTML dentro do JavaScript**. Ele facilita a criação de interfaces, tornando o código mais legível e intuitivo.

Por que usar JSX?

Normalmente, no JavaScript puro, precisaríamos criar elementos HTML usando `document.createElement()` ou `innerHTML`, o que pode ser confuso e trabalhoso. Com JSX, podemos escrever o código de forma mais simples e parecida com HTML.

Exemplo sem JSX (JavaScript puro):

```
const title = document.createElement("h1");
title.innerText = "Olá, mundo!";
document.body.appendChild(title);
```

Exemplo com JSX (React):

```
const App = () => {
  return <h1>Olá, mundo!</h1>;
};
```

Regras do JSX

1 Todo código JSX precisa estar dentro de um único elemento pai

O JSX não permite retornar dois elementos separados. Para evitar erros, envolva tudo dentro de uma `<div>` ou de um `Fragment (<>...</>)`.

2 As propriedades (props) devem ser escritas em camelCase

Diferente do HTML tradicional, onde as propriedades são escritas em letras minúsculas, no JSX elas seguem o formato `camelCase`. Por exemplo, class vira className, onclick vira onClick e for vira htmlFor.

3 Expressões JavaScript podem ser usadas dentro do JSX

Podemos adicionar valores dinâmicos dentro do JSX utilizando `{ }`. Isso permite, por exemplo, exibir variáveis, chamar funções ou realizar operações diretamente no código.

4 O JSX deve sempre ser fechado corretamente

Diferente do HTML, todas as tags em JSX precisam ser fechadas corretamente, mesmo aquelas que normalmente não precisariam, como `` e `
`.

5 JSX não aceita atributos sem valores

No HTML, podemos escrever atributos sem valores, como `disabled` ou `checked`. No JSX, esses atributos precisam receber `true` ou `false`.

Essas regras ajudam a manter o código JSX estruturado e evitam erros comuns ao escrever componentes React. 

O que é uma função?

Uma função em JavaScript é um **bloco de código reutilizável** que pode ser chamado para executar uma ação específica. No React, usamos funções para criar **componentes** e organizar melhor nosso código.

Partes de uma função

- 1 **Nome da função** → Toda função tem um nome que a identifica.
- 2 **Parâmetros (opcional)** → São valores que podemos passar para a função.
- 3 **Instruções** → O código que será executado dentro da função.
- 4 **Retorno (opcional)** → A função pode devolver um valor quando chamada.

Exemplo de uma função simples

- ◆ Essa função soma dois números e retorna o resultado:

Aqui:

- somar é o nome da função.
- a e b são os parâmetros.
- return a + b; faz a soma e retorna o resultado.

```
function somar(a, b) {  
    return a + b;  
}
```

Para chamar essa função:

```
console.log(somar(2, 3)); // Saída: 5
```

Funções no JSX (React)

No React, as funções podem ser usadas para criar **componentes**.

- ♦ Exemplo de um **componente funcional** que exibe uma mensagem:

```
function Mensagem() {  
  return <h1>Olá, bem-vindo ao React!</h1>;  
}
```

Aqui:

- Mensagem é o nome do componente.
- Ele não recebe parâmetros.
- Retorna um elemento JSX (<h1>).

Para exibir esse componente em um app React, basta chamá-lo como uma tag JSX:

```
<Mensaje</pre>
```

Conclusão

Funções são **blocos de código reutilizáveis** que ajudam a organizar e estruturar melhor um programa. No React, usamos **funções para criar componentes**, facilitando a criação de interfaces dinâmicas. 🚀

1. Ajustando o Título da Página

Quando abrimos uma página no navegador, o título exibido na aba é definido pela **tag <title>** dentro do arquivo index.html. No nosso caso, vamos alterar esse título para que ele reflita o nome do nosso projeto:

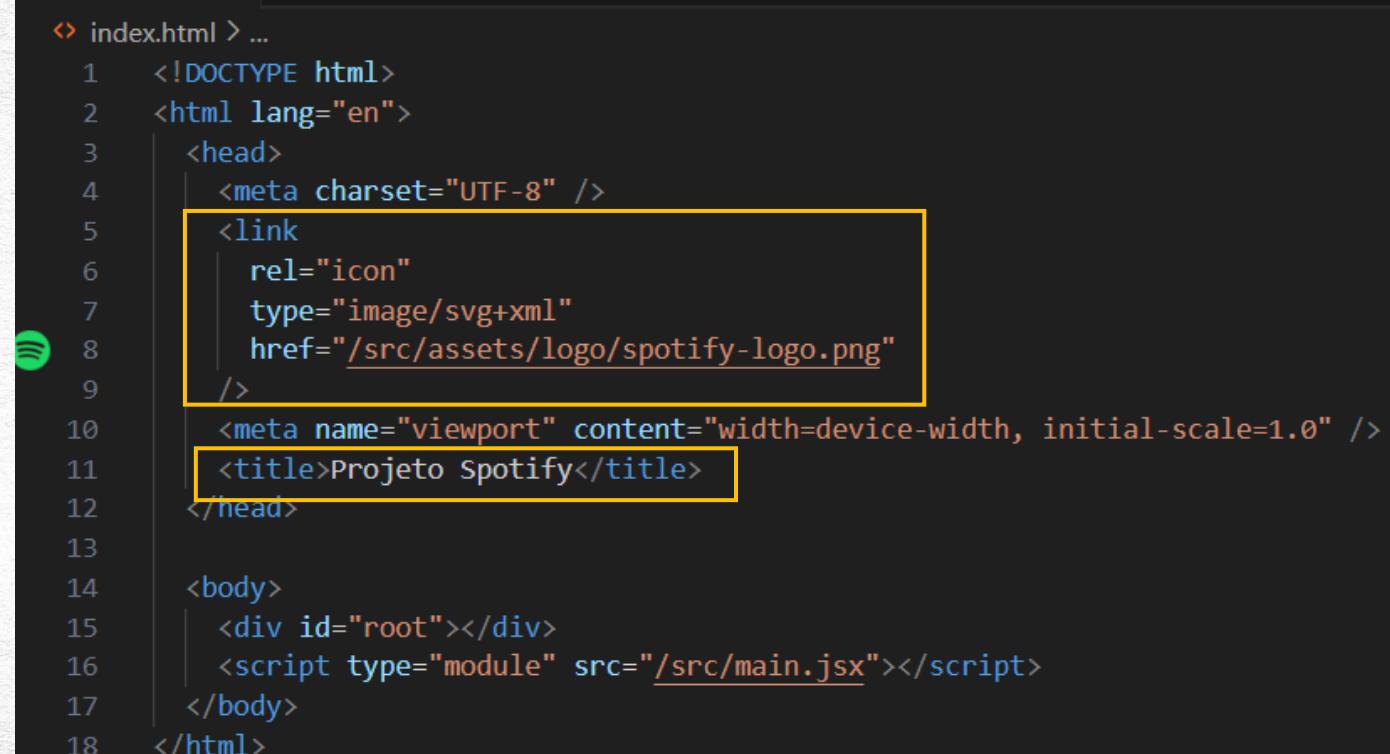
Isso faz com que o nome da aba do navegador apareça como "Projeto Spotify" ao rodarmos o aplicativo.

2. Alterando o Ícone da Página (Favicon)

O ícone que aparece ao lado do título da aba do navegador é chamado de **favicon**. Ele é definido na tag `<link rel="icon">`, que informa ao navegador qual imagem usar:

Aqui, estamos dizendo que o ícone será o arquivo **spotify-logo.png**, que está dentro da pasta `/src/assets/logo/`.

Se o ícone não aparecer de imediato, tente **limpar o cache do navegador** ou **reiniciar o servidor do Vite**.



```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link
6        rel="icon"
7        type="image/svg+xml"
8        href="/src/assets/logo/spotify-logo.png"
9      />
10     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
11     <title>Projeto Spotify</title>
12   </head>
13
14   <body>
15     <div id="root"></div>
16     <script type="module" src="/src/main.jsx"></script>
17   </body>
18 </html>
```

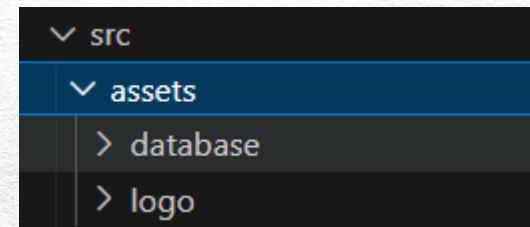
3. Copiando a Pasta de Assets

Agora, antes de seguirmos para os próximos passos do projeto, precisamos garantir que temos os arquivos necessários.

📌 **Tarefa:** Vá até a **pasta inicial da aula** e copie a pasta **assets** para dentro do seu projeto.

Dentro dessa pasta, você encontrará:

- ✓ O **logo** do Spotify, que usaremos como ícone da aba.
- ✓ Um **arquivo de dados**, que será nosso "banco de dados" para carregar informações no projeto.
- ⚠ **Por enquanto, ignore esse arquivo de dados.** Vamos explicá-lo melhor mais para frente na aula!



Agora, com o título e o ícone ajustados, podemos continuar desenvolvendo nosso projeto. 🚀

1. Importando o React

No início de um arquivo de componente React, geralmente importamos o React com:

```
import React from "react";
```

- 📌 No React moderno (com JSX), essa importação nem sempre é obrigatória, mas ainda é uma boa prática adicioná-la.

2. Criando um Componente com Arrow Function

No React, um **componente funcional** pode ser criado usando uma **arrow function**, como neste exemplo:

```
const App2 = () => {  
  return <div>App2</div>;  
};
```

- 📌 Esse código define um **componente chamado App2**, que retorna um elemento <div> com o texto "App2".

3. Exportando o Componente

Para que possamos usar esse componente em outro lugar do projeto, precisamos exportá-lo:

- 📌 **export default** permite que o componente seja importado com qualquer nome, sem necessidade de chaves {}.

```
export default App2;
```

- ♦ Exemplo de Importação

```
import MeuComponente from "./App2"; // Pode ser qualquer nome
```

Se o **export** não tiver o default:

```
export const App2 = () => <div>App2</div>;
```

Então a importação precisa ter o nome correto e estar entre {}:

```
import { App2 } from "./App2";
```

4. Atalho rafce no VS Code

O **rafce** é um **atalho da extensão ES7+ React/Redux/React-Native snippets** no VS Code.

Ele gera automaticamente um **componente funcional com exportação padrão**:

```
import React from "react";

const App2 = () => {
  return <div>App2</div>;
};

export default App2;
```

📌 Esse atalho **economiza tempo** na criação de componentes!



5. Nomeação de Componentes e Variáveis

Nomeação de Componentes → PascalCase

No React, componentes são nomeados usando a **primeira letra maiúscula** e cada palavra iniciando com maiúscula:

- ✓ MeuComponente
- ✓ ListaDeTarefas

Nomeação de Variáveis e Funções → camelCase

Variáveis e funções seguem o padrão **camelCase**, onde a primeira palavra começa com minúscula e as seguintes com maiúscula:

- ✓ minhaVariavel
- ✓ calcularSoma()
- ✓ nomeDoUsuario

Conclusão

- 📌 **Componentes no React são funções que retornam elementos JSX.**
- 📌 **Podemos exportar com export default (qualquer nome na importação) ou export normal (importação precisa ser entre {} e com o nome correto).**
- 📌 **O padrão PascalCase é usado para nomear componentes, e camelCase para variáveis e funções.**
- 📌 **O atalho rafce ajuda a criar rapidamente componentes funcionais no VS Code.**

Com isso, já temos uma base sólida para estruturar nossos componentes no React! 

Agora vamos organizar melhor nosso projeto criando uma **pasta components** dentro do diretório src. Dentro dessa pasta, vamos criar um **componente chamado Header.jsx**, que será responsável por armazenar o **cabeçalho** da nossa réplica do Spotify.

1. Criando a Pasta e o Arquivo



Passos:

- 1 **Dentro da pasta** src, crie uma nova pasta chamada **components**.
- 2 Dentro dessa pasta, crie um arquivo chamado **Header.jsx**.

2. Criando o Componente Header

Agora, dentro do arquivo Header.jsx, vamos criar nosso primeiro **componente funcional** no React.

```
import React from "react";
import logoSpotify from "../assets/logo/spotify-logo.png";

const Header = () => {
  return (
    <div className="header">
      <img src={logoSpotify} alt="Logo do Spotify" />

      <a className="header__link" href="/">
        <h1>Spotify</h1>
      </a>
    </div>
  );
};

export default Header;
```

3. Explicação do Código

◆ Importando Dependências

- import React from "react"; → Importa o React para criar o componente.
- import logoSpotify from "../assets/logo/spotify-logo.png"; → Importa a logo do Spotify, que será usada no cabeçalho.

◆ Criando o Componente Header

- Criamos uma **função chamada Header** que retorna um **elemento JSX**.
- Dentro do JSX, temos uma <div> com a classe header, que será estilizada com CSS.

◆ Estruturando o Conteúdo

- **Imagen da logo:**
- **Título clicável:** <h1>Spotify</h1>

◆ Exportando o Componente

- export default Header; permite que esse componente seja **importado e usado em outras partes do projeto**.

4. Utilizando o Componente Header

Agora que criamos o **componente Header**, podemos importá-lo dentro do nosso **App.jsx** para que ele apareça em todas as páginas:

📌 Abra o arquivo App.jsx e adicione a importação do Header:

Agora, toda vez que executarmos o projeto, o cabeçalho com a logo do Spotify será exibido! 🚀

Conclusão

- ✓ Criamos um **componente separado** para o cabeçalho.
- ✓ Organizamos o código dentro da pasta **components**.
- ✓ Importamos e utilizamos o Header dentro do App.jsx.

Agora, nossa aplicação está mais organizada e fácil de manter! No próximo passo, podemos continuar adicionando mais componentes. 🎶 🔥

```
import React from "react";
import Header from "./components/Header";

const App = () => {
  return (
    <div>
      <Header />
      <h2>Bem-vindo à réplica do Spotify!</h2>
    </div>
  );
};

export default App;
```

No JSX, existem duas formas de escrever tags de componentes:

1 Com Tag de Abertura e Fechamento

```
<Header></Header>
```

- Essa é a forma **tradicional** de escrever componentes.
- Usamos uma **tag de abertura (`<Header>`)** e uma **tag de fechamento (`</Header>`)**.
- Dentro dessas tags, podemos adicionar **conteúdo interno** se necessário.

2 Com Self-Closing Tag (`<Header />`)

```
<Header />
```

- Essa é a **forma mais curta e recomendada** quando o componente **não precisa de conteúdo interno**.
- Não precisa de uma tag de fechamento separada, pois o próprio JSX entende que a tag está fechada automaticamente.

❖ Regra importante:

- Componentes e elementos que **não precisam de conteúdo interno** (como ``, `
`, `<input />`, etc.) geralmente são escritos com **self-closing tag** para deixar o código mais limpo.

- **CSS (Cascading Style Sheets)** é a linguagem usada para estilizar elementos HTML. No CSS, **todo elemento é tratado como uma caixa** e segue um modelo chamado "**Modelo de Caixa**" (**Box Model**).

1 O Modelo de Caixa (Box Model)

Se inspecionarmos um elemento no **DevTools** (Ferramentas do Desenvolvedor), veremos um esquema que representa um **quadrado dividido em 4 partes principais**:

📌 Componentes do Box Model:

- **Conteúdo (content)** → O espaço interno onde o texto ou outros elementos aparecem.
- **Padding** → Espaço entre o conteúdo e a borda.
- **Borda (border)** → A linha que contorna o elemento.
- **Margem (margin)** → O espaço externo entre um elemento e outro.

Exemplo no DevTools

Ao inspecionar um elemento com CSS no navegador, ele pode aparecer assim:

```
margin: 20px  
border: 5px solid black  
padding: 20px  
width: 500px  
background-color: antiquewhite
```

O que isso significa?

- **width: 500px;** → Define a largura do conteúdo interno.
- **background-color: antiquewhite;** → Define a cor de fundo do elemento.
- **padding: 20px;** → Adiciona **espaço interno** entre o conteúdo e a borda.
- **border: solid 5px black;** → Adiciona uma **borda preta de 5px**.
- **margin: 20px;** → Adiciona **espaço externo** em volta do elemento.

Dica Importante:

Por padrão, o tamanho total do elemento pode ser maior do que o valor definido em width e height, pois o padding e a border são adicionados ao tamanho. Para garantir que o tamanho total não ultrapasse o width definido, podemos usar a propriedade:

```
box-sizing: border-box;
```

Isso faz com que **padding e borda sejam incluídos no tamanho total do elemento**.

2 Nomeação de Classes no CSS (Metodologia BEM)

Para manter o código CSS organizado, utilizamos a metodologia **BEM (Block, Element, Modifier)**. Essa abordagem facilita a leitura e a reutilização do código.

📌 Estrutura da Metodologia BEM:

- **Block (Bloco)** → Representa um **componente independente**.
- **Element (Elemento)** → Parte do bloco que **depende dele** para existir.
- **Modifier (Modificador)** → Variantes do elemento ou bloco.

Exemplos de Nomeação BEM

1 Bloco → Representa um **componente independente**

```
.header {  
    background-color: black;  
    color: white;  
}
```

- header → Bloco principal.

2 Elemento → Representa um **elemento dentro do bloco**

```
.header_link {  
    text-decoration: none;  
}
```

- header_link → Elemento dentro do header.

3 Modificador → Representa **uma variação do elemento**

```
.header_link--small {  
    font-size: 12px;  
}
```

- header_link--small → Modificador do header_link.

Tag Vazia no React – Fragment (<></>)

No React, quando precisamos agrupar vários elementos sem adicionar uma <div> extra, utilizamos **Fragments (<>...</>)**.

Exemplo sem Fragment:

```
return (
  <div>
    <h1>Olá</h1>
    <p>Bem-vindo ao projeto</p>
  </div>
);
```

Isso adiciona uma <div> desnecessária ao HTML.

Exemplo com Fragment:

```
return (
  <>
    <h1>Olá</h1>
    <p>Bem-vindo ao projeto</p>
  </>
);
```

Dessa forma, não adicionamos uma <div> extra ao HTML, mantendo o código mais limpo.



Agora que entendemos o **modelo de caixa (Box Model)** e a **metodologia BEM**, vamos criar nosso arquivo de estilos principal: index.css.

1 Reset CSS

Antes de estilizar os elementos, aplicamos um **reset CSS** para garantir que todas as margens e preenchimentos estejam padronizados em todos os navegadores.

💡 Reset CSS:

- **margin: 0; padding: 0;** → Remove margens e preenchimentos padrão.
- **box-sizing: border-box;** → Mantém a largura do elemento fixa, incluindo padding e borda.

```
* {  
    margin: 0;  
    padding: 0;  
    box-sizing: border-box;  
}
```

2 Configuração Geral do body

Agora definimos um estilo global para o corpo da página.

💡 Estilos do body:

- **background-color: black;** → Define o fundo preto.
- **color: white;** → Define a cor do texto como branco.
- **font-family: "Montserrat", sans-serif;** → Usa a fonte **Montserrat**.

```
body {  
    background-color: black;  
    color: white;  
    font-family: "Montserrat", sans-serif;  
}
```

3 Personalizando os Links <a>

Queremos que todos os links herdem a cor do texto e removemos o sublinhado padrão.

Estilos dos links:

- **color: inherit;** → O link herda a cor do texto do elemento pai.
- **text-decoration: none;** → Remove o sublinhado padrão dos links.

```
a {  
  color: inherit;  
  text-decoration: none;  
}
```

4 Definição do #root

O #root é a **div principal** onde o React renderiza toda a aplicação.

Estilização do #root:

- **height: 100svh;** → Define a altura total da tela, utilizando **100% do viewport height seguro (svh)**.
- **display: flex;** → Permite organizar os elementos dentro do #root.
- **flex-direction: column;** → Organiza os elementos em **coluna**.

```
#root {  
  height: 100svh;  
  display: flex;  
  flex-direction: column;  
}
```

5 Estilizando o Cabeçalho (.header)

Criamos o cabeçalho para manter a logo e os links bem organizados.

Estilos do header:

- **padding: 10px 20px;** → Adiciona um espaço interno ao cabeçalho.
- **display: flex;** → Organiza os itens horizontalmente.
- **justify-content: space-between;** → Distribui os itens uniformemente.
- **align-items: center;** → Alinha verticalmente ao centro.

```
.header {  
  padding: 10px 20px;  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
}
```



6 Interação com os Links no Cabeçalho

Queremos adicionar um efeito quando o usuário passa o mouse sobre os links.

💡 Estilos do Hover:

- **text-decoration: underline;** → Adiciona um sublinhado quando o usuário passa o mouse.

```
a {  
  color: inherit;  
  text-decoration: none;  
}
```

Conclusão

Agora temos um **arquivo index.css inicial** que organiza e estiliza a estrutura básica do nosso projeto.

- ✓ Resetamos os estilos padrão.
- ✓ Definimos um tema global (cores, fonte).
- ✓ Configuramos o **#root** para estruturar o conteúdo.
- ✓ Criamos o **cabeçalho (.header)** e ajustamos a interação dos links.

Na próxima etapa, continuaremos refinando nosso CSS e adicionando mais componentes! 💫 🎶

Agora vamos estruturar o componente **Main**, responsável por exibir as listas de **artistas e músicas populares**.

1 O que é o Main?

O Main será um **componente principal** da nossa aplicação. Ele conterá:

- Uma **lista de artistas populares**.
- Uma **lista de músicas populares**.
- Imagens e botões de play usando **FontAwesome**.

2 Criando o Arquivo Main.jsx

📌 Caminho para criar o arquivo:

📁 Dentro da pasta components, crie um arquivo chamado **Main.jsx**.

📌 Importação dos Recursos Necessários:

Antes de começar a escrever o componente, importamos:

- O **React**.
- O **ícone de play** da biblioteca FontAwesome.

3 Estrutura do Componente Main

Dentro do Main, organizamos o layout em seções:

- ◆ **Cabeçalho da Lista (item-list_header)**: exibe o título da seção e um link para "Mostrar tudo".
- ◆ **Container da Lista (item-list_container)**: contém os itens da lista.
- ◆ **Cada Item (single-item)**: representa um artista ou uma música.
- ◆ **Imagen e Ícone (single-item_div-image-button)**: contém a foto do artista/música e o botão de play.

4 Organização do Código

No código, temos dois blocos principais:

- ◆ **Lista de Artistas Populares**

- 📌 **Elementos principais:**

- ✓ Um **título (h2)** indicando "Artistas populares".
 - ✓ Um **link** para visualizar todas as opções.
 - ✓ Um **container** com os itens da lista.

- 📌 **Cada artista contém:**

- ✓ **Uma imagem** do artista.
 - ✓ Um **ícone de play** posicionado sobre a imagem.
 - ✓ O nome do artista.
 - ✓ A categoria (exemplo: "Artista").

- ◆ **Lista de Músicas Populares**

- 📌 **Semelhante à lista de artistas**, mas agora exibimos as músicas mais populares.

- 📌 **Cada música contém:**

- ✓ **Uma imagem** do álbum ou do artista.
 - ✓ Um **ícone de play** sobre a imagem.
 - ✓ O título da música.
 - ✓ A categoria (exemplo: "Música").

5 Uso de Classes CSS

Para estilizar corretamente, seguimos a **metodologia BEM**:

Exemplos de classes usadas no código:

- **item-list** → Define a lista de itens (artistas ou músicas).
- **single-item** → Representa cada item individual da lista.
- **single-item_image** → Estiliza a imagem do artista/música.
- **single-item_icon** → Define o botão de play.

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

const Main = () => {
  return (
    <div className="main">
      {/* Lista de Artistas Populares */}
      <div className="item-list">
        <div className="item-list__header">
          <h2>Artistas populares</h2>
          <a className="item-list__link" href="/">
            Mostrar tudo
          </a>
        </div>
      </div>
    </div>
  );
}

export default Main;
```



```
<div className="item-list__container">
  <div className="single-item">
    <div className="single-item__div-image-button">
      <div className="single-item__div-image">
        
    </div>

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

```
<div className="single-item__texts">
  <div className="single-item__2lines">
    <p className="single-item__title">Artista X</p>
    </div>

    <p className="single-item__type">Artista</p>
    </div>
  </div>
</div>
```

```
<div className="item-list__container">  
  <div className="single-item">  
    <div className="single-item__div-image-button">  
      <div className="single-item__div-image">  
          
      </div>  
    </div>
```

```
    <FontAwesomeIcon  
      className="single-item__icon"  
      icon={faCirclePlay}  
    />  
  </div>  
  
<div className="single-item__texts">  
  <div className="single-item__2lines">  
    <p className="single-item__title">  
      Amo Noite E Dia - Live In São Paulo / 2010  
    </p>  
  </div>  
  
  <p className="single-item__type">Música</p>  
</div>
```

```
export default Main;
```

📌 Explicação do Código

- **Estrutura Principal (`<div className="main">`)**
 - Engloba todo o conteúdo da seção principal.
- **Lista de Artistas Populares (`<div className="item-list">`)**
 - Exibe o título "Artistas populares" e um link "Mostrar tudo".
 - Contém um container onde cada artista é representado dentro de um `<div className="single-item">`.
- **Lista de Músicas Populares (`<div className="item-list">`)**
 - Estrutura semelhante à de artistas, mas agora para músicas.
- **Cada Item da Lista (`<div className="single-item">`)**
 - Contém uma **imagem** do artista/música.
 - Exibe um **ícone de play** sobre a imagem.
 - Mostra o **nome** e o **tipo** (Artista ou Música).
- **Uso do FontAwesomeIcon**
 - Importamos e utilizamos o ícone faCirclePlay da biblioteca FontAwesome para o botão de play.

Agora que criamos a estrutura do componente **Main**, vamos entender como estilizar essa seção para que ela tenha uma aparência mais agradável e se encaixe no design da réplica do Spotify.

1. Estilos para `.main`

Explicação

- background-color: brown;**
 - Define a cor de fundo como marrom, mas essa cor será sobreposta pela `background-image`.
- padding: 20px;**
 - Adiciona um espaço interno de **20px** dentro da `.main`, afastando os elementos das bordas.
- margin: 0 10px 10px;**
 - Adiciona **10px de margem lateral** e **10px de margem inferior**, criando um pequeno espaço ao redor do componente.
- border-radius: 15px;**
 - Faz com que os cantos do elemento fiquem **arredondados**.
- flex: 1;**
 - Define que esse elemento pode crescer e ocupar **o máximo de espaço disponível** dentro do layout.
- background-image: linear-gradient(to bottom, #062d14, black);**
 - Adiciona um efeito de **degradê**, indo de um tom verde escuro (#062d14) até preto (black).
 - Esse efeito cria uma **sensação de profundidade** e deixa o layout mais elegante.

```
.main {  
  background-color: brown;  
  padding: 20px;  
  margin: 0 10px 10px;  
  border-radius: 15px;  
  flex: 1;  
  background-image: linear-gradient(to bottom, #062d14, black);  
}
```

🎯 2. Estilos para .main_texts

📌 Explicação

- **display: flex;**
 - Ativa o **Flexbox**, permitindo que os elementos dentro de .main_texts sejam organizados de forma flexível.
 - **justify-content: space-between;**
 - Distribui os elementos horizontalmente, deixando **um espaço igual** entre eles.
 - **align-items: center;**
 - Garante que os itens fiquem **alinhados verticalmente ao centro**.
- 📌 **Resumindo:**
- Se tivermos um **título à esquerda** e um **link à direita**, eles ficarão bem distribuídos.

```
.main_texts {  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}
```

🔗 3. Efeito no Hover do Link

📌 Explicação

- **Quando o usuário passa o mouse sobre um link** com a classe .main_link, o efeito **sublinha** o texto.
- Isso melhora a **usabilidade** e deixa claro que o texto é **clicável**.

```
.main_link:hover {  
    text-decoration: underline;  
}
```