



# Intensivão de **JAVASCRIPT**

# T

## Apostila Completa

**AULA 3**

**Guia passo a passo: Projeto Cardápio Digital**



Parte 1

# Instalando as ferramentas





# Instalando as ferramentas – VS CODE

Caso você ainda não tenha o Visual Studio Code instalado, basta seguir os procedimentos abaixo. A instalação do VS Code é totalmente gratuita, e você pode usá-lo em seu computador sem precisar pagar nada. O link para fazer o download do programa é mostrado abaixo:

<https://code.visualstudio.com/>

- 1 – Baixe o arquivo de instalação correspondente ao seu sistema operacional (Windows, MacOS ou Linux).
- 2 – Execute o arquivo de instalação e siga as instruções na tela. Em geral, é só continuar clicando em "Próximo".
- 3 – No Windows, durante a instalação, marque a opção "Add to path" para adicionar o VS Code às suas variáveis de ambiente.



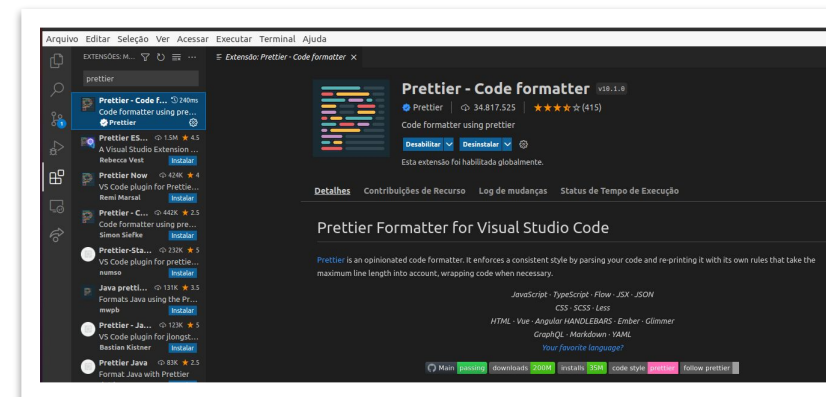


# Instalando as ferramentas – VS CODE - EXTENSÕES

As extensões no Visual Studio Code são pequenos programas que adicionam recursos extras ao editor de código. Elas são como "plugins" que podem ser instalados para personalizar e estender as funcionalidades do VS Code.

A extensão Prettier adiciona a capacidade de formatar automaticamente o código, tornando-o mais legível e organizado.

Para instalar a extensão Prettier no Visual Studio Code:



- Clique no ícone de extensões no menu lateral esquerdo (ou use o atalho "Ctrl+Shift+X").
- Na barra de pesquisa, digite "Prettier".
- A extensão "Prettier - Code Formatter" deve aparecer nos resultados da pesquisa. Clique no botão "Instalar" ao lado dela.
- Após a instalação, você pode configurar o Prettier como o formatador padrão para o seu projeto. Para fazer isso, abra as configurações do Visual Studio Code (menu "File" > "Preferences" > "Settings" ou use o atalho "Ctrl+,").
- Na barra de pesquisa das configurações, digite "Default Formatter" e selecione a opção "Editor: Default Formatter".
- No campo de valor, digite "esbenp.prettier-vscode" e pressione Enter para salvar as alterações.
- Agora, o Prettier está instalado e configurado como o formatador padrão no Visual Studio Code. Você pode usar o comando "Format Document" (menu "Edit" > "Format Document" ou atalho "Shift+Alt+F") para formatar o código.



# Instalando as ferramentas – 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:

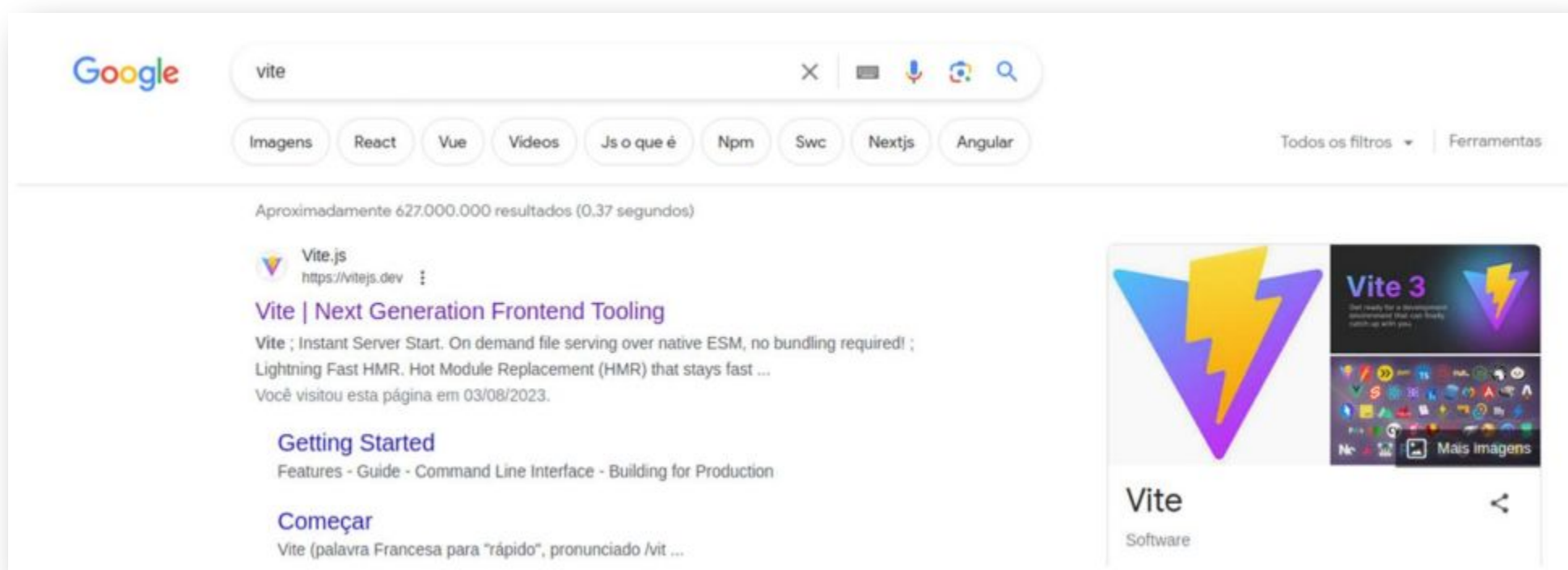


- Acesse o site oficial do Node.js em <https://nodejs.org>.
- Na página inicial, você verá duas versões para download: LTS (Long Term Support) e Current. A versão LTS é recomendada para a maioria dos usuários, pois é mais estável e possui suporte a longo prazo. Selecione a versão LTS ou a versão mais 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.



# Instalando as ferramentas – Vite.js

O Vite.js é um build tool (ferramenta de construção) e um servidor de desenvolvimento rápido para projetos web. Ele foi projetado para melhorar a experiência de desenvolvimento, oferecendo tempos de compilação instantâneos e recarregamento rápido do navegador. Ele permite que os desenvolvedores escrevam código moderno e aproveitem recursos como o hot module replacement (substituição de módulo em tempo real) para uma experiência de desenvolvimento mais produtiva.





# Instalando as ferramentas – Vite.js



O Vite é uma ferramenta de construção que oferece uma experiência de desenvolvimento mais rápida e leve para projetos web modernos. Ele simula um servidor durante o desenvolvimento para melhorar o desempenho e a produtividade. Aqui estão os passos para entender como o Vite simula um servidor:

- O Vite possui uma opção de configuração chamada `server.port` que permite especificar a porta do servidor. Por padrão, ele usa a porta 3000.
- Durante o desenvolvimento, o Vite utiliza um servidor de desenvolvimento embutido para fornecer funcionalidades adicionais, como substituição de módulo instantânea (HMR) extremamente rápida. O HMR permite atualizar o código no navegador em tempo real, sem a necessidade de recarregar a página.
- Além disso, o Vite oferece uma API de baixo nível chamada `configureServer`, que permite adicionar intermediários de SSR (Server-Side Rendering) ao servidor de desenvolvimento. Os intermediários de SSR pré-interpretam a aplicação para HTML no lado do servidor e, em seguida, a hidratam no cliente.



# Instalando as ferramentas – Vite.js



- O Vite também possui comandos de linha de comando, como `vite dev` e `vite preview`, que podem ser usados para executar aplicações com SSR. É possível adicionar intermediários de SSR ao servidor de desenvolvimento com `configureServer` e ao servidor de pré-visualização com `configurePreviewServer`.
- Durante o processo de construção, o Vite utiliza o Rollup para empacotar o código e produzir recursos estáticos altamente otimizados para produção. Isso garante um desempenho eficiente e uma melhor experiência para os usuários.

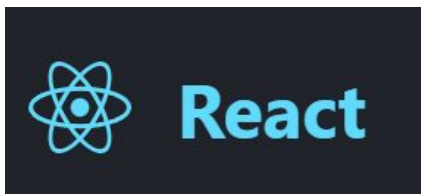
<https://vitejs.dev/>

Neste momento, estamos explicando os conceitos do Vite e como ele funciona internamente. Estamos fornecendo uma visão detalhada de como o Vite opera e como ele trabalha nos bastidores. Mas, quando começarmos nosso projeto, vamos criar um programa usando o Vite para rodar nosso site de E-Commerce. E explicaremos passo a passo como dar os primeiros passos nesse processo.





# Instalando as ferramentas – React.js



O React é uma biblioteca JavaScript que permite aos desenvolvedores criar interfaces de usuário interativas e reutilizáveis. Sua abordagem baseada em componentes facilita a construção de aplicativos web modulares e escaláveis, onde elementos como itens individuais, categorias e botões de interação são encapsulados em componentes distintos.

Estamos fornecendo uma visão detalhada de como o React opera e como ele trabalha nos bastidores. Quando começarmos nosso projeto, vamos criar um programa usando o React para rodar nosso site de cardápio digital. O React oferece uma manipulação eficiente do DOM, garantindo que as atualizações na interface do usuário sejam processadas de forma rápida e responsiva. Isso é crucial para proporcionar aos usuários uma experiência de navegação fluida e ágil, especialmente ao explorar as diversas opções disponíveis no cardápio.

Além disso, o React oferece um sistema eficaz para gerenciar o estado da aplicação. Isso é fundamental em um cardápio digital, onde o estado dos itens selecionados pelos usuários precisa ser atualizado e refletido instantaneamente na interface, garantindo uma interação intuitiva e em tempo real com o cardápio. Durante o processo, explicaremos passo a passo como dar os primeiros passos nesse processo de desenvolvimento, integrando o React em nosso projeto de cardápio digital.

# Instalando as ferramentas – Dúvidas: Vite ou Live Server?



VITE e Live Server são ferramentas diferentes com propósitos distintos no desenvolvimento web. O Live Server é uma ferramenta simples que serve arquivos estáticos e atualiza automaticamente o navegador quando os arquivos são modificados. É útil para projetos simples e pequenos, onde não há necessidade de muitas configurações adicionais.

Por outro lado, o VITE é uma ferramenta mais avançada e otimizada para o desenvolvimento moderno de aplicações web. Ele oferece um ambiente de desenvolvimento rápido e eficiente, com recarga rápida (hot module replacement) e suporte para módulos ES6 nativos, o que significa que você pode usar importações nativas de módulos no navegador sem a necessidade de transpilação.

Portanto, ao utilizar o React para construir um projeto de cardápio digital, o VITE seria uma escolha mais robusta e eficiente. Ele oferece uma experiência de desenvolvimento mais avançada, com melhor desempenho e suporte a recursos modernos do JavaScript, o que pode melhorar significativamente a produtividade e a qualidade do seu projeto. Além disso, o VITE é altamente otimizado para o desenvolvimento com React, o que torna a integração entre essas tecnologias mais suave e eficiente.

Parte 2

# Apresentação do Projeto Cardápio Digital!

# Apresentação do Projeto Cardápio Digital!

Bem-vindos ao **Cardápio Digital**, onde vamos explorar a implementação da lógica utilizando React. Nosso objetivo é oferecer uma experiência dinâmica para a seleção de pratos, sobremesas e bebidas, tornando a escolha simples e divertida. Com o uso do React, iremos facilitar a interação do usuário, permitindo uma navegação intuitiva entre as opções do menu.

Dentro deste guia, você encontrará um passo a passo detalhado para implementar a lógica em React no nosso site **Cardápio Digital**.

Estamos empolgados para criar uma plataforma gastronômica interativa e convidativa. Prepare-se para uma experiência deliciosa e inovadora!

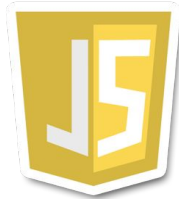


Parte 3

# 0 Javascript

# O Javascript

## O que é JavaScript?



JavaScript, uma linguagem de programação de alto nível e dinâmica, é um elemento vital na construção de experiências interativas no mundo digital. Amplamente utilizado para aprimorar a interatividade em páginas da web, o JavaScript capacita os desenvolvedores a controlar elementos, criar animações, processar dados e muito mais. Sua execução no navegador do cliente agiliza a interação do usuário, já que o código é processado localmente, resultando em uma experiência mais ágil e eficiente.

No contexto do nosso projeto de cardápio digital, o JavaScript desempenha um papel central na dinâmica e na interatividade da interface. Ao integrá-lo às páginas web, os desenvolvedores podem personalizar e aprimorar a experiência do usuário de forma única e eficaz.

Criado por Brendan Eich em 1995 com o intuito inicial de trazer animações e alertas às páginas da web, o JavaScript evoluiu ao longo do tempo. Com o suporte da Microsoft em 1996, tornou-se uma linguagem fundamental na programação web, impulsionando a expansão e a inovação na criação de conteúdo digital.

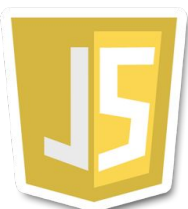
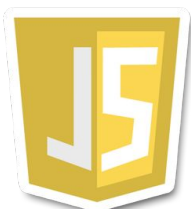
# O Javascript

A evolução do JavaScript trouxe o ES6 (ECMAScript 2015), uma versão que introduziu recursos avançados, como declarações de variáveis mais flexíveis (let e const), classes, módulos e promessas. Esses aprimoramentos tornaram o JavaScript uma linguagem mais robusta e adaptável.

Quando integramos o JavaScript ao nosso projeto de cardápio digital, ele se torna a força propulsora por trás da experiência interativa. Encarregado de manipular HTML e CSS, o JavaScript possibilita a atualização dinâmica da interface do usuário. Ele opera no lado do cliente, assegurando uma resposta ágil e eficaz, sem depender excessivamente do servidor.

No contexto específico do cardápio digital, o JavaScript desempenha o papel de condutor, controlando a exibição de itens, respondendo às interações do usuário e acrescentando elementos dinâmicos à experiência de navegação. Com ele, podemos oferecer aos clientes a capacidade de explorar o cardápio, personalizar suas escolhas e interagir de forma intuitiva com as opções disponíveis.

Em resumo, a implementação do JavaScript em nosso projeto de cardápio digital não só proporciona uma experiência interativa, mas também adiciona profundidade e controle aos elementos visuais, elevando a experiência de escolha e personalização para os usuários.



Parte 4

# Iniciando Projeto - Vite





# Iniciando Projeto - Vite



Para iniciar o nosso projeto, vamos criar uma nova pasta com o nome "Projetos". Essa pasta será o local onde iremos armazenar todos os arquivos do nosso site de Cardápio Digital. Dentro dessa pasta, colocaremos os arquivos que serão utilizados para estilização, arquivos adicionais (imagens, cardápio) e configuração do ambiente de desenvolvimento, como a criação do nosso programa utilizando o Vite.

Vou te explicar passo a passo como criar uma pasta nova no Windows:

- Primeiro, abra o "Explorador de Arquivos" clicando no ícone da pasta amarela na barra de tarefas ou pressionando a tecla "Windows" + "E" no teclado.
- Navegue até o local onde você deseja criar a nova pasta. Por exemplo, você pode escolher criar a pasta na área de trabalho ou dentro de outra pasta existente.
- Com o local selecionado, clique com o botão direito do mouse em um espaço vazio da janela do Explorador de Arquivos. Um menu de opções será exibido.

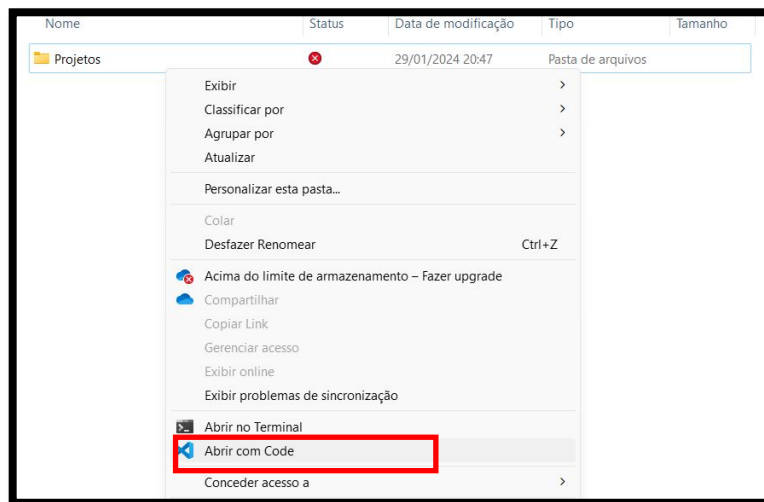
# Iniciando Projeto - Vite

- No menu de opções, passe o cursor sobre a opção "Novo" e, em seguida, clique em "Pasta". Uma nova pasta será criada com o nome "Nova pasta" destacado.
- Digite o nome desejado para a nova pasta. No nosso caso, digite "Projetos" como o nome da pasta.
- Pressione a tecla "Enter" no teclado para confirmar o nome da pasta. A nova pasta será criada no local selecionado.

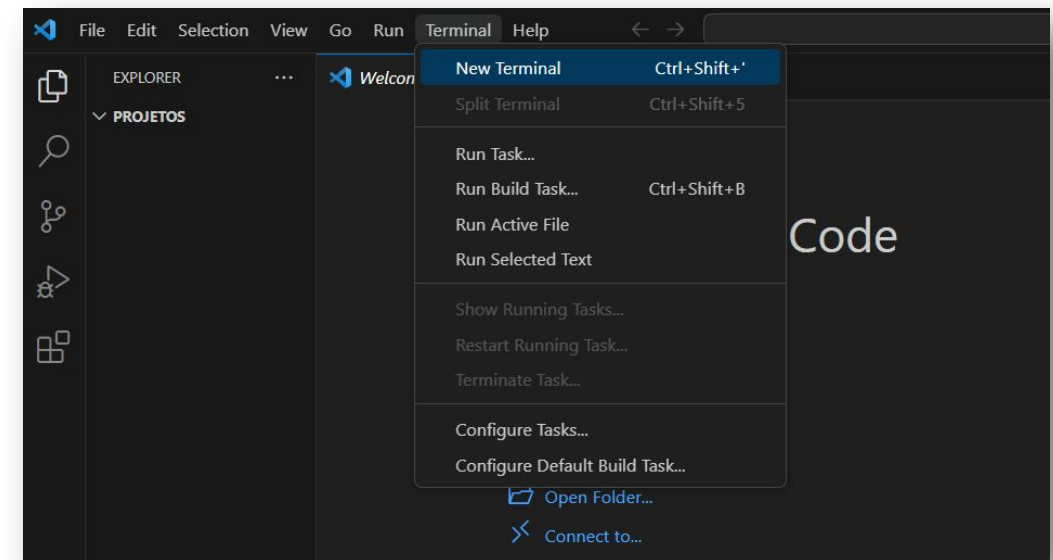
Nome	Status	Data de modificação	Tipo	Tamanho
 Projetos		26/09/2025 16:33	Pasta de arquivos	

# Iniciando Projeto - Vite

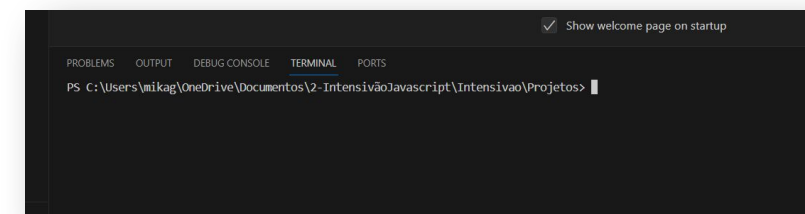
**1** - Dentro da pasta recém criada "Projetos", você irá clicar com o botão direito do mouse e abrir a opção "Abrir com Code" para iniciarmos com o Vite



**2** - Abra o terminal do VS Code, clicando na aba Terminal e depois em "New Terminal"(Novo Terminal):



E um terminal irá abrir na parte inferior do VS Code:



# Iniciando Projeto - Vite

**3** - No terminal, digite o comando NPM do Vite (o comando "npm create vite@latest" é usado para criar um novo projeto Vite usando o gerenciador de pacotes npm)

```
npm create vite@latest
```

**4** - O terminal irá iniciar o processo do Vite, após o comando de NPM. O próximo passo é escolher um nome para o seu projeto Vite.

- O "**nome**" se refere ao nome do seu projeto Vite. Por exemplo, você pode dar a ele um nome descritivo, como "hashtaurante".

```
PS C:\Users\mikag\OneDrive\Projetos> npm create vite@latest  
✓ Project name: ... hashtaurante
```

# Iniciando Projeto - Vite

**5** – Vamos selecionar o template ou estrutura inicial para o seu projeto Vite, que será o modelo React

- **"React"**: refere-se a um projeto Vite que utiliza a biblioteca JavaScript React para construir interfaces de usuário dinâmicas e reutilizáveis em aplicativos web. Com o React, os desenvolvedores podem criar componentes modulares e eficientes para uma experiência de desenvolvimento moderna e escalável.

```
? Select a framework: » - Use arrow-keys. Return to submit.  
  Vanilla  
  Vue  
  > React  
  Preact  
  Lit  
  Svelte  
  Solid  
  Qwik
```

**6** – Por fim, ao escolher "javascript" após executar o comando "npm create vite@latest", você está selecionando o template de JavaScript puro para o seu projeto Vite.

```
✓ Select a variant: » JavaScript
```

# Iniciando Projeto - Vite

Você deve se deparar com uma janela parecida com essa, o que quer dizer que as configurações foram feitas com Sucesso.

```
Scaffolding project in C:\Users\mikag\OneDrive\Projetos\hahstaurante...  
  
Done. Now run:  
  
  cd hashtaurante  
  npm install  
  npm run dev
```

E agora vamos iniciar a nossa aplicação Vite no navegador, seguindo alguns comandos.

**1** – No terminal digite "cd nomeDaPastaProjeto":

```
PS C:\Users\mikag\OneDrive\Projetos> cd hashtaurante  
PS C:\Users\mikag\OneDrive\Projetos\hashtaurante> █
```

# Iniciando Projeto - Vite

**2** – Execute NPM install para instalar as dependências de um projeto. Quando você executa o comando "npm install" no terminal, o npm verifica o arquivo "package.json" do seu projeto para identificar as dependências listadas nele.

```
PS C:\Users\mikag\OneDrive\Projetos\hashtaurante> npm install  
[.....] | idealTree:hashtaurante: sill idealTree buildDeps
```

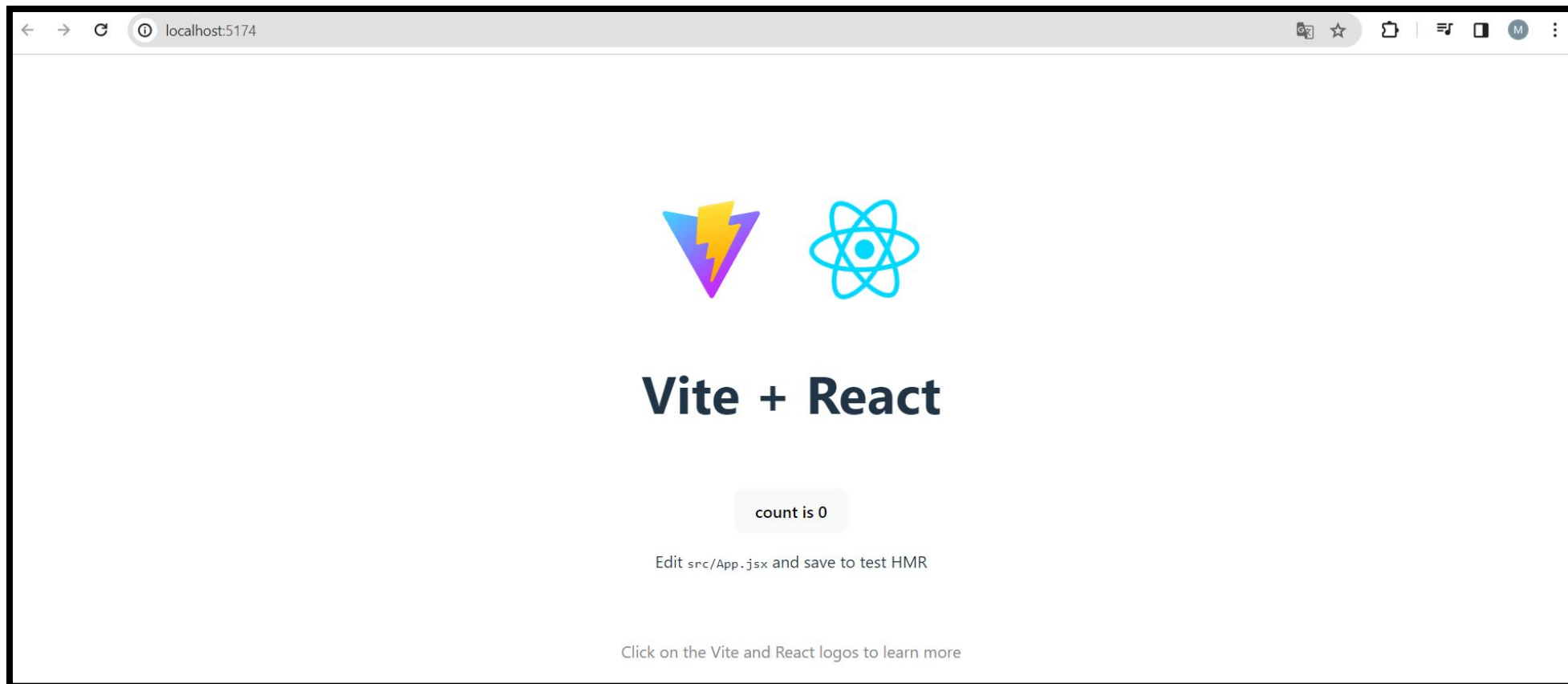
```
PS C:\Users\mikag\OneDrive\Projetos\hashtaurante> npm install  
  
added 270 packages, and audited 271 packages in 1m  
  
97 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

**3** – Rode "npm run dev" – para iniciar o servidor de desenvolvimento do Vite no navegador.

```
PS C:\Users\mikag\OneDrive\Projetos\hashtaurante> npm run dev  
  
> hahstaurante@0.0.0 dev  
> vite  
  
Port 5173 is in use, trying another one...  
  
VITE v5.0.12 ready in 3639 ms  
  
→ Local:   http://localhost:5174/  
→ Network: use --host to expose  
→ press h + enter to show help
```

# Iniciando Projeto - Vite

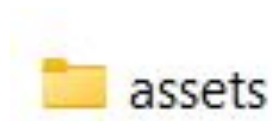
**Pronto!** A sua aplicação Vite já está no ar! Você pode abrir o seu navegador e acessar o endereço "http://localhost:5174" (ou a porta específica informada no terminal) para visualizar a sua aplicação Vite em tempo real.



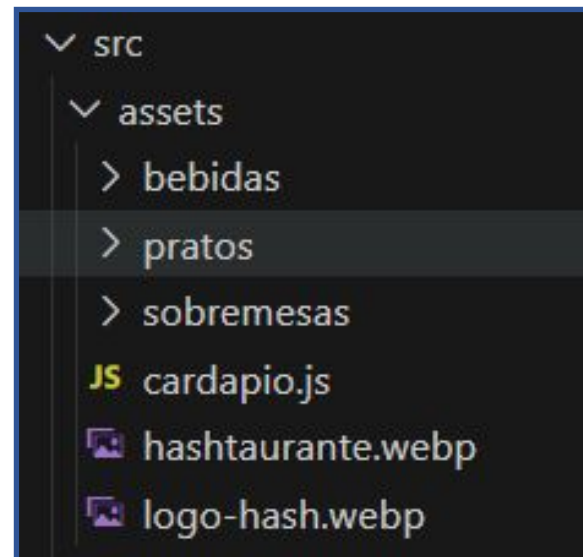
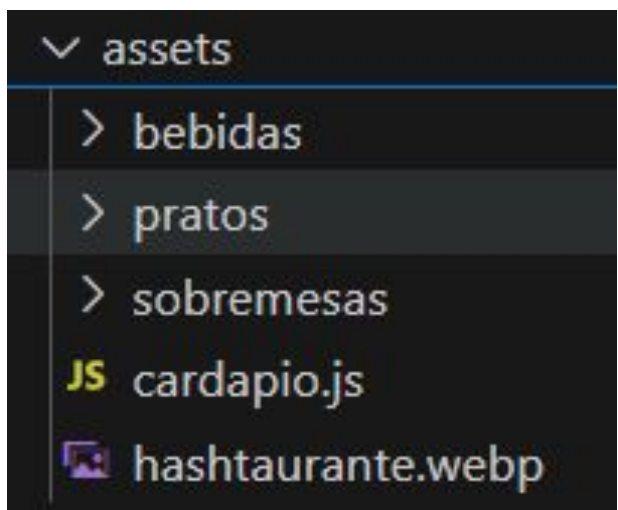


# Iniciando Projeto - Vite

Nesse momento, vocês farão o download dos arquivos disponíveis dentro da pasta "DoZero", esta pasta possui uma subpasta chamada "assets" :



Adicione os arquivos à pasta "assets" do nosso projeto, localizada dentro da pasta que configuramos recentemente com o Vite. Isso significa que os arquivos devem ser colocados dentro da pasta "src" que está dentro do diretório do seu projeto Vite. E você pode substituir os arquivos existentes.



# Iniciando Projeto - Vite

Com o nosso projeto aberto no VS Code, vamos remover os arquivos que não serão necessários para nós. Isso acontece porque o Vite já cria uma estrutura inicial para a aplicação, mas queremos criar nosso próprio site de Cardápio Digital.

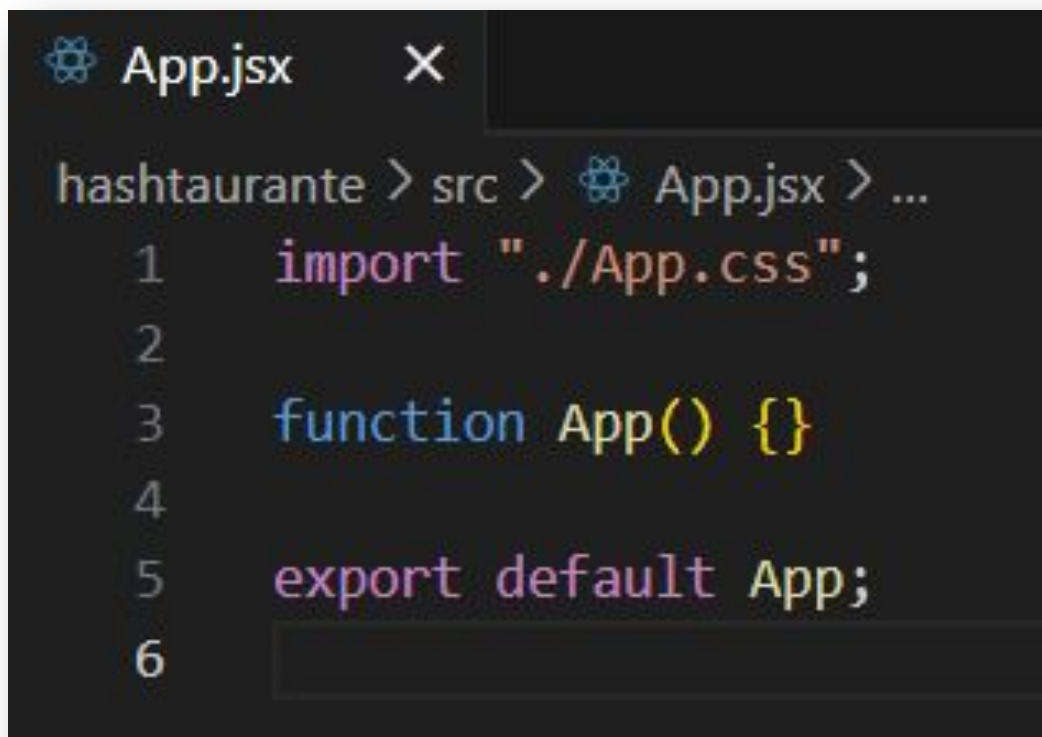
**Index.css:** Vamos excluí-lo. E também vamos remover a linha que importa o arquivo principal de estilo, que geralmente está no arquivo **main.jsx**.

```
hashtaurante > src > main.jsx
1  import React from 'react'
2  import ReactDOM from 'react-dom/client'
3  import App from './App.jsx'
4  import './index.css'
5
6  ReactDOM.createRoot(document.getElementById('root')).render(
7    <React.StrictMode>
8      <App />
9    </React.StrictMode>,
10 )
11
```

O JSX é uma extensão do JavaScript frequentemente usada com o React para escrever componentes de interface de usuário de forma mais legível e expressiva. Ela permite que você misture marcação HTML e JavaScript em um único arquivo, simplificando o desenvolvimento de aplicativos web interativos.

# Iniciando Projeto - Vite

O próximo passo é abrir o arquivo App.jsx e excluir o conteúdo da função App, pois vamos implementar as funcionalidades do nosso Cardápio Digital. Isso significa que vamos limpar o arquivo App.jsx, removendo todo o código dentro da função App, para que possamos começar do zero e adicionar as nossas próprias funcionalidades personalizadas para o cardápio digital. E excluindo as importações de arquivos que não vamos utilizar:



```
App.jsx X
hashtaurante > src > App.jsx > ...
1  import './App.css';
2
3  function App() {}
4
5  export default App;
6
```

Parte 5

# Primeiros passos



# Primeiros Passos

O arquivo `main.jsx` é o ponto de entrada principal da nossa aplicação React. Ele importa as bibliotecas necessárias e nosso componente principal (`App`), e em seguida usa o `ReactDOM` para renderizar esse componente na página web, dentro do elemento com o id `"root"`.

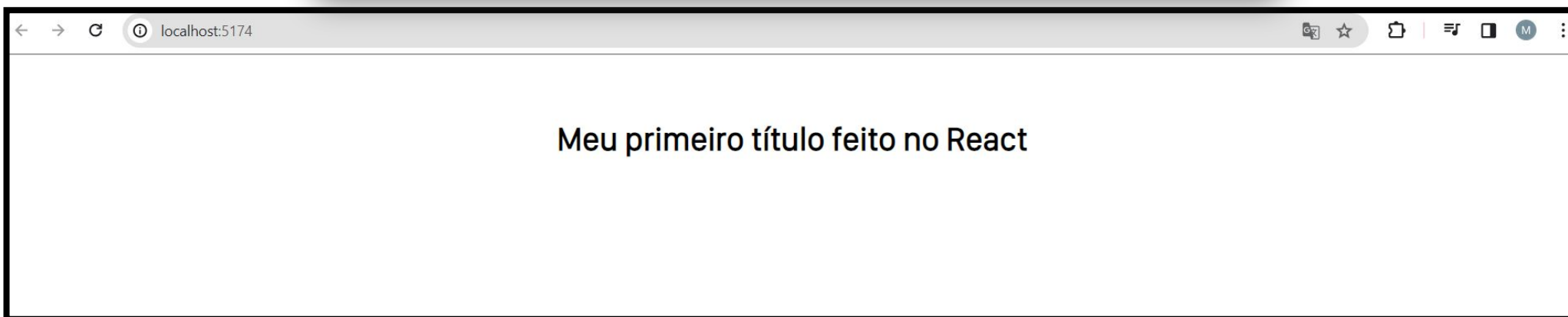
No arquivo `main.jsx`:

- **`import React from "react";`**: Importamos a biblioteca React para criar interfaces de usuário.
- **`import ReactDOM from "react-dom/client";`**: Importamos o `ReactDOM`, usado para renderizar elementos React na web.
- **`import App from "./App.jsx";`**: Importamos o componente `App` de `App.jsx`, trazendo o código de nossa aplicação.
- **`ReactDOM.createRoot(document.getElementById("root")).render(`**: Criamos uma raiz de renderização na página, especificamente no elemento com o id `"root"`.
- **`<React.StrictMode>` e `</React.StrictMode>`**: Envolve o componente `App` com o modo estrito do React, identificando possíveis problemas e fornecendo avisos extras.
- **`<App />`**: Renderiza o componente `App` dentro do **`<React.StrictMode>`**, garantindo que todo o seu conteúdo e comportamento estejam ativos na interface do usuário.

# Primeiros Passos

Assim, ao alterarmos o arquivo `App.jsx`, podemos renderizar nosso primeiro elemento. O React nos permite interagir com JavaScript e HTML em um mesmo arquivo, facilitando o desenvolvimento de interfaces dinâmicas e interativas.

```
hashtaurante > src > App.jsx > ...  
1  import './App.css';  
2  
3  function App() {  
4    return <h1>Meu primeiro título feito no React</h1>;  
5  }  
6  
7  export default App;  
8
```



# Primeiros Passos

Mas o React vai ainda mais além! Observe o código a seguir:

```
hashtaurante > src > App.jsx > ...  
1  import './App.css';  
2  
3  ✓ function App() {  
4    const tituloPagina = "Meu primeiro título feito no React";  
5    return <h1>{tituloPagina}</h1>;  
6  }  
7  
8  export default App;
```

Vamos explicar a lógica por trás da função `App()` usando JSX, uma sintaxe do React que combina JavaScript com HTML de forma dinâmica:

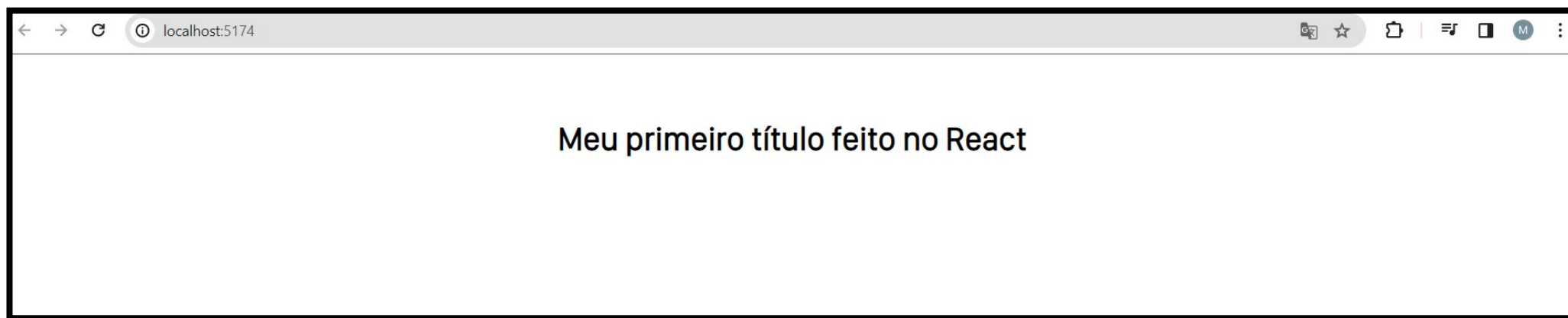
**Declaração da função `App()`:** Aqui, estamos declarando uma função chamada `App()`. No React, os componentes são frequentemente definidos como funções que retornam elementos JSX.

**Definição de uma constante:** Dentro da função `App()`, declaramos uma constante chamada `tituloPagina` e atribuímos a ela o valor "Meu primeiro título feito no React". Esta constante armazena o conteúdo que queremos exibir dinamicamente na página.

# Primeiros Passos

**Retorno do JSX:** Usando JSX, podemos escrever código que se parece muito com HTML dentro de JavaScript. No retorno da função, colocamos o elemento `<h1>` e dentro dele, usamos chaves `{}` para inserir o conteúdo da constante `tituloPagina`. Isso permite que o valor contido na constante seja exibido dinamicamente como o texto do título na página, tornando-o "Meu primeiro título feito no React".

Em resumo, a função `App()` usa JSX para criar dinamicamente um título de página que exibe o conteúdo da constante `tituloPagina`. Essa é uma das principais vantagens do React: a capacidade de criar interfaces de usuário dinâmicas e interativas de forma simples e elegante.





Parte 6

# Construindo a estrutura da página

# Construindo a Estrutura da página

Vamos começar a estruturar nosso projeto e o primeiro item que queremos mostrar na página é a imagem de um restaurante, que será como uma "capa" para a página. Para que isso funcione corretamente, é importante entender como o React lida com o carregamento de arquivos.

O React não carrega arquivos diretamente do sistema de arquivos local, ou seja, quando você usa a tag ``, o React tentará carregar a imagem a partir do caminho relativo especificado (`./assets/hashtaurante.webp`). No entanto, o React não pode carregar imagens diretamente de arquivos locais dessa maneira.

```
✓ function App() {  
  |   return ;  
  |  
  }
```

O React espera que as imagens sejam importadas como módulos JavaScript e referenciadas como variáveis dentro do código. Isso permite que o webpack (ou outro bundler) processe essas imagens durante a compilação e as inclua no pacote final da aplicação.

Então, em vez de referenciar a imagem diretamente de um arquivo local como você faria no HTML tradicional, no React você deve importar a imagem no arquivo JavaScript ou JSX onde você deseja usá-la.

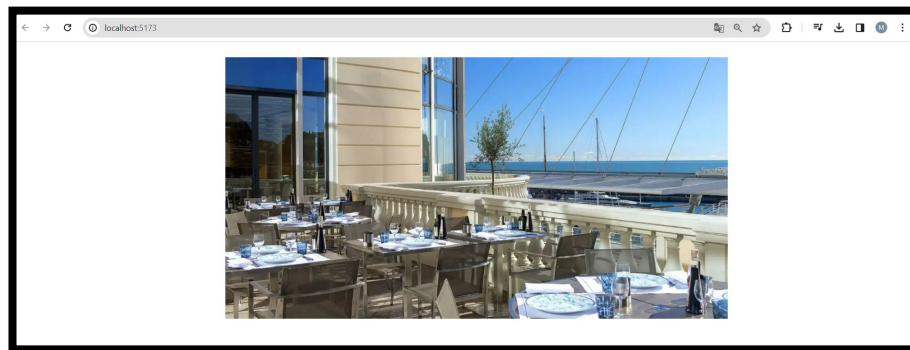
# Construindo a Estrutura da página

No React, o **import** é uma declaração usada para importar funcionalidades, componentes, módulos ou arquivos de outros arquivos JavaScript ou JSX para o arquivo atual em que está sendo usado. Isso é essencial para organizar e modularizar o código em aplicações React.

Por exemplo, ao construir um aplicativo React, você pode ter vários componentes em arquivos separados para melhorar a legibilidade e a manutenção do código. Usando **import**, você pode trazer esses componentes de outros arquivos para o arquivo principal ou para outros componentes, permitindo o uso e a composição desses componentes em diferentes partes do aplicativo.

Nesse momento iremos importar o arquivo de imagem que está localizado na pasta assets, e vamos utilizar dentro do nosso projeto. Essa imagem é referente a uma imagem de restaurante.

```
hashtaurante > src > App.jsx > ...  
1  import './App.css';  
2  import hashtauranteImg from './assets/hashtaurante.webp';  
3  
4  function App() {  
5    |   return <img src={hashtauranteImg} />;  
6  }  
7  
8  export default App;
```



# Construindo a Estrutura da página

Após a importação da imagem, vamos adicionar estilos para ela. O arquivo CSS que você baixou contém toda a folha de estilo para o projeto. Como estamos focando no desenvolvimento em React e JavaScript, vamos utilizar esse arquivo CSS para estilizar nossos componentes e elementos da interface.

```
hashtaurante > src > App.jsx > ...
1  import './App.css';
2  import hashtauranteImg from './assets/hashtaurante.webp';
3
4  function App() {
5    |   return <img src={hashtauranteImg} className="capa" />;
6    |
7    |
8    export default App;
```

O **className** é uma propriedade usada no React para atribuir classes CSS a elementos JSX. No React, a propriedade **className** é usada em vez de **class** para evitar conflitos com a palavra-chave **class** do JavaScript.

# Construindo a Estrutura da página

```
hashtaurante > src > JS cardapio.js > burgerPicanha
1 import enroladinhoMignonImg from './assets/pratos/enroladinho-mignon.jpeg';
2 import casal20Img from './assets/pratos/casal-20.jpeg';
3 import burgerPicanhaImg from './assets/pratos/burger-picanha.jpeg';
4 import fishChipsImg from './assets/pratos/fish-chips.jpeg';
5 import risotoCamaraoTrufadoImg from './assets/pratos/risoto-camarao-trufado.jpeg';
6
7 import brownieImg from './assets/sobremesas/brownie.jpeg';
8 import cocadaFornoImg from './assets/sobremesas/cocada-forno.jpeg';
9 import petitGateauImg from './assets/sobremesas/petit-gateau.jpeg';
10
11 import aguaSemGasImg from './assets/bebidas/agua-sem-gas.jpeg';
12 import aguaComGasImg from './assets/bebidas/agua-com-gas.jpeg';
13 import cocaColaImg from './assets/bebidas/coca-cola.jpeg';
14 import guaranaImg from './assets/bebidas/guarana.jpeg';
15 import heinekenImg from './assets/bebidas/heineken.jpeg';
16
17 export const enroladinhoMignon = {
18   nome: 'Enroladinho de Mignon',
19   preco: 'R$ 64,90',
20   imagem: enroladinhoMignonImg,
21   descricao: 'Finíssimas fatias de filé mignon enroladas no parmesão',
22 };
23 const casal20 = {
24   nome: 'Casal 20',
25   preco: 'R$ 29,00',
26   imagem: casal20Img,
27   descricao: 'Casal perfeito pão de alho caseiro e linguiça toscana grill',
28 };
29 const burgerPicanha = {
30   nome: 'Burger de Picanha',
31   preco: 'R$ 44,90',
32   imagem: burgerPicanhaImg,
33   descricao:
34     'Burger de Picanha Angus, Queijo Cheddar, Crisp de Cebola e Geléia de Bacon',
35 };
36 const fishChips = {
37   nome: 'Fish and Chips',
38   preco: 'R$ 79,90',
39   imagem: fishChipsImg,
40   descricao:
41     'Isclas de Peixe Empanada na Farinha Panko e Flocos de Milho e Batata Frita Palito',
42 };
```

Um arquivo muito importante para o nosso projeto, e que você também realizou o download, é o **"cardapio.js"**. O código dentro dele representa a importação de imagens e a definição de objetos que representam pratos principais, sobremesas e bebidas em um cardápio.

As imagens são importadas para serem usadas como parte dos objetos, que incluem o nome, preço, imagem e descrição de cada item. Isso é JavaScript, não JSX, porque estamos apenas importando arquivos e definindo objetos de dados, sem manipulação direta do DOM ou renderização de elementos na interface do usuário.

# Construindo a Estrutura da página

Nesse momento, precisamos entender o conceito de Componente no React:

Um componente em React é uma peça reutilizável e independente de interface do usuário que encapsula uma parte específica da funcionalidade e da aparência da sua aplicação. Os componentes podem ser compostos por elementos HTML, outros componentes ou lógica de programação, e são construídos usando a sintaxe JSX do React. Eles ajudam na organização do código, facilitam a manutenção e promovem a reutilização em toda a aplicação. Em resumo, os componentes são os blocos de construção fundamentais para criar interfaces de usuário dinâmicas e escaláveis no React.

Então agora dentro da pasta src, vamos criar um novo arquivo chamado "navegacao.jsx", e criaremos a função **Navegacao()** que será responsável pelos botões de navegação da nossa página.

## Construindo o componente **Navegacao**

O objetivo desse código é criar um **menu de categorias** para o nosso projeto. Vamos entender como ele funciona:

Aqui estamos importando o **React**, necessário para criar nossos componentes.

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

# Construindo a Estrutura da página

1. Estamos criando o componente **Navegacao**, que recebe **duas propriedades** (props):
  - **setCategoriaSelecionada**: uma função que altera a categoria escolhida.
  - **categoriaSelecionada**: o valor da categoria que está ativa no momento.
2. Definimos um array com as categorias que queremos mostrar na tela. Isso deixa o código mais **flexível**: se precisarmos adicionar novas categorias no futuro, basta inserir mais itens na lista.

```
const Navegacao = ({ setCategoriaSelecionada, categoriaSelecionada }) => {  
  const categorias = ["Pratos Principais", "Sobremesas", "Bebidas"];  
  console.log(categoriaSelecionada);  
}
```

3. Renderizando as categorias. Aqui temos alguns pontos importantes:
  - **categorias.map(...)**  
Usamos o método **map** para percorrer o array de categorias e gerar um **<p>** para cada uma delas.
  - **onClick={() => setCategoriaSelecionada(index)}**  
Quando o usuário clica em uma categoria, chamamos a função **setCategoriaSelecionada**, passando o índice da categoria clicada. Isso atualiza o estado no componente pai.
  - **key={index}**  
O React pede uma **key** única para cada item de listas. Aqui usamos o índice.



# Construindo a Estrutura da página

## 4. Classe condicional

- Se a categoria atual (`index`) for igual à categoria selecionada, adicionamos uma classe extra (`categoria--selecionada`) para aplicar um estilo especial.

```
return (  
  <div className="categorias">  
    {categorias.map((categoria, index) => (  
      <p  
        onClick={() => setCategoriaSelecionada(index)}  
        key={index}  
        className={`categoria${  
          index === categoriaSelecionada ? " categoria--selecionada" : ""  
        }}`  
      >  
        {categoria}  
      </p>  
    ))}  
  </div>  
);
```

## 5. Exportando o componente

- Assim podemos importar e usar o `Navegacao` em outros lugares do projeto.

```
export default Navegacao;
```



# Construindo a Estrutura da página

No React, o **export** é uma declaração usada para exportar funções, variáveis, constantes, ou mesmo componentes de um arquivo para que possam ser importados e usados em outros arquivos JavaScript ou JSX. Isso permite que você compartilhe funcionalidades e componentes entre diferentes partes da sua aplicação React.

Quando você exporta algo usando **export**, você pode importá-lo em outros arquivos usando **import**, tornando o código mais modular e reutilizável. Em resumo, o **export** no React permite que você torne suas funcionalidades e componentes disponíveis para uso em outros lugares da sua aplicação.

Vamos utilizar a palavra-chave "default", que indica a exportação principal do arquivo. Isso significa que quando exportamos algo como padrão, ele se torna a principal exportação desse arquivo. Então para utilizar a nossa função/componente precisamos exportar ele primeiro:

```
const Navegacao = ({ setCategoriaSelecionada, categoriaSelecionada }) => {  
  const categorias = ["Pratos Principais", "Sobremesas", "Bebidas"];  
  console.log(categoriaSelecionada);  
  
  return (  
    <div className="categorias">  
      {categorias.map((categoria, index) => (  
        <p  
          onClick={() => setCategoriaSelecionada(index)}  
          key={index}  
          className={`categoria${  
            index === categoriaSelecionada ? " categoria--selecionada" : ""  
          }}`  
        >  
          {categoria}  
        </p>  
      ))}  
    </div>  
  );  
};  
  
export default Navegacao;
```

# Construindo a Estrutura da página

O arquivo App.jsx é frequentemente usado como o componente principal da aplicação. Ele atua como o ponto central onde outros componentes são renderizados e organizados. De forma didática, o App.jsx funciona como o cérebro da aplicação, coordenando e centralizando todos os outros componentes.

Quando você desenvolve uma aplicação React, o arquivo App.jsx é geralmente onde você define a estrutura geral da sua interface de usuário, importa e renderiza outros componentes conforme necessário. Isso permite uma organização clara e facilita a manutenção do código, pois todos os componentes podem ser gerenciados de forma centralizada a partir do App.jsx.

Então dentro do arquivo "App.jsx" vamos importar o componente de Navegação que acabamos de criar.

```
hashtaurante > src > App.jsx > ...  
1  import './App.css';  
2  import hashtauranteImg from './assets/hashtaurante.webp';  
3  import Navegacao from './navegacao';  
4  
5  function App() {  
6    | return <img src={hashtauranteImg} className="capa" />;  
7  }  
8  
9  export default App;
```

# Construindo a Estrutura da página

No React, você pode criar componentes personalizados que funcionam de forma semelhante aos elementos HTML padrão. Isso é possível porque o React permite que você defina componentes como funções ou classes que retornam JSX, uma sintaxe que se assemelha ao HTML.

Quando você define um componente no React, como por exemplo o componente **«Navegacao»** no seu código, você está essencialmente criando uma nova tag que pode ser usada em seu JSX, assim como uma tag HTML padrão como **«div»** ou **«p»**.

Ou seja, você importa o componente **«Navegacao»** e o usa dentro do componente **App** da mesma forma que você usaria uma tag HTML:

```
hashtaurante > src > App.jsx > ...
1  import './App.css';
2  import hashtauranteImg from './assets/hashtaurante.webp';
3  import Navegacao from './navegacao';
4
5  function App() {
6    return <img src={hashtauranteImg} className="capa" />;
7    <Navegacao></Navegacao>;
8  }
9
10 export default App;
```

# Construindo a Estrutura da página

No contexto do código fornecido, a linha `<Navegacao></Navegacao>;` está fora da função `return` do componente `App`, o que significa que não será renderizada corretamente. Para corrigir isso, precisamos envolver a imagem e o componente `<Navegacao>` em um único elemento de nível superior.

Usamos os fragmentos `<></>` (também conhecidos como fragmentos) para criar esse elemento de nível superior sem introduzir um nó extra no DOM.

Quando você precisa agrupar elementos adjacentes em um único elemento de nível superior, mas não deseja adicionar um `<div>` ou qualquer outro elemento HTML extra ao DOM, os fragmentos são uma solução conveniente.

```
hashtaurante > src > App.jsx > ...
1  import './App.css';
2  import hashtauranteImg from './assets/hashtaurante.webp';
3  import Navegacao from './navegacao';
4
5  function App() {
6    return (
7      <>
8        <img src={hashtauranteImg} className="capa" />;
9        <Navegacao></Navegacao>;
10     </>
11   );
12 }
13
14 export default App;
```

# Construindo a Estrutura da página

## Estruturando o **App** com estado e props

### 1. Usando **useState**

```
import { useState } from "react";
```

Aqui estamos criando um **estado** chamado **categoriaSelecionada**.

```
function App() {  
  const [categoriaSelecionada, setCategoriaSelecionada] = useState(0);
```

- Ele começa no valor **0** (ou seja, a primeira categoria da lista).
- Para alterar esse valor, usamos a função **setCategoriaSelecionada**.

### 👉 O que é estado (**useState**)?

No React, o **estado** é como uma memória que guarda informações que podem mudar durante a execução da aplicação. Sempre que o estado muda, o React **re-renderiza** o componente, mostrando o valor atualizado na tela.

# Construindo a Estrutura da página

## 2. Estrutura inicial do App

- Criamos uma **div container** para agrupar todo o conteúdo.
- Dentro dela, temos a **div banner**, que exibe uma imagem (**ImgRestaurante**).

Essa parte é mais visual e serve como cabeçalho do nosso projeto.

## 3. Usando o componente Navegacao com props

Passamos duas informações importantes para o **Navegacao**:

- **categoriaSelecionada**: indica qual categoria está ativa.
- **setCategoriaSelecionada**: permite que o componente **Navegacao** altere esse valor quando o usuário clicar em uma categoria.

⚡ Dessa forma, o **App controla o estado** e o **Navegacao** apenas **usa e modifica esse estado**.

```
import { useState } from "react";
import "../App.css";
import ImgRestaurante from "../assets/hashtaurante.webp";

import Navegacao from "../components/Navegacao";

function App() {
  const [categoriaSelecionada, setCategoriaSelecionada] = useState(0);

  return (
    <>
      <div className="container">
        <div className="banner">
          <img src={ImgRestaurante} alt="Imagem do Hashtaurante" />
        </div>

        <Navegacao
          categoriaSelecionada={categoriaSelecionada}
          setCategoriaSelecionada={setCategoriaSelecionada}
        />
      </div>
    </>
  );
}

export default App;
```



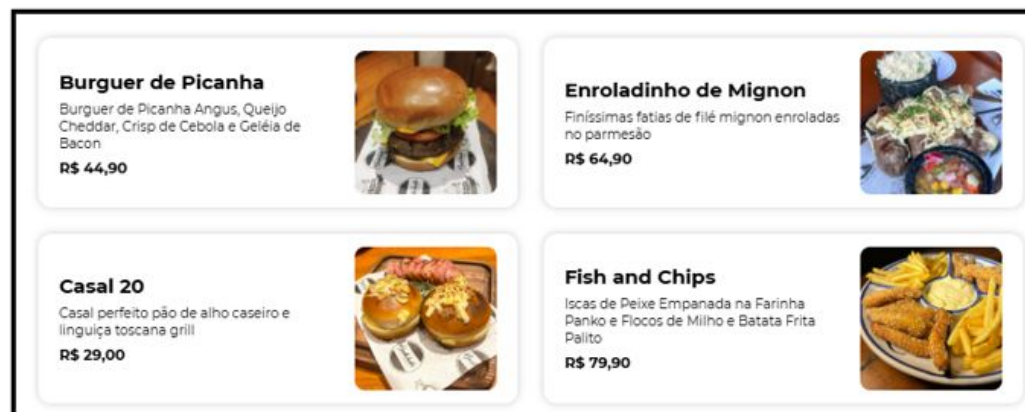
# Construindo a Estrutura da página

Feito isso, vamos criar mais dois componentes dentro da pasta **src**, chamado **"Card.jsx"** e **"Cards.jsx"** que serão responsável por montar um card de um item do cardápio e a lista de itens do cardápio.

Exemplo renderização de Card:



Exemplo renderização de Cards:



# Construindo a Estrutura da página

O conceito por trás desse componente Card.jsx é criar um item genérico que pode ser usado para exibir informações comuns de vários objetos que compartilham a mesma estrutura de propriedades, como nome, preço e imagem.

Esse componente é projetado para ser reutilizável e flexível, permitindo que você passe diferentes objetos como propriedades e renderize suas informações de forma consistente e padronizada em sua aplicação React. Isso promove a modularidade e a reutilização de código, uma vez que você pode usar o mesmo componente para exibir diferentes tipos de dados sem precisar reescrever o código repetidamente.

Para criar o modelo do nosso card, vamos usar o primeiro objeto dos itens do cardápio, que está no arquivo cardapio.js, como referência. Vamos aplicar um export temporário para obter os dados desse primeiro item, que é o prato "Enroladinho de Mignon". Isso nos permitirá acessar e utilizar os dados desse item em outros lugares do nosso código até que tenhamos um sistema mais robusto de gerenciamento de dados.

```
17 export const enroladinhoMignon = {  
18   nome: "Enroladinho de Mignon",  
19   preco: "R$ 64,90",  
20   imagem: enroladinhoMignonImg,  
21   descricao: "Finíssimas fatias de filé mignon enroladas no parmesão",  
22 };
```



# Construindo a Estrutura da página

Exemplo de estrutura para renderizar o componente:

```
import { enroladinhoMignon } from './cardapio';

function Card() {
  return (
    <div className="container-item-cardapio">
      <div className="informacoes-item-cardapio">
        <h2>{enroladinhoMignon.nome}</h2>
        <p>{enroladinhoMignon.descricao}</p>
        <p>{enroladinhoMignon.preco}</p>
      </div>
      <img
        src={enroladinhoMignon.imagem}
        alt={enroladinhoMignon.nome}
        className="imagem-item-cardapio"
      />
    </div>
  );
}

export default Card;
```

# Construindo a Estrutura da página

No código, estamos importando os dados do prato "Enroladinho de Mignon" do arquivo `cardapio.js` usando a declaração **`import { enroladinhoMignon } from './cardapio';`**. Em seguida, estamos definindo um componente chamado `ItemCardapio` que renderiza as informações desse prato, como nome, descrição, preço e imagem.

Dentro do componente `ItemCardapio`, usamos JSX para estruturar o layout do cardápio. O nome, descrição e preço do prato são renderizados dentro de elementos **`<h2>`**, **`<p>`** e **`<p>`**, respectivamente, utilizando as propriedades do objeto `enroladinhoMignon`.

A imagem do prato é renderizada usando a propriedade **`imagem`** do objeto `enroladinhoMignon` dentro de um elemento **`<img>`**. As classes CSS são aplicadas para estilizar o layout do cardápio. Por fim, exportamos o componente `ItemCardapio` para que ele possa ser usado em outros lugares da nossa aplicação. Essencialmente, este componente representa um item genérico do cardápio que pode ser reutilizado para exibir informações de outros pratos de maneira consistente.

# Construindo a Estrutura da página

Para visualizarmos o card que criamos, vamos importar esse componente para o arquivo App.jsx e adicionar a estrutura da página como elemento.

```
function App() {  
  const [categoriaSelecionada, setCategoriaSelecionada] = useState(0);  
  
  return (  
    <>  
      <div className="container">  
        <div className="banner">  
          <img src={ImgRestaurante} alt="Imagem do Hashtaurante" />  
        </div>  
  
        <Navegacao  
          categoriaSelecionada={categoriaSelecionada}  
          setCategoriaSelecionada={setCategoriaSelecionada}  
        />  
  
        <Card />  
      </div>  
    </>  
  );  
}  
  
export default App;
```



Pratos Principais

Sobremesas

Bebidas

Pratos Principais

## Enroladinho de Mignon

Finíssimas fatias de filé mignon enroladas  
no parmesão

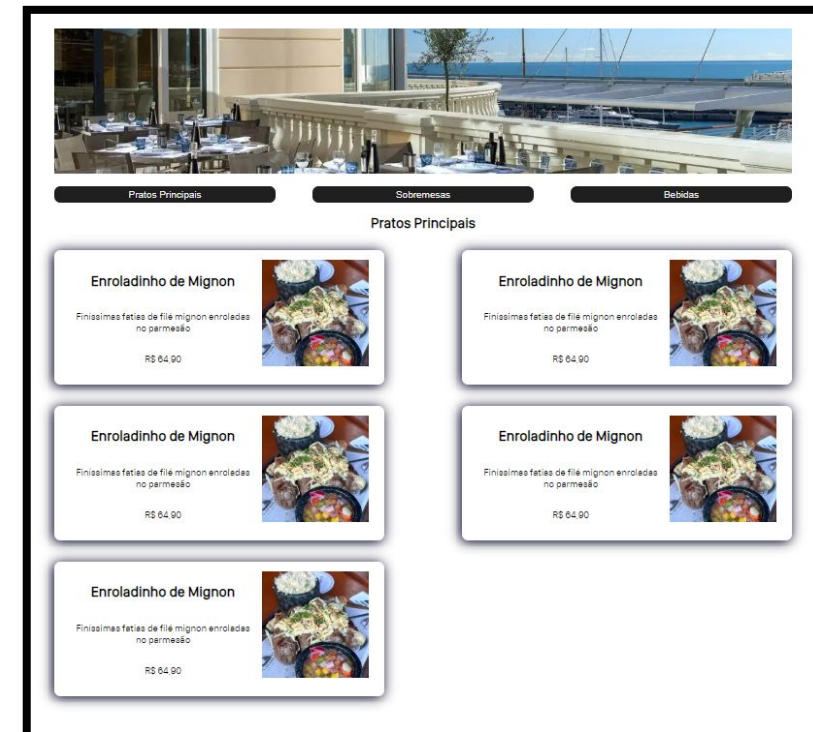
R\$ 64,90



# Construindo a Estrutura da página

O React tem uma funcionalidade poderosa chamada componentização, que nos permite criar componentes reutilizáveis. Se desejamos criar cards semelhantes, o React nos oferece esse recurso. Para visualizar essa geração de componentes, você verá vários cards sendo repetidos, mesmo que sejam do mesmo item do cardápio. Isso ocorre porque podemos reutilizar o mesmo componente para renderizar informações diferentes de forma consistente e padronizada em nossa aplicação.

```
function App() {  
  const [categoriaSelecionada, setCategoriaSelecionada] = useState(0);  
  
  return (  
    <>  
      <div className="container">  
        <div className="banner">  
          <img src={ImgRestaurante} alt="Imagem do Hashtaurante" />  
        </div>  
  
        <Navegacao  
          categoriaSelecionada={categoriaSelecionada}  
          setCategoriaSelecionada={setCategoriaSelecionada}  
        />  
  
        <Card />  
        <Card />  
        <Card />  
        <Card />  
        <Card />  
      </div>  
    </>  
  );  
}  
  
export default App;
```



# Construindo a Estrutura da página

Agora vamos fazer algumas alterações nos nossos componentes para transformá-los em cards de item genérico e poder renderizar a lista de cards com apenas um componente sendo renderizado no App, o componente "Cards". Para isso, vamos usar a seguinte estrutura no componente "Card":

Os parâmetros são colocados entre chaves `{}` na assinatura da função (`{ nome, descricao, preco, imagem }`). Isso é chamado de "destructuring" em JavaScript. Essa sintaxe permite extrair propriedades de objetos de forma mais conveniente.

Neste caso, a função **Card** espera receber um objeto como parâmetro, e com a sintaxe de destructuring, estamos extraindo diretamente as propriedades **nome**, **descricao**, **preco** e **imagem** desse objeto. Isso significa que, ao chamar essa função, podemos passar um objeto com essas propriedades e ele será automaticamente desestruturado para as variáveis correspondentes dentro da função.

```
const Card = ({ itemCategoria }) => {
  const { nome, preco, imagem, descricao } = itemCategoria;

  return (
    <div className="card">
      <div className="card__textos">
        <h2 className="card__titulo">{nome}</h2>

        <p>{descricao}</p>

        <p className="card__preco">{preco}</p>
      </div>

      <div className="card__img">
        <img src={imagem} alt="Imagem do Produto" />
      </div>
    </div>
  );
};

export default Card;
```

# Construindo a Estrutura da página

Agora a ideia é mostrar como o **Card** organiza os dados do item de categoria na tela:

## Construindo o layout do Card

Depois de desestruturar as propriedades (**nome**, **preco**, **imagem**, **descricao**), o próximo passo é organizar esses dados no **retorno JSX**.

### 1. Estrutura principal

- Toda a informação do produto fica dentro dessa div com a classe **card**.
- Essa classe será útil no CSS para aplicar estilo e organizar o layout.

### 2. Área de textos

Criamos uma **div só para os textos** do card.

Usamos classes específicas (**card\_\_titulo**, **card\_\_preco**) para facilitar a estilização.

O conteúdo é preenchido dinamicamente com as props:

- **{nome}** → título do produto.
- **{descricao}** → pequena descrição.
- **{preco}** → valor formatado.



# Construindo a Estrutura da página

## 3. Área da imagem

- Temos a **imagem do produto**, recebida via prop `imagem`.
- O atributo `alt` garante acessibilidade e boas práticas de SEO.
- A classe `card__img` permite ajustar o tamanho e o posicionamento no CSS.
- 

O componente `Card` é responsável por **apresentar os dados de um item da categoria** em formato visual:

- Nome do produto
- Descrição
- Preço
- Imagem

Ou seja, ele transforma as **informações recebidas por props** em uma interface bonita e organizada.

# Construindo a Estrutura da página

Componente **Cards** — passo a passo de construção

## Importações

- Importamos o componente **Card** (cada cartão) e as 3 listas (**pratosPrincipais**, **sobremesas**, **bebidas**) do arquivo **cardapio.js**.

```
import Card from "../Card";  
import { pratosPrincipais, sobremesas, bebidas } from "../assets/cardapio.js";
```

## Array **itensCategoria**

- Criamos um array que contém as 3 listas. A posição no array corresponde ao índice usado pelo **Navegacao/App**: 0 → **pratosPrincipais**, 1 → **sobremesas**, 2 → **bebidas**.
- Isso facilita selecionar a lista correta via **itensCategoria[categoriaSelecionada]**.

```
const Cards = ({ categoriaSelecionada }) => {  
  const itensCategoria = [pratosPrincipais, sobremesas, bebidas];
```



# Construindo a Estrutura da página

## Proteção contra **undefined**

- `const listaAtual = itensCategoria[categoriaSelecionada] || []`; garante que, se por algum motivo `categoriaSelecionada` estiver fora do intervalo, não quebramos a aplicação — mostramos uma lista vazia.

## Render com **.map()**

- Para cada item em `listaAtual` criamos um `<Card>` e passamos o objeto `itemCategoria` via prop.
- **Key**: ideal usar uma `key` estável (ex.: `itemCategoria.id`). Use `index` só se não houver outra opção.

## Fallback visual

- Se a lista estiver vazia, mostramos uma mensagem amigável ao usuário.

```
return (  
  <div className="cards">  
    {itensCategoria[categoriaSelecionada].map((itemCategoria, index) => (  
      <Card key={index} itemCategoria={itemCategoria} />  
    ))}  
  </div>  
);
```

# Construindo a Estrutura da página

## Integração no App

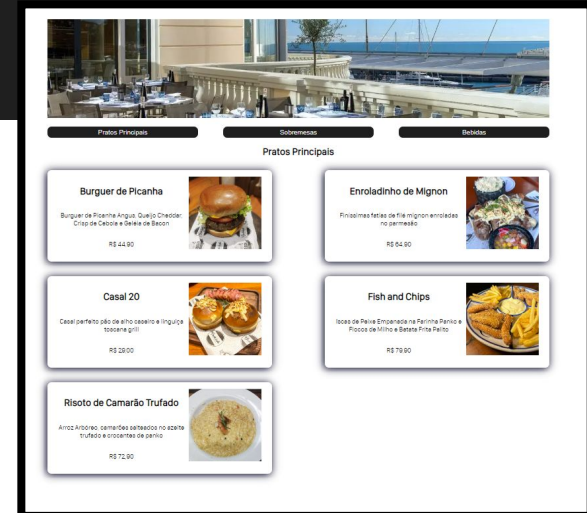
**Cards** só precisa saber o índice (número) da categoria ativa. Com isso, ele escolhe qual array renderizar e mostra os **Cards** correspondentes.

## Fluxo geral (resumido)

- Usuário clica em uma categoria em **Navegacao** → **setCategoriaSelecionada(index)** é chamado.
- **App** atualiza **categoriaSelecionada** (estado) → React re-renderiza **App**.
- **Cards** recebe o novo **categoriaSelecionada** via props → escolhe a lista correta e renderiza os **Cards**.
- **Card** recebe **itemCategoria** e mostra nome, descrição, preço e imagem.

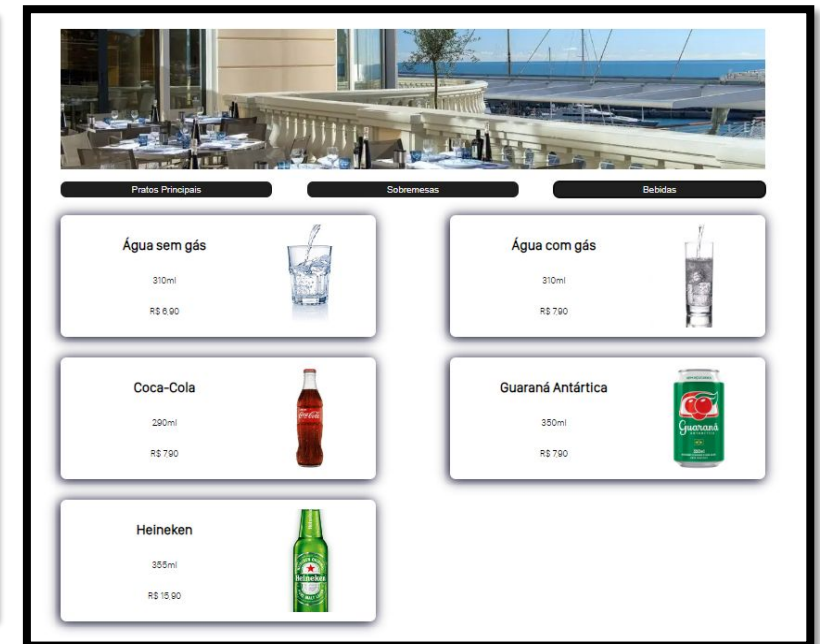
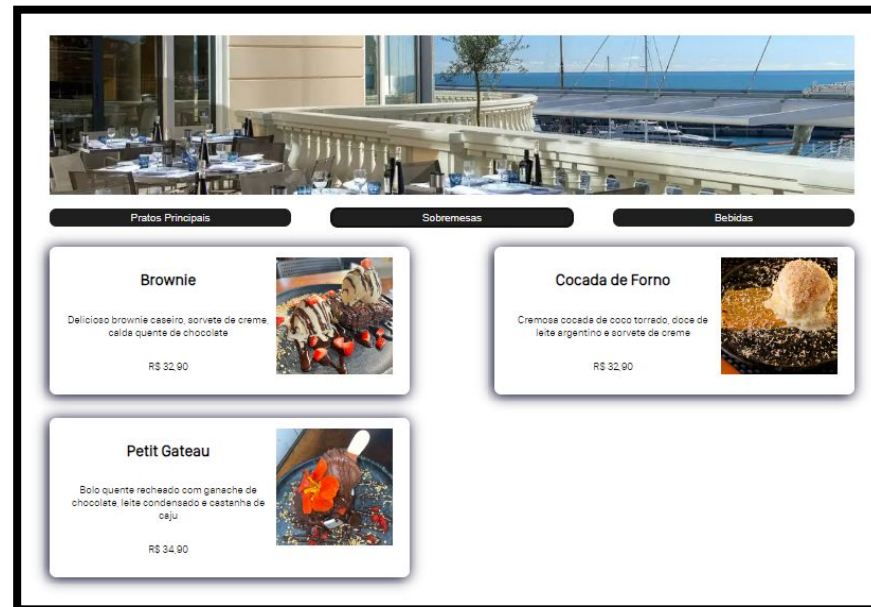
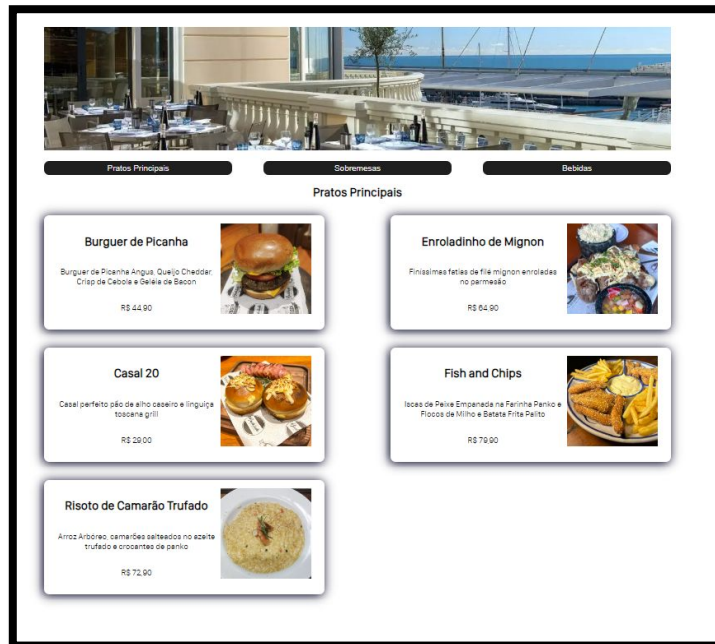
```
function App() {
  const [categoriaSelecionada, setCategoriaSelecionada] = useState(0);

  return (
    <>
      <div className="container">
        <div className="banner">
          <img src={ImgRestaurante} alt="Imagem do Hashtaurante" />
        </div>
        <Navegacao
          categoriaSelecionada={categoriaSelecionada}
          setCategoriaSelecionada={setCategoriaSelecionada}
        />
        <Cards categoriaSelecionada={categoriaSelecionada} />
      </div>
    </>
  );
}
```



# Construindo a Estrutura da página

Parabéns! Com todas as funcionalidades e componentes implementados, nosso Cardápio Digital agora está completo. Com dinamismo e uma ótima experiência para o usuário, estamos prontos para compartilhar nossas criações com o mundo.



Parte Bônus

Vídeos

Complementares

# Vídeos Complementares



## Como sair do ZERO no React em Apenas UMA AULA

Hashtag Programação • 6.8K views • 6 months ago

[Como sair do ZERO no React em Apenas UMA AULA](#)



## Tudo que você precisa saber de JavaScript antes de aprender React

Hashtag Programação • 3.7K views • 7 months ago

[Tudo que você precisa saber de JavaScript antes de aprender React](#)

Ainda não segue a gente no Instagram e nem é inscrito no  
nosso canal do Youtube? Então corre lá!



[@hashtagprogramacao](https://www.instagram.com/hashtagprogramacao)



[youtube.com/@HashtagProgramacao](https://youtube.com/@HashtagProgramacao)

[youtube.com/@DevHashtag](https://youtube.com/@DevHashtag)

