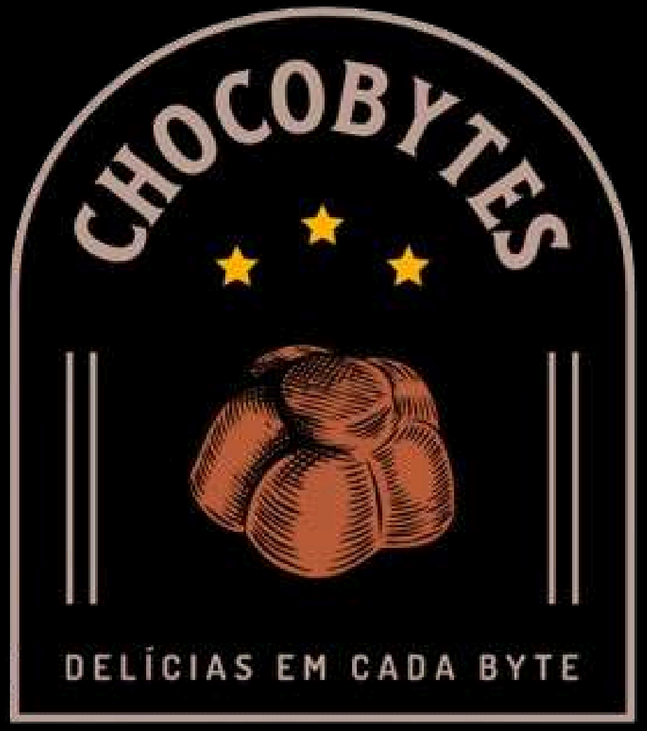


ChocoBytes: Desenvolvendo uma loja de chocolates com React JS

NATHALIA MACEDO
DESENVOLVEDORA FULL STACK

Descrição do Projeto

Vamos criar uma loja de chocolates onde teremos:
Página principal, onde teremos um banner de chamada para ação, e alguns produtos renderizados na tela!
Catálogo de vendas: Cartões com foto, preço e informações do chocolate.
Também temos o carrinho, onde é possível adicionar, editar ou remover itens.
Guardaremos todas essas informações no Local Storage, então ao sair e entrar novamente na página, as informações seguem presentes!
Também teremos uma aba de favoritos e sobreNos.

Descrição do Projeto

Atenção: Visando conseguir finalizar o projeto, os arquivos CSS serão disponibilizados para todos vocês! Vamos focar em fazer as funcionalidades, ao invés de estilizar!

Por gentileza siga tudo o que está aqui a vera! Se está comletra maiúscula, crie com letra maiúscula! Se tem a classe X, crie a classe X.

Como vamos pegar o estilo pronto, é importante seguir a risca tudo o que está descrito aqui!

Começando

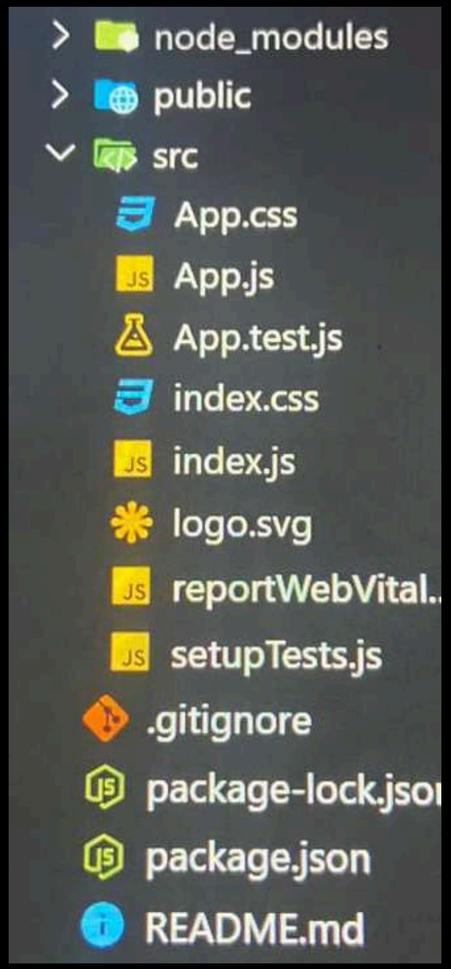
A primeira coisa que faremos é criar o nosso projeto, utilizando o comando npx create-react-app chocobytes (sem letras maiúsculas no nome do projeto, tudo minúsculo!

Crie uma pasta FORA DA REDE (Ao usar esse comando, é baixado uma pasta chamada node modules, ela possui mais de 300 bibliotecas dentro, é extremamente pesada. Se fizer na rede, ainda funciona, mas vai levar 15min só pra criar o projeto), abra no VS code, abra o terminal e dê o comando:

```
npx create-react-app chocobytes
```

Começando

Depois de criado o projeto, essa será a estrutura:



Dentro da pasta src, vamos criar uma pasta app, e colocar os arquivos:
App.css e App.js dentro dessa pasta

Também apagaremos os arquivos App.test.js, logo.svg,reportWebVitals, SetupTests.

Ficará apenas index.js, index.css, e a pasta app, com os dois arquivos dentro!

Editando arquivos

Dentro do arquivo App.css e App.js, vamos apagar tudo.

Em index.js, apagaremos as linhas:

```
import App from './App';
import reportWebVitals from './reportWebVitals';
```

```
reportWebVitals();
```

Dentro da pasta src

Dentro da pasta src, fora da pasta app, criaremos mais três pastas: Assets, Components e Pages!

Em Assets, colocaremos todos os arquivos de imagem que serão utilizados!

Em Pages, vamos criar as páginas, exemplo: Home, catálogo, etc...

Em components, vamos criar nossos componentes!

Mas o que seria componentes?

Componentes

No React, utilizamos HTML, CSS e Javascript juntos! Criamos nossa própria tag. Imaginem a tag a (âncora). Ela possui estrutura (<a>), tem estilo (vem subinhada e azul) e funcionalidade (o atributo href faz com que leve para outra página ou seção, o atributo target define em qual aba será aberto).

É exatamente isso que criaremos! Vamos ter uma tag (uma estrutura html), vamos atribuir a ela um estilo (css) e terá funcionalidades (javascript).

Exemplo: Teremos cards de chocolate. Vamos criar o modelo do card, e trocar apenas as informações! Assim facilitamos o processo de criação do site!

Diferença entre componentes e páginas

Criamos um componente quando queremos criar o modelo de algo que será chamado várias vezes. Como assim?

Criei o modelo card, quero chamar 50 cards. Toda aquela estrutura, estilo e funcionalidade virá apenas em escrever o nome da tag!

As páginas, são onde ficam os componentes!

Voltando pro código

Nosso projeto não possui um back end, por isso, vamos criar os dados que serão renderizados (nomes, preços e descrição dos produtos)!

A primeira coisa que precisamos fazer é pegar as imagens que usaremos e colocar dentro da pasta Assets, todas elas.

Criaremos uma pasta chamada Context dentro da src, e dentro do Context, um arquivo chamado produtos.js

Após colocar todos os arquivos na pasta assets e criar o arquivo produtos.js dentro da pasta, faremos as importações das imagens!

Contexto

No react, nós fazemos um pouco diferente! Temos os dados, nós meio que embrulhamos a estrutura nos dados!

Mas como assim? Exemplo: Vamos ter vários cards de produtos. Eu preciso envolver esse card nos dados:

```
<dados>
<Card/>
</dados>
```

O que vamos criar primeiro é nosso provedor!

Contexto

Explicando o código que vocês receberam:

```
//importando as imagens e hooks que vamos utilizar
import React, { createContext, useContext, useState, useEffect } from 'react';
import cakeRed from '../Assets/cakeRed.png';
import cupcake from '../Assets/cupcake.png';
import cheesecake from '../Assets/cheesecake.png';
import TortaLemon from '../Assets/tortaLemon.png';
import buffet from '../Assets/Buffet.png';
import pavlova from '../Assets/pavlova.png';
import tiramisu from '../Assets/tiramisu.png';
import Maracuja from '../Assets/maracuja.png';
import Limao from '../Assets/Limão.png';
import cakeDourado from '../Assets/cakeDourado.png';
import cakeStrawberry from '../Assets/cakeStranberry.png';
```

Contexto

Hooks são funções do React onde cada uma faz alguma coisa! createContext é para criar o contexto, useContext é para utilizar o contexto, useState é para criar variáveis, e useEffetc é pra lidar com efeitos colaterais!

```
//importando as imagens e hooks que vamos utilizar
import React, { createContext, useContext, useState, useEffect } from 'react';
import cakeRed from '../Assets/cakeRed.png';
import cupcake from '../Assets/cupcake.png';
import cheesecake from '../Assets/cheesecake.png';
import TortaLemon from '../Assets/tortaLemon.png';
import buffet from '../Assets/Buffet.png';
import pavlova from '../Assets/pavlova.png';
import tiramisu from '../Assets/tiramisu.png';
import Maracuja from '../Assets/maracuja.png';
import Limao from '../Assets/Limão.png';
import cakeDourado from '../Assets/cakeDourado.png';
import cakeStrawberry from '../Assets/cakeStranberry.png';
```

Contexto

```
//criando o contexto
const ChocolatesContext = createContext();
//usando o contexto e o tornando exportável
export const useChocolates = () => useContext(ChocolatesContext);
```

Abaixo disso (grande demais para mostrar em print), temos o provedor do contexto. Então, resumidamente, ele que vai envolver as estruturas onde os dados serão utilizados!

Feito isso, vamos começar a programar!

Contexto

A primeira coisa que faremos é criar um espaço no localStorage para guardar duas listas: Carrinho e favoritos.

Como o que vai ser guardado são dados, é aqui que vamos definir as funções, e por ser um contexto exportável, podemos chamar essas funções em partes específicas do código, como em outro arquivo por exemplo!

Para isso, primeiro, precisamos das variáveis, e aqui no React elas funcionam de forma um pouco diferente!

Variáveis no React - Pasta Context

No React, não podemos definir variáveis da forma como estamos acostumados. Inicialmente, precisamos de duas coisas, o nome da variável, e uma função que altere seu valor! Fica assim:

```
//carrinho é o nome da variável
//setCarrinho é a função que atualiz seu valor
//useState é o hook do react onde colocamos o valor inicial da variável
//Nesse caso (EXEMPLO) ela começa como uma string vazia
  const [carrinho, setCarrinho] = useState('')
```

Mas no nosso caso, temos duas situações: Estamos usando o localStorage, ou seja, armazenamento local, é um local onde podemos guardar informações na nossa máquina, mesmo se fechar e abrir novamente, elas seguem ali, presentes!

Variável carrinho

Então, imaginem, pode ser que precise jogar uma lista vazia no localStorage, caso seja o primeiro acesso do usuário ao site, mas preciso também que, caso não seja o primeiro acesso, ele recupere a lista do localStorage!

```
const [carrinho, setCarrinho] = useState(() => {
    //usando o getItem para pegar o tem carrinho
    //O localStorage guarda as informações como string
    const localData = localStorage.getItem('carrinho');
    //criando um ternário
    //Será retornado a resposta da seguinte pergunta:
    //Existe alguma coisa no localStorage?
    //Se sim, os dados que estão lá, estão como uma lista de strings!
    //Precisamos dele no formato objeto javascript
    //para poder acessar os valores das chaves.
    //Se não tiver nada no localStorage, o estado inicial
    //da variável carrinho, vai ser uma lista vazia
    return localData ? JSON.parse(localData) : [];
  });
```

Criando a variável favoritos

Faremos a mesma coisa com a variável favoritos! Tudo igual, muda apenas que agora a variável não é mais carrinho!

```
//Faremos a exata mesma coisa com a variável favoritos
  //Criando variável e função que altera seu estado, tentando pegar itens no localStorage
  //Se existir, vai transformar em lista de objetos, tornando os dados acessíveis.
  //Se não, nossa variável será uma lista vazia
  const [favoritos, setFavoritos] = useState(() => {
    const localFavoritos = localStorage.getItem('favoritos');
    return localFavoritos ? JSON.parse(localFavoritos) : [];
  });

  useEffect(() => {
    localStorage.setItem('favoritos', JSON.stringify(favoritos));
  }, [favoritos]);
```

Adicionando Favoritos

```
//Criando a função que adiciona um chocolate a lista de favoritos
 //temos o chocolate como parâmetro, não da pra adicionar
 //um chocolate favorito se não houver o chocolate
 const adicionarFavorito = (chocolate) => {
   //chamando a função que altera o valor da variavel favoritos
   //prevFavoritos é um nome fictício que damos, para todos os outros
   //chocolates que já estão na lista!
   //Essas reticências é o que chamamos de spread operator!
   //Resumidamente, vamos deixar todos os chocolates que já estão na lista, mas adicionar um!
   setFavoritos((prevFavoritos) => [...prevFavoritos, chocolate]);
 };
```

Removendo favoritos

```
//Para a função de remover, não recebemos todos os dados do chocolate, apenas
 //o id. Os nomes dos doces podem ser iguais, mas o id jamais será
 const removerFavorito = (id) => {
   //usando novamente a função de alterar o valor dos favoritos,
   //chamamos todo mundo que já esta na lista de favoritos de prevFavoritos,
   //E vamos alterar essa lista, filtrando ela:
   //Para cada item dentro dessa lista, me retorne uma nova lista que tenha
   //todos os itens, exceto aquele com o id
   // que foi passado como parâmetro!
   //Essa nova lista vai ser guardada dentro da variável favoritos!
   setFavoritos((prevFavoritos) => prevFavoritos.filter((item) => item.id !== id));
 };
```

Adicionando ao Carrinho

É assim que nosso card de carrinho vai ficar. Ao clicar no X no canto superior direito, ele remove o item do carrinho, e também dá pra aumentar a quantidade de itens que quer comprar!

Adicionando ao Carrinho

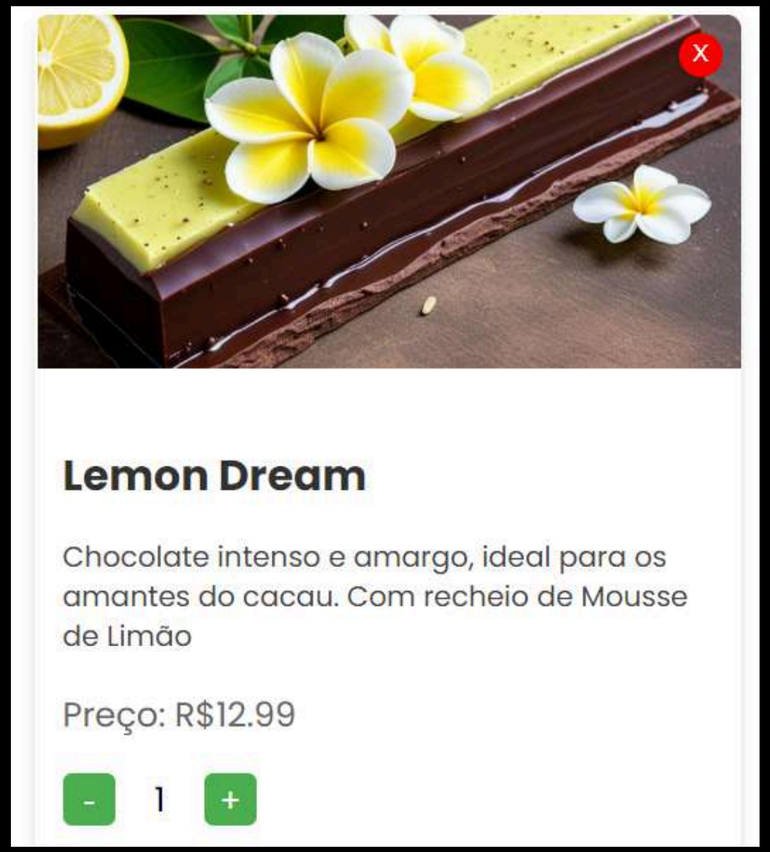
```
//criando a função que vai adicionar um chocolate ao carrinho de compras!
  //obviamente temos como parâmetro o chocolate!
  const adicionarAoCarrinho = (chocolate) => {
    //criando um novo item que vai pro carrinho, onde esse item tem
    //todas as informações, e a quantidade inicial
    const novoItem = { ...chocolate, quantidade: 1 }; // Definindo a quantidade inicial como 1
    //atualizando o que esta guardado dentro do carrinho
    setCarrinho((prevCarrinho) => {
      //criando uma nova lista, onde pega toda a lista antiga (prevCarrinho)
      //e adiciona a nova lista!
      const novoCarrinho = [...prevCarrinho, novoItem];
      //jogando nosso novo carrinho dentro do localStorage
      //A mesma chave carrinho, então ele vai reescrever o que está
      //na chave.
      localStorage.setItem('carrinho', JSON.stringify(novoCarrinho)); // Atualizando o localStorage
      //retornamos essa nova lista
      return novoCarrinho;
    });
```

Remover do Carrinho

```
//Para remover o item do carrinho, mesmo processo de remover favoritos,
//A unica diferença é que agora é pra remover do carrinho, mas o código é
//praticamente o mesmo
  const removerDoCarrinho = (id) => {
    setCarrinho((prevCarrinho) => prevCarrinho.filter((item) => item.id !== id));
  };
```

Mas, vamos dar uma olhada novamente em como vai ficar nosso card renderizado?

Alterando a quantidade de itens



Precisamos que, ao clicar no botão verde de +, ele adicione mais um ao valor que esta no meio, e também que altere o preço do produto, afinal, dois custam o dobro!

Incrementando a quantidade de itens

```
//criando a funçõ que incrementa a quantidade
  //Pegamos o produto pelo id
  const incrementarQuantidade = (id) => {
    //Alterando a lista carrinho, afinal, se aumntou a quantidade
    //ele quer mais de um!
    setCarrinho((prevCarrinho) =>
      //Percorremos toda nossa lista atual de carrinho, procurando o item
    //Que tem o id passado na chamada da função
      prevCarrinho.map((item) =>
        //Ao percorrer a lista, se o id do item for igual ao id passado como
      //argumento, retorne esse item, mas mexa na chave quantidade, vamos adicionar mais um!
      //Caso não encontre, apenas retorne o item
        item.id === id ? { ...item, quantidade: item.quantidade + 1 } : item
      )
    );
  };
```

Decrementando a quantidade de itens

```
//função que irá decrementar a quantidade
 //recebemos o id do produto que queremos diminuir a quantidade
 const decrementarQuantidade = (id) => {
   //vamos alterar a lista carrinho
   setCarrinho((prevCarrinho) =>
     //percorrendo todos os itens da lista atual
     prevCarrinho.map((item) =>
       //procurando pelo id que foi passado, e vendo se a quantidade é maior
     // do que 1. Se não for, não podemos diminuir, certo?
       item.id === id && item.quantidade > 1
       //Se sim, vamos pegar o item, e alterar a propriedade quantidade
       //para menos 1
         ? { ...item, quantidade: item.quantidade - 1 }
         //Se não, apenas retornamos nosso item como está!
         : item
     )
   );
 };
```

Terminando nosso provedor de dados

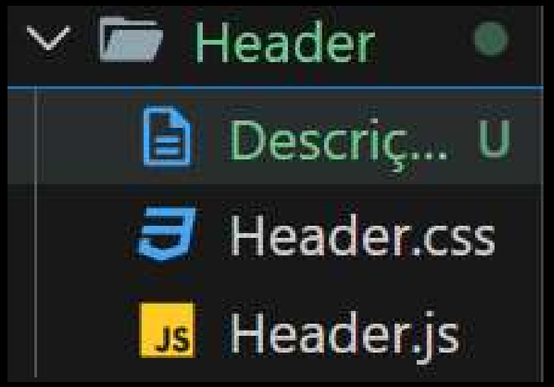
```
 //Aqui, estamos determinando o que nosso provedor de dados vai levar para as
  //outras páginas! Todas as funções e listas que criamos até então!
  //Quando chamarem pelo provedor de dados, tudo isso será retornado!
  return (
    <ChocolatesContext.Provider value={{
      chocolates,
      adicionarFavorito,
      removerFavorito,
      setChocolates,
      favoritos,
      setFavoritos,
      carrinho,
      setCarrinho,
      adicionarAoCarrinho,
      removerDoCarrinho,
      incrementarQuantidade,
      decrementarQuantidade
    }}>
      {/* Lembrando que iremos envolver alguém com esses dados, por isso estamos
      passando a propriedade children, o provedor sempre terá um filho, que é a estrutura onde os dados irão aparecer  */}
      {children}
    </ChocolatesContext.Provider>
  );
};
```

Criando nossa pasta Header dentro de components

O header é renderizado dentro de todas as páginas, já que é graças a ele que conseguimos navegar entre as páginas!

Não precisamos envolver esse componente nos dados, já que sua principal finalidade é a navegação entre páginas!

Crie uma pasta Header dentro da pasta components, e dentro dessa pasta, criaremos dois arquivos: Header.js e Header.css



Antes de qualquer coisa...

Dentro do Header vamos usar componentes que são parte de uma biblioteca chamada react-router-dom.

No terminal, percorra as pastas até a pasta raiz do projeto, e dê o comando:

```
npm install react-router-dom
```

```
npm install react-icons
```

Dentro do Header.js

```
//Importando o react e o useState, por que teremos variáveis aqui também!
//Importando a tag Link do react-router-dom
//Importando os ícones do react que usaremos
//Importando a imagem da logo e o css
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import { FaShoppingCart, FaHeart,FaBars, FaTimes,FaBuilding,FaHome } from 'react-icons/fa';
import logo from '../../Assets/Artboard_2-removebg-preview.png';
import './Header.css';
```

A tag Link é como a tag <a> do React, ela serve para levar para outra página!

Criando o Header

```
//criando nosso componente Header
function Header() {
  //Vamos criar o nosso botão hamburger,
  //e ele altera entre o hamburguer e o X
  //Para isso, precisamos controlar seu estado!
  //Criamos a variável e a função que altera seu
  //estado!
  const [menuOpen, setMenuOpen] = useState(false);

  //Criamos uma função que será chamada ao clicar no botão
  //Ela altera o estado da variável menuOpen
  //O valor dela vai ser a resposta para a pergunta:
  //O valor guardado lá dentro é false?
  //Se sim, true, e aí o valor será atualizado para true!
  //Se não, false, e o valor fica false!
  const toggleMenu = () => {
    setMenuOpen(!menuOpen);
  };
```

Criando o Header

```
return (
    //usando header como elemento pai
    <header>
      {/*Adicionando imagem */}
      <img src={logo} alt="logo" className="logo" />
    {/*A classe da barra de navegação altera */}
    {/*Por que ela pode aparecer ou sumir*/}
    {/*Se o valor de menuOpen for true, a classe é active */}
    {/*Se não, ela permanece sem classe! Permanece aparecendo */}
      <nav className={menuOpen ? 'active' : ''}>
```

Criando o Header

```
  <ul>
    <li>
      {/*Chamamndo os ícones */}
     <FaHome/> <Link className='link' to={'/'}>Home</Link>
      </li>
    <li>
      <FaBars />
      <Link className='link' to='/catalogo'>Catálogo</Link>
    </li>
    <li>
      <FaHeart /> <Link className='link' to='/favoritos'>Favoritos</Link>
    </li>
    <li>
      <FaShoppingCart /> <Link className='link' to='/carrinho'>Carrinho</Link>
    </li>
    <li>
      <FaBuilding /> <Link className='link' to='/sobreNos'>Sobre Nós</Link>
    </li>
  </ul>
</nav>
```

Criando o Header

```
        <div className="menu-icon" onClick={toggleMenu}>
          {menuOpen ? <FaTimes /> : <FaBars />}
        </div>

      </header>
    );
}

export default Header;
```

Nesse caso, dependendo do valor da variável menuOpen, teremos ícones diferentes! Esse é o ícone que aparece em dispositivos menores. Se o valor for true, o hamburguer aparece, se o usuário clicar, abre o menu, e então o ícone de X aparece!

Criando o home

Precisamos testar, ver como ficou nosso Header. O componente não pode ser renderizado sem uma página! Por isso, criaremos nossa página Home, apenas para testar nosso header por enquanto!

Dentro de Pages, vamos criar uma pasta Home, e dentro dela, dois arquivos: Home.js e Home.css

Dentro de Home.js, vamos colocar o seguinte código:
Importando nosso componente Header

```
import Header from '../../Components/Header/Header'
```

Criando o home

Criar nossa página Home:

```
//criando nossa página Home
function Home(){
    //chamando o header pra aparecer
    //dentro da página
return(
    <Header/>
)
}
//Tornando o Home exportável
//Para ser chamado em outras partes
export default Home
```

Criando rotas e chamando a Home

Para definirmos as rotas, vamos voltar no nosso arquivo App.js, e apagar tudo o que tem lá, absolutamente tudo! Instalamos a biblioteca antes, então agora basta chamar os componentes!

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
```

Também vamos chamar nossa página Home:

```
import Home from '../Pages/Home/Home';
```

Criando rotas

Para criarmos as rotas, usaremos o Routes, Router, e o Route!

Router é um componente que encapsula toda a aplicação React, fornecendo o contexto de roteamento necessário para que a navegação funcione corretamente. Ele utiliza a API de histórico do navegador para manter a interface de usuário sincronizada com a URL.

Routes é um contêiner que agrupa todos os componentes Route. Ele é responsável por renderizar o componente Route correspondente com base na URL atual.

Route define uma rota específica na sua aplicação. Cada Route possui uma path (caminho) e um element (elemento) que será renderizado quando a URL corresponder ao caminho definido.

Criando rotas

Para criarmos as rotas, usaremos o Routes, Router, e o Route!

Para colocar nosso código pra rodar, vamos entrar no diretório raiz (o nome da pasta principal onde o projeto inteiro está dentro), e dar o comando npm start!

```
//importando o react
import React from 'react';
//importando os componentes para criar as rotas
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
//Importando o Home
import Home from '../Pages/Home/Home';
i

//Criando nosso componente App
function App() {
  return (
      <Router>
        <Routes>
        {
            //criando a rota pra home
            //A barrinha vazia significa a página
            //inicial
        }
          <Route path='/' element={<Home />} />
        </Routes>
      </Router>

  );
}
//Tornando nosso componente exportável
export default App;
```

Criando nosso componente SessionAction

No nosso site, temos um componente onde chama o cliente a conhecer a ChocoBytes:

É ela que criaremos agora!

Criando nosso componente SessionAction

Dentro da nossa pasta components, vamos criar uma pasta SessionAction, e dentro dela, criaremos dois arquivos: Session.js e Session.css

Dentro do Session.js:

```
//importação do React
import React from 'react';
//Importação do css
import './Session.css';
//importação da imagem que utilizaremos
import banner from '../../Assets/banca.png';
```

Criando nosso componente SessionAction

Vamos criar a estrutura, que consiste em uma imagem e um texto, lado a lado:

```
//Criando nosso componente Session
function Session() {
  return (
    //Criando o container
    <section className='banner'>
      {
        //chamando a imagem
      }
      <img src={banner} alt="logo" className="banner-img" />
      <div className="banner-text">
        {
        //Conteúdo de texto
        }
        <h2>Conheça a ChocoBytes</h2>
        <p>
          Descubra uma variedade deliciosa de chocolates feitos com ingredientes de qualidade. Explore nossas opções e encontre o seu favorito hoje mesmo!
        </p>
      </div>
    </section>
  );
}
//Tornando nosso conteúdo exportável
export default Session;
```

Chamando nosso componente Session dentro de Home

Dentro da nossa pasta Home, vamos fazer a importação do Session, e colocar abaixo do Header:

```
//fazendo a importação do Header
import Header from '../../Components/Header/Header'
//Fazendo a importação do Session
import Session from "../../Components/SessionAction/Session";

//criando nossa página home
function Home(){
    //definindo o que ela retorna ao
    //ser chamada
return(
//encapsulando nosso conteúdo
    <>
    <Header/>
    {
        //chamando Session
    }
    <Session/>
    </>
)
}
//tornando nosso conteúdo exportável
export default Home
```

Criando componente ChocoCard

Vamos criar os cards de chocolate que serão renderizados no catálogo e em uma sessão da nossa página inicial!

As páginas geralmente são a junção dos componentes, por isso costumam ser mais curtinhas!

Dentro da pasta components, vamos criar um arquivo ChocoCard, e criar dentro dela, dois arquivos: Card.js e Card.css

Criando componente ChocoCard

No final ficará assim, mas precisamos criar a estrutura primeiro!

Dentro do Card.js

```
//importando o react e o useState por que teremos variáveis aqui
//Importando o estilo
//importando ícone de coração para usar na hora de favoritar
//Importando nosso contexto
import React, { useState } from 'react';
import './Card.css';
import { FaHeart } from 'react-icons/fa';
import { useChocolates } from '../../Context/produtos';
```

```
//criando nosso componente card, e ele precisa receber tudo que os dados tem!
function Card({ id, image, tipo, preco, descricao }) {
  //Pegando as funções e a lista  que utilizaremos nesse componente
  //Pegando lá do contexto
  const { adicionarFavorito, removerFavorito, favoritos, adicionarAoCarrinho } = useChocolates();
```

Dentro do Card.js

```
//importando o react e o useState por que teremos variáveis aqui
//Importando o estilo
//importando ícone de coração para usar na hora de favoritar
//Importando nosso contexto
import React, { useState } from 'react';
import './Card.css';
import { FaHeart } from 'react-icons/fa';
import { useChocolates } from '../../Context/produtos';
```

```
//criando nosso componente card, e ele precisa receber tudo que os dados tem!
function Card({ id, image, tipo, preco, descricao }) {
  //Pegando as funções e a lista  que utilizaremos nesse componente
  //Pegando lá do contexto
  const { adicionarFavorito, removerFavorito, favoritos, adicionarAoCarrinho } = useChocolates();
```

Dentro do Card.js

```
// favoritos.some((item) => item.id
// === id) é uma função que verifica
// se algum item na lista favoritos
// tem o mesmo id que o chocolate atual.
// A função some percorre a lista
//favoritos e aplica a função
//callback (item) => item.id === id a cada elemento.
// Se algum item na lista favoritos
//tiver um id que corresponde ao id
//do chocolate atual, a função some
//retorna true. Caso contrário, retorna false.
const isFavorito = favoritos.some((item) => item.id === id);
const [isAdded, setIsAdded] = useState(false); // Estado para rastrear se o item foi adicionado
```

Dentro do Card.js

```
//função que será chamada ao clicar
  //no coração
 const handleFavoritoClick = () => {
   //Se o valor na variável isFavorito
   // for true, significa que o item
   //ja esta favoritado.
   //Se o usuário clicou no coração
   //Depois de ja ter adicionado,
   //ele quer remover dos favoritos
   if (isFavorito) {
     //chamando a função que remove
     //e mandando o id
     removerFavorito(id);
   } else {
     //Se o coração não foi clicado, o
     //usuário quer adicionar aquele
     //produto a lista de favoritos
     //Para isso, precisamos mandar
     //todas as informações e adicionar
     //a lista de favoritos
     adicionarFavorito({ id, tipo, preco, descricao, imagem: image });
   }
```

Dentro do Card.js

```
/Renderizando nosso card
  return (
    <div className="card">
      {/*Passando nossas propriedades image e tipo, que é o nome do produto */}
      <img className="card-image" src={image} alt={tipo} />
      <div className="card-content">
        <h2 className="card-title">
          {tipo}
          {/*Colocando o ícone de coração */}
          <FaHeart
          {/*Chamando a função de favoritar*/}
            onClick={handleFavoritoClick}
            {/*O estilo altera, se a variavel isFavorito */}
                        {/*For true, ele fica vermelho */}

            style={{ color: isFavorito ? 'red' : 'black', cursor: 'pointer' }}
          />
        </h2>
```

Dentro do Card.js

```
{/*Arredondando o preço para duas casas decimais */}
        <p className="card-price">Preço: R${preco.toFixed(2)}</p>
        <p className="card-description">{descricao}</p>
      </div>
      <button
      /*Chamando a função de adicionar ao carrinho */
        onClick={handleAddToCart}
        //o estilo do cartão muda dependendo da variável isAdded
        //Se clicar a primeira vez, ele fica laranja e muda o
        //conteúdo para adicionado!
        //Se clicar uma segunda vez no mesmo botão, ele remove
        //do carrinho e volta ao estilo comum
        className={`add-to-cart-button ${isAdded ? 'added' : ''}`}
      >
        {isAdded ? 'Adicionado' : 'Adicionar ao Carrinho'}
      </button>
    </div>
  );
}

export default Card;
```

Criando o componente catálogo

Nossa página Home é composta por 3 componentes: Header, Session, e os cards. Mas vejam bem, nosso componente card renderiza todos os cards da lista, e não quero isso! Isso seria meu catálogo, quero renderizar apenas 9 cards, mas pra isso, precisamos criar o componente catálogo, afinal, definimos a estilização dos cards, mas precisamos definir também o estilo do container.

Dentro da pasta components, vamos criar uma pasta chamada Catalogo, e dentro dela, dois arquivos: Catalogo.js e Catalogo.css

Criando o componente catálogo

```
//importação do React
import React from 'react';
//Chamando o componente Card, afinal esse
//é o container dos cards
import Card from '../ChocoCard/Card';
//importando o css
import './Catalogo.css';
//Chamando nosso contexto
import { useChocolates } from '../../Context/produtos';
```

Criando o componente catálogo

```
//criando nosso componente catálogo
function Catalogo() {
  //chamando nossos dados
  const { chocolates } = useChocolates();
  //criando uma lista, que pega apenas os 9 primeiros itens
  //da lista chocolates
  const primeirosNoveChocolates = chocolates.slice(0, 9);
```

Criando o componente catálogo

```
//definindo o que vai ser retornado ao chamar esse componente
  return (
    //container
    <div className="catalogo">
      {
        //percorrendo a lista com 9 chocolates e passando
        //as propriedades pro card
      }
      {primeirosNoveChocolates.map(chocolate => (
        <Card

          key={chocolate.id}
          id={chocolate.id}
          image={chocolate.imagem}
          tipo={chocolate.tipo}
          preco={chocolate.preco}
          descricao={chocolate.descricao}
        />
      ))}
    </div>
  );
}

//tornando nosso conteúdo exportável
export default Catalogo;
```

Agora que terminamos todos os componentes que compõe nossa página Home, vamos juntar tudo e temos uma página pronta!

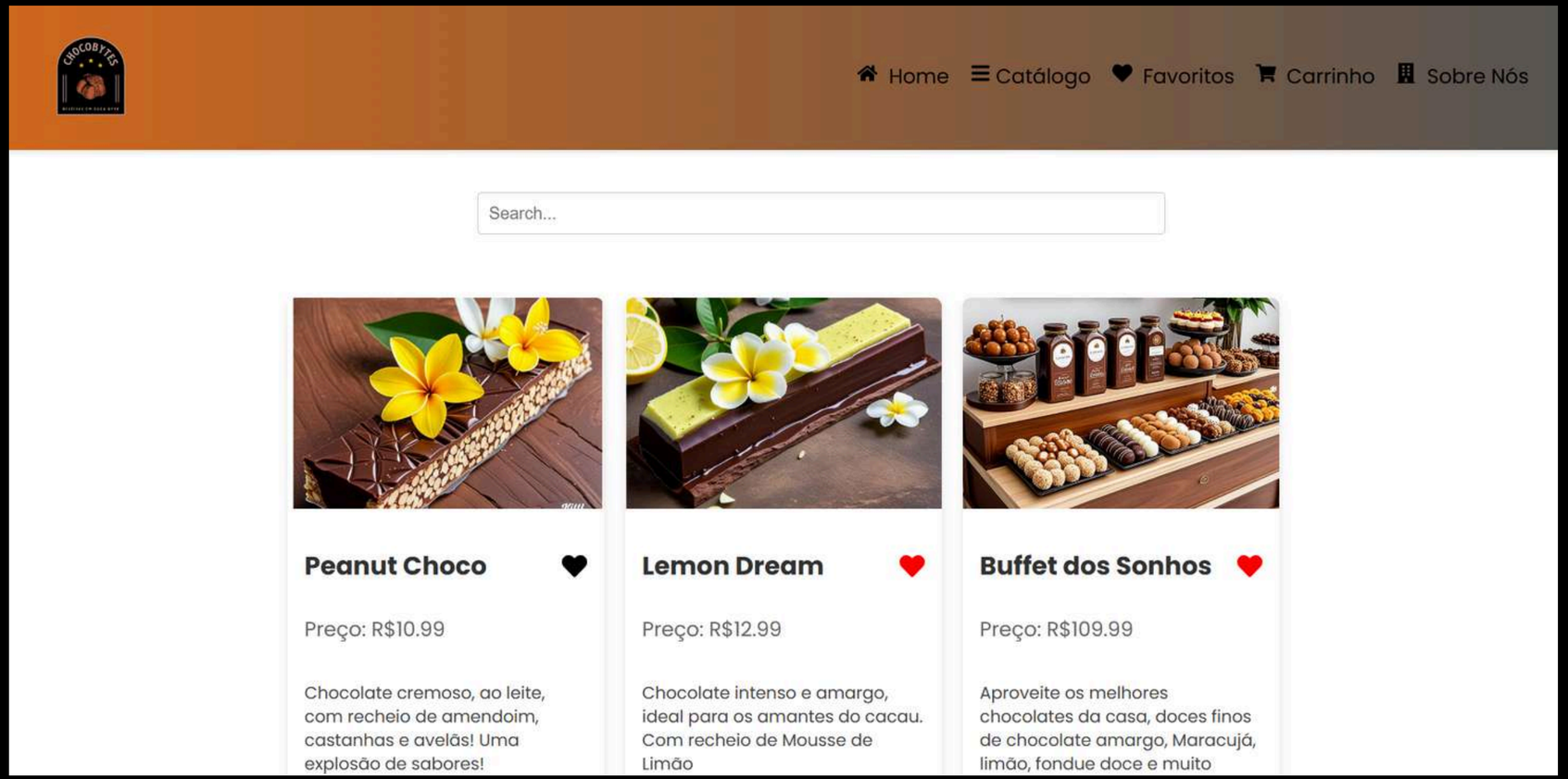
Juntando todos os componente na Home

```
//importando nossos componentes
import Header from '../../Components/Header/Header'
import Catalogo from "../../Components/Catalogo/Catalogo";
import Session from "../../Components/SessionAction/Session";
//importando nosso contexto, afinal, o catálogo esta renderizando os
//cards, e os cards precisam dos dados para serem renderizados!
import { ChocolatesProvider } from "../../Context/produtos";
function Home(){
return(

    <>
    <Header/>
    <Session/>
    {
        //Envolvendo nosso catálogo com os dados
    }
    <ChocolatesProvider>
    <Catalogo/>
    </ChocolatesProvider>
    </>
)
}

export default Home
```

Terminamos nossa página Home! Lembrando, o componente Catálogo não é a Página Catálogo! Essa vamos criar agora!

Criando os componentes da nossa página Catálogo!

Antes de qualquer coisa, é assim que vai ficar nossa página Catálogo: Barra de navegação (Header), uma barrinha para pesquisar os itens, e vários cards para renderizar nossos produtos!



Criando os componentes da nossa página Catálogo!

Já temos nosso Header, e vamos construir a página após ter todos os componentes! Hora de começar nossa barrinha de pesquisa!

Crie uma pasta dentro de components, chamada NavBar, e dentro dela, dois arquivos: NavBar.js e NavBar.css:

Esse componente é bem curtinho!

Criando os componentes da nossa página Catálogo!

```jsx
//importação do React
import React from 'react';
 // Estilização do componente
import './NavBar.css';

//criando o componente, ele depende do evento de
//onchange, ou seja, do ato do usuário
//digitar alguma coisa
function SearchInput({ onChange }) {

  //criando a função que pega o valor digitado
  //dentro do input
  const handleInputChange = (event) => {
    onChange(event.target.value); // Chama a função onChange passando o valor do input
  };

  //definindo nosso retorno:
  return (
    <div className="search-input-container">
      <input type="text" className="search-input" placeholder="Search..." onChange={handleInputChange} />
    </div>
  );
}
//tornando nosso componente exportável
export default SearchInput;
```

Criando os componentes da nossa página Catálogo!

Temos nosso Header, barra de navegação, e os cards, assim como os produtos que serão renderizados. Hora de criar a página!

Dentro de Pages, vamos criar a pasta Catálogo, e dentro dela, dois arquivos: index.js e Catalogo.css

```
//importando react e useState por que teremos
//variáveis aqui
import React, { useState } from 'react';
//chamando nosso contexto por que os cards precisam
//estar envoltos nele
import { useChocolates } from '../../Context/produtos';
//chamando o componente card que vai renderizar
// nossos dados
import Card from '../../Components/ChocoCard/Card';
//chamando nossa barra de pesquisa
import SearchInput from '../../Components/NavBar/NavBar';
import './Catalogo.css'; // Importar o CSS
//importando o Header
import Header from '../../Components/Header/Header';
```

Página catálogo!

```
//criando nosso componente Pagina de Catalogo
function PageCatalogo() {
  //usando nosso contexto
  const { chocolates } = useChocolates();
  //criando o filtro que vai ser aplicado na barra
  //de pesquisa
  const [filtro, setFiltro] = useState('');

  // Função para atualizar o filtro de busca
  const handleSearchChange = (value) => {
    setFiltro(value);
  };

  // Filtrar chocolates com base no texto digitado no
  // input, ele pega a lista e testa o nome do produto,tanto com letra maiúscula quanto minúscula

  const chocolatesFiltrados = chocolates.filter(chocolate =>
    chocolate.tipo.toLowerCase().includes(filtro.toLowerCase())
  );
```

Página catálogo!

Com isso, terminamos nossa página catálogo, agora basta importar nosso componente e criar a rota lá no App.js:

```
import PageCatalogo from '../Pages/Catalogo/index';
```

Dentro de Routes, vamos criar um novo Route:

```
<Route path='/catalogo' element={<PageCatalogo  />} />
```

```
//nosso retorno chamando o Header, a barra de pesquisa
// e os cards, que puxa os dados do contexto,
//que também importamos
  return (
    <>
      <Header />
      <div className="page-catalogo">
        <div className="search-input-container">
          <SearchInput onChange={handleSearchChange} />
        </div>
        <div className="catalogo">
          {chocolatesFiltrados.map(chocolate => (
            <Card
              key={chocolate.id}
              id={chocolate.id}
              image={chocolate.imagem}
              tipo={chocolate.tipo}
              preco={chocolate.preco}
              descricao={chocolate.descricao}
            />
          ))}
        </div>
      </div>
    </>
  );
}

export default PageCatalogo;
```