Tutorial de aplicação (blog) em React Js

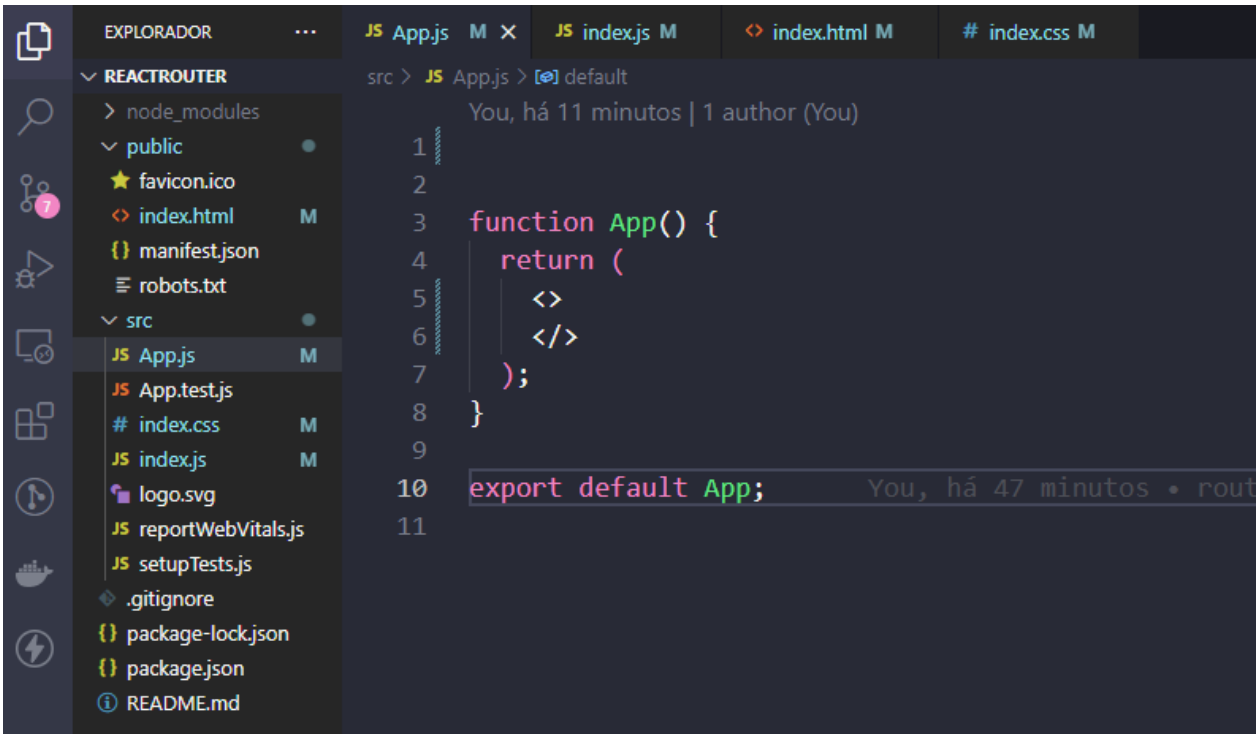
Tópicos do Tutorial:

Criando o projeto
React Router DOM
Roteamento aninhado
Hooks
CSS - Modules
Props
Components (Function Components)
Importação Absoluta de Componentes
React Hook Form
Validação simplificada
Validação com Yup

Vamos seguir sempre a documentação da biblioteca →
https://reactrouter.com/en/main
E com a documentação do React Js → https://reactjs.org/
Link do repositório do Tutorial feito na aula:
https://github.com/cristijung/ReactRouter.git

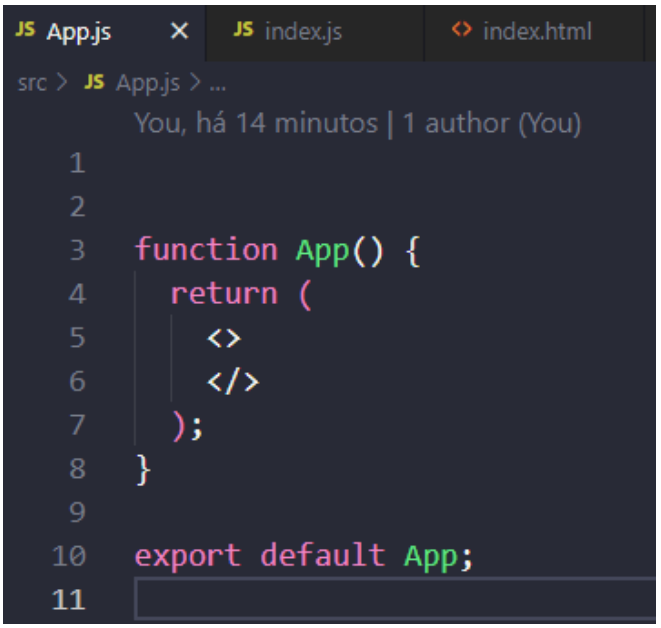
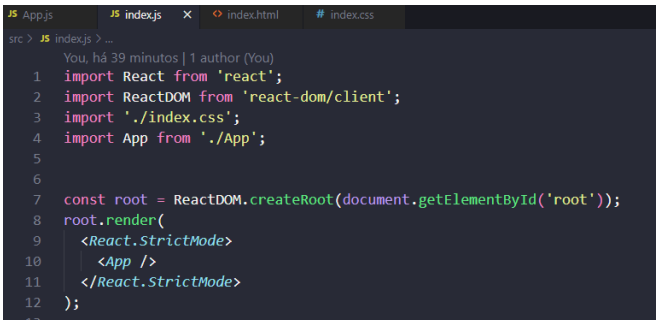
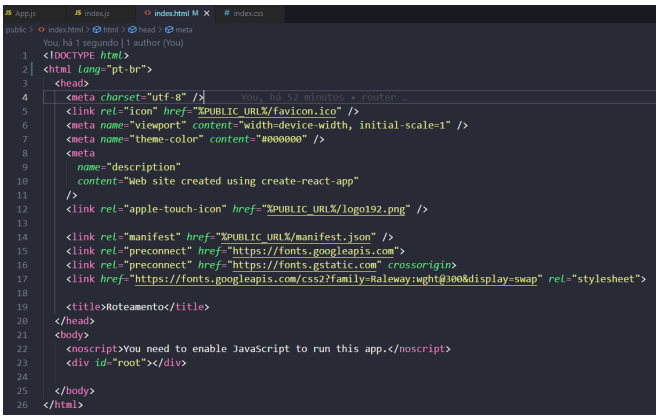
Passos do Tutorial

Vamos criar um novo projeto no React. Vamos deixar a estrutura dos arquivos conforme a imagem abaixo:



Codificação e organização:

Arquivo	Código

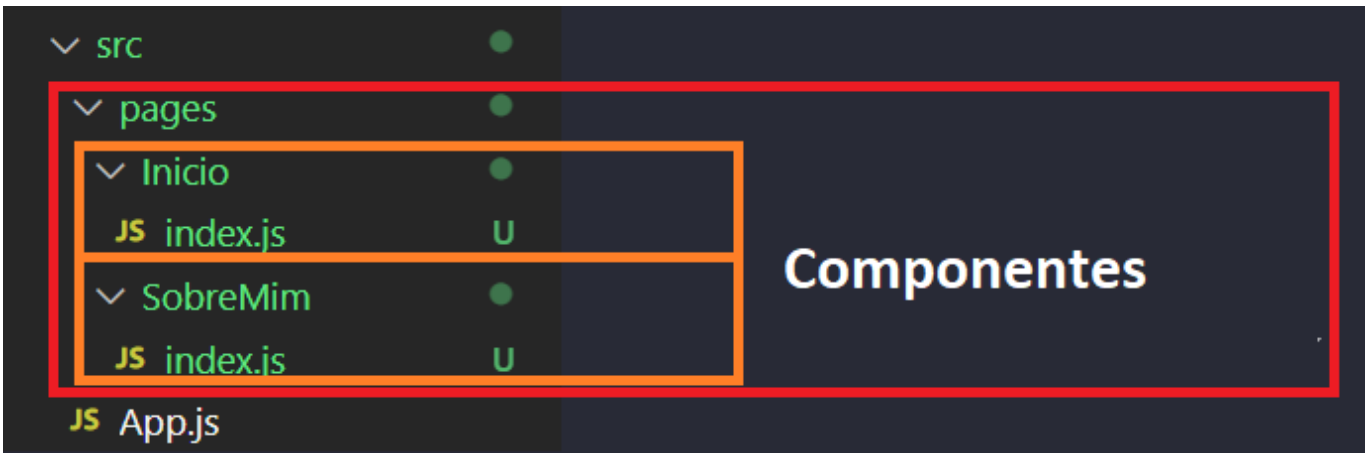
App.js	
index.js	
index.css **Observação: pegue a fonte do Google Fonts e não esqueça de colocar a referência no Head do index.html.**	estilização do [css]
index.html	

Vamos agora instalar a biblioteca do React Router DOM no projeto.

```
PS C:\React\ReactRouter> npm install react-router-dom@6
```

Observe que no arquivo package.json, a referência já foi adicionada.

Na próxima etapa vamos criar nossos componentes ainda bem simplificados.



Na codificação deles, vamos deixar somente:

Componente Inicio:

```
You, há 40 segundos | 1 author (You)
1
2   function Inicio () {
3       return(
4           <>
5
6           <h1> Este é o início </h1>
7
8           </>
9       );
10  }
11
12  export default Inicio;          You, há 27 seg
```

Componente SobreMim:

```
You, há 2 minutos | 1 author (You)
1
2   function sobreMim () {
3       return(
4           <>
5
6           <h1> Este é o Sobre Mim </h1>
7
8           </>
9       );
10  }
11
12  export default sobreMim;        You, há 2 minutos •
```

Agora vamos retornar para o nosso arquivo App.js e fazer as configurações necessárias para que a nossa rota esteja em perfeita funcionalidade:

Primeiro iremos importar os nossos componentes e o React Router DOM no arquivo App.js

```
You, há 1 segundo | 1 author (You)
1  import { BrowserRouter, Route, Routes } from "react-router-dom";
2  import Inicio from "./pages/Inicio";
3  import sobreMim from "./pages/SobreMim";
```

Na verdade, nem precisamos importar, quando já declaramos os componentes do React Router DOM, o VsCode automaticamente já importa.

Por enquanto nosso arquivo App.js, está desta forma:

```
You, há 2 minutos | 1 author (You)
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Inicio from "./pages/Inicio";
import sobreMim from "./pages/SobreMim";
      You, há 19 horas • router …
function App() {
  return (
    <>
    <BrowserRouter>
      <Routes>
        <Route/>
      </Routes>
    </BrowserRouter>
    </>
  );
}

export default App;
```

Vamos conhecer o que cada declaração de componente é e qual a sua funcionalidade:

1. **BrowserRouter:** é um componente responsável por informar a nossa aplicação que teremos um roteamento de componentes, por conta disso ele ficará em volta dos componentes **<Routes> e </Routes>**.
2. **Routes:** Este componente é como se fosse um roteador, ele vai conter as rotas especificadas para as **'pages'** que compõem a nossa aplicação, ele envolve o componente auto fechante **<Routes/>** e o usamos quando a nossa aplicação possuir muitas rotas.
3. **Route:** Componente responsável por associar a rota. Nele temos três parâmetros:

a. **Component ou Element - depende da versão:** O parâmetro element (component) recebe o componente que precisa ser exibido ao acessar a rota.
b. **Path:** O parâmetro path é o caminho na URL que precisa ser acessado para mostrar o componente, definido pelo parâmetro component.
c. **Exact:** O parâmetro exact determina qual o componente vai ser exibido apenas se a rota for igual ao definido entre aspas, no nosso caso se for exatamente '**/**'.

E agora, vamos fazer o roteamento da nossa aplicação. Observe a imagem abaixo, nesta aplicação faremos o roteamento no componente pai, o App.js

```
You, há 3 minutos | 1 author (You)
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Inicio from "./pages/Inicio";
import SobreMim from "./pages/SobreMim"

function App() {
  return (
    <>
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Inicio />} />
        <Route path="/SobreMim" element={<SobreMim />} />
        <Route path="*" element={<div>Erro 404 - Página não encontrada.</div>} />
      </Routes>
    </BrowserRouter>
    </>
  );
}

export default App;
```

Um outro ponto importante a ressaltar aqui é a linha:

```
<Route path="*" element={<div>Erro 404 - Página não encontrada.</div>} />
```

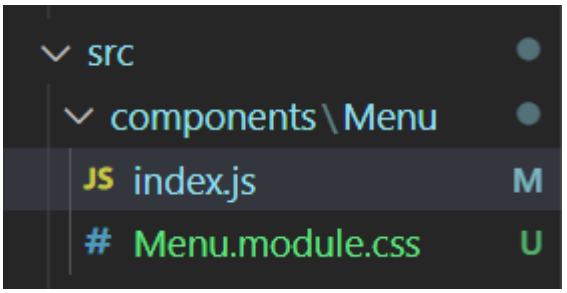
O asterisco funciona como um coringa e será direcionado para a nossa página de Erro 404. O coringa tem a função de rotear uma página específica (404), quando

não encontrar a página original definida para aquele roteamento, ou será ativada quando o usuário escolher alguma opção inexistente na aplicação.

Na próxima etapa iremos fazer o nosso menu que irá substituir a digitação na barra de endereço no navegador, ou seja, vamos começar a implementar a navegabilidade na aplicação.

Num primeiro ponto, queremos que a pequena barra de Menu fique disponível para toda a aplicação então iremos fazer os seguintes passos:

1. **Criar uma pasta components para a criação de componentes, dentro da pasta src;**
2. **Criar uma pasta para o componente Menu;**
3. **Criar um componente para o Menu;**
4. **Estilizar - arquivo menu.module.css;**
5. **Inserir o componente no arquivo App.js**

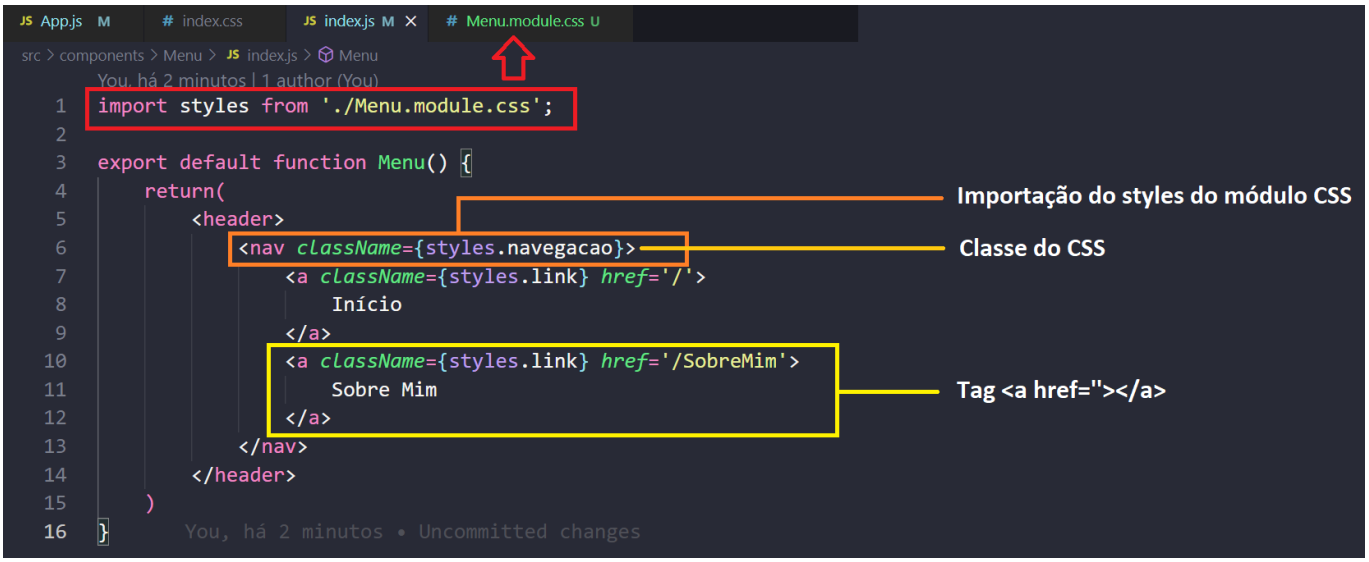


Um esclarecimento ....

Por que é aconselhável usar CSS-modules no React?

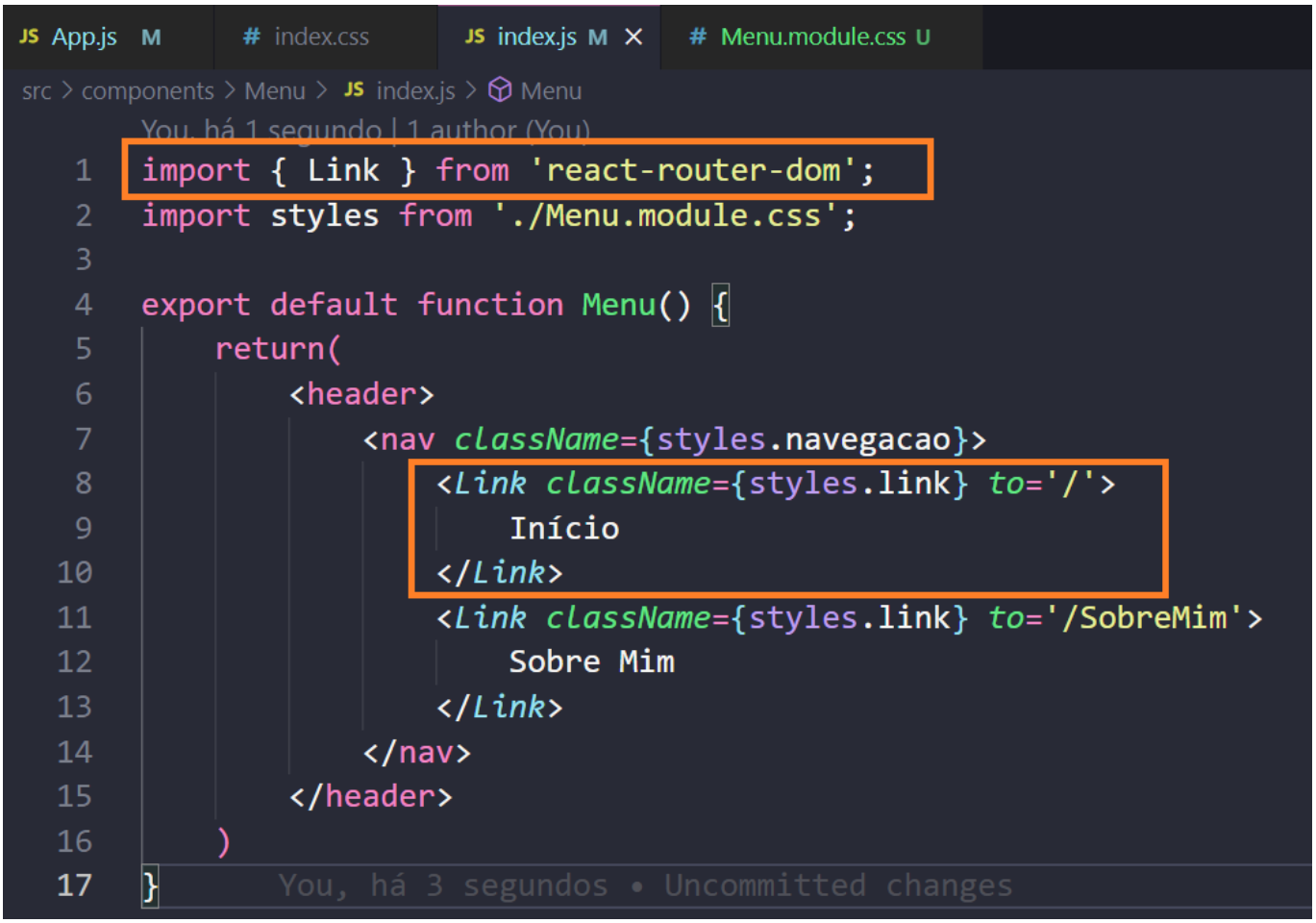
É importante a definição no momento do projeto como a estilização será desenvolvida e acoplada nos componentes, conforme o projeto vai crescendo as decisões já tomadas no projeto podem se tornar um problema e, para não sair do padrão definido pelo design system ao qual o projeto pertence, a declaração dos **classNames** pode afetar diretamente as premissas da aplicação.
Quando usamos o **CSS-module** é possível criar estilos únicos para cada componente, porque são arquivos css em que os classNames são definidos

localmente, **isso significa que os estilos ali criados, só serão declarados dentro daquele escopo, e não globalmente, evitando conflitos entre estilos.**

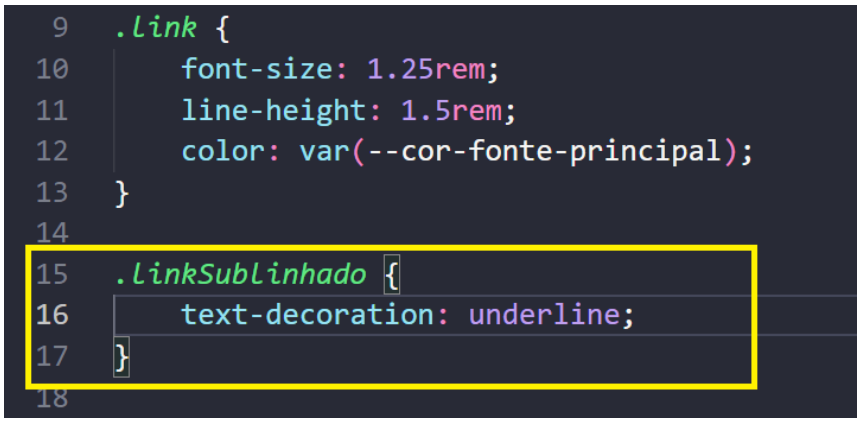


O componente <Lin></Link> do React Router DOM

Pois então ….. a tag **<a>** é nativa do Html e sempre que clicamos em um link ela sempre irá recarregar a página e sabemos que no React estamos utilizando sempre a mesma página Html. Para que possamos usar o dinamismo de troca entre páginas, utilizando o JavaScript puro, ou nesse nosso caso, a biblioteca React Router Dom. Vamos usar um componente específico do React Router DOM. O componente renderiza um elemento acessível e só 'pula' para o roteamento definido e o browser lida com a transição das rotas.
https://reactrouter.com/en/main/components/link

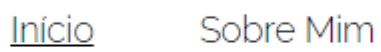


Ok para a nossa navegação, vamos agora ver como podemos acoplar mais estilizações nos elementos de um componente. No arquivo Menu.module.css vamos inserir ...



E como declarar no nosso link?

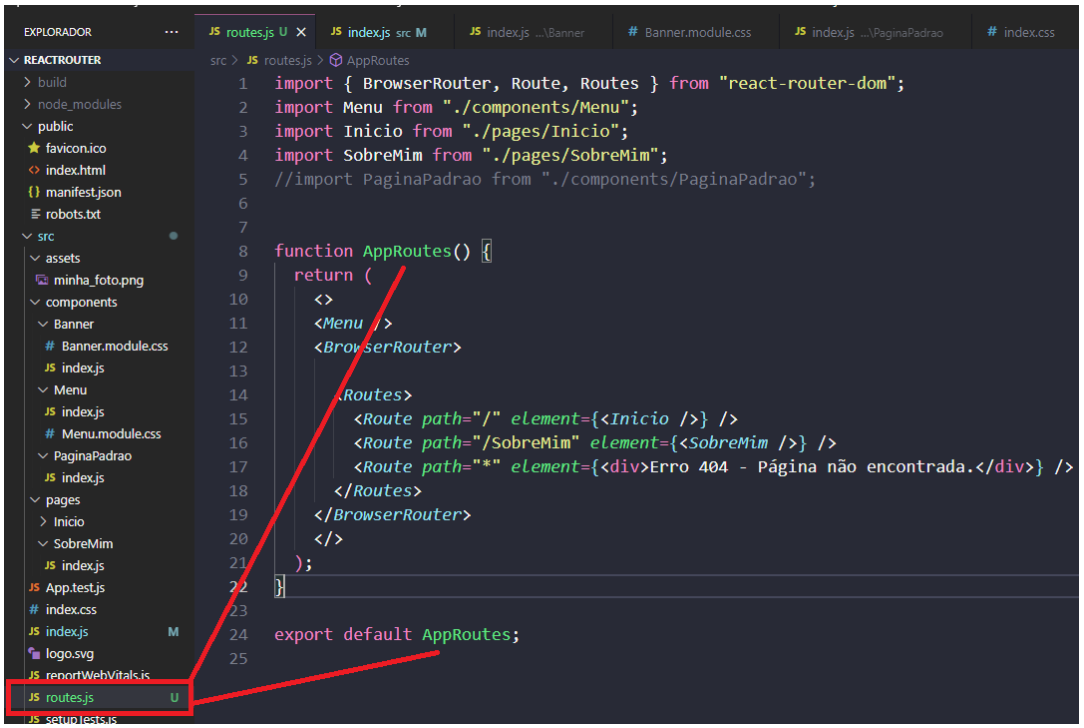
```
<Link to='/' className={`
${styles.navegacao}
${styles.linkSublinhado}
`}>
    Início
</Link>
```



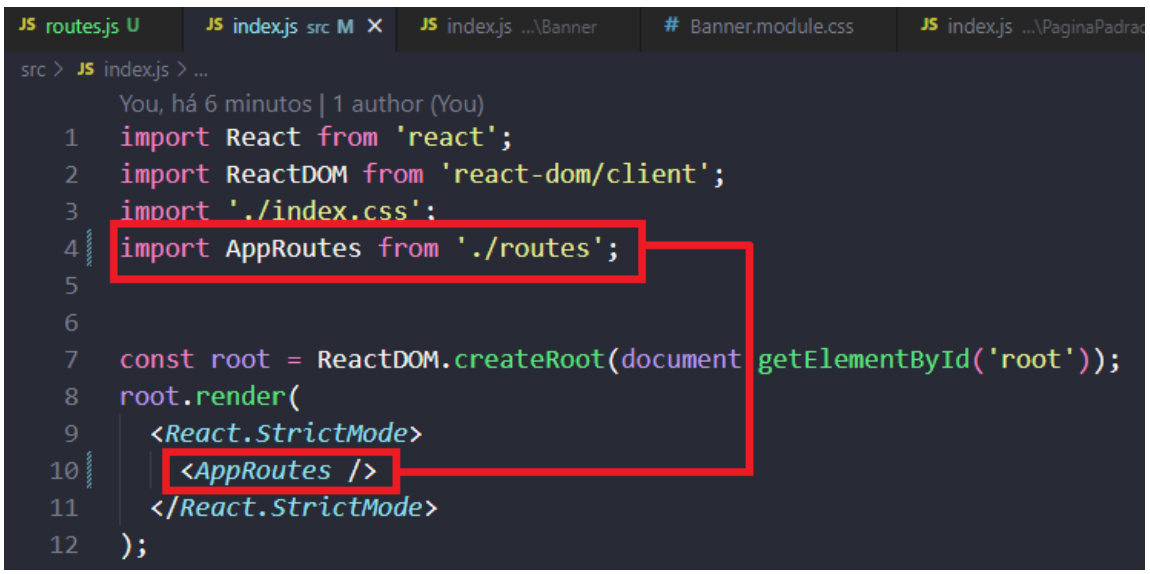
Dica de configuração de importação absoluta no projeto React → arquivo jsconfig.js

Importing a Component | Create React App

Agora vamos fazer uma pequena alteração para deixar a aplicação um pouco mais padrão.
Vamos renomear o arquivo App.js para routes.js

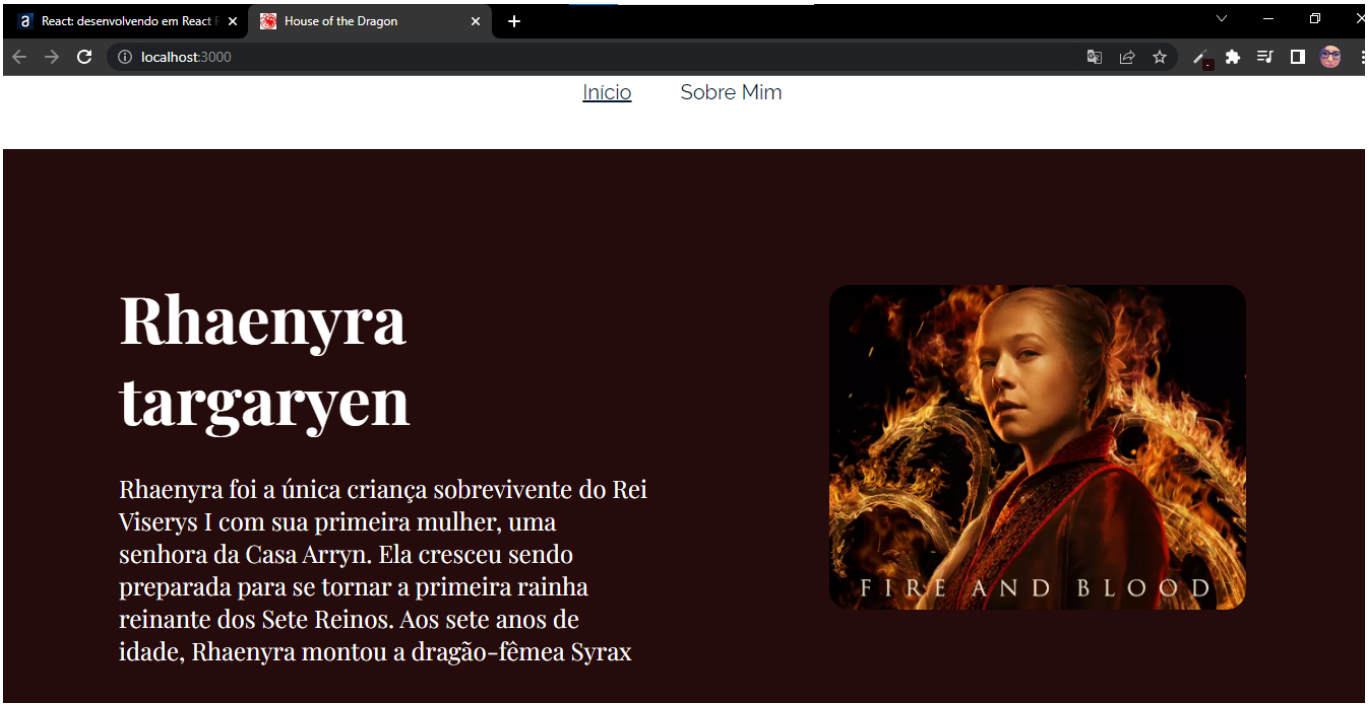


E no arquivo index.js, deixaremos assim:

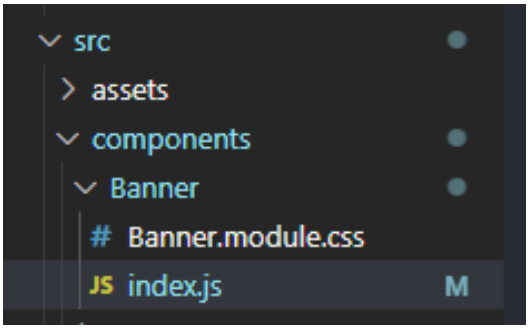


Criando o Banner da nossa aplicação

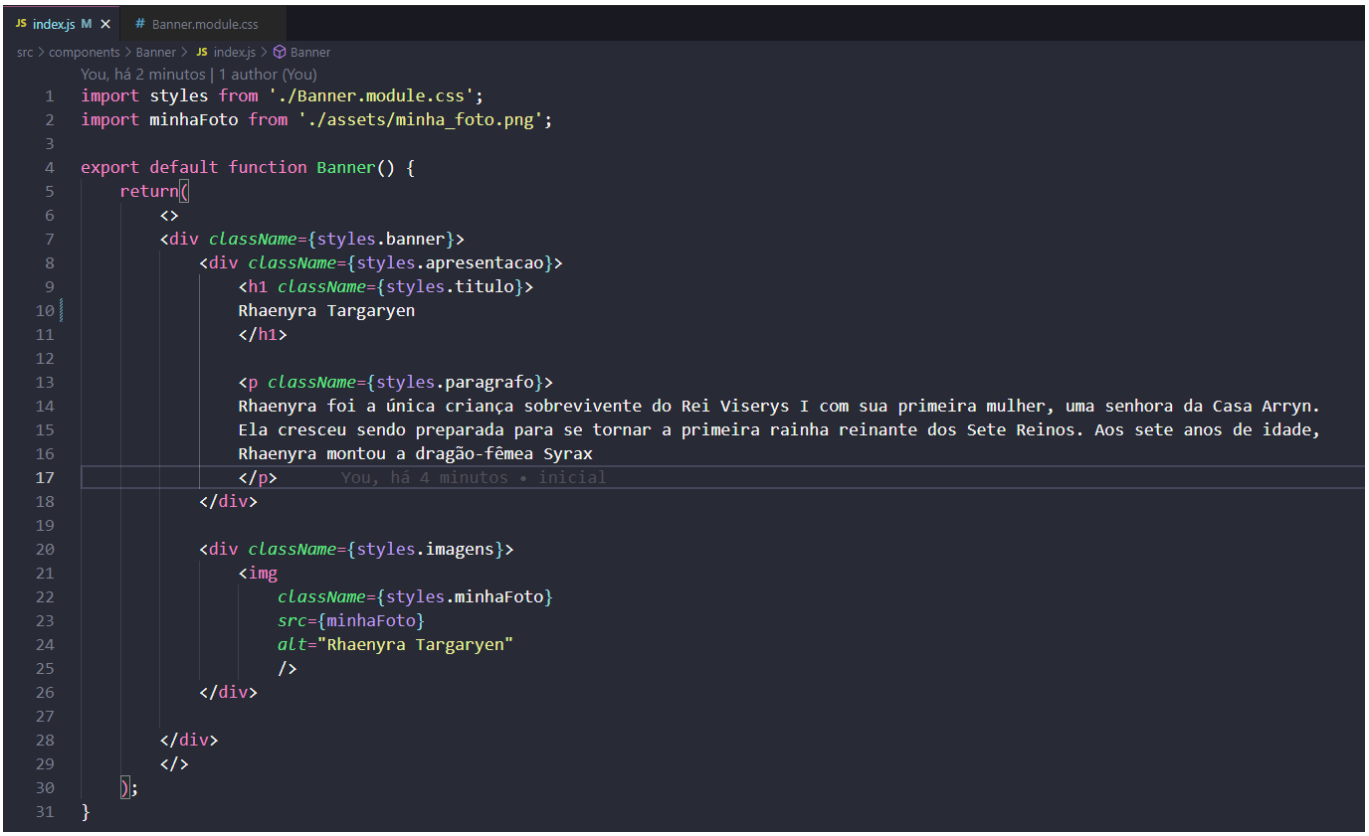
O banner irá aparecer em quase todas as nossas páginas da nossa aplicação, ele só não irá aparecer na página 404.
Então, pensando nisso, no VSCode não iremos colocar o banner em routes.js para que ele não apareça em todas as páginas. Por enquanto, vamos criar um componente chamado banner, e importá-lo na página de início, depois na página de 'sobre mim' neste primeiro momento. (exemplo do banner na imagem abaixo)



Vamos a criação do Componente:



O código do componente e, não esqueça que a estilização já foi disponibilizada e já se encontra no repositório deste tutorial. Tanto na estilização, quanto no conteúdo do componente você pode personalizar como desejar.

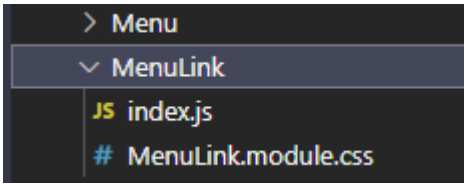


Finalizado o Banner, vamos configurar agora de forma mais adequada o nosso Menu.

Na próxima etapa vamos criar um outro componente de Menu para que possamos encapsular as informações do menu que precisamos passar entre os outros componentes

Componente MenuLink

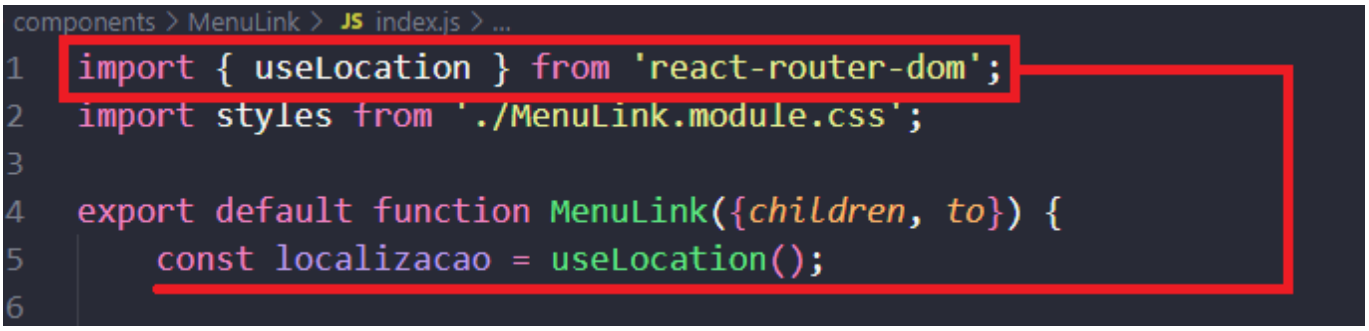
Chegou a hora de encapsular as funcionalidades do nosso menu e esta ação resultará em reutilização efetiva de código, **uso de props e hooks do React.**



Código do arquivo index.js

```
index.js  ×  MenuLink.module.css
src > components > MenuLink > index.js > ...
1  import { useLocation } from 'react-router-dom';
2  import styles from './MenuLink.module.css';
3
4  export default function MenuLink({children, to}) {
5      const localizacao = useLocation();
6
7      return(
8          <Link className={`
9              ${styles.link}
10             ${localizacao.pathname === to ? styles.linkDestacado : ''}
11         `} to={to}>
12             {children}
13         </Link>
14     )
15 }
16
```

Estudando o código:



Aqui importamos o useLocation, de react-router-dom. O **useLocation é um hook do React Router Dom**, e ele vai retornar alguma coisa para a nossa aplicação e estamos atribuindo uma const chamada de **localizacao**.

Para saber mais:

React Hooks: o que é e como funciona? | Zup
Explicando todos React Hooks com exemplos | by Sérgio Junior | React Brasil | Medium
Hooks API Reference – React

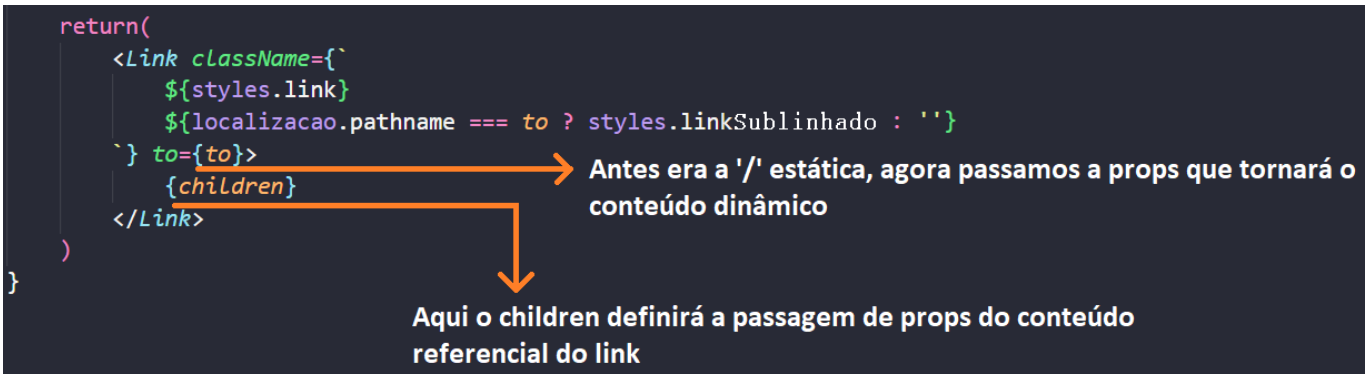
Já sabemos que Hooks são uma forma que você pode escrever componentes React de uma maneira menos verbosa e mais performática se utilizando de funções para poder acessar vários recursos de forma mais simplificada. **O hook que estamos usando, retorna a localização real das páginas e componentes que estão sendo renderizados no navegador.**

Na linha ....

```
export default function MenuLink({children, to}) {
```

Estamos passando uma props que receberá o nome dos menus que iremos substituir lá no componente Menu (na verdade, depois iremos retirar os Links que lá estão e iremos substituir pelo MenuLink).

A props se chama **'to'** e estamos criando uma desestruturação Js nestes parâmetros → Estamos declarando a **prop children** e criando dela uma outra prop chamada **'to'**.



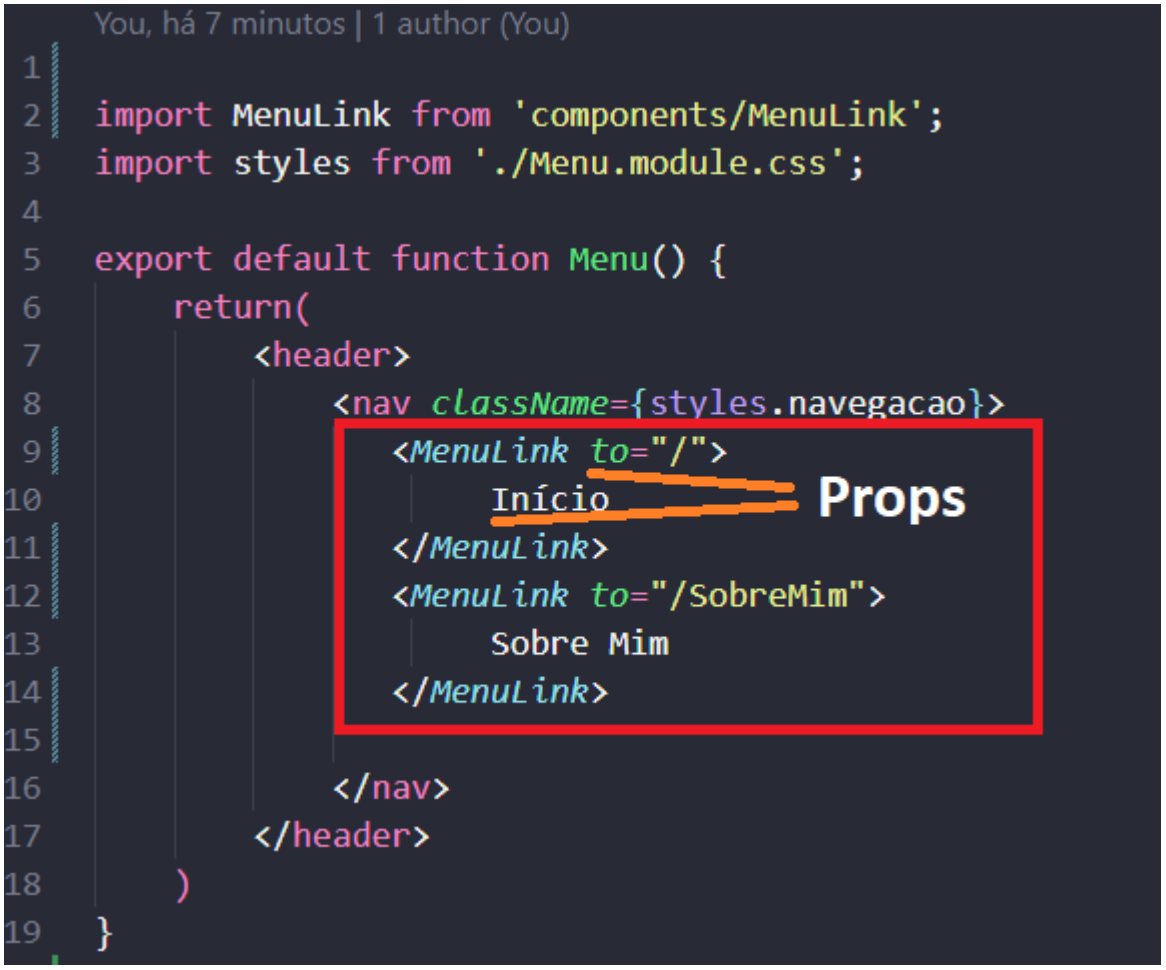
Nesta linha .....

```
${localizacao.pathname === to ? styles.linkSublinhado : ''}
```

Aqui estamos usando um ternário para que o sublinhado do texto do link fique à mostra somente no menu que está na ativação.
Vamos copiar as classes criadas no arquivo **Menu.module.css** que se referem aos links e recortar para o arquivo **MenuLink.module.css**

```
.link {
    font-size: 1.25rem;
    line-height: 1.5rem;
    color: var(--cor-fonte-principal);
}

.linkSublinhado {
    text-decoration: underline;
}
```

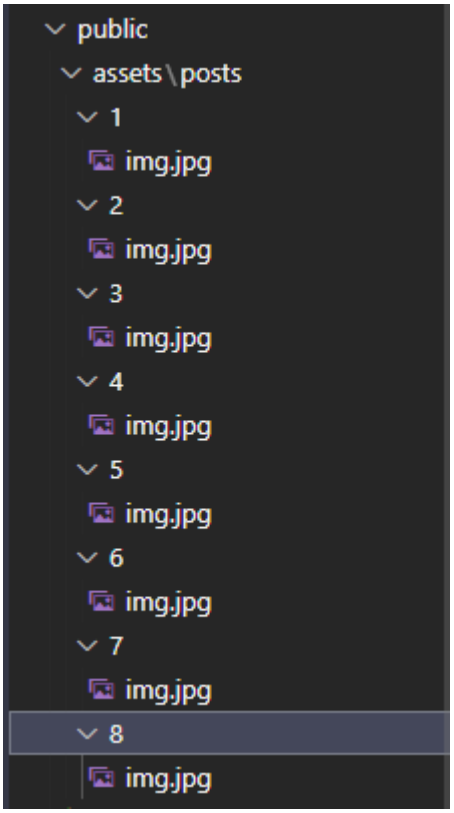
Para que possamos finalizar a componentização do nosso Menu, vamos acessar o arquivo index.js do Menu.


You, há 7 minutos | 1 author (You)
import MenuLink from 'components/MenuLink';
import styles from './Menu.module.css';
export default function Menu() {
return(
<header>
<nav className={styles.navegacao}>
<MenuLink to="/">
Início
</MenuLink>
<MenuLink to="/SobreMim">
Sobre Mim
</MenuLink>
</nav>
</header>
)
}
Props

Finalizando o Componente Inicio

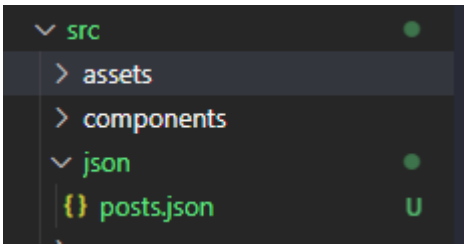
Neste componente, como estamos desenvolvendo um blog, iremos finalizar o componente Início e criar os componentes dos **posts e rodapé**.

Vamos começar a organização do nosso projeto para que ele possa receber os posts.
Dentro da pasta **public**, iremos criar uma pasta **assets/posts**. Dentro dela:



Dentro de cada pasta, iremos colocar uma imagem que fará parte dos nossos posts. Dê preferência para imagens do mesmo tamanho e na proporção horizontal... todas com a mesma extensão e o mesmo nome, pois usaremos um map para colocá-las na tela.

Dentro da pasta src, iremos criar uma pasta com nome de json para colocar o nosso json para pegar os textos dos posts



O modelo deste json será uma lista de objetos. Cada um desses objetos vai representar um post e possuem as seguintes propriedades:

- **id → numérica**
- **título → string**
- **texto → string**

E vamos colocar o conteúdo do Json ali. O arquivo com o tema do blog deste tutorial será disponibilizado no repositório.

Vamos para o **componente Inicio** e vamos criar o arquivo de estilização: **Inicio.molude.css**

```
.posts {
    padding: 0 6vw 3.625rem;
    display: flex;
    justify-content: center;
    flex-wrap: wrap;
    gap: 1.5rem;
}

@media (max-width: 1100px) {
    .posts {
        margin-top: 0;
        padding: 2rem 1.5rem 3.625rem;
    }
}

@media (max-width: 744px) {
    .posts {
        padding: 2rem 1rem 3rem;
    }
}
```

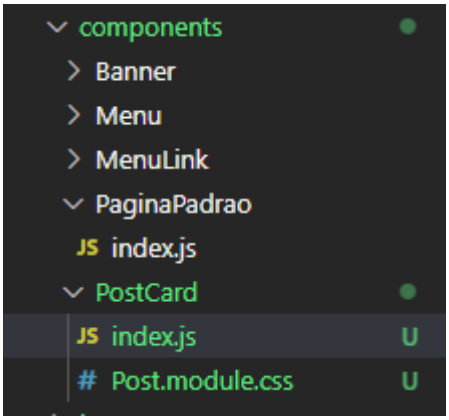
Voltamos ao componente ...

Na linha 3 observe que importamos o arquivo Json.
Dentro das tags ** e **, vamos declarar um map → **{posts.map((post) => (**, para que possamos chamar esse parâmetro de **post**, que obviamente vai ser cada um dos posts.

A tag **<li>** vai retornar uma <li **key={post.id}**>, é interessante que cada li tenha um **atributo key para o React fazer a renderização correta,** e estamos referenciando o post.id, que é aquele atributo do nosso json.

Componente PostCard

Vamos criar o componente PostCard.

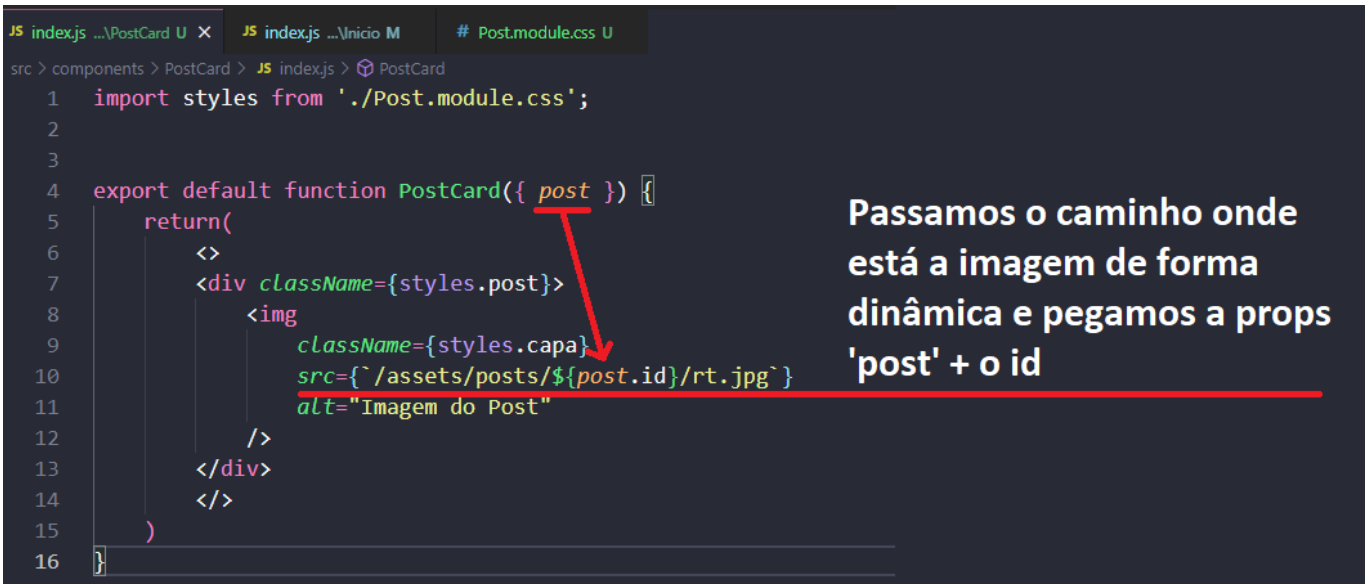


Estilização do Post.module.css

```
.post {
    text-align: center;
    width: 282px;
    box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.08);
    border-radius: 0px 0px 10px 10px;
    padding-bottom: 1.5rem;
    transition: transform .2s;
}

.post:hover {
    transform: translate(0, -1rem);
}

.capa {
    width: 100%;
}

.titulo {
    font-family: var(--fonte-secundaria);
    font-size: 1.25rem;
    color: var(--azul-escuro);
    line-height: 1.75rem;
    margin: 1.5rem 0 1.75rem;
}

@media (max-width: 1100px) {
    .post {
        width: 336px;
```

```
    }

    .titulo {
        font-weight: 600;
    }

    .botaoLer {
        padding: 0.6875rem 1.5rem;
        font-size: 1.375rem;
    }
}
```

No componente PostCard:



Ainda dentro da div criada, vamos digitar:

```
<h2 className={styles.titulo}>{post.titulo}</h2>
            <button className={styles.botaoLer}>Ler</button>
{/** Btn provisório */}
```

Vamos retornar ao index.js do componente Inicio, precisamos declarar o post para que ele apareça na tela da aplicação.

Código do componente PostCar, com botão provisório, depois criamos um componente para ele.

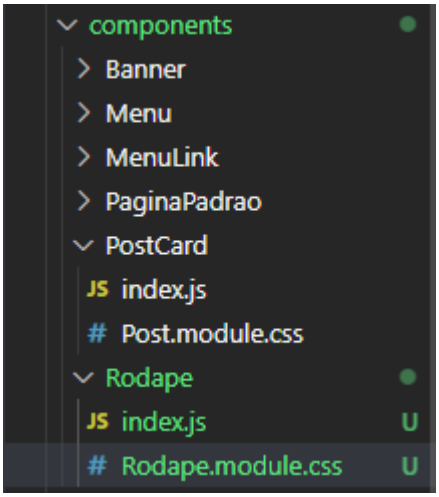
```
export default function PostCard({ post }) {
    return(
        <>      You, há 12 minutos • tela inicio pronta
        <div className={styles.post}>
            <img
                className={styles.capa}
                src={`/assets/posts/${post.id}/img.jpg`}
                alt="Imagem do Post"
            />
            <h2 className={styles.titulo}>{post.titulo}</h2>
            <button className={styles.botaoLer}>Ler</button>

        </div>

        </>
    )
}
```

Componente Inicio ....

```
You, há 30 minutos | 1 author (You)
1  import Banner from "components/Banner"; //configuração de importação absoluta do React
2  import styles from './Inicio.module.css';
3  import posts from 'json/posts.json';
4  import PostCard from "components/PostCard";
5
6  function Inicio () {
7      return(
8          <>
9          <main>
10             <Banner />
11             <ul className={styles.posts}>
12                 {posts.map((post) => (
13                     <li key={post.id}>
14                         <PostCard post={post} />
15                     </li>
16                 ))}
17             </ul>
18         </main>
19         </>
20     );
21 }
22
23 export default Inicio;      You, há 5 horas • inicial
```

Componente Rodape → Footer

Vamos agora criar o rodapé da nossa aplicação. Criamos o componente e os demais arquivos:



O CSS será disponibilizado para vocês → estilização simples.
Só colocamos um texto para que possamos ver o componente onde iremos importar.

```
1  import styles from './Rodape.module.css';
2
3  export default function Rodape() {
4      return(
5          <>
6           Aqui é o rodapé
7          </>
8      )
9  }
10
11
```

Uma observação aqui:
Como o rodapé irá aparecer em toda a aplicação, iremos importar o componente no arquivo routes.js fora do Routes.

```
9  function AppRoutes() {
10   return (
11     <>
12     <BrowserRouter>
13         <Menu />
14       <Routes>
15         <Route path="/" element={<Inicio />} />
16         <Route path="/SobreMim" element={<SobreMim />} />
17         <Route path="*" element={<div>Erro 404 - Página não encontrada.</div>} />
18       </Routes>
19       <Rodape />
20     </BrowserRouter>
21     </>
22   );
23 }
```

Ele já estará visível no navegador e a próxima etapa será fazer o JSX do componente:

Post.module.css | index.js ...\Inicio | index.js ...\Rodape U | minha-foto.png | marca-registrada.svg U | routes.js M

src > components > Rodape > index.js > ...

```
1  import styles from './Rodape.module.css';
2  import { ReactComponent as MarcaRegistrada } from 'assets/marca-registrada.svg';
3
4  export default function Rodape() {
5      return(
6          <footer className={styles.rodape}>
7           <MarcaRegistrada />
8           HBO
9          </footer>
10     )
11 }
```

Observe que na linha 2 estamos com a sintaxe:

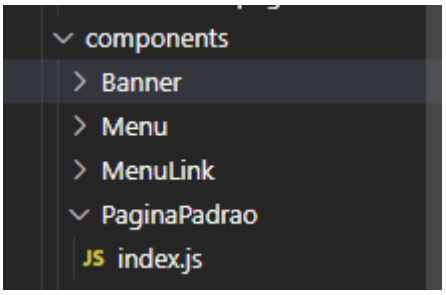
```
import { ReactComponent as MarcaRegistrada } from
'assets/marca-registrada.svg';
```

Esta sintaxe é possível devido ao pacote SVGR, que já vem por padrão em um projeto React. **Esse pacote permite que utilizemos um SVG como um componente React, assim não precisamos utilizá-lo como uma tag img.**

Configurando Rotas Aninhadas

As rotas aninhadas servem para estruturas de aplicação no React quando precisamos estruturar uma aplicação com rotas principais e sub-rotas para acessar informações de sub-níveis, e para isso o React-Router facilita a organização de estrutura de roteamento da nossa aplicação.

Então, vamos criar o componente **PaginaPadrao** e desenvolver a nossa estrutura base das telas ali, para que depois possamos configurar as rotas.



Vamos declarar o nosso arquivo da forma mais padronizada possível, ou seja, como a gente quer que uma matriz de arquivo fique:

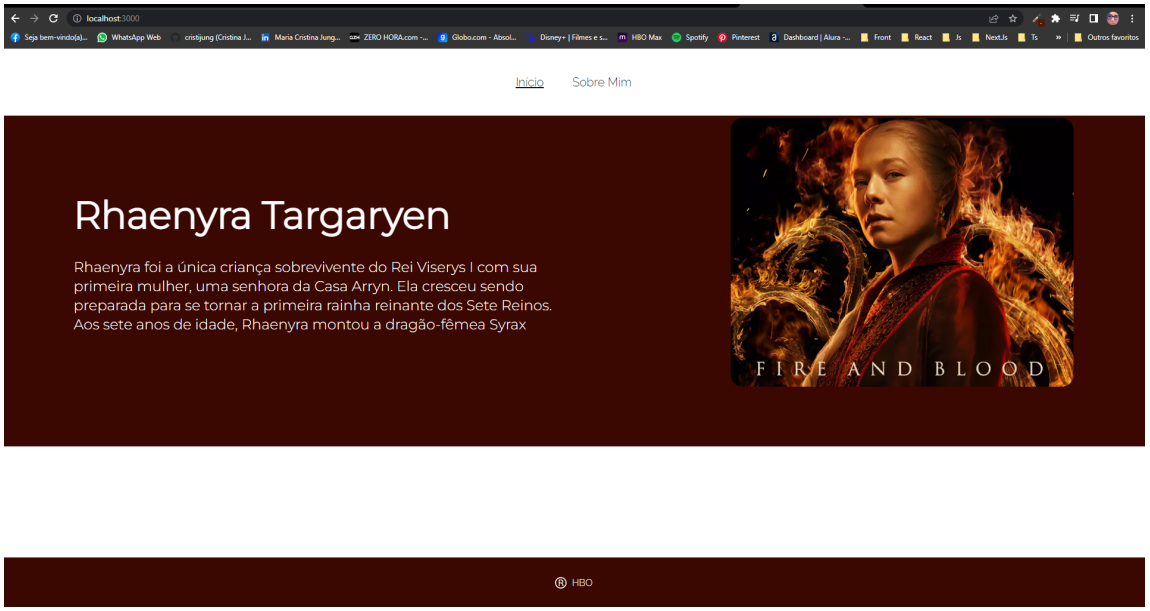
```
You, há 14 minutos | 1 author (You)
import Banner from "components/Banner";

export default function PaginaPadrao() {
    return (
        <main>
            <Banner />



        </main>
    )
}       You, anteontem • inicial
```

Agora no arquivo routes.js, iremos configurar as rotas aninhadas e importar o nosso componente PaginaPadrao.

```
You, há 1 segundo | 1 author (You)
import PaginaPadrao from "components/PaginaPadrao";
import Rodape from "components/Rodape";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Menu from "./components/Menu";
import Inicio from "./pages/Inicio";
import SobreMim from "./pages/SobreMim";
//import PaginaPadrao from "./components/PaginaPadrao";


function AppRoutes() {
  return (
    <>
    <BrowserRouter>
        <Menu />
      <Routes>
        <Route path="/" element={<PaginaPadrao/>}>
          <Route path="/" element={<Inicio />} />
          <Route path="/SobreMim" element={<SobreMim />} />
        </Route>

        <Route path="*" element={<div>Erro 404 - Página não encontrada.</div>} />
      </Routes>
      <Rodape />
    </BrowserRouter>
    </>
  );
}

export default AppRoutes;       You, anteontem • inicial

```

Incluimos outro Route onde definimos que a rota principal será o componente <PaginaPadrao /> e as demais rotas serão filhas desta principal

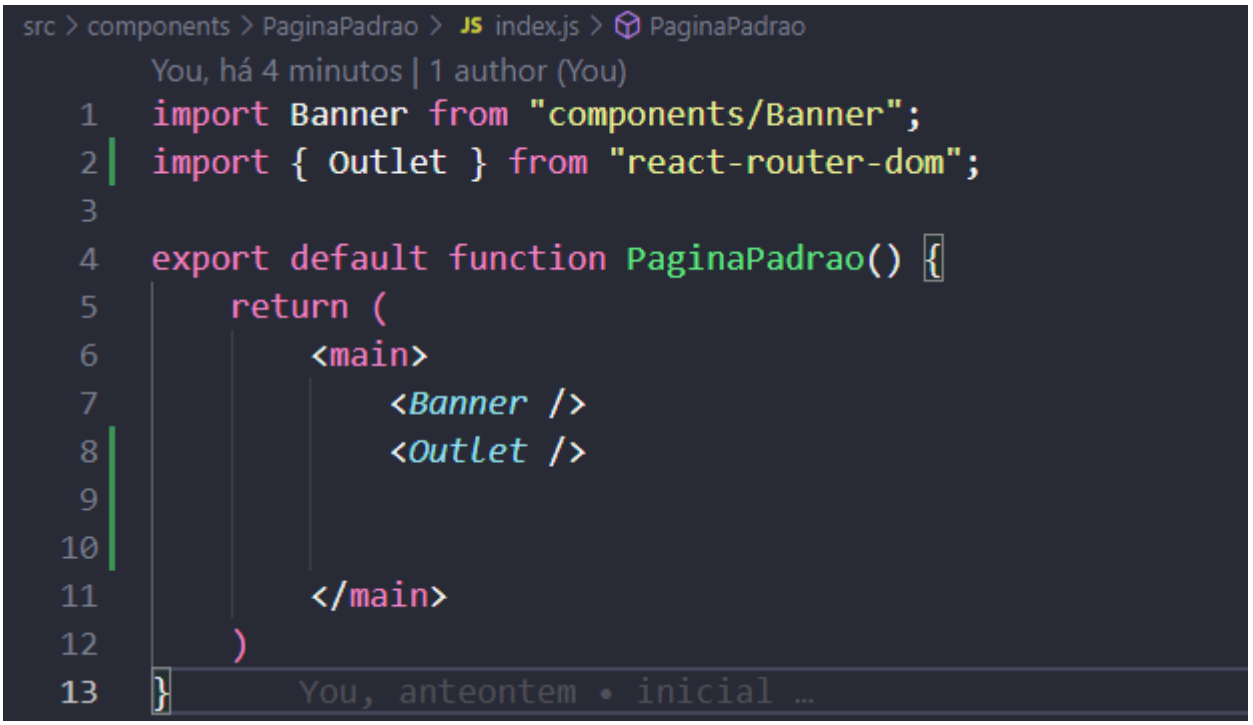
Agora vamos para o PaginaPadrao e vamos configurar a Rota naquele arquivo: Olhando no navegador, encontramos isso ….



No index da PaginaPadrao ….

Importamos o componente <Outlet /> → Um <Outlet> **deve ser usado em elementos de rota pai para renderizar seus elementos de rota filho.** Isso permite que a interface do usuário aninhada apareça quando as rotas filhas são renderizadas. Se a rota pai corresponder exatamente, ela renderizará uma rota de índice filho ou nada se não houver nenhuma rota de índice.

Outlet v6.4.3 | React Router



Está pronto o nosso Roteamento Aninhado.

Componente do React Hook Form

O React Hook Forms é uma biblioteca que auxilia a organizar e padronizar as validações dos formulários por toda a aplicação. Ele também cria validação de formulário simples, alinhado com os existentes dentro do próprio HTML, as validações suportadas são:

- required;
- min;
- max;
- maxlength;
- minlength;
- pattern;
- validate.

O React Hook Form adota o uso de entradas não controladas em vez de depender do estado. Essa abordagem torna os formulários mais eficientes e reduz o número de novas renderizações desnecessárias.

Documentação do React Hook Form: https://react-hook-form.com/

Após a criação e desenvolvimento do projeto, para a instalação do React Hook Form, digite o comando:

```
npm install react-hook-form
```

Para utilizar o react hook form precisamos importar uma funcionalidade chamada useForm:

```
import { useForm } from "react-hook-form";
```

Depois que importamos o useForm da biblioteca react-hook-form podemos utilizar o processo de desestruturação - conceito da linguagem JavaScript, para extrair duas funcionalidades chamadas register e handleSubmit.

```
const { register, handleSubmit } = useForm();
```

O **register** pode ser usado para personalizar a validação em qualquer campo como por exemplo em casos comuns que precisamos definir que um campo nome tenha um mínimo X de caracteres.
Podemos inclusive passar o register para o input através da propriedade **ref.**
Dessa forma estamos dizendo que este campo está disponível para validação e assim permite que o seu valor seja rastreado para alterações.

```
<input name="ultimo_nome" ref = { register ({ pattern: /^[A-Za-z]+$/i }) }
/>
```

Observe que foi passado para o register e o regex que vai fazer com que o campo de input siga o padrão passado a ele.

Além do **register** também extraímos o método **handleSubmit**. É ele que vai passar os dados do formulário quando a validação do formulário for bem-sucedida. **Ele é inserido na propriedade onSubmit do formulário**

```
<form onSubmit = { handleSubmit(onSubmit) } > /*
Dados do formulário */ </form>
```

Repare que o método **handleSubmit** recebe uma função como parâmetro. **É essa função que será chamada caso a validação seja bem-sucedida.**

```
You, há 1 segundo | 1 author (You)
1  import React from "react";
2  import { useForm } from "react-hook-form";
3
4  export default function Opiniao() {
5    const { register, handleSubmit } = useForm();
6    const onSubmit = (data) => console.log(data);
7
8    return (
9      <form onSubmit = { handleSubmit(onSubmit) } >
10         <input name="primeiro_nome"  ref = { register({ required: true, maxLength: 20 }) } />
11         <input name="email" type="email" ref = { register({ min: 18, max: 99 }) } />
12         <input name="content" ref = { register({ maxLength: 200 }) } />
13         <input type="submit" />
14       </form>
15   );
16 }
```

Vamos verificar as validações simples no formulário acima:

- **Linha 10:** o campo primeiro-nome precisa ter o tamanho máximo de 20 caracteres e é de preenchimento obrigatório.
- **Linha 11:** o campo email terá um tamanho definido porém o preenchimento não é obrigatório (é um exemplo), só para mostrar que aqui não utilizamos required: true como foi feito no campo primeiro-nome.
- **Linha 12:** no campo content vamos apenas definir um valor máximo que ele pode receber.
- **Linha 13:** inserimos o botão do formulário que ao ser clicado vai acionar o

evento onSubmit e com isso vai chamar o método handleSubmit.

Exibindo erros no formulário:

Ao validar os campos queremos que o usuário saiba qual campo ele preencheu indevidamente e isso é importante pois faz parte da experiência do usuário. Para isso podemos instalar a biblioteca **@hookform/error-message** executando o comando:

```
npm install @hookform/error-message
```

Em cada input vamos inserir um texto dentro de required. É esse texto que ficará disponível na variável **errors** → É o texto que irá retornar quando houver algum erro que caia na validação.

Para exibir os erros na tela. Usaremos o componente ErrorMessage.

Depois de cada input vamos inserir o componente ErrorMessage para que, caso haja um erro de validação, a mensagem seja exibida.

Para cada componente ErrorMessage passamos dois parâmetros:

1. **errors** que vai receber a mensagem de erro representada pela variável errors (extraída de useForm);
2. **name** que é o mesmo do input ao qual queremos exibir a mensagem caso a validação não seja bem sucedida.

Observe a próxima imagem:

```
You, há 1 segundo | 1 author (You)
import React from "react";
import { useForm } from "react-hook-form";
import { ErrorMessage } from "@hookform/error-message";
export default function Opiniao() {
  const { register, handleSubmit, errors } = useForm();
  const onSubmit = (data) => console.log(data);

  return (
    <form onSubmit = { handleSubmit(onSubmit) } >
        <input name="primeiro_nome"
            ref = {
                register({
                    required: "O campo precisa ter no máximo 20 caracteres",
                    maxLength: 20 }) }
        />
        <ErrorMessage errors = { errors } name="primeiro_nome" />
        <input name="email"
            type="email"
            ref = {
                register({
                    required: "Email inválido",
                    min: 18, max: 99 }) }
        />
        <ErrorMessage errors = { errors } name="email" />
        <input name="content"
            ref = {
                register({
                    required: "Máximo de 200 caracteres",
                    maxLength: 200 }) }
        />
        <ErrorMessage errors = { errors } name="content" />      You, há 17 segundo
        <input type="submit" />
      </form>
  );
}
```

VEM SER DBC

Validação com Yup - React Hook Form

O uso do Yup em projetos React torna a vida muito mais fácil para validar os dados de aplicativos que consomem e manipulam informações que envolvam a lógica dos negócios.

Com o Yup, é possível criar um objeto formatado que se parece com o esquema pretendido para um objeto e, em seguida, usamos as funções do utilitário Yup para verificar se os objetos de dados correspondem a esse esquema e assim, validando-os conforme a necessidade da aplicação.

Instalação do Yup:

```
npm install @hookform/resolvers yup
```

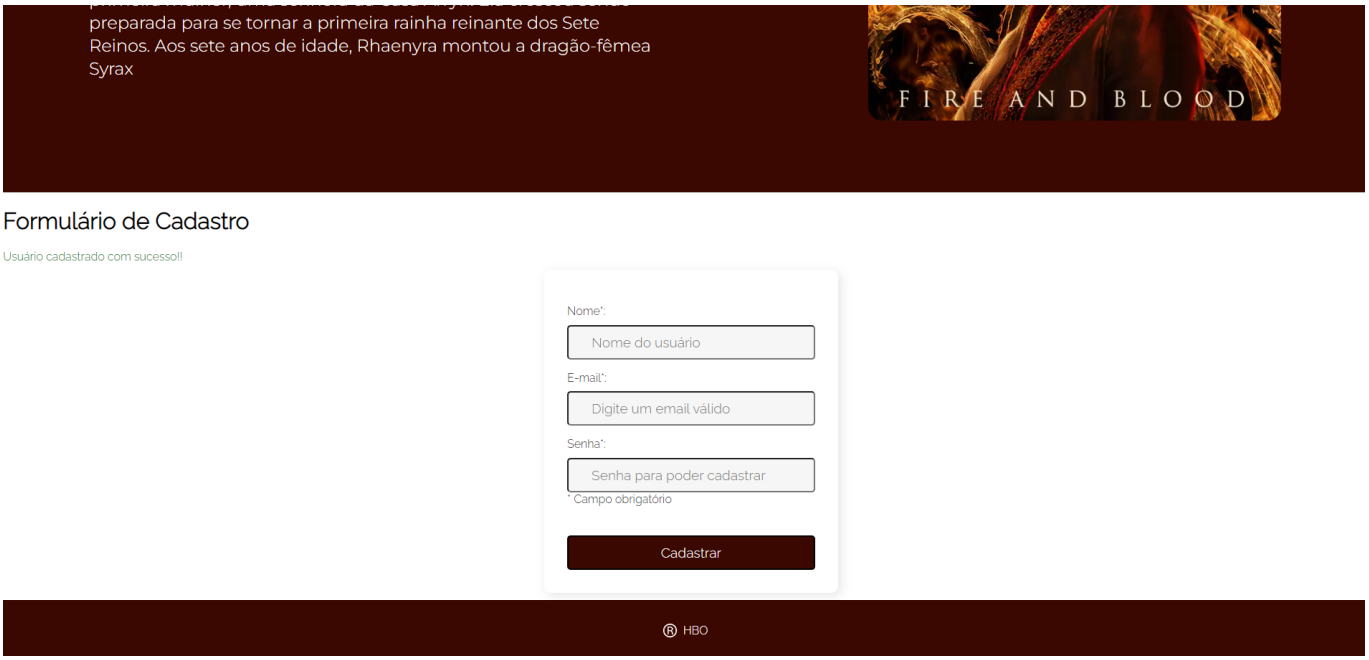
E para a importação no objeto ….

```
import  *  as yup from ‘yup’;
```

Documentação do Yup:

GitHub - jquense/yup: Dead simple Object schema validation

O código do formulário está declarado dentro do index.js no componente Opiniao.
O CSS-modules está no arquivo de estilização Opiniao.module.css;



Código do formulário com validação Yup:

```
import styles from './Opiniao.module.css';
import React, { useState } from 'react';
import { useForm } from "react-hook-form";
import * as yup from 'yup';


function Opiniao () {
    const [user, setUser] = useState({
        name: '',
        email: '',
        password: ''
    });

    const [status, setStatus] = useState({
        type: '',
        mensagem: ''
    });

    //Aqui recebe os dados do form
    const valueInput = e => setUser({ ...user, [e.target.name]: e.target.value });

    //Neste local podemos enviar os dados para o back end ou mesmo
    // para um srv fake jason
    const addUser = async e => {
        e.preventDefault();  //evento padrão para não ter q recarregar todo o form

        if (!(await validate())) return;
        const saveDataForm = true;
```

```
        if (saveDataForm) {
            setStatus({
                type: 'success',
                mensagem: "Usuário cadastrado com sucesso!!"
            });
            setUser({
                name: '',
                email: '',
                password:''
            });
        } else {
            setStatus({
                type: 'error',
                mensagem: "Erro! Usuário não foi cadastrado!"
            });
        }
    }
    //schema assíncrono para fazer a validação dos inputs
    //foi utilizado um Try/Catch para dar um frufru no tratamento de erros
    async function validate(){
        let schema = yup.object().shape({
          password: yup.string("Erro: Necessário preencher o campo senha!")
            .required("Erro: Necessário preencher o campo senha!")
            .min(6,"Erro: A senha deve ter no mínimo 6 caracteres!"),

            email: yup.string("Erro: Necessário preencher o campo e-mail!")
            .required("Erro: Necessário preencher o campo e-mail!")
            .email("Erro: Necessário preencher o campo com e-mail válido!"),

            name: yup.string("Erro: Necessário preencher o campo nome!")
            .required("Erro: Necessário preencher o campo nome!")
        });

        try{
          await schema.validate(user);
          return true;
        }catch (err) {
          setStatus({
            type: 'error',
            mensagem: err.errors
          });
          return false;
        }
      }


      //abaixo, o formulário
    return(
        <>
        <h1> Formulário de HOTD</h1>

        {status.type === 'success' ? <p style={{ color: "#00ff00" }}>{status.mensagem}</p> : ""}
        {status.type === 'error' ? <p style={{ color: "#ff0000" }}>{status.mensagem}</p> : ""}

        <div className={styles.formu}>
        <form onSubmit={addUser}>
            <div className={styles.field}>
                <label className={styles.label}>Nome*: </label>
                <input type="text" name="name" placeholder="Nome do usuário" onChange={valueInput}
value={user.name}  />
            </div>
            <div className={styles.field}>
                <label className={styles.label}>E-mail*: </label>
                <input type="email" name="email" placeholder="Digite um email válido"
onChange={valueInput}  value={user.email} />
            </div>
            <div className={styles.field}>
```

```
                <label className={styles.label}>Senha*: </label>
                <input type="password" name="password" placeholder="Senha para poder cadastrar"
autoComplete="on" onChange={valueInput} value={user.password} />
            </div>
        * Campo obrigatório<br /><br />

        <button type="submit">Cadastrar</button>
      </form>
      </div>

        </>
    );
}

export default Opiniao;
```