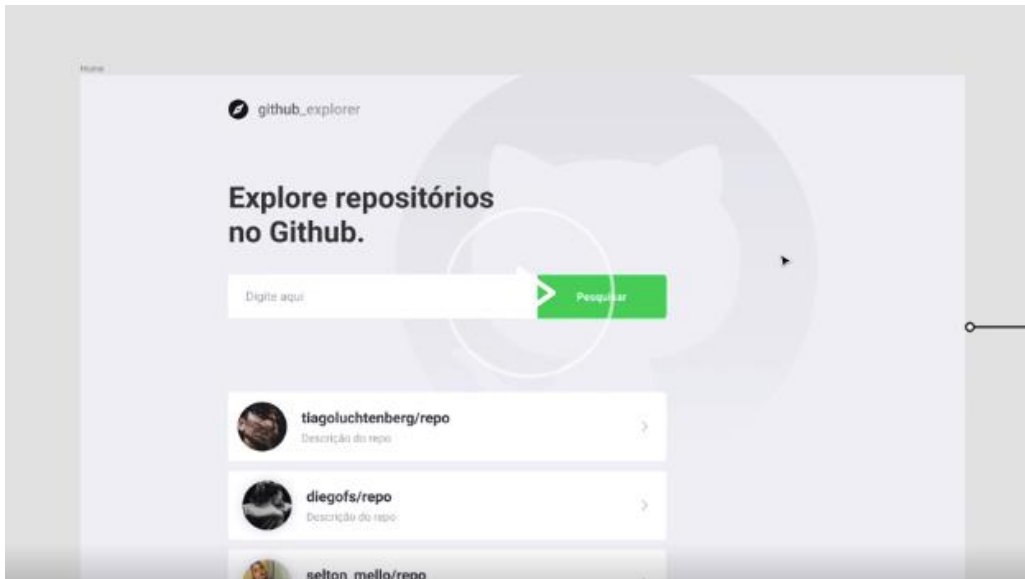


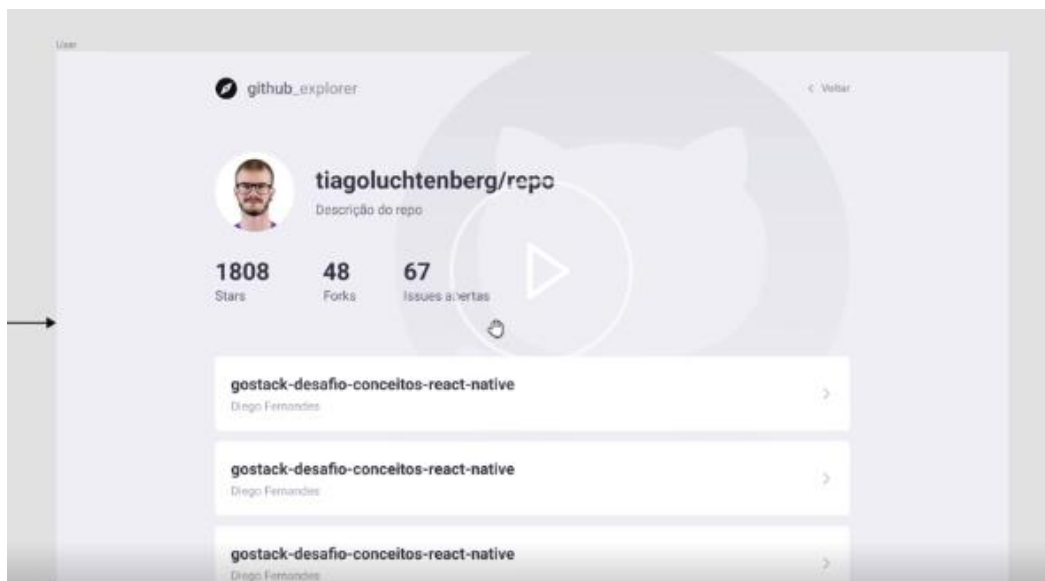
# Criando uma aplicação completa com React Native

Utilizaremos uma API pública do GitHub (que será nosso backend), para criar uma aplicação com o nome de GitHub Explorer. Ela fará buscas nos repositórios do GitHub a partir de uma tela de consulta.

## Primeira tela (tela principal)



## Segunda tela (será aberta ao clicar em um dos elementos da lista da primeira tela)



## 1. Criando a estrutura básica do projeto

1.1. Na pasta onde deseja criar o seu projeto, digite o comando abaixo:

```
create-react-app meu-github-explorer --template=typescript
```

Este comando já cria a estrutura básica de um projeto React, já disponibilizando o uso do TypeScript. Já vem com o webpack e o babel instalado.

1.2. Abra a pasta do projeto no VsCode e em seguida façamos as seguintes alterações na estrutura do projeto:

- Delete os arquivos: readme.md.
- Dentro da pasta SRC, delete os arquivos: App.css, App.test.tsx, index.css, logo.svg e serviceWorker.ts

1.3. Na pasta “SRC”, abra o arquivo “index.tsx” e remova os conteúdos marcados abaixo:

```
src > index.tsx
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import * as serviceWorker from './serviceWorker';
6
7 ReactDOM.render(
8   <React.StrictMode>
9     <App />
10   </React.StrictMode>,
11   document.getElementById('root')
12 );
13
14 // If you want your app to work offline and load fast
15 // unregister() to register() below. Note this comes
16 // pitfalls.
17 // Learn more about service workers: https://bit.ly/
18 serviceWorker.unregister();
```

1.4. Ainda nesta pasta, deixe o arquivo “App.tsx”, com apenas este conteúdo:

```
src > App.tsx > ...
1 import React from 'react';
2
3 function App() {
4   return (
5     <h1>Hello World</h1>
6   );
7 }
8
9 export default App;
```

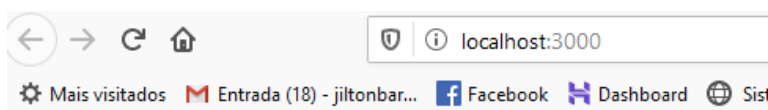
1.5. Na pasta “public”, exclua os arquivos: favicon.ico, logo192.png, logo512.png e manifest.json.

1.6. Ainda nesta pasta, abra o arquivo “index.html” e remova as linhas que fazem referência aos arquivos que foram deletados e mais alguns conteúdos. Ele deve ficar com o seguinte conteúdo:

```
public > index.html > ...
1 <!DOCTYPE html>
2 <html Lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <meta name="viewport" content="width=device-width,
6       initial-scale=1" />
7     <meta name="theme-color" content="#3A3A3A" />
8     <title>Meu Github Explorer</title>
9   </head>
10   <body>
11     <noscript>You need to enable JavaScript to run this app.</noscript>
12     <div id="root"></div>
13   </body>
14 </html>
```

## 2. Inicializando nosso servidor

No terminal do VsCode digite o comando: **yarn start** (o browser será aberto automaticamente, com a mensagem Hello World, com a URL localhost:3000)



# Hello World

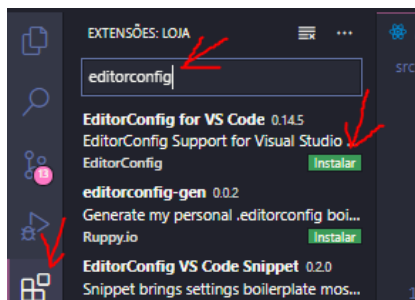
Quando executamos este comando (yarn start), o React executa o script que está dentro do arquivo “package.json”. Dentro deste arquivo também temos o script “build”, que é utilizado quando for necessário gerar o arquivo que será publicado no servidor. Dos scripts que estão neste arquivo, recomenda-se não utilizar o

“eject”, porque ele vai gerar vários arquivos na estrutura do projeto, que só faria sentido se quisesse fazer uma customização mais complexa no projeto. É muito raro precisar fazer isso.

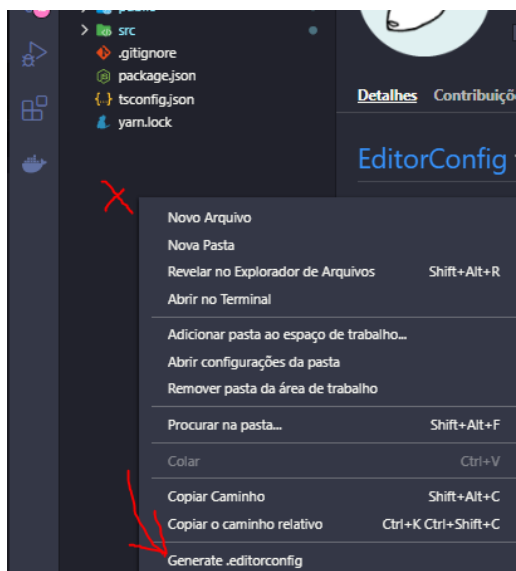
O arquivo “index.tsx”, da pasta “src” é responsável por pegar o conteúdo do arquivo “index.tsx” desta mesma pasta e jogar dentro do arquivo “index.html”, que está dentro da pasta “public”. Para conseguir executar este processo, o arquivo “index.tsx”, utiliza a biblioteca “react-dom”. O “App.tsx” é o componente principal da nossa aplicação.

### 3. Configurando nosso editor

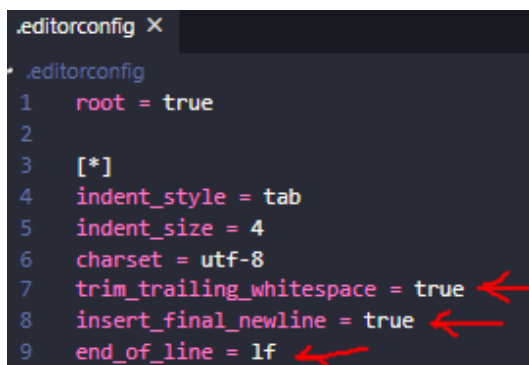
#### 3.1. Instale a extensão EditorConfig do VsCode



#### 3.2. Clique o botão direito nesta área e selecione a opção “Generate .editorconfig”. Esta ação vai gerar este arquivo na pasta raiz do projeto.



#### 3.3. Abra este arquivo e mantenha ele com o seguinte conteúdo:



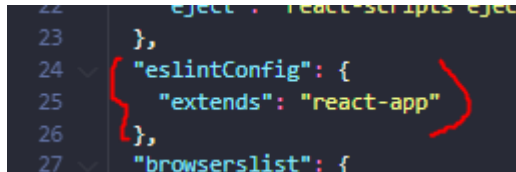
#### 4. Configurando o ESLint (Obs.: este recurso deixa o VsCode mais pesado)

O que você vai ver aqui, são padrões de configuração utilizadas pelo professor, que aprimorou durante toda a sua experiência de desenvolvimento com React.

4.1. Instale o Eslint pelo terminal do VsCode, somente para o ambiente de desenvolvimento:

**yarn add eslint -D**

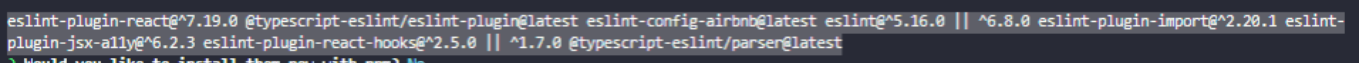
4.2. No arquivo “package.json”, remova a referência ao Eslint, que é uma configuração padrão.



```
22   "scripts": {
23     "start": "react-scripts start",
24     "build": "react-scripts build",
25     "test": "react-scripts test",
26     "eject": "react-scripts eject"
27   },
28   "eslintConfig": {
29     "extends": "react-app"
30   },
31   "browserslist": {
```

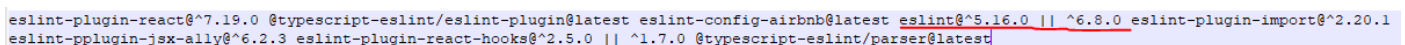
4.3. No terminal execute o comando: yarn eslint --init (ele vai fazer uma série de perguntas)

- Na primeira pergunta selecione a última opção
- Na segunda pergunta selecione a primeira opção (javascript modules (import/export))
- A resposta da terceira pergunta é o React.
- Na quarta pergunta responda Yes para Typescript.
- Na quinta pergunta selecione “browser”
- Para a pergunta “user a popular style guide” selecione “airbnb”
- Na pergunta seguinte selecione a opção JSON
- Na pergunta seguinte responda NO (não utilizaremos o npm)
- Em seguida ele vai gerar uma lista de dependências. Selecione e copie esta lista



```
eslint-plugin-react@^7.19.0 @typescript-eslint/eslint-plugin@latest eslint-config-airbnb@latest eslint@^5.16.0 || ^6.8.0 eslint-plugin-import@^2.20.1 eslint-plugin-jsx-a11y@^6.2.3 eslint-plugin-react-hooks@^2.5.0 || ^1.7.0 @typescript-eslint/parser@latest
? Would you like to install them now with npm? (y/n)
```

4.4. O próximo passo é, o terminal digite yarn add e cole aqui o conteúdo que você copiou. Antes de executar o comando, faça nele as seguintes alterações. Remova as partes sublinhadas e acrescente -D no final.

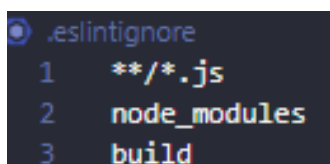


```
eslint-plugin-react@^7.19.0 @typescript-eslint/eslint-plugin@latest eslint-config-airbnb@latest eslint@^5.16.0 || ^6.8.0 eslint-plugin-import@^2.20.1
eslint-plugin-jsx-a11y@^6.2.3 eslint-plugin-react-hooks@^2.5.0 || ^1.7.0 @typescript-eslint/parser@latest
```

Este é o comando:

```
yarn add eslint-plugin-react@^7.19.0 @typescript-eslint/eslint-plugin@latest eslint-config-airbnb@latest
eslint-plugin-import@^2.20.1 eslint-plugin-jsx-a11y@^6.2.3 eslint-plugin-react-hooks@^2.5.0 @typescript-
eslint/parser@latest -D
```

4.5. Crie o arquivo “.eslintignore” na pasta raiz, com o seguinte conteúdo:



```
.eslintignore
1  **/*.js
2  node_modules
3  build
```

Com isso, o Eslint ignorará todos os arquivos “.js” em qualquer pasta do projeto. Ignorará todos os arquivos da pasta node\_modules e também os da pasta build (onde serão gerados nossos arquivos para o ambiente de produção).

4.6. Abra o arquivo “eslintrc.json” e faça as seguintes alterações:

- Em extends, inclua: "plugin:@typescript-eslint/recommended"
- Em plug-ins, inclua: "react-hooks",

- Nas rules, acrescente o seguinte conteúdo:

```
"rules": {
  "react-hooks/rules-of-hooks": "error",
  "react-hooks/exhaustive-deps": "warn",
  "react/jsx-filename-extension": [1, {"extensions": [".tsx"]}],
  "import/prefer-default-export": "off",
  "import/extensions": [
    "error",
    "ignorePackages",
    {
      "ts": "never",
      "tsx": "never"
    }
  ]
}
```

4.7. Agora execute o seguinte comando: **yarn add eslint-import-resolver-typescript -D**

4.8. Ainda, terminado as rules acrescente logo abaixo o “settings”:

```
"settings": {
  "import/resolver": {
    "typescript": {}
  }
}
```

Isto possibilitará ao Eslint reconhecer o Typescript.

## 5. Adicionando o Prettier ao projeto

yarn add prettier eslint-config-prettier eslint-plugin-prettier -D

5.1. Acrescente estas duas linhas à tag “extends” do arquivo “.eslintrc.json”

```
"prettier/@typescript-eslint",
"plugin:prettier/recommended"
```

5.2. Na tag “plugins” acrescente também o prettier

```
"prettier"
```

5.3. E na tag “rules”, a seguinte linha:

```
"prettier/prettier": "error",
```

5.4. Crie o arquivo “prettier.config.js” na pasta raiz do projeto, com o seguinte conteúdo:

```
module.exports = {
  singleQuote: true,
  trailingComma: 'all',
  allowParens: 'avoid'
}
```

## 6. Criando as rotas

Nosso aplicativo terá duas telas, então criaremos a rota para fazer esta navegação. Utilizaremos a biblioteca “react-router-dom” para esta funcionalidade.

yarn add react-router-dom

yarn add @types/react-router-dom -D

6.1. Crie a pasta “pages” dentro da pasta “src”.E dentro dela, a pasta “dashborad”, com o arquivo “index.tsx”, com o seguinte conteúdo:

```
src > pages > dashboard > index.tsx > ...
1  import React from 'react';
2
3  const Dashboard: React.FC = () => {
4    return <h1>Dashboard</h1>
5  };
6
7  export default Dashboard;
```

**Linha 3** – criamos uma variável do tipo “React.FC”. FC é a abreviação de Function Component.

- 6.2. Ainda dentro da pasta “src”, crie a pasta “repository” e dentro dela o arquivo “index.tsx”, com o mesmo conteúdo do arquivo anterior, apenas modificando o nome Dashboard, para “Repository”:

```
src > pages > repository > index.tsx > ...
1  import React from 'react';
2
3  const Repository: React.FC = () => {
4    return <h1>Repository</h1>
5  };
6
7  export default Repository;
```

- 6.3. Crie a pasta “routes” dentro da pasta “src”. E dentro dela o arquivo “index.tsx”, com o seguinte conteúdo:

```
index.tsx x
src > routes > index.tsx > ...
1  import React from 'react';
2  import { Switch, Route } from 'react-router-dom';
3
4  import Dashboard from '../pages/dashboard';
5  import Repository from '../pages/repository';
6
7  const Routes: React.FC = () => (
8    <Switch>
9      <Route path="/" exact component={Dashboard} />
10     <Route path="/repository" exact component={Repository} />
11   </Switch>
12 )
13
14 export default Routes;
```

**Linha 8** – Se não tivesse o “switch”, as duas páginas seriam exibidas na mesma tela.

**Linha 9** – considerando que a página “Dashboard” será a página principal (ou primeira página), ela deverá ser acessada somente com a barra após a URL. É necessário o termo “exact” para que o roteador reconheça, que a página principal será acessada digitando somente a barra no final da URL.

- 6.4. O Conteúdo do nosso arquivo App.tsx, agora deverá ficar da seguinte forma:

```
App.tsx x
src > App.tsx > ...
1  import React from 'react';
2  import { BrowserRouter } from 'react-router-dom';
3  import Routes from './routes';
4
5  const App: React.FC = () =>
6    <BrowserRouter>
7      <Routes />
8    </BrowserRouter>
9  export default App;
```

**Linha 5** – esta é uma forma abreviada de criar as funções da forma com fizemos anteriormente. Isso somente quando se trata de funções simples, sem conteúdo.

## 7. Estilizando os componentes

No terminal execute o comando: **yarn add styled-components**

E também **yarn add @types/styled-components -D**

Esta biblioteca, isola os arquivos CSS por componente, de forma que o CSS de um componente não interfere no CSS de outro componente. Mas isso não impede de podermos criar também CSSs globais.

- 7.1. Criaremos um arquivo com o nome de “styles.ts” dentro da pasta “dashboard”. Se criássemos como “styles.css” ele se tornaria um css global na aplicação.

O conteúdo inicial deste arquivo será o seguinte:

```
src > pages > dashboard > styles.ts > ...
1  import styled from 'styled-components';
2
3  export const Title = styled.h1`
4    font-size: 48px;
5    color: #3A3A3A;
6  `;
```

Para ficar assim com estas cores, é necessário instalar o plugin “styled-components” do vscode.

- 7.2. Agora, no arquivo “index.ts”, da pasta “dashboard”, faça a importação deste arquivo de estilos e altere a tag <h1> para <Title>, que é o estilo que criamos na linha 3 acima.

```
src > pages > dashboard > index.tsx > ...
1  import React from 'react';
2
3  import { Title } from './styles';
4
5  const Dashboard: React.FC = () => {
6    return <Title>Explore repositórios no Github</Title>
7  };
8
9  export default Dashboard;
```

A utilização do “styled-components”, possibilita que o CSS no React, tenha a mesma sintaxe do CSS do html.

- 7.3. Criaremos agora nosso estilo css glogal. Na pasta “src”, crie a pasta “styles” e dentro dela o arquivo “global.ts”, com o seguinte conteúdo:

```
src > styles > global.ts > ...
1  import { createGlobalStyle } from 'styled-components';
2
3  export default createGlobalStyle`
4
5      *{
6          margin: 0;
7          padding: 0;
8          outline: 0;
9          box-sizing: border-box;
10     }
11
12     body {
13         background: #F0F0F5;
14     }
15 `;
```

Este arquivo nós importaremos dentro do arquivo App.ts da nossa pasta “src”, que ficará da seguinte forma:

```
src > App.tsx > ...
1  import React from 'react';
2  import { BrowserRouter } from 'react-router-dom';
3  import GlobalStyle from './styles/global';
4  import Routes from './routes';
5
6  const App: React.FC = () =>
7  {
8      <BrowserRouter>
9          <Routes />
10      </BrowserRouter>
11      <GlobalStyle />
12  }
13  export default App;
```

**Linha 3** – importamos o GlobalStyle que criamos acima.

**Linhas 7 e 12** – Criamos estas tag vazias, por uma necessidade do React manter o conteúdo sempre entre tags.

**Linha 10** – Aplicando o GlobalStyle dessa forma, ele envolve todo o conteúdo da página.

A fonte utilizada no layout do projeto é a ‘Roboto’. Podemos pegá-la no [fonts.google.com/specimen/Roboto](https://fonts.google.com/specimen/Roboto)

- 7.4. Selecione a Regular 400 e a bold 700, clique na aba “embeded” e copie o código.

<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">

- 7.5. Cole este código no head do arquivo index.html, da pasta public.

7.6. Acrescente esta fonte ao arquivo de css global. Ele deverá ficar da seguinte forma:

```
src > styles > global.ts > ...
1 import { createGlobalStyle } from 'styled-components';
2 import githubBackground from '../assets/github-background.svg';
3
4 export default createGlobalStyle`
5   *{
6     margin: 0;
7     padding: 0;
8     outline: 0;
9     box-sizing: border-box;
10  }
11
12  body {
13    background: #F0F0F5 url(${githubBackground}) no-repeat 70% top;
14    -webkit-font-smoothing: antialiased;
15  }
16
17  body, input, button { ←
18    font: 16px Roboto, sans-serif;
19  }
20
21  #root { /* div da tela principal */
22    max-width: 960px;
23    margin: 0 auto; /*centraliza*/
24    padding: 40px 20px;
25  }
26
27  button {
28    cursor: pointer;
29  }
30
31 `;
```

**Linha 2** – importamos a imagem que utilizemos como fundo de tela.

**Linha 14** – aplicamos a imagem ao fundo da tela, com 70% de margem esquerda e alinhada com o topo.

**Linha 19** – aplicamos a fonte “roboto” que importamos do fonts.google (caso ocorra problema ele aplicará a fonte sans-serif)

**Linha 22** – “root” é o ID da div da nossa tela principal. Utiliza-se o **max-width** para que a largura possa se adaptar à tela de um celular.

7.7. Colocar o logo do Github, no topo da tela.

No arquivo index.tsc, da pasta “dashboard”, faça o import da imagem ‘logo.svg’.

```
src > pages > dashboard > index.tsx > ...
1 import React from 'react';
2 import logoImg from '../assets/logo.svg';
3
4 import { Title } from './styles';
5
6 const Dashboard: React.FC = () => {
7   return(
8     <>
9       <img src={logoImg} alt="Github Explorer"/>
10      <Title>Explore repositórios no Github</Title>
11    </>
12  );
13 };
14
15 export default Dashboard;
```

**Linha 2** – importamos a imagem da logo.

**Linha 7** – colocamos agora todo o conteúdo entre parênteses. Quando temos somente um elemento não precisamos dos parênteses. Antes tínhamos somente o <Title>.

**Linha 8** – Quando temos dois elementos, um embaixo do outro, precisamos ter a tag ‘fragments’ (<>), que fecha na linha 11.

7.8. No arquivo de estilos da pasta “dashboard”, acrescentamos mais duas linhas:

**Linha 6** – Limitamos a largura do h1 para 450px, de modo que tenha uma quebra de linha, exatamente como no layout.

**Linha 7** – Ajuste de entrelinha.

```
src > pages > dashboard > styles.ts > ...
1 import styled from 'styled-components';
2
3 export const Title = styled.h1`
4   font-size: 48px;
5   color: #3A3A3A;
6   max-width: 450px;
7   line-height: 56px;
8
9   margin-top: 80px;
10 `;
```



## 8. Acrescentando o campo de pesquisa na tela principal

```
src > pages > dashboard > index.tsx > ...
1  import React from 'react';
2  import logoImg from '../assets/logo.svg';
3
4  import { Title, Form } from './styles';
5
6  const Dashboard: React.FC = () => {
7    return(
8      <>
9        <img src={logoImg} alt="Github Explorer"/>
10       <Title>Explore repositórios no Github</Title>
11
12       <Form action="">
13         <input placeholder="Digite o nome do repositório"/>
14         <button type="submit">Pesquisar</button>
15       </Form>
16     </>
17   );
18 };
19
20 export default Dashboard;
```

**Linha 4** – importamos o componente Form. Mas antes precisamos criá-lo, no arquivo “styles.ts” da pasta do dashboard.

**Linha 12** – Utilizamos o componente form. Observe que, por se tratar de um componente, utiliza-se letra maiúscula (Form)

### 8.1. Estilizando o campo de pesquisa

- Instale a biblioteca “polished”: **yarn add polished** (para aplicarmos sobreamento a um botão)
- Crie o componente Form dentro do arquivo de estilos da pasta dashboard, com o seguinte conteúdo:

```
13  export const Form = styled.form`
14    margin-top: 40px;
15    max-width: 700px;
16    display: flex;
17
18    input {
19      flex: 1;
20      height: 70px;
21      padding: 0 24px;
22      border: 0;
23      border-radius: 5px 0 0 5px;
24      color: #3a3a3a;
25
26      &::placeholder{
27        color: #a8a8b3;
28      }
29    }
30
31    button {
32      width: 210px;
33      height: 70px;
34      background: #04D361;
35      border-radius: 0px 5px 5px 0px;
36      border: 0;
37      color: #FFF;
38      font-weight: bold;
39
40      &:hover{ /*a mesma coisa que button:hover*/
41        background: ${shade(0.2, '#04D361')}
42      }
43    }
44  `;
```

**Linha 13** – criamos nosso componente Form. Dentro deste componente criamos o css dos seus elementos internos (input e button).

**Linha 16** – coloca um elemento ao lado do outro. No caso o input ao lado do button.

**Linha 19** – coloca a largura do input limitada à largura do button.

**Linha 26** – é a mesma coisa que input:placeholder

**Linha 40** – é a mesma coisa que button:hover

**Linha 41** – O recurso \${..} é a maneira de implementarmos javascript dentro do CSS. A função “shade()” é uma função da biblioteca “polished”, que acabamos de instalar.

### 8.2. Estilizando a lista de repositórios

Observe que para cada item da nossa lista de repositórios, temos um ícone clicável à direita. Para implementarmos este ícone utilizaremos a biblioteca “react-icons”

**yarn add react-icons**

- Em seguida importamos esta biblioteca no arquivo index.tsx da pasta “dashboard”. Utilizaremos somente o elemento FiArrowRight desta biblioteca, que é justamente a seta para a direita.

```
import { FiChevronRight } from 'react-icons/fi';
```

8.3. Agora, nosso arquivo index.tsx, ficará da seguinte forma:

```
src > pages > dashboard > index.tsx > ...
1  import React from 'react';
2  import { FiChevronRight } from 'react-icons/fi';
3  import logoImg from '../../assets/logo.svg';
4
5  import { Title, Form, Repositories } from './styles';
6
7  const Dashboard: React.FC = () => {
8    return(
9      <>
10       <img src={logoImg} alt="Github Explorer"/>
11       <Title>Explore repositórios no Github</Title>
12
13       <Form action="">
14         <input placeholder="Digite o nome do repositório"/>
15         <button type="submit">Pesquisar</button>
16       </Form>
17
18       <Repositories>
19         <a href="teste">
20           
22           <div>
23             <strong>Ilton Barbosa Repository</strong>
24             <p>teste descrição repositório</p>
25             <FiChevronRight size={20}/>
26           </div>
27         </a>
28       </Repositories>
29     </>
30   );
31 };
```

**Linha 2** – temos o import do react-icons/fi.

**Linha 18** – implementamos o componente Repositories. Lembrando que precisamos criar o estilo para este componente no arquivo de estilos da pasta dashboard.

**Linha 25** – implementação da seta para a direita, conforme está no nosso layout.

Por enquanto estamos utilizando aqui, somente dados estáticos, que serão substituídos por dados dinâmicos quando fizermos a integração com o nosso backend.

8.4. Criando o estilo Repositories no arquivo de estilos da pasta dashboard.

```
46 export const Repositories = styled.div`
47   margin-top: 80px;
48   max-width: 700px;
49
50   a {
51     background: #fff;
52     border-radius: 5px;
53     width: 100%;
54     padding: 24px;
55     display: flex;
56     text-decoration: none;
57
58     align-items: center;
59     transition: transform 0.2s;
60
61     &:hover{
62       transform: translateX(10px);
63     }
64
65     & + a:hover {
66       margin-top: 16px;
67     }
68
69     img {
70       width: 64px;
71       height: 64px;
72       border-radius: 50%;
73     }
74 `;
```

**Linha 46** – criamos o componente de estilo “Repositories”.

**Linha 55** – display: flex para alinhamento dos elementos numa mesma linha.

**Linha 59 e 62** – move o elemento suavemente para a direita, ao passar o mouse sob ele.

**Linha 65** – Se o elemento <a> for precedido de outro elemento <a>, ele acrescentará um espaço entre um e outro, de 16px. Este recurso é utilizado para listas de elementos.

```

75     div {
76       margin-left: 16px;
77
78       strong {
79         font-size: 20px;
80         color: #3D3D4D;
81       }
82
83       p {
84         font-size: 18px;
85         color: #A8A8B3;
86         margin-top: 4px;
87       }
88     }
89
90     svg {
91       margin-left: auto;
92       color: #c9c9d6;
93     }
94   }
95 ;

```

**Linha 75** – esta DIV é a que temos dentro do <a href> na lista de repositórios na nossa página principal.

**Linha 90** – é a seta para a direita que temos na lista de repositórios na página principal.

## 9. Integrando o sistema com a API do Github

9.1. Instale o axios: **yarn add axios**

9.2. Crie a pasta “services” na raiz do projeto, e dentro dela o arquivo “api.ts”, com o seguinte conteúdo:

```

src > services > api.ts > ...
1  import axios from 'axios';
2
3  const api = axios.create({
4    baseURL: 'https://api.github.com'
5  });
6
7  export default api;

```

9.3. Importe este arquivo dentro do arquivo “index.tsx” da pasta “dashboard”.

```
import api from '../services/api';
```

9.4. Importe também para este mesmo arquivo os recursos UseStat e FormEvent, do React que já temos na importação ( linha 1), para que possamos armazenar nossos repositórios em algum lugar.

```
import React, {useState, FormEvent } from 'react';
```

9.5. Criamos também a interface que representará o repositório do github que utilizaremos na nossa aplicação para buscar dados. Desse repositório utilizaremos somente 3 atributos. Veja linhas 8 a 15.

```

src > pages > dashboard > index.tsx > Dashboard
1  import React, {useState, FormEvent } from 'react';
2  import { FiChevronRight } from 'react-icons/fi';
3  import api from '../services/api';
4  import logoImg from '../assets/logo.svg';
5
6  import { Title, Form, Repositories } from './styles';
7
8  interface Repository {
9    full_name: string;
10   description: string;
11   owner: {
12     login: string;
13     avatar_url: string;
14   };
15 }
16
17 const Dashboard: React.FC = () => {

```

O arquivo “index.tsx” da pasta “dashboard”, segue com o seguinte conteúdo:

```
17 const Dashboard: React.FC = () => {
18   const [newRepo, setNewRepo] = useState('');
19   const [repositories, setRepositories] = useState<Repository[]>([]);
20
21   async function handleAddRepository(event: FormEvent<HTMLFormElement>): Promise<void>{
22     event.preventDefault();
23
24     const response = await api.get<Repository>(`repos/${newRepo}`);
25     const repository = response.data;
26     setRepositories([ ... repositories, repository]);
27     setNewRepo('');
28   }
29   return(
30     <>
31     <img src={logoImg} alt="Github Explorer"/>
32     <Title>Explore repositórios no Github</Title>
33
34     <Form onSubmit={handleAddRepository}>
35       <input
36         value={newRepo}
37         onChange={e => setNewRepo(e.target.value)}
38         placeholder="Digite o nome do repositório"/>
39       <button type="submit">Pesquisar</button>
40     </Form>
41
42     <Repositories>
43       {repositories.map(repository => (
44         <a key={repository.full_name} href="teste">
45           <img src={repository.owner.avatar_url} alt={repository.owner.login}/>
46           <div>
47             <strong>{repository.full_name}</strong>
48             <p>{repository.description}</p>
49           </div>
50           <FiChevronRight size={20}/>
51         </a>
52       ))}
53     </Repositories>
54   </>
55 );
56
57 };
58
59 export default Dashboard;
```

**Linha 18** – criamos a variável (*constante*), newRepo e na estrutura dela temos a função “setNewRepo” que é utilizada para atribuímos valor a esta variável (*Princípio da orientação a objeto, nunca setar diretamente um valor a uma variável*).

**Linha 19** – criamos também a variável “repositories” e estamos instanciando nela a classe “useState” da biblioteca React, que importamos na linha 1. Veja que estamos passando como parâmetro a interface que criamos na linha 8. Dessa forma estamos informando que a nossa variável “repositories” receberá conteúdo somente do tipo da nossa interface “repositories”. Veja que ela também tem a função setRepositories().

**Linha 21** – Esta é uma função assíncrona porque ela fará consultas a uma url externa, que é a API do github, que retornará os repositórios da consulta que faremos. Veja que esta função recebe como parâmetro um evento do formulário, que no caso seria quando o usuário clica no botão “submit”.

**Linha 22** – evita que ocorra um reload na página após clicar no botão submit.

**Linha 24** – faz a pesquisa no repositório do github a partir da palavra que o usuário digitou no campo de pesquisa. Veja que a palavra digitada pelo usuário ficou armazenada na variável “newRepo”. O valor foi armazenado nesta variável por meio da ação “onChange”, que temos no formulário (linha 37). Veja também que estamos utilizando o componente “api” que criamos na pasta servisse e que contem a url do repositório no github.

**Linha 25** – o conteúdo retornado da API do github é armazenado na variável “response.data”.

**Linha 26** – Estamos acrescentando o conteúdo retornado da API do github ao final do array “Repositories” que criamos na linha 19.

**Linha 27** – estamos limpando o campo de consulta do nosso formulário.

**Linha 34** – ao clicar no botão “submit”, o formulário executa a função “handleAddRepository” que criamos na linha 19.

**Linha 37** – a palavra digitada no campo de pesquisa é armazenada na variável newRepo.

**Linha 42 a 54** – lista o resultado da consulta.

**Linha 50** – é a seta para a direita que temos na nossa lista de repositórios.

## 10. Lidando com erros

O que fazer se o usuário clicar no botão “submit” sem que tenha digitado nada no campo de pesquisa, ou tenha digitado um termo que não existe no repositório.

10.1. No arquivo de estilo da pasta “dashboard”, crie o seguinte estilo:

```
46 export const Error = styled.span`
47   display: block;
48   color: #c53030;
49   margin-top: 8px;
50 `;
```

A mensagem será exibida em uma tag <span>

10.2. No arquivo index da pasta Dashboard, importe este estilo (linha 6) e acrescente as seguintes alterações:

```
20 → const [inputError, setInputError] = useState('');
21
22 async function handleAddRepository(event: FormEvent<HTMLFormElement>): Promise<void>{
23   event.preventDefault();
24   if(!newRepo){
25     setInputError('Digite o autor/nome do repositório');
26     return;
27   }
28
29 → try{
30   const response = await api.get<Repository>(`repos/${newRepo}`);
31   const repository = response.data;
32
33   setRepositories([ ... repositories, repository]);
34   setNewRepo('');
35   setInputError('');
36   }catch (err){
37 →   setInputError('Erro na busca por este repositório');
38 }
39
40 return(
41   <>
42   <img src={logoImg} alt="Github Explorer"/>
43   <Title>Explore repositórios no Github</Title>
44
45   <Form onSubmit={handleAddRepository}>
46     <input
47       value={newRepo}
48       onChange={e => setNewRepo(e.target.value)}
49       placeholder="Digite o nome do repositório"/>
50     <button type="submit">Pesquisar</button>
51   </Form>
52 → { inputError && <Error>{inputError}</Error>}
```

**Linha 20** – criamos a variável que vai armazenar a mensagem de erro.

**Linha 24** – se o usuário não digitar nada no campo do formulário, a variável “newRepo” estará vazia. Se não colocar o return da linha 26, a função continuará executando as próximas linhas.

**Linhas 29 a 36** – colocamos estas linhas dentro de um try catch. Se ocorrer algum erro na pesquisa, a mensagem a ser exibida é a da **linha 37**.

**Linha 35** – limpamos a mensagem de erro após uma segunda consulta.

**Linha 52** - A mensagem de erro será exibida logo abaixo do campo de pesquisa, somente se a variável “inputError” não estiver vazia.

### 10.3. Aperfeiçoando um pouco mais os erros.

Se quisermos que a nossa caixa do formulário fique em destaque, indicando que o campo precisa ser preenchido, teremos que criar uma interface de erro no nosso CSS e passá-lo como parâmetro no nosso componente Form e em seguida implementá-la no nosso formulário.

```
4 interface FormProps {
5   hasError: boolean;
6 }
7
8 export const Title = styled.h1`
9   font-size: 48px;
10  color: #3A3A3A;
11  max-width: 450px;
12  line-height: 56 px;
13
14  margin-top: 80px;
15 `;
16
17 export const Form = styled.form<FormProps>`
18   margin-top: 40px;
19   max-width: 700px;
20   display: flex;
21
22   input {
23     flex: 1;
24     height: 70px;
25     padding: 0 24 px;
26     border: 0;
27     border-radius: 5px 0 0 5px;
28     color: #3a3a3a;
29     border: 2px solid #FFF;
30     border-right: 0px;
31
32     ${({ props }) => props.hasError && css `
33       border-color: #c53030;
34     `}
35   }
36 `;
```

**Linha 4** – criamos nossa interface com uma variável booleana.

**Linha 17** – passamos nossa interface como parâmetro para o nosso componente Form. Assim a variável “hasError” passou a ser uma propriedade (característica) do nosso formulário.

**Linha 29 e 30** – colocamos uma borda branca, que será alterada para vermelho quando o erro ocorrer.

**Linha 32** – ele verifica se o formulário possui alguma propriedade e se possuir verifica se ela é true. Se sim aplica a borda vermelha ao campo de pesquisa do nosso formulário. Mas para isso é necessário implementá-la no formulário, como veremos a seguir.

### 10.4. No nosso formulário inclua a propriedade “hasError”, da seguinte forma:

Estamos transformando a variável “inputError” no nosso arquivo “index.tsx” da pasta “dashboard”, para

```
<Form hasError={!!inputError} onSubmit={handleA
<input
```

o tipo booleano. Se ela conter alguma mensagem de erro, então ela passará a ter o valor “true”. Dessa forma o campo input ficará com a borda vermelha.

## 11. Salvando o resultado da busca em um storage local.

O storage local seria como um banco de dados textual ou simplesmente um cookie do browser, de forma que possa manter a lista de resultados das nossas pesquisas de repositórios do github.

### 11.1. Inclua o **useEffect** no import da linha 1

```
import React, {useState, useEffect, FormEvent } from 'react';
```

### 11.2. Altere a declaração da variável **repositories**, para o seguinte:

```
20 const [repositories, setRepositories] = useState<Repository[]>(() => {
21   const storedRepositories = localStorage.getItem('@GithubExplorer:repositories');
22
23   if(storedRepositories){
24     return JSON.parse(storedRepositories);
25   }
26 });
27
28 useEffect(() => {
29   localStorage.setItem('@GithubExplorer:repositories',
30     JSON.stringify(repositories),);
31 }, [repositories]);
```

**Linha 20** – Trocamos o array que tínhamos no final da linha, por uma função. Esta função cria um repositório local com o nome de '@GithubExplorer:repositories' (ela é do tipo repositories).

**Linha 23** – Se o storage não estiver vazio ele faz a conversão inversa do JSON para array e armazena na variável repositories, da linha 20. É esta variável que exibe a lista de variáveis na tela.

**Linha 28** – o useEffect é um recurso que fica observando a variável 'repositories'. Se ela sofrer alguma alteração no seu conteúdo, ele então atualiza o repositório local.

## 12. Navegando de uma página para outra

12.1. Começamos importando o componente Link da biblioteca react-router-dom

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

12.2. Trocamos agora, o nosso <a href> que tínhamos na linha 71, pelo seguinte:

```
70     {repositories.map(repository => (  
71       <Link key={repository.full_name} to={`/${repository.full_name}`} >  
72         <img src={repository.owner.avatar_url} alt={repository.owner.login}/>
```

Somente trocar o "a" pela palavra "Link" e o "href" por "to" com este conteúdo aí, que é o parâmetro que vai ser lido/capturado pela próxima página, que vai mostrar detalhes do repositório clicado. Lembrando que também é necessário trocar o "</a>" por "</Link>" na linha 78.

12.3. Agora temos que informar à nossa rota de repositório, que ela agora recebe um parâmetro:

```
<Route path="/" exact component={Dashboard} />  
<Route path="/repositories/:repository+" exact component={Repository} />  
</Switch>
```

O sinal de "+" indica que tudo o que vier após a palavra "repository" faz parte dela, como parâmetro.

12.4. Alteraremos agora nosso arquivo "index.tsx" da pasta "page/repository".

```
src > pages > repository > index.tsx > ...  
1  import React from 'react';  
2  import { useRouteMatch } from 'react-router-dom';  
3  
4  interface RepositoryParams {  
5    repository: string;  
6  }  
7  
8  const Repository: React.FC = () => {  
9    const { params } = useRouteMatch<RepositoryParams>();  
10  
11  
12    return <h1>Repository: {params.repository}</h1>  
13  };  
14  
15  export default Repository;  
16
```

Criamos a interface e a colocamos como parâmetro do useRouteMatch (linha 9).

Agora na linha 12, já conseguimos exibir o título do repositório que clicamos, na página de detalhes do repositório. O próximo passo é exibir o restante das informações sobre o repositório.

## 13. Criando a página de detalhes do repositório

Primeiro montaremos toda a parte visual, com dados estáticos e depois que estiver tudo ok, substituiremos pelos dados dinâmicos.



13.1. Na pasta “page/repositor”, crie o arquivo styles.ts, com o seguinte conteúdo:

```
src > pages > repository > styles.ts > Issues
1  import styled from 'styled-components';
2
3  export const Header = styled.header`
4    display: flex;
5    align-items: center;
6    justify-content: space-between;
7
8    a {
9      display: flex;
10     align-items: center;
11     text-decoration: none;
12     color: #a8a8b3;
13     transition: color 0.2s;
14
15     &:hover {
16       color: #666;
17     }
18   }
19
20   svg {
21     margin-right: 4px;
22   }
23 `;
```

```
25  export const RepositoryInfo = styled.section`
26    margin-top: 80px;
27
28    header {
29      display: flex;
30      align-items: center;
31
32      img{
33        width: 120px;
34        height: 120px;
35        border-radius: 50%;
36      }
37
38      div {
39        margin-left: 24px;
40
41        strong {
42          font-size: 36px;
43          color: #3d3d4d;
44        }
45
46        p {
47          font-size: 18px;
48          color: #737380;
49          margin-top: 4px;
50        }
51      }
52    }
53 `;
```

```
54  ul {
55    display: flex;
56    list-style: none;
57    margin-top: 40px;
58
59    li {
60      & + li {
61        margin-left: 80px;
62      }
63      strong{
64        display: block;
65        font-size: 36px;
66        color: #3d3d4d;
67      }
68      span {
69        display: block;
70        margin-top: 4px;
71        color: #6c6c80;
72      }
73    }
74  }
75
76 `;
```

```
78  export const Issues = styled.div`
79    margin-top: 80px;
80
81    a {
82      background: #fff;
83      border-radius: 5px;
84      width: 100%;
85      padding: 24px;
86      display: flex;
87      text-decoration: none;
88
89      align-items: center;
90      transition: transform 0.2s;
91
92      &:hover{
93        transform: translateX(10px);
94      }
95
96      & + a: &:hover {
97        margin-top: 16px;
98      }
99
100     div {
101       margin: 0 16px;
102       flex: 1;
103
104       strong {
105         font-size: 20px;
106         color: #3d3d4d;
107       }
108
109       p {
110         font-size: 18px;
111         color: #a8a8b3;
112         margin-top: 4px;
113       }
114     }
115
116     svg {
117       margin-left: auto;
118       color: #c6c6d6;
119     }
120   }
121 `;
```

O conteúdo do estilo Issues foi praticamente copiado do CSS de Repositories, retirando somente a parte de imagem. O professor recomenda que quando a redundância de código acontece somente uma vez, não há necessidade de criar um componente à parte para ser reaproveitado.



13.2. O arquivo “index.tsx” da pasta “page/repository”, ficará da seguinte forma, por enquanto somente com dados estáticos:

```
src > pages > repository > index.tsx > Repository
1  import React from 'react';
2  import { useRouteMatch, Link } from 'react-router-dom';
3  import { FiChevrnsLeft, FiChevronRight } from 'react-icons/fi';
4  import logoImg from '../../assets/logo.svg';
5
6  import { Header, RepositoryInfo, Issues } from './styles';
7
8
9  interface RepositoryParams {
10    repository: string;
11  }
12
13  const Repository: React.FC = () => {
14    const { params } = useRouteMatch<RepositoryParams>();
15
16
17    return (
18      <>
19        <Header>
20          <img src={logoImg} alt="Github Explorer"/>
21          <Link to="/">
22            <FiChevrnsLeft size={16}/>
23            Voltar
24          </Link>
25        </Header>
26
27        <RepositoryInfo>
28          <header>
29            <img src="" alt=""/>
30            <div>
31              <strong>Rocketseat/unform</strong>
32              <p>Descrição</p>
33            </div>
34          </header>
35          <ul>
36            <li>
37              <strong>1808</strong>
38              <span>Stars</span>
39            </li>
40            <li>
41              <strong>48</strong>
42              <span>Forks</span>
43            </li>
44            <li>
45              <strong>67</strong>
46              <span>Issues abertas</span>
47            </li>
48          </ul>
49        </RepositoryInfo>
50
51        <Issues>
52          <Link to="asasasas">
53            <div>
54              <strong>sasasasas</strong>
55              <p>sasasasas</p>
56            </div>
57            <FiChevronRight size={20}/>
58          </Link>
59        </Issues>
60      </>
61    );
62  };
63
64  export default Repository;
```

### 13.3. Integração desta página com a API do Github.

- Importar a api

```
import api from '../services/api';
```

- Criar as interfaces do conteúdo que virá do repositório do github.

```
13 interface Repository {
14     full_name: string;
15     description: string;
16     stargazers_count: number;
17     forks_count: number;
18     open_issues_count: number;
19     owner: {
20         login: string;
21         avatar_url: string;
22     };
23 }
24
25 interface Issue {
26     id: number;
27     title: string;
28     html_url: string;
29     user: {
30         login: string;
31     }
32 }
```

- Implementar o “useState” nas nossas variáveis.

```
34 const Repository: React.FC = () => {
35     const [repository, setRepository] = useState<Repository | null>(null);
36     const [issues, setIssues] = useState<Issue[]>([]);
37     const { params } = useRouteMatch<RepositoryParams>();
38
39     useEffect(() => {
40         api.get(`repos/${params.repository}`).then(response => {
41             setRepository(response.data);
42         });
43         api.get(`repos/${params.repository}/issues`).then(response => {
44             setIssues(response.data);
45         });
46     }, [params.repository]);
47
48
49     return (
```

Observe a diferença no uso do useState em “repository” e em “issues”. No caso a variável “issues” é um array. O **useRouteMatch** é para especificarmos qual o tipo de dado que será enviado/recebido via url/parâmetro.

O useEffect busca os dados de repositório e issues da api do Github. Essa é uma maneira de implementar a busca de dados em servidores externos, sem a necessidade de utilizar funções assíncronas. Este recurso não aceita o uso do “async”. O “**then**” tem praticamente a mesma função do “async”. O uso do useEffect se aplica quando precisamos trazer conteúdos diferentes ao mesmo tempo, sem que um dependa do outro. No caso, o conteúdo de repository não depende do conteúdo de issues. Ambos estão sendo trazidos a partir da consulta por “params.repository”, que é o parâmetro enviado via URL, pela página principal.

- Abaixo do return (linha 49), temos o <header> que não sofreu alteração. E mais abaixo, temos a implementação da exibição dos dados do repositório e a lista de issues.

```
59     { repository && (  
60         <RepositoryInfo>  
61             <header>  
62                 <img src={repository.owner.avatar_url} alt={repository.owner.login}/>  
63                 <div>  
64                     <strong>{repository.full_name}</strong>  
65                     <p>{repository.description}</p>  
66                 </div>  
67             </header>  
68             <ul>  
69                 <li>  
70                     <strong>{repository.stargazers_count}</strong>  
71                     <span>Stars</span>  
72                 </li>  
73                 <li>  
74                     <strong>{repository.forks_count}</strong>  
75                     <span>Forks</span>  
76                 </li>  
77                 <li>  
78                     <strong>{repository.open_issues_count}</strong>  
79                     <span>Issues abertas</span>  
80                 </li>  
81             </ul>  
82         </RepositoryInfo>  
83     )}  
84  
85     <Issues>  
86         {issues.map(issue => (  
87             <Link key={issue.id} to={issue.html_url}>  
88                 <div>  
89                     <strong>{issue.title}</strong>  
90                     <p>{issue.user.login}</p>  
91                 </div>  
92                 <FiChevronRight size={20}/>  
93             </Link>  
94         )})  
95     </Issues>  
96 </>  
97 );  
98 };
```

**Linha 59** – verifica se a variável “repository” não está vazia.

**Linha 60** – RepositoryInfo é o nosso componente css.

**Linha 86** – O recurso “map” funciona como um foreach.

Fonte: Curso de React da Rocketseat

Maio/2020