

## Objetivos:

- i. Criar um projeto React TypeScript;
- ii. Routes;
- iii. Rotas com restrição de acesso.

### i. Criar um projeto React TypeScript

Siga os passos para criar uma aplicação React TS:

- a) Acesse pelo prompt do CMD o local que você deseja criar o projeto React e digite o comando a seguir para criar o projeto React:

```
npx create-react-app front --template typescript
```

- b) O projeto foi criado na pasta **front**. No CMD acesse a pasta **front** e abra ela no VS Code;
- c) Ao lado tem-se a estrutura de pastas e arquivos da aplicação criada pelo Create React App. Para simplificar o projeto:
  - Delete os arquivos sinalizados pela seta vermelha;
  - Substitua os códigos dos arquivos index.html (Figura 1), index.tsx (Figura 2) e App.tsx (Figura 3).

- d) Crie a pasta **.vscode** e o arquivo **settings.json** (assim como é mostrado pelas setas azuis) e coloque o JSON da Figura 4. Essas propriedades são usadas pelo VS Code quando estamos digitando os códigos, são chamadas de funcionalidades prontas do VS Code, por exemplo:

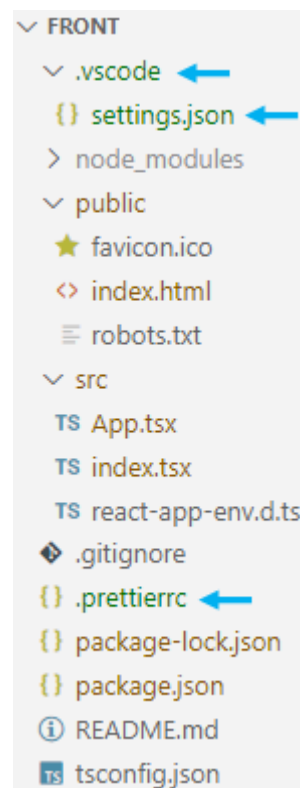
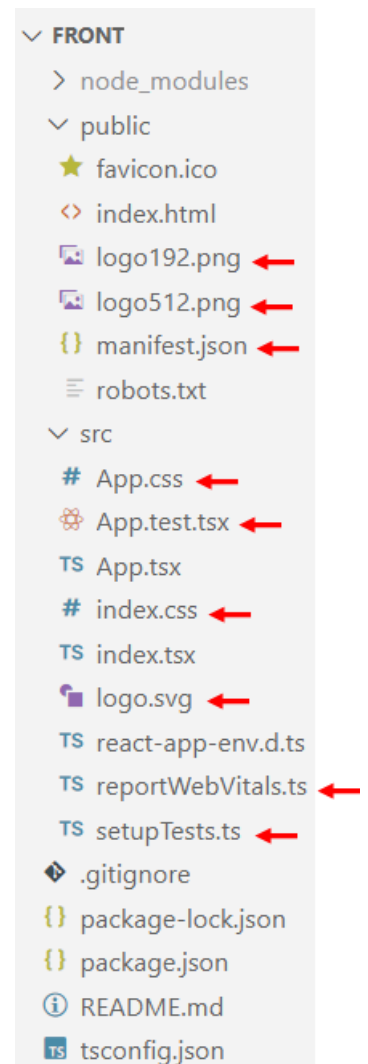
**defaultFormatter** é o prettier a ser utilizado na formatação do código. Você precisará instalar a extensão/plugin **esbenp.prettier-vscode** ou algum outro prettier no seu VS Code;

**editor.bracketPairColorization.enabled** colore os **()**, **[]** e **{}** do código.

- e) Crie o arquivo **.prettierrc** (Figura 5) para sobrescrever as configurações padrões do Prettier

(<https://prettier.io/docs/en/configuration.html>);

- f)



- g) Adicione a dependência `npm i react-router-dom` (<https://www.npmjs.com/package/react-router-dom>);
- h) Para subir o projeto digite `npm run start` ou `npm start` no terminal do VS Code. A aplicação estará na porta padrão 3000;

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Figura 1 – Código do arquivo public/index.html.

```
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render( <App /> );
```

Figura 2 – Código do arquivo src/index.tsx.

```
function App() {
  return (
    <div>
      bom dia
    </div>
  );
}

export default App;
```

Figura 3 – Código do arquivo src/App.tsx.

```
{
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.bracketPairColorization.enabled": true,
  "editor.formatOnSave": true,
  "editor.formatOnPaste": true,
  "editor.wordWrap": "on"
}
```

Figura 4 – Código do arquivo `.vscode/settings.json`.

```
{
  "singleQuote": false,
  "tabWidth": 2,
  "useTabs": false,
  "semi": true
}
```

Figura 5 – Código do arquivo `.prettierrc`.




## ii. Routes

Em sites tradicionais, o navegador solicita um documento de um servidor web, baixa e avalia recursos CSS e JavaScript e renderiza o HTML enviado do servidor. Quando o usuário clica em um link, ele inicia o processo novamente para uma nova página.

O roteamento do lado do cliente permite que o aplicativo atualize a URL de um clique no link sem fazer outra solicitação para outro documento do servidor. Em vez disso, o aplicativo pode renderizar imediatamente uma nova interface do usuário usando recursos disponíveis no cliente. Aqui, estes recursos serão os **componentes**, ou seja, os componentes podem ser endereçados por rotas URLs. Como exemplo, `http://localhost:3000/about` pode endereçar um componente de nome About.

O React Router é uma biblioteca de roteamento leve e com todos os recursos para a biblioteca React JavaScript. O React Router é executado em todos os lugares em que o React é executado; na web, no servidor (usando node.js) e no React Native.

O roteamento do lado do cliente é criado usando os componentes `BrowserRouter`, `Routes` e `Route` dos pacotes `react-router` e `react-router-dom`. No exemplo da Figura 6 foram definidas três rotas para os componentes `Home`, `About` e `Generico`. A seguir tem-se o teste das rotas no navegador:

 localhost:3000	 localhost:3000/about	 localhost:3000/teste
Bem-vindo	Exemplo de rotas	Rota desconhecida

```
import { Route, Routes } from "react-router";
import { BrowserRouter } from "react-router-dom";

function App() {
  return <AppRoutes />;
}

function AppRoutes() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="*" element={<Generico />} />
      </Routes>
    </BrowserRouter>
  );
}

function About() {
  return <div>Exemplo de rotas</div>;
}

function Home() {
  return <div>Bem-vindo</div>;
}

function Generico() {
  return <div>Rota desconhecida</div>;
}

export default App;
```

Figura 6 – Exemplo de rotas - código do arquivo src/App.tsx.

### iii. Rotas com restrição de acesso

No exemplo da Figura 7 foram definidas as rotas para os componentes **A**, **B**, **C** e **D**, e em caso de rota desconhecida será endereçado o componente **Erro**.

As rotas foram definidas nas funções **UmRoutes** e **DoisRoutes**. A diferença entre elas está apenas nos componentes roteados, ou seja, elas não fazem restrição de acesso.

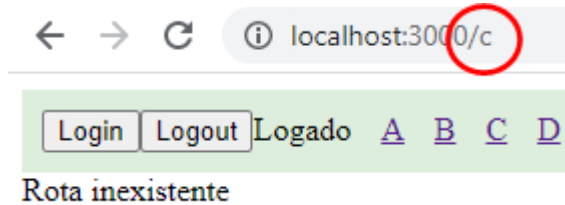
A restrição de acesso está na função **Rotas** que retorna apenas uma das rotas de acordo com o conteúdo da propriedade **logado**:

```
function Rotas() {
  const { logado } = useContext(Contexto);
  return logado ? <UmRoutes /> : <DoisRoutes />;
}
```

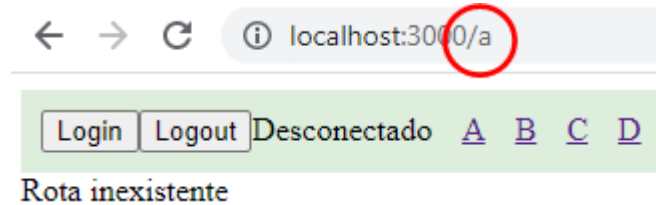
```
}
```

Essa organização do código faz com que a definição das rotas seja trocada mudando a propriedade de estado `logado`.

A rota `/c` não existe quando está logado, pois foi utilizada a função `UmRoutes` para definir as rotas:



A rota `/a` não existe quando não está logado, pois foi utilizada a função `DoisRoutes` para definir as rotas:



O componente `Link` do pacote `react-router-dom` é usado para criar links.

```
import { createContext, useContext, useState } from "react";
import { Route, Routes } from "react-router";
import { BrowserRouter, Link } from "react-router-dom";

function App() {
  return (
    <ContextoProvider>
      <Rotas />
    </ContextoProvider>
  );
}

function Rotas() {
  const { logado } = useContext(Contexto);
  return logado ? <UmRoutes /> : <DoisRoutes />;
}

function UmRoutes() {
  return (
    <BrowserRouter>
      <Menu />
      <Routes>
        <Route path="*" element={<Erro />} />
        <Route path="/a" element={<A />} />
        <Route path="/b" element={<B />} />
      </Routes>
    </BrowserRouter>
  );
}

function DoisRoutes() {
  return (
```

```

    <BrowserRouter>
      <Menu />
      <Routes>
        <Route path="/" element={<Erro />} />
        <Route path="/c" element={<C />} />
        <Route path="/d" element={<D />} />
      </Routes>
    </BrowserRouter>
  );
}

function Menu() {
  const { logado, setLogado } = useLogado();
  return (
    <div style={{ backgroundColor: "#ded", padding: 10 }}>
      <button onClick={() => setLogado(true)}>Login</button>
      <button onClick={() => setLogado(false)}>Logout</button>
      <span>{logado ? "Logado" : "Desconectado"}</span>
      <Link to="a" style={{ marginLeft: 15 }}>
        A
      </Link>
      <Link to="b" style={{ marginLeft: 15 }}>
        B
      </Link>
      <Link to="c" style={{ marginLeft: 15 }}>
        C
      </Link>
      <Link to="d" style={{ marginLeft: 15 }}>
        D
      </Link>
    </div>
  );
}

function A() {
  return <div>Componente A</div>;
}

function B() {
  return <div>Componente B</div>;
}

function C() {
  return <div>Componente C</div>;
}

function D() {
  return <div>Componente D</div>;
}

```

```
function Erro() {
  return <div>Rota inexistente</div>;
}

// definição do contexto
interface Props {
  logado: boolean;
  setLogado: Function;
}

const Contexto = createContext<Props>({} as Props);

function ContextoProvider({ children }: any) {
  const [logado, setLogado] = useState(false);
  return (
    <Contexto.Provider value={{ logado, setLogado }}>
      {children}
    </Contexto.Provider>
  );
}

// definição do Hook
function useLogado() {
  const context = useContext(Contexto);
  return context;
}

export default App;
```

Figura 7 – Exemplo de rotas com restrição de acesso- código do arquivo src/App.tsx.