



@tiacademybrasil

# Front-end com React

Profº. Erinaldo

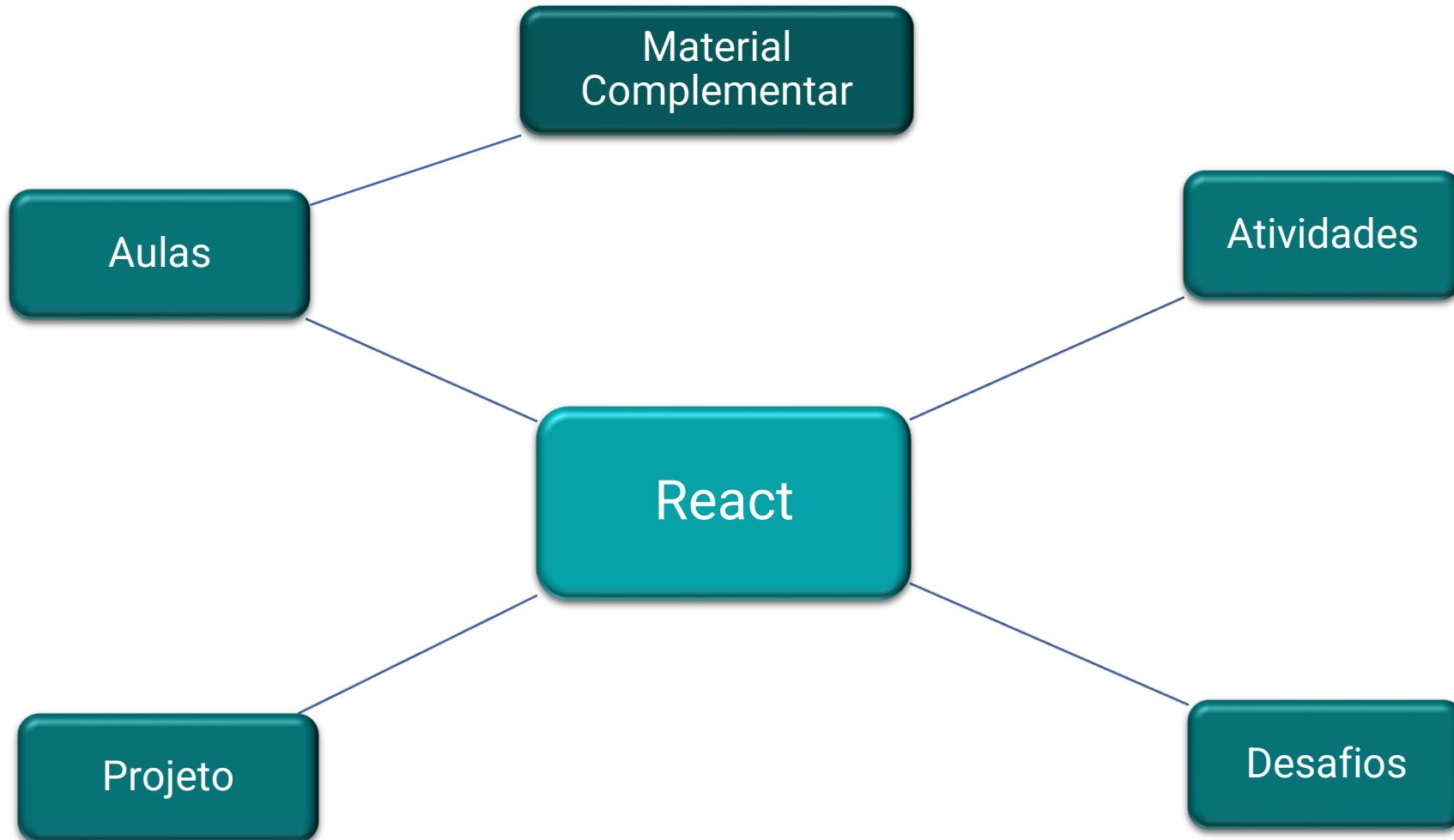
# Apresentação



## Prof. Me. **Erinaldo Sanches Nascimento**

- Formado em Ciência da Computação (Universidade Santa Cecília - Santos/SP), Especialista em Administração e Desenvolvimento de Banco de Dados (UTFPR - Medianeira/PR), Mestre em Bioinformática (UTFPR - Cornélio Procópio/PR).
- Professor na educação superior nas modalidades presencial e EAD pela Unicesumar (Maringá/PR). Coordena o Ensino Médio Profissional em Informática pela SEEDPR em Sarandi/PR.
- Analista e desenvolvedor web na TICLab.

# Estrutura do Curso



# Estrutura do Curso

Dia 1

Como utilizar o  
React

Síncrono

Dia 2

Implementar as  
Consultas

Assíncrono /  
Síncrono

Dia 3

Implementar os  
Cadastro

Síncrono /  
Assíncrono

Dia 4

Implementar as  
Alterações

Assíncrono /  
Síncrono

Dia 5

Implementar as  
Exclusões

Assíncrono /  
Síncrono

Material Complementar

Atividade Guiada

Atividade Prática

Encontros ao vivo

18 horas

4 horas

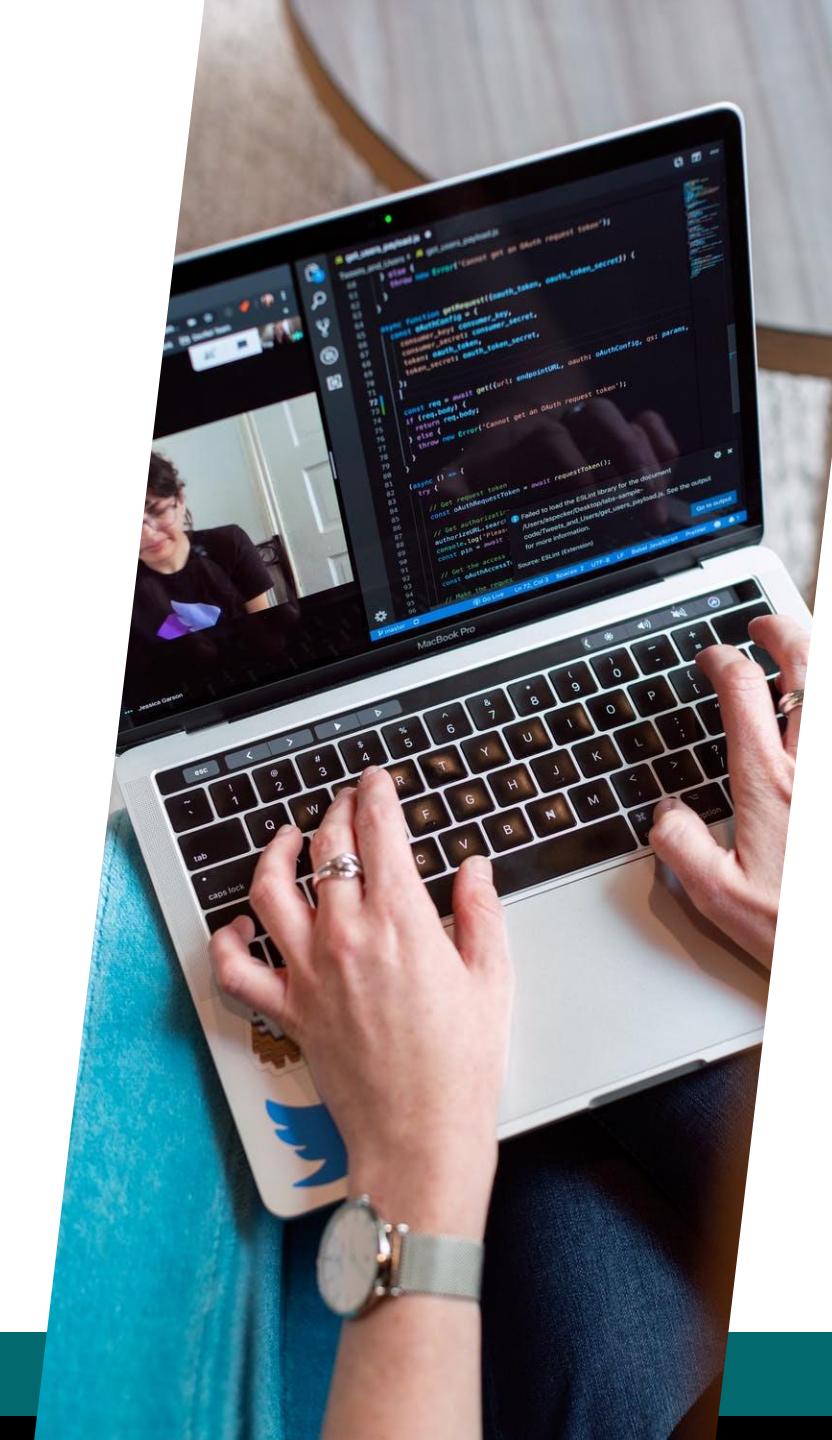
Supporte Assíncrono

## Antes de começar...

Você já aprendeu a criar páginas estáticas e estilizadas com HTML e CSS. Também utilizou a linguagem de programação Javascript para desenvolver as estruturas lógicas fundamentais.

Na sequência você desenvolveu um back-end em linguagem de programação Nodejs, implementando as principais rotas para uma aplicação.

Agora chegou o momento de implementar o front-end da aplicação utilizando a biblioteca React.



## Apresentação do problema

Vamos relembrar o sistema a ser implementado.

Trata-se de uma empresa do setor de TIC (Tecnologia da Informação e Comunicação) que deseja controlar dos seus clientes, serviços e solicitações de forma on-line.

O controle implica em inserir, remover, alterar e consultar todos os envolvidos na atividade da empresa.

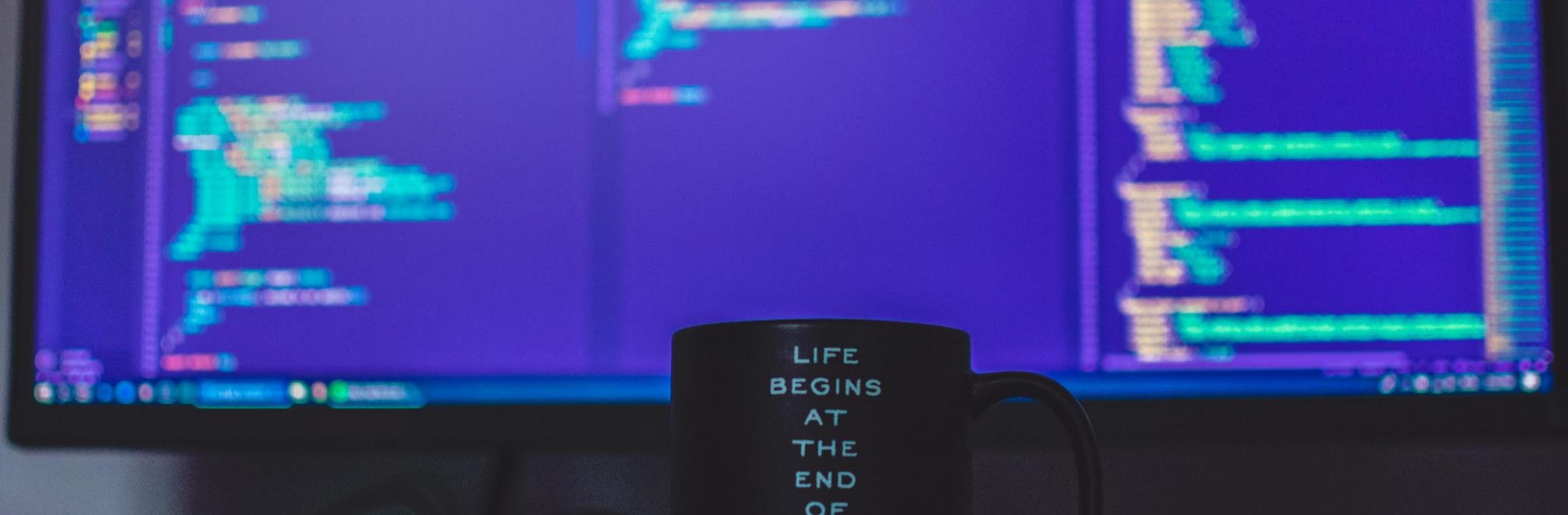


# Mockup do projeto

A equipe de designer desenvolveu uma representação da solução do projeto para a empresa.

The image displays two side-by-side desktop screenshots of a web application. The left screenshot, titled 'Desktop - 1', shows a landing page for 'Services TI Academy'. It features a banner with the text 'Tudo o que você precisa de solução em TIC ao alcance de um clique!' and a 'Residential proxy' section with icons for User, Proxy Server, Proxy IP, and Target. Below this are three buttons: 'Clientes' (Clients) with a hand icon, 'Serviços' (Services) with a 24/7 icon, and 'Pedidos' (Orders) with a document icon. A small image of network cables is at the bottom left, and the text 'Redução de gastos e aumento da produtividade' (Cost reduction and productivity increase) is at the bottom right. The right screenshot, titled 'Desktop - 2', shows a user profile for 'Erinaldo Sanches Nascimento'. The profile includes a photo, CPF (123.456.789-00), RG (12.345.678-90), Telefone ((44) 3222-2222), Celular ((44) 99998-9898), E-mail (erinaldosnascimento@gmail.com), CEP (87.000-000), Logradouro (Av. Cosme Lomes, 1700), Bairro (Nova Aurora), Cidade (Maringá), UF (PR), Nascimento (26/06/1976), and Cliente desde (10/08/2016). Navigation icons for Home and Pedidos are visible in the top right of the second screenshot.

<https://www.figma.com/file/NCol7WCXA63x4R7v8LmNIH/TIAcademyServices?node-id=0%3A1>



@tiacademybrasil

# Introdução ao React

AULA 1 | Vamos praticar?

# Dia 1: Introdução ao React

1. Iniciando um novo projeto
2. Criar as páginas da aplicação

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



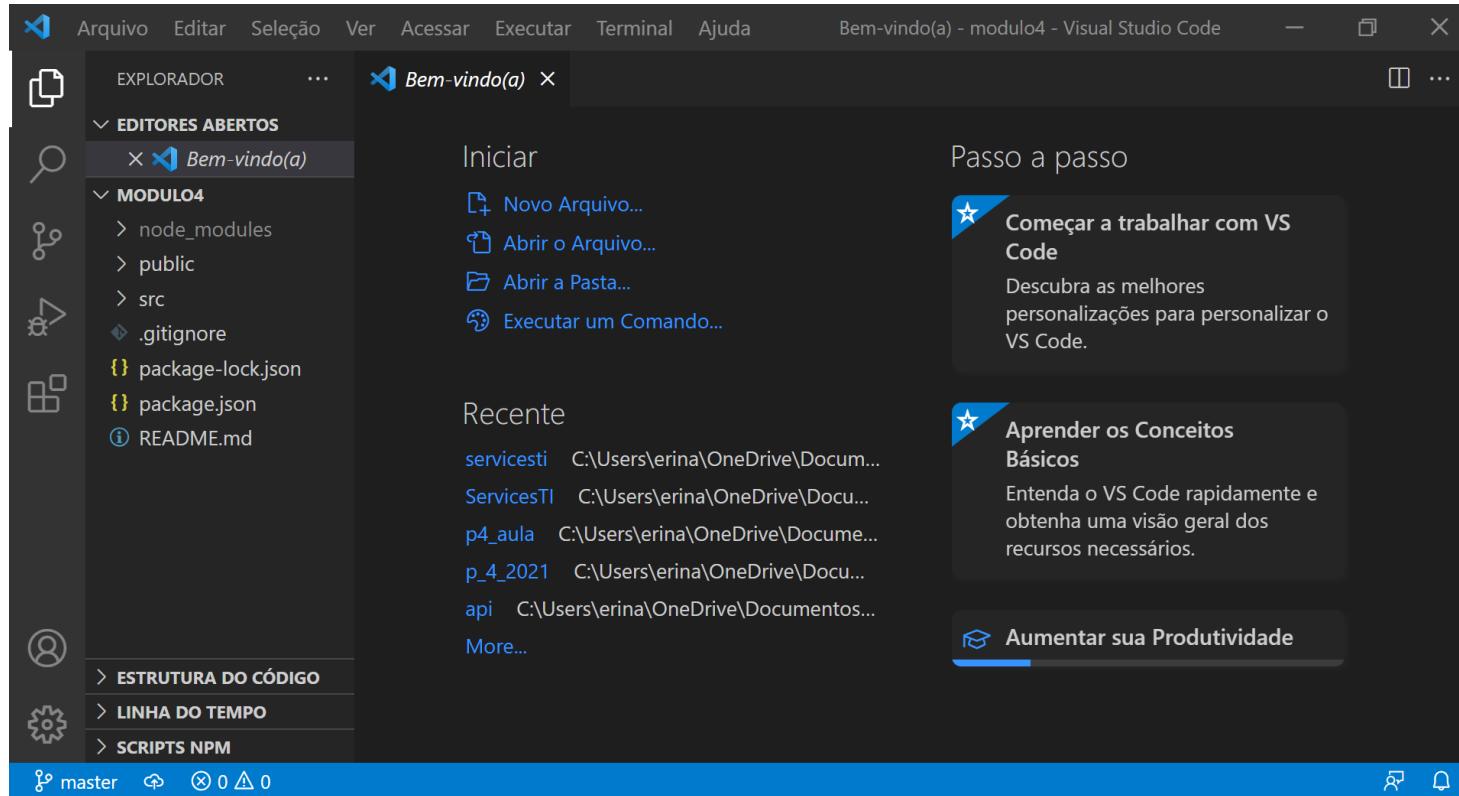
## Iniciando um novo projeto

1. Acesse o site do React, que é uma biblioteca JavaScript para construção de interfaces de usuário, <https://pt-br.reactjs.org/>.
2. Na documentação você vai encontrar um material para construir um novo projeto, <https://pt-br.reactjs.org/docs/create-a-new-react-app.html>.
3. No terminal, na pasta que está no mesmo nível do `ciclo3` você vai executar o comando `npx create-react-app ciclo4`.
4. Instalado, acesse o diretório `ciclo4` com o comando `cd ciclo4`.



# Iniciando um novo projeto

## 5. Abra o seu editor de código.



## 6. Abra um novo terminal no VSCode.



# Iniciando um novo projeto

7. Inicie o projeto com `npm start`.



8. Mude o título da aplicação para Services TI Academy no arquivo `index.html` que está na pasta `public`.

Onde vamos escrever o código dos nossos componentes?



# Iniciando um novo projeto

9. Abra a pasta `src` e acesso o arquivo `App.js`.
10. Na tag do parágrafo altere o texto.

```
src > JS App.js > App  
1 import logo from './logo.svg';  
2 import './App.css';  
3  
4 function App() {  
5   return (  
6     <div className="App">  
7       <header className="App-header">  
8         <img src={logo} className="App-logo" alt="logo" />  
9         <p>  
10           Olá, mundo!  
11         </p>  
12         <a  
13           className="App-link"  
14           href="https://reactjs.org"  
15           target="_blank"  
16           rel="noopener noreferrer"  
17         >  
18           Learn React  
19         </a>  
20       </header>  
21     </div>  
22   );
```

Recarregue  
a página



# Iniciando um novo projeto

11. Abra na pasta `src` e acesse o arquivo `index.js` e exclua as linhas selecionadas.

```
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css'; // Line 3
4  import App from './App';
5  import reportWebVitals from './reportWebVitals';
6
7  ReactDOM.render(
8    <React.StrictMode>
9      <App />
10     </React.StrictMode>,
11     document.getElementById('root')
12   );
13
14  // If you want to start measuring performance in your app, pass a function
15  // to log results (for example: reportWebVitals(console.log))
16  // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
17  reportWebVitals(); // Lines 14-17
18
```

12. Exclua, também, os arquivos `App.test.js`, `logo.svg`, `reportWebVitals.js` e `index.css`.



## Iniciando um novo projeto

13. Mantenha o arquivo App.css, mas apague todo o seu conteúdo.
14. No arquivo App.js deixe apenas uma div.

```
src > JS App.js > App
1   import './App.css';
2
3   function App() {
4     return (
5       <div>
6         |
7       </div>
8     );
9   }
10
11  export default App;
```



# Iniciando um novo projeto

15. Crie um h1 com o texto Seja bem-vindo a Services TIAcademy.

```
src > JS App.js > App
1   import './App.css';
2
3   function App() {
4     return (
5       <div>
6         → <h1>Seja bem-vindo a Services TIAcademy.</h1>
7       </div>
8     );
9   }
10
11  export default App;
12
```



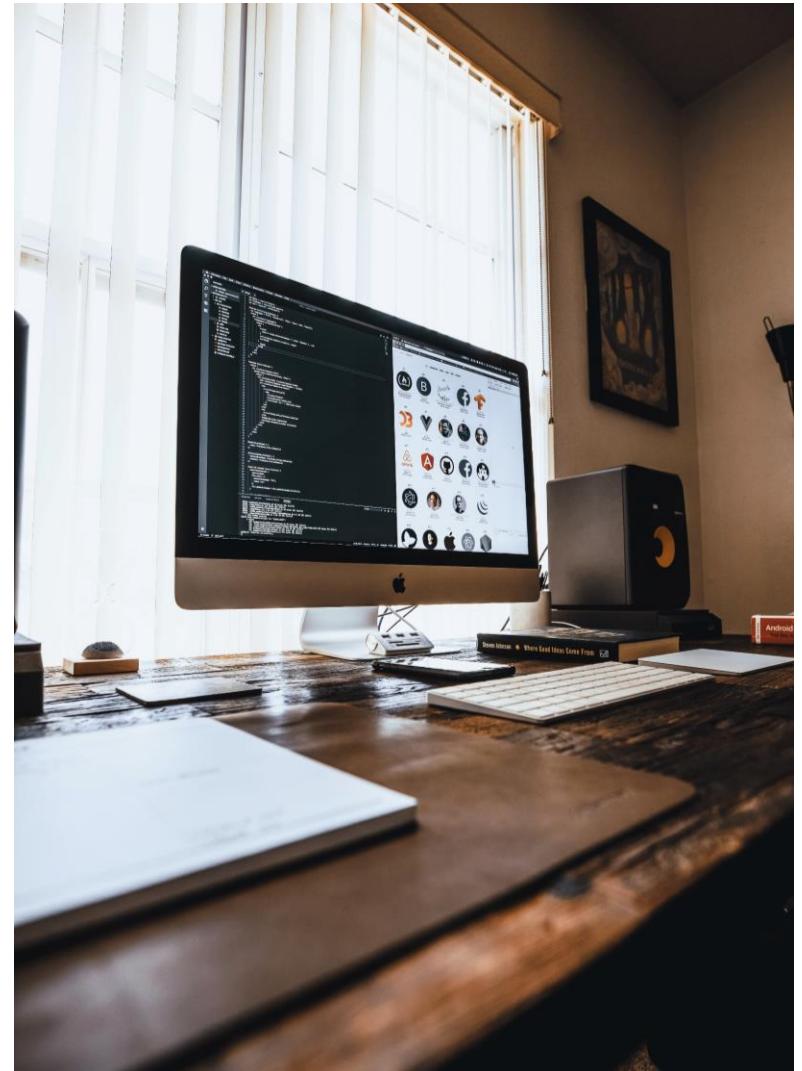
## Criar as páginas da aplicação

1. Na pasta `src` crie uma nova pasta chamada `pages`, a pasta `Home` para a primeira página e, nessa última página, o arquivo `index.js`.
2. Instale a dependência `react-router-dom` para gerenciar as rotas da aplicação. Faça a partir do endereço.  
<https://reactrouter.com/web/guides/quick-start>, com o comando `npm install react-router-dom`.
3. Importe a dependência no arquivo `App.js`.
4. Importe o arquivo `index.js` da página `Home`.
5. Passe a rota do componente `Home`.



# Criar as páginas da aplicação

```
src > JS App.js > ...
1  import {BrowserRouter as Router, Switch, Route} from 'react-router-dom';
2
3  import {Home} from './pages/Home/';
4
5  function App() {
6    return (
7      <div>
8        <Router>
9          <Switch>
10            <Route path="/" component={Home}/>
11          </Switch>
12        </Router>
13      </div>
14    );
15  }
16
17  export default App;
18 |
```



# Criar as páginas da aplicação

## 6. Criar o index.js da página Home.

```
src > pages > Home > JS index.js > ...
1  export const Home = () => {
2    return(
3      <div>Home</div>
4    );
5  };
```

Recarregue  
a página

7. Crie no diretório pages uma nova pasta chamada VisualizarCliente e dentro dela um arquivo index.js.
8. Copie o código index.js da página Home para o arquivo de mesmo nome o diretório VisualizarCliente, alterando apenas o texto.



# Criar as páginas da aplicação

## 9. Faça as alterações no arquivo App.js.

```
src > JS App.js > ...
1  import {BrowserRouter as Router, Switch, Route} from 'react-router-dom';
2
3  import {Home} from './pages/Home/';
4  import {VisualizarCliente} from './pages/VisualizarCliente/';
5
6  function App() {
7    return (
8      <div>
9        <Router>
10       <Switch>
11         <Route exact path="/" component={Home}/>
12         <Route path="/visualizar-cliente" component={VisualizarCliente}/>
13       </Switch>
14     </Router>
15   </div>
16 );
17 }
18
19 export default App;
```



## 10. Recarregue a página.

# Criar as páginas da aplicação

## Exercícios:

1. Organize dentro de pages 3 diretórios: Cliente, Servico e Pedido.
2. Mova o componente VisualizarCliente para o diretório Cliente.
3. Atualizar e arquivo App.js e teste novamente a aplicação.
4. Crie a rota de visualização para cada novo diretório e teste.
5. Pensando na aplicação, você pode criar rotas para cada uma das funcionalidades do CRUD em cada objeto do sistema.



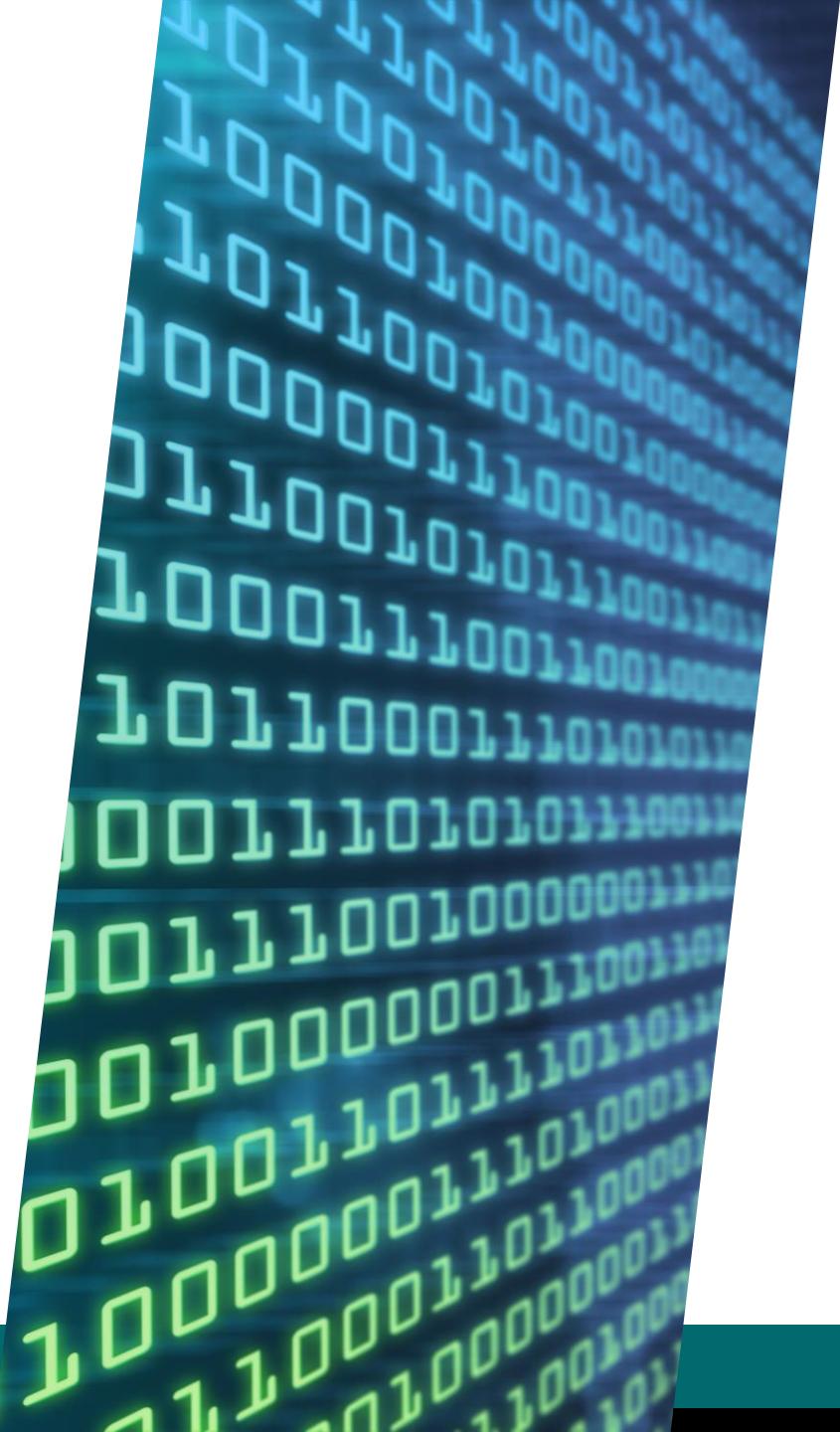
## O que aprendemos...

Você iniciou um novo projeto utilizando o React e criou as páginas da aplicação.



## O que vem em seguida...

Você vai Implementar layout da aplicação e incluir a logo do projeto.

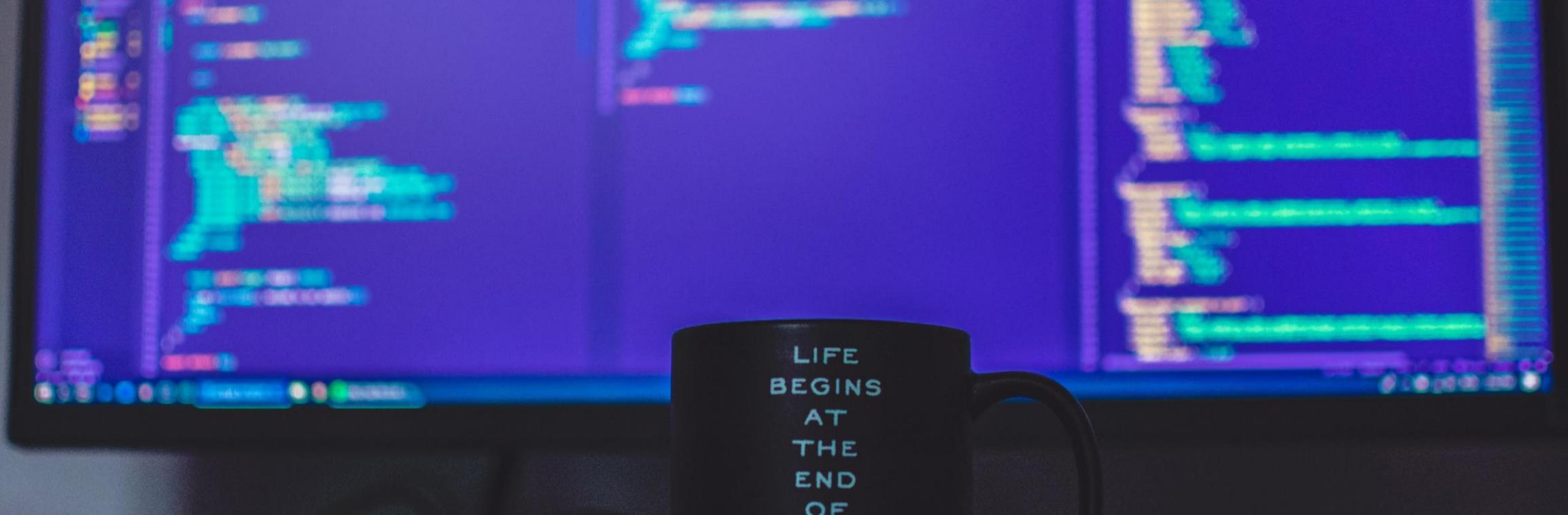




@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Introdução ao React

AULA 2 | Vamos praticar?

# Dia 1: Introdução ao React

1. Iniciando um novo projeto
2. Criar as páginas da aplicação
3. Implementar layout

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



# Dia 1: Introdução ao React

1. Implementar layout
2. Incluir a logo do projeto

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



## Implementar layout

1. Instalar a biblioteca reactstrap para utilizar o Bootstrap no React.
  - a) Acesse o endereço <https://reactstrap.github.io/>.
  - b) No site vá para a parte de adicionar Bootstrap (*Adding Bootstrap*) e execute os dois comandos para instalar o Bootstrap e o Reactstrap.

```
npm install --save bootstrap
```

```
npm install --save reactstrap
```
  - c) Importe o Bootstrap no arquivo index.js do diretório src.



# Implementar layout

```
src > JS index.js
● 1 import React from 'react';
  2 import ReactDOM from 'react-dom';
  3 import App from './App';
  4
  5 import 'bootstrap/dist/css/bootstrap.min.css'; ●
  6
  7 ReactDOM.render(
  8   <React.StrictMode>
  9     <App />
 10   </React.StrictMode>,
 11   document.getElementById('root')
 12 );
```



## Implementar layout

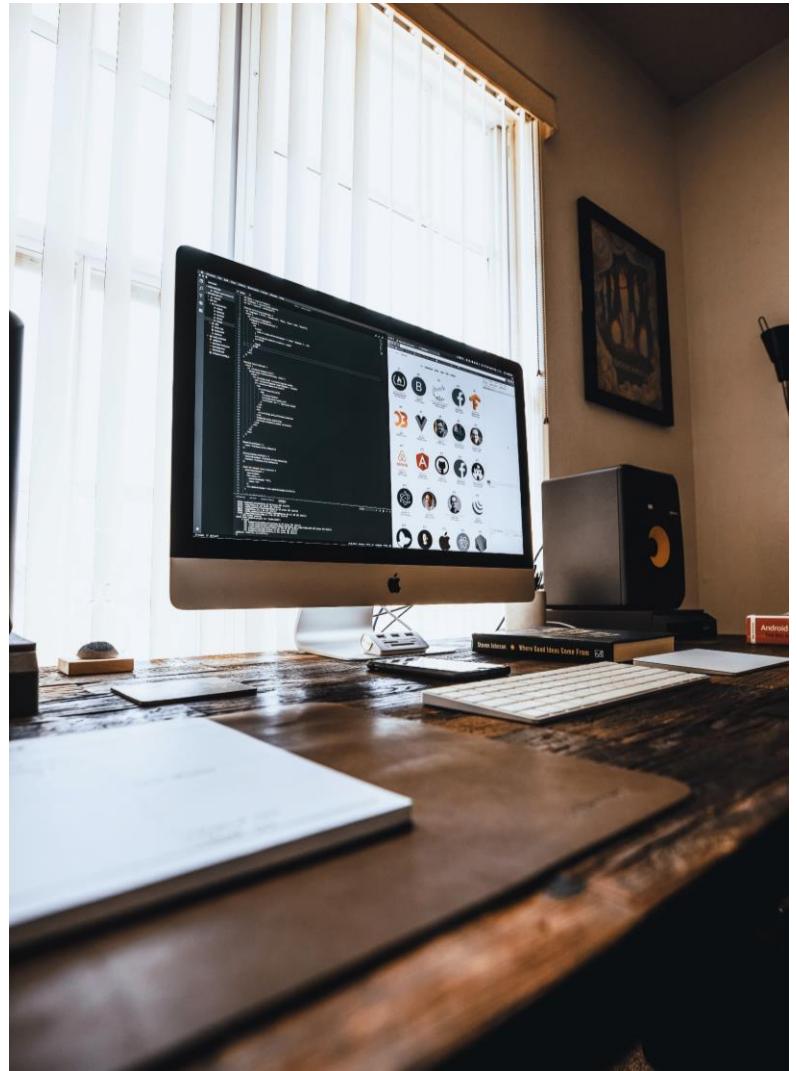
2. Execute novamente o servidor e identifique alguma alteração.
3. Acesse no site do Reactstrap a parte de componentes,  
<https://reactstrap.github.io/components/alerts/>.
4. Acesse o componente Navbar,  
<https://reactstrap.github.io/components/navbar/>.
5. No projeto, em `src`, crie um diretório chamado `components` e crie um novo componente chamado `Menu.js`.
6. Copie o código de um dos componentes criados anteriormente e altere conforme a seguir.



# Implementar layout

```
src > components > JS Menu.js > ...
1  export const Menu = () => {
2    return(
3      <div>
4        <h1>Menu</h1>
5      </div>
6    );
7  };
```

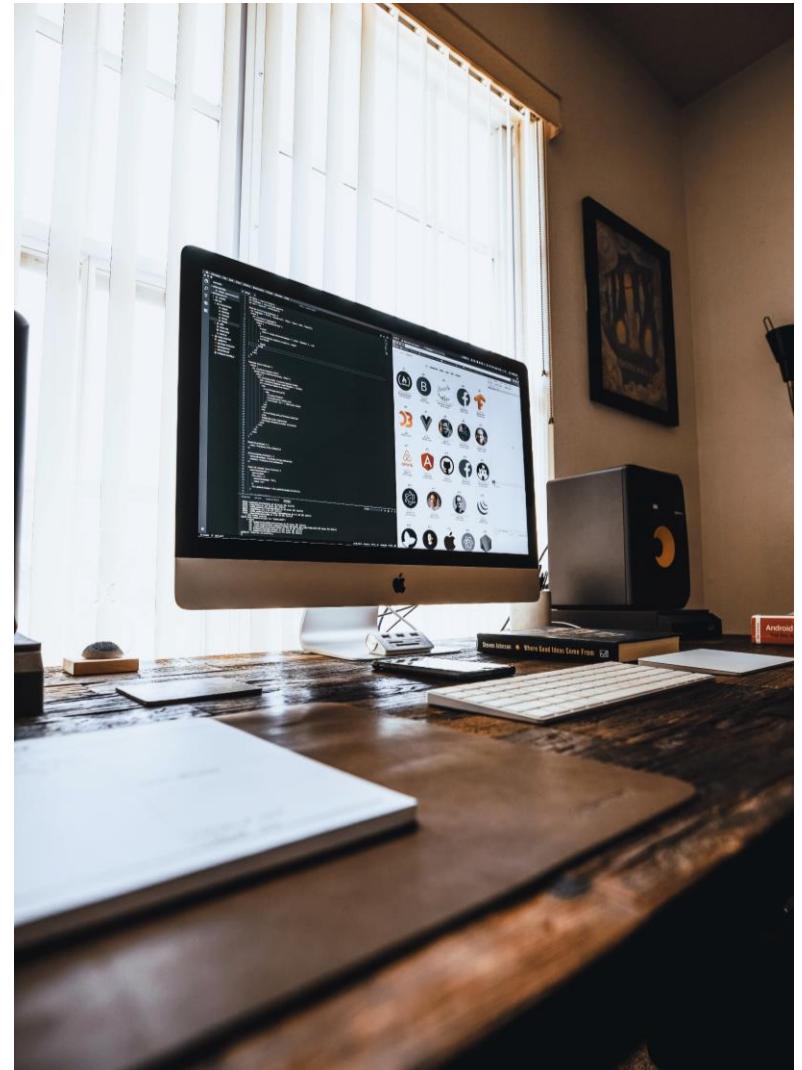
7. Abra o arquivo App.js e importe o componente Menu.
8. Utilize o componente, abrindo e fechando-o.



# Implementar layout

```
1 import {BrowserRouter as Router, Switch, Route} from 'react-router-dom';
2
3 import {Home} from './pages/Home/';
4 import {VisualizarCliente} from './pages/Cliente/VisualizarCliente';
5
6 import {Menu} from './components/Menu';
7
8 function App() {
9   return (
10   <div>
11     <Menu />
12     <Router>
13       <Switch>
14         <Route exact path="/" component={Home}/>
15         <Route path="/visualizar-cliente" component={VisualizarCliente}/>
16       </Switch>
17     </Router>
18   </div>
19 );
20 }
21
22 export default App;
```

8. Verifique no navegador, passando pelas rotas.



# Implementar layout

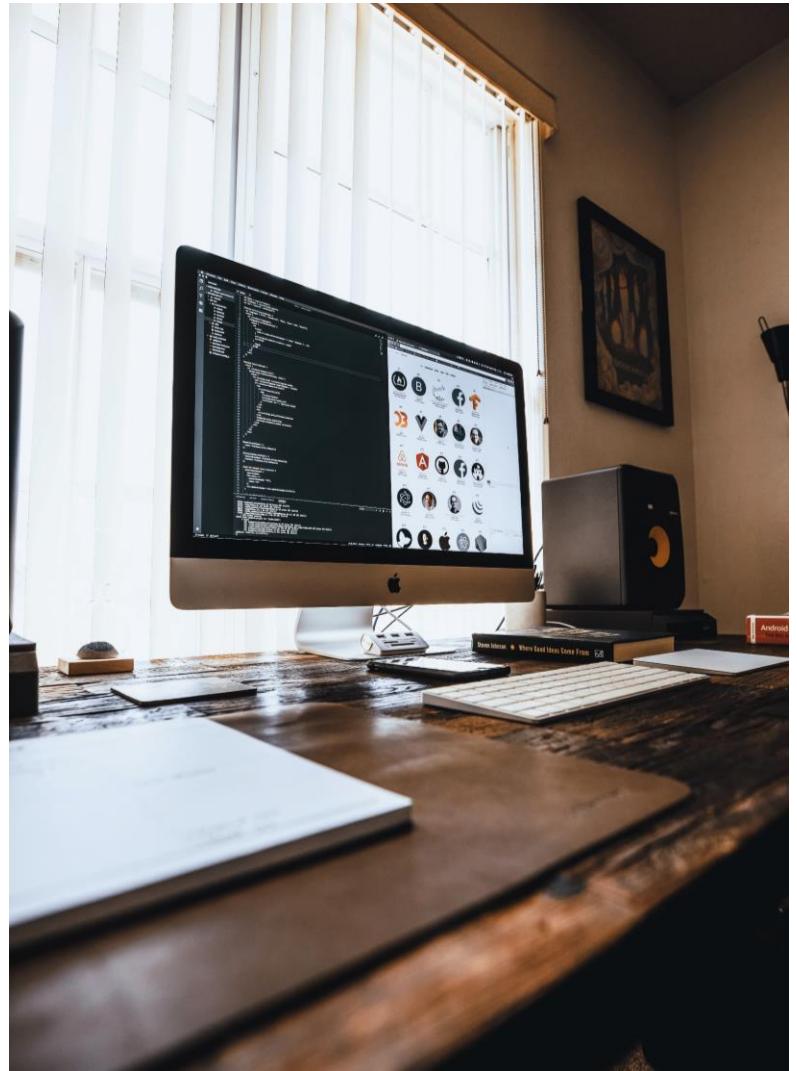
9. Altere o componente Menu conforme a seguir:
  - a) Copie as partes conforme indicado na documentação do reactstrap.
  - b) Copie as linhas de importações.
  - c) Na importação do reactstrap inclua Container.
  - d) Insira props para receber valor.
  - e) As duas linhas logo abaixo a declaração.

```
src > components > JS Menus > [x] Menu
1  import React, { useState } from 'react';
2  import {
3    Collapse,
4    Navbar,
5    NavbarToggler,
6    NavbarBrand,
7    Nav,
8   NavItem,
9    NavLink,
10   UncontrolledDropdown,
11   DropdownToggle,
12   DropdownMenu,
13   DropdownItem,
14   NavbarText,
15   Container
16 } from 'reactstrap';
17
18 export const Menu = (props) => [
19   const [isOpen, setIsOpen] = useState(false);
20   {
21     const toggle = () => setIsOpen(!isOpen);
22     return(
```



# Implementar layout

```
src > components > JS Menu.js > Menu
1  import React, { useState } from 'react';
2  import {
3    Collapse,
4    Navbar,
5    NavbarToggler,
6    NavbarBrand,
7    Nav,
8    NavItem,
9    NavLink,
10   UncontrolledDropdown,
11   DropdownToggle,
12   DropdownMenu,
13   DropdownItem,
14   NavbarText,
15   Container
16 } from 'reactstrap';
17
18 export const Menu = (props) => [
19   const [isOpen, setIsOpen] = useState(false);
20
21   const toggle = () => setIsOpen(!isOpen);
22   return(
```



# Implementar layout

10. No return faça as alterações:

- Copie entre as div, substituindo a tag h1, todo o conteúdo da Navbar.
- Teste no navegador se carregou corretamente.

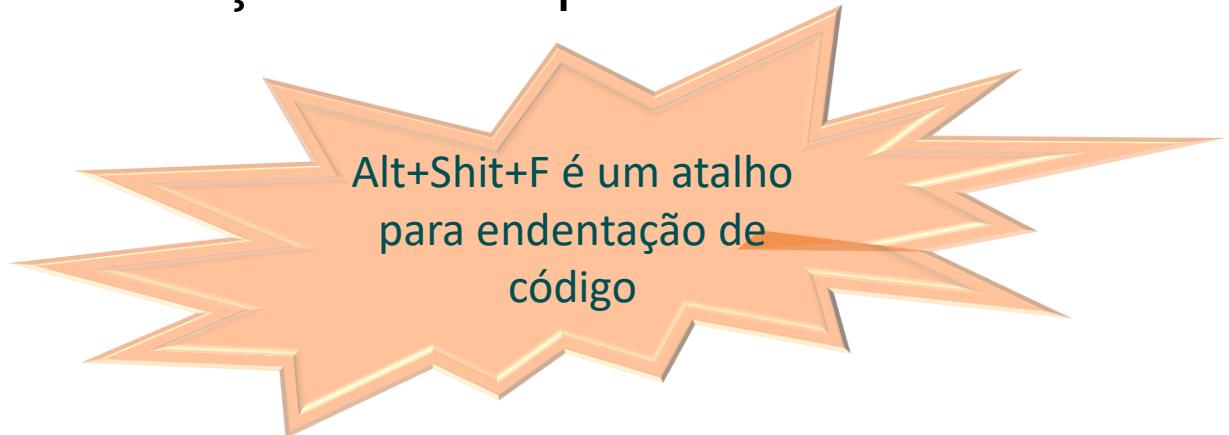
11. Na documentação existem várias opções de cores.

- Utilizar info na propriedade color.
- Utilizar dark ao invés de light para mudar a cor do texto.
- Na parte inferior exclua a linha contendo o componente NavbarText.
- Excluir o dropdown definido entre os componentes UncontrolledDropdown.
- Exclua o componente do github.
- Altere o componente NavItem para Home.
- Altere o componente NavBrand para Services TI Academy.



## Implementar layout

12. Insira o menu dentro do componente container. O Container vai logo abaixo da declaração do componente Navbar.



13. Exclua os componentes que não estão sendo utilizados.
14. Confira o código e execute.



# Implementar layout

```
src > components > JS Menu.js > ...
1  import React, { useState } from 'react';
2  import {
3    Collapse,
4    Navbar,
5    NavbarToggler,
6    NavbarBrand,
7    Nav,
8    NavItem,
9    NavLink,
10   Container
11 } from 'reactstrap';
```

```
13  export const Menu = (props) => {
14    const [isOpen, setIsOpen] = useState(false);
15
16    const toggle = () => setIsOpen(!isOpen);
17    return (
18      <div>
19        <Navbar color="info" dark expand="md">
20          <Container>
21            <NavbarBrand href="/">Services TI Academy</NavbarBrand>
22            <NavbarToggler onClick={toggle} />
23            <Collapse isOpen={isOpen} navbar>
24              <Nav className="mr-auto" navbar>
25                <NavItem>
26                  <NavLink href="/">Home</NavLink>
27                </NavItem>
28              </Nav>
29            </Collapse>
30          </Container>
31        </Navbar>
32      </div>
33    );
34  };
```

# Implementar layout

15. Abra o `index.js` do componente Home para fazer o alinhamento.

```
src > pages > Home > JS index.js > [e] Home
1  import {Container} from 'reactstrap';
2
3  export const Home = () => {
4      return(
5          <div>
6              <Container> ←
7                  <h1>Home</h1>
8              </Container> ←
9          </div>
10     );
11 };

```

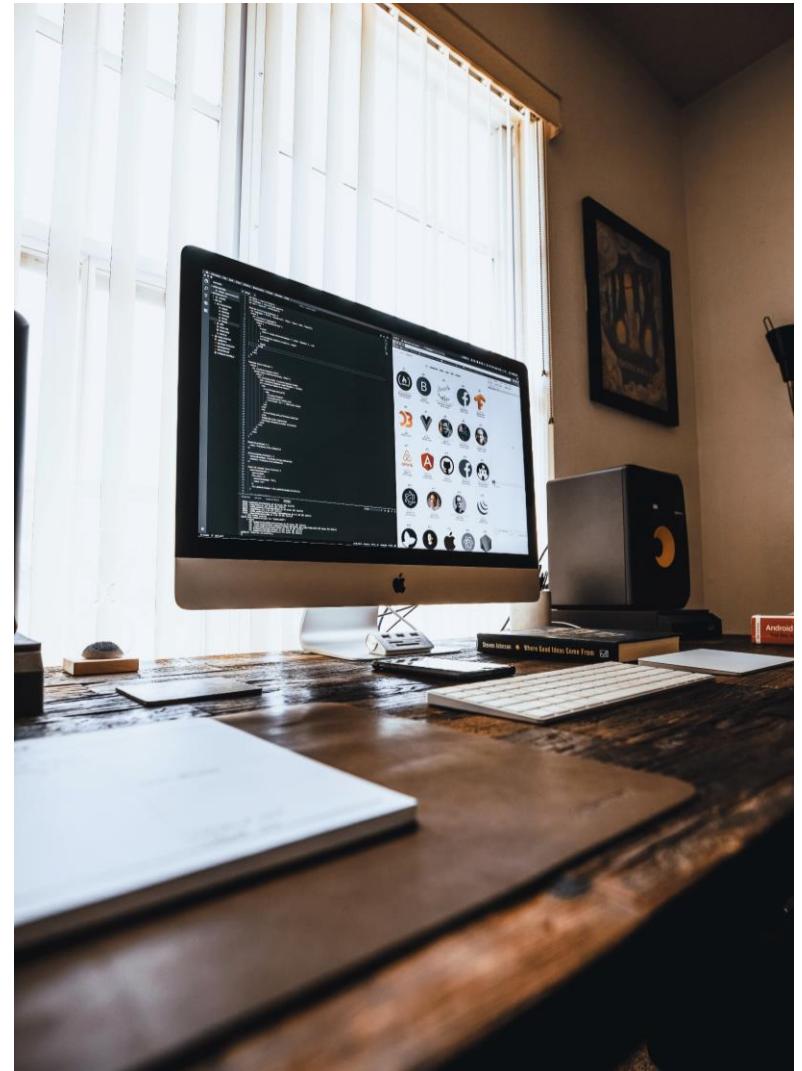
16. Agora, utilize alguns seletores do Bootstrap, como: `d-flex`, `m-auto`, `p-2` e `btn`.



# Implementar layout

```
src > pages > Home > JS index.js > ...
● 1 import {Container} from 'reactstrap';
2
3 export const Home = () => {
4     return(
5         <div>
6             <Container>
7                 <div className="d-flex">
8                     <div className="mr-auto p-2">
9                         <h1>Home</h1>
10                    </div>
11                    <div className="p-2">
12                        <a href="/visualizar-cliente" className="btn btn-success
13                            btn-sm">Visualizar</a>
14                    </div>
15                </Container>
16            </div>
17        );
18    };
19};
```

17. Altere a formatação para que o mouse só fique preenchido quando o usuário passar o mouse sobre o botão.



# Implementar layout

```
1 import {Container} from 'reactstrap';
2
3 export const Home = () => {
4     return(
5         <div>
6             <Container>
7                 <div className="d-flex">
8                     <div className="mr-auto p-2">
9                         <h1>Home</h1>
10                    </div>
11                    <div className="p-2">
12                        <a href="/visualizar-cliente" className="btn btn-outline-success
13                            btn-sm">Visualizar</a>
14                    </div>
15                </div>
16            </Container>
17        </div>
18    );
19};
```

17. Altere a formatação para que o mouse só fique preenchido quando o usuário passar o mouse sobre o botão.



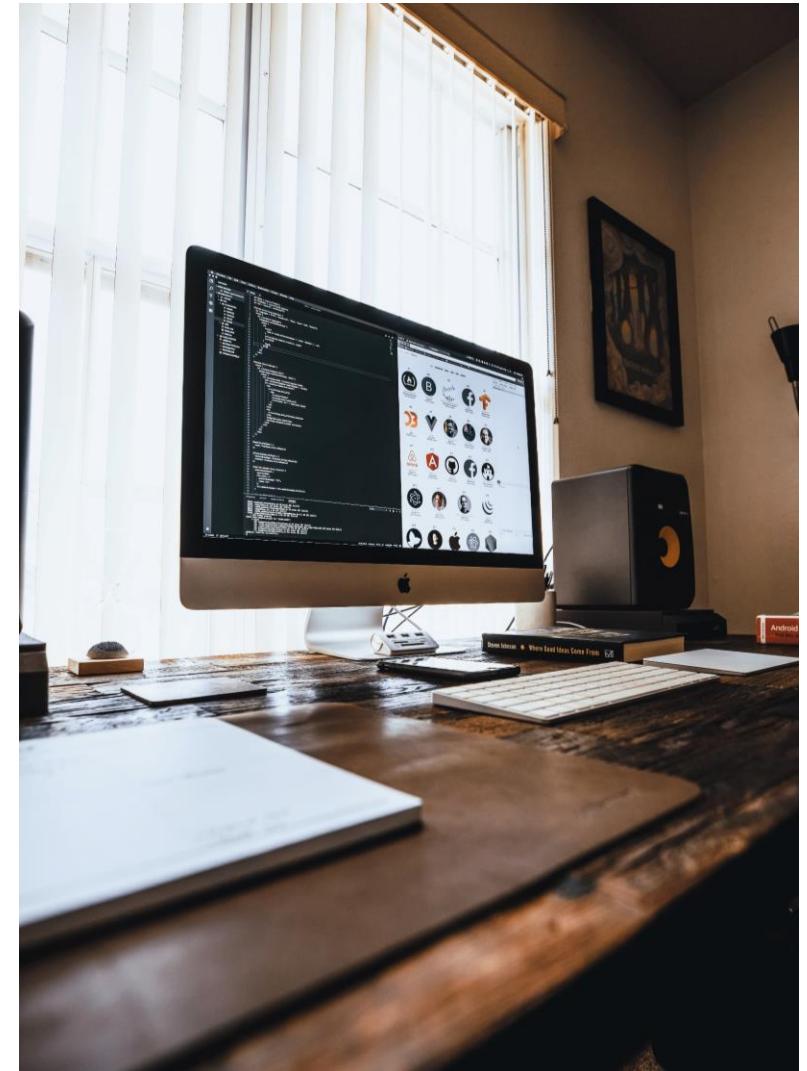
# Exercício

Vamos  
praticar!

Faça um novo botão que chame o componente `VisualizarServico` por intermédio da rota `/visualizar-serviço`.

Faça o mesmo para `VisualizarPedido` usando a rota `/visualizar-pedido`.

Teste e veja a funcionalidade. Ao clicar no botão acesse a página de serviços e retorne para a página principal.



## Incluir a logo do projeto

1. Na pasta public, abra o arquivo index.html.
2. Indique que a página será português Brasil.
3. Para usar o ícone do projeto é fundamental alterar o arquivo favicon.ico.

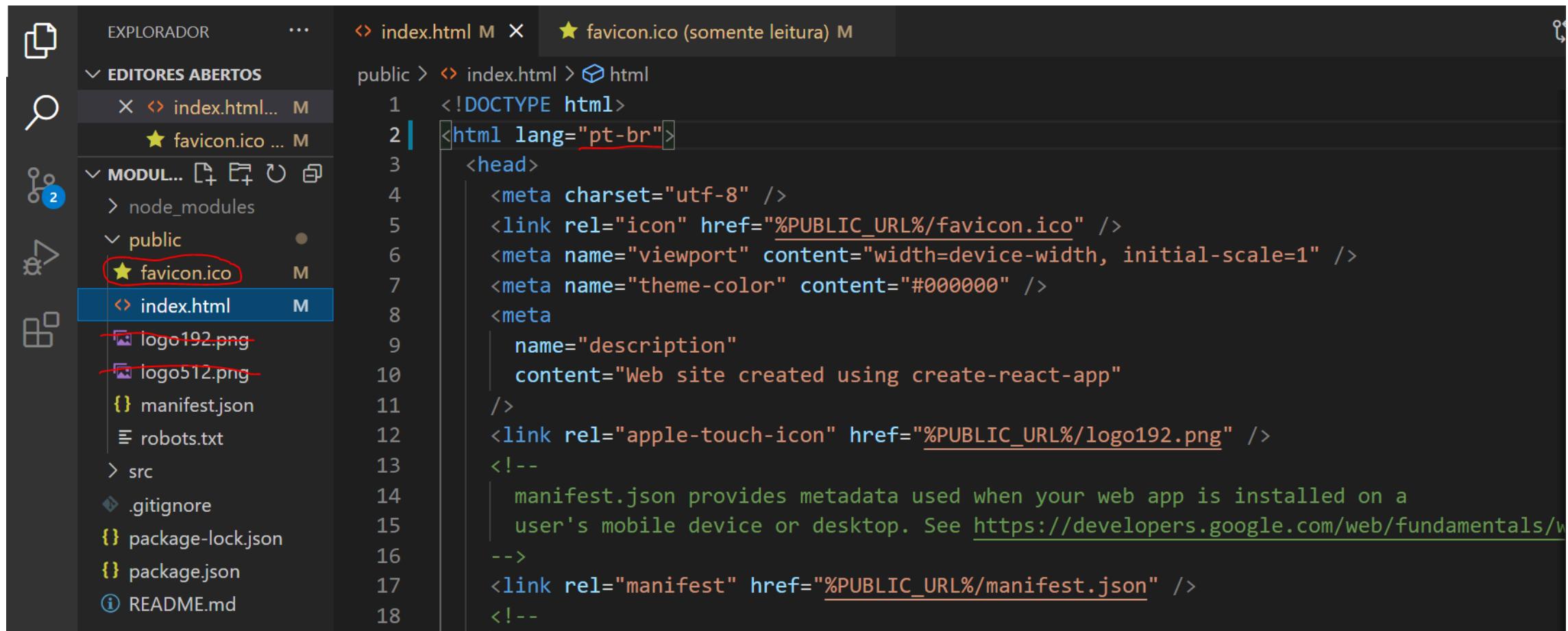
Um site de ícones gratuitos:  
<https://flaticon.com/>

Um site para converter png em ico: <https://convertio.co/>

- a) Exclua o arquivo favicon.ico original.
  - b) Substitua pelo novo arquivo convertido.
  - c) Não se esqueça de manter o nome favicon.ico e no diretório public da aplicação.
4. Pode excluir os arquivos logo192 e logo512 no mesmo diretório.



# Incluir a logo do projeto



The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar, which lists files and folders. In the center is the Editor pane, showing the content of index.html.

**Explorer Sidebar:**

- EXPLORADOR
- EDITORES ABERTOS
  - index.html M
  - favicon.ico (somente leitura) M
- MODULOS
  - node\_modules
  - public
    - favicon.ico M
    - index.html M
    - logo192.png
    - logo512.png
    - manifest.json
    - robots.txt
  - src
  - .gitignore
  - package-lock.json
  - package.json
  - README.md

**Editor (index.html):**

```
public > index.html > html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <meta name="theme-color" content="#000000" />
8     <meta
9       name="description"
10      content="Web site created using create-react-app"
11    />
12    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13    <!--
14      manifest.json provides metadata used when your web app is installed on a
15      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/w
16    -->
17    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18    <!--
```

# Atualizar o repositório

Até aqui tudo bem?



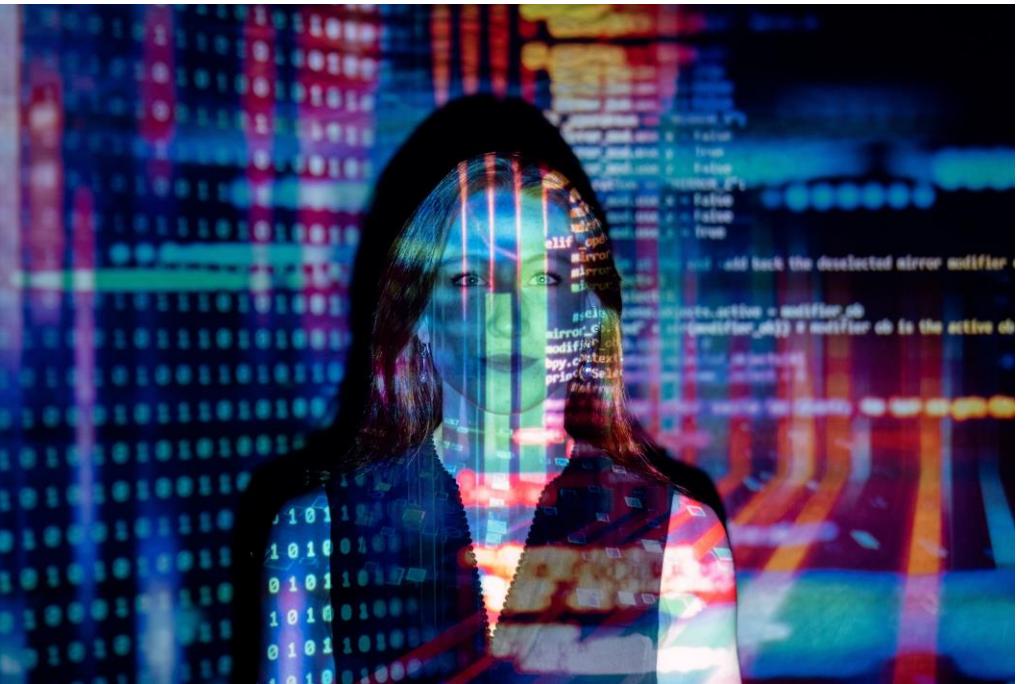
## O que aprendemos...

Você iniciou um novo projeto utilizando o React, criou as páginas da aplicação, implementou o layout da aplicação e incluiu a logo do projeto.



## O que vem em seguida...

Vamos implementar o CRUD da nossa aplicação para funcionar via web.





@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Front-end com React

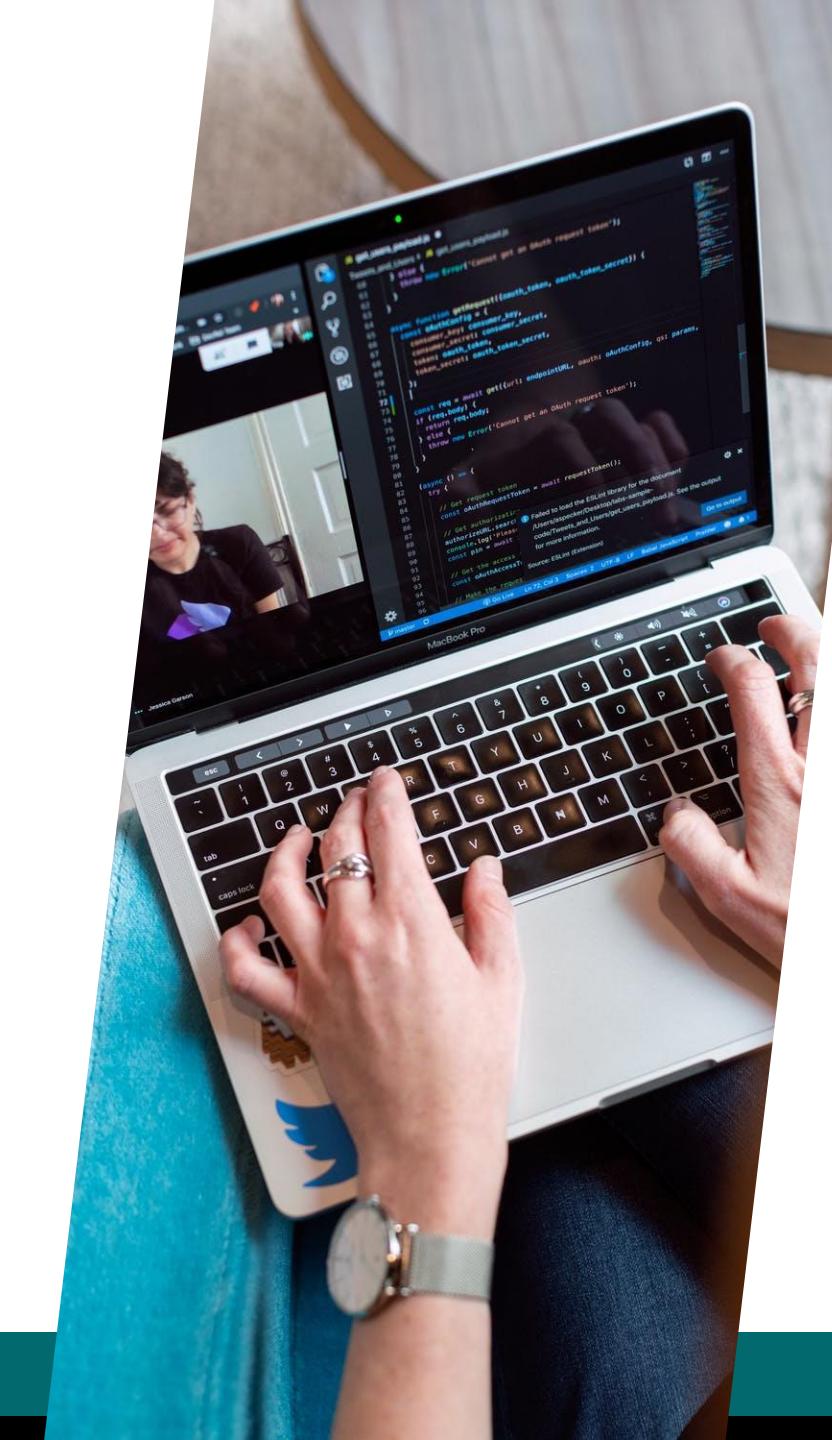
Dia 2 | Profº. Erinaldo

## Antes de começar...

Na última aula você iniciou uma aplicação front-end com ReactJS.

Concluiu a implementação de um componente Menu utilizando o framework Reactstrap.

Agora você vai acessar o banco de dados e manipular os dados a partir do front-end.

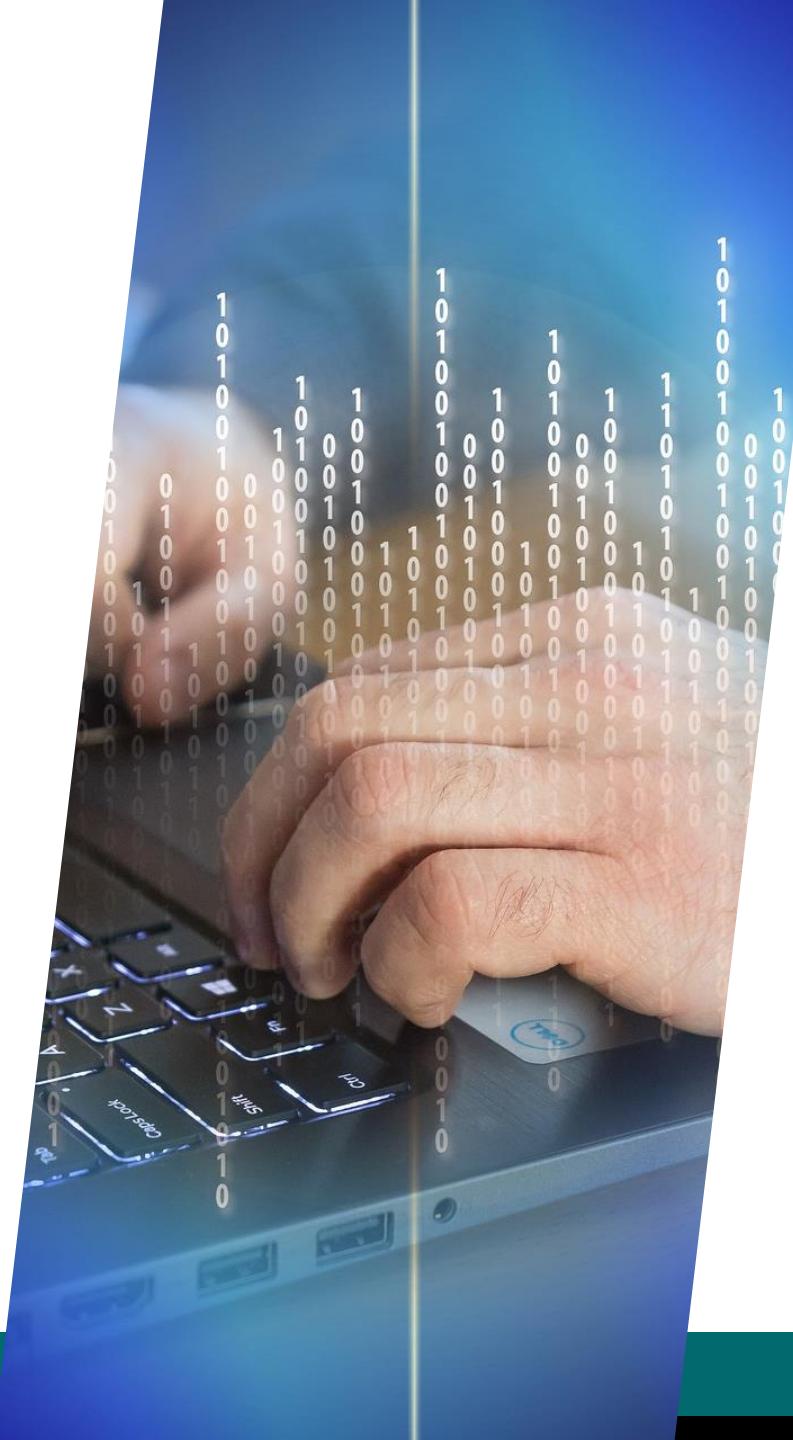


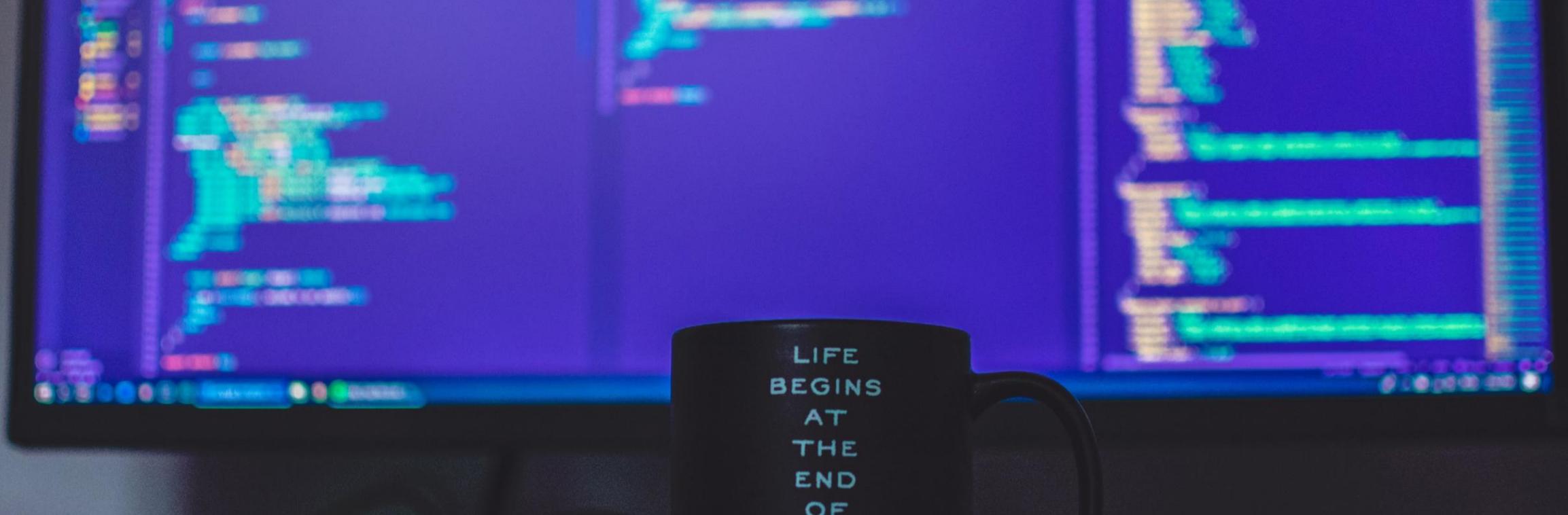
## Dia 2: Implementar as consultas

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

4<sup>a</sup> semana





@tiacademybrasil

# Implementar os consultas

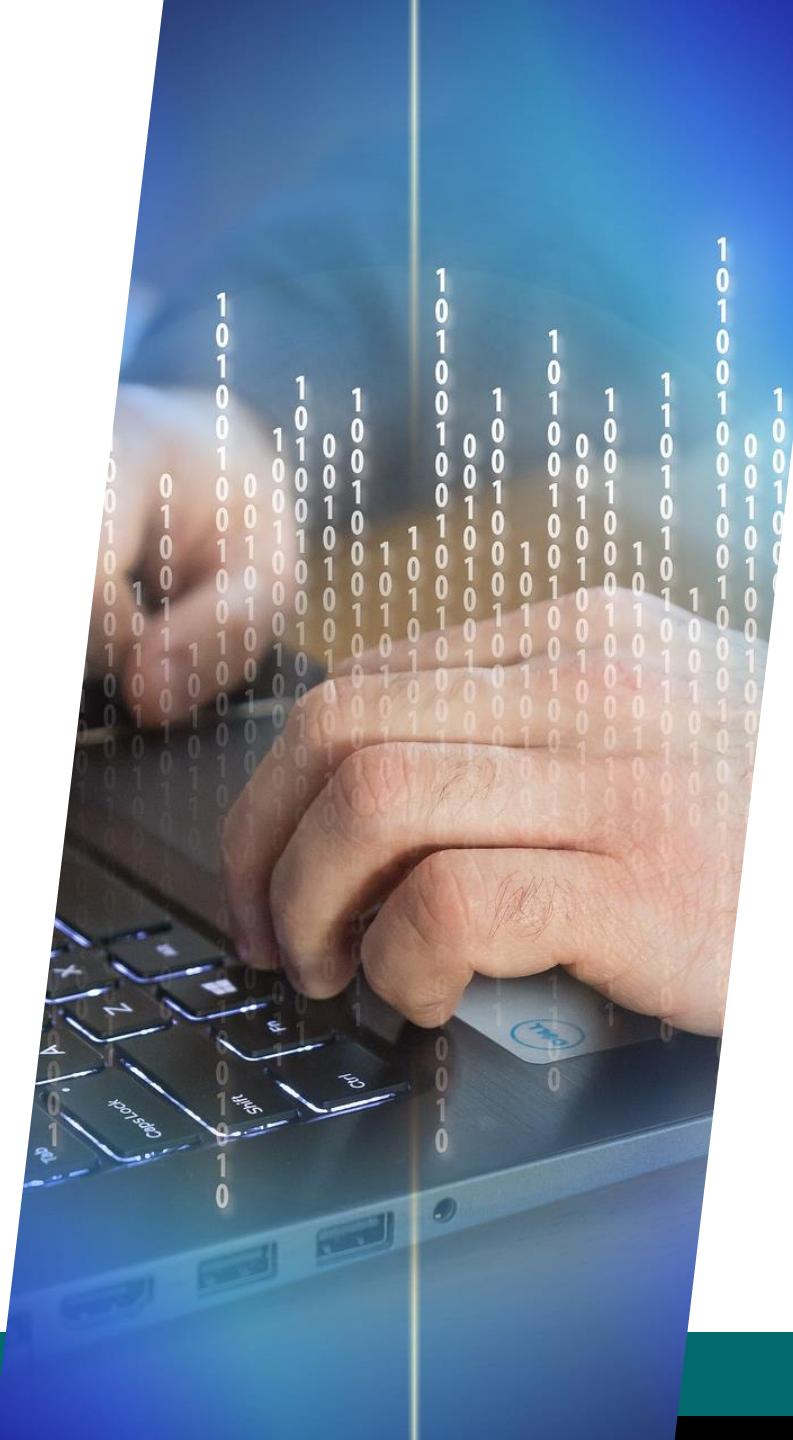
Aula 3 | Vamos praticar?

## Dia 2: Implementar as consultas

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



## Alterar layout da página visualizar serviço

1. Importar, no index.js do componente VisualizarServiço, componentes do reactstrap.
2. Abaixo das div coloque tudo dentro de um Container.
3. Na documentação do reactstrap acesso a documentação do componente Table, acesse:  
<https://reactstrap.github.io/components/tables/>.



# Alterar layout da página visualizar serviço

```
src > pages > Servico > VisualizarServico > JS index.js > [ej] VisualizarServico  
1   import { Container, Table } from 'reactstrap';  
2  
3   export const VisualizarServico = () => {  
4       return(  
5           <div>  
6               <Container>  
7                   <h1>Visualizar informações do serviço</h1>  
8                   <Table>  
9                       |  
10                      </Table>  
11                  </Container>  
12             </div>  
13     );  
14};
```



4. Utilize uma tabela com o formato de linhas com cores alternadas, *striped rows*.

## Alterar layout da página visualizar serviço

5. Altere as colunas de cabeçalho da tabela para as colunas do objeto Servico.
6. Crie uma quarta coluna, além de id, nome e descrição, chamada ações.

```
src > pages > Servico > VisualizarServico > JS index.js > ...  
1  import { Container, Table } from 'reactstrap';  
● 2  
3  export const VisualizarServico = () => {  
4    return(  
5      <div>  
6        <Container>  
7          <h1>Visualizar informações do serviço</h1>  
8          <Table striped>  
9            <thead>  
10           <tr>  
11             <th>ID</th>  
12             <th>Nome</th>  
13             <th>Descrição</th>  
14             <th>Ações</th>  
15           </tr>  
16         </thead>  
17         </Table>  
18       </Container>  
19     </div>  
20   );  
21 };
```



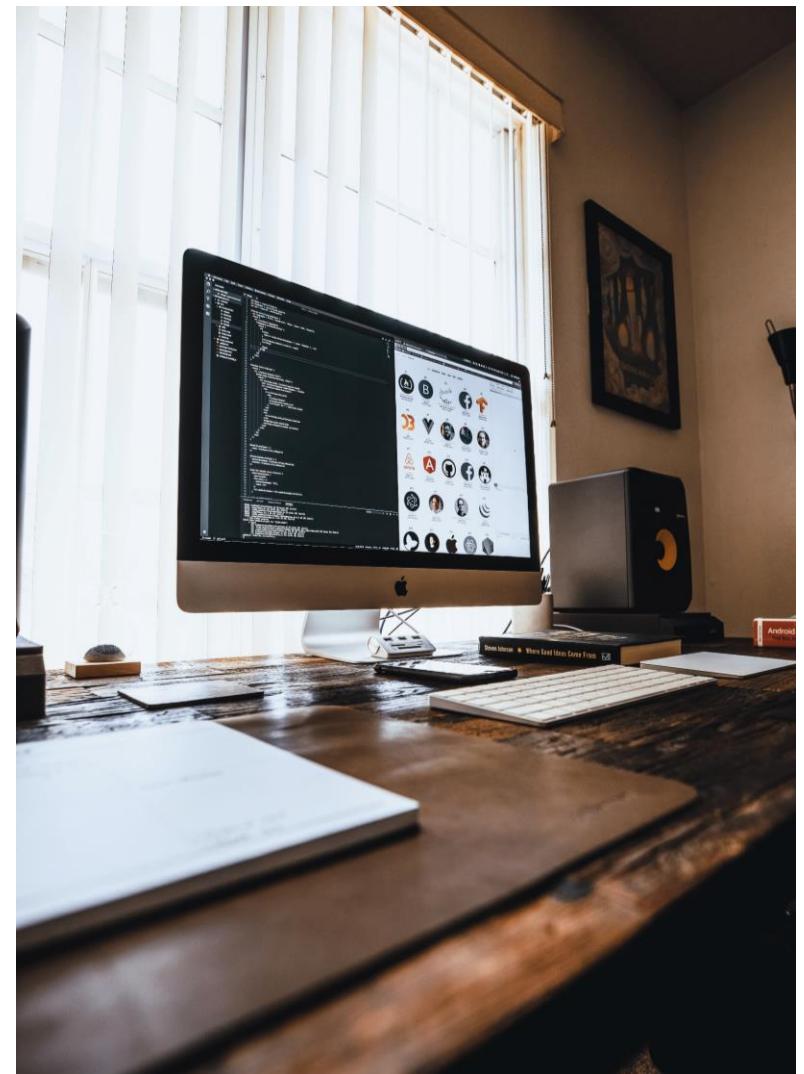
## Alterar layout da página visualizar serviço

7. Copie e cole as informações das linhas da tabela disponíveis no próprio material do componente Table do reactstrap. Essas informações devem ficar dentro do corpo da tabela (`tbody`) e abaixo do cabeçalho da tabela (`thead`).



# Alterar layout da página visualizar serviço

```
6 <Container>
7   <h1>Visualizar informações do serviço</h1>
8   <Table striped>
9     <thead>
10    <tr>
11      <th>ID</th>
12      <th>Nome</th>
13      <th>Descrição</th>
14      <th>Ações</th>
15    </tr>
16  </thead>
17  <tbody>
18    <tr>
19      <td>1</td>
20      <td>Mark</td>
21      <td>Otto</td>
22      <td>@mdo</td>
23    </tr>
24    <tr>
25      <td>2</td>
26      <td>Jacob</td>
27      <td>Thornton</td>
28      <td>@fat</td>
29    </tr>
30    <tr>
31      <td>3</td>
32      <td>Larry</td>
33      <td>the Bird</td>
34      <td>@twitter</td>
35    </tr>
36  </tbody>
37 </Table>
38 </Container>
```



## O que vem em seguida

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

**4<sup>a</sup> semana**

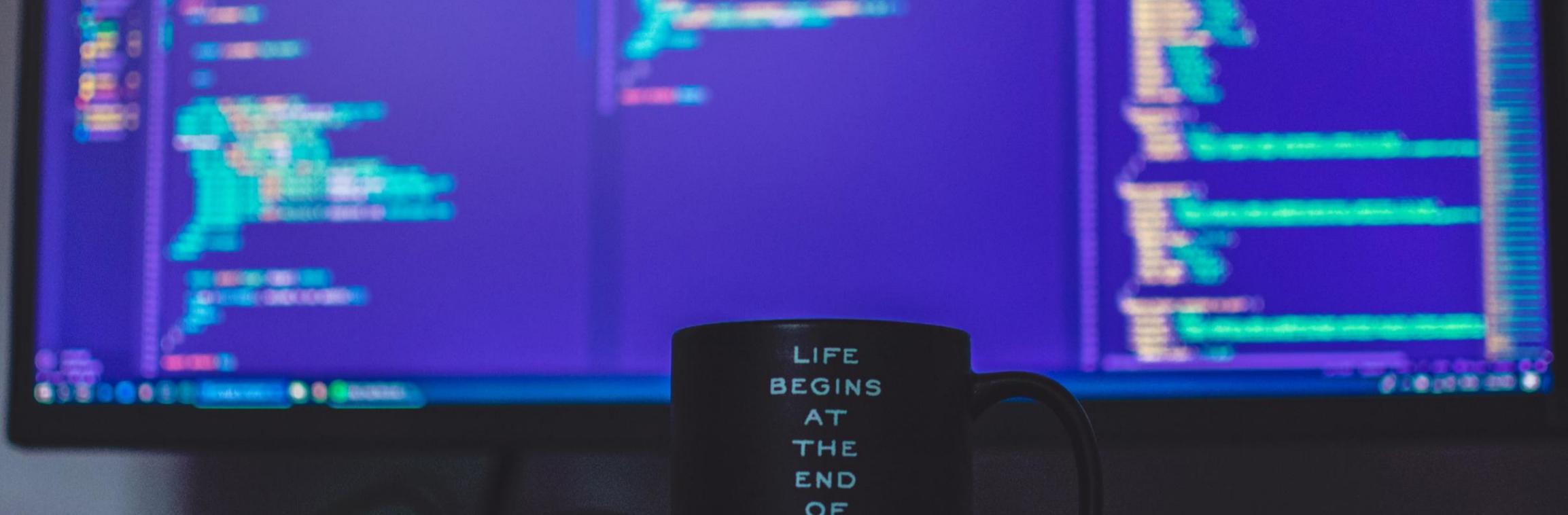




@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Implementar os consultas

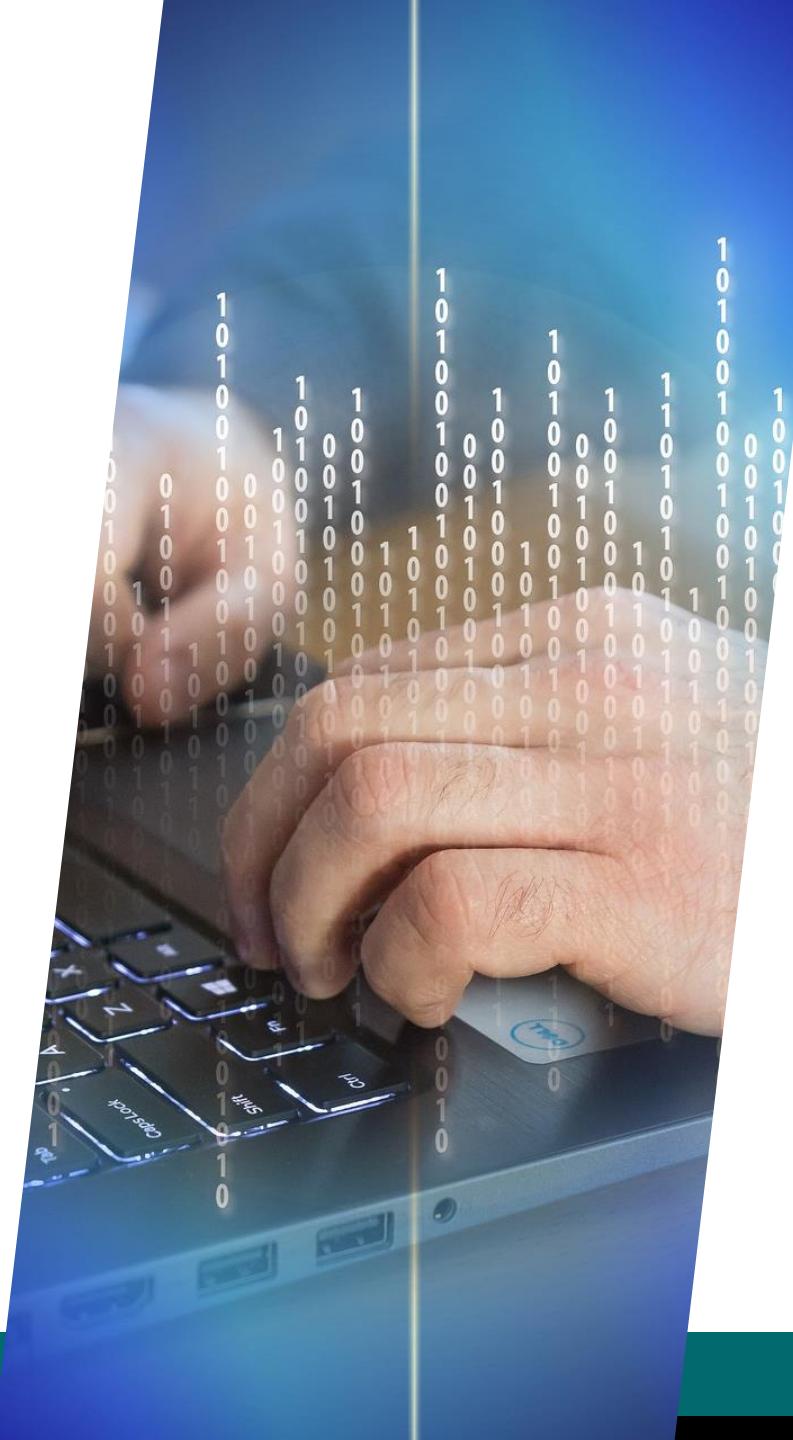
Aula 4 | Vamos praticar?

## Dia 2: Implementar as consultas

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



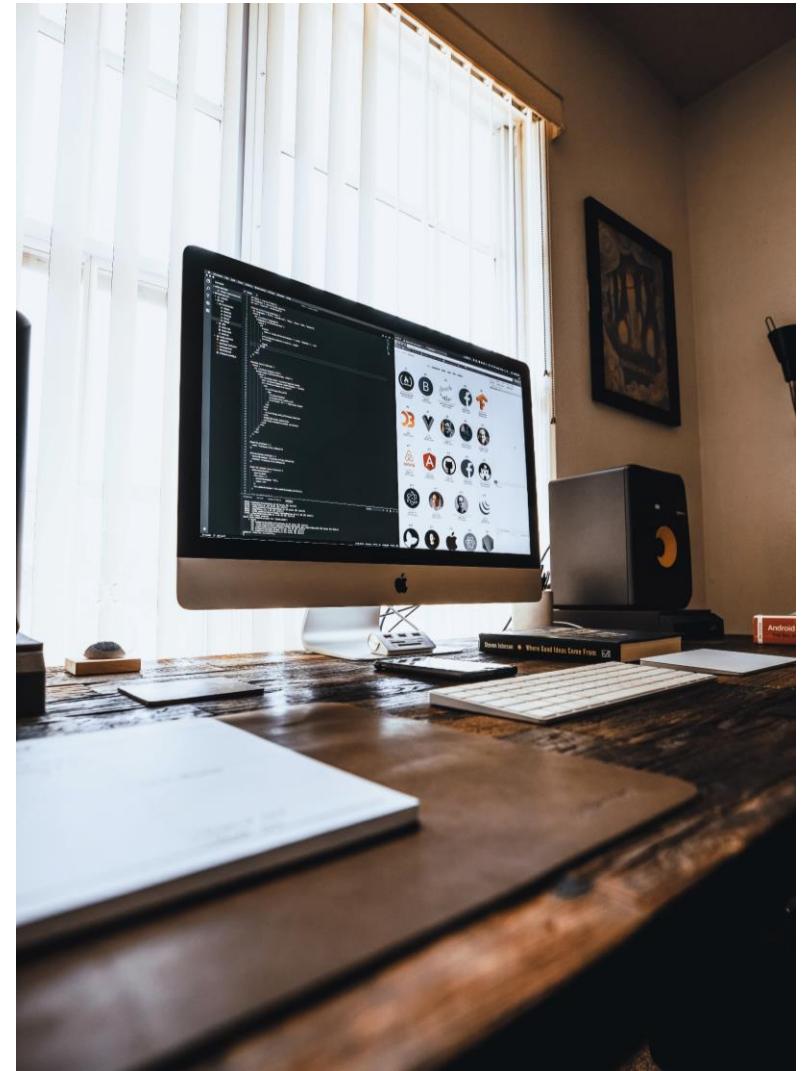
## Conectar-se à API

1. Abra um segundo editor de código.
2. Abra nesse segundo editor o projeto na pasta ServicesTI.
3. Verifique se a porta do servidor da aplicação é 3001.

```
197 | let port=process.env.PORT || 3001;
198 | app.listen(port,(req,res)=>{
199 |   |   console.log('Servidor ativo');
200 | });


```

4. Execute a aplicação e teste as rotas no Postman. Nosso interesse agora é a rota com o método get nos serviços.



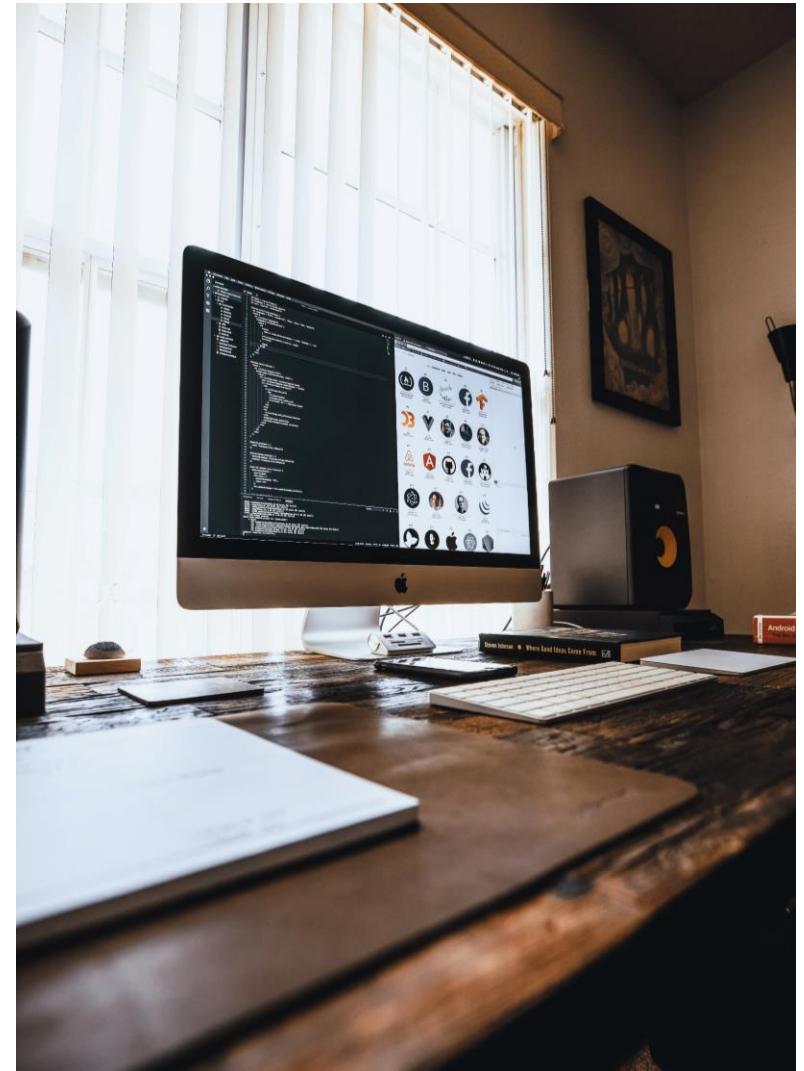
# Conecitar-se à API

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main workspace is titled "modulo3 / Consultar" and contains a GET request for "http://localhost:3001/listaservicos". The "Body" tab is selected, showing the response body in JSON format. The response status is 200 OK, and the response body is:

```
1  [
2   "servicos": [
3     {
4       "id": 5,
5       "nome": "Rede de computadores",
6       "descricao": "Desenvolve projeto de redes locais e implementa a solução",
7       "createdAt": "2021-08-20T20:31:27.000Z",
8       "updatedAt": "2021-08-20T20:31:27.000Z"
9     },
10    {
11      "id": 2,
12      "nome": "Nodejs",
13      "descricao": "Desenvolvimento de aplicação back-end",
14      "createdAt": "2021-08-17T18:57:34.000Z",
15      "updatedAt": "2021-08-17T18:57:34.000Z"
16    },
17    {
18      "id": 6,
19      "nome": "Java SE",
20      "descricao": "Programação orientada a objetos",
21      "createdAt": "2021-08-23T18:35:33.000Z",
22      "updatedAt": "2021-08-30T12:05:00.000Z"
23    },
24    {
25      "id": 4,
```

## Conectar-se à API

5. Acesse a documentação do axios em <https://www.npmjs.com/package/axios#installing>.
6. Instale a dependência axios no projeto modulo4, executando o comando npm install --save axios.
7. Após a instalação abra o arquivo index.js do VisualizarServico para incluir a dependência no projeto.
8. Visualize na documentação os métodos de requisição utilizados pelos axios, em <https://www.npmjs.com/package/axios#request-method-aliases>.



## Conectar-se à API

```
src > pages > Servico > VisualizarServico > JS index.js > [VisualizarServico]  
1 import axios from 'axios';  
2 import { Container, Table } from 'reactstrap';  
3  
4 export const VisualizarServico = () => {  
5     return(
```

9. No diretório src crie um diretório config para configuração. Crie um arquivo chamado index.js.
10. Insira a linha de código contendo o endereço onde roda a API do projeto.

```
src > config > JS index.js > ...  
1 export const api = "http://localhost:3001";
```



# Utilizar o método GET para retornar dados

1. Acesso o arquivo `index.js` do componente `VisualizarServico` e importe a api para ele.

```
src > pages > Servico > VisualizarServico > JS index.js > [🔗] VisualizarServico
  1 import axios from 'axios';
  2 import { Container, Table } from 'reactstrap';
  3
 4 import { api } from '../../../../../config';
  5
  6 export const VisualizarServico = () => {
```

2. Crie uma constante no formato lista que receberá os dados que a API vai retornar.
3. Crie uma função `getServicos` para devolver os dados.



# Utilizar o método GET para retornar dados

```
7  export const VisualizarServico = () => {
8
9      //iniciar um array vazio que recebe os dados
10     const [data, setData] = useState([]);
11
12     //função assíncrona getServicos
13     const getServicos = async() =>{
14         await axios.get(api)
15         .then((respose) =>{
16             console.log(respose.data.servicos);
17             setData(respose.data.servicos);
18         })
19         .catch(() =>{
20             console.log("Erro: Sem conexão com a API.")
21         })
22     }
23
24     //instanciar a função
25     useEffect(()=>{
26         getServicos();
27     });
28 }
```



## Utilizar o método GET para retornar dados

4. Faça uma alteração no arquivo index.js da aplicação.

```
src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import App from './App';
4
5 import 'bootstrap/dist/css/bootstrap.min.css';
6
7 ReactDOM.render(
8   <React.Fragment>
9     <App />
10    </React.Fragment>,
11    document.getElementById('root')
12 );
```

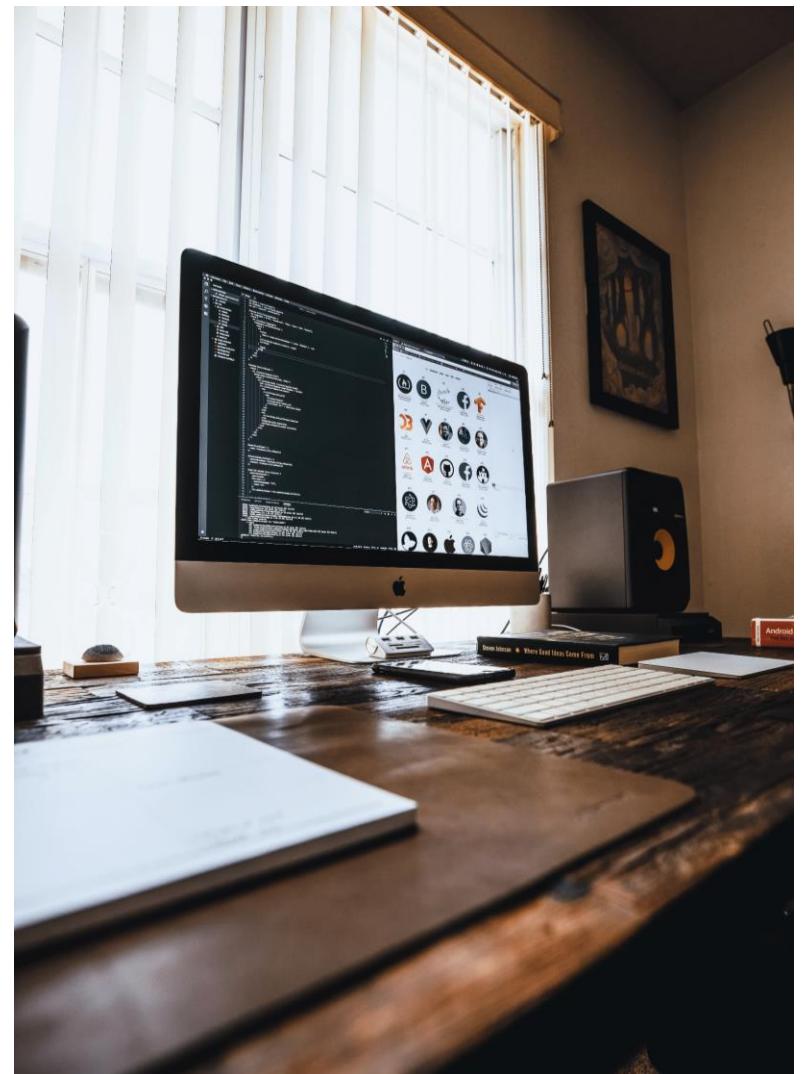
Utilizado quando um componente retorna múltiplos elementos.



# Utilizar o método GET para retornar dados

## 5. Alterar a página para receber os dados da consulta.

```
33 <Table striped>
34   <thead>
35     <tr>
36       <th>ID</th>
37       <th>Nome</th>
38       <th>Descrição</th>
39       <th>Ações</th>
40     </tr>
41   </thead>
42   <tbody>
43     {data.map(item => (
44       <tr key={item.id}>
45         <td>{item.id}</td>
46         <td>{item.nome}</td>
47         <td>{item.descricao}</td>
48         <td className="text-center">Botão</td>
49       </tr>
50     )));
51   </tbody>
52 </Table>
```



# Utilizar o método GET para retornar dados

Serivices TI Academy Home

## Visualizar informações do serviço

ID	Nome	Descrição	Ações
5	Rede de computadores	Desenvolve projeto de redes locais e implementa a solução	Botão
2	Nodejs	Desenvolvimento de aplicação back-end	Botão
6	Java SE	Programação orientada a objetos	Botão
4	Infraestrutura	Instalação, configuração e manutenção de hardware	Botão
1	HTML/CSS/JS	Páginas estáticas e dinâmicas estilizadas	Botão
3	Delphi	Manutenção e suporte a sistemas legados em Delphi	Botão



## O que vem em seguida

1. Alterar layout da página visualizar serviço
  2. Conectar-se à API
  3. Utilizar o método GET para retornar dados
  4. Mostrar o erro na página
  5. Criar os botões das ações
- TOTAL: 22 horas**
- 4<sup>a</sup> semana**

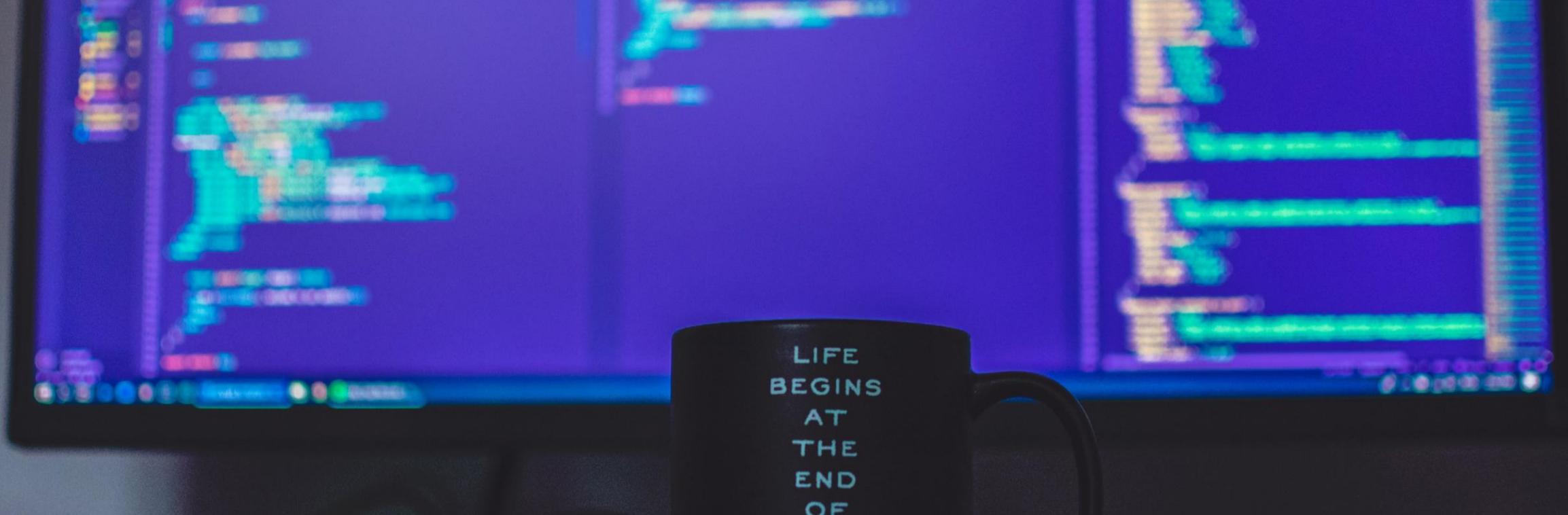




@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Implementar os consultas

Aula 5 | Vamos praticar?

## Dia 2: Implementar as consultas

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



## Mostrar o erro na página

Ao parar o servidor e tentar executar aparece um erro no *console log*, Erro: Sem conexão com a API.

O objetivo é fazer esse erro aparecer na página da aplicação. Portanto,

1. Iniciar um *array* vazio para receber as mensagens de *status* da aplicação.

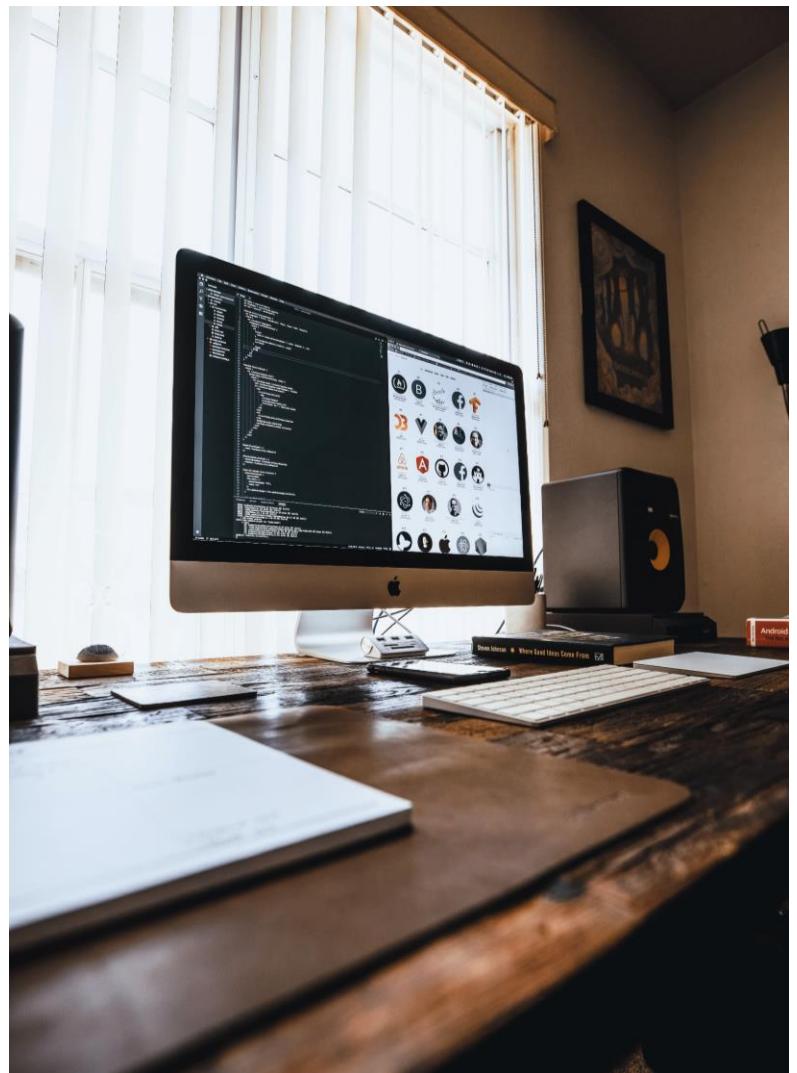
```
9      //iniciar um array vazio que recebe os dados
10     const [data, setData] = useState([]);
11
12     const [status, setStatus] = useState({
13       type: '',
14       message: ''
15     })
16
```



## Mostrar o erro na página

2. Alterar na função getServicos para passar a mensagem de erro.

```
17 //função assíncrona getServicos
18 const getServicos = async () => {
19   axios.get(api)
20     .then((respose) => {
21       console.log(respose.data.servicos);
22       setData(respose.data.servicos);
23     })
24     .catch(() => {
25       setStatus({
26         type: 'error',
27         message: 'Erro: Sem conexão com a API.'
28       })
29     })
30 }
```



# Mostrar o erro na página

## 3. Incluir um alerta antes da tabela.

```
41         <h1>Visualizar informações do serviço</h1>
42     </div>
43     <Alert color="danger">
44         {status.message}
45     </Alert>
46     <Table striped>
47         <thead>
48             <tr>
49                 <th>ID</th>
50                 <th>Nome</th>
51                 <th>Descrição</th>
52                 <th>Ações</th>
53             </tr>
54         </thead>
55         <tbody>
56             {data.map(item => (
57                 <tr key={item.id}>
58                     <td>{item.id}</td>
59                     <td>{item.nome}</td>
60                     <td>{item.descricao}</td>
61                     <td className="text-center">Botão</td>
62                 </tr>
63             ))}
64         </tbody>
```



# Mostrar o erro na página

Ao executar a aplicação com o servidor da API parado...

Serivices TI Academy Home

## Visualizar informações do serviço

Erro: Sem conexão com a API.

ID	Nome	Descrição	Ações
----	------	-----------	-------

Na execução sem erro, para não aparecer a barra de alerta...

```
43 {status.type === 'error' ? <Alert color="danger">{status.message}</Alert> : ""}
44
45 <Table striped>
46   <thead>
47     <tr>
48       <th>ID</th>
49       <th>Nome</th>
50       <th>Descrição</th>
51       <th>Ações</th>
52     </tr>
53   </thead>
```



## Um parênteses no front-end

Vamos implementar uma consulta que retorne todos os itens de pedidos relacionados a um serviço.

```
131 ˜ app.get('/servico/:id/pedidos', async(req, res)=>{
132      await itempedido.findAll({
133          where: {ServiçoId: req.params.id}})
134      .then(item =>{
135          return res.json({
136              error: false,
137              item
138          });
139      }).catch(function(erro){
140          return res.status(400).json({
141              error: true,
142              message: "Erro: não foi possível conectar!"
143          });
144      });
145  });


```

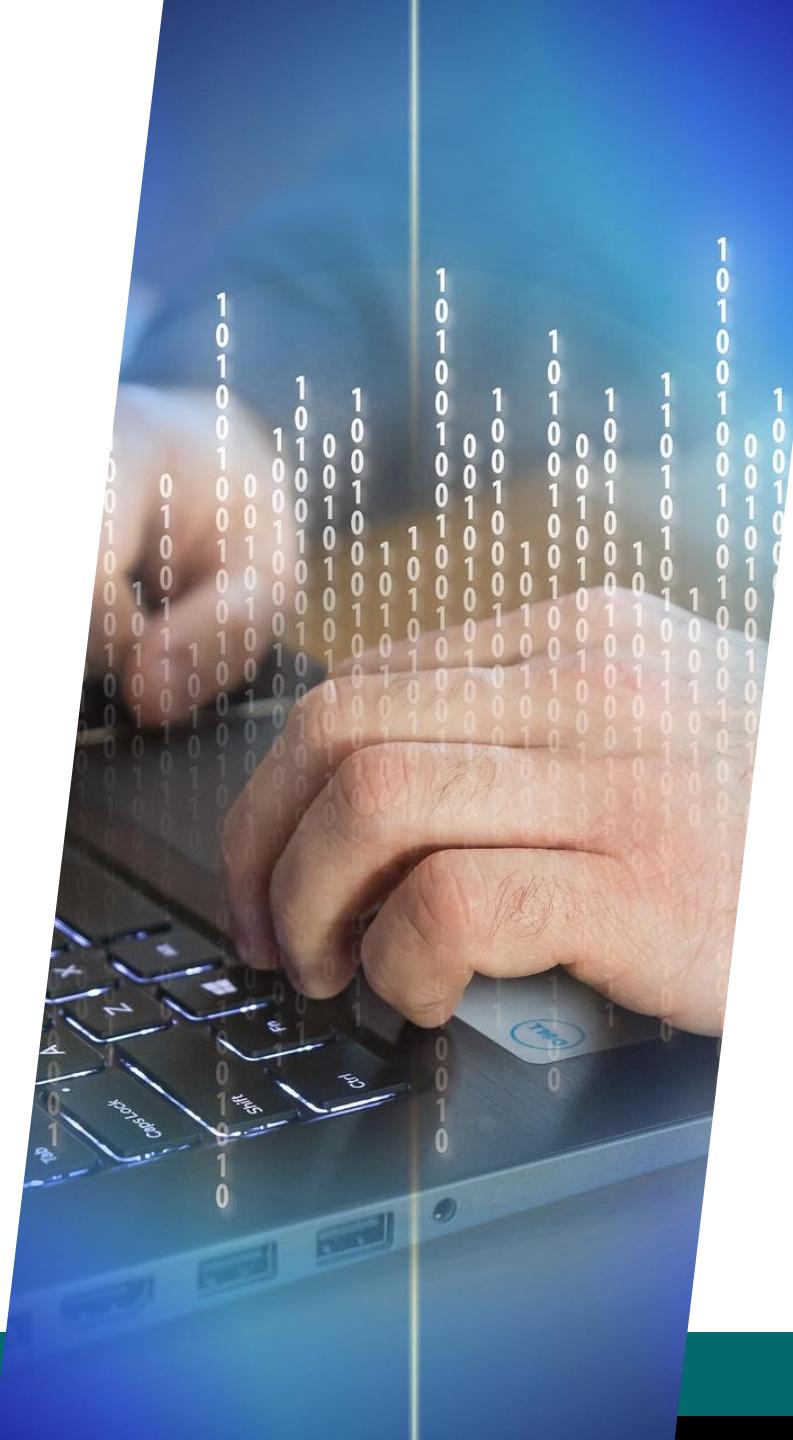


## O que vem em seguida

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

**4<sup>a</sup> semana**

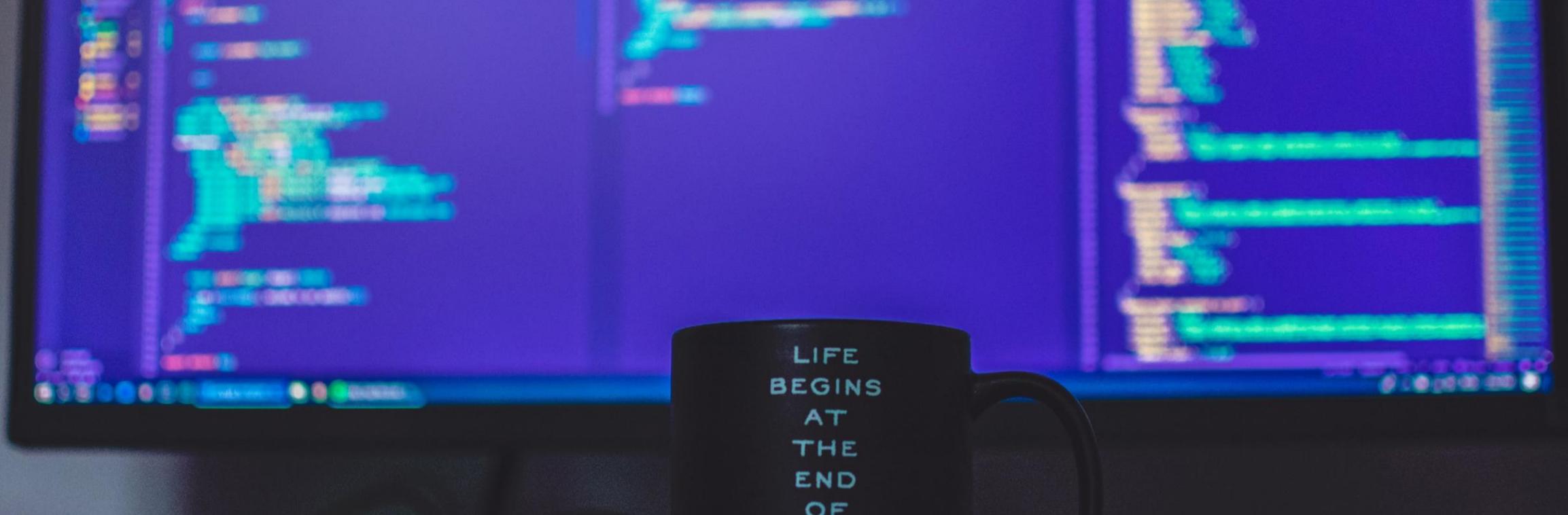




@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Implementar os consultas

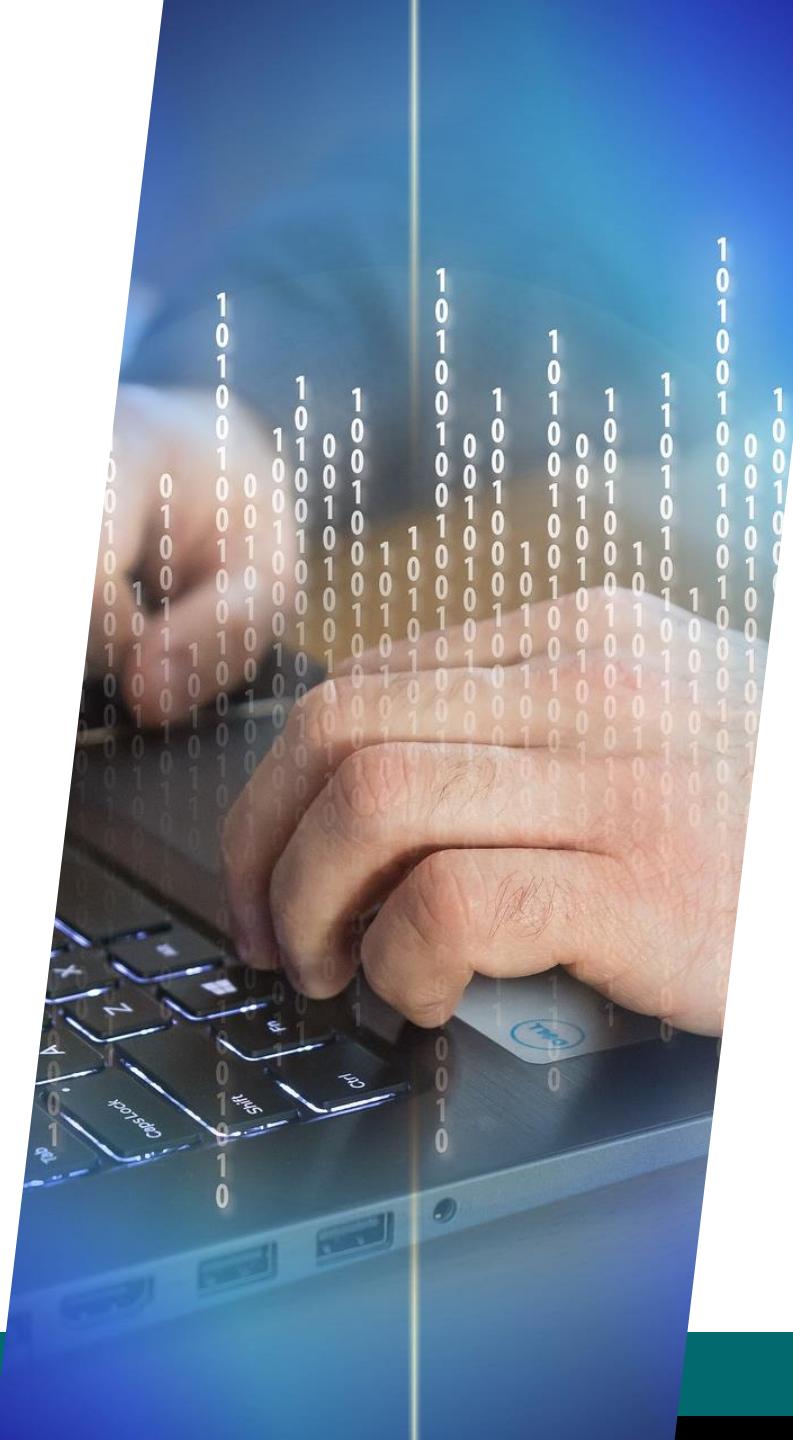
Aula 6 | Vamos praticar?

## Dia 2: Implementar as consultas

1. Alterar layout da página visualizar serviço
2. Conectar-se à API
3. Utilizar o método GET para retornar dados
4. Mostrar o erro na página
5. Criar os botões das ações

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



## Criar os botões das ações

1. Incluir o componente Link do react-router-dom.
2. Crie o link para consultar um serviço pelo id.
3. Utilize um botão para o link.

```
56 <tbody>
57   {data.map(item => (
58     <tr key={item.id}>
59       <td>{item.id}</td>
60       <td>{item.nome}</td>
61       <td>{item.descricao}</td>
62       <td className="text-center">
63         <Link to={"/visualizar-pedido/" + item.id}>
64           <button className="btn btn-outline-primary btn-sm">Consultar</button>
65         </Link>
66       </td>
67     </tr>
68   ))}
</tbody>
```

Vai ser necessário criar em Servico um componente Item, um index.js padrão para ser exportado.



## Criar os botões das ações

4. Verifique se o componente Servico está recebendo o id.

```
src > pages > Servico > Servico > JS index.js > ...
1  export const Servico = (props) => {
2    console.log(props.match.params.id);
3    return(
4      <div>
5        <h1>Visualizar informações do serviço</h1>
6      </div>
7    );
8  };
```

5. Execute e verifique se o id está sendo recebido, utilize o Inspecionar.
6. Funcionou? Comente a linha 2.



## Criar os botões das ações

7. Para passar o `id` recebido por parâmetro para o componente.

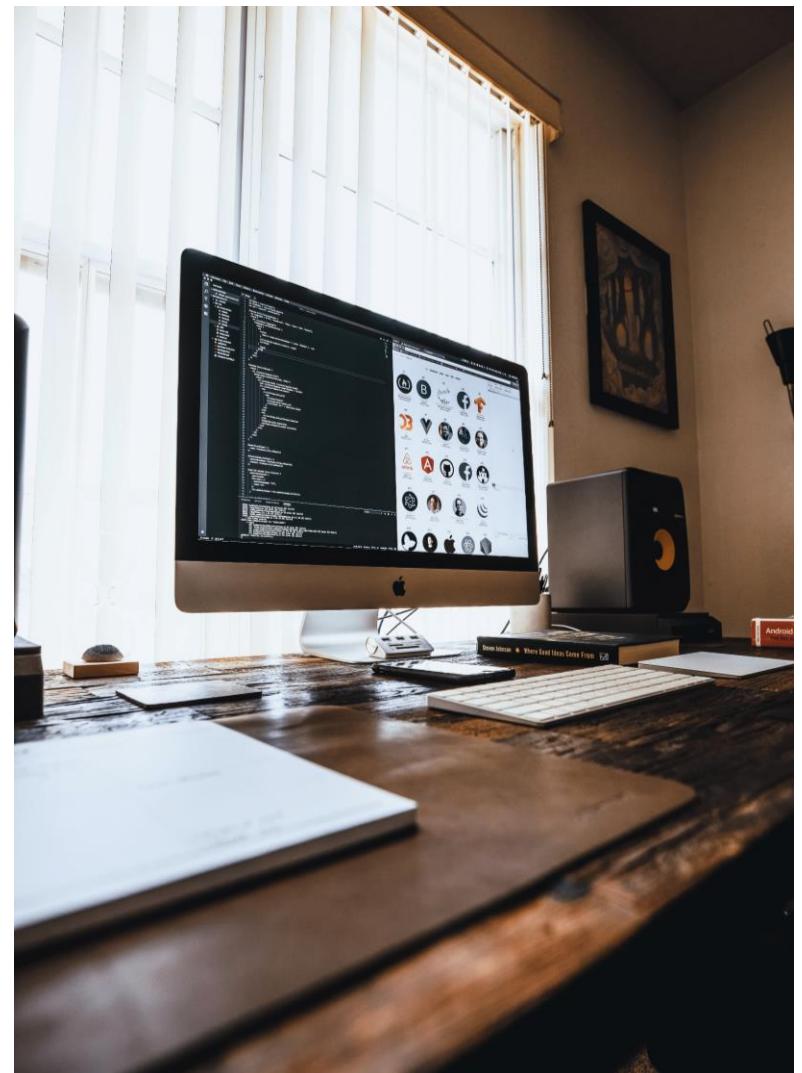
```
src > pages > Servico > Servico > JS index.js > [ej] Servico
1  import { useState } from "react/cjs/react.development";
2
3  export const Servico = (props) => [
4      //console.log(props.match.params.id);
5
6      const [data, setData] = useState([]);
7      const [id, setId] = useState(props.match.params.id);
8
9      return(
10         <div>
11             <h1>Visualizar informações do serviço</h1>
12         </div>
13     );
14 ];
```



## Criar os botões das ações

8. Para passar o `id` recebido por parâmetro para o componente.

```
6  export const Servico = (props) => {
7      //console.log(props.match.params.id);
8
9      const [data, setData] = useState([]);
10     const [id, setId] = useState(props.match.params.id);
11
12    useEffect(() => {
13        const getServico = async() =>{
14            await axios.get(api+"/totalservico/"+id)
15            .then((response) => [
16                console.log(response.data_pedidos);
17                setData(response.data_pedidos);
18            ])
19            .catch(() =>{
20                console.log("Erro: Sem conexão com a API.");
21            })
22        }
23        getServico();
24    }, [id]);
```

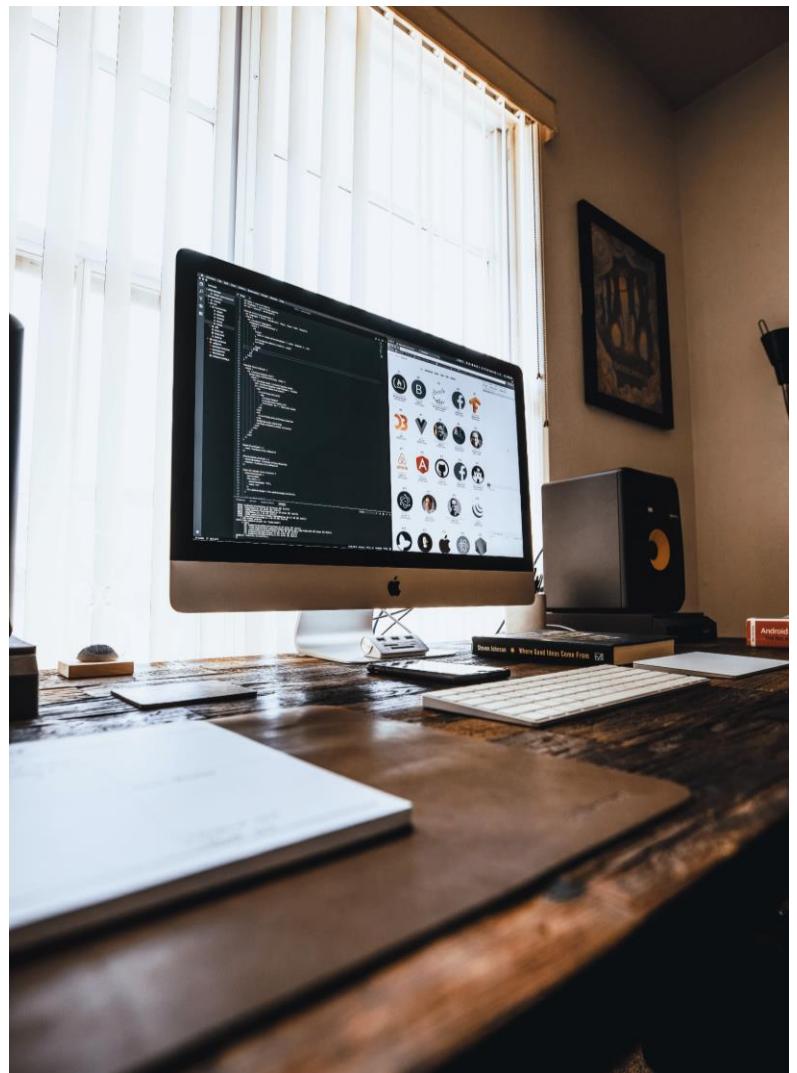


## Criar os botões das ações

9. Altere o *return* do componente Servico.

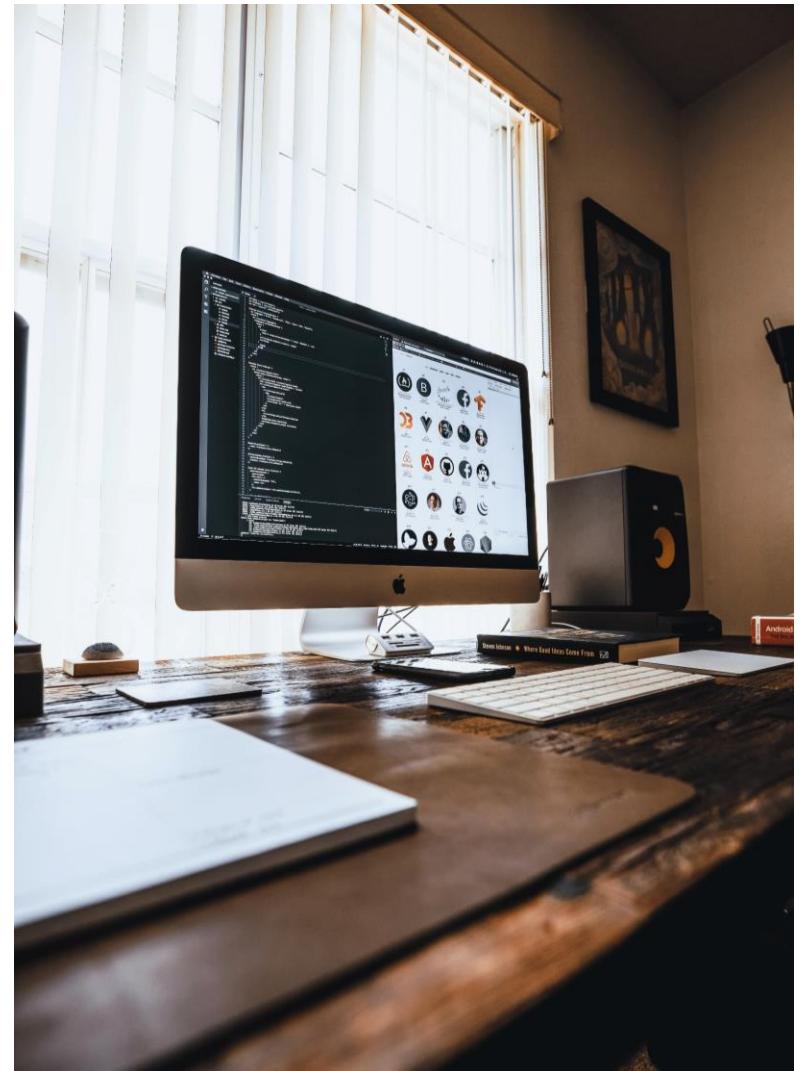
```
27     return (
28         <div className="d-flex">
29             <div className="mr-auto p-2">
30                 <h1>Quantidade de pedidos do serviço</h1>
31             </div>
32             <div className="p-2">
33                 <Link to="/visualizar-servico" className="btn btn-outline-success
34                                btn-sm">Listar</Link>
35             </div>
36         </div>
37     );
38 };
```

10. Imprime na página o valor retornado pela API.



# Criar os botões das ações

```
28     return (
29       <div>
30         <Container>
31           <div className="d-flex">
32             <div className="mr-auto p-2">
33               <h1>Quantidade de pedidos do serviço</h1>
34             </div>
35             <div className="p-2">
36               <Link to="/visualizar-servico" className="btn btn-outline-success
37                 btn-sm">Listar</Link>
38             </div>
39           </div>
40           <dl className="row">
41             <dt className="col-sm-3">Valor total</dt>
42             <dd className="col-sm-9">{data}</dd>
43           </dl>
44         </Container>
45       </div>
46     );
47   );
```



10. Exibir mensagem de erro.

# Criar os botões das ações

```
14  const [status, setStatus] = useState({  
15      type: '',  
16      message: ''  
17  })  
18  
19  useEffect(() => {  
20      const getServico = async () => {  
21          await axios.get(api + "/totalservico/" + id)  
22              .then((response) => {  
23                  //console.log(response.data.pedidos);  
24                  setData(response.data.pedidos);  
25              })  
26              .catch(() => {  
27                  setStatus({  
28                      type: 'error',  
29                      message: 'Erro: Sem conexão com a API.'  
30                  })  
31              })  
32      getServico();  
33  }, [id]);  
34  
35
```



# Criar os botões das ações

## 11. Exibir o alerta de erro.

```
36     return (
37       <div>
38         <Container>
39           <div className="d-flex">
40             <div className="mr-auto p-2">
41               <h1>Quantidade de pedidos do serviço</h1>
42             </div>
43             <div className="p-2">
44               <Link to="/visualizar-servico" className="btn btn-outline-success
45                 btn-sm">Listar</Link>
46             </div>
47           </div>
48           <hr className="m-1"/>
49           {status.type === 'error' ? <Alert color="danger">{status.message}</Alert> : ""}
50
51           <dl className="row">
52             <dt className="col-sm-3">Valor total</dt>
53             <dd className="col-sm-9">{data}</dd>
54           </dl>
55         </Container>
56
57       </div>
58     );

```



## Atualizar o repositório

Até aqui tudo bem?



## O que aprendemos...

Alterar layout da página visualizar serviço

Conectar-se à API

Utilizar o método GET para retornar dados

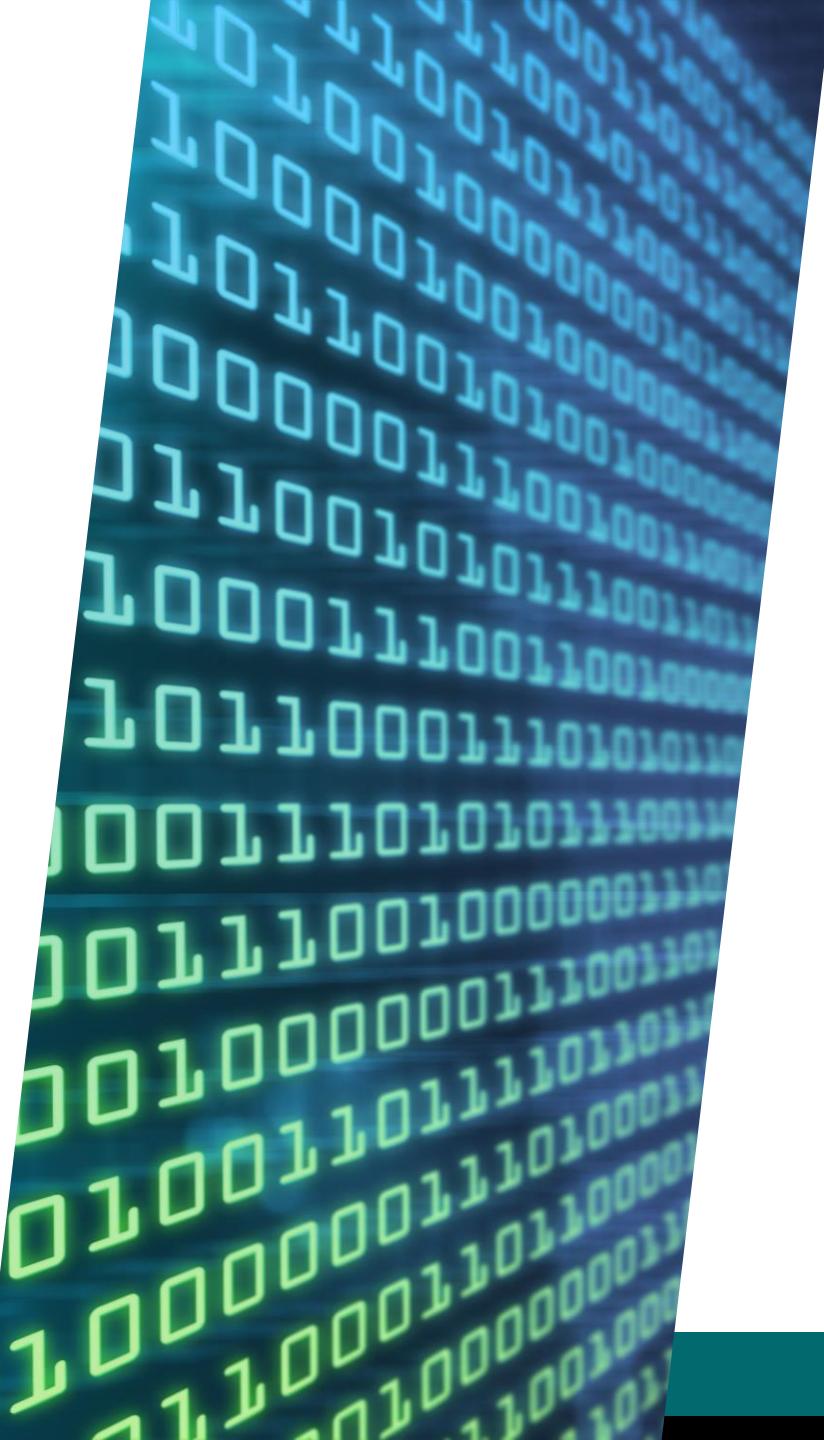
Mostrar o erro na página

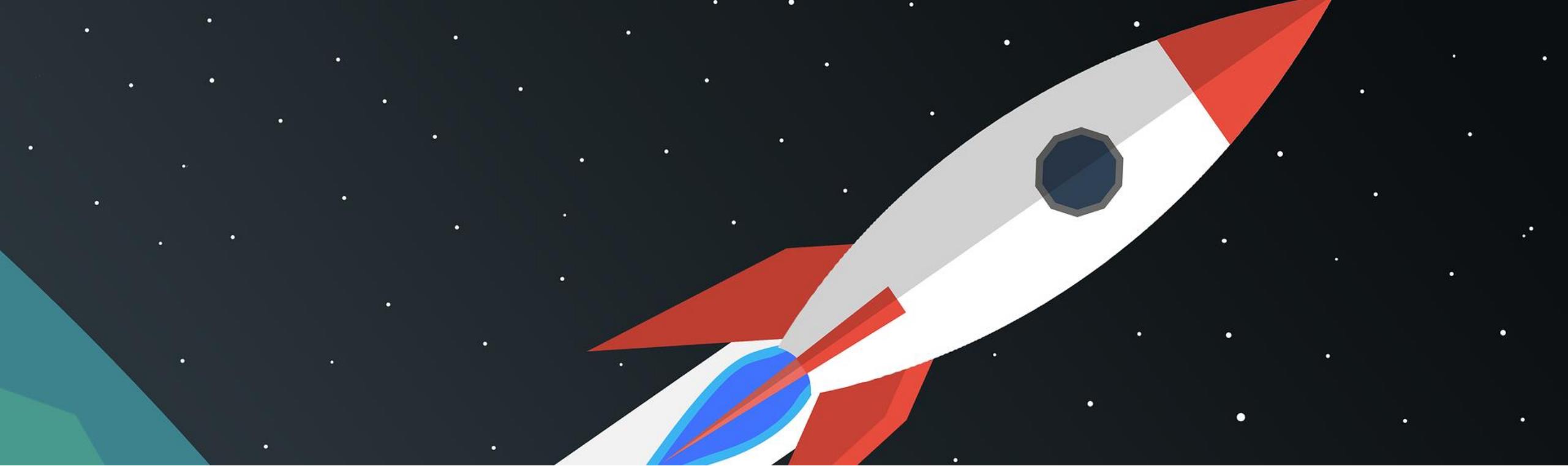
Criar os botões das ações



## O que vem em seguida

No próxima aula vamos fazer o cadastro do CRUD da nossa aplicação funcionar via web.





---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Front-end com React

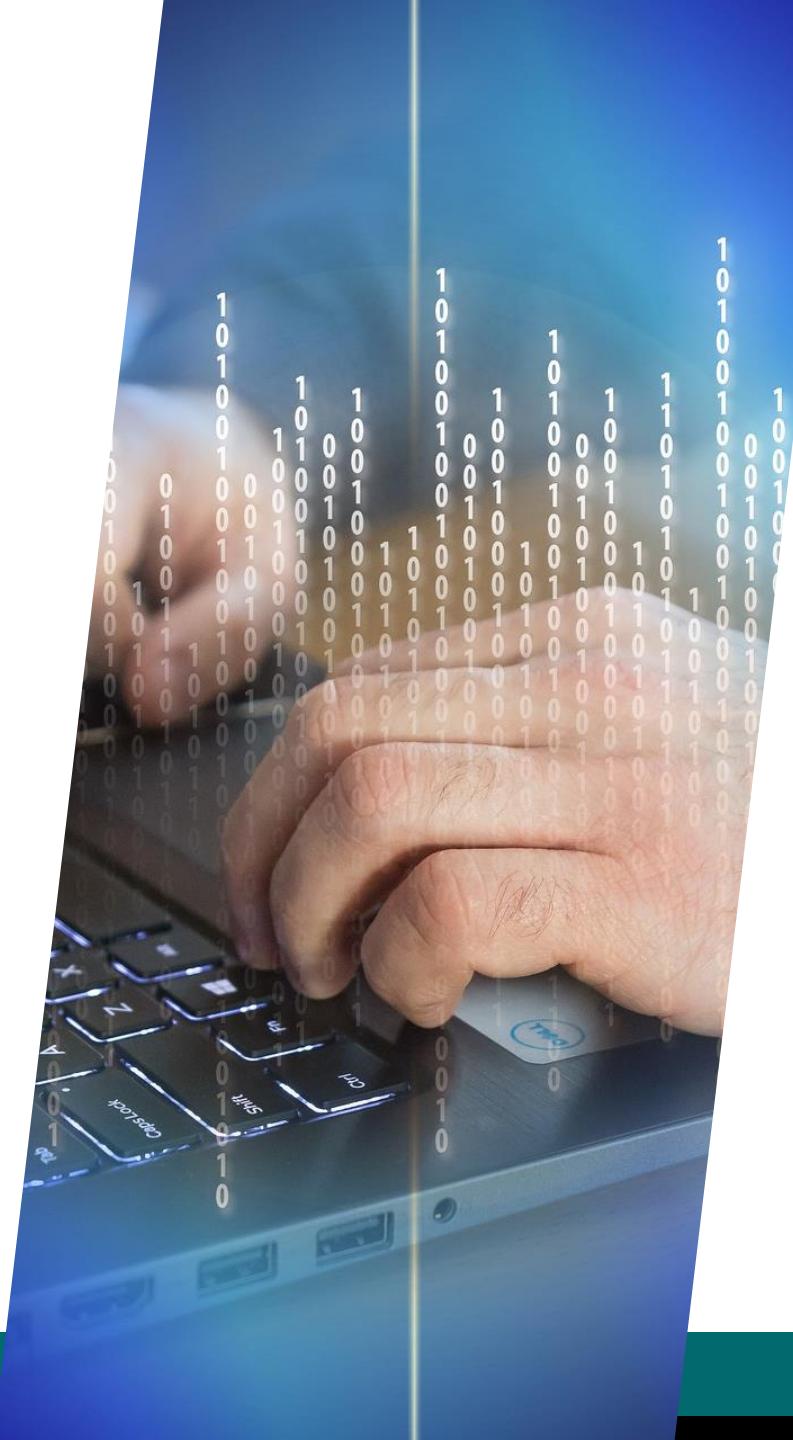
DIA 3 | Profº. Erinaldo

## Dia 3: Implementar os cadastros

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



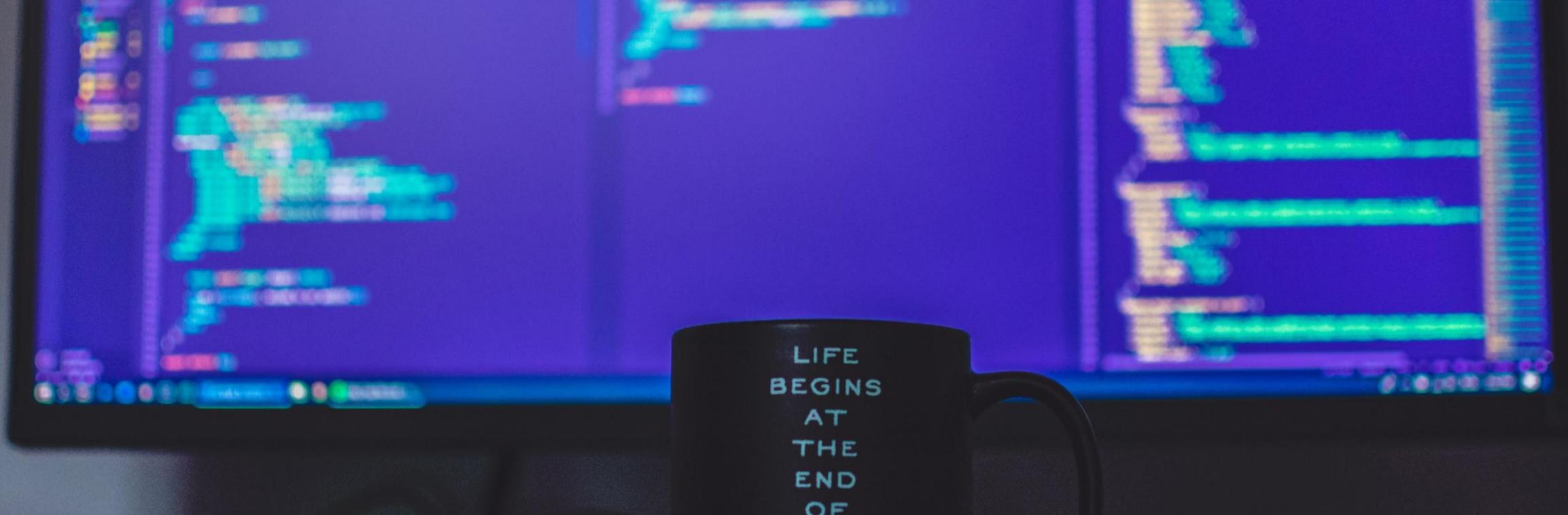
## Antes de começar...

Nas últimas aulas iniciamos uma aplicação front-end com ReactJS e implementamos um componente Menu utilizando o framework Reactstrap.

Na sequência acessamos a API no back-end e o banco de dados para consultar a partir do nosso front-end.

Nesta aula vamos implementar a funcionalidade de cadastrar novos objetos a partir do front-end.





@tiacademybrasil

# Implementar os Cadastros

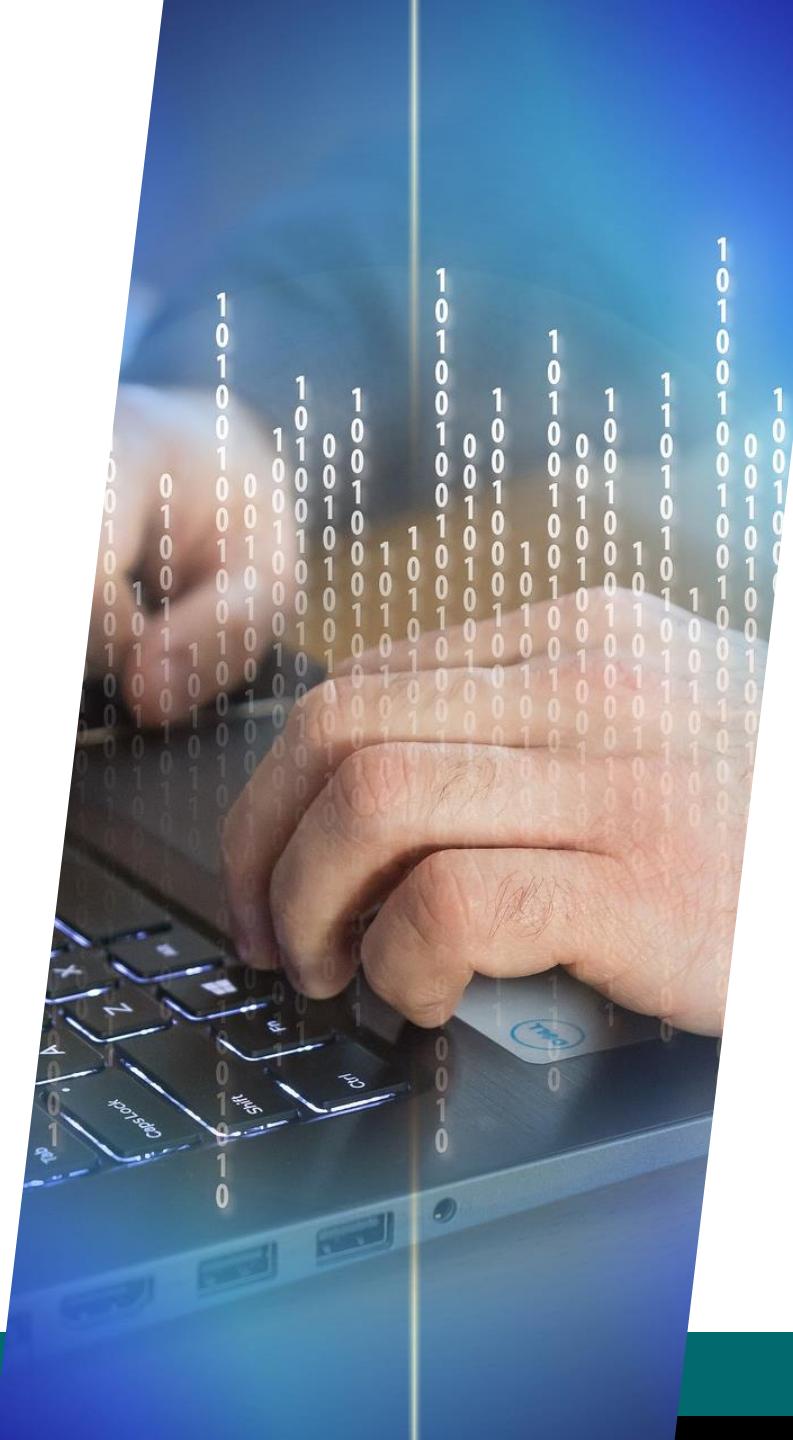
Aula 7 | Vamos praticar?

## Dia 3: Implementar os cadastros

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

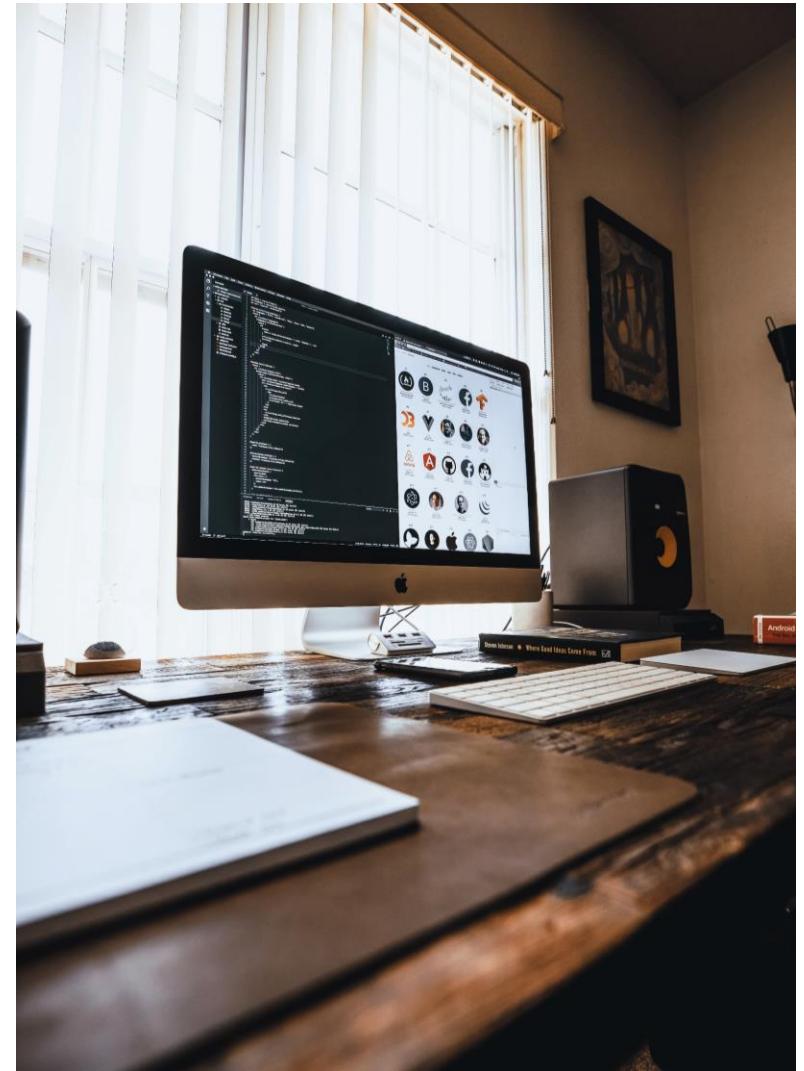
**4<sup>a</sup> semana**



## Criar o botão cadastrar

1. Certifique-se de estar conectado ao servidor de banco de dados.
2. Certifique-se de estar com o back-end executando.
3. Certifique-se de estar com o Postman ativado.
4. Abra o index.js do VisualizarServico e crie um novo botão para cadastrar novos serviços.

```
38     return (
39       <div>
40         <Container>
41           <div className="d-flex">
42             <div className="mr-auto p-3">
43               <h1>Visualizar informações do serviço</h1>
44             </div>
45             <div className="p-3">
46               <a href="/cadastrar-servico" className="btn btn-outline-success
47                 btn-sm">Cadastrar</a>
48             </div>
49           </div>
```



## Criar o layout da página cadastrar serviço

1. Certifique-se de estar conectado ao servidor de banco de dados.
2. Certifique-se de estar com o back-end executando.
3. Certifique-se de estar com o Postman ativado.
4. No diretório Servico do seu front-end crie uma pasta chamada Cadastrar.
5. Na pasta Cadastrar crie um novo arquivo chamado index.js.
6. Copie um modelo básico para o arquivo index.js.



# Criar o layout da página cadastrar serviço

src > pages > Servico > Cadastrar > **JS** index.js > ...

```
1  export const Cadastrar = () => {
2      return (
3          <div>
4              <h1>Cadastrar serviço</h1>
5          </div>
6      );
7  };
```

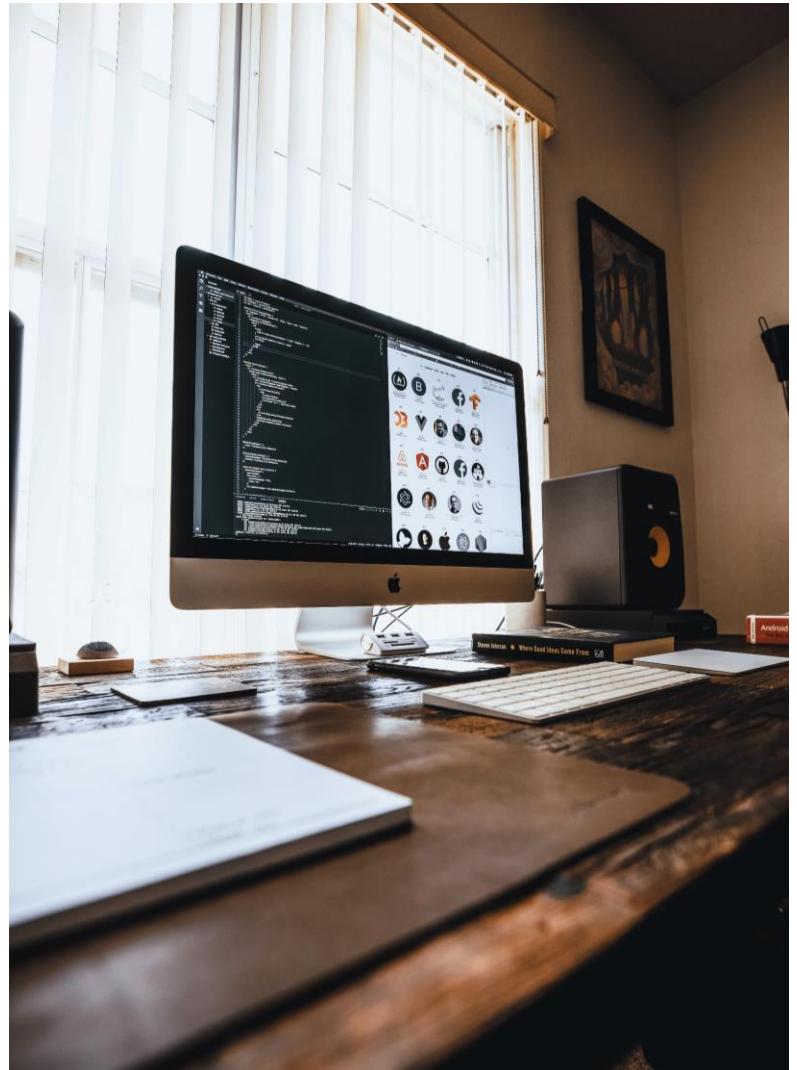
## 7. Insira uma nova rota no arquivo App.js.

```
12  function App() {
13      return (
14          <div>
15              <Menu />
16              <Router>
17                  <Switch>
18                      <Route exact path="/" component={Home}/>
19                      <Route path="/visualizar-cliente" component={VisualizarCliente}/>
20                      <Route path="/visualizar-servico" component={VisualizarServico}/>
21                      <Route path="/visualizar-pedido" component={VisualizarPedido}/>
22                      <Route path="/servico/:id" component={Serviço}/>
23                      <Route path="/cadastrar-servico" component={Cadastrar}/>
24                  </Switch>
25              </Router>
26          </div>
27      );
28  }
```



## Criar o layout da página cadastrar serviço

8. No arquivo index.js do componente Cadastrar no diretório Servico, faça a importação do Container e Alert do reactstrap.
9. Insira um container e inclua a div flex.
10. Importe o Link do react-router-dom.
11. Insira uma linha com a tag hr.



# Criar o layout da página cadastrar serviço

```
src > pages > Servico > Cadastrar > js index.js > ...
1 import { Container, Alert } from 'reactstrap';
2 import { Link } from 'react-router-dom';
3
4 export const Cadastrar = () => {
5   return (
6     <Container>
7       <div className="d-flex">
8         <div className="mr-auto p-2">
9           <h1>Cadastrar Serviço</h1>
10        </div>
11        <div className="p-2">
12          <Link to="/visualizar-servico" className="btn btn-outline-success
13            btn-sm">Listar</Link>
14        </div>
15      </div>
16
17      <hr className="m-1"/>
18    </Container>
19  );
20};
```

Agora vamos criar o formulário de cadastro



## Criar o formulário de cadastro de serviço

1. Importo o componente Form do reactstrap.
2. Abra a documentação do Form no reactstrap pelo link <https://reactstrap.github.io/components/form>.
3. Crie dois FormGroup com Label e Input para preencher os campos nome e descrição do serviço.
4. Crie um botão utilizando o componente Button do reactstrap.
5. Faça com que o usuário clique no botão o formulário possa ser submetido



# Criar o formulário de cadastro de serviço

```
20      <Form className="p-2" onSubmit={cadServico}>
21        <FormGroup className="p-2">
22          <Label>Nome</Label>
23          <Input type="text" name="nome" placeholder="Nome do serviço" />
24        </FormGroup>
25
26        <FormGroup className="p-2">
27          <Label>Descrição</Label>
28          <Input type="text" name="descricao" placeholder="Descrição do serviço" />
29        </FormGroup>
30
31          <Button type="submit" outline color="success">Cadastrar</Button>
32        </Form>
33      </Container>
34    </div>
35  );
36};
```

Agora vamos criar a  
função **cadServico**

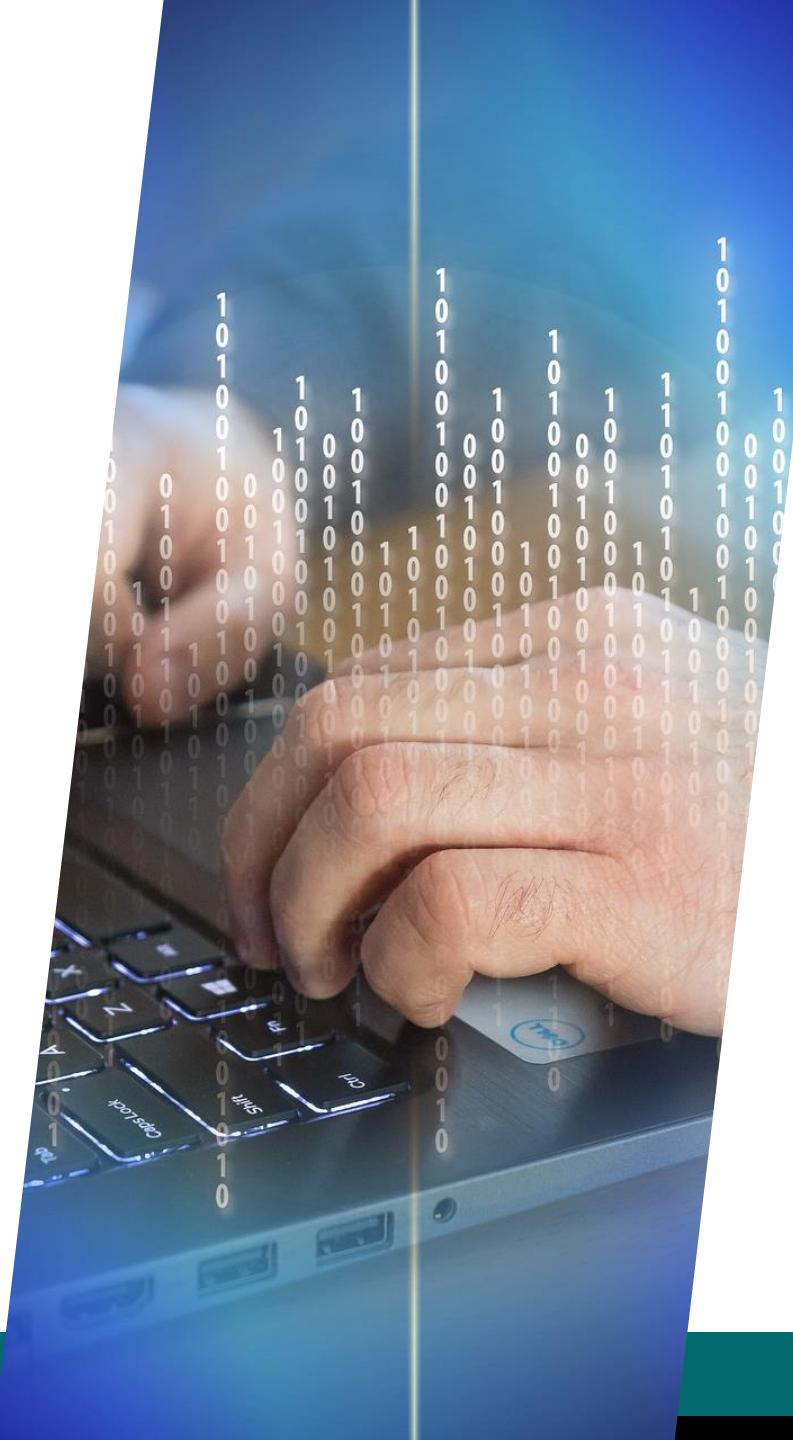


## Dia 3: Implementar os cadastros

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

**4<sup>a</sup> semana**

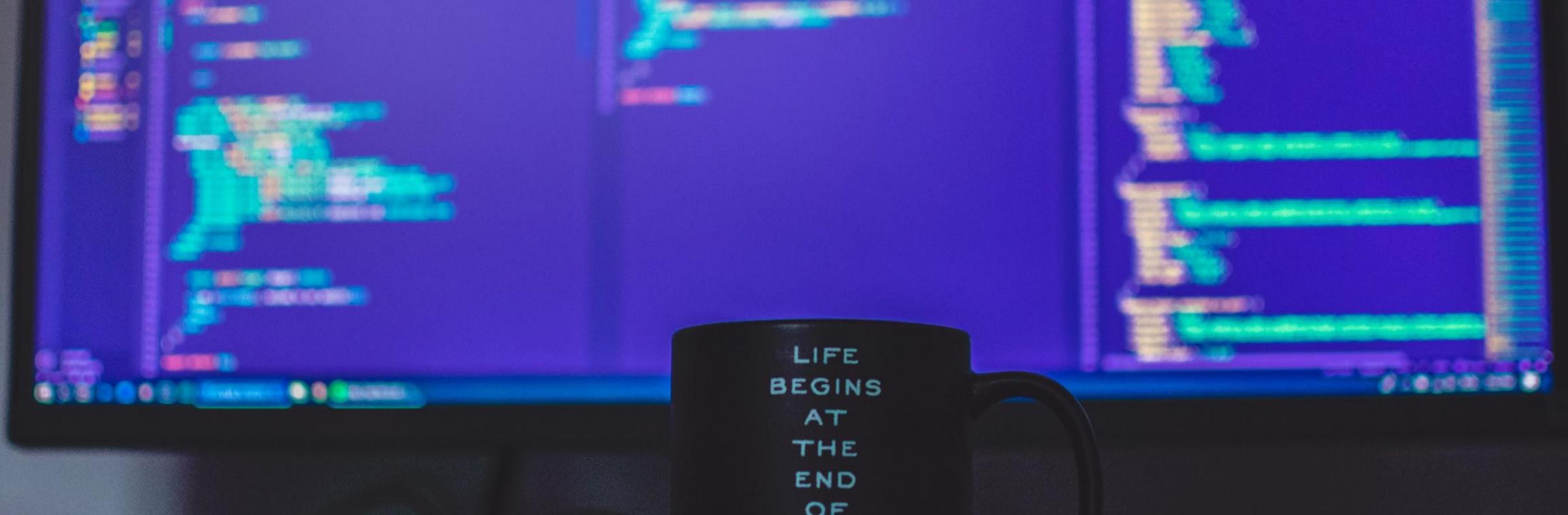




@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Implementar os Cadastros

Aula 8 | Vamos praticar?

## Dia 3: Implementar os cadastros

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



## Criar a função para cadastrar

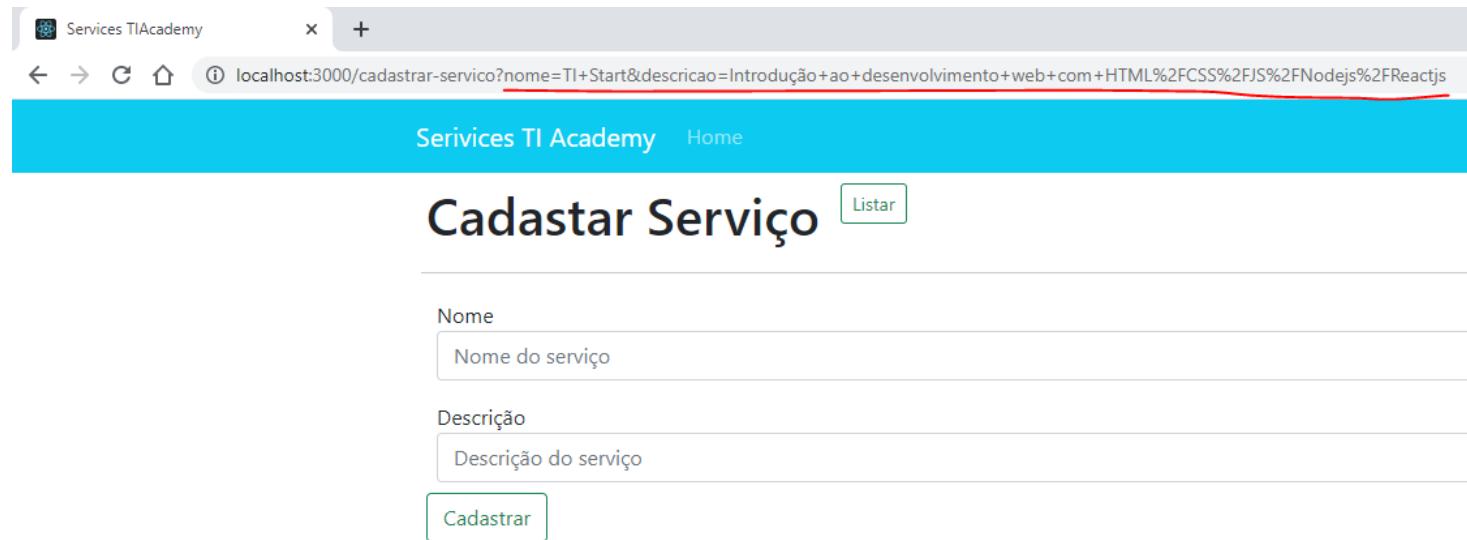
1. Para testar, crie a função cadServiço como demonstrada a seguir.

```
4  export const Cadastrar = () => {  
5  
6      const cadServiço = async => {  
7          console.log("Cadastrar");  
8      }  
9  
10     return (  
11         <div>  
12             <Container>|
```

Note que ao executar as informações passadas no formulário ficam aparecendo na barra de endereço e que a página é recarregada.

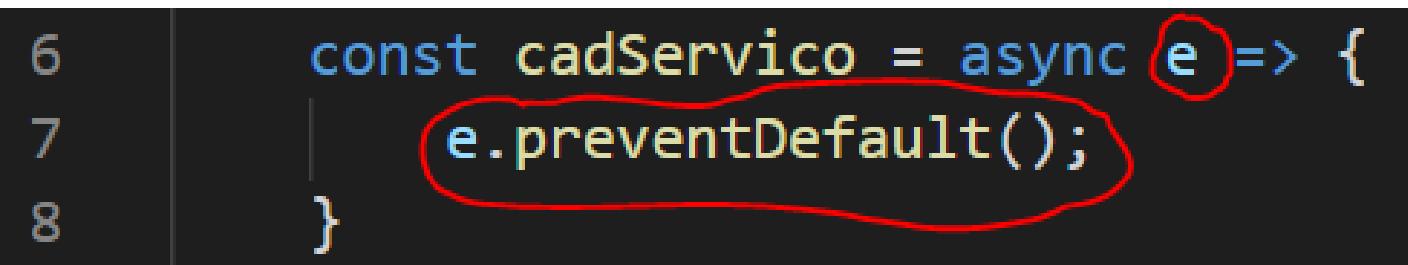


# Criar a função para cadastrar



A screenshot of a web browser window titled "Services TI Academy". The URL in the address bar is "localhost:3000/cadastrar-servico?nome=TI+Start&descricao=Introdução+ao+desenvolvimento+web+com+HTML%2FCSS%2FJS%2FNodejs%2FReactjs". The page has a blue header with the text "Services TI Academy" and "Home". Below the header, there is a title "Cadastrar Serviço" and a "Listar" button. The form contains two input fields: "Nome" (with placeholder "Nome do serviço") and "Descrição" (with placeholder "Descrição do serviço"). At the bottom is a green "Cadastrar" button.

2. Altere a função preventDefault ao evento para corrigir.



A screenshot of a code editor showing a snippet of JavaScript code. The code defines a function "cadServico" as an asynchronous function. The body of the function contains a line of code: "e.preventDefault();". The word "e" is circled in red, and the entire line "e.preventDefault();" is also circled in red.

```
6 const cadServico = async e => {
7   e.preventDefault();
8 }
```



## Criar a função para cadastrar

3. Crie um objeto com os campos do serviço.
4. Crie uma constante `valorInput` para receber os valores do formulário.
5. Altere o formulário para passar os dados caso os campos sejam alterados.
6. Altere a função `cadServico` para mostrar os dados do serviço.

```
5 export const Cadastrar = () => {
6
7     const [servico, setServico] = useState({
8         nome: '',
9         descricao: ''
10    })
11
12    const valorInput = e => setServico({...servico, [e.target.name]:e.target.value})
13
14    const cadServico = async e => {
15        e.preventDefault();
16        console.log(servico)
17    }
}
```



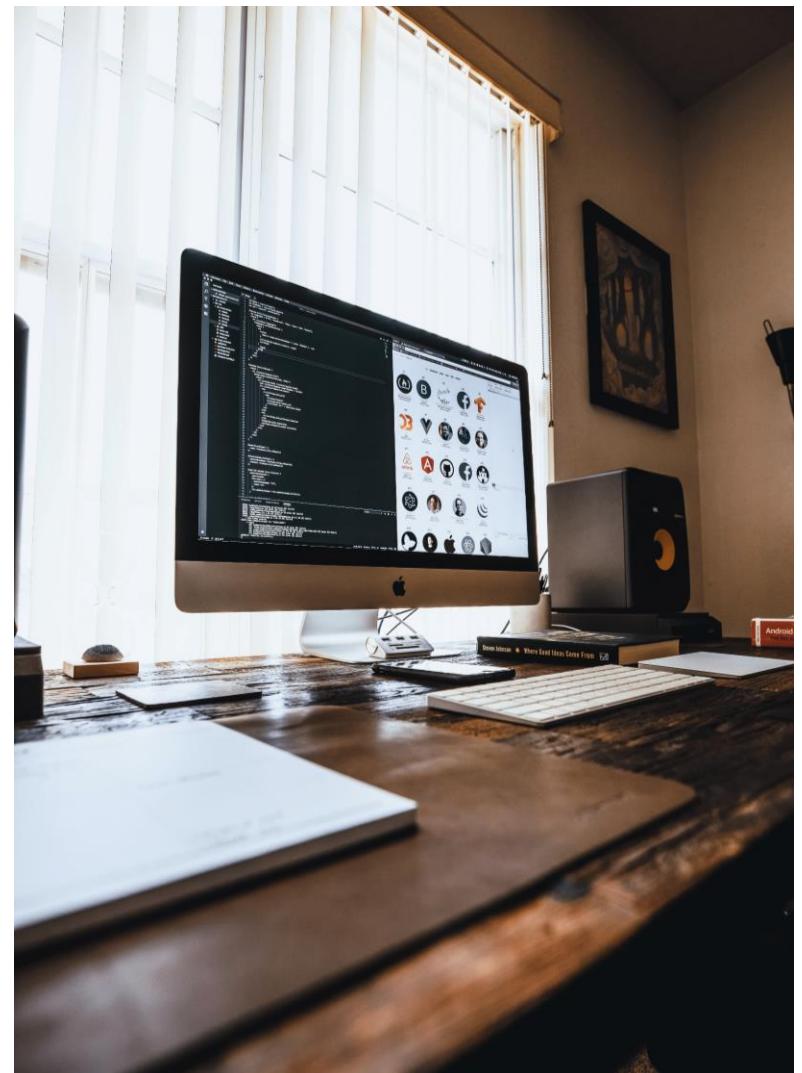
# Criar a função para cadastrar

```
<Form className="p-2" onSubmit={cadServico}>
  <FormGroup className="p-2">
    <Label>Nome</Label>
    <Input type="text" name="nome"
           placeholder="Nome do serviço" onChange={valorInput} />
  </FormGroup>

  <FormGroup className="p-2">
    <Label>Descrição</Label>
    <Input type="text" name="descricao"
           placeholder="Descrição do serviço" onChange={valorInput}/>
  </FormGroup>

  <Button type="submit" outline color="success">Cadastrar</Button>
</Form>
```

Teste o resultado das alterações no **Console**.



# Criar a função para cadastrar

The screenshot shows a web browser window with the URL `localhost:3000/cadastrar-servico`. The page title is "Services TI Academy". The main content is a form titled "Cadastrar Serviço" with fields for "Nome" (containing "TI Start") and "Descrição" (containing "Introdução ao desenvolvimento web com HTML/CSS/JS/Nodejs/Reactjs"). A "Cadastrar" button is at the bottom. To the right, the browser's developer tools are open, specifically the "Console" tab. A red box highlights the "TI Start" object in the console output, which includes properties like "name" and "descricao".

7. Para enviar os dados para a API altere a função `cadServico`.

Ao usar o `axios`, para passar cabeçalhos personalizados, forneça um objeto contendo os cabeçalhos (`headers`) como último argumento.



# Criar a função para cadastrar

```
16  const cadServico = async e => {
17    e.preventDefault();
18    console.log(servico);
19
20    const headers = {
21      'Content-Type': 'application/json'
22    }
23
24    await axios.post(api+"/servicos",servico, {headers})
25    .then((response) => {
26      console.log(response);
27    })
28    .catch(()=>{
29      console.log("Erro: Sem conexão com a API.");
30    })
31 }
```



# Criar a função para cadastrar

Services TI Academy Home

## Cadastrar Serviço

Nome  
TI Start

Descrição  
Introdução ao desenvolvimento web com HTML/CSS/JS/Nodejs/Reactjs

Cadastrar

[HMR] Waiting for update signal from WDS...  
[name: 'TI Start', descricao: 'Introdução ao desenvolvimento web com HTML/CSS/JS/Nodejs/Reactjs']  
[name: 'TI Start'  
[[Prototype]]: Object  
index.js:26  
[data: {}, status: 200, statusText: 'OK', headers: {}, config: {}]  
[config: {url: 'http://localhost:3001/servicos', method: 'post', data: {'nome':'TI Start','descricao':'Introdução ao desenvolvimento web com HTML/CSS/JS/Nodejs/Reactjs'}, headers: {content-length: '56', content-type: 'application/json; charset=utf-8'}, request: XMLHttpRequest (readyState: 4, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, onread...  
status: 200  
statusText: 'OK'  
[[Prototype]]: Object

Serivices TI Academy Home

## Visualizar informações do serviço

Cadastrar

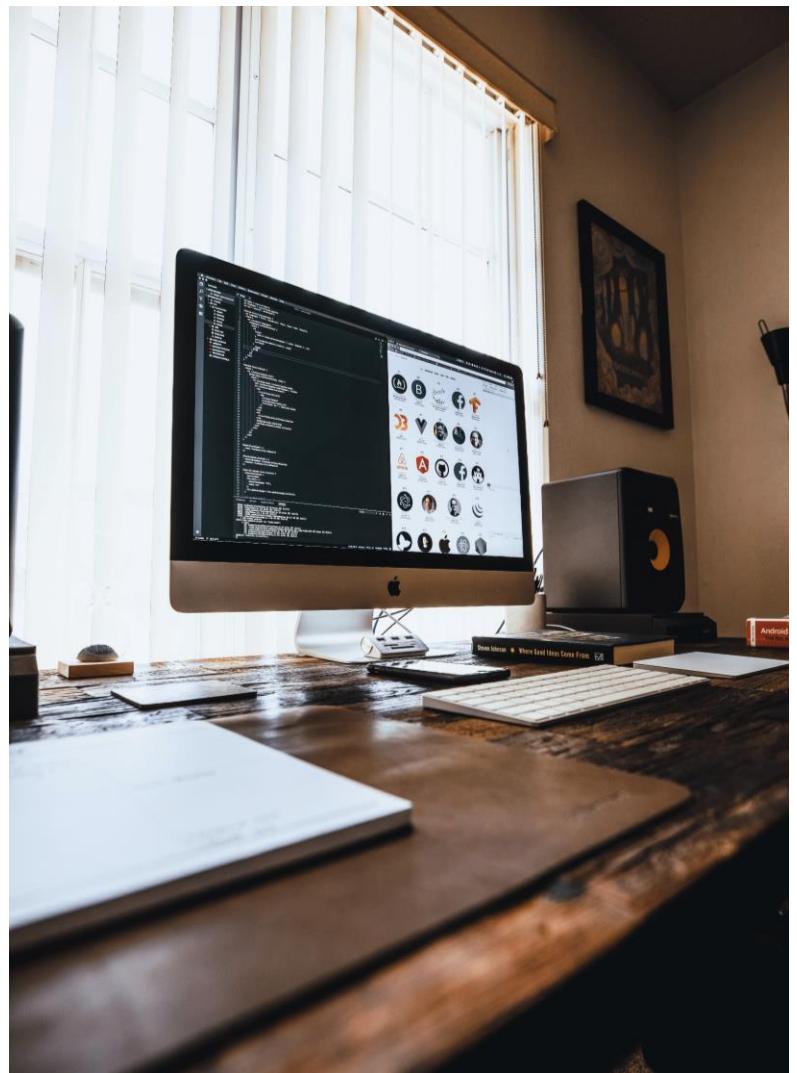
ID	Nome	Descrição	Ações
7	TI Start	Introdução ao desenvolvimento web com HTML/CSS/JS/Nodejs/Reactjs	<a href="#">Consultar</a>
5	Rede de computadores	Desenvolve projeto de redes locais e implementa a solução	<a href="#">Consultar</a>
2	Nodejs	Desenvolvimento de aplicação back-end	<a href="#">Consultar</a>
6	Java SE	Programação orientada a objetos	<a href="#">Consultar</a>
4	Infraestrutura	Instalação, configuração e manutenção de hardware	<a href="#">Consultar</a>
1	HTML/CSS/JS	Páginas estáticas e dinâmicas estilizadas	<a href="#">Consultar</a>
3	Delphi	Manutenção e suporte a sistemas legados em Delphi	<a href="#">Consultar</a>



## Criar a função para cadastrar

8. Faça a alteração na função cadServico para exibir a mensagem de sucesso.

```
16      const cadServico = async e => {
17        e.preventDefault();
18        console.log(servico);
19
20        const headers = {
21          'Content-Type': 'application/json'
22        }
23
24        await axios.post(api+"/servicos",servico, {headers})
25        .then((response) => {
26          console.log(response.data.message);
27        })
28        .catch(()=>{
29          console.log("Erro: Sem conexão com a API.");
30        })
31    }
```



# Implementar o tratamento de erro

1. Crie um objeto para receber e mostrar o status do serviço implementado.

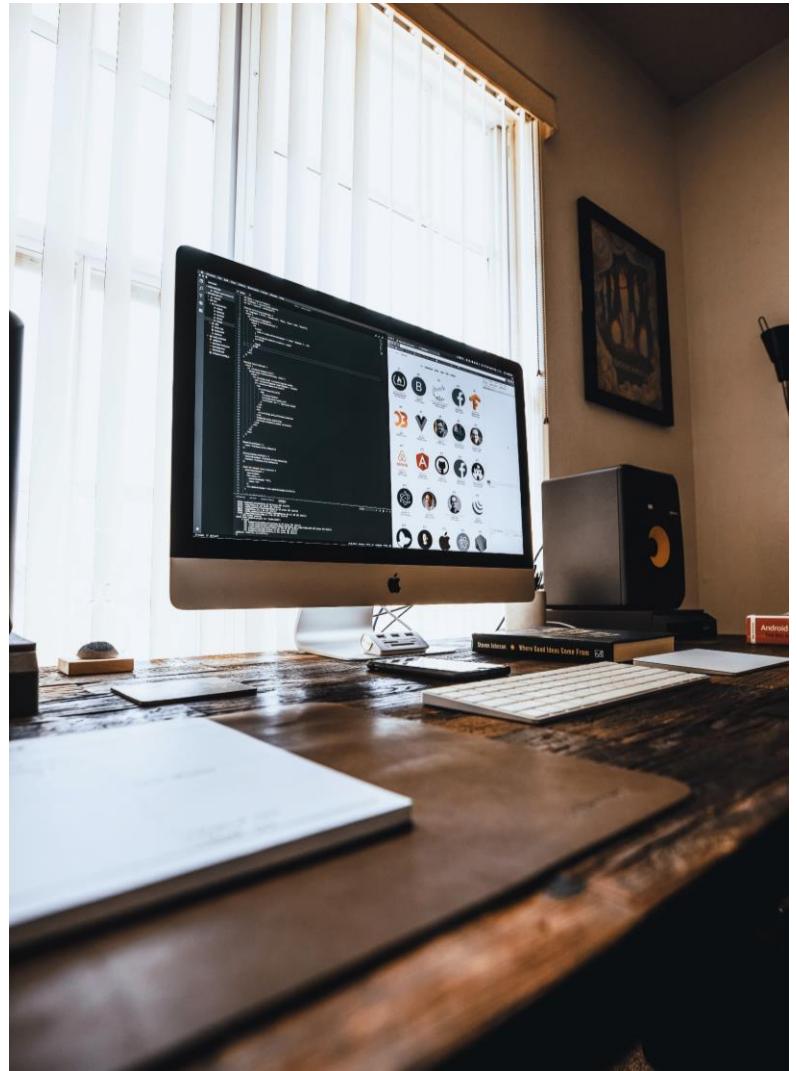
```
7  export const Cadastrar = () => {  
8  
9      const [servico, setServico] = useState({  
10         nome: '',  
11         descricao: ''  
12     });  
13  
14     const [status, setStatus] = useState({  
15         type: '',  
16         message: ''  
17     });  
18  
19     const cadastrar = async () => {  
20         try {  
21             const response = await fetch(`http://localhost:3001/servicos`, {  
22                 method: 'POST',  
23                 headers: {  
24                     'Content-Type': 'application/json'  
25                 },  
26                 body: JSON.stringify(servico)  
27             })  
28             const data = await response.json()  
29             if (response.ok) {  
30                 setServico(data)  
31                 setStatus({ type: 'success', message: 'Serviço cadastrado com sucesso!' })  
32             } else {  
33                 setStatus({ type: 'error', message: 'Erro ao cadastrar o serviço.' })  
34             }  
35         } catch (error) {  
36             setStatus({ type: 'error', message: 'Ocorreu um erro ao cadastrar o serviço.' })  
37         }  
38     }  
39  
40     return (  
41         <div>  
42             <h2>Cadastrar Serviço</h2>  
43             <form>  
44                 <input type="text" value={servico.nome} onChange={e => setServico({ ...servico, nome: e.target.value })} />  
45                 <input type="text" value={servico.descricao} onChange={e => setServico({ ...servico, descricao: e.target.value })} />  
46                 <button onClick={cadastrar}>Cadastrar</button>  
47             </form>  
48             <div>  
49                 {status.message}  
50             </div>  
51         </div>  
52     )  
53 }
```

2. Altere a função cadastrar para tratar o sucesso ou o erro.



# Implementar o tratamento de erro

```
29         await axios.post(api+"/servicos",servico, {headers})
30     .then((response) => {
31         //console.log(response.data.message);
32         if(response.data.error){
33             setStatus({
34                 type: 'error',
35                 message: response.data.message
36             });
37         }else{
38             setStatus({
39                 type: 'success',
40                 message: response.data.message
41             });
42         }
43     })
44     .catch(()=>{
45         setStatus({
46             type: 'error',
47             message: 'Erro: Sem conexão com a API.'
48         });
49     });
50 }
```



## Implementar o tratamento de erro

3. Implemente a alerta de erro para ser exibida no formulário.

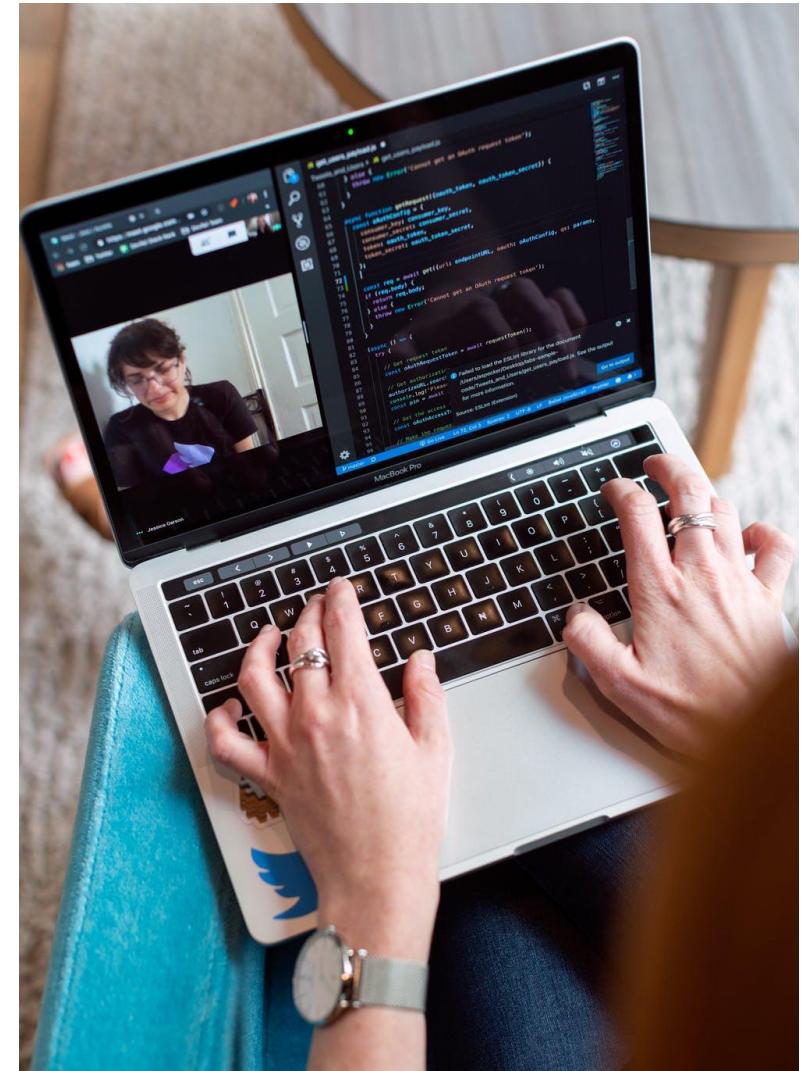
```
65      <hr className="m-1" />
66
67      {status.type === 'error' ? <Alert color="danger">{status.message}</Alert> : ""}
68
69      {status.type === 'success' ? <Alert color="success">{status.message}</Alert> : ""}
70
71      <Form className="p-2" onSubmit={cadServico}>
```

A partir desse instante não há mais necessidade de exibir os console.log.



## Exercícios

1. Implemente o botão Limpar no formulário.
2. Faça o cadastro de um novo cliente.
3. Faça o cadastro de um novo pedido.

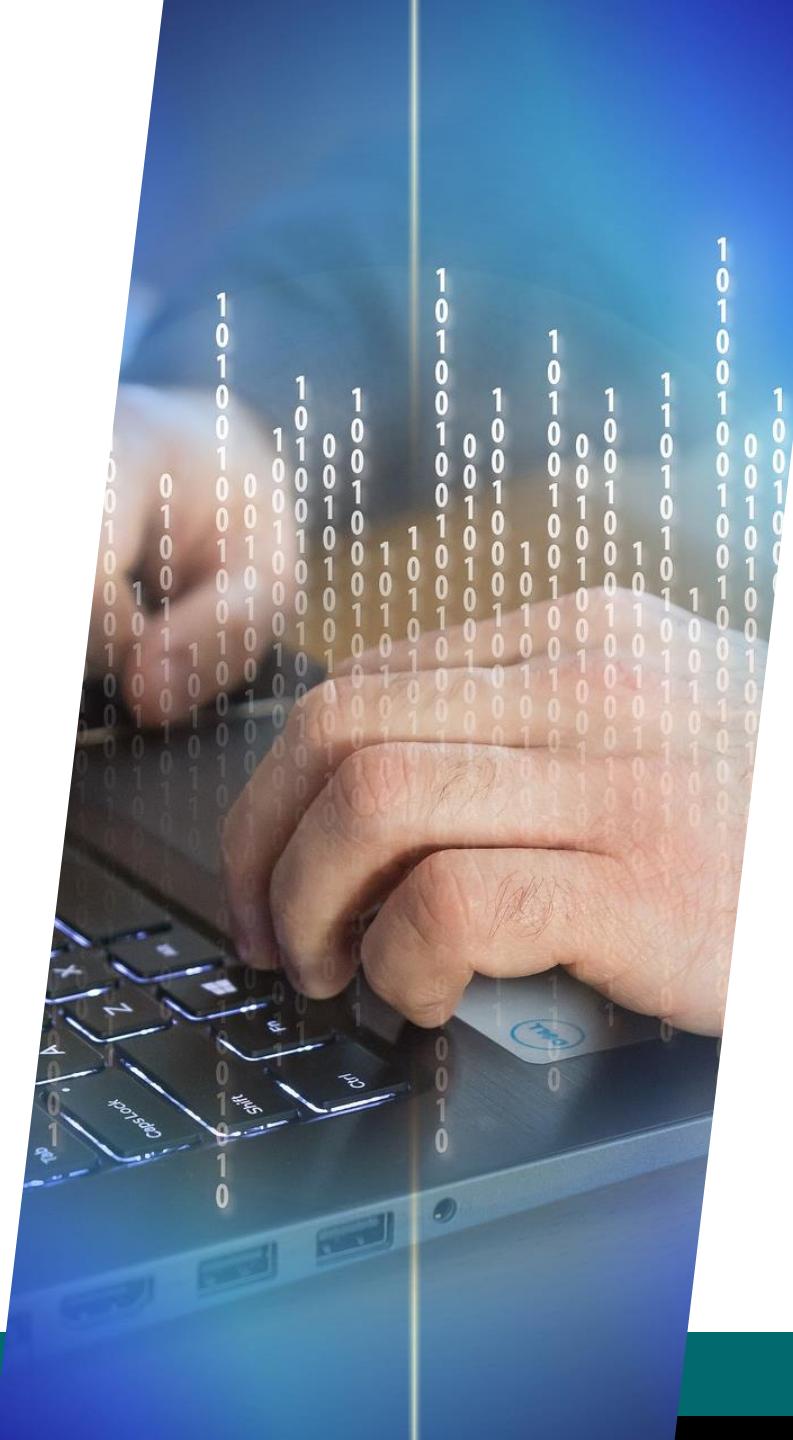


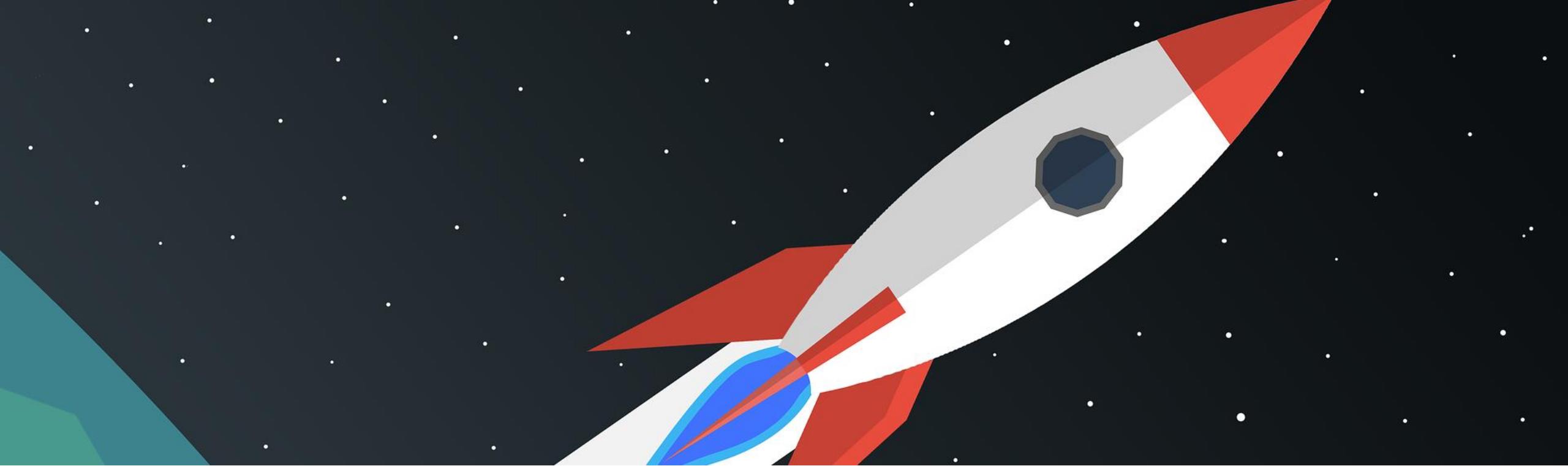
## O que aprendemos...

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

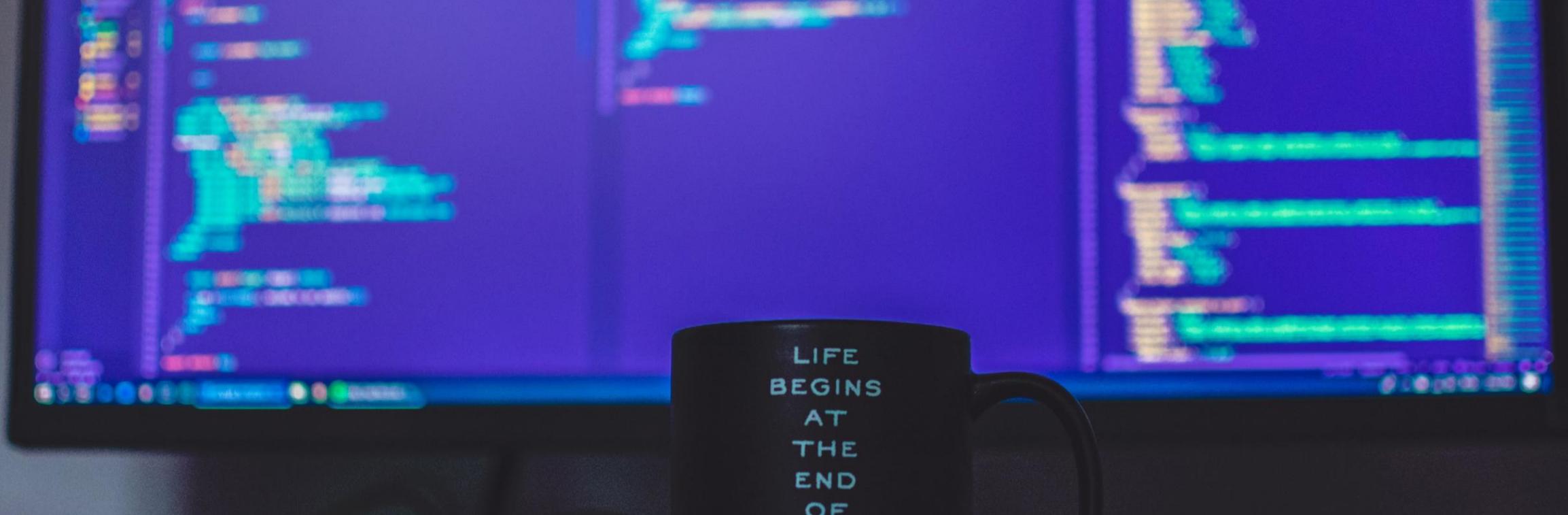
**4<sup>a</sup> semana**





---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil



# Implementar os Cadastros

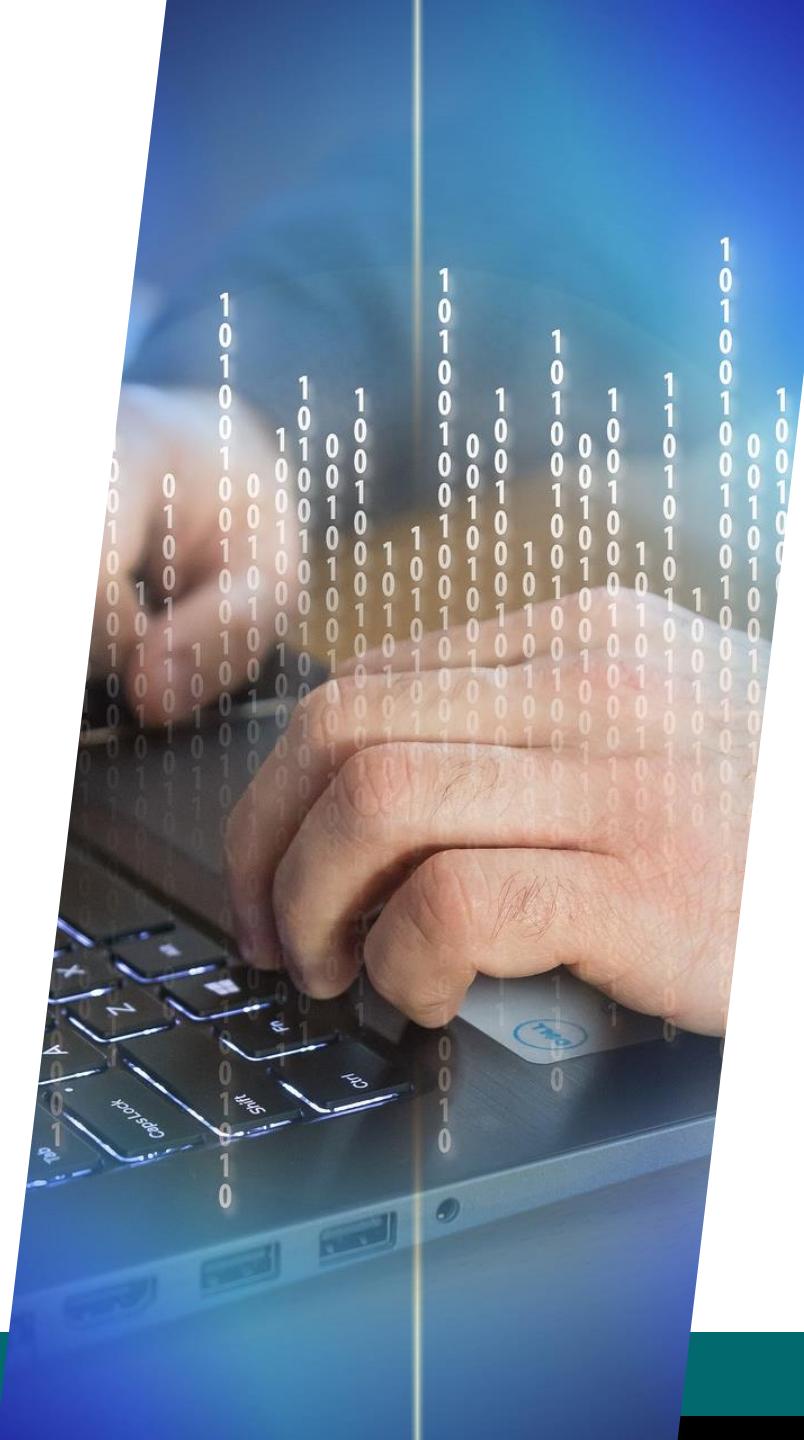
AULA 9 | Vamos praticar?

## Dia 3: Implementar os cadastros

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

**4<sup>a</sup> semana**

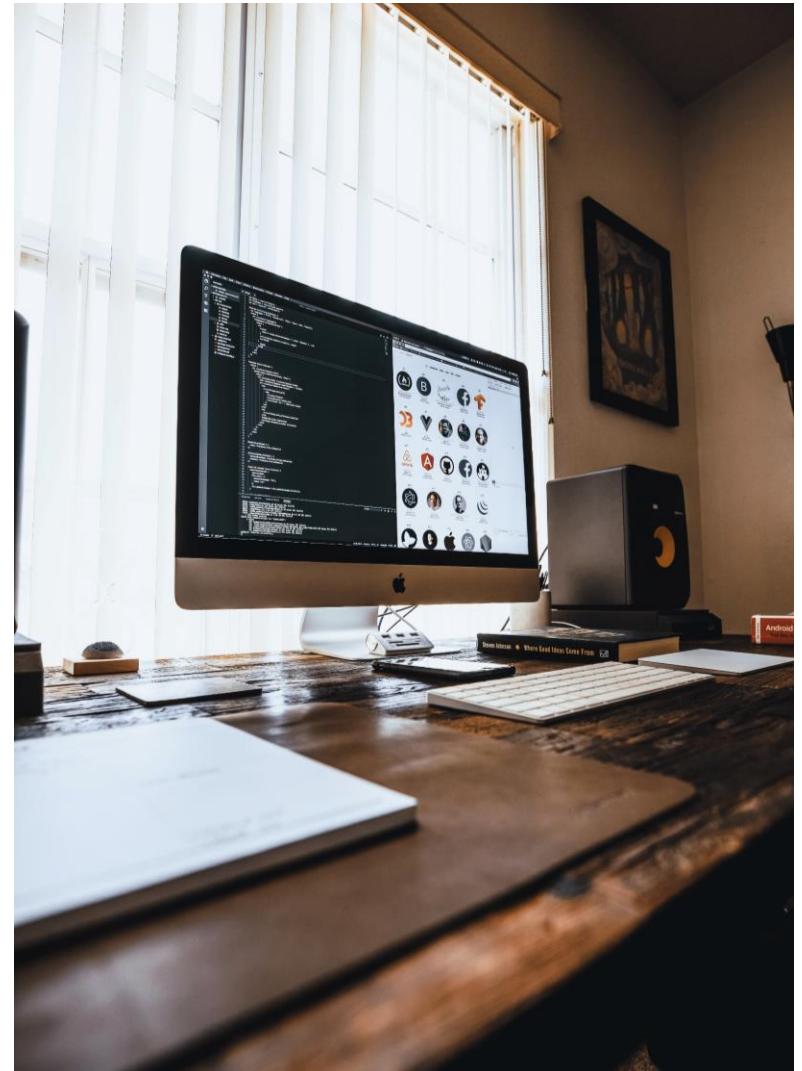


## Chorinho...

É possível que no tempo em que o usuário aguarda a mensagem de confirmação do cadastramento ele fique ansioso e pressione várias vezes o botão cadastrar, duplicando registros desnecessários do mesmo serviço.

Vamos implementar uma função que é capaz de contornar esse problema. Para isso é necessário:

- Pausar a execução no back-end.
- Bloquear para que o usuário não clique no botão Cadastrar enquanto aguarda.



## Pausar a execução no back-end

```
16 app.post('/servicos', async (req, res) => {
17   await aguardar(3000);
18
19   function aguardar(ms){
20     return new Promise((resolve) =>{
21       setTimeout(resolve, ms);
22     });
23   };
24
25   await servico.create(
26     req.body
27   ).then(function (){
28     return res.json({
29       error: false,
30       message: "Serviço criado com sucesso!"
31     })
32   }).catch(function (erro){
33     return res.status(400).json({
34       error: true,
35       message: "Erro no cadastro do serviço."
36     })
37   });
38 });


```

A Promise é facilmente identificada por códigos que utilizam o `.then` para *callback* de sucesso e `.catch` para erros.



# Chorinho...

Bloquear para que o usuário não clique nõo botão Cadastrar enquanto aguarda.

1. Alterar o objeto Status no front-end.

```
14     const [status, setStatus] = useState({  
15         formSave: false,  
16         type: '',  
17         message: ''  
18     });
```

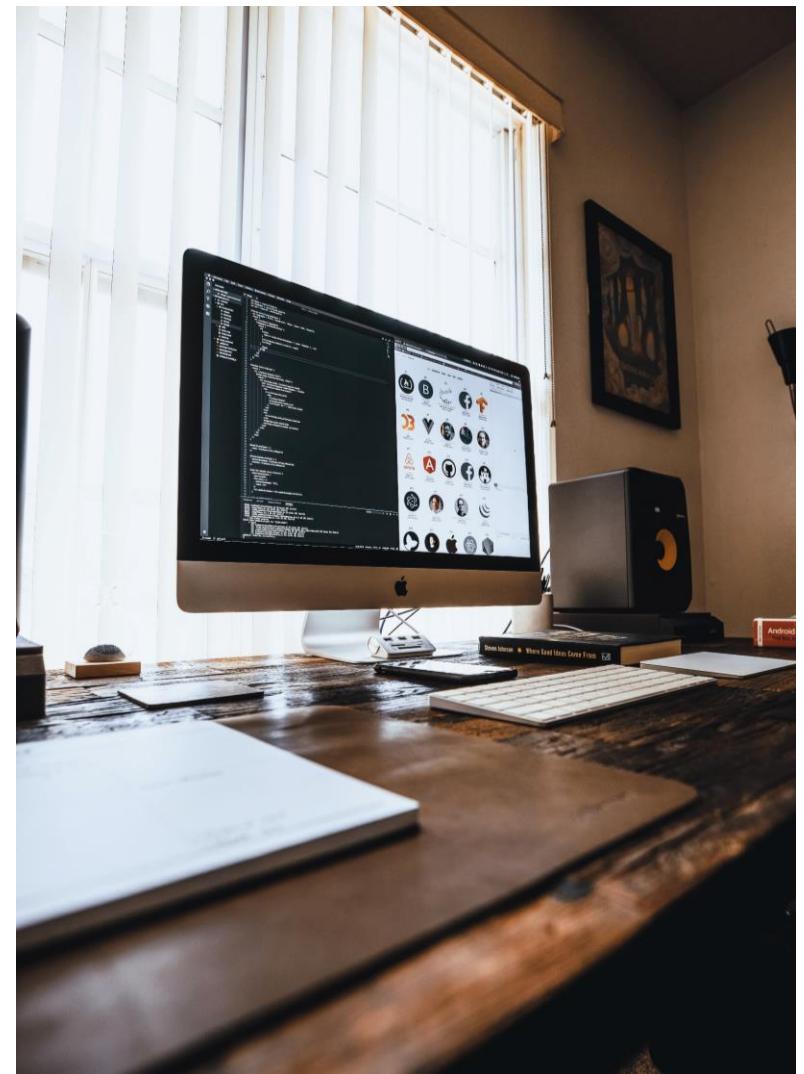
2. Altere o valor inicial formState dentro do cadServico.

```
22     const cadServico = async e => {  
23         e.preventDefault();  
24         //console.log(servico);  
25  
26         setStatus({  
27                 formSave:true  
28         });
```



### 3. Inclua na função cadServiço, no .then e .catch a seguinte alteração de status.

```
34      await axios.post(api+"/servicos",servico, {headers})
35  .then((response) => {
36      //console.log(response.data.message);
37      if(response.data.error){
38          setStatus({
39              formSave: false,
40              type: 'error',
41              message: response.data.message
42          });
43      }else{
44          setStatus({
45              formSave: false,
46              type: 'success',
47              message: response.data.message
48          });
49      }
50  })
51  .catch(()=>{
52      setStatus({
53          formSave: false,
54          type: 'error',
55          mensage: 'Erro: Sem conexão com a API.'
56      });
57  });
58 }
```



4. Altere o código do botão Cadastrar para validar.

```
92           {status.formSave ?  
93             <Button type="submit" outline color="success" disabled>Salvando...</Button> :  
94             <Button type="submit" outline color="success">Cadastrar</Button>}  
95
```

Teste e veja se está funcionando.

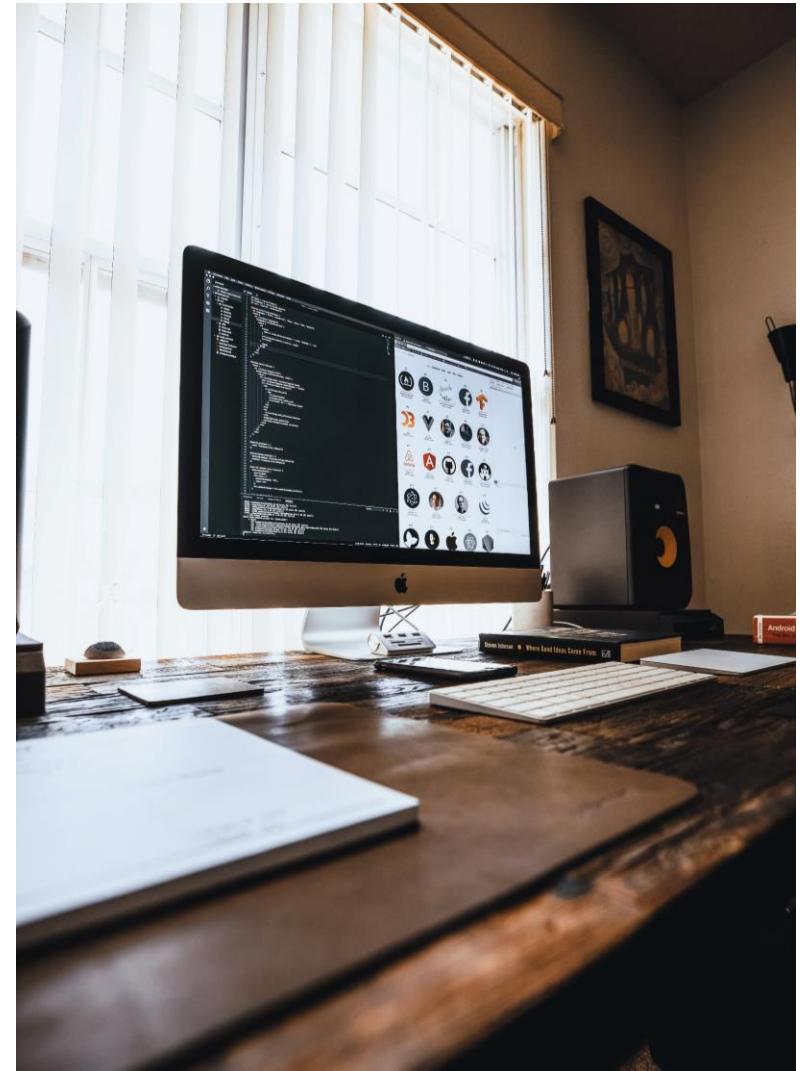
Agora... Vamos inserir um componente Spinners para deixar uma animação na espera.

Para maiores informações acesse o endereço <https://reactstrap.github.io/components/spinners/>.



## 5. Insira o componente Spinners no botão Cadastrar do formulário.

```
93     {status.formSave ?  
94         <Button type="submit" outline color="success" disabled>Salvando...  
95         |   <Spinner size="sm" color="success"/></Button> :  
96         <Button type="submit" outline color="success">Cadastrar</Button>}
```



## Atualizar o repositório

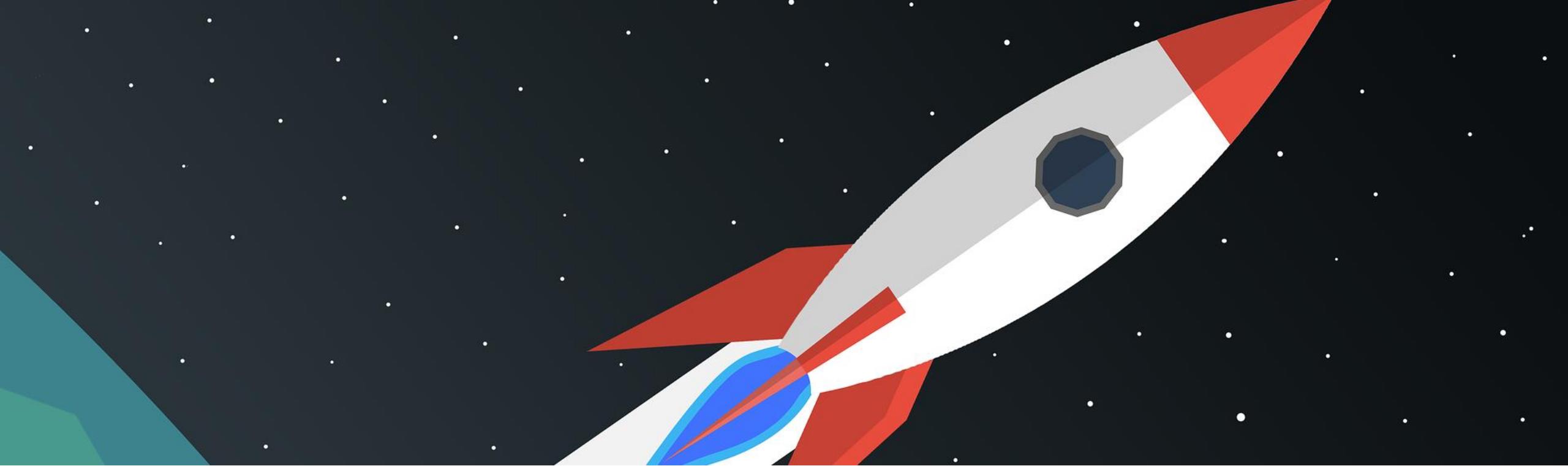
Até aqui tudo bem?



## O que vem em seguida

No próximo encontro vamos alterar a partir do nosso sistema web.





@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Front-end com React

DIA 4 | Profº. Erinaldo

## Dia 4: Implementar as alterações e exclusões

1. Criar o botão editar
2. Criar o componente editar
3. Implementar o formulário
4. Enviar para a API
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

**4<sup>a</sup> semana**



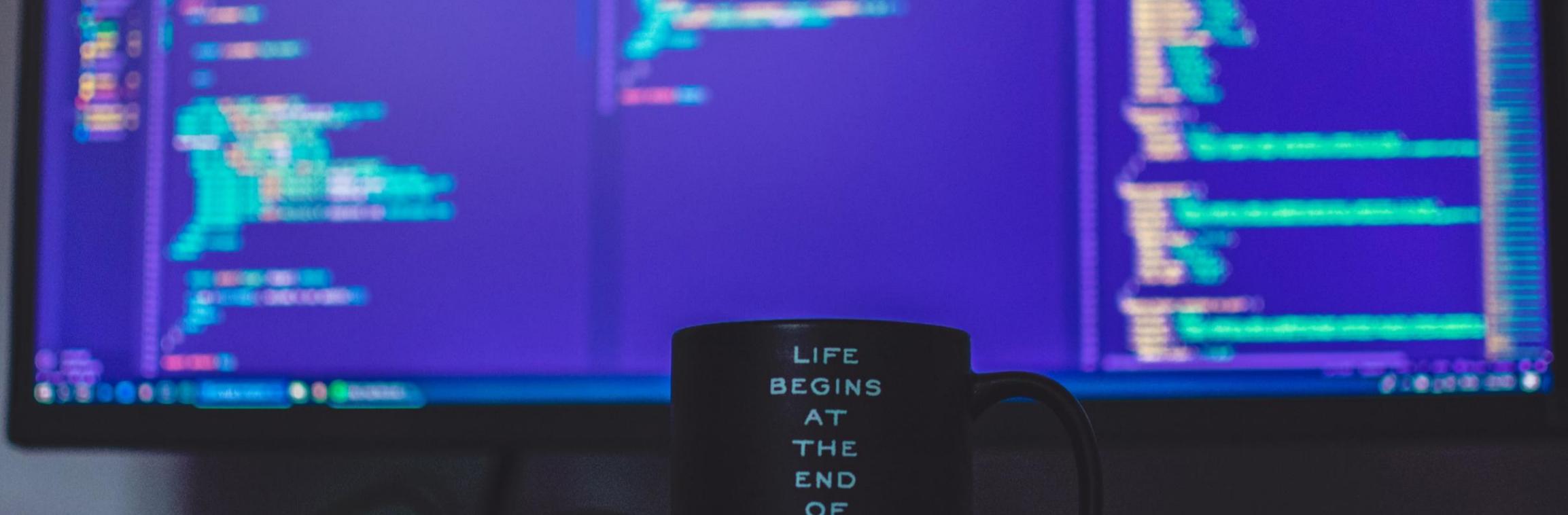
## Antes de começar...

Nas últimas aulas iniciamos uma aplicação front-end com ReactJS e implementamos um componente Menu utilizando o framework Reactstrap.

Na sequência acessamos a API no *back-end* e o banco de dados para consultar e implementar os cadastros a partir do *front-end*.

Na aula de hoje vamos implementar as funcionalidades para editar objetos a partir do *front-end*.





@tiacademybrasil

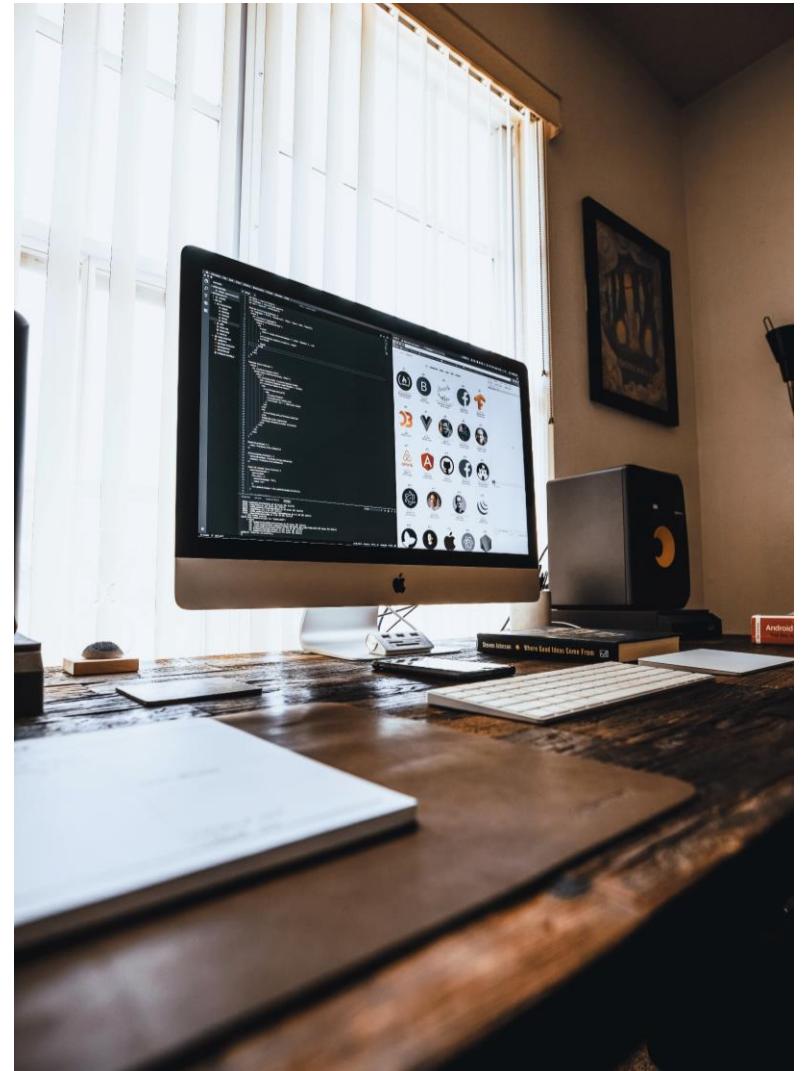
# Implementar as Alterações

Vamos praticar?

## Criar o botão editar

1. Certifique-se de estar conectado ao servidor de banco de dados.
2. Certifique-se de estar com o back-end executando.
3. Certifique-se de estar com o Postman ativado.
4. Abra o index.js do VisualizarServico e crie um novo botão para editar serviços.

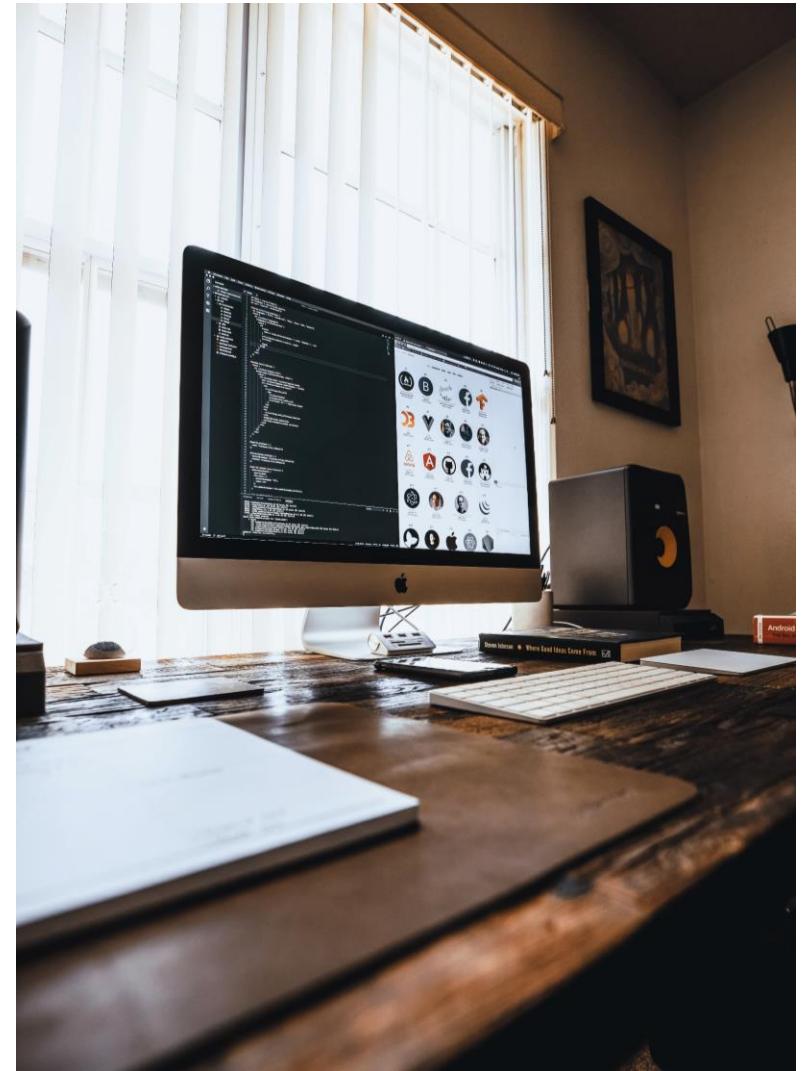
```
src > pages > Servico > VisualizarServico > JS index.js > VisualizarServico > data.map() callback
64      <tbody>
65        {data.map(item => (
66          <tr key={item.id}>
67            <td>{item.id}</td>
68            <td>{item.nome}</td>
69            <td>{item.descricao}</td>
70            <td className="text-center">
71              <Link to={"/servico/" + item.id}
72                className="btn btn-outline-primary btn-sm">Consultar</Link>
73              <Link to={"/editarservico/" + item.id}
74                className="btn btn-outline-warning btn-sm">Editar</Link>
75            </td>
76          </tr>
77        ))}
```



## Criar o botão editar

5. Insira um botão, também, no index.js do componente Servico.

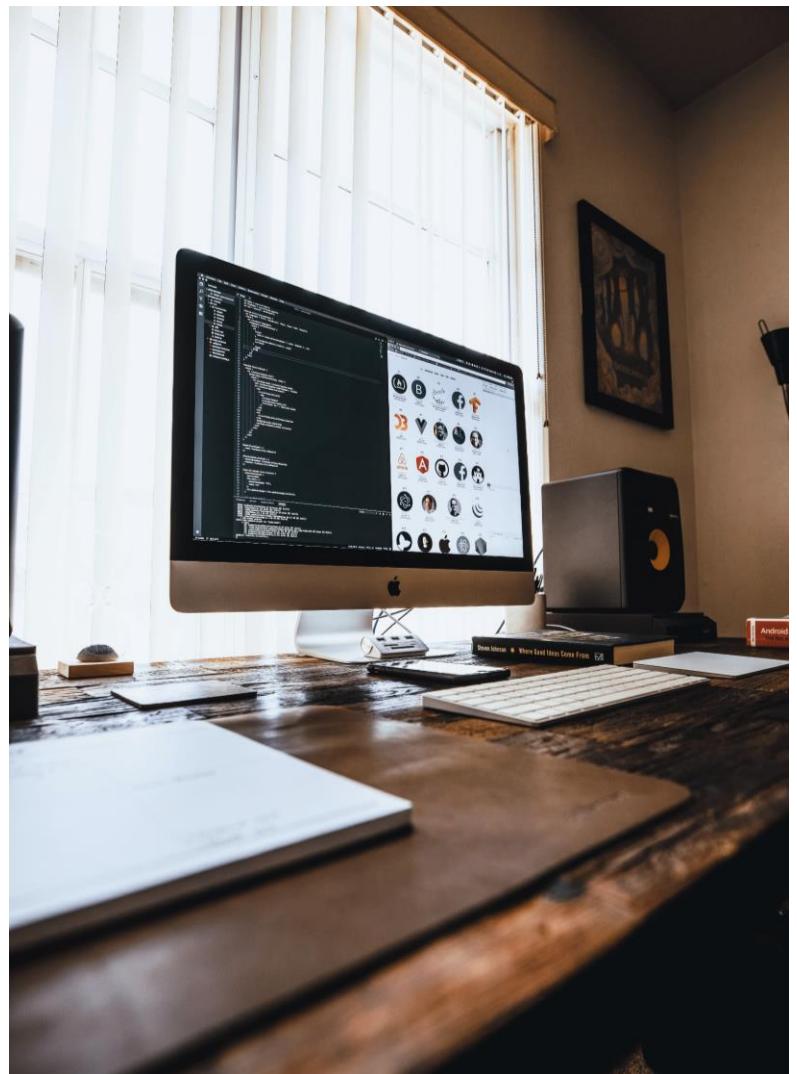
```
src > pages > Servico > Servico > js index.js > Servico  
28     return (  
29         <div>  
30             <Container>  
31                 <div className="d-flex">  
32                     <div className="mr-auto p-2">  
33                         <h1>Informações do Serviço</h1>  
34                     </div>  
35  
36                     <div className="p-2">  
37                         <Link to="/visualizarservico"  
38                             className="btn btn-outline-primary  
39                                 btn-sm mr-1">  
40                             Serviços  
41                         </Link>  
42                         <Link to={"/editarservico/" + data.id}  
43                             className="btn btn-outline-warning btn-sm">Editar</Link>  
44                     </div>  
45                 </div>  
46                 <dl className="row">  
47                     <dt className="col-sm-3">Nome</dt>  
48                     <dd className="col-sm-9">{data.nome}</dd>  
49                 </dl>
```



## Criar o componente editar

1. No diretório Servico do seu front-end crie uma pasta chamada Editar.
2. Na pasta Editar crie um novo arquivo chamado index.js.
3. Copie um modelo básico para o arquivo index.js.

```
src > pages > Servico > Editar > JS index.js > [Edit] Editar
1  import { Container } from "reactstrap"
2
3  export const Editar = () => {
4      return (
5          <div>
6              <Container>
7                  <div>
8                      <h1>Editar serviço</h1>
9                  </div>
10             </Container>
11         </div>
12     )
13 }
```



## Criar o componente editar

4. Altere o arquivo App.js para incluir o componente e a rota Editar.

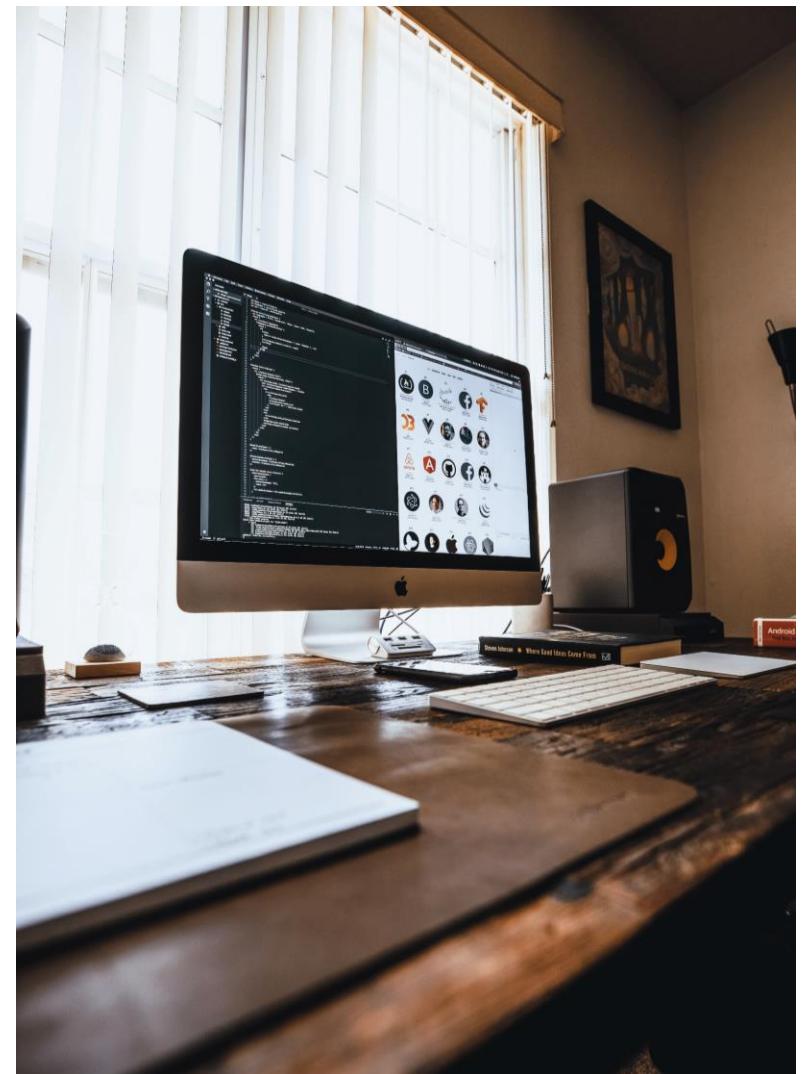
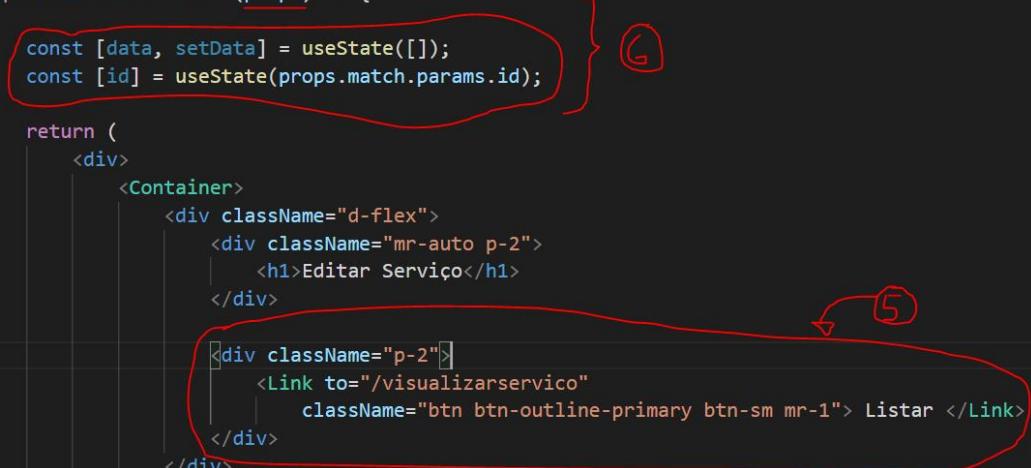
```
src > JS App.js > ...
8 | import { Cadastrar } from './pages/Servico/Cadastrar';
9 | import { Editar } from './pages/Servico/Editar';
10
11 function App() {
12   return (
13     <div>
14       <Menu/>
15       <Router>
16         <Switch>
17           <Route exact path="/" component={Home}/>
18           <Route path="/visualizarcliente" component={VisualizarCliente}/>
19           <Route path="/visualizarservico" component={VisualizarServico}/>
20           <Route path="/servico/:id" component={Serviço}/>
21           <Route path="/cadastrarservico" component={Cadastrar}/>
22           <Route path="/editarservico/:id" component={Editar}/>
23         </Switch>
24       </Router>
25     </div>
26   );
27 }
```



## Criar o componente editar

5. De volta ao index.js do Editar, insira um botão listar.
6. Para passar o `id` recebido por parâmetro para o componente.

```
src > pages > Servico > Editar > js index.js > [edit] Editar
● 1 ~ import { Link } from "react-router-dom";
  2 import { useState } from "react/cjs/react.development";
  3 import { Container } from "reactstrap"
  4
  5 export const Editar = (props) => {
  6
  7   const [data, setData] = useState([]);
  8   const [id] = useState(props.match.params.id);
  9
 10
 11   return (
 12     <div>
 13       <Container>
 14         <div className="d-flex">
 15           <div className="mr-auto p-2">
 16             <h1>Editar Serviço</h1>
 17           </div>
 18           <div className="p-2">
 19             <Link to="/visualizarservico"
 20                   className="btn btn-outline-primary btn-sm mr-1"> Listar </Link>
 21           </div>
 22         </div>
 23       </Container>
 24     </div>
 25   )
 26 }
```



## Criar o componente editar

7. Para visualizar o serviço específico, insira mais um botão no Editar.

```
src > pages > Servico > Editar > JS index.js > [✖] Editar
● 16   return (
  17     <div>
  18       <Container>
  19         <div className="d-flex">
  20           <div className="mr-auto p-2">
  21             <h1>Editar Serviço</h1>
  22           </div>
  23
  24           <div className="p-2">
  25             <Link to="/visualizarservico"
  26               className="btn btn-outline-primary btn-sm mr-1"> Listar </Link>
  27             <Link to={"/servico/" + id}
  28               className="btn btn-outline-primary btn-sm mr-1">Consultar</Link>
  29           </div>
  30     </div>
```

8. Crie o Status para verificação de sucesso ou de erro.



# Criar o componente editar

```
5  export const Editar = (props) => {  
6  
7      const [data, setData] = useState([]);  
8      const [id] = useState(props.match.params.id);  
9  
10     const [status, setStatus] = useState({  
11         formSave:false,  
12         type:'',  
13         message:''  
14     });  
15 }
```

## 9. Insira os tratamentos de erro na página.

```
32             <hr className="m-1" />  
33  
34             {status.type === 'error' ? <Alert color="danger">  
35                 {status.message}</Alert>:""  
36             }  
37  
38             {status.type === 'success' ? <Alert color="success">  
39                 {status.message}</Alert>:""  
40             }  
41         </Container>  
42     </div>  
43 }
```



## Criar o componente editar

A função de atualizar é uma mistura de consultar e cadastrar, portanto...

10. Crie uma constante para cada campo do formulário iniciando com vazio.

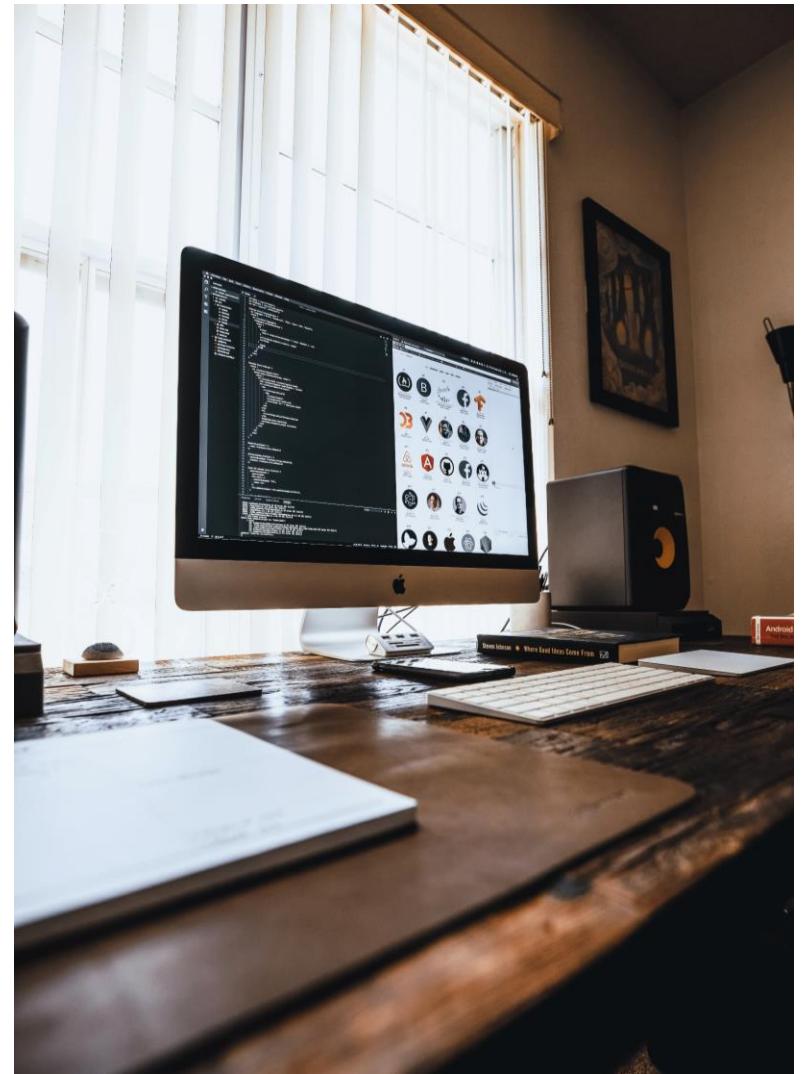
```
src > pages > Servico > Editar > JS index.js > [e] Editar
  1 import { Link } from "react-router-dom";
  2 import { useState } from "...react/cjs/react.development";
  3 import { Alert, Container } from "reactstrap"
  4
  5 export const Editar = (props) => {
  6
  7   const [data, setData] = useState([]);
  8   const [id] = useState(props.match.params.id);
  9   const [nome, setNome] = useState('');
10   const [descricao, setDescricao] = useState('');
11 }
```



# Implementar o formulário

1. Crie o formulário e a função `edtServico` para efetivar a alteração.

```
src > pages > Servico > Editar > js index.js > [e] Editar
  43             {status.type === 'success' ? <Alert color="success">
  44               {status.message}</Alert> : ""}
  45
  46             <Form className="p-2" onSubmit={edtServico}>
  47               <FormGroup className="p-2">
  48                 <Label>Nome</Label>
  49                 <Input type="text" name="nome"
  50                   placeholder="Nome do serviço" onChange={valorInput} />
  51               </FormGroup>
  52
  53               <FormGroup className="p-2">
  54                 <Label>Descrição</Label>
  55                 <Input type="text" name="descricao"
  56                   placeholder="Descrição do serviço" onChange={valorInput} />
  57               </FormGroup>
  58
  59             {status.formSave ?
  60               <Button type="submit" outline color="warning" disabled>Salvando...
  61                 <Spinner size="sm" color="warning" /></Button> :
  62                 <Button type="submit" outline color="warning">Salvar</Button>
  63             }
  64           </Form>
  65         </Container>
  66       </div>
  67     )
  68   }
  69 }
```



# Implementar o formulário

2. Crie a função `edtServico` para efetivar a alteração.

```
src > pages > Servico > Editar > JS index.js > Editar  
5  export const Editar = (props) => {  
6  
7      const [data, setData] = useState([]);  
8      const [id] = useState(props.match.params.id);  
9      const [nome, setNome] = useState('');  
10     const [descricao, setDescricao] = useState('');  
11  
12     const [status, setStatus] = useState({  
13         formSave: false,  
14         type: '',  
15         message: ''  
16     });  
17  
18     const edtServico = async e =>{  
19         e.preventDefault();  
20         console.log("Editar");  
21     }  
22 }
```



# Implementar o formulário

## 3. Acrescente o useEffect.

```
26     useEffect(() => {
27         const getServico = async () => {
28             await axios.get(api + "/servico/" + id)
29                 .then((response) => {
30                     console.log(response.data.servico);
31                     setData(response.data.servico);
32                 })
33                 .catch(() => {
34                     console.log("Erro: Não foi possível conectar a API.");
35                 })
36         }
37         getServico();
38     }, [id]);
```

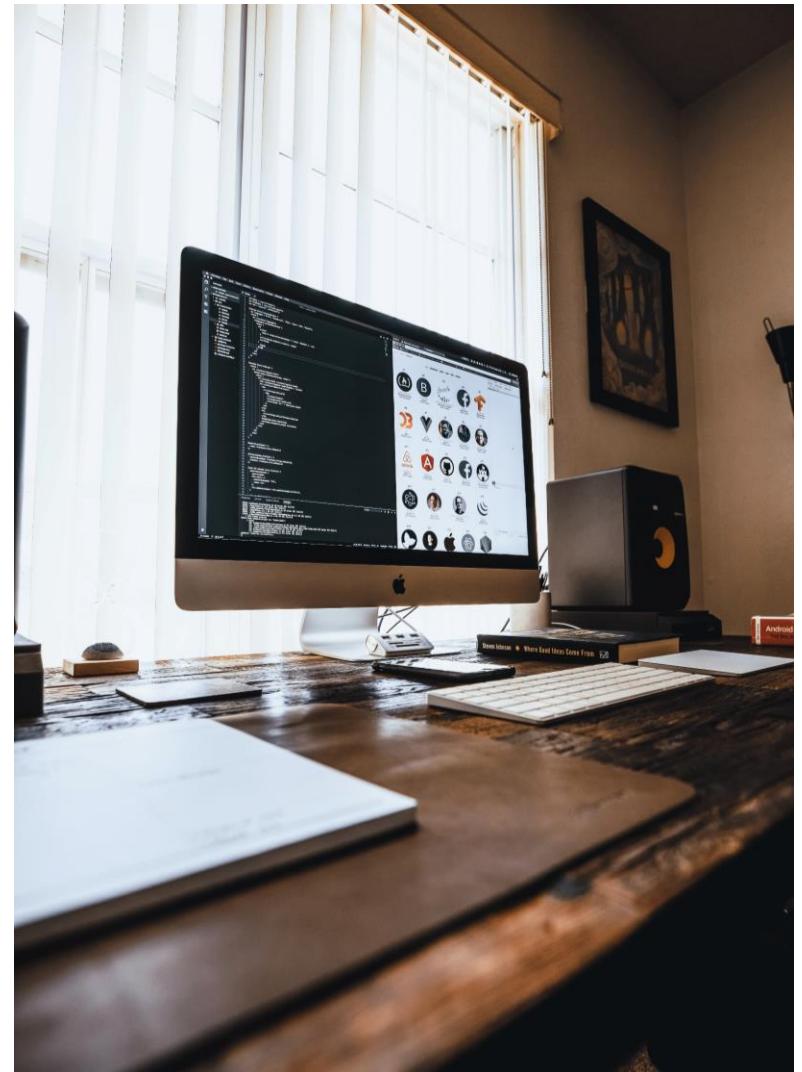
Verifique se está retornando no `console.log`. Se verificar pode comentar a linha 30 correspondente do código.



# Implementar o formulário

4. No formulário insira o atributo value que recebe o campo que retorna no console.log.

```
src > pages > Servico > Editar > JS index.js > ...
62           {status.message}</Alert> : ""
63
64           <Form className="p-2" onSubmit={edtServico}>
65             <FormGroup className="p-2">
66               <Label>Nome</Label>
67               <Input type="text" name="nome"
68                 placeholder="Nome do serviço" value={nome} />
69             </FormGroup>
70
71             <FormGroup className="p-2">
72               <Label>Descrição</Label>
73               <Input type="text" name="descricao"
74                 placeholder="Descrição do serviço" value={descricao} />
75             </FormGroup>
76
77             {status.formSave ?
78               <Button type="submit" outline color="warning" disabled>Salvando...
79                 <Spinner size="sm" color="warning" /></Button> :
80               <Button type="submit" outline color="warning">Salvar</Button>}
81
82           </Form>
83
84         </Container>
85       </div>
86     )
87 }
```



# Implementar o formulário

## 5. Agora vamos ajustar para setar nos campos corretos no useEffect.

```
src > pages > Servico > Editar > js index.js > Editar > useEffect() callback > getServico > then() callback
● 19      });
20
21      const edtServico = async e =>{
22          e.preventDefault();
23          console.log("Editar");
24      }
25
26      useEffect(() => {
27          const getServico = async () => {
28              await axios.get(api + "/servico/" + id)
29                  .then((response) => {
30                      //console.log(response.data.servico);
31                      setNome(response.data.servico.nome);
32                      setDescricao(response.data.servico.descricao);
33                  })
34                  .catch(() => {
35                      console.log("Erro: Não foi possível conectar a API.");
36                  })
37          }
38          getServico();
39      }, [id]);
40
```

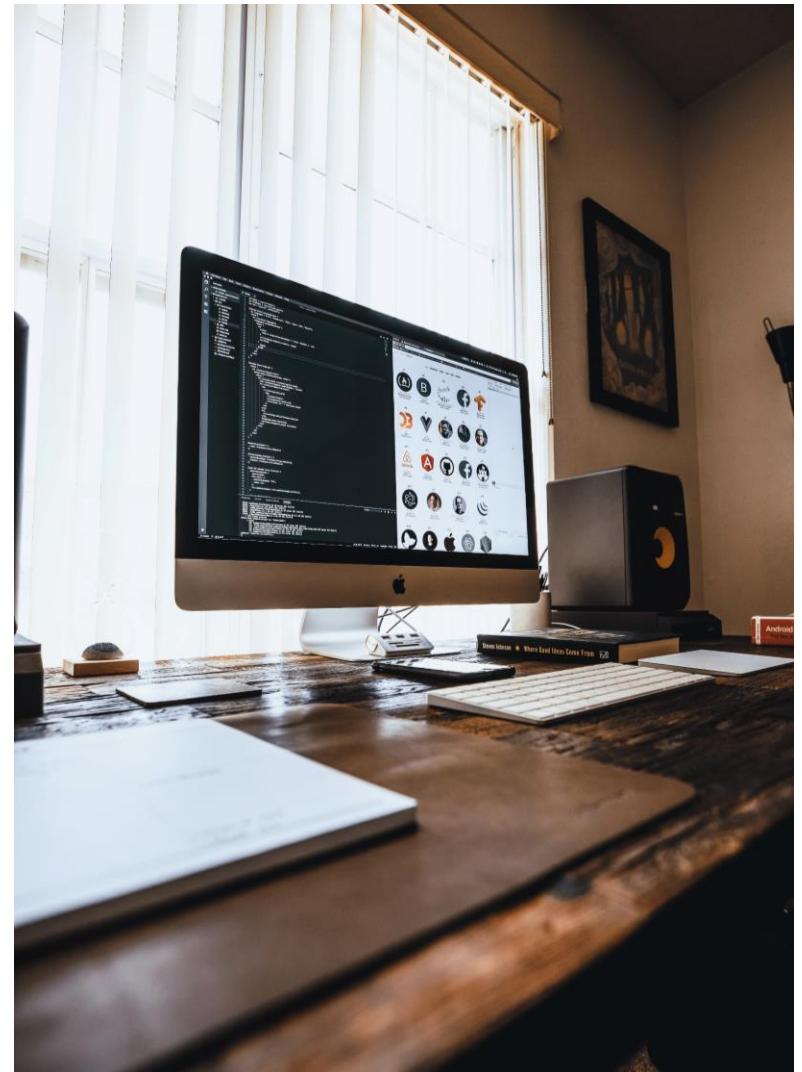


# Enviar para a API

Verifique se os campos no formulário estão sendo preenchidos.

1. No formulário insira o atributo `onChange`.

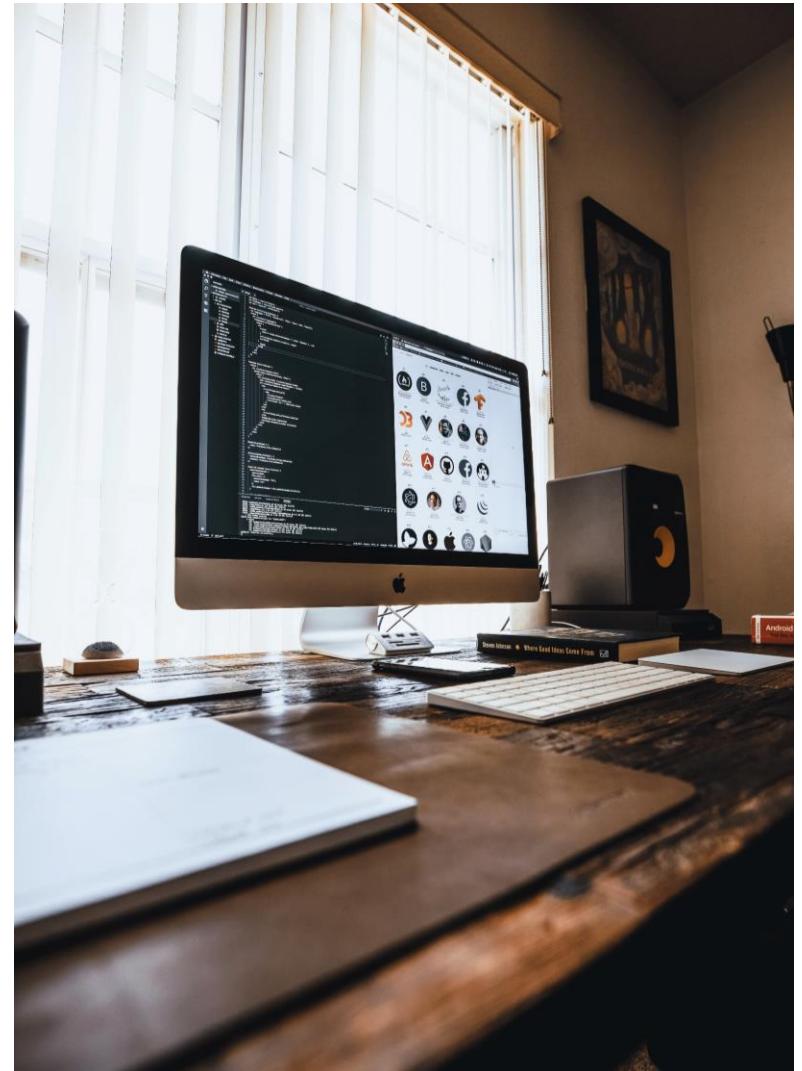
```
src > pages > Servico > Editar > js index.js > [e] Editar
62         {status.type === 'success' ? <Alert color="success">
63             {status.message}</Alert> : ""}
64
65     <Form className="p-2" onSubmit={edtServico}>
66         <FormGroup className="p-2">
67             <Label>Nome</Label>
68             <Input type="text" name="nome"
69                 placeholder="Nome do serviço" value={nome}
70                 onChange={e => setNome(e.target.value)} />
71         </FormGroup>
72
73         <FormGroup className="p-2">
74             <Label>Descrição</Label>
75             <Input type="text" name="descricao"
76                 placeholder="Descrição do serviço" value={descricao}
77                 onChange={e => setDescricao([e.target.value])} />
78         </FormGroup>
79
80         {status.formSave ?
81             <Button type="submit" outline color="warning" disabled>Salvando...
82                 <Spinner size="sm" color="warning" /></Button> :
83             <Button type="submit" outline color="warning">Salvar</Button>
84
85     </Form>
86
```



## Enviar para a API

### 2. Implemente a passagem dos parâmetros nos campos do formulário.

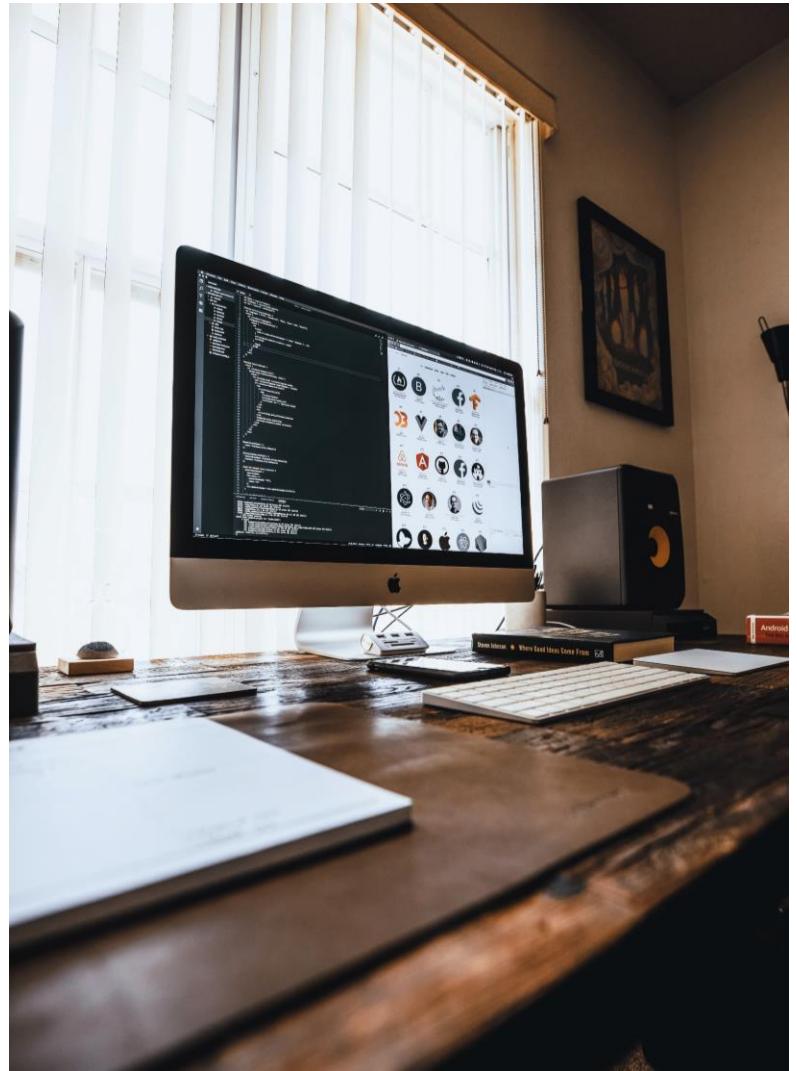
```
21 const edtServico = async e =>{
22     e.preventDefault();
23
24     const headers = {
25         'Content-Type': 'application/json'
26     };
27
28     await axios.put(api+"/editarservico",{id, nome, descricao}, {headers})
29     .then((response)=>{
30         console.log(response.data.error);
31         console.log(response.data.message);
32     })
33     .catch(() =>{
34         setStatus({
35             type: 'error',
36             message: 'Erro: Não foi possível conectar a API.'
37         });
38     });
39 }
40 }
```



# Enviar para a API

## 3. Faça a verificação de erro ao enviar os dados.

```
27
28     await axios.put(api+"/editarservico",{id, nome, descricao}, {headers})
29     .then((response)=>{
30         //console.log(response.data.error);
31         //|console.log(response.data.message);
32         if(response.data.error){
33             setStatus({
34                 formSave: false,
35                 type: 'error',
36                 message: response.data.message
37             });
38         }else{
39             setStatus({
40                 formSave: false,
41                 type: 'success',
42                 message: response.data.message
43             });
44         }
45     })
46     .catch(() =>{
47         setStatus({
48             formSave: false,
49             type: 'error',
50             message: 'Erro: Não foi possível conectar a API.'
51         });
52     })
53 }
```



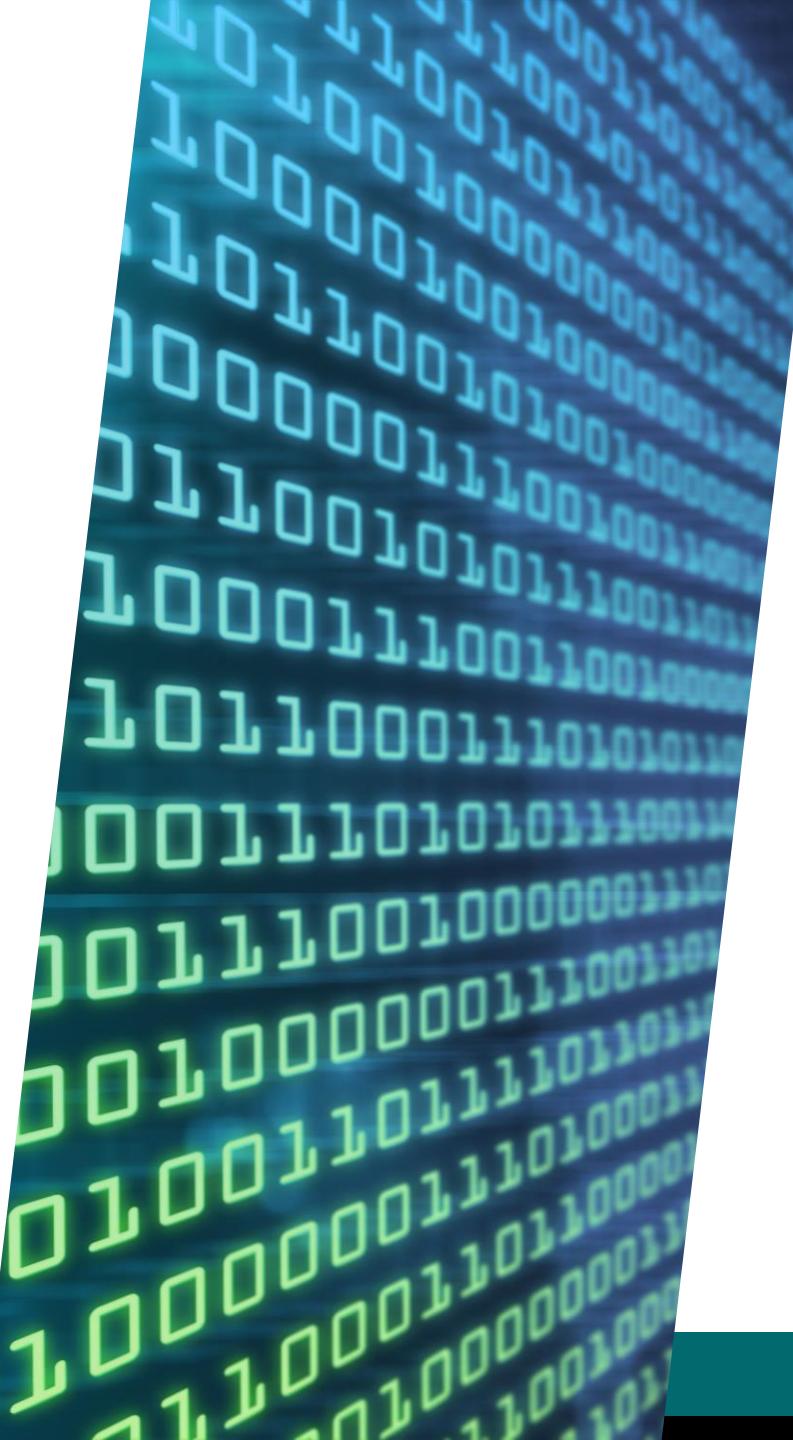
# Atualizar o repositório

Até aqui tudo bem?



## O que vem em seguida

No próximo encontro vamos excluir a partir do nosso sistema web.





@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)



@tiacademybrasil

# Front-end com React

DIA 4 | Profº. Erinaldo

## Dia 4: Implementar as alterações e exclusões

1. Criar o botão cadastrar
2. Criar o layout da página cadastrar serviço
3. Criar o formulário de cadastro de serviço
4. Criar a função para cadastrar
5. Implementar o tratamento de erro

**TOTAL: 22 horas**

**4<sup>a</sup> semana**

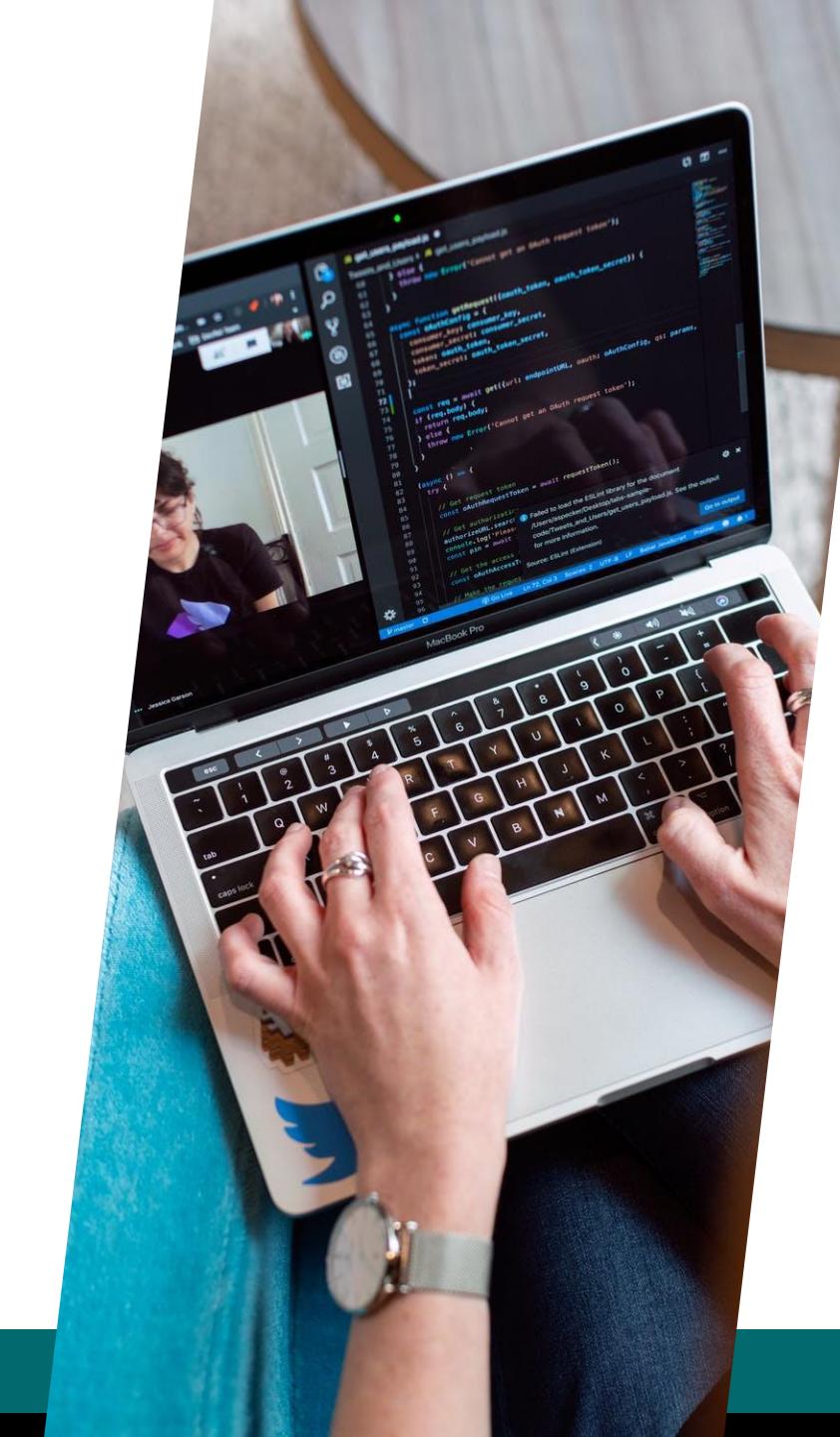


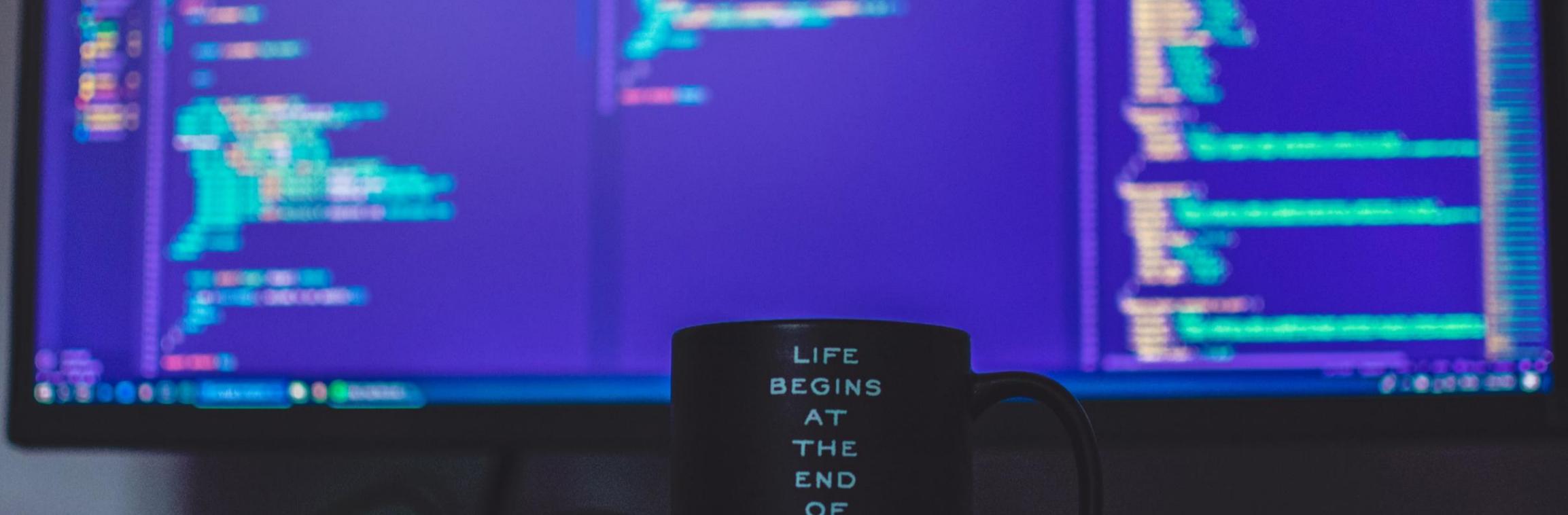
## Antes de começar...

Nas últimas aulas iniciamos uma aplicação front-end com ReactJS e implementamos um componente Menu utilizando o framework Reactstrap.

Na sequência acessamos a API no *back-end* e o banco de dados para consultar, cadastrar e editar a partir do *front-end*.

Na aula de hoje vamos implementar as funcionalidades para excluir objetos a partir do *front-end*.





@tiacademybrasil

# Implementar as Exclusões

Vamos praticar?

## Criar o layout da página cadastrar serviço

1. Certifique-se de estar conectado ao servidor de banco de dados.
2. Certifique-se de estar com o back-end executando.
3. Certifique-se de estar com o Postman ativado.
4. No diretório Servico do seu front-end crie uma pasta chamada Cadastrar.
5. Na pasta Cadastrar crie um novo arquivo chamado index.js.
6. Copie um modelo básico para o arquivo index.js.



# Implementar a exclusão

1. No formulário de listar os serviços, abaixo do botão editar, insira as tags span.

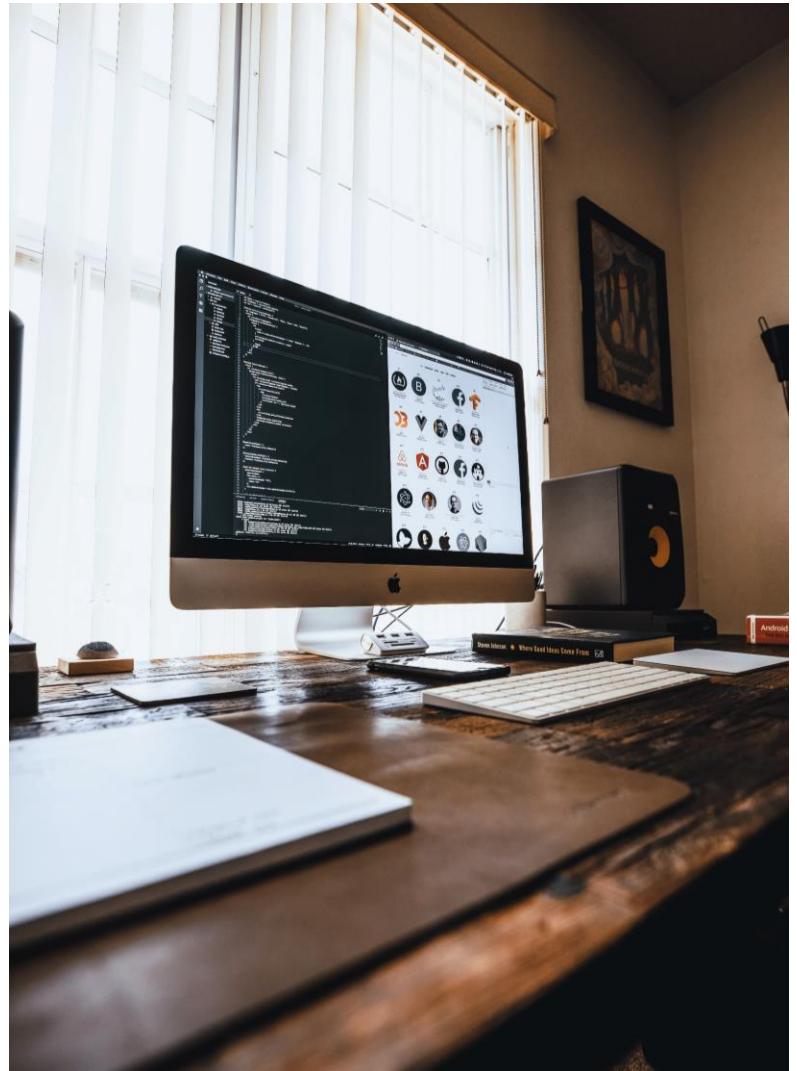
```
89 <td className="text-center">
90   <Link to={"/servico/" + item.id}>
91     className="btn btn-outline-primary btn-sm mr-1">Consultar</Link>
92   <Link to={"/editarservico/" + item.id}>
93     className="btn btn-outline-warning btn-sm mr-1">Editar</Link>
94   <span className="btn btn-outline-danger btn-sm mr-1"
95     onClick={()=> apagarServico(item.id)}>Excluir</span>
96 </td>
97 </tr>
98 </tbody>
99 </Table>
100
```

2. Crie a função apagarServico.



# Implementar a exclusão

```
35  const apagarServico = async(idServiço) =>{
36      console.log(idServiço)
37
38      const headers = {
39          'Content-Type': 'application/json'
40      };
41
42      await axios.delete(api+"/apagarservico/"+idServiço, {headers})
43      .then((response)=>{
44          console.log(response.data.error);
45      })
46      .catch(() =>{
47          setStatus({
48              type: 'error',
49              message: "Erro: Não foi possível se conectar a API"
50          });
51      })
52  }
```



# Implementar a exclusão

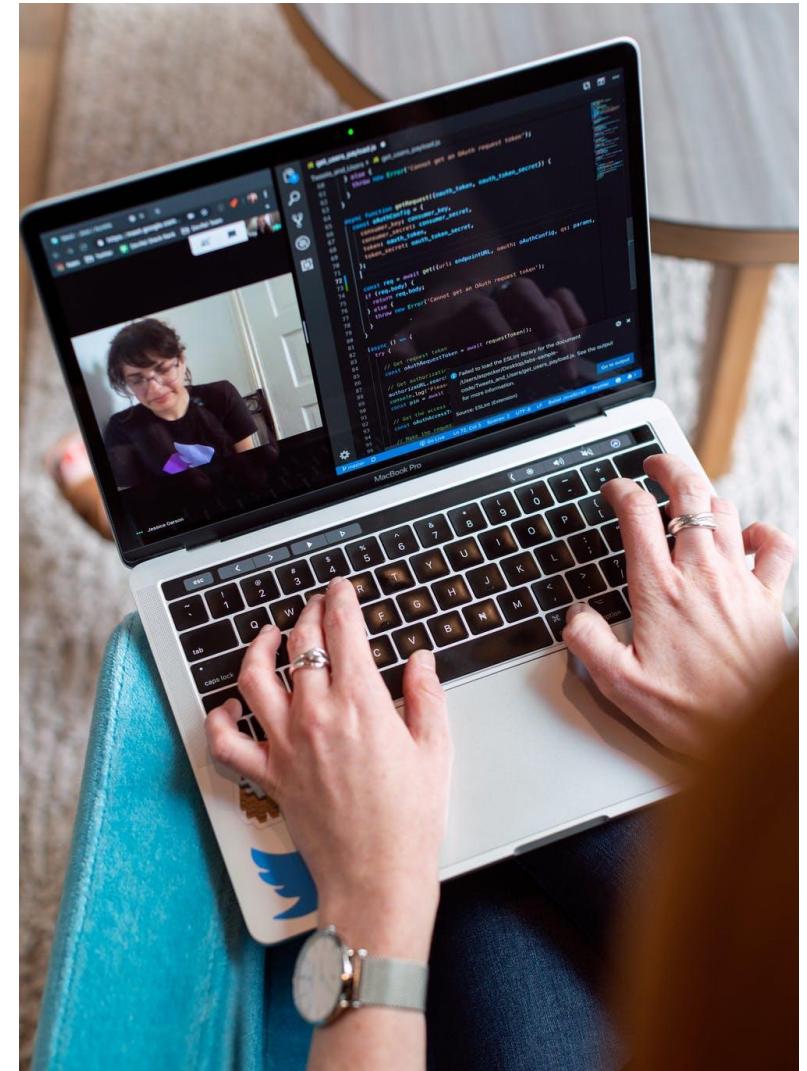
## 3. Chama o método para atualizar a página.

```
35  const apagarServico = async(idServiço) =>{
36      console.log(idServiço)
37
38      const headers = {
39          'Content-Type': 'application/json'
40      };
41
42      await axios.delete(api+"/apagarservico/"+idServiço, {headers})
43      .then((response)=>{
44          console.log(response.data.error);
45          getServicos();|
46      })
47      .catch(() =>{
48          setStatus({
49              type: 'error',
50              message: "Erro: Não foi possível se conectar a API"
51          });
52      })
53  }
```



# Exercícios

1. Implemente a atualização do cliente.
2. Implemente a atualização do pedido.
3. Implemente a exclusão do cliente.
4. Implemente a exclusão do pedido.



# Atualizar o repositório

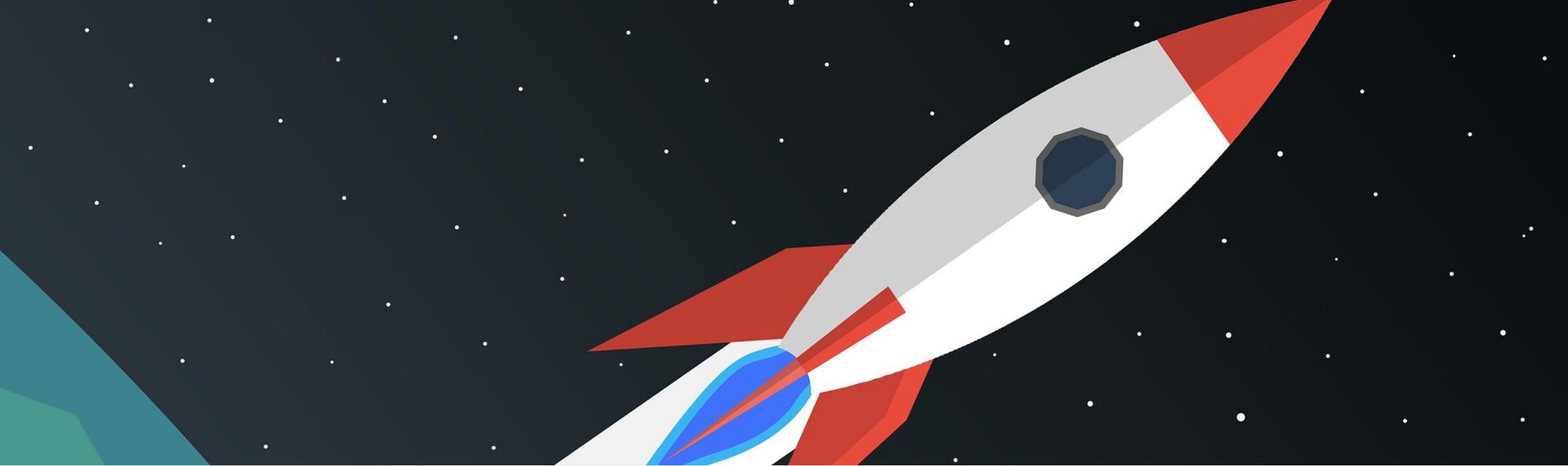
Até aqui tudo bem?



## Desafio

Vamos lá! O desafio agora é manipular o seu aplicativo.





@tiacademybrasil

---

#TAKEOFF  
@tiacademybrasil  
[www.tiacademybrasil.com.br](http://www.tiacademybrasil.com.br)