

Integração Básica de APIs

Labenu_



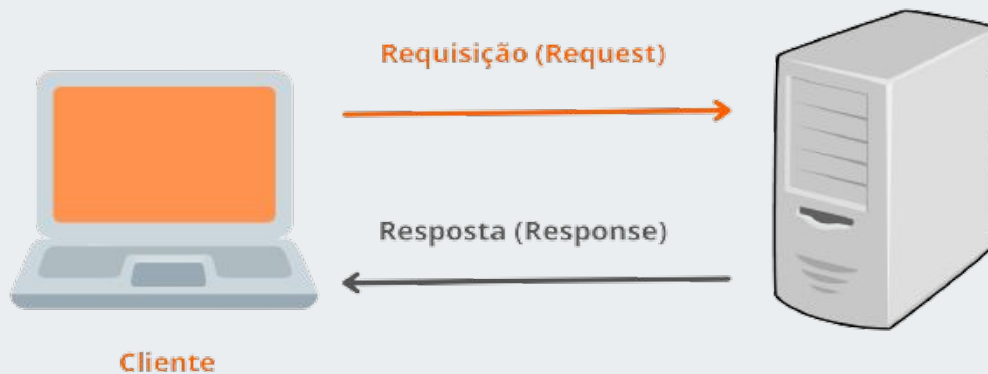
O que já vimos?

- Como criar aplicações React lidando com interatividade do usuário
- O que é e como funciona uma API
- Como consumir uma API no Postman



Relembrando...

- As interações acontecem por meio **requisições** HTTP
 - Frontend executa requisições
 - Backend responde



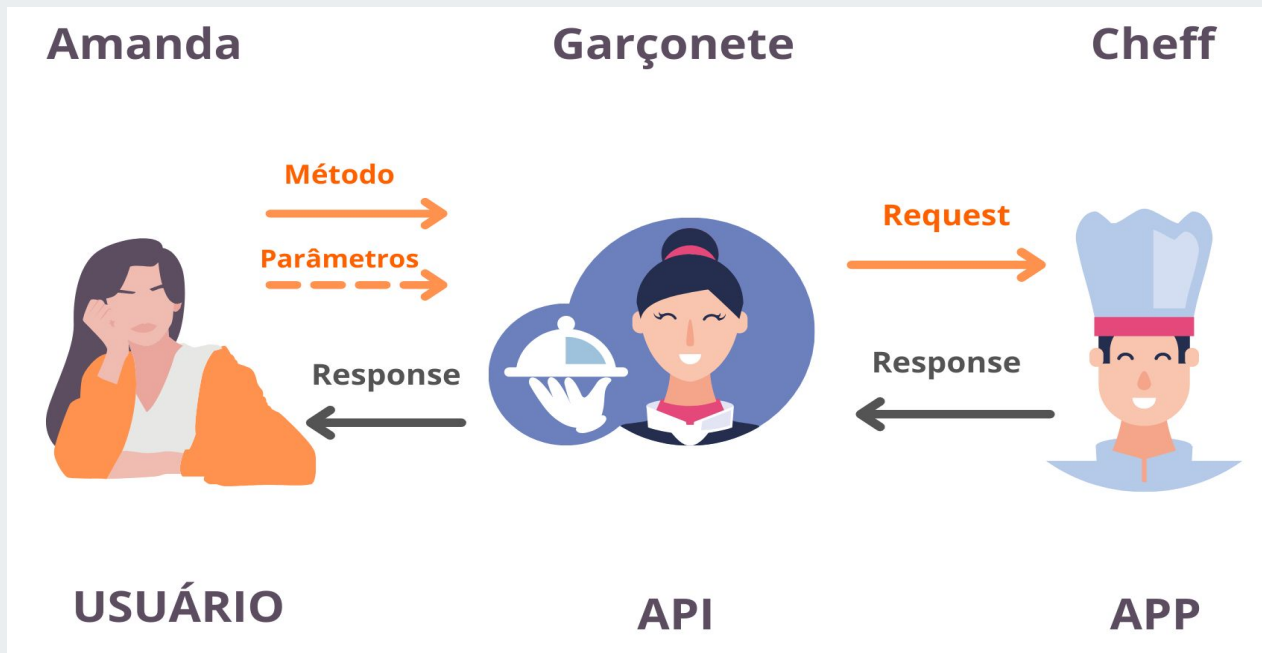
Cliente => Inicia a comunicação, fazendo um pedido para o servidor

Servidor => Recebe os pedidos do cliente, e obrigatoriamente fornece uma resposta.



Relembrando...

- **API** é uma **interface** não visual que permite a interação com o **backend**



Relembrando...

Importante: Leia a Documentação!!!!

The screenshot shows a REST client interface with the following components and annotations:

- Labenusers / getAllUsers**: An orange box highlights the collection and endpoint name, with an arrow pointing to the text **nome da coleção / nome do endpoint**.
- Método**: A teal arrow points to the **GET** dropdown menu.
- Path**: A teal arrow points to the URL `https://us-central1-labenu-apis.cloudfunctions.net/labenusers/users`.
- Headers (7)**: A pink box highlights the Headers tab, with a pink arrow pointing to the text **Onde passará Authorization**.
- Body**: A green box highlights the Body tab, with a green arrow pointing to the text **Onde passará info do Body**.
- Endpoint Formula**: A blue arrow points from the Method and Path to the text **ENDPOINT = Método + Path**.
- Status HTTP**: A red box highlights the **Status: 401 Unauthorized** message, with a red arrow pointing to the text **Status HTTP**.
- Response Text**: A purple box highlights the response body text `1 Verifique se você está passando o header "Authorization"`, with a purple arrow pointing to the text **Resposta da Requisição**.
- Image**: A photograph of a cat sitting in front of a glass door with a "No Pets" sign. Below the image, the text **401 Unauthorized** is displayed.

O que vamos ver hoje?

- Como consumir uma API usando Javascript
- Como lidar com dados externos no React
- Dados dinâmicos nas nossas aplicações



Requisições em Javascript

Labenu_



axios

- **Biblioteca** para fazer requisições
- "Igual" ao Postman, só que dentro do código
- Existem formas nativas (sem bibliotecas), mas o *axios* facilita várias coisas para nós
- Instalando:

```
$ npm install axios
```



axios

- Sintaxe - Exemplo

```
1 import axios from 'axios'
2
3 axios.get('https://minha-api.com', {
4   headers: {
5     "Authorization": 'nome-sobrenome-turma'
6   }
7 })
```



Problemas 🙈

- E se a requisição demorar muito?
 - Devemos parar a execução do código? NÃO
- E se der erro na requisição?
 - Parâmetros errados
 - Servidor com problema
 - Usuário sem internet



Tempo de Requisição

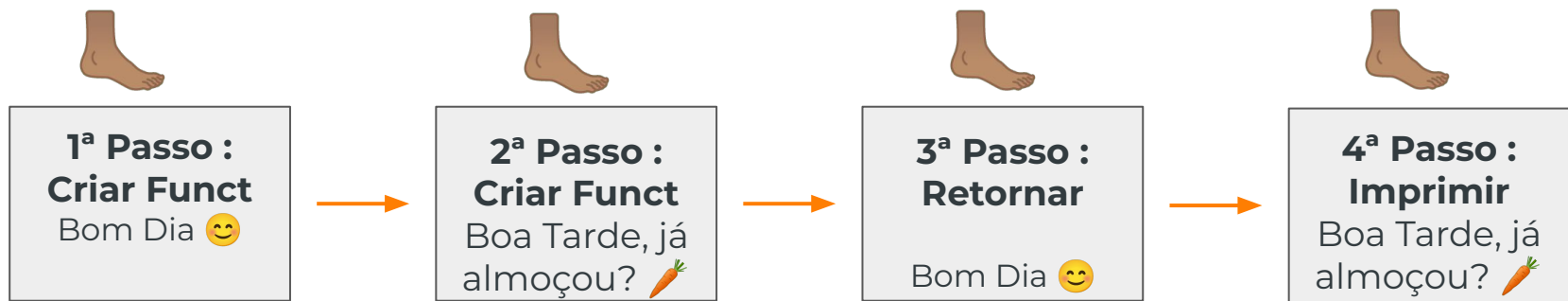
- Não temos controle sobre o tempo da requisição
- Não queremos que a aplicação fique travada enquanto esperamos
- Javascript criou uma solução: **assincronicidade**



Sincronicidade

Vamos ver na prática! 

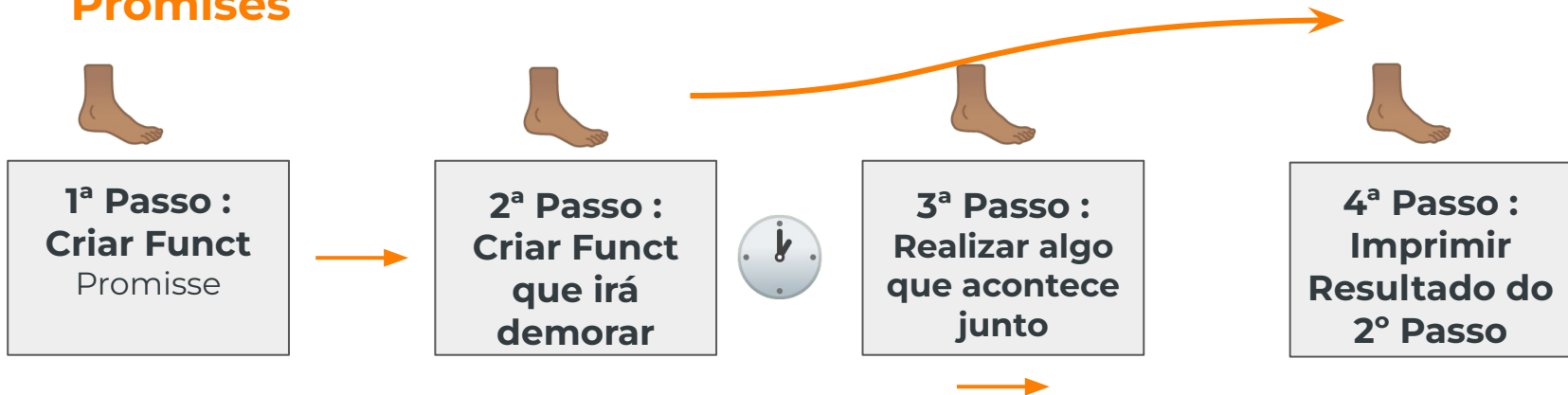
- Normalmente, o código é executado **linha após linha**
- Quando uma função é chamada, o código aguarda a **execução dela para prosseguir**



Assincronicidade

Vamos ver na prática! 

- Podemos fazer com que o Javascript execute **funções demoradas** de forma assíncrona
- A diferença é que o código **não espera sua conclusão** para prosseguir
- Nova estrutura nos permite lidar com isso de forma mais intuitiva: **Promises**



Promises

```
1 import axios from 'axios'
2
3 const request = axios.get('https://minha-api.com', {
4   headers: {
5     "Content-Type": 'application/json',
6     "Authorization": 'nome-sobrenome-turma'
7   }
8 })
9
10 request.then((response) => {
11   console.log(response.data)
12 })
```



Tratando erros 🙋

- Até agora, trabalhamos somente com dados e códigos criados por nós
- Todos os erros eram **responsabilidade nossa**
- Quando usamos integrações externas, estamos sujeitos a **erros que fogem do nosso controle**
- Precisamos **tratar** esse erro para que nossa aplicação não pare de funcionar



Tratando erros em promises 🙋

- Depois de todos os `.then()`, colocamos um `.catch()`



Deu Certo Faça Isso



Deu Errado Faça Isso

```
1 import axios from 'axios'
2
3 const request = axios.get('https://minha-api.com', {
4   headers: {
5     "Content-Type": 'application/json',
6     "Authorization": 'nome-sobrenome-turma'
7   }
8 })
9
10 request.then((response) => {
11   console.log(response.data)
12 }).catch((error) => {
13   console.log(error.message)
14 })
```



axios - Sintaxe

Sem body

```
1 import axios from 'axios'
2
3 axios.get('https://minha-api.com', {
4   headers: {
5     Authorization: 'nome-sobrenome-turma'
6   }
7 }).then((response) => {
8   console.log(response.data)
9 }).catch((error) => {
10  console.log(error.response.data)
11 })
```

Com body


```
1 import axios from 'axios'
2
3 const body = {
4   name: "Bob",
5   email: "bob@gmail.com"
6 }
7
8 axios.post('https://minha-api.com', body, {
9   headers: {
10     Authorization: 'nome-sobrenome-turma'
11   }
12 }).then((response) => {
13   console.log(response.data)
14 }).catch((error) => {
15   console.log(error.response.data)
16 })
17
```



axios - Sintaxe

 Método

 Endereço

 Headers

 Body
enviado

 Body da
Resposta

Sem body

```
1 import axios from 'axios'
2
3 axios.get('https://minha-api.com', {
4   headers: {
5     Authorization: 'nome-sobrenome-turma'
6   }
7 }).then((response) => {
8   console.log(response.data)
9 }).catch((error) => {
10  console.log(error.response.data)
11 })
```

Com body

```
1 import axios from 'axios'
2
3 const body = {
4   name: "Bob",
5   email: "bob@gmail.com"
6 }
7
8 axios.post('https://minha-api.com', body, {
9   headers: {
10     Authorization: 'nome-sobrenome-turma'
11   }
12 }).then((response) => {
13   console.log(response.data)
14 }).catch((error) => {
15   console.log(error.response.data)
16 })
17
```

1º Endereço → 2º Body → 3º Headers





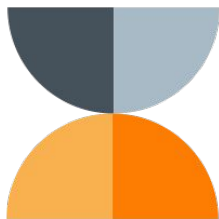
Exercício 1 - JS Puro

- Criar uma função que busca playlists e imprime no console usando a API do Labefy explorada ontem
- Criar função que cria uma nova playlist com nome como um parâmetro e imprime no console se deu sucesso ou erro
- [Documentação da API](#)



Pausa para relaxar 🤪

10 min



- Requisições em JS \Rightarrow usamos axios
- Dois novos problemas são introduzidos:
tempo da requisição e **erros**
 - **Tempo de requisição:** resolvido com assincronicidade usando **Promises**.
Usamos o método **.then()** para controlar ações que acontecem ao final da execução
 - **Erros:** são tratados usando o método **.catch()** em uma Promise



Requisições no React

Labenu_



Buscando Dados

- Normalmente queremos buscar dados para mostrar na tela ou tomar alguma ação baseada neles
- Depois de pegar o dado, é necessário guardá-lo em algum lugar
- Para isso, usamos o **estado**



Buscando Dados Automaticamente

- Frequentemente, vamos querer carregar os dados automaticamente assim que a tela carrega
- Requisições para buscar dados devem ser executadas **depois** da renderização dos componentes
- Não queremos que a tela fique travada enquanto a requisição é feita
- Para isso, usamos os métodos de **lifecycle**



Fazendo a Requisição

- Para buscar dados automaticamente, devemos usar o método de lifecycle `componentDidMount()`
- Fazer a requisição usando o axios e colocar o resultado no **estado**




```
1 class App extends React.Component {
2   state = {
3     playlists: [],
4   };
5
6   componentDidMount = () => {
7     this.pegarPlaylists()
8   };
9
10  pegarPlaylists = () => {
11    axios.get("https://us-central1-labenu-apis.cloudfunctions.net/labefy/playlists", {
12      headers: {
13        Authorization: "nome-turma"
14      }
15    }).then((resposta) => {
16      this.setState({ playlists: resposta.data.result.list })
17    }).catch((err) => {
18      console.log(err.message)
19    });
20  };
21
22  render() {
23    const renderedPlaylists = this.state.playlists.map((playlist) => {
24      return <p>{playlist.name}</p>;
25    });
26
27    return (
28      <div >
29        {renderedPlaylists}
30      </div>
31    );
32  }
33 }
```





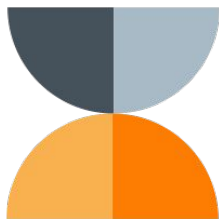
Exercício 2 - Pegar Dados

- Crie uma aplicação React que bate na API Labefy
- Crie uma função que pega a lista de playlists
- Mostre a lista apenas com os nomes das playlists na tela



Pausa para relaxar 🐾💤

5 min



Enviando Dados

- Requisições para enviar dados normalmente ocorrerão com base em um **evento** (ação do usuário)
- Frequentemente, mandaremos dados vindos de um **formulário**
- Para fazer isso, usamos os métodos que lidam com eventos



```

1 class App extends React.Component {
2   state = {
3     playlistValue: ""
4   };
5
6   criarPlaylist = () => {
7     const body = {
8       name: this.state.playlistValue
9     };
10
11     axios.post("https://us-central1-labenu-apis.cloudfunctions.net/labefy/playlists", body, {
12       headers: {
13         Authorization: "severo-dumont"
14       }
15     })
16     .then((resposta) => {
17       console.log(resposta.data)
18     }).catch((error) => {
19       console.log(error.message);
20     });
21   };
22
23   onChangePlaylistValue = (event) => {
24     this.setState({ playlistValue: event.target.value });
25   };
26
27   render() {
28
29     return (
30       <div >
31         <input
32           placeholder="Nome da Playlist"
33           value={this.state.playlistValue}
34           onChange={this.onChangePlaylistValue}
35         />
36         <button onClick={this.criarPlaylist}>Criar Playlist</button>
37       </div>
38     );
39   }
40 }

```





Exercício 3 - Enviar Dados

- Faça um formulário para criar uma nova playlist. Ao clicar no botão "Enviar", devemos chamar a função que cria uma nova playlist no servidor
- Depois que uma playlist for criada, a lista de playlists deve ser atualizada automaticamente



Resumo

Labenu_



Resumo

- Usaremos a lib axios para fazer requisições
- Isso traz vantagens mas introduz dois novos desafios: **tempo de requisição** e **erros**
- Para lidar com o tempo de requisição, dizemos para o código **continuar rodando** enquanto esperamos a requisição terminar utilizando **Promises** ⇒ **.then()**



Resumo

- Para tratamento de **erros**, utilizaremos o `.catch()`
- No React, quando queremos pegar dados:
 - Requisição no `componentDidMount()`
 - Guardamos os dados no state



Dúvidas? 🧐

Labenu_





Obrigado(a)!