

async e await

Labenu_



O que vamos ver hoje?

- Resolução do exercício proposto na aula passada
- `async/await`



Resolução Exercício

Labenu_





Exercício 1

Utilizando a API [Labenusers](#), crie duas telas:

- Tela de Cadastro de usuários
 - Solicita nome e email
 - Cria usuário e mostra mensagem de sucesso ou erro
- Tela de Lista de Usuários
 - Lista com os nomes de usuários
 - Botão de deletar cada usuário



async e await

Labenu_



Funções Assíncronas

- Na última aula vimos que para fazer requisições precisamos usar funções assíncronas
- Para facilitar um pouco a nossa vida, existem as Promises, com o `.then()` e o `.catch()`



Funções Assíncronas

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



Jeito novo - `async/await`

- Existe uma maneira de pedir para que o código **ESPERE** a execução da requisição antes de progredir
- Podemos usar a palavra **await** antes de uma Promise
- Isso só é possível dentro de funções assíncronas, que devem ser marcadas com a palavra **async**
- Resultado da Promise é passado diretamente para uma **variável**



Jeito novo - async/await

- Resultado da promise é passado diretamente para uma variável

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



Mas e os erros?

- Para tratar erros, usamos a sintaxe **try/catch**, como mostrado abaixo:

```
1 import axios from 'axios'
2
3 const getUsers = async () => {
4   try {
5     const response = await axios.get('https://users-api.com/users', {
6       headers: {
7         Authorization: 'nome-sobrenome-turma'
8       }
9     })
10
11     console.log(response.data)
12   } catch(error) {
13     console.log(error.response)
14   }
15 }
```



Mas e os erros?

- Para tratar erros, usamos a sintaxe **try/catch**, como mostrado abaixo:

```
1 import axios from 'axios'
2
3 const getUsers = async () => {
4   try {
5     const response = await axios.get('https://users-api.com/users', {
6       headers: {
7         Authorization: 'nome-sobrenome-turma'
8       }
9     })
10
11     console.log(response.data)
12   } catch(error) {
13     console.log(error.response)
14   }
15 }
```



Mas e os erros?

- Bloco dentro do **try** é executado

```
1 import axios from 'axios'
2
3 const getUsers = async () => {
4   try {
5     const response = await axios.get('https://users-api.com/users', {
6       headers: {
7         Authorization: 'nome-sobrenome-turma'
8       }
9     })
10
11     console.log(response.data)
12   } catch(error) {
13     console.log(error.response)
14   }
15 }
```



Mas e os erros?

- Se der erro, execução é interrompida e o bloco do **catch** é executado

```
1 import axios from 'axios'
2
3 const getUsers = async () => {
4   try {
5     const response = await axios.get('https://users-api.com/users', {
6       headers: {
7         Authorization: 'nome-sobrenome-turma'
8       }
9     })
10
11     console.log(response.data)
12   } catch(error) {
13     console.log(error.response)
14   }
15 }
```

Vamos ver na prática! 



Cuidado!

- Não transformar **métodos de lifecycle** em funções **assíncronas**
- Comportamentos inesperados podem acontecer
- Criar **função auxiliar** e **invocar** do método de lifecycle



Cuidado!

```
1 import React from 'react'
2 import axios from 'axios'
3
4 class App extends React.Component {
5   componentDidMount() {
6     this.getUsers()
7   }
8
9   getUsers = async () => {
10    try {
11      const response = await axios.get('https://users-api.com/users', {
12        headers: {
13          Authorization: 'nome-sobrenome-turma'
14        }
15      })
16
17      console.log(response.data)
18    } catch(error) {
19      console.log(error.response)
20    }
21  }
22 }
```





Exercício 2

Refatore as requisições do exercício anterior usando a sintaxe `async & await`

- Tela de Cadastro de Usuários
 - Criação do usuário (POST)
- Tela de Lista de Usuários
 - Listar usuários (GET)
 - Deletar usuário (DELETE)



Resumo

Labenu_



Resumo

- A sintaxe **async/await** é uma maneira diferente de lidar com a assincronicidade
- É equivalente à sintaxe **.then()** que vimos anteriormente; você pode escolher qual usar
- **Uso:** a função que vai conter a Promise deve ser marcada com a palavra **async** e antes da execução do request devemos escrever o comando **await**



Resumo

- Para tratamento de erros, usamos o bloco **try/catch**
- Não devemos marcar funções de **ciclo de vida** como `async`, isso pode causar alguns problemas inesperados
- Neste caso, criamos uma **função auxiliar** e chamamos ela no método de ciclo de vida desejado





Obrigado(a)!