**Aula 47 Aprofundamento em Promise**

**Exercícios de Tarde:**

No exercício de hoje, vamos treinar os conceitos de Promises com Typescript usando a mesma API e as mesmas funcionalidades que vimos durante a aula. Cada um dos exercícios abaixo possui uma pequena explicação, algumas dicas e um conjunto de perguntas. Crie um arquivo para cada exercício e responda as perguntas em comentários.

Documentação da API:

[Postman Documenter](https://documenter.getpostman.com/view/4233568/SztA99xo)

Não se esqueçam de instalar o axios com a tipagem correta

npm install axios @types/axios

* Exercício 1

Vamos começar fazendo uma função nomeada simples que retorne todos os assinantes da nossa aplicação. Ela deve ser assíncrona, porque utiliza o async para fazer a comunicação com o banco de dados. Além disso, por ora, inidique que ela vai retornar um array de "qualquer coisa".

Lembre-se de criar uma função "main" para testar suas funções assíncronas

*a. Qual endpoint você deve utilizar para isso?*

*b. Como indicamos o tipo de uma função assíncrona que retorna um "array de qualquer coisa"?*

*c. Implemente uma função nomeada que faça o que foi pedido.*

* + Dicas

c.

async function getSubscribers(): Promise<any[]> {

const users = await axios.get(`${baseUrl}/subscribers/all`);

return users.data;

};

* Exercício 2

Agora, para treinar um pouco da sintaxe, reescreva a função do exercício 1 utilizando *arrow function*.

*a. Explique, com suas palavras, a diferença da sintaxe de uma função nomeada assíncrona e uma arrow function assíncrona*

*b. Implemente a função solicitada, usando arrow function*

* + Dicas
  + const getSubscribers = async (): Promise<any[]> => {
  + const users = await axios.get(`${baseUrl}/subscribers/all`);
  + return users.data;
  + };
* Exercício 3

Agora, vamos melhorar um pouco a nossa função, criando uma tipagem que represente os assinantes da nossa aplicação. No nosso caso, eles possuem um id, name e email, como indicado abaixo:

type User = {

id: string;

name: string;

email: string;

}

*a. Se apenas trocarmos o retorno da função para: Promise<User[]> , teremos algum erro de tipagem?*

*b. Na aula, comentamos que sempre fazemos um mapeamento do resultado de uma Promise, caso seja inidicado que ela retorne um Promise<any>. Explique com as suas palavras o porquê de fazermos isso*

*c. Reimplemente a função, corretamente.*

* + Dicas

c.

const getSubscribers = async (): Promise<User[]> => {

const users = await axios.get(`${baseUrl}/subscribers/all`);

return users.data.map((res: any) => {

return {

id: res.id,

name: res.name,

email: res.email,

};

});

};

* Exercício 4

Vamos continuar as nossas funcionalidades da API. Crie uma função que permita criar uma nova notícia.

*a. Qual é o tipo dessa função? Por quê?*

*b. Implemente a função solicitada*

* + Dicas

b.

async function createNews(

title: string,

content: string,

date: number

): Promise<void> {

await axios.put(`${baseUrl}/news`, {

title: "Novidade a caminho",

content: "Labenu: uma nova escola de programação",

date: 1590522289000,

});

}

* Exercício 5

Agora, implemente uma função que receba um array de usuários e uma mensagem e envie essa mensagem como notificação para todos os usuários. A princípio, não utilize o Promise.all

*a. O que aconteceria se fizéssemos a requisição dentro de um forEach? É recomendável utilizá-lo nesse caso?*

*b. Implemente a função solicitada*

* + Dicas

b.

const sendNotifications = async (

users: User[],

message: string

): Promise<void> => {

const promiseArray: Promise<any>[] = [];

for (const user of users) {

await axios.post(`${baseUrl}/notifications/send`, {

subscriberId: user.id,

message: message,

});

}

console.log("All notifications sent");

};

* Exercício 6

Vamos reescrever a função anterior utilizando o Promise.all. Antes disso, responda as perguntas abaixo:

*a. O que o Promise.all faz?*

*b. Quais as vantagens de se utilizar o Promise.all no caso de se enviar as notificações para todos os usuários?*

*c. Reimplemente a função utilizando Promise.all*

* + Dicas

c.

const sendNotifications = async (

users: User[],

message: string

): Promise<void> => {

const promiseArray = [];

for (const user of users) {

promiseArray.push(

axios.post(`${baseUrl}/notifications/send`, {

subscriberId: user.id,

message: message,

})

);

}

await Promise.all(promiseArray);

};

* Exercício 7

Para finalizar, vamos implementar algumas funcionalidades:

*a. Crie uma função que crie um novo assinante no nosso jornal*

*b. Crie uma função que seja responsável pela criação de uma notícia. Só que, dessa vez, além de criar a notícia, ela deve enviar uma notificação para cada um dos usuários*

*c. Crie uma função que pegue todas as notificações de todos os usuários da aplicação*

* + Dicas

a.

const createSubscriber = async (name: string, email: string) => {

await axios.put(`${baseUrl}/subscribers`, {

name,

email

});

};

b.

const createAndSendNotifications = async () => {

try {

await createNews(

"Novidade a caminho",

"Labenu: uma nova escola de programação",

1590522289000

);

const users = await getSubscribers();

await sendNotifications(users, "Testando mensagens");

} catch (err) {

console.log("err: ", err.message);

}

};

c.

const getAllNotifications = async (): Promise<any> => {

const users = await getSubscribers();

const notificationPromises = [];

for (const user of users) {

notificationPromises.push(

axios.get(`${baseUrl}/subscribers/${user.id}/notifications/all`)

);

}

const result = await Promise.all(notificationPromises);

const dataFromResult = result.map((res) => res.data);

return dataFromResult;

};

* Desafios

No desafio de hoje, precisaremos que você entenda como criar uma nova Promise. Leia o arigo abaixo para entender:

[Converting callbacks to promises | Zell Liew](https://zellwk.com/blog/converting-callbacks-to-promises/)

O desafio é: você deve criar uma *arrow function* assíncrona que imprima a mensagem "Oi" no terminal depois de 5 segundo a partir do momento em que o script foi rodado.

* + Dica

Pesquise pela função setTimeout do node para implementar o "tempo de espera" de 5 segundos para imprimir a mensgem