7. Funções Assíncronas

Em JavaScript, utilizamos as funções assíncronas em situações onde temos uma operação que não é executada instantaneamente ou de forma síncrona. Essas funções são declaradas de forma muito semelhante às funções normais (ou síncronas), porém com uma pequena diferença, a palavra especial `async`:

```
async function consultaUsuarios() {
        // implementação
}
```

Essas funções possuem um tipo de retorno diferente, chamado `Promise` (promessa), que indica que o valor pode ou não estar disponível no futuro.

Assim como em outras linguagens, as funções assíncronas de JavaScript servem para executarmos uma tarefa sem bloquear o código, utilizamos o `async` junto do `await` para indicar onde a função deve aguardar pelo resultado de uma `Promise`.

```
async function getBlogPost() {
        let response = await
fetch('https://jsonplaceholder.typicode.com/posts/1');
        let post = await response.json();
        console.log(post);
}

getBlogPost();
```

Na função acima, podemos ver que o código espera a conclusão da função `fetch` para atribuir seu resultado à variável `response`, em seguida, aguarda a conversão de `response` utilizando a função `.json()`, e por fim retorna o conteúdo do post para o console.

Promise

Como dito anteriormente, `Promise` é uma classe especial cujo propósito é indicar a possível conclusão de uma operação, que poderá retornar um valor. Um objeto dessa classe possui três "estados" (momentos da execução), que são:

Pending
Nesse estado, a `Promise` se encontra pendente, ou seja, a operação ainda está em andamento, e seu resultado ou falha não foi retornado para o código;

Fulfilled
Nesse estado, a `Promise` foi concluída, e seu valor já está disponível;

Rejected
Nesse estado, a `Promise` foi concluída, porém por alguma razão seu valor não está disponível.
Exemplo: Falha de conexão

`resolve` e `reject`

Uma `Promise` pode ser concluída de duas formas, no cenário positivo, temos sua resolução, no cenário negativo, temos sua rejeição, observe o exemplo abaixo:

```
function verificarUsuario(usuario) {
        return new Promise((resolve, reject) => {
                setTimeout(() => {
                        if (usuario === "admin") {
                                resolve("Usuário verificado");
                        } else {
                                reject("Acesso negado");
                                }
                        }, 1000);
                });
        }
}
```

Após 1 segundo (1000 milissegundos) a `Promise` é resolvida ou rejeitada, com base no valor informado no parâmetro `usuario`.

`then`, `catch` e `finally`

Essas três funções estão disponíveis em toda `Promise`, e servem para:

`then`
É executada após a resolução da `Promise`, ela pode ser encadeada mais de uma vez para listar diversas operações a serem realizadas com o valor retornado, caso `then` realize outra ação assíncrona, o `then` seguinte será executado após o retorno do valor assíncrono.

```
async function getPostTitle() {
        await fetch('https://jsonplaceholder.typicode.com/posts/1')
                .then(res => res.json())
                .then(res => res.title);
}
```

Nesse exemplo, a função `fetch` é executada, após sua conclusão, o retorno é convertido em objeto através da chamada de `.json()`, note que essa é uma função assíncrona, porém o código só é retomado após sua conclusão, esse é um caso especial, onde o `async` e `await` estão implícitos, pois a `arrow function` retorna diretamente o valor da `Promise` `res.json()`. Após isso, título da resposta é retornado no segundo `then`.

`catch`
É executada somente caso a `Promise` seja rejeitada, `catch` pode ser chamado para realizar o tratamento desse erro, de forma semelhante ao bloco `try/catch`.

`finally`
Sempre é executada após os `then` e `catch` da `Promise`.

Considerando o exemplo anterior da função `verificarUsuario`, podemos utilizar `then` e `catch` da seguinte forma:

```
verificarUsuario("admin")
        .then(message => console.log(message))
        // "Usuário verificado"
        .catch(error => console.log(error));
        // Não será chamado neste caso

verificarUsuario("visitante")
        .then(message => console.log(message))
        // Não será chamado neste caso
        .catch(error => console.log(error));
        // "Acesso negado"
```

Desafios

Fácil

Crie uma função assíncrona que busque o conteúdo de https://jsonplaceholder.typicode.com/posts/1, e em seguida, converte o valor para um objeto.

Agora modifique a função para receber um parâmetro numérico, este parâmetro será usado no lugar do "id" do post (posts/1, sendo 1 o id).

Teste alguns valores de ids e verifique as diferenças dos retornos no console.

Médio

Faça um tratamento de erro utilizando o `catch` na função anterior, caso um erro ocorra, `catch` deve retornar `undefined`.

Difícil

Faça outras duas funções **separadas** para buscar os posts e usuários das seguintes rotas:
https://jsonplaceholder.typicode.com/posts/
https://jsonplaceholder.typicode.com/users/

Utilize o bloco `then` da busca de usuários para convertê-los em JSON e retornar somente os campos `id`, `username` e `email`.

Em seguida, faça um loop e para cada usuário, e adicione um array com as postagens associadas ao seu userId em um novo atributo chamado `posts`.

Dica: utilizamos `map` para retornar uma versão modificada de um array, e `filter` para retornar uma lista de elementos que satisfaçam uma condição.