# Criando Promises

DigitalHouse >
Coding School

# Índice

# Estados das Promises

As promises podem ter 3 diferentes estados: Pending (pendente), Fulfilled (cumprida), Rejected (rejeitada).

**Pending**: É o estado de nascimento de uma promise, significa dizer que ela ainda não obteve retorno sobre a operação pretendida, portanto não sabe-se dizer se foi bem ou mal sucedida.

# Estados das Promises

**Fulfilled**: Significa que a promessa foi cumprida, ou seja, a operação prometida ocorreu com sucesso e retornou o esperado.

**Rejected**: Quer dizer que ocorreu algum erro no decorrer da promessa e a operação relacionada a ela falhou.

# Anatomia da Promise

Uma promisse recebe **uma função** que recebe 2 parâmetros: **resolve** e **reject**.

```
{}   const minhaPromise = new Promise((resolve, reject) => {})
```

# Anatomia da Promise

O **resolve** deve ser chamado para finalizar a promise de forma bem sucedida, enquanto o **reject** deve servir para quando os pré-requisitos da promise não forem cumpridos.

```
{}
```

```
const minhaPromise = new Promise((resolve, reject) => {
    if(1) {
        resolve("A promise foi bem sucedida!")
    }
    reject("Ocorreu um erro!")
})
```

# Anatomia da Promise

Também podemos passar **argumentos** para as promises, basta fazer como está no exemplo:

```
const promiseComArgs = (argumento) => new Promise((resolve, reject) => {
    if(argumento > 1) resolve("O argumento é válido")
    reject("argumento inválido")
})
```

{}

# Consumindo suas Promises .then()

A função assíncrona retornará um resultado ou não. Enquanto isso, o código **continua em execução**.

```
obterUsuarios()
    .then(function(data){
        console.log(data);
    });

console.log("Continua a executar")
```

Função assíncrona.

Código que pode continuar a ser executado enquanto a promise está em execução.

# Promises .then()

A função assíncrona retornará um resultado ou não. Enquanto isso, o código **continua em execução**.

```
{}
```

```
obterUsuarios()

    .then(function(data){
        console.log(data);
    });

console.log("Continua a executar!")
```

Execute o console.log() SOMENTE SE **obterUsuarios()** retornar um resultado. Este é recebido por **.then()** dentro de seu retorno de chamada, neste caso no parâmetro **data**.

# Promises - Requisições Aninhadas

Às vezes, os **.then()** podem ter promises internas. Para resolver isso, precisamos utilizar outro **.then()** que entre em execução uma vez que se resolva o anterior.

```
{}
obterUsuarios()
    .then(function(data){
        return filtrarDados(data);
    })
    .then(function(filteredData){
        console.log(filteredData);
    })
```

# ATENÇÃO!

É importante lembrar que o **.then()** precisa retornar os dados processados para que possam ser usados por outro **.then()**.

# Promises .catch()

Caso **NÃO** obtenha resultado, é gerado um erro. Para isso, usamos **.catch()**, que encapsula quaisquer erros que possam ser gerados por meio de promessas. Dentro desse método, decidimos o que fazer com o erro. Ele é recebido como um parâmetro no callback **.catch ()**. No exemplo a seguir, mostraremos o erro no console:

{}

```
obterUsuarios()
    .then(function(data){
        console.log(data);
    })
    .catch(function(error){
        console.log(error);
    })
```

# ATENÇÃO!

É importante lembrar que toda promise precisa de um **.catch()** para tratamento de exceções!

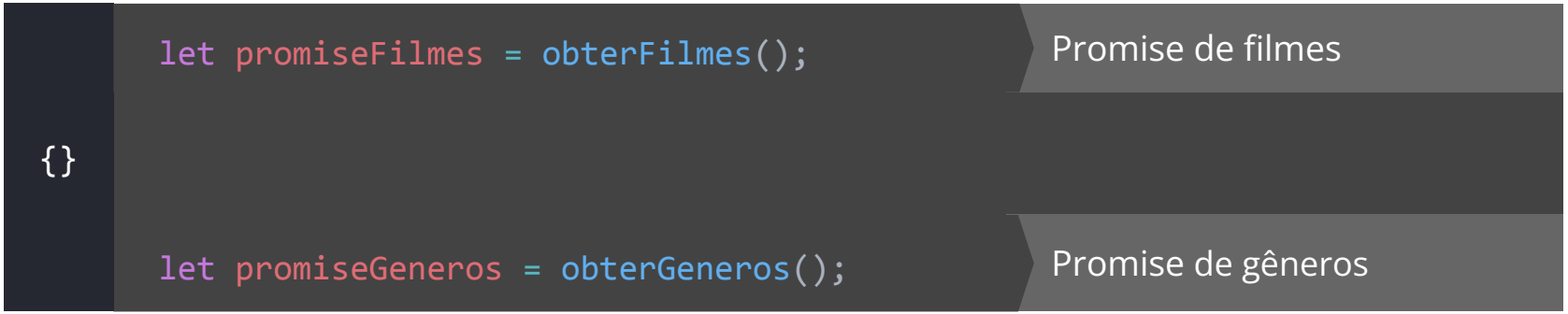
# Promises - Promise.all()

## Promises em conjunto

Às vezes, precisamos que duas ou mais promessas sejam resolvidas para realizar uma determinada ação. Para isso, usamos Promise.all(). Isso conterá uma série de promessas de que, uma vez resolvidas, um .then() será executado com os resultados da mesma.

# Promises - Promise.all()

O que devemos fazer primeiro é salvar em variáveis as promessas que precisamos obter.



# Promises - Promise.all()

A próxima etapa é usar o método **Promise.all()** que conterá um array com as promises que salvamos anteriormente.

```
{}  Promise.all([promiseFilmes, promiseGeneros])
```

Adicionando promises no array.

# Promises - Promise.all()

A callback .then () recebe uma matriz com os resultados das promessas cumpridas.

{}

```
Promise.all([promiseFilmes, promiseGeneros])

.then(function([resultadoFilmes,resultadoGeneros]){
  console.log(resultadoFilmes, resultadoGeneros);
})
```

O **.then()** somente executará se ambas as promises se cumprirem.

# Promises - Documentação

Para saber mais sobre uso de promises e Promise.all (), você pode acessar a documentação oficial do Mozilla, clicando nos seguintes links:

- Promise.all()
- Uso de promises

DigitalHouse>
Coding School