Promises JavaScript:

Promises são como promessas da vida real, ela está esperando algum retorno;

Para criar uma promessa instanciamos a classe Promise;

Que leva dois argumentos resolve (solução) e reject (erro)

Para encadear mais processos utilizamos o método then.

O resolve e o reject só pode receber um parâmetro por vez.

//Quando uma promisse é criada quer dizer que algo vai acontecer nesse código

//Como por exemplo, nós vamos esperar uma resposta de um servidor externo

// e assim que essa resposta for recebida nós iremos resolver essa promessa

//Quando tudo estiver certo com a promessa nós utilizaremos o "resolve"

//Quando der algum erro ou a resposta não foi a esperada utilizaremos o "reject"

//Nós podemos continuar a promessa baseada na resposta que nós recebemos utilizando o then

const myPromise = new Promise((resolve, reject) => {

    const nome = 'matheus' //Imagine que esse nome chegou de uma requisição externa

    if(nome === 'matheus'){

        resolve('Usuário Matheus encontrado!')

    } else {

        reject('o usuário Matheus não foi encontrado!')

    }

})

//Nós usamos o metodo then() e catch para poder ver a promessa sendo executada

//A promessa só vai ser executada com o metodo then() e/ou catch, sem ele o console.log ficará em branco

//O resolve() executa o then, e o reject() executa o catch

//Se o seu código entrou no bloco reject() e nao existe um "catch" o código dara erro

myPromise.then((resposta) => {

    console.log(resposta)

}).catch(resposta => {

    console.log(resposta)

})

Retorno:

Usuário Matheus encontrado!

Se:

const nome = 'gabriel' //Imagine que esse nome chegou de uma requisição externa

retorno:

o usuário Matheus não foi encontrado!

Muitas vezes você busca coisa de recursos onde não existe a certeza exata de quando esses recursos vão voltar pra você.

Ex: Você se conecta a uma API e pede dados de um usuário, ai você (seu código) não tem a noção exata de quando esses dados vão retornar pra você, para posteriormente você poder trabalhar com esses dados. A solução antiga seriam as funções de call-back, ou seja, você pede um dado e você passa como parâmetro de uma função uma função de call-back que será executada quando aquele dado estiver pronto, e as promises vieram para resolver esse problema.

Obs: precisamos fazer algo que ocorre de forma assíncrona parecer acontecer de forma síncrona.

Código problema:

function rand(min, max){

    min \*= 1000

    max \*= 1000

    return Math.floor(Math.random() \* (max - min) + min)

}

function esperaAi(msg, tempo){

    setTimeout(() => {

        console.log(msg)

    }, tempo)

}

esperaAi('Frase 1', rand(1, 3))

esperaAi('Frase 2', rand(1, 3))

esperaAi('Frase 3', rand(1, 3))

Retorno:

Frase 2

Frase 1

Frase 3

Outra possibilidade de retorno:

Frase 1

Frase 3

Frase 2

O caso acima é um código assíncrono, não temos controle de qual “frase” será executada primeira pois cada uma delas tem um tempo (neste exemplo o tempo é de 1 a 3 segundos) de resposta que é gerado aleatoriamente, uma possível solução para resolver esse problema e fazer com que as frases fiquem sincronizadas (‘frase1, frase2, frase2”) é utilizando funções call-back.

Utilizando call-back:

function rand(min, max){

    min \*= 1000

    max \*= 1000

    return Math.floor(Math.random() \* (max - min) + min)

}

function esperaAi(msg, tempo, callback){

    setTimeout(() => {

        console.log(msg)

        if(callback) callback()

    }, tempo)

}

esperaAi('Frase 1', rand(1, 3), function(){

    esperaAi('Frase 2', rand(1, 3), function(){

        esperaAi('Frase 3', rand(1, 3))

    })

})

O único retorno possível é:

Frase 1

Frase 2

Frase 3

Não importa o tempo de resposta de cada frase, a “frase 3” só pode ser executada após a “frase 2”, e a “frase 2” só pode ser executada após a “frase 1”.

Podemos resolver esse problema de outra forma, ao invés de usar call-back vamos usar Promises.

function rand(min, max){

    min \*= 1000

    max \*= 1000

    return Math.floor(Math.random() \* (max - min) + min)

}

function esperaAi(msg, tempo){

    return new Promise((resolve, reject) => {

        setTimeout(() => {

            resolve(msg)

        }, tempo)

    })

}

esperaAi('Frase 1', rand(1, 3))

    .then( resposta => {

        console.log(resposta)

        return esperaAi('Frase 2', rand(1, 3)) //Estamos retornando outra promise dentro deste "then()"

    })

    .then(resposta => {

        console.log(resposta)

        return esperaAi('Frase 3', rand(1, 3))

    })

    .then(resposta => {

        console.log(resposta)

    })

/\* As promises são executadas em paralelo, isso significa que o javascript executa os códigos que ja estão prontos

antes da promise, pois a promise precisa esperar um valor chegar ate ela \*/

console.log('este código será executado antes da promise')

Retorno:

este código será executado antes da promise

Frase 1

Frase 2

Frase 3