11

Experimento com promise

Clóvis, desejando compreender melhor o efeito do encadeamento de *promises* (*promise chainning*) e o tratamento de erro, criou três funções que retornam *promises*. Ele simulou um processamento assíncrono usando `setTimeout`, ou seja, a chamada de `resolve` de cada *promise* será chamada depois de alguns segundos. Veja que cada função tem um tempo de espera diferente da outra:

```
<!-- teste promise -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <script>

        function a(falhar) {

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

                setTimeout(() => {

                    if(falhar) {

                        reject('PROMISE A FALHOU');
                    } else {

                        console.log('PROMISE A RESOLVIDA');
                        resolve('DADO A');
                    }

                }, 2000);
            });
        }

        function b(falhar) {

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

                setTimeout(() => {

                    if(falhar) {

                        reject('PROMISE B FALHOU');
                    } else {

                        console.log('PROMISE B RESOLVIDA')
                        resolve('DADO B');
                    }

                }, 1000);
```

```
            });
        }

        function c(falhar) {

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

                setTimeout(() => {

                    if(falhar) {

                        reject('PROMISE C FALHOU');
                    } else {

                        console.log('PROMISE C RESOLVIDA')
                        resolve('DADO C');
                    }

                }, 500);
            });
        }

    </script>
</body>
```

Clóvis, espertamente, fez com que cada função recebesse um parâmetro. Se o valor passado for `true`, a *promise* será rejeitada. Uma maneira de simular um erro durante seu processamento.

Ele fez o seguinte teste:

```
<!-- teste promise -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <script>

        // código das funções omitido

        // teste, fazendo a promise c ser rejeitada

            a()
            .then(dado => {
                console.log(dado);
                // O RETORNO DA PROMISE B ESTARÁ DISPONÍVEL NO PRÓXIMO THEN
                return b();
            })
            .then(dado => {
                console.log(dado);

                /* FORÇANDO A REJEIÇÃO DA PROMISE. TEM QUE IR DIRETO PARA O CATCH.
                SE NÃO TIVESSE REJEITADO, O RETORNO DE C ESTARIA DISPONÍVEL NO PRÓXIMO
```

```
            return c(true);
        })
        .then(dado => {
            console.log(dado);
        })
        .catch(erro => console.log(erro));

    </script>
</body>
```

O objetivo de Clóvis é saber se a função `catch` será chamada se a *promise* retornada por `c` for rejeitada, inclusive mostrando no console a mensagem de erro ** PROMISE C FALHOU ***. *Dessa forma, ele terá certeza que durante o encadeamento das *promises*, qualquer erro será capturado em um único lugar, no caso em `.catch`.

Sobre o código acima, marque as alternativas que julgar verdadeira:

Selecione 2 alternativas

A A *promise* C demorará mais de 1 segundo para ser chamada, apesar do valor de `setTimeout` ser meio segundo (500ms).

B É exibida no console a mensagem "PROMISE C FALHOU".

C Nenhuma mensagem será exibida no console.