Aula 02.1: Promises

Promises em JavaScript e Uso de Async/Await

Introdução ao Conceito de Promises

O que é uma Promise?

Uma **Promise** em JavaScript é um objeto que representa a eventual conclusão (ou falha) de uma operação assíncrona e seu valor resultante. Promises permitem que você lide com operações assíncronas de uma maneira mais gerenciável e compreensível, ao invés de usar callbacks, que podem levar ao problema conhecido como "callback hell".

Estados de uma Promise

Uma Promise pode estar em um dos seguintes estados:

- **Pending (Pendente)**: Estado inicial, ainda não concluída ou rejeitada.
- **Fulfilled (Concluída)**: A operação foi concluída com sucesso, e a Promise possui um valor.
- **Rejected (Rejeitada)**: A operação falhou, e a Promise possui um motivo para a falha.

Exemplo Simples de uma Promise

```
javascriptCopy code
const aPromise = new Promise((resolve, reject) => {
    let success = true;

    if (success) {
        resolve("A operação foi bem-sucedida!");
    } else {
        reject("Houve uma falha na operação.");
```

```
    }
});

aPromise
    .then(result => {
        console.log(result); // A operação foi bem-sucedid
a!
    })
    .catch(error => {
        console.error(error); // Houve uma falha na operaçã
o.
    });
```

Usando Promises com Operações de Arquivos

Leitura de Arquivos com Promises

No contexto de leitura de arquivos em Node.js, você pode usar o módulo `fs` com Promises para tornar o código mais fácil de ler e manter.

Exemplo: Leitura de Arquivos com Promises

```
javascriptCopy code
const fs = require('fs').promises;

fs.readFile('arquivo.txt', 'utf8')
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error('Erro ao ler o arquivo:', error);
    });
```

Neste exemplo, a função `readFile` retorna uma Promise que é resolvida com o conteúdo do arquivo, ou rejeitada se houver um erro.

Escrita de Arquivos com Promises

Você também pode usar Promises para escrever arquivos de maneira assíncrona.

Exemplo: Escrita de Arquivos com Promises

```
javascriptCopy code
const fs = require('fs').promises;

fs.writeFile('novoArquivo.txt', 'Este é um novo conteúdo.')
    .then(() => {
        console.log('Arquivo escrito com sucesso!');
    })
    .catch(error => {
        console.error('Erro ao escrever o arquivo:', erro
r);
    });
```

Aqui, a função `writeFile` retorna uma Promise que é resolvida quando o arquivo é escrito com sucesso, ou rejeitada em caso de erro.

Introdução ao Async/Await

O que é Async/Await?

`async/await` é uma sintaxe introduzida no ES2017 (ES8) que permite trabalhar com Promises de maneira mais simples e legível. Ao usar `async/await`, você pode escrever código assíncrono que se parece mais com código síncrono, eliminando a necessidade de encadeamento de `.then()`.

Como Funciona?

- `async`: Uma função marcada com `async` sempre retorna uma Promise.
- `await`: A palavra-chave `await` pode ser usada dentro de funções `async` para pausar a execução da função até que a Promise seja resolvida ou rejeitada.

Exemplo: Leitura de Arquivos com Async/Await

Aqui está como você pode usar `async/await` para ler um arquivo:

```
javascriptCopy code
const fs = require('fs').promises;

async function lerArquivo() {
    try {
        const data = await fs.readFile('arquivo.txt', 'utf
8');
        console.log(data);
    } catch (error) {
        console.error('Erro ao ler o arquivo:', error);
    }
}

lerArquivo();
```

Como funciona:

- A função `lerArquivo` é marcada como `async`.
- `await fs.readFile('arquivo.txt', 'utf8')` pausa a execução da função até que a Promise retornada por `readFile` seja resolvida.
- O uso de `try/catch` permite capturar e lidar com erros que possam ocorrer durante a leitura do arquivo.

Exemplo: Escrita de Arquivos com Async/Await

Você também pode escrever arquivos usando `async/await` da seguinte maneira:

```
javascriptCopy code
const fs = require('fs').promises;

async function escreverArquivo() {
    try {
        await fs.writeFile('novoArquivo.txt', 'Conteúdo esc
rito com async/await.');
        console.log('Arquivo escrito com sucesso!');
    } catch (error) {
        console.error('Erro ao escrever o arquivo:', erro
```

```
r);
    }
}

escreverArquivo();
```

Neste exemplo, a função `escreverArquivo` usa `await` para aguardar a conclusão da escrita do arquivo antes de prosseguir.