Aula 9

JavaScript Assíncrono: A Web em Movimento

Roteiro

- O Problema: Por que o código síncrono trava a tela?
- A Primeira Solução: setTimeout e o conceito de Callbacks.
- A Base Moderna: O que são Promises?
- Conversando com a Internet: A Fetch API.
- A Sintaxe Limpa: Async/Await.
- Mão na Massa: Exercício prático com uma API real.

O Problema: Síncrono vs. Assíncrono

- Conceito Síncrono: JavaScript executa uma tarefa de cada vez, em ordem.
- Analogia: Restaurante com um único garçom que espera cada pedido ficar pronto antes de atender outro.
- Demonstração do Problema:

```
console.log("Primeiro passo");
alert("Eu travo tudo! Nada acontece até você me fechar.");
console.log("Passo final (só executa depois do alert)");
```

- Solução Assíncrona: Executa tarefas demoradas em segundo plano sem congelar a interface.

A Primeira Solução: setTimeout e Callbacks

- **`setTimeout`**: Executa uma função (callback) após um tempo definido.
- Callback: Função passada como argumento para ser executada mais tarde.
- Exemplo:

```
console.log("Pedido enviado para a cozinha.");
setTimeout(() => { console.log("Sua pizza está pronta!"); }, 2000);
console.log("Cliente esperando com a bebida na mesa.");
```

- Atenção: Muitos callbacks aninhados podem levar ao 'Callback Hell'.

A Base Moderna: Promises

- Promise: Objeto que representa o eventual sucesso ou falha de uma operação assíncrona.
- Analogia: Recibo de lanchonete que garante a entrega futura ou aviso de problema.
- Estados: Pending, Fulfilled, Rejected.
- Uso: `.then()` para sucesso, `.catch()` para falha.

Conversando com a Internet: A Fetch API

- Fetch API: Ferramenta moderna para requisições HTTP no navegador.
- `fetch()` retorna uma Promise.
- Exemplo:

```
fetch('https://jsonplaceholder.typicode.com/users/1')
.then(r => r.json())
.then(usuario => console.log(usuario.name))
.catch(e => console.error(e));
```

- Nota: `response.json()` também retorna uma Promise.

A Sintaxe Limpa: Async/Await

- Async/Await: Açúcar sintático sobre Promises para código mais legível.
- `async`: Antes de função, garante que retorna Promise.
- `await`: Pausa execução até Promise resolver.
- Exemplo:

```
async function buscarUsuario() {
    try {
        const r = await
        fetch('https://jsonplaceholder.typicode.com/users/1');
        const usuario = await r.json();
        console.log(usuario.name);
    } catch (e) {
     console.error(e);
    }
}
```

- Vantagem: Uso simples de try...catch para erros.

Exemplo: FM-DB API

- Criar função buscarFilmes(nome) que:
  1. Recebe nome do filme.
  2. Usa fetch e async/await para buscar na API.
  3. Lista pôster e nome do filme
- URL: https://imdb.iamidiotareyoutoo.com/search?q=${query}
- Dica: Analise a estrutura do objeto recebido

Resumo da Aula

- Código assíncrono evita travar interfaces.
- Evolução: Callbacks → Promises.
- Fetch API para requisições de rede.
- Async/Await para código limpo e legível.