RESUMO DA AULA

12 Async e Await

O async e o await s o funcionalidades do JavaScript que vieram para facilitar ainda mais o trabalho com Promises, tornando o c digo ass ncrono mais intuitivo e leg vel. Antes de aprender sobre eles, utiliz vamos o m todo then() para encadear as chamadas de Promises, o que j resolvia muitos problemas de legibilidade e evitava o famoso callback hell. Por m, com async/await, o c digo se torna ainda mais direto, como se estiv ssemos trabalhando com fun es s ncronas. O que async? A palavra-chave async utilizada para declarar que uma fun o vai trabalhar de forma ass ncrona. Sempre que usamos async antes de uma fun o, garantimos que ela retorna uma Promise, mesmo que internamente a fun o utilize apenas c digo s ncrono. O que await? O await utilizado para esperar o resultado de uma Promise. Em vez de utilizar o then para lidar com o retorno da Promise, usamos await para "pausar" a execu o at que a Promise seja resolvida ou rejeitada. Vale lembrar que await s pode ser usado dentro de fun es que tenham sido declaradas com async. Vamos entender na pr tica como isso funciona utilizando um exemplo que j vimos anteriormente: Refatorando com async/await C digo original com .then(): let ferverAgua = (chaleiraEstaNoFogao, fogaoEstaLigado) => { return new Promise((resolve, reject) => { if (chaleiraEstaNoFogao && fogaoEstaLigado) { console.log('Passo 1 finalizado: gua foi fervida'); resolve(); } else { reject('Erro: necess rio colocar a chaleira com gua e ligar o fog o.'); } }); } let passarOCafe = () => { return new Promise((resolve) => { console.log('Passo 2 finalizado: Caf foi passado'); resolve(); }); } let tomarCafe = () => { return new Promise((resolve) => { console.log('Passo 3 finalizado: Terminei de tomar o caf '); resolve(); }); } let lavarXicara = () => { return new Promise((resolve) => { console.log('Passo 4 finalizado: Terminei de lavar a x cara'); resolve(); }); } let chaleiraEstaNoFogao = true; let fogaoEstaLigado = true; ferverAgua(chaleiraEstaNoFogao, fogaoEstaLigado) .then(() => passarOCafe()) .then(() => tomarCafe()) .then(() =>

lavarXicara()) .then(() => console.log('Ritual do caf finalizado!')); Como podemos ver, o uso do .then() j facilita o fluxo, mas ainda fica um pouco verboso. Vamos transformar esse c digo para usar async/await. Refatorando com async/await: async function iniciarProcessoDeFazerCafe() { try { const chaleiraEstaNoFogao = true; const fogaoEstaLigado = true; await ferverAgua(chaleiraEstaNoFogao, fogaoEstaLigado); await passarOCafe(); await tomarCafe(); await lavarXicara(); console.log('Ritual do caf finalizado!'); } catch (erro) { console.error(erro); } } Explicando a refatora o Fun o iniciarProcessoDeFazerCafe: A fun o que vai executar todo o processo declarada como async. Isso permite que usemos await dentro dela. Uso de await: A cada chamada de fun o (ferverAgua, passarOCafe, etc.), usamos await para esperar que a Promise seja resolvida antes de continuar para a pr xima linha. Bloco try...catch: Como a fun o async retorna uma Promise, qualquer erro que ocorrer durante a execu o das fun es await ser capturado no bloco catch. Dessa forma, lidamos com erros de maneira elegante e sem a necessidade de usar .catch() em cada chamada. Vantagens do async/await Legibilidade: O fluxo do c digo fica mais parecido com um c digo s ncrono, facilitando a leitura e manuten o. Tratamento de erros: Podemos usar try...catch para tratar erros de maneira centralizada. Facilidade ao lidar com m ltiplas Promises: Em alguns casos, precisamos aguardar que v rias Promises sejam resolvidas antes de continuar. Com async/await, podemos usar Promise.all() e await para esperar todas as Promises ao mesmo tempo. Usando async/await em fun es ass ncronas Imagine que queremos realizar uma s rie de chamadas de API para buscar dados de usu rios. Podemos escrever o c digo utilizando async/await da seguinte forma: async function buscarDadosDeUsuarios() { try { const usuario = await fetch('https://jsonplaceholder.typicode.com/users/1'); const usuarioDados = await usuario.json(); console.log(`Usu rio: ${usuarioDados.name}`); } catch (erro) { console.error('Erro ao buscar os dados do usu rio:', erro); } } buscarDadosDeUsuarios(); Conclus o O async/await torna o c digo ass ncrono mais simples e pr ximo de um c digo s ncrono. Em compara o com o uso do .then() em Promises, ele facilita a leitura e o tratamento de erros, tornando o fluxo do programa mais intuitivo. Quando estiver trabalhando com Promises, prefira async/await sempre que poss vel. Na pr xima aula, vamos falar sobre como tratar erros no JavaScript, um tema

essencial para garantir a estabilidade do seu c digo.