RESUMO DA AULA

11 Then

No JavaScript, o uso de fun es ass ncronas e promessas (Promises) facilita a cria o de c digos mais organizados e leg veis. Uma das grandes vantagens das Promises a possibilidade de usar o m todo then para trabalhar com fluxos ass ncronos de maneira encadeada, evitando o famoso problema conhecido como callback hell. O que then? O then um m todo usado para lidar com uma Promise que foi resolvida. Ele permite que voc defina um conjunto de opera es que devem ser executadas ap s a conclus o bem-sucedida de uma Promise. Basicamente, o then pode ser traduzido como "ent o". Ou seja, ele indica que quando a opera o for conclu da com sucesso, ent o outra opera o deve ser executada. Por exemplo: let ferverAgua = (chaleiraEstaNoFogao, fogaoEstaLigado) => { return new Promise((resolve, reject) => { if (chaleiraEstaNoFogao && fogaoEstaLigado) { console.log('Come ando o processo de ferver a gua...'); resolve(" gua fervida"); } else { reject("Erro: necess rio colocar a chaleira com a gua e ligar o fog o."); } }); }; let chaleiraEstaNoFogao = true; let fogaoEstaLigado = true; ferverAgua(chaleiraEstaNoFogao, fogaoEstaLigado) .then((mensagem) => { console.log(mensagem); // " gua fervida" return "Passar o caf "; }) .then((proximoPasso) => { console.log(proximoPasso); // "Passar o caf " }) .catch((erro) => { console.error(erro); // Se a promise falhar, captura o erro aqui. }); No exemplo acima: O then chamado ap s a promessa ser resolvida (resolve). Isso significa que o c digo dentro dele s ser executado ap s a gua ser fervida. O then pode retornar um novo valor, que passado automaticamente para o pr ximo then. Caso a promessa falhe (se reject for chamado), o catch ser executado para tratar o erro. Resolvendo o Callback Hell Antes das Promises, era comum usar fun es de callback para tratar fluxos ass ncronos. Isso levava a uma estrutura de c digo aninhada conhecida como callback hell. No exemplo abaixo, vamos ver como o then pode ajudar a transformar o callback hell em um fluxo linear e mais leg vel. Exemplo de Callback Hell: function

colocarAguaPraFerver(callback) { console.log("Colocando gua para ferver..."); setTimeout(() => { console.log(" gua ferveu!"); callback(); }, 3000); } function prepararFiltro(callback) { console.log("Preparando o filtro..."); setTimeout(() => { console.log("Filtro pronto!"); callback(); }, 2000); } function passarOCafe(callback) { console.log("Passando o caf ..."); setTimeout(() => { console.log("Caf pronto!"); callback(); }, 1000); } // Exemplo de Callback Hell colocarAguaPraFerver(() => { prepararFiltro(() => { passarOCafe(() => { console.log("Tudo pronto para servir!"); }); }); }); O c digo acima funciona, mas rapidamente se torna dif cil de ler e manter. Agora vamos ver como o mesmo fluxo pode ser escrito com then usando Promises. Resolvendo com then Para usar then, precisamos que cada fun o retorne uma Promise. Isso nos permite encadear chamadas then e ter um fluxo de execu o linear e f cil de entender. function colocarAguaPraFerver() { return new Promise((resolve) => { console.log("Colocando gua para ferver..."); setTimeout(() => { console.log(" gua ferveu!"); resolve(); }, 3000); }); } function prepararFiltro() { return new Promise((resolve) => { console.log("Preparando o filtro..."); setTimeout(() => { console.log("Filtro pronto!"); resolve(); }, 2000); }); } function passarOCafe() { return new Promise((resolve) => { console.log("Passando o caf ..."); setTimeout(() => { console.log("Caf pronto!"); resolve(); }, 1000); }); } // Usando o then para encadear fun es colocarAguaPraFerver() .then(() => prepararFiltro()) .then(() => passarOCafe()) .then(() => { console.log("Tudo pronto para servir!"); }) .catch((erro) => { console.error("Erro durante o processo: " + erro); }); Como funciona o then? colocarAguaPraFerver(): A primeira Promise come a a ser executada. Quando o processo de ferver a gua conclu do, a fun o resolve chamada e o pr ximo then ativado. prepararFiltro(): Ap s a gua ferver, essa fun o executada para preparar o filtro de caf . passarOCafe(): Quando o filtro est pronto, passamos o caf . then(() => {...}): Quando todas as etapas anteriores s o conclu das com sucesso, exibimos a mensagem final. Tratamento de Erro (catch): Se qualquer uma das Promises falhar (chamar reject), o fluxo ser interrompido e o catch ser acionado para lidar com o erro. Encadeamento de then Uma das grandes vantagens do then que ele permite o encadeamento de

opera es. Cada then retorna uma nova Promise, o que permite criar fluxos complexos de maneira simples. new Promise((resolve) => { resolve(10); }) .then((valor) => { console.log("Valor inicial:", valor); // Valor inicial: 10 return valor * 2; }) .then((novoValor) => { console.log("Dobro do valor:", novoValor); // Dobro do valor: 20 return novoValor + 5; }) .then((resultadoFinal) => { console.log("Resultado final:", resultadoFinal); // Resultado final: 25 }); Conclus o O then um m todo poderoso e essencial ao trabalhar com Promises no JavaScript. Ele transforma o fluxo ass ncrono em um fluxo mais organizado e leg vel, tornando o c digo mais f cil de manter e compreender. Ele elimina o problema de pir mide do callback hell e melhora significativamente a forma como lidamos com fun es ass ncronas no JavaScript. Na pr xima aula, vamos aprender sobre async/await, que uma sintaxe mais moderna e intuitiva para trabalhar com Promises.
