Transformando Código Assíncrono com Async/Await em JavaScript

Por Escola Dnc

Introdução

Neste ebook, vamos explorar como transformar código JavaScript assíncrono baseado em Promises para uma estrutura mais legível e fácil de manter usando async/await. Abordaremos a refatoração de um sistema de processamento de pedidos online, destacando as principais etapas e benefícios dessa abordagem moderna de programação assíncrona.

Entendendo o Contexto

Antes de mergulharmos nas técnicas de refatoração, é importante entender o cenário que estamos trabalhando:

- Temos um sistema de processamento de pedidos online
- O fluxo inclui confirmação do pedido e processamento do pagamento
- Originalmente, o código utilizava Promises encadeadas com `.then()`
- O objetivo é transformar esse código para usar async/await

```
O uso de async/await torna o código assíncrono mais fácil de ler e entender, aproximando-o da estrutura de código síncrono tradicional.
```

Refatorando a Confirmação do Pedido

Transformando Promises em Funções Assíncronas

O primeiro passo na nossa refatoração é transformar a Promise de confirmação do pedido em uma função assíncrona:

```
async function confirmarPedido() {  // Lógica de confirmação do pedido  return new Promise((resolve, reject) => {    // Implementação da confirmação  });}
```

Pontos importantes:

- A função `confirmarPedido` é declarada como `async`
- Ela retorna uma nova Promise
- Isso encapsula a lógica de confirmação do pedido

Benefícios da Abordagem

1. **Melhor legibilidade:** A função assíncrona torna mais claro o propósito do código
2. **Encapsulamento:** A lógica de confirmação fica isolada em sua própria função
3. **Reutilização:** A função pode ser facilmente chamada em diferentes partes do código

Processando o Pagamento

Após a confirmação do pedido, o próximo passo é processar o pagamento. Vamos refatorar essa parte do código:

Criando a Função de Processamento de Pagamento

```
async function processarPagamento(pedido) {  // Lógica de processamento do pagamento  return new Promise((resolve, reject) => {    // Implementação do processamento  });}
```

Observações:

- Assim como `confirmarPedido`, `processarPagamento` é uma função assíncrona
- Ela recebe o pedido como parâmetro
- Retorna uma Promise que resolve ou rejeita baseado no resultado do processamento

A Importância do Async

Embora não seja estritamente necessário usar `async` quando uma função já retorna uma Promise, mantê-lo traz benefícios:

1. **Clareza:** Torna explícito que a função é assíncrona
2. **Consistência:** Mantém um padrão uniforme no código
3. **Flexibilidade:** Permite fácil adição de lógica assíncrona adicional no futuro

```
Usar 'async' em funções que retornam Promises é uma boa prática
para manter o código consistente e fácil de entender.
```

Integrando as Funções Assíncronas

Com nossas funções assíncronas definidas, podemos agora integrar tudo em um fluxo coeso:

Estruturando o Fluxo Principal

```
async function processarPedidoOnline() {  try {    const
pedidoConfirmado = await confirmarPedido();    console.log("Pedido
confirmado com sucesso");        const pagamentoProcessado = await
processarPagamento(pedidoConfirmado);    console.log("Pagamento
aprovado");        console.log("Tudo deu certo no seu pedido,
aguardando envio");  } catch (erro) {    console.error("Ocorreu um
erro:", erro);  }}
```

Análise do código:

- Usamos `await` para esperar a resolução de cada etapa assíncrona
- O código fica sequencial e fácil de seguir
- Tratamento de erros é simplificado com o bloco `try/catch`

Vantagens dessa Abordagem

1. **Fluxo linear:** O código se parece mais com código síncrono tradicional
2. **Facilidade de leitura:** A sequência de operações é clara e direta

3. **Melhor tratamento de erros:** Um único bloco `try/catch` pode lidar com erros de múltiplas operações assíncronas

Testando e Verificando o Resultado

Após a refatoração, é crucial testar o código para garantir que tudo funcione como esperado:

Executando o Código

Ao executar o código refatorado, devemos ver uma saída semelhante a esta:

1. "Iniciando o pedido online"
2. "Pedido foi confirmado com sucesso"
3. "Aguardando o pagamento ser aprovado pelo cartão"
4. "O pagamento foi aprovado"
5. "Tudo deu certo no seu pedido, aguardando envio"

Verificação de Funcionamento

- Cada etapa do processo é claramente indicada no console
- A sequência lógica é mantida: confirmação do pedido, processamento do pagamento, conclusão
- Mensagens informativas guiam o usuário através do processo

```
A saída do console fornece uma visão clara do fluxo do pedido,
tornando mais fácil identificar em que etapa o processo está.
```

Conclusão

Neste ebook, exploramos como transformar código assíncrono baseado em Promises para uma estrutura utilizando async/await em JavaScript. Vimos que esta abordagem oferece várias vantagens:

1. **Maior legibilidade:** O código se torna mais fácil de entender e manter
2. **Fluxo mais natural:** A estrutura se aproxima do código síncrono tradicional
3. **Melhor tratamento de erros:** Simplifica o manejo de exceções em operações assíncronas
4. **Modularidade:** Funções assíncronas podem ser facilmente reutilizadas e testadas

Ao adotar async/await em seus projetos JavaScript, você pode criar código mais limpo, mais fácil de debugar e mais fácil de expandir no futuro. Esta técnica é especialmente útil em cenários complexos de programação assíncrona, como no processamento de pedidos online que exploramos aqui.

Lembre-se sempre de testar adequadamente seu código refatorado para garantir que todas as funcionalidades originais sejam preservadas. Com prática e experiência, você se tornará mais proficiente no uso de async/await, melhorando significativamente a qualidade e manutenibilidade de seu código JavaScript.