Criando um Sistema de Pedidos Online com Promises em JavaScript

Por Escola Dnc

Introdução

Neste ebook, vamos explorar como criar um sistema de pedidos online utilizando Promises em JavaScript. O objetivo é simular um processo de compra, desde a confirmação do pedido até o pagamento, utilizando programação assíncrona. Este conteúdo é baseado em uma aula prática que resolve um exercício proposto anteriormente.

Estrutura Básica do Sistema de Pedidos

O sistema de pedidos online que vamos criar consiste em duas partes principais:

1. Confirmação do pedido
2. Processamento do pagamento

Cada uma dessas etapas será implementada como uma Promise separada, permitindo que o fluxo de execução seja controlado de forma assíncrona.

Confirmação do Pedido

A primeira etapa do nosso sistema é a confirmação do pedido. Vamos criar uma Promise que simula este processo:

```
const promessaDePedidoConfirmado = new Promise((resolve, reject) => {  console.log("Iniciando o pedido online");  setTimeout(() => {  const sucesso = true;    if (sucesso) {      console.log("Pedido confirmado com sucesso");      resolve({ confirmation: true, payment: "awaiting" });    } else {      reject("O pedido não pode ser confirmado. Tente novamente.");    }  }, 5000);});
```

Pontos importantes:

- Utilizamos `setTimeout` para simular um atraso de 5 segundos no processamento do pedido.
- A Promise resolve com um objeto contendo informações sobre a confirmação e o status do pagamento.
- Em caso de falha, a Promise é rejeitada com uma mensagem de erro.

Processamento do Pagamento

A segunda etapa é o processamento do pagamento. Esta função retorna uma nova Promise que simula a aprovação do pagamento:

```
const promessaDePagamento = (pedido) => {  return new
Promise((resolve, reject) => {    console.log("Aguardando pagamento
ser aprovado pelo cartão");    setTimeout(() => {      const
pagamentoAprovado = true;      if (pagamentoAprovado) {
console.log("Pagamento aprovado com sucesso");        resolve({
confirmation: true, payment: "approved" });      } else {
reject("Ocorreu um erro ao realizar o pagamento. Pedido
cancelado.");      }    }, 5000);  });};
```

Observações:

- Esta função aceita um parâmetro `pedido`, que pode conter informações relevantes como ID do pedido.
- Novamente, usamos `setTimeout` para simular o tempo de processamento do pagamento.
- O resultado é similar à Promise de confirmação do pedido, mas com o status de pagamento atualizado.

Encadeamento de Promises

Para executar nosso fluxo de pedido completo, vamos encadear as Promises usando o método `.then()`:

```
promessaDePedidoConfirmado  .then((pedido) => {    return promessaDePagamento(pedido);  })  .then((pagamento) => {  console.log(pagamento);    console.log("Sucesso ao realizar o pagamento. Aguardando envio do pedido.");  })  .catch((erro) => {  console.error(erro);  });
```

Explicação do fluxo:

1. Iniciamos com a Promise de confirmação do pedido.
2. Se o pedido for confirmado, passamos para a Promise de pagamento.
3. Após o pagamento ser aprovado, logamos o resultado e uma mensagem de sucesso.
4. Se ocorrer algum erro em qualquer etapa, ele será capturado pelo `.catch()`.

Assincronicidade e Ordem de Execução

Um ponto importante a ser observado é a natureza assíncrona das Promises. Veja o exemplo:

```
promessaDePedidoConfirmado  .then((pedido) => {    return promessaDePagamento(pedido);  })  .then((pagamento) => {
console.log("Pagamento aprovado. Aguardando envio do pedido.");
});console.log("Promessa resolvida");
```

Resultado esperado:

1. "Iniciando o pedido online"
2. "Promessa resolvida"
3. "Pedido confirmado com sucesso"
4. "Aguardando pagamento ser aprovado pelo cartão"
5. "Pagamento aprovado com sucesso"
6. "Pagamento aprovado. Aguardando envio do pedido."

Note que "Promessa resolvida" é logado antes da conclusão do processo. Isso ocorre porque o código fora da cadeia de Promises é executado imediatamente, enquanto as operações assíncronas continuam em segundo plano.

Boas Práticas e Considerações

1. **Tratamento de Erros**: Sempre inclua blocos `.catch()` para lidar com possíveis erros em suas Promises.
2. **Simulação vs. Ambiente Real**: No exemplo, usamos `setTimeout` para simular atrasos. Em um ambiente de produção, essas operações seriam substituídas por chamadas reais a APIs ou bancos de dados.
3. **Organização do Código**: Mantenha suas Promises e funções assíncronas bem organizadas e modulares para facilitar a manutenção e teste.
4. **Logging**: Use `console.log()` estrategicamente para depurar e entender o fluxo de execução do seu código assíncrono.
5. **Parâmetros e Retornos**: Seja consistente com os objetos que você passa e retorna em suas Promises para manter a clareza do código.

Conclusão

Neste ebook, exploramos como criar um sistema básico de pedidos online usando Promises em JavaScript. Abordamos a criação de Promises para simular a confirmação de pedidos e processamento de pagamentos, bem como o encadeamento dessas Promises para criar um fluxo completo.

Compreender o funcionamento das Promises e a natureza assíncrona do JavaScript é fundamental para desenvolver aplicações web modernas e eficientes. Com este conhecimento, você está melhor preparado para lidar com operações assíncronas em seus projetos futuros.

Lembre-se de praticar estes conceitos e explorar mais sobre programação assíncrona em JavaScript para aprimorar suas habilidades de desenvolvimento.