Documentação de Arquitetura de Software - Sistema de E-commerce

1. Visão Geral do Sistema

O sistema de e-commerce permite que os usuários façam compras de produtos online. O sistema é composto por várias partes: a interface do cliente (web e mobile), a API backend que gerencia os pedidos e o catálogo de produtos, e um sistema de pagamento que se integra a terceiros para processar transações.

O objetivo principal do sistema é fornecer uma experiência fluida de compra para os usuários e garantir que os pedidos sejam gerenciados de maneira eficaz, desde o momento da compra até a entrega.

Principais funcionalidades:

- Navegação no catálogo de produtos.
- Carrinho de compras e gerenciamento de pedidos.
- Pagamentos online.
- Gerenciamento de estoque.
- Integração com transportadoras para envio.

2. Componentes do Sistema

O sistema de e-commerce é dividido nos seguintes componentes principais:

1. **Frontend (Aplicações Web e Mobile)**
    - Interface gráfica usada pelos clientes para navegar, pesquisar produtos, adicionar ao carrinho e fazer pedidos.
    - Utiliza frameworks JavaScript (React para Web, React Native para Mobile).
    - Comunicação com o backend via API REST.
2. **Backend (API de E-commerce)**
    - API RESTful que fornece as funcionalidades principais de gerenciamento de pedidos, controle de estoque, usuários e pagamentos.
    - Desenvolvido em Node.js com Express.js e conectado a um banco de dados relacional (MySQL).
3. **Banco de Dados (MySQL)**
    - Armazena dados relacionados a produtos, pedidos, usuários e estoque.
    - Estrutura relacional com tabelas para produtos, pedidos, clientes e transações.
4. **Sistema de Pagamento (Terceiros)**
    - Integração com um serviço externo (como PayPal ou Stripe) para processar pagamentos online.
    - O backend realiza a comunicação com o gateway de pagamento para gerenciar transações.
5. **Serviço de Envio**

    - Integração com serviços de transporte (como FedEx ou Correios) para gerar etiquetas de envio e rastrear entregas.
6. **Painel de Administração**
    - Interface administrativa para funcionários gerenciarem produtos, pedidos e clientes.
    - Acesso restrito por autenticação e controle de permissões.

3. Interações entre os Componentes

Aqui está um exemplo de como os componentes interagem em um fluxo típico de compra de um produto:

1. O cliente seleciona um produto no **frontend** e adiciona ao carrinho de compras.
2. O frontend envia o pedido via API para o **backend**, que processa a requisição e verifica a disponibilidade do produto no **banco de dados**.
3. Se o produto estiver disponível, o backend cria um pedido e solicita o pagamento via **sistema de pagamento** (por exemplo, PayPal).
4. O sistema de pagamento confirma a transação e envia a resposta ao backend.
5. Após a confirmação, o backend reduz o estoque no banco de dados e atualiza o status do pedido.
6. Finalmente, o backend solicita a geração de uma etiqueta de envio ao **serviço de envio**, que retorna o código de rastreamento da encomenda.

Diagrama de Interação (Simplificado):

Diagramas 1….
Diagramas 2…
Diagramas 3…

4. Decisões Arquiteturais (ADR)

4.1. Uso de Microsserviços vs. Monolito

- **Decisão:** Adotar uma arquitetura monolítica.
- **Justificativa:** O sistema está em seus estágios iniciais de desenvolvimento, e a equipe é pequena. Adotar uma arquitetura monolítica permitirá maior velocidade de desenvolvimento e integração inicial. Microsserviços podem ser considerados em fases posteriores para escalar partes específicas, como o processamento de pedidos e pagamentos.
- **Alternativa Rejeitada:** Arquitetura baseada em microsserviços. A complexidade adicional de gerenciar microsserviços (orquestração, comunicação, deploy) foi considerada desnecessária no momento.

4.2. Banco de Dados Relacional (MySQL)

- **Decisão:** Utilizar MySQL como banco de dados relacional.

- **Justificativa:** MySQL é uma solução madura e amplamente utilizada que se adapta bem às necessidades de um e-commerce com transações ACID. A equipe já tem experiência com MySQL e a consistência das transações é um fator importante.
- **Alternativa Rejeitada:** Bancos de dados NoSQL como MongoDB foram considerados, mas a estrutura relacional oferece melhor suporte para transações e consistência.

4.3. Framework de Frontend (React)

- **Decisão:** Utilizar React para o frontend web e React Native para o mobile.
- **Justificativa:** O React oferece uma boa combinação de performance, reusabilidade de componentes e comunidade ativa. React Native permite o compartilhamento de código entre as versões web e mobile, otimizando o desenvolvimento.
- **Alternativa Rejeitada:** Angular foi considerado, mas a curva de aprendizado seria mais íngreme para a equipe atual, que já tem familiaridade com React.

5. Ambiente de Implantação

O sistema de e-commerce será implantado em uma infraestrutura na nuvem utilizando contêineres Docker para facilitar o gerenciamento e a escalabilidade.

- **Servidor de Aplicação:** O backend e frontend serão hospedados em contêineres Docker em uma plataforma de nuvem (como AWS ou Azure).
- **Banco de Dados:** Um servidor MySQL gerenciado na nuvem será utilizado.
- **Sistema de Pagamento e Envio:** As integrações com PayPal e transportadoras são feitas via API externa e estão fora do ambiente de implantação.
- **Balanceamento de Carga:** Será implementado com Nginx ou um serviço gerenciado da nuvem (como AWS Elastic Load Balancer) para distribuir as requisições e garantir alta disponibilidade.

6. Requisitos e Restrições

6.1. Requisitos Funcionais:

- O sistema deve permitir que usuários façam login, naveguem em produtos e realizem compras.
- O sistema deve processar pagamentos via PayPal e emitir uma nota fiscal eletrônica.
- O sistema deve enviar notificações por e-mail para confirmar a compra e fornecer o código de rastreamento.

6.2. Requisitos Não Funcionais:

- O sistema deve suportar até 10.000 usuários simultâneos sem perda de performance.
- O tempo de resposta para consultas ao catálogo de produtos deve ser inferior a 2 segundos.
- O sistema deve ser compatível com navegadores modernos e dispositivos móveis.

6.3. Restrições Técnicas:

- A aplicação deve ser desenvolvida utilizando Node.js no backend e React no frontend.
- O sistema de pagamento deve ser compatível com PayPal e outros gateways suportados no futuro.

Esse é um exemplo básico de como a documentação arquitetural pode ser estruturada para um projeto de e-commerce. A profundidade e os detalhes podem variar conforme o tamanho e a complexidade do projeto, mas o formato geral é útil para garantir que todos os aspectos do sistema sejam bem compreendidos.