PedidoSync

Autor:

Gabriel Pereira Costa - RM 359027

Introdução

Este documento descreve as atividades, tecnologias e arquitetura envolvidas no desenvolvimento do projeto da equipe YnovaTI, criado no contexto do Tech Challenge. O objetivo principal é construir um Sistema de Gerenciamento de Pedidos Online, utilizando Desenvolvimento Orientado ao Domínio (DDD) aliado à arquitetura de microsserviços com Spring.

A solução visa atender às necessidades de sistemas modernos de pedidos, abordando desde o cadastro e controle de clientes e produtos até o processamento de pagamentos e controle de estoque. O sistema será construído para garantir alta coesão, baixa dependência entre serviços, persistência de dados isolada, e comunicação eficaz entre os microsserviços.

A aplicação será composta por módulos autônomos, cada um responsável por uma parte crítica do fluxo de pedidos, e será projetada para facilitar a escalabilidade, manutenibilidade e adaptabilidade, aplicando boas práticas de desenvolvimento backend.

A solução permitirá à empresa gerenciar pedidos de maneira eficiente, com funcionalidades como:

- **Cadastro de clientes**, com controle de dados pessoais e validação de CPF único;
- **Cadastro e manutenção de produtos**, incluindo nome, SKU e preço, com validação de duplicidade de SKU;
- **Gerenciamento de estoque**, com controle de quantidade disponível por produto;
- **Recebimento e processamento de pedidos online**, com cálculo automático do valor total e aplicação de regras de negócio;
- **Integração preparada com sistemas de pagamento online (via adaptadores mock)**, com suporte a simulação de falhas e estornos;
- **Atualização automática do status dos pedidos**, com base no sucesso ou falha do pagamento e disponibilidade de estoque;

- **Envio de notificações ou confirmações de pedido (simulado)**, para garantir comunicação eficiente com o cliente;
- **Geração de relatórios e análises**, relacionados a volume de pedidos, status e movimentação de estoque;
- **Sistema escalável e modular**, com foco em isolamento de domínios e serviços independentes.

Definição do Problema

A gestão de pedidos em ambientes digitais pode ser complexa e desafiadora devido a diversos fatores operacionais e técnicos. Os principais problemas enfrentados atualmente incluem:

Gerenciamento ineficiente de pedidos: A falta de integração entre clientes, produtos, estoque e pagamento pode gerar erros no processamento de pedidos, atrasos na confirmação e falhas na entrega da informação ao cliente.

Falta de visibilidade em tempo real: Sem uma arquitetura bem estruturada, é difícil acompanhar o status dos pedidos, disponibilidade de produtos ou falhas no pagamento de forma centralizada e atualizada, o que compromete a tomada de decisões.

Comunicação ineficaz entre sistemas: A ausência de uma comunicação clara e padronizada entre os serviços de cliente, estoque, pagamento e pedido pode causar inconsistências, como baixa de estoque incorreta ou pedidos aprovados sem confirmação de pagamento.

Problemas durante picos de demanda: Em períodos de alta carga, como promoções ou sazonalidades, sistemas monolíticos ou mal dimensionados podem apresentar lentidão, falhas de disponibilidade ou processamento incorreto de pedidos.

Experiência do cliente prejudicada: Clientes podem não receber atualizações claras sobre o status dos pedidos (ex: sucesso, falha por estoque ou pagamento), gerando frustração, dúvidas e perda de confiança na plataforma.

Requisitos Funcionais

1. **Cadastro de Clientes**: O sistema deve permitir que usuários sejam cadastrados como clientes, fornecendo dados como nome completo, CPF (único), data de nascimento, e endereços (rua, número, CEP, cidade, estado). O CPF não pode ser duplicado no sistema.

2. **Cadastro de Produtos:**
   O sistema deve permitir o cadastro de produtos, incluindo informações como nome do produto, SKU (único), e preço. A duplicidade de SKU deve ser evitada.

3. **Gerenciamento de Estoque:**

   O sistema deve controlar a quantidade disponível de cada produto no estoque. Sempre que um pedido for feito, a quantidade correspondente deverá ser atualizada automaticamente.

4. **Recebimento e Processamento de Pedidos:**

   O sistema deve permitir que clientes realizem pedidos, informando os SKUs dos produtos desejados, a quantidade de cada item, e os dados do pagamento (como número do cartão de crédito). O backend deve calcular automaticamente o valor total do pedido.

5. **Atualização de Status dos Pedidos:**

   O sistema deve definir e atualizar o status do pedido com base no resultado do processamento:

   “ABERTO” ao receber o pedido;

   “FECHADO_COM_SUCESSO” se o estoque estiver disponível e o pagamento for aprovado;

   “FECHADO_SEM_ESTOQUE” se não houver estoque suficiente (com estorno do pagamento, se feito);

   “FECHADO_SEM_CREDITO” se o pagamento for recusado (com retorno do estoque).

6. **Fila de Processamento Assíncrona:**

   Todos os pedidos recebidos devem ser salvos inicialmente em uma fila, permitindo seu processamento posterior de forma assíncrona, garantindo escalabilidade e tolerância a falhas.

7. **Integração com Serviço de Pagamento (Simulado):**

   O sistema deve simular a integração com um serviço externo de pagamento por meio de um adaptador mock, respeitando os princípios da arquitetura limpa para permitir substituição futura por uma integração real.

8. **Consulta de Pedidos:**

   Os clientes devem poder consultar o status e o histórico de seus pedidos por meio de uma interface web ou API.

9. **Notificações Simuladas (Opcional):**

   O sistema pode incluir o envio de notificações (mockadas) sobre o status do pedido, como confirmações ou falhas, via e-mail ou outro canal.

10. **Relatórios e Análises:**

- O sistema deve gerar relatórios com métricas de pedidos, como:
- Quantidade total de pedidos;
- Volume de produtos vendidos;
- Taxas de sucesso/falha no processamento;
- Histórico de movimentação de estoque.

Requisitos Não Funcionais

1. **Escalabilidade**: O sistema deve ser capaz de lidar com picos de demanda, como em lançamentos de produtos, promoções ou grandes volumes simultâneos de pedidos, sem comprometer a performance ou estabilidade dos serviços.
2. **Segurança**: As informações sensíveis, como dados pessoais dos clientes e informações de pagamento, devem ser armazenadas e transmitidas de forma segura, utilizando criptografia, autenticação adequada e conformidade com boas práticas de proteção de dados (como LGPD).
3. **Confiabilidade**: O sistema deve garantir alta disponibilidade, tolerância a falhas e consistência nos dados, assegurando que os pedidos sejam corretamente registrados, processados e que os clientes recebam feedback sobre o status de seus pedidos.
4. **Usabilidade** : A aplicação deve ser intuitiva e acessível, com uma interface clara tanto para os usuários finais (clientes) quanto para os operadores do sistema (administração), reduzindo a curva de aprendizado.
5. **Desempenho**: O sistema deve garantir tempos de resposta rápidos para as principais operações, como:

- Criação de pedidos;
- Consulta de status;
- Atualização de estoque;
- Processamento de pagamentos.

Requisitos Técnicos

1. **Banco de Dados** : Cada microsserviço deve utilizar um **banco de dados independente**, de acordo com sua necessidade. NoSQL MongoDB.
2. **Integração com API de Pagamento**: O sistema deve incluir um adaptador para integração com uma API de pagamento (ex: PayPal, Mercado Pago), simulando o processamento via um mock. Essa integração deverá ser facilmente substituível no futuro, conforme os princípios da arquitetura limpa.
3. **Interface Web (Responsiva):** : A solução deverá incluir uma **interface web responsiva**, acessível por navegadores modernos em dispositivos desktop e mobile, permitindo que os clientes façam pedidos, consultem status e visualizem informações relevantes.
4. **Armazenamento em Nuvem**: A aplicação poderá ser hospedada em serviços em nuvem como AWS, Azure ou Google Cloud, aproveitando recursos como escalabilidade automática, persistência, backups e balanceamento de carga.
5. **Monitoramento e Logs** : O sistema deverá implementar ferramentas de monitoramento em tempo real e geração de logs estruturados, possibilitando a identificação de falhas, análise de desempenho e rastreamento de pedidos. Sugestões incluem: Prometheus + Grafana, ELK Stack ou Loki.
6. **Orquestração com Docker Compose:** Todo o sistema deverá ser conteinerizado com Docker, e a orquestração dos serviços feita via Docker Compose, facilitando a implantação, testes e distribuição do projeto.
7. **Testes Automatizados com Cobertura:** O sistema deve garantir pelo menos 80% de cobertura de testes automatizados usando Jacoco, abrangendo os principais fluxos e validando regras críticas de negócio**.**

Solução Proposta com DDD

A solução será desenvolvida utilizando o **DDD (Desenvolvimento Orientado ao Domínio)**, promovendo uma separação clara entre os diferentes contextos de negócio, garantindo uma estrutura modular, de fácil manutenção e escalável. Os seguintes **domínios de negócio** foram identificados e serão modelados:

- **Domínio de Cadastro**:

  Responsável pelo gerenciamento das entidades fundamentais do sistema :

- Cadastro de **clientes**, com informações como nome, CPF, data de nascimento e endereços;
- Cadastro de **produtos**, com nome, SKU e preço, garantindo unicidade de SKUs;
- Estruturação das entidades básicas para uso nos demais domínios do sistema.

- **Domínio de Pedido** :

  Responsável por todo o ciclo de vida de um pedido:

- Criação de novos pedidos, com seleção de produtos e quantidades;
- Validação de estoque e cálculo automático do valor total;
- Atualização de status de pedido conforme o fluxo: **aberto**, **fechado com sucesso**, **fechado sem estoque**, ou **fechado sem crédito**;
- Consulta do histórico e status dos pedidos realizados por um cliente.

- **Domínio de Estoque:**

  Responsável pela gestão de inventário dos produtos:

- Controle de quantidades disponíveis;
- Redução de estoque ao registrar novos pedidos;
- Reversão de estoque em caso de falha de pagamento ou cancelamento;
- Interface com o domínio de pedidos via eventos e mensageira.

- **Domínio de Pagamento :**

  Responsável pelo fluxo de pagamento e seu processamento:

- Integração com um serviço de pagamento externo (mockado), para fins de simulação;
- Tratamento de respostas do serviço de pagamento, com possibilidade de simular sucesso ou falha;
- Estorno de valores quando necessário;
- Atualização do pedido com o resultado do pagamento.

- **Domínio de Relatórios e Avaliação :**

Responsável por consolidar dados operacionais e oferecer insights:

- Geração de **relatórios de desempenho** do sistema: volume de pedidos, produtos mais vendidos, falhas por estoque ou crédito, entre outros;
- Coleta de **feedback do cliente (simulado)** após finalização do pedido, com possibilidade de registrar avaliações ou sugestões;
- Geração de **indicadores para apoio à decisão**, contribuindo para a melhoria contínua do sistema e dos processos internos.

- **Domínio de Experiência**: Este domínio é responsável por garantir uma experiência satisfatória e transparente para o cliente durante e após o processo de realização do pedido. As funcionalidades incluem.
    - Acompanhamento em tempo real do status do pedido por meio da interface web;
    - Envio de notificações simuladas (via e-mail ou outros canais mockados) sobre o andamento do pedido — como confirmação, processamento, finalização ou falha;
    - Registro de interações relacionadas à jornada do cliente com o sistema de pedidos;
    - Disponibilização de histórico de pedidos e seus detalhes (itens, datas, valores, status).
    -
- **Domínio de Avaliação e Feedback** O domínio de Avaliação tem como objetivo coletar e analisar o feedback dos clientes sobre suas experiências com o sistema de pedidos. As funcionalidades envolvem:
- Envio automático de **solicitações de avaliação** após a conclusão de um pedido.
- Coleta de **comentários, sugestões e classificações** feitas pelos usuários.
- **Armazenamento estruturado das avaliações**, permitindo análise por período, cliente ou tipo de problema;
- Geração de **relatórios e indicadores** com base nas avaliações, auxiliando os gestores na **tomada de decisões estratégicas** e na **melhoria contínua do sistema**;
- Possibilidade de correlacionar feedbacks com dados operacionais (por exemplo, frequência de falhas por estoque ou pagamento).

Event Storming no Sistema de Gestão de Restaurantes

O processo de Event Storming foi realizado por meio de sessões colaborativas envolvendo os principais stakeholders: gestores, funcionários e clientes. O objetivo foi mapear as interações e fluxos de informação entre os atores e o sistema, identificando eventos significativos, comandos e agregados para uma modelagem precisa do domínio**.**

Principais Eventos e Comandos Identificados

1. Cadastro e Gerenciamento

O sistema deve permitir **gerenciar** (cadastrar, editar, excluir):

- **Clientes**:
    - Dados obrigatórios: nome, CPF (único), data de nascimento, endereço completo (rua, número, CEP etc.).
    - Não é permitido duplicar CPF no sistema.
- **Produtos**:

        - Dados obrigatórios: nome, SKU (identificador único), preço.
        - Não é permitido duplicar SKU no sistema.
- **Estoque**:
    - Cada produto deve ter um controle de **quantidade disponível**.
    - A movimentação de estoque será gerenciada com base nos pedidos recebidos.

2. **Processamento de Pedidos**

O sistema deve permitir o **recebimento e processamento de pedidos de clientes**.

Dados obrigatórios para solicitação de pedido:

- Identificador do cliente (CPF ou ID).
- Lista de produtos (SKU e quantidade de cada item).
- Dados de pagamento:
    - Número do cartão de crédito (simulado).

Regras de negócio:

- Ao receber um pedido:
    - O status inicial do pedido será `ABERTO`.
    - Deve ser feita:
        - **Baixa de estoque** com base na quantidade de cada produto.
        - **Simulação do pagamento** via adaptador externo (mock).
        - **Cálculo do valor total do pedido** no backend.

Possíveis fluxos e status finais:

1. **Sucesso Completo**:
    - Pedido processado, pagamento aprovado, estoque ajustado.
    - Status final: `FECHADO_COM_SUCESSO`.
2. **Sem Estoque**:
    - Caso não haja quantidade suficiente de algum item:
        - Pagamento (se efetuado) deve ser estornado.
        - Status final: `FECHADO_SEM_ESTOQUE`.
3. **Falha no Pagamento**:
    - Caso o pagamento falhe (ex: saldo insuficiente):
        - Estoque deve ser **revertido** (quantidades restituídas).
        - Status final: `FECHADO_SEM_CREDITO`.

.

Identificação de Agregados

Foram identificados **quatro agregados principais**, cada um encapsulando um conjunto coeso de responsabilidades e dados:

1. **Cliente**
    - Responsável pelo gerenciamento de dados dos clientes (nome, CPF, data de nascimento, endereço).
    - Garante unicidade do CPF.
2. **Produto**
    - Gerencia as informações de produtos (nome, SKU, preço).
    - Garante unicidade do SKU.
    - Envolve controle direto com o estoque.
3. **Estoque**
    - Responsável por manter o controle de quantidade disponível de produtos.
    - Processa ajustes de quantidade a partir de pedidos e cancelamentos.
4. **Pedido**
    - Centraliza o processo de realização de pedidos.
    - Orquestra: verificação de estoque, cálculo de valor, requisição de pagamento, e atualização de status.
    - Coordena com os agregados Cliente, Produto, Estoque e Pagamento.

Fluxos de Informação

1. **Cadastro de Dados Iniciais**
    - Admin ou sistema realiza cadastro de clientes, produtos e estoque inicial.
2. **Solicitação de Pedido**
    - Cliente informa produtos desejados (por SKU), quantidades e dados de pagamento.
    - Pedido é colocado em uma **fila para processamento assíncrono**.
3. **Processamento do Pedido (Pedido Receiver)**
    - Consome da fila, consulta estoque e verifica disponibilidade.
    - Calcula o valor total.
    - Inicia o processo de pagamento com gateway externo (mockado).
    - Atualiza o status do pedido com base nos resultados.

4. **Baixa ou Reversão de Estoque**
    - Estoque é ajustado automaticamente:
        - Reduzido em caso de sucesso.
        - Restaurado em caso de falha no pagamento.
5. **Resposta ao Cliente**
    - O status final do pedido (`FECHADO_COM_SUCESSO`, `FECHADO_SEM_ESTOQUE`, `FECHADO_SEM_CREDITO`) é retornado via API REST.

Demandas e Necessidades por Perfil

Administrador / Gestor Técnico

- **Demandas**: Gerenciar cadastro de clientes e produtos, monitorar pedidos e estoque, garantir integridade do sistema.
- **Necessidades**: Dashboards de pedidos e estoque, alertas de baixa disponibilidade, relatórios de falha/sucesso em pedidos.

Cliente (Usuário Final)

- **Demandas**: Fazer pedidos de forma simples e confiável.
- **Necessidades**: Interface clara, retorno em tempo real do status do pedido, transparência sobre preços e disponibilidade.

Considerações Técnicas

Para garantir **escalabilidade**, **resiliência** e **modularidade**, o sistema foi projetado com as seguintes tecnologias e princípios:

- **Spring Boot (Java)**: Backend robusto com suporte nativo a APIs REST e integração com mensageria.
- **MongoDB / PostgreSQL**: Persistência de dados (NoSQL ou relacional, a critério da equipe).
- **Flyway**: Controle de versões do banco de dados (quando relacional).
- **OpenAPI / Swagger**: Documentação automática e interativa das APIs.
- **Docker + Docker Compose**: Containerização de todos os microsserviços e execução unificada.
- **Mensageria (ex: Kafka ou RabbitMQ)**: Processamento assíncrono dos pedidos.
- **Jacoco**: Cobertura de testes (mínimo de 80%).
- **Git e GitHub**: Versionamento e colaboração.
- **Maven**: Gerenciamento de dependências e build.

Ferramentas auxiliares: IntelliJ IDEA, Postman, Mongo Client, Miro (modelagem de eventos), Notion (documentação de equipe).

Conclusão

A abordagem baseada em **DDD (Domain-Driven Design)**, aliada ao uso de **Event Storming** e **Arquitetura Limpa**, permitiu uma clara separação de responsabilidades e excelente alinhamento com as regras de negócio. Com isso, o sistema garante:

- **Alto acoplamento interno e baixo acoplamento externo** por microsserviço.
- **Manutenibilidade e escalabilidade** para o crescimento da aplicação.
- **Facilidade de adaptação** para integrações reais com gateways de pagamento e sistemas de estoque.

Essa estrutura resulta em um sistema **robusto, confiável e preparado para produção em ambientes críticos de e-commerce** ou gestão de pedidos.