System Design Document - MercadoAPI

1. Visão Geral do Sistema

Descrição: O MercadoAPI é uma API REST desenvolvida para a gestão de pedidos e pagamentos, oferecendo um fluxo completo de compras online. O sistema inclui funcionalidades como autenticação segura, controle de estoque, processamento de pagamentos via Stripe e envio de e-mails para recuperação de senha.

A arquitetura foi projetada com foco em escalabilidade, segurança e facilidade de manutenção, seguindo boas práticas de desenvolvimento de software. O sistema permite que usuários realizem pedidos, gerenciem suas compras e acompanhem o status da entrega, enquanto administradores podem gerenciar o catálogo de produtos, acompanhar pagamentos e processar reembolsos.

URL da API em produção: MercadoAPI Railway

Principais funcionalidades:

- Autenticação e autorização com JWT.
- Gerenciamento de pedidos, desde a criação até a finalização.
- Controle de estoque, com reserva de produtos no carrinho e atualização automática.
- Processamento de pagamentos via Stripe, com suporte para webhooks.
- Envio de e-mails para redefinição de senha.
- Implantação automatizada utilizando Docker e Railway.

2. Tecnologias Utilizadas

O MercadoAPI utiliza um conjunto de tecnologias modernas para garantir segurança, eficiência e escalabilidade:

Linguagem e Framework:

- Java 17
- Spring Boot 3.4.2 (Web, Security, Data JPA, Validation)

Autenticação e Segurança:

- Spring Security com JWT
- Hash de senhas com BCrypt
- Autorização baseada em perfis de usuário (cliente e admin)

Banco de Dados:

- PostgreSQL (Docker/Railway) – Ambiente de produção
- H2 Database – Ambiente de testes

Pagamentos:

- Stripe Webhooks para processamento de pagamentos e reembolsos automáticos

Infraestrutura e Deploy:

- Docker (para banco de dados e ambiente de desenvolvimento)
- Railway (hospedagem da API)
- CI/CD automatizado com GitHub

Serviços Adicionais:

- Envio de e-mails via SMTP para recuperação de senha
- Tratamento centralizado de exceções com GlobalExceptionHandler

3. Arquitetura e Estrutura do Projeto

3.1 Visão Geral da Arquitetura

O MercadoAPI segue uma arquitetura baseada no padrão Camadas (Layered Architecture), separando as responsabilidades da aplicação de forma modular para garantir manutenção, escalabilidade e reutilização de código. As principais camadas do sistema são:

Controllers (Apresentação - API REST): Responsável por expor os endpoints e receber requisições HTTP.

Services (Regras de Negócio): Contém a lógica central da aplicação.

Repositories (Acesso a Dados): Gerencia a comunicação com o banco de dados.

Entities (Modelo de Dados): Representa as tabelas do banco de dados.

DTOs (Transferência de Dados): Define os objetos usados para entrada e saída de dados.

Exception (Tratamento de Erros): Centraliza a gestão de exceções para respostas padronizadas.

Config (Configurações Gerais): Inclui configurações como CORS e segurança.

3.2 Estrutura do Projeto

O projeto segue uma organização clara de pacotes, conforme detalhado abaixo:

Config

- WebConfig: Define configurações globais, como CORS e formatação de respostas JSON.

Entities (Modelos de Dados)

Enums:

- OrderStatus – Enumera os possíveis estados de um pedido.
- Role – Define os papéis dos usuários (Cliente/Admin).

Classes de Entidade:

- Address – Representa o endereço do usuário.
- Order – Modelo principal de um pedido.
- OrderItem – Itens pertencentes a um pedido.
- OrderStatusHistory – Histórico de status de um pedido.
- PasswordResetToken – Gerencia tokens para redefinição de senha.
- Product – Representa um produto do sistema.
- User – Modelo de usuário do sistema.

Repositories (Camada de Persistência)

Interfaces do Spring Data JPA para comunicação com o banco de dados:

- OrderRepository
- OrderItemRepository
- ProductRepository
- UserRepository
- PasswordResetTokenRepository

Services (Regras de Negócio)

- AuthService – Gerencia autenticação, registro e JWT.
- EmailService – Serviço para envio de e-mails.
- OrderService – Processa pedidos e histórico de status.
- PaymentService – Integração com Stripe para pagamentos.
- ProductService – Gerencia produtos e estoque.
- UserService – Operações relacionadas a usuários.

DTOs (Objetos de Transferência de Dados)

Pacote request (Requisições de Entrada)

- AddressDTO
- ForgotPasswordRequestDTO
- LoginRequestDTO
- OrderItemRequestDTO
- OrderStatusUpdateRequestDTO
- ProductDTO
- ResetPasswordRequestDTO
- UserRegistrationDTO

Pacote response (Respostas da API)

- LoginResponseDTO
- OrderDTO
- OrderItemResponseDTO
- OrderStatusHistoryDTO
- OrderSummaryDTO
- PaymentIntentResponse
- UserDTO

Controllers (Endpoints da API)

- AuthController – Gerencia autenticação e recuperação de senha.
- OrderController – Gerencia pedidos e status.
- PaymentController – Processa pagamentos via Stripe.
- ProductController – Gerencia produtos.
- UserController – Operações relacionadas a usuários.
- WebHookController – Captura eventos do Stripe.

Exception (Tratamento Centralizado de Erros)

- BusinessException – Exceções de regra de negócio.
- ErrorResponse – Modelo de resposta padronizada para erros.
- ForbiddenException – Erro de permissão insuficiente (403).
- OrderStatusException – Erro ao tentar mudar status de pedido.
- PaymentException – Erros de processamento de pagamento.
- ResourceNotFoundException – Erro para entidades não encontradas.
- StockException – Erro relacionado ao estoque de produtos.
- UnauthorizedException – Falha na autenticação (401).

4. Modelo de Dados

O modelo de dados do MercadoAPI foi estruturado para garantir a consistência das informações e a eficácia das operações do sistema. Abaixo estão os principais componentes do banco de dados e seus relacionamentos.

4.1. Entidades e Relacionamentos

Usuário (User)

A entidade User representa os clientes e administradores do sistema. Cada usuário tem um endereço vinculado e um papel no sistema.

ID: Identificador único do usuário.

Nome, E-mail e Senha: Dados essenciais para autenticação e contato.

CPF: Identificador único do usuário.

Papel (Role): Define o nível de acesso do usuário.

Endereço: Relacionamento @OneToOne com Address.

Endereço (Address)

A entidade Address armazena os dados de endereço de um usuário.

ID: Identificador único do endereço.

Rua, Número, Complemento, Bairro, Cidade, Estado, CEP: Dados de localização.

Usuário: Relacionamento @OneToOne com User.

Produto (Product)

A entidade Product representa os itens disponíveis para venda no sistema.

ID: Identificador único do produto.

Nome, Descrição, Categoria: Dados de identificação.

Preço: Valor do produto.

Quantidade em Estoque e Quantidade Reservada: Controle de disponibilidade.

Imagem: URL da imagem do produto.

Pedido (Order)

A entidade Order representa um pedido realizado por um usuário.

ID: Identificador único do pedido.

Usuário: Relacionamento @ManyToOne com User.

Data de Criação: Timestamp do pedido.

Código de Rastreamento: Identificador para acompanhar a entrega.

Itens do Pedido: Relacionamento @OneToMany com OrderItem.

Histórico de Status: Relacionamento @OneToMany com OrderStatusHistory.

ID da Intent de Pagamento e Reembolso: Integração com sistema de pagamentos.

Item do Pedido (OrderItem)

A entidade OrderItem representa um item dentro de um pedido.

ID: Identificador único do item.

Pedido: Relacionamento @ManyToOne com Order.

Produto: Relacionamento @ManyToOne com Product.

Quantidade: Quantidade do item no pedido.

Preço Unitário: Preço do produto no momento da compra.

Histórico de Status do Pedido (OrderStatusHistory)

A entidade OrderStatusHistory armazena as mudanças de status de um pedido ao longo do tempo.

ID: Identificador único do histórico.

Status: Status do pedido.

Data de Atualização: Timestamp da mudança.

Pedido: Relacionamento @ManyToOne com Order.

Token de Recuperação de Senha (PasswordResetToken)

A entidade PasswordResetToken gerencia os tokens de recuperação de senha.

ID: Identificador único do token.

Token: Chave única de recuperação.

Usuário: Relacionamento @OneToOne com User.

Data de Expiração: Controle de validade do token.

4.2. Relacionamentos do Modelo de Dados

User (1) --- (1) Address

User (1) --- (N) Order

Order (1) --- (N) OrderItem

OrderItem (N) --- (1) Product

Order (1) --- (N) OrderStatusHistory

User (1) --- (1) PasswordResetToken

Esse modelo permite que as informações sobre pedidos, estoque e pagamentos sejam bem gerenciadas, garantindo a integridade e a rastreabilidade das operações no sistema.

5. Fluxo de Dados e Operações

5.1. Fluxo de Pedido

1. **Criação do Carrinho:** O usuário inicia um pedido ao criar um carrinho (Order com status CART).
2. **Adição de Itens:** O usuário adiciona itens ao carrinho (OrderItem). O estoque é reservado.
3. **Checkout:** O usuário finaliza o carrinho, mudando o status para PENDING.
4. **Pagamento:** O usuário realiza o pagamento via Stripe. Quando confirmado, o status muda para PAID, e o estoque reservado é descontado do estoque real.
5. **Processamento e Entrega:** O pedido passa pelos status PROCESSING, OUT_FOR_DELIVERY e COMPLETED.
6. **Cancelamento:** O pedido pode ser cancelado em certos estados, restaurando o estoque conforme necessário.
7. **Reembolso:** Se cancelado após o pagamento, o valor é reembolsado via Stripe.

5.2. Fluxo de Autenticação e Controle de Acesso

- Registro: Usuários criam contas.
- Login: JWT é gerado e usado para autenticação nas requisições.
- Autorização: Acesso a endpoints é controlado por roles (USER, ADMIN).

5.3. Fluxo de Pagamento

Checkout: O usuário confirma os itens e inicia o pagamento.

Processamento: A API chama o serviço da Stripe.

Confirmação: Stripe confirma o pagamento, e o pedido é atualizado para PAID.

6. Exceções

O tratamento de exceções na MercadoAPI é feito de forma centralizada por meio da classe GlobalExceptionHandler, que usa @ControllerAdvice para capturar e padronizar as respostas de erro enviadas aos clientes da API. Esse mecanismo melhora a manutenção do código e garante que as respostas sejam consistentes.

6.1 Estrutura da Resposta de Erro

Todas as respostas de erro seguem o formato padronizado da classe ErrorResponse, que contém:

timestamp: *Data e hora do erro.*

status: *Código HTTP do erro.*

error: *Razão do erro (exemplo: "Bad Request").*

message: *Mensagem descritiva do erro.*

errorCode: *Código interno da aplicação para rastreamento do erro.*

path: *URI do endpoint onde o erro ocorreu.*

6.2 Exceções Tratadas

A seguir, estão as principais exceções tratadas no GlobalExceptionHandler:

ResourceNotFoundException (404 - Not Found):

Disparada quando um recurso (ex.: produto, pedido, usuário) não é encontrado no banco de dados.

BusinessException (400 - Bad Request):

Captura erros de regra de negócio, como tentativa de criar um pedido com informações inválidas.

ForbiddenException (403 - Forbidden):

Lançada quando um usuário tenta acessar um recurso sem permissão.

UnauthorizedException (401 - Unauthorized):

Usada para erros de autenticação, como JWT inválido ou credenciais incorretas.

OrderStatusException (409 - Conflict):

Ocorre quando há tentativa de mudar um pedido para um status inválido.

StockException (422 - Unprocessable Entity):

Lançada quando um pedido não pode ser finalizado devido à falta de estoque.

PaymentException (500 - Internal Server Error):

Captura erros internos relacionados ao processamento de pagamento.

MethodArgumentNotValidException (400 - Bad Request):

Captura erros de validação de dados, retornando mensagens detalhadas sobre os campos inválidos.

Exception (500 - Internal Server Error):

Tratamento genérico para exceções não previstas, garantindo que erros inesperados sejam capturados e registrados adequadamente.

Com esse tratamento estruturado, a API garante respostas claras e padronizadas, facilitando a depuração e melhorando a experiência do desenvolvedor.

7. Perfis de Ambiente (application.properties)

O MercadoAPI utiliza diferentes perfis de ambiente para gerenciar configurações distintas de acordo com o contexto de execução da aplicação. Os perfis são definidos no arquivo application.properties principal e suas variações são especificadas em arquivos individuais para cada ambiente.

7.1. Configuração Geral (application.properties)

O arquivo application.properties principal contém definições básicas compartilhadas entre todos os ambientes:

```
spring.application.name=mercadoapi

spring.profiles.active=${APP_PROFILE}

spring.jpa.open-in-view=false


cors.origins=${CORS_ORIGINS:http://localhost:5173,http://localhost:3000}
```

spring.application.name: Define o nome da aplicação.

spring.profiles.active: Define o perfil ativo da aplicação (test, dev, prod).

spring.jpa.open-in-view: Desabilitado para evitar problemas de performance.

cors.origins: Configuração de CORS para permitir requisições do frontend.

7.2. Ambiente de Teste (application-test.properties)

O ambiente de teste utiliza um banco de dados em memória H2 para testes automatizados.

```
# H2 Connection

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.username=sa

spring.datasource.password=


# H2 Client

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console
```

 # Show SQL

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format_sql=true

 # JWT

jwt.secret=your_secret_string

jwt.expiration=86400000

 # Reset Password URL

app.reset-password-base-url=https://localhost:8080/api/auth/reset-password

 # Email

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=seu_email@gmail.com

spring.mail.password=${EMAIL_APP_PASSWORD}

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

 # Pagamento (Stripe)

stripe.apiKey=${STRIPE_API_KEY}

stripe.webhook.secret=${STRIPE_WEBHOOK_SECRET}

7.3. Ambiente de Desenvolvimento (application-dev.properties)

No ambiente de desenvolvimento, utiliza-se um banco de dados PostgreSQL local.

spring.datasource.url=jdbc:postgresql://localhost:5433/mercadoapi

spring.datasource.username=postgres

spring.datasource.password=1234567

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

spring.jpa.hibernate.ddl-auto=none

 # JWT

jwt.secret=your_secret_string

jwt.expiration=${JWT_EXPIRATION:86400000}

 # Reset Password URL

app.reset-password-base-url=https://${APP_HOST}/api/auth/reset-password

 # Email

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=${EMAIL_APP_USERNAME}

spring.mail.password=${EMAIL_APP_PASSWORD}

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

 # Pagamento (Stripe)

stripe.apiKey=${STRIPE_API_KEY}

stripe.webhook.secret=${STRIPE_WEBHOOK_SECRET}

7.4. Ambiente de Produção (application-prod.properties)

No ambiente de produção, os valores sensíveis são armazenados em variáveis de ambiente.

spring.datasource.url=${DB_URL}

spring.datasource.username=${DB_USERNAME}

spring.datasource.password=${DB_PASSWORD}

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

spring.jpa.hibernate.ddl-auto=none

 # JWT

jwt.secret=${JWT_SECRET}

jwt.expiration=${JWT_EXPIRATION}

 # Reset Password URL

app.reset-password-base-url=https://${APP_HOST}/api/auth/reset-password

 # Email

spring.mail.host=${EMAIL_APP_HOST}

spring.mail.port=${EMAIL_APP_PORT}

spring.mail.username=${EMAIL_APP_USERNAME}

spring.mail.password=${EMAIL_APP_PASSWORD}

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

 # Pagamento (Stripe)

stripe.apiKey=${STRIPE_API_KEY}

stripe.webhook.secret=${STRIPE_WEBHOOK_SECRET}

7.5. Considerações Finais

Os perfis de ambiente garantem uma separação clara entre desenvolvimento, teste e produção, permitindo ajustes fáceis sem necessidade de modificar o código-fonte. Variáveis sensíveis (como senhas e chaves de API) são mantidas seguras por meio de variáveis de ambiente.

8. Configuração de Segurança e CORS

8.1. CORS (Cross-Origin Resource Sharing)

A API implementa uma configuração de CORS para permitir requisições vindas de origens específicas. O valor das origens permitidas é definido por meio da propriedade cors.origins no arquivo de configuração. A configuração permite todos os métodos HTTP e aplica-se a todos os endpoints da API.

8.2. Configuração de Segurança

A segurança do sistema é baseada no Spring Security e JWT (JSON Web Token), garantindo autenticação e autorização seguras para os usuários.

Autenticação e Filtros de Segurança

- A autenticação é baseada em JWT, onde cada usuário recebe um token após o login.
- O token contém informações do usuário e tempo de expiração, sendo validado em cada requisição subsequente.
- O filtro JwtAuthenticationFilter verifica se a requisição possui um token válido e extrai as informações do usuário para autenticação no contexto de segurança do Spring.

Gerenciamento de Tokens JWT

- Os tokens são assinados com uma chave secreta configurada via application.properties.
- A expiração do token é configurável e definida pelo valor de jwt.expiration.
- O JwtUtil é responsável por gerar e validar tokens, além de extrair informações do usuário a partir do token JWT.

Controle de Acesso

- Os endpoints /api/auth/** e /api/webhooks/** são públicos.
- Todos os demais endpoints exigem autenticação.
- O controle de acesso é feito através do Spring Security e anotação @PreAuthorize, garantindo que apenas usuários com os devidos privilégios possam acessar recursos protegidos.

Armazenamento de Credenciais

As senhas dos usuários são armazenadas no banco de dados utilizando BCryptPasswordEncoder, garantindo um alto nível de segurança.

O serviço CustomUserDetailsService carrega os detalhes do usuário a partir do banco de dados e converte para um formato compatível com o Spring Security.

8.3. Resumo

A API implementa uma política de segurança robusta utilizando Spring Security e JWT. A configuração de CORS permite acesso controlado às origens configuradas, garantindo que apenas aplicações autorizadas consumam os serviços. A autenticação e autorização são gerenciadas de forma segura, protegendo os dados do sistema contra acessos não autorizados.

9. Endpoints da API

A API do MercadoAPI expõe diversos endpoints para gerenciar pedidos, pagamentos, controle de estoque e autenticação. Eles estão organizados conforme as responsabilidades do sistema.

9.1 Autenticação

POST /api/auth/register → Registro de um novo usuário.

POST /api/auth/login → Autenticação de usuário e geração de token JWT.

POST /api/auth/forgot-password → Envio de e-mail para redefinição de senha.

POST /api/auth/reset-password → Redefinição de senha com token recebido por e-mail.

9.2 Gestão de Pedidos

POST /api/orders → Criar um novo pedido (carrinho).

POST /api/orders/{orderId}/items → Adicionar itens ao pedido.

PUT /api/orders/{orderId}/checkout → Finalizar o pedido (checkout).

PATCH /api/orders/{orderId}/status → Atualizar o status do pedido (admin).

PUT /api/orders/{orderId}/cancel → Cancelar um pedido.

GET /api/orders/{orderId} → Buscar detalhes de um pedido específico.

GET /api/orders/my-orders → Listar pedidos do usuário logado.

GET /api/orders/tracking/{trackingCode} → Buscar pedido por código de rastreio.

GET /api/orders → Listar todos os pedidos (admin).

9.3 Pagamentos

POST /api/payments/create?orderId={orderId}&currency=brl{currency} → Criar um pagamento para um pedido.

POST /api/webhooks → Webhook do Stripe para capturar pagamentos e reembolsos.

9.4 Controle de Estoque

GET /api/products → Listar produtos disponíveis.

GET /api/products/{productId} → Buscar detalhes de um produto específico.

POST /api/products → Criar um novo produto (admin).

PUT /api/products/{productId} → Atualizar informações de um produto (admin).

DELETE /api/products/{productId} → Remover um produto (admin).

10. Implantação

A API MercadoAPI está implantada utilizando Docker e Railway, garantindo escalabilidade e deploy automatizado via CI/CD.

10.1 Ambiente de Produção

A API está hospedada no Railway e pode ser acessada pela seguinte URL base:

https://mercadoapi-production.up.railway.app/api

10.2 Pipeline de CI/CD

O projeto conta com um pipeline de CI/CD configurado para automação do deploy. O fluxo inclui:

- Testes automatizados → Antes do deploy, garantindo a estabilidade do código.
- Build e Deploy Automático → Utilizando o GitHub e integração com o Railway.
- Infraestrutura em Docker → Banco de dados PostgreSQL configurado via Docker.