API Restful E-commerce - Documentação Completa

---

1. Visão Geral

Esta documentação descreve a API de E-commerce desenvolvida para gerenciar **usuários, produtos, categorias, carrinhos de compras, pedidos, clientes (para OAuth2), endereços, métodos de pagamento, cupons, notificações e métricas de dashboard**. A API segue princípios **RESTful** e utiliza **JSON** para troca de dados. A segurança é implementada com **Spring Security**, abrangendo autenticação via formulário, OAuth2 para login social e JWT para proteção de recursos.

---

2. Arquitetura

A aplicação segue uma **arquitetura em camadas**, com as seguintes responsabilidades principais:

- **Modelos (io.github.gabrielpetry23.ecommerceapi.model)**: Representam as **entidades do domínio** da aplicação e são mapeados para as tabelas do banco de dados utilizando JPA (Jakarta Persistence API).
- **Repositories (io.github.gabrielpetry23.ecommerceapi.repository)**: Fornecem a abstração para **acesso e manipulação dos dados** persistidos no banco de dados, utilizando Spring Data JPA.
- **Services (io.github.gabrielpetry23.ecommerceapi.service)**: Contêm a **lógica de negócios** da aplicação, orquestrando as operações e interagindo com os repositories. Incluem serviços para gerenciamento de cupons, notificações e métricas de dashboard, além de um serviço para envio de lembretes de carrinho abandonado.
- **Controllers (io.github.gabrielpetry23.ecommerceapi.controller)**: Recebem as **requisições HTTP**, delegam a lógica para os services e retornam as respostas para o cliente.
- **DTOs (io.github.gabrielpetry23.ecommerceapi.controller.dto)**: **Data Transfer Objects**, utilizados para transportar dados entre as camadas e expor informações específicas nas requisições e respostas da API.
- **Mappers (io.github.gabrielpetry23.ecommerceapi.controller.mappers)**: Responsáveis por **converter entre as entidades de modelo e os DTOs**, facilitando a separação de responsabilidades.
- **Validators (io.github.gabrielpetry23.ecommerceapi.validators)**: Implementam a **lógica de validação** para garantir a integridade dos dados de entrada.

- **Segurança (io.github.gabrielpetry23.ecommerceapi.security)**: Configurações e componentes relacionados à **autenticação e autorização**, incluindo JWT, OAuth2 e controle de acesso por roles.
- **Configurações (io.github.gabrielpetry23.ecommerceapi.configuration)**: Classes de configuração para o Spring, incluindo banco de dados, segurança, OAuth2 e Swagger.
- **Exceções (io.github.gabrielpetry23.ecommerceapi.exceptions)**: Classes de **exceções personalizadas** para tratamento de erros específicos da aplicação.
- **Handlers de Exceção (io.github.gabrielpetry23.ecommerceapi.controller.common.GlobalExceptionHandler)**: Centralizam o **tratamento de exceções** para retornar respostas de erro consistentes e informativas.

---

3. Modelo de Dados (Diagrama Entidade-Relacionamento)

---

4. Endpoints da API

A seguir, uma lista detalhada dos endpoints da API, com seus métodos HTTP, URIs e permissões de acesso:

USUÁRIOS (/users)

Método	Endpoint	Descrição	Autorização

POST	/users	Criar um novo usuário	Público
GET	/users/{id}	Obter um usuário específico	USER (próprio), ADMIN, MANAGER
PUT	/users/{id}	Atualizar um usuário	USER (próprio), ADMIN, MANAGER
DELETE	/users/{id}	Excluir um usuário	ADMIN, MANAGER
POST	/users/login	Login (Autenticação via formulário)	Público

PRODUTOS (/products)

Método	Endpoint	Descrição	Autorização
POST	/products	Criar um novo produto	ADMIN, MANAGER
GET	/products	Listar todos os produtos (com paginação)	Público
GET	/products/search	Pesquisar produtos (com filtros)	Público
GET	/products/{id}	Obter um produto específico	Público
PUT	/products/{id}	Atualizar um produto	ADMIN, MANAGER
DELETE	/products/{id}	Excluir um produto	ADMIN, MANAGER
GET	/products/{id}/reviews	Obter as reviews de um produto	Público
POST	/products/{id}/reviews	Criar uma review	USER
DELETE	/products/{id}/reviews/{reviewId}	Excluir uma review	USER (próprio), ADMIN, MANAGER
POST	/products/{id}/images	Adicionar imagem ao produto	ADMIN, MANAGER
DELETE	/products/{id}/images/{imageId}	Remover imagem do produto	ADMIN, MANAGER

CATEGORIAS (/categories)

Método	Endpoint	Descrição	Autorização
GET	/categories	Listar todas as categorias	Público
GET	/categories/{id}	Obter uma categoria específica	Público
POST	/categories	Criar uma categoria	ADMIN, MANAGER
PUT	/categories/{id}	Atualizar uma categoria	ADMIN, MANAGER
DELETE	/categories/{id}	Excluir uma categoria	ADMIN, MANAGER
GET	/categories/{id}/products	Listar produtos por categoria	Público

CARRINHOS (/carts)

Método	Endpoint	Descrição	Autorização
GET	/carts/{id}	Obter um carrinho específico	USER (próprio), ADMIN, MANAGER
POST	/carts	Criar um novo carrinho (associado ao USER logado)	USER
POST	/carts/{id}/items	Adicionar item ao carrinho	USER (dono)
PUT	/carts/{id}/items/{itemId}	Atualizar item no carrinho	USER (dono)
DELETE	/carts/{id}/items/{itemId}	Remover item do carrinho	USER (dono)
DELETE	/carts/{id}	Esvaziar/Excluir o carrinho	USER (próprio), ADMIN, MANAGER
GET	/users/{userId}/cart	Obter carrinho de um usuário	USER (próprio), ADMIN, MANAGER

PEDIDOS (/orders)

Método	Endpoint	Descrição	Autorização
POST	/orders	Criar novo pedido	USER
GET	/orders	Listar todos os pedidos (com paginação)	ADMIN, MANAGER
GET	/orders/{id}	Obter um pedido específico	USER (próprio), ADMIN, MANAGER
GET	/users/{userId}/orders	Obter pedidos de um usuário (com paginação)	USER (próprio), ADMIN, MANAGER
PUT	/orders/{id}/status	Atualizar status de um pedido	ADMIN, MANAGER
GET	/orders/{orderId}/tracking	Obter informações de rastreamento de um pedido	USER (próprio), ADMIN, MANAGER
POST	/orders/{orderId}/coupon	Aplicar cupom de desconto a um pedido	USER (próprio)

CLIENTES (OAuth) (/clients)

Método	Endpoint	Descrição	Autorização
POST	/clients	Criar um novo Client	ADMIN, MANAGER
GET	/clients/{id}	Obter um Client específico	ADMIN, MANAGER
PUT	/clients/{id}	Atualizar um Client	ADMIN, MANAGER
DELETE	/clients/{id}	Excluir um Client	ADMIN, MANAGER

ENDEREÇOS (/users/{userId}/addresses)

Método	Endpoint	Descrição	Autorização
GET	/users/{userId}/addresses	Obter endereços de um usuário	USER (próprio), ADMIN, MANAGER

GET	/users/{userId}/addresses/{addressId}	Obter endereço específico	USER (próprio), ADMIN, MANAGER
POST	/users/{userId}/addresses	Criar endereço	USER (próprio)
PUT	/users/{userId}/addresses/{addressId}	Atualizar endereço	USER (próprio), ADMIN, MANAGER
DELETE	/users/{userId}/addresses/{addressId}	Deletar endereço	USER (próprio), ADMIN, MANAGER

MÉTODOS DE PAGAMENTO (/users/{userId}/payment-methods)

Método	Endpoint	Descrição	Autorização
GET	/users/{userId}/payment-methods	Obter métodos de pagamento	USER (próprio), ADMIN, MANAGER
GET	/users/{userId}/payment-methods/{paymentMethodId}	Obter método de pagamento	USER (próprio), ADMIN, MANAGER
POST	/users/{userId}/payment-methods	Criar método de pagamento	USER (próprio)
PUT	/users/{userId}/payment-methods/{paymentMethodId}	Atualizar método de pagamento	USER (próprio), ADMIN, MANAGER
DELETE	/users/{userId}/payment-methods/{paymentMethodId}	Deletar método de pagamento	USER (próprio), ADMIN, MANAGER

NOTIFICAÇÕES (/users/{userId}/notifications)

Método	Endpoint	Descrição	Autorização
GET	/users/{userId}/notifications	Obter todas as notificações de um usuário	USER (próprio), ADMIN, MANAGER
GET	/users/{userId}/notifications/unread	Obter notificações não lidas	USER (próprio), ADMIN, MANAGER

		de um usuário	
PUT	/users/{userId}/notifications/mark-all-as-read	Marcar todas as notificações de um usuário como lidas	USER (próprio)
PUT	/users/{userId}/notifications/{notificationId}/mark-as-read	Marcar uma notificação específica como lida	USER (próprio)

CUPONS (/coupons)

Método	Endpoint	Descrição	Autorização
POST	/coupons	Criar um novo cupom	ADMIN, MANAGER
GET	/coupons	Listar todos os cupons (com paginação)	ADMIN, MANAGER

DASHBOARD (/dashboard)

Método	Endpoint	Descrição	Autorização
GET	/dashboard/metrics	Obter métricas do dashboard (pedidos por mês, produtos mais vendidos, valor total de vendas)	ADMIN, MANAGER

---

5. Autenticação e Autorização

A API utiliza **Spring Security** para autenticação e autorização, com as seguintes estratégias:

- **Autenticação via Formulário (/login ou /users/login)**: Para usuários administrativos e gerenciais acessarem funcionalidades específicas através de uma interface web (se implementada).
- **Login Social (OAuth2)**: Permite que usuários se autentiquem utilizando suas contas de provedores de identidade (e.g., Google, Facebook). O endpoint /oauth2/login inicia o fluxo de autenticação, e o /authorized recebe o código de autorização.
- **JWT (JSON Web Tokens)**: Utilizado para proteger os recursos da API. Após a autenticação (via formulário ou OAuth2), um JWT pode ser gerado e enviado nas requisições subsequentes no header Authorization: Bearer <token>. O JwtCustomAuthenticationFilter processa esses tokens, e as **roles do usuário** contidas no token são utilizadas para autorização.

Os **roles de usuário** definidos são:

- **USER**: Usuário comum da plataforma.
- **MANAGER**: Usuário com permissões para gerenciar produtos, categorias e cupons.
- **ADMIN**: Usuário com permissões administrativas completas, incluindo gerenciamento de clientes OAuth e acesso a métricas de dashboard.

O acesso aos endpoints é controlado através de anotações @PreAuthorize nos controllers, verificando se o usuário autenticado possui o role necessário.

---

6. Tratamento de Erros

A API implementa um **tratamento global de exceções** através do GlobalExceptionHandler. As respostas de erro seguem um formato consistente (ResponseError) e incluem o código de status HTTP, uma mensagem de erro e, em alguns casos, detalhes adicionais (e.g., campos inválidos na validação).

Os principais **códigos de status de erro** utilizados incluem:

- **400 Bad Request**: Requisição malformada ou inválida (IllegalArgumentException, OperationNotAllowedException).
- **401 Unauthorized**: Não autenticado.
- **403 Forbidden**: Acesso negado devido à falta de permissões (AccessDeniedException).
- **404 Not Found**: Recurso não encontrado (EntityNotFoundException, ResourceNotFoundException).
- **409 Conflict**: Violação de integridade de dados, como registros duplicados (DuplicateRecordException).
- **422 Unprocessable Entity**: Erros de validação (MethodArgumentNotValidException, InvalidFieldException, InvalidCouponException).
- **500 Internal Server Error**: Erro inesperado no servidor (RuntimeException, EmailSendingException).

---

7. Configurações

A aplicação utiliza arquivos de configuração do Spring (e.g., application.properties ou application.yml) para definir propriedades como URL do banco de dados, credenciais, configurações de segurança e propriedades do OAuth2.

As classes de configuração em io.github.gabrielpetry23.ecommerceapi.configuration incluem:

- **AuthorizationServerConfiguration**: Configura o servidor de autorização OAuth2.
- **DatabaseConfiguration**: Configura a conexão com o banco de dados (utilizando HikariCP).
- **OpenApiConfiguration**: Configura a documentação Swagger/OpenAPI.
- **ResourceServerConfiguration**: Configura o servidor de recursos, protegendo os endpoints da API com JWT e definindo as regras de autorização.
- **WebConfiguration**: Configura aspectos gerais da web, como a view controller para a página de login.
- **WebSocketConfig**: Configuração para o suporte a WebSockets, utilizado para notificações em tempo real.

---

8. Segurança

A segurança é um aspecto fundamental desta API, implementada através de:

- **HTTPS**: É altamente recomendado que a API seja servida através de HTTPS para garantir a **criptografia das comunicações** entre o cliente e o servidor, protegendo dados sensíveis.
- **Validação de Entrada**: A API utiliza as anotações de validação do Jakarta Bean Validation (@NotNull, @NotBlank, @Size, etc.) nos DTOs, que são verificadas pelo Spring (@Valid) nos controllers. Isso ajuda a prevenir ataques de injeção e garante a integridade dos dados. Erros de validação são tratados pelo GlobalExceptionHandler e retornam um status 422 Unprocessable Entity com detalhes dos campos inválidos.
- **Proteção contra CSRF (Cross-Site Request Forgery)**: Embora desabilitado na configuração (csrf(AbstractHttpConfigurer::disable)), em cenários de produção com interfaces web tradicionais, a proteção CSRF é crucial para prevenir ataques onde um site malicioso tenta realizar ações em nome de um usuário autenticado sem o seu consentimento. Se uma interface web for integrada, reativar e configurar a proteção CSRF é essencial.
- **Segurança das Senhas**: As senhas dos usuários são armazenadas de forma segura no banco de dados utilizando o algoritmo de hash **bcrypt**, configurado no PasswordEncoder bean. Isso garante que as senhas reais não sejam armazenadas em texto plano.

- **OAuth2 Client Secrets**: Os segredos dos clientes OAuth2 (client_secret) são armazenados de forma segura no banco de dados e devem ser tratados com confidencialidade.
- **Controle de Acesso Granular**: A utilização de roles e a anotação @PreAuthorize permitem um controle de acesso preciso aos recursos da API, garantindo que apenas usuários autorizados possam realizar determinadas ações.
- **Atualizações de Segurança**: Manter as dependências do projeto (Spring Boot, bibliotecas de segurança, etc.) atualizadas é crucial para mitigar vulnerabilidades de segurança conhecidas.

---

9. Swagger/OpenAPI

A documentação da API está disponível através do **Swagger/OpenAPI**, acessível nos seguintes endpoints (configurados em WebSecurityCustomizer para serem ignorados pela segurança):

- /v3/api-docs/**
- /swagger-ui.html/**
- /swagger-ui/**

A configuração do OpenAPI é realizada na classe OpenApiConfiguration, que define informações gerais sobre a API (título, versão, descrição, contato) e configura o esquema de segurança Bearer JWT para autenticação. Utilize a interface Swagger UI para explorar os endpoints da API, visualizar os modelos de requisição e resposta e realizar testes interativos.

---

10. Tecnologias Utilizadas

- **Java**: Linguagem de programação principal.
- **Spring Boot**: Framework para desenvolvimento rápido de aplicações Java.
- **Spring Data JPA**: Abstração para acesso a dados utilizando JPA.
- **Hibernate**: Implementação do JPA utilizada pelo Spring Data JPA.
- **PostgreSQL**: Banco de dados relacional utilizado para persistência.
- **HikariCP**: Pool de conexões JDBC de alta performance.
- **Spring Security**: Framework para segurança da aplicação (autenticação e autorização).
- **Spring Security OAuth2**: Framework para implementação de OAuth2.
- **JSON Web Tokens (JWT)**: Padrão para criação de tokens de acesso seguros.
- **Spring WebSocket**: Suporte a comunicação WebSocket para notificações em tempo real.
- **MapStruct**: Biblioteca para geração de código de mappers entre objetos Java.
- **Lombok**: Biblioteca para reduzir a boilerplate de código Java.
- **Jakarta Bean Validation**: API para validação de beans.

- **Swagger/OpenAPI**: Ferramenta para documentação da API RESTful.
- **Maven**: Ferramenta de gerenciamento de dependências e build.
- **Spring ShedLock**: (Potencialmente) Para garantir que tarefas agendadas (como lembretes de carrinho) sejam executadas apenas uma vez em ambientes distribuídos.

---

11. Como Utilizar a API

Este guia rápido o ajudará a colocar a API em funcionamento e a testar seus endpoints principais usando o Swagger UI.

Pré-requisitos

Certifique-se de ter o Docker e o Docker Compose instalados em sua máquina.

1. Inicializar a Aplicação

Clone o repositório:

```
git clone https://github.com/gabrielpetry23/ecommerce-api.git
```

```
cd ecommerce-api
```

Atenção sobre as Variáveis de Ambiente:

No arquivo docker-compose.yml, você encontrará variáveis de ambiente para serviços externos (como Google e E-mail). Para rodar o projeto localmente, por favor:

- **Substitua os placeholders** (ex: SUA_CLIENT_ID_LOCAL_OU_DUMMY) por suas próprias credenciais de desenvolvimento/teste, ou por valores dummy se essas funcionalidades não forem cruciais para a demonstração local.

2. Limpar e Construir/Iniciar a Aplicação: Para garantir que o banco de dados seja inicializado com os dados de teste mais recentes, é **crucial** remover os volumes persistidos antes de subir os contêineres. Isso garante que o script data.sql seja executado sempre que o contêiner do banco de dados for inicializado e evita problemas com IDs já existentes.

```
docker-compose down --volumes
```

```
docker-compose up -d --build
```

    - O comando down --volumes remove os dados anteriores do banco, incluindo o volume persistente db_data.

- O comando up -d --build reconstrói a imagem da aplicação (caso haja mudanças no código Dockerfile) e inicia todos os serviços (banco de dados e API) em segundo plano.

Aguarde a Inicialização: Aguarde alguns segundos (ou observe os logs com docker-compose logs -f) até que a aplicação Spring Boot (ecommerce_api_local) esteja totalmente iniciada. Você verá mensagens como Started EcommerceapiApplication in X.X seconds no log do contêiner ecommerce_api_local.

3. Acessar o Swagger UI

Uma vez que a aplicação esteja de pé, você pode acessar a interface do Swagger UI para explorar e testar os endpoints:

- **Swagger UI:** http://localhost:8080/swagger-ui/index.html

4. Usuários de Teste e Credenciais

O banco de dados da aplicação já vem **pre-populado** com alguns dados de teste, incluindo usuários, categorias, produtos e carrinhos, através do script data.sql. Você pode usar as seguintes credenciais para autenticação:

Usuário	Email	Senha	Papel
Gabriel Silva	gabriel.user@example.com	gabriel123	USER
Ana Gerente	gerente.admin@example.com	gerente123	MANAGER
Maria Souza	maria.customer@example.com	maria123	USER
João Pereira	joao.customer@example.com	joao123	USER

Importante: Como os IDs são gerados dinamicamente (gen_random_uuid()) no data.sql a cada vez que o banco é recriado (docker-compose down --volumes), os UUIDs listados acima são apenas para referência e serão diferentes no seu ambiente. Você precisará obter os IDs reais via API após o login ou listagem.

5. Fluxo de Teste Comum (Exemplo)

Siga estes passos para testar um fluxo básico de autenticação, listagem de produtos e criação de um novo carrinho/pedido.

Obter Token de Acesso (Login): A autenticação pode ser feita de duas formas:

- **Via Endpoint /auth/login (Recomendado para uso com a API):**
  - No Swagger UI, expanda o endpoint AuthController.
  - Selecione /auth/login [POST].
  - Clique em "Try it out".

No campo "Request body", insira as credenciais de um usuário (ex: gabriel.user@example.com e gabriel123):

```
{
  "email": "gabriel.user@example.com",
  "password": "gabriel123"
}
```

  - Clique em "Execute".
  - Copie o accessToken da resposta.
- **Via Botão "Authorize" do Swagger UI (Login simplificado para testes no Swagger):**
  - No topo da página do Swagger UI, clique no botão "Authorize".
  - Uma janela pop-up aparecerá. Selecione a opção bearerAuth (OAuth2, implicit).
  - No campo "Username", insira o email do usuário (ex: gabriel.user@example.com).
  - No campo "Password", insira a senha do usuário (ex: gabriel123).
  - Clique em "Authorize" e depois "Close". O Swagger UI fará a requisição de login automaticamente e armazenará o token para as próximas requisições.

Autorizar Requisições no Swagger (se você usou o endpoint /auth/login):

- No topo da página do Swagger UI, clique no botão "Authorize".
- Cole o accessToken que você copiou no campo Value para bearerAuth (OAuth2, implicit).
- Clique em "Authorize" e depois "Close". Agora suas requisições estarão autenticadas.

Acessar Perfil do Usuário Logado:

- Expanda o endpoint UserController.
- Selecione /users/me [GET].
- Clique em "Try it out" e depois "Execute".
- Você deverá ver os detalhes do usuário logado, incluindo o id (UUID). **Guarde este id!** Ele será necessário para outras operações.

Listar Produtos e Obter IDs:

- Expanda o endpoint ProductController.
- Selecione /products [GET].
- Clique em "Try it out" e "Execute".
- Analise a lista de produtos retornada. Copie os ids de produtos que você deseja adicionar ao carrinho/pedido (ex: o ID de um Laptop Gamer XYZ).

Criar um Novo Endereço (se necessário para um novo pedido):

- Expanda o endpoint AddressController.
- Selecione /addresses [POST].
- Clique em "Try it out".

No "Request body", insira os detalhes do endereço usando o userId que você obteve no passo 3.

```
{
  "userId": "UUID_DO_SEU_USUARIO_AQUI",
  "street": "Rua Exemplo",
  "number": "456",
  "complement": "Apto 202",
  "city": "Cidade Teste",
  "state": "TS",
  "zipCode": "99999-999",
```

```
  "country": "Brasil"
}
```

- Clique em "Execute". Copie o id do endereço retornado.

Criar um Carrinho e Adicionar Itens:

- Expanda o endpoint CartController.
- Selecione /carts [POST] para criar um novo carrinho para o usuário logado (usando o userId do passo 3). O id do carrinho será retornado.
- Selecione /cart-items [POST] para adicionar itens ao carrinho.

No "Request body", use o cartId que você acabou de obter e os productIds que você copiou no passo 4.

```
{
  "cartId": "UUID_DO_SEU_CARRINHO_AQUI",
  "productId": "UUID_DO_PRODUTO_AQUI",
  "quantity": 1
}
```

- Repita para adicionar mais produtos, se desejar.

Finalizar um Pedido:

- Expanda o endpoint OrderController.
- Selecione /orders [POST].
- Clique em "Try it out".
- No "Request body", preencha os detalhes, usando os ids que você obteve (ou use IDs de referência existentes no data.sql para testes).

Exemplo usando cartId:

```
{
  "userId": "UUID_DO_SEU_USUARIO_AQUI",
```

```
  "status": "PENDING",
  "deliveryAddressId": "UUID_DO_ENDERECO_DO_USUARIO_AQUI",
  "paymentMethodId": "UUID_DO_METODO_PAGAMENTO_DO_USUARIO_AQUI",
  "cartId": "UUID_DO_CARRINHO_DO_USUARIO_AQUI"
}
```

Exemplo construindo orderItems diretamente (sem cartId):

```
{
  "userId": "UUID_DO_SEU_USUARIO_AQUI",
  "status": "PENDING",
  "deliveryAddressId": "UUID_DO_ENDERECO_DO_USUARIO_AQUI",
  "paymentMethodId": "UUID_DO_METODO_PAGAMENTO_DO_USUARIO_AQUI",
  "orderItems": [
    {
      "productId": "UUID_DO_LAPTOP_GAMER_AQUI",
      "quantity": 1,
      "price": 7500.00
    },
    {
      "productId": "UUID_DO_FONE_DE_OUVIDO_AQUI",
      "quantity": 2,
      "price": 250.00
    }
  ]
}
```

- Clique em "Execute". O id do pedido criado será retornado.

Verificar Pedidos Existentes (do data.sql):

- Selecione /orders/{orderId} [GET] e use o id de um dos usuários de teste (ex: Gabriel, Maria, João) para ver os pedidos associados a eles. Como os IDs dos pedidos no data.sql também são dinâmicos agora, você precisará:
  - Fazer login como gerente.admin@example.com (gerente123).
  - Acessar /orders [GET] para listar **todos** os pedidos (o papel MANAGER tem permissão).
  - Identificar os pedidos de Gabriel, Maria e João pelos seus respectivos userIds e status.

Notificações em Tempo Real: Para receber notificações em tempo real via WebSocket, conecte-se ao endpoint /ws (ou /app, dependendo da sua configuração de stomp) do WebSocket e inscreva-se nos tópicos relevantes.

- **Exemplo:** /topic/user-notifications/{userId} para notificações específicas do usuário.

6. Parar a Aplicação

Para parar os contêineres e liberar os recursos (mantendo os dados do banco, a menos que você adicione --volumes):

```
docker-compose down
```

---

12. Regras de Negócio

Usuários e Perfis

- **Autenticação Obrigatória para Recursos Protegidos**: A maioria dos endpoints da API exige que o usuário esteja autenticado e possua um JWT (JSON Web Token) válido no cabeçalho da requisição.
- **Controle de Acesso por Role**: O sistema define três roles (perfis) de usuário:
  - **USER**: Pode gerenciar seu próprio perfil (endereços, métodos de pagamento), criar carrinhos e pedidos, adicionar reviews a produtos e visualizar suas notificações.

    - **MANAGER**: Possui todas as permissões de USER e, além disso, pode gerenciar produtos (criar, listar, atualizar, excluir), categorias e cupons. Pode também listar todos os pedidos e clientes, e atualizar o status de pedidos.
    - **ADMIN**: Possui acesso irrestrito a todas as funcionalidades do sistema, incluindo o gerenciamento completo de usuários, clientes OAuth, e acesso às métricas de dashboard.
- **Senhas Criptografadas**: As senhas dos usuários são armazenadas no banco de dados usando **bcrypt**, garantindo que não sejam recuperáveis em texto plano.
- **Verificação de E-mail (Implícita)**: Embora não haja um endpoint explícito de confirmação de e-mail na lista fornecida, a existência da tabela email_verification_tokens sugere que, em algum ponto do fluxo de criação de usuário, um processo de verificação de e-mail é esperado para validar a autenticidade do endereço de e-mail do usuário.

Produtos e Categorias

- **Unicidade de Categorias**: Cada categoria deve ter um nome único.
- **Disponibilidade de Produto**: Produtos possuem um stock (estoque) que deve ser verificado antes de serem adicionados ao carrinho ou a um pedido. A API deve garantir que a quantidade comprada não exceda o estoque disponível.
- **Preço e Estoque Obrigatórios**: Todo produto deve ter um price (preço) e stock (estoque) definidos.
- **Imagens de Produto**: Produtos podem ter múltiplas imagens, e uma delas pode ser marcada como principal (is_main).

Carrinhos de Compras

- **Carrinho por Usuário**: Cada usuário autenticado pode ter apenas um carrinho de compras ativo.
- **Adição de Itens**: Somente produtos com estoque disponível podem ser adicionados ao carrinho. A quantidade deve ser um número inteiro positivo.
- **Atualização de Itens**: É possível ajustar a quantidade de um item no carrinho. A validação de estoque se aplica aqui também.
- **Esvaziar Carrinho**: O carrinho pode ser esvaziado completamente ou itens individuais podem ser removidos.
- **Lembretes de Carrinho Abandonado**: Um serviço agendado (CartReminderService) verifica periodicamente carrinhos que não foram atualizados há mais de 24 horas (ABANDONED_HOURS) e que não receberam um lembrete nas últimas 48 horas (REMINDER_COOLDOWN_HOURS). Para esses carrinhos, uma notificação é enviada ao usuário.

Pedidos

- **Criação de Pedido a partir do Carrinho**: Um pedido só pode ser criado se o carrinho do usuário não estiver vazio.

- **Validação de Endereço e Pagamento**: O endereço de entrega e o método de pagamento selecionados para o pedido devem pertencer ao usuário que está criando o pedido.
- **Consistência de Preço**: O preço dos itens no pedido é capturado no momento da criação do pedido e não se altera se o preço do produto original for modificado posteriormente.
- **Status do Pedido**: Pedidos seguem um fluxo de status predefinido (PENDING, PAID, IN_PREPARATION, IN_DELIVERY, DELIVERED, CANCELLED). Transições de status inválidas devem ser impedidas (ex: não é possível ir de DELIVERED para PENDING).
- **Geração de Rastreamento**: Quando o status de um pedido muda para PAID, um código de rastreamento e detalhes simulados são gerados automaticamente e associados ao pedido.
- **Aplicação de Cupons**: Um cupom pode ser aplicado a um pedido durante a sua criação, e o total do pedido será recalculado com base no desconto do cupom.
- **Esvaziamento do Carrinho**: Após a criação bem-sucedida de um pedido, o carrinho do usuário associado é esvaziado.
- **Notificações de Pedido**: Notificações são enviadas ao usuário em eventos importantes do ciclo de vida do pedido, como criação e atualização de status.

Cupons

- **Código Único**: Cada cupom deve ter um código único.
- **Tipos de Desconto**: Cupons podem oferecer um discount_amount (valor fixo) ou discount_percentage (porcentagem).
- **Validade**: Cupons possuem uma data de validade (valid_until) e um status is_active. Um cupom só é válido se estiver ativo e dentro do seu período de validade.
- **Notificação de Novos Cupons**: Quando um novo cupom ativo é criado, todos os usuários com o perfil USER recebem uma notificação sobre o cupom.

Notificações

- **Notificações por Usuário**: As notificações são vinculadas a usuários específicos.
- **Status de Leitura**: Notificações podem ser marcadas como lidas (read_at), individualmente ou em massa.
- **Notificações em Tempo Real**: As notificações são enviadas em tempo real aos usuários através de WebSockets.
- **Persistência**: Todas as notificações enviadas são persistidas no banco de dados.

E-mails

- **Fila de E-mails para Processamento Assíncrono**: A aplicação utiliza uma fila (email_queue) para gerenciar o envio de e-mails de forma assíncrona, prevenindo que a operação de envio de e-mail bloqueie o fluxo principal da aplicação.

- **Verificação de E-mail de Usuário**: Usuários recém-criados devem passar por um processo de verificação de e-mail (utilizando email_verification_tokens) para confirmar a autenticidade de seu endereço de e-mail.
- **Resiliência no Envio de E-mails**: O sistema é projetado para incluir mecanismos de retentativa e tratamento de erros para o envio de e-mails, garantindo que mesmo falhas temporárias não impeçam a entrega de comunicações importantes.

Dashboard

- **Métricas Agregadas**: O dashboard fornece métricas de vendas, como o número de pedidos por mês, os produtos mais vendidos e o valor total de vendas.
- **Acesso Restrito**: As métricas do dashboard são acessíveis apenas por usuários com os perfis ADMIN ou MANAGER.

---

13. Testes de Integração

O projeto conta com uma suíte abrangente de **testes de integração para todos os controllers**, garantindo que os endpoints da API funcionem corretamente em conjunto com as camadas de serviço e segurança. Esses testes utilizam MockMvc para simular requisições HTTP e verificar o comportamento das respostas, incluindo códigos de status, headers e corpo JSON.

As principais tecnologias e bibliotecas utilizadas nos testes de integração são:

- **JUnit 5**: Framework para escrita e execução de testes.
- **Spring Boot Test**: Fornece suporte para testes de integração em aplicações Spring Boot.
- **@AutoConfigureMockMvc**: Configura automaticamente o MockMvc para testes de controller.
- **MockMvc**: Objeto para simular requisições HTTP para os controllers.
- **ObjectMapper**: Utilizado para serializar e desserializar objetos Java para/de JSON.
- **@MockitoBean**: Permite mockar e injetar beans do Spring, isolando a camada do controller e controlando o comportamento dos serviços.
- **Spring Security Test**: Oferece utilitários para simular autenticação (e.g., com JWT) e proteção CSRF em testes.

---

14. Contribuição

Interessado em aprimorar este projeto? Siga as diretrizes abaixo para facilitar o processo de colaboração e garantir a qualidade do código. Sua contribuição é muito bem vinda!

Como Contribuir

1. **Faça um Fork do Repositório:**
    - Crie um fork do repositório principal para a sua conta do GitHub.

Clone o Seu Fork:

```
git clone https://github.com/SEU_USUARIO/ecommerce-api.git
```

```
cd ecommerce-api
```

2. **Crie uma Nova Branch:**
    - Crie uma branch para a sua funcionalidade ou correção. Utilize um nome descritivo (ex: feature/nome-da-funcionalidade, bugfix/correcao-de-login).

```
git checkout -b feature/minha-nova-funcionalidade
```

3. **Desenvolva Suas Alterações:**
    - Implemente as suas alterações, seguindo as convenções de código do projeto.
    - Certifique-se de que todas as alterações estão funcionando corretamente e que os testes existentes (se houver) ainda passam. Se aplicável, adicione novos testes para a sua funcionalidade/correção.
4. **Commit Suas Alterações:**
    - Escreva mensagens de commit claras e concisas, explicando o que foi feito.
      git add .

```
git commit -m "feat: Adiciona endpoint para gerenciamento de cupons"
```

    - *Sugestão de Convenção de Commits (opcional, mas profissional):*
        - feat: (para novas funcionalidades)
        - fix: (para correção de bugs)
        - docs: (para mudanças na documentação)
        - style: (para formatação, sem mudança no código)
        - refactor: (para refatoração de código)
        - test: (para adição ou correção de testes)
        - chore: (para tarefas de manutenção, como atualizações de dependência)

Envie Suas Alterações para o Seu Fork:

```
git push origin feature/minha-nova-funcionalidade
```

5. **Abra um Pull Request (PR):**
    - Vá para o repositório original no GitHub e crie um Pull Request da sua branch para a branch main (ou master, dependendo da branch principal do projeto).
    - Forneça uma descrição detalhada das suas alterações, explicando o problema que resolve e como a solução funciona.
    - Se o PR resolver um problema específico, mencione a *issue* correspondente (ex: Closes #123).

Convenções de Código e Padrões

- **Java:** Siga as convenções de código padrão do Java e os princípios SOLID.
- **Spring Boot:** Utilize as melhores práticas do Spring Boot, como injeção de dependência e anotações apropriadas.
- **Formatação:** Mantenha a formatação consistente com o código existente (pode ser útil configurar um Prettier ou linter no IDE para isso).
- **Nomenclatura:** Utilize nomes de variáveis, métodos e classes claros e significativos.
- **Testes:** Priorize a escrita de testes unitários e de integração para garantir a robustez das funcionalidades.

Comunicação

- Em caso de dúvidas ou discussões sobre novas funcionalidades, sinta-se à vontade para abrir uma *Issue* no repositório antes de iniciar o desenvolvimento.

---

15. Próximos Passos e Melhorias

- Adição de testes unitários e de integração para aumentar a confiabilidade da aplicação.
- Implementação de paginação e filtragem mais avançadas em outros endpoints.
- Melhorias no tratamento de erros e mensagens de resposta mais detalhadas.
- Implementação de logs mais abrangentes para monitoramento da aplicação.
- Considerar a utilização de um sistema de cache para melhorar a performance.
- Implementação de verificação de e-mail de usuário.
- Mecanismos de resiliência para o envio de e-mails, como retentativas.
- Possível integração com um serviço de gateway de pagamento real.