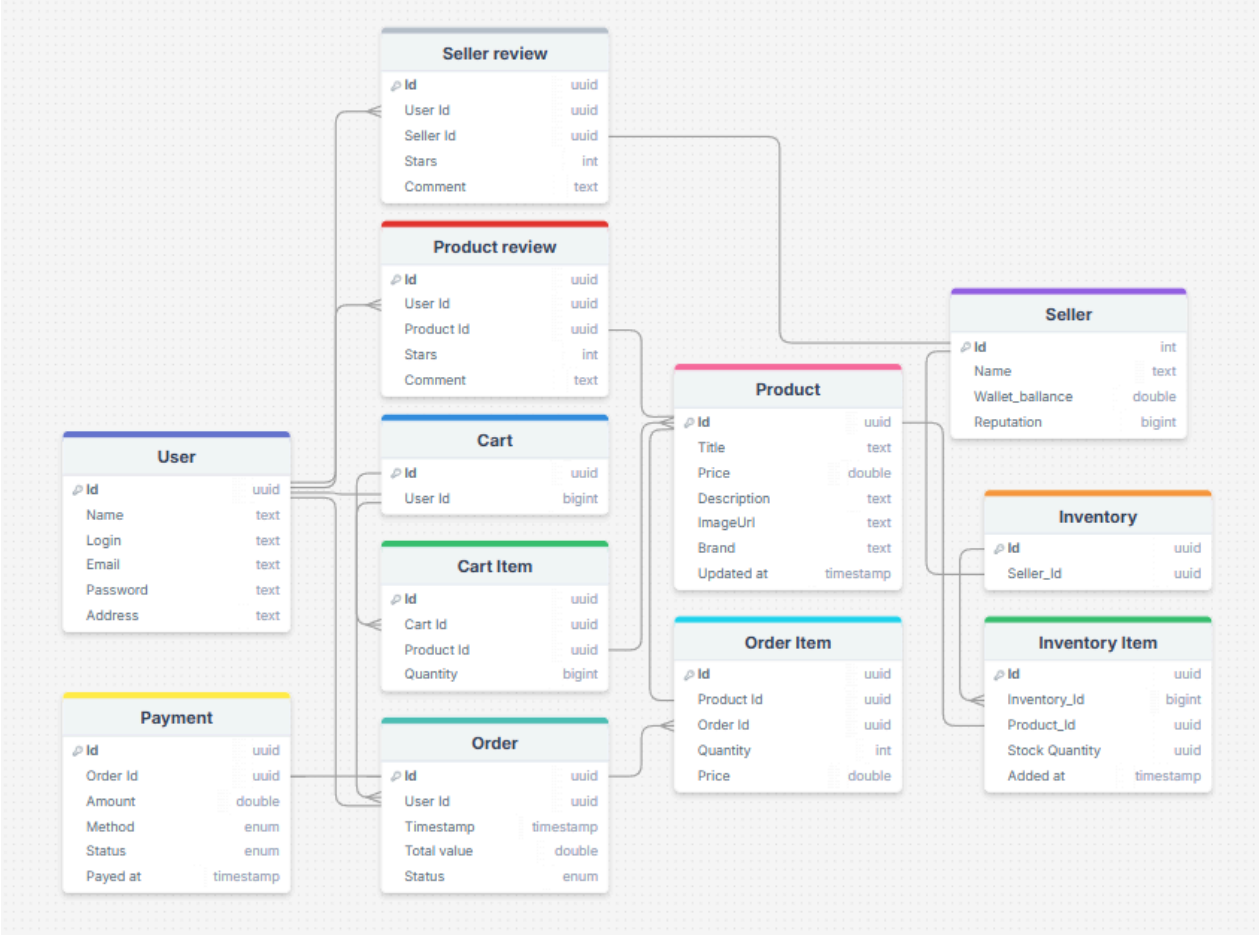
Meli Prototype Documentation

Projeto

Estudo dos elementos do projeto e suas interações.

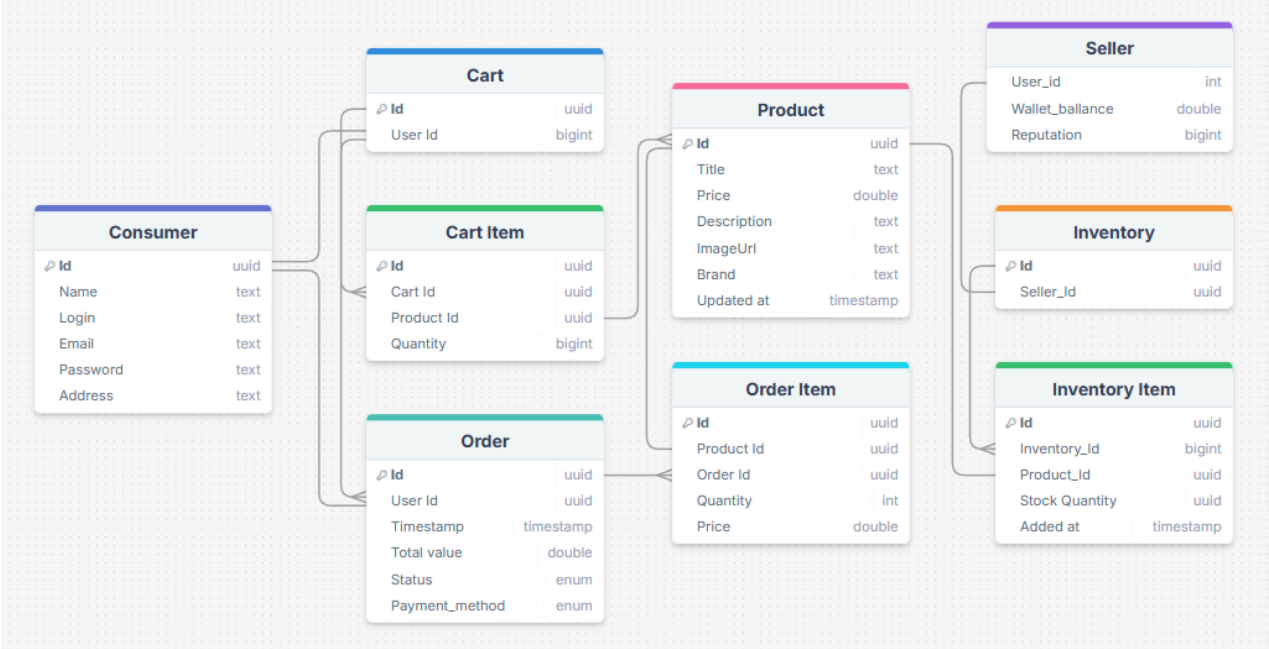
Completo

Estudo considerando todos os elementos contidos em um caso ideal:



Simplificado

Planejamento contemplando apenas o que o projeto irá considerar:



Decisão de desenvolvido

O Projeto possui apenas 3 classes de persistência: users.json, products.json e orders.json. Com o intuito de simplificar, ambos o inventário do Vendedor e o carrinho do Consumidor foram representados por mapas de ProductId e quantidade em suas respectivas classes.

Elementos do projeto

Frontend

Camada de apresentação da aplicação, acessada através do localhost:8080 pelo navegador. É a responsavel por renderizar a interface do usuário (UI), exibir informações e capturar interações do usuário.

Possui as seguintes interfaces:

- Geral:
  - header.html: Presente em todos as páginas, implementado com em conjunto com um header.js, é responsável por persistir os dados de login entre as diferentes páginas. Contém um campo de busca para pesquisar produtos.
  - sighUp.html: Página inicial para cadastro de usuário.
  - logIn.html: Página inicial para login do usuário
  - home.html: Lista todos os produtos.

    - detailProduct.html: Mostra os detalhes do produto selecionado. Permite a “compra com 1 click” e “adicionar produto ao carrinho”
- Do vendedor:
    - manageProducts.html: Possui duas tabelas, um balancete reportando todos os produtos do vendedor que foram vendidos e uma lista de produtos do vendedor. Na lista de produtor é possivel adicionar mais produtos e editar ou deletar produtos existentes.
    - addProduct.html: Acessada atravéz de manageProducts.html, é um formulário para o cadastro de novos produtos.
    - editProduct.html: Também acessada através de manageProducts.html, permite editar um produto específico.
- Do comprador:
    - cart.html: Lista os produtos que o usuário adicionou ao carrinho. Permite a finalização da compra
    - consumersOrders.html: Lista os pedidos do usuário.

Backend

Controllers

Camada de apresentação da API, é o ponto de entrada para as requisições HTTP no Backend: recebe as requisições do Frontend, as mapeia para métodos Java específicos e delega a lógica de negócio para cadama de Serviço. Utiliza a anotação @RestController que indica que a classe é um controlador e que seus métodos retornam dados diretamente (JSON), e mapeia metódos HTTP específicos para URLs, utilizando @GetMapping, @PostMapping, @PutMapping e @DeleteMapping.

O projeto possui os seguintes controladores (nomes autoexplicativos): CartController.java, OrderController.java, ProductController.java e UserController.java. Mais detalhes na sessao de API.

Services

A camada de Serviço contém a lógica de negócio principal e fluxo de trabalho da aplicação. Ela coordena as operações, aplica as regras de negócio e interage com a camada de Repositorio para acessar e manipular dados. Utiliza a anotação @Service que indica que a classe é um componente de serviço

O projeto possui os seguintes serviços (nomes autoexplicativos): CartService.java, OrderService.java, ProductService.java e UserService.java.

Repository

Responsável por abstrair a lógica de acesso a dados, interage diretamente com os arquivos JSON para realizar operações CRUD (Create, Read, Update e Delete). Utiliza a anotação @Repository que indica que a classe é um componente de acesso a dados. Suas funções também incluem converter dados do formato de armazenamento JSON para objetos Java e vice-versa.

O projeto possui os seguintes repositorios (nomes autoexplicativos): OrderRepository.java, ProductRepository.java e UserRepository.java

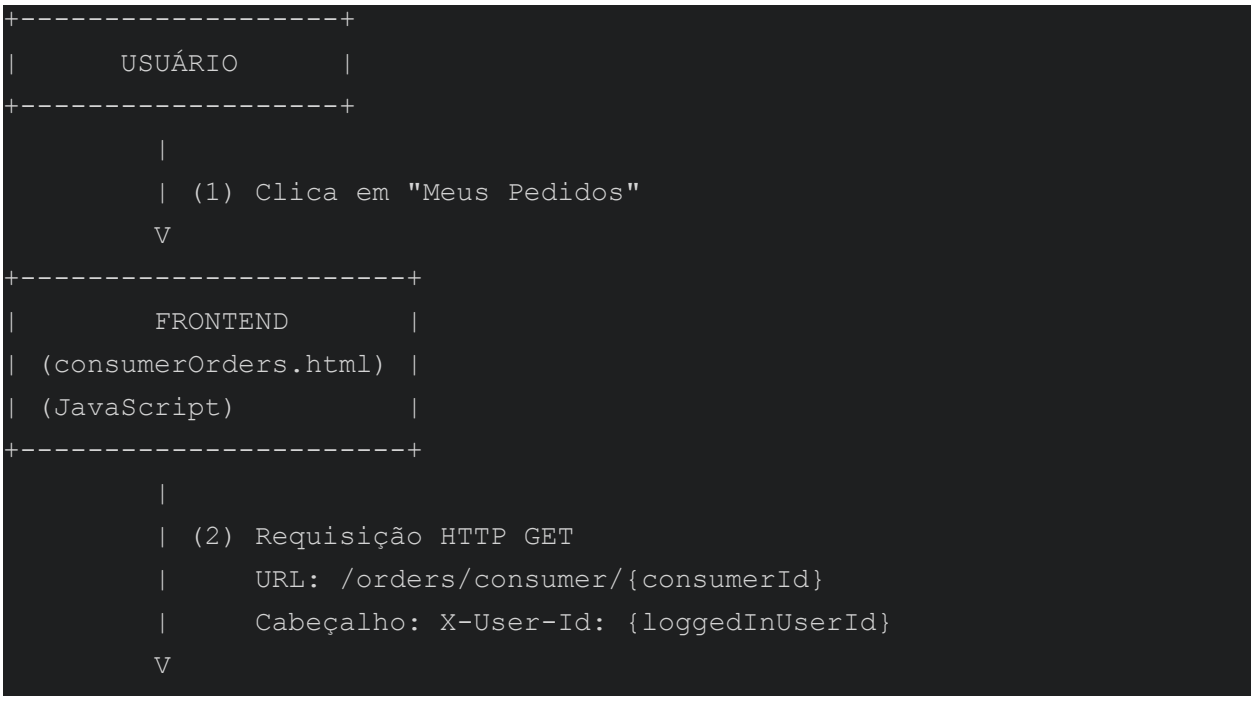
DTO

São classes simples, sem regras de negócio, usadas exclusivamente para transferir dados entre as diferentes camadas da aplicação ou entre o Backend e o Frontend, definem a estrutura de dados que serão enviados ou recebidos. Possibilita desacoplar os modelos de domínio (Order.java, Product.java, etc) da forma como os dados são apresentados ou recebidos pela API.

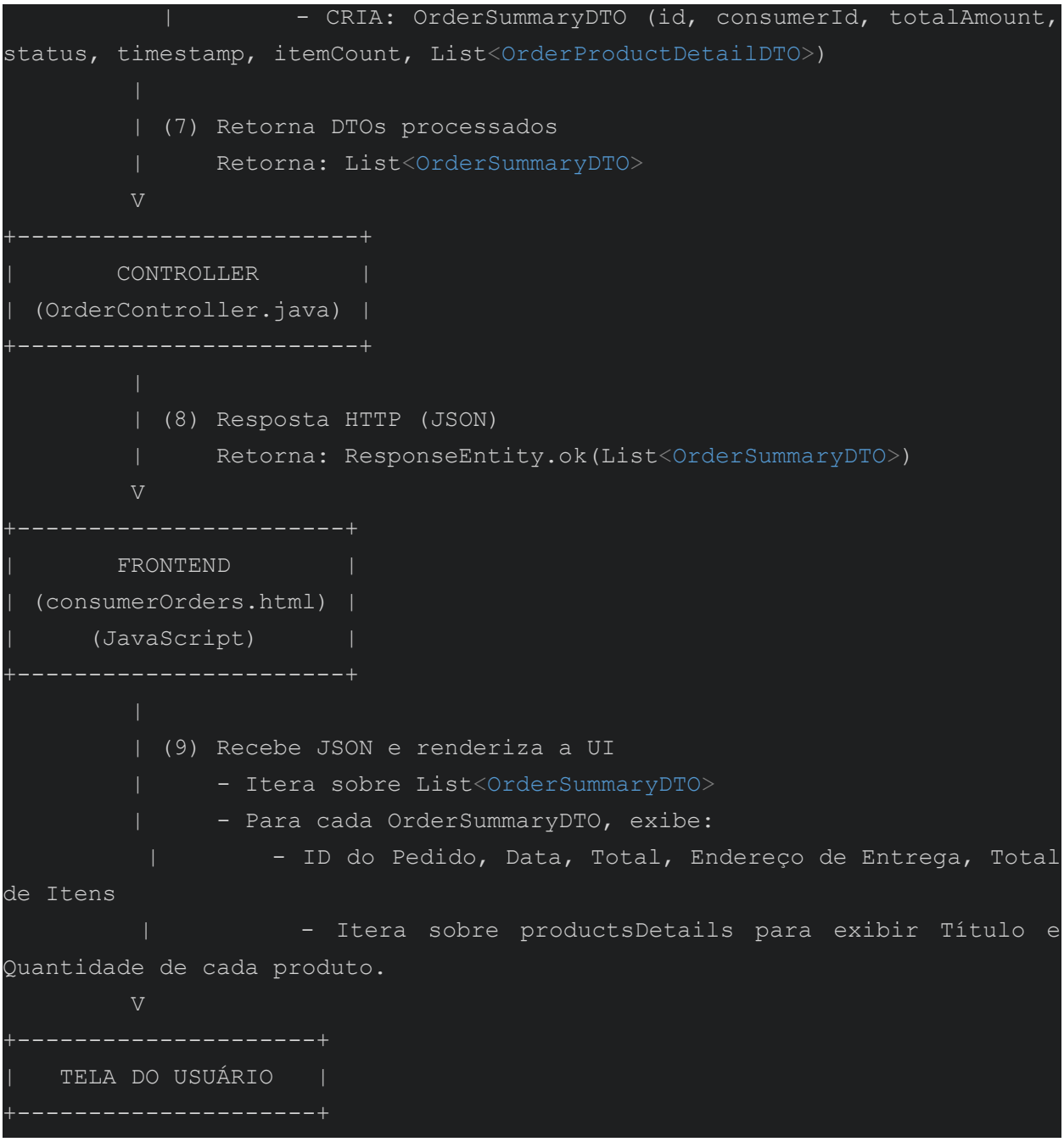
O projeto possui os seguintes DTOs (nomes autoexplicativos): BuyRequestDTO.java, LoginRequestDTO.java, OrderProductDetailDTO.java, OrderSummaryDTO.java, SellerOrderDTO.java e UserRequestDTO.java.

Exemplo de fluxo de trabalho dos elementos do projeto

Exemplo do fluxo de trabalho do Consumidor Visualizando seus pedidos.



```
+------------------------+
|       CONTROLLER       |
| (OrderController.java) |
+------------------------+
         |
         | (3) Delega lógica de negócio
         |     Chama: orderService.getOrdersByConsumerId(consumerId)
         |     (Passa: consumerId)
         V
+---------------------+
|       SERVICE       |
| (OrderService.java) |
+---------------------+
         |
         | (4) Acesso a dados brutos
         |     Chama: orderRepository.getAll()
         V
+------------------------+
|       REPOSITORY       |
| (OrderRepository.java) |
+------------------------+
         |
         | (5) Retorna dados brutos
         |     Retorna: List<Order> (lido de orders.json)
         V
+---------------------+
|       SERVICE       |
| (OrderService.java) |
+---------------------+
         |
         | (6) Processamento de Dados:
         |     - Filtra List<Order> por consumerId
         |     - PARA CADA Order:
         |       - Calcula itemCount (soma das quantidades no Map<Integer,
Integer> products)
         |       - PARA CADA produto no Map<Integer, Integer> products:
         |         - Chama: productService.getProductById(productId)
         |         - Obtém: Product (title, imageUrl, etc.)
          |             - CRIA: OrderProductDetailDTO (productId, title,
quantity, imageUrl)
```

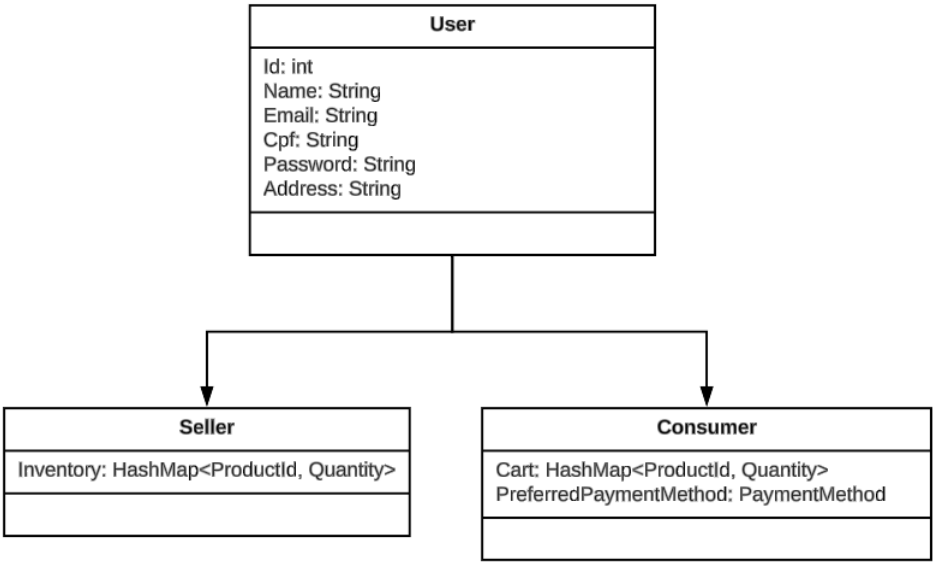

- CRIA: OrderSummaryDTO (id, consumerId, totalAmount, status, timestamp, itemCount, List<OrderProductDetailDTO>)
(7) Retorna DTOs processados
Retorna: List<OrderSummaryDTO>
CONTROLLER
(OrderController.java)
(8) Resposta HTTP (JSON)
Retorna: ResponseEntity.ok(List<OrderSummaryDTO>)
FRONTEND
(consumerOrders.html)
(JavaScript)
(9) Recebe JSON e renderiza a UI
- Itera sobre List<OrderSummaryDTO>
- Para cada OrderSummaryDTO, exibe:
- ID do Pedido, Data, Total, Endereço de Entrega, Total de Itens
- Itera sobre productsDetails para exibir Título e Quantidade de cada produto.
TELA DO USUÁRIO

Casos de uso

Premissas

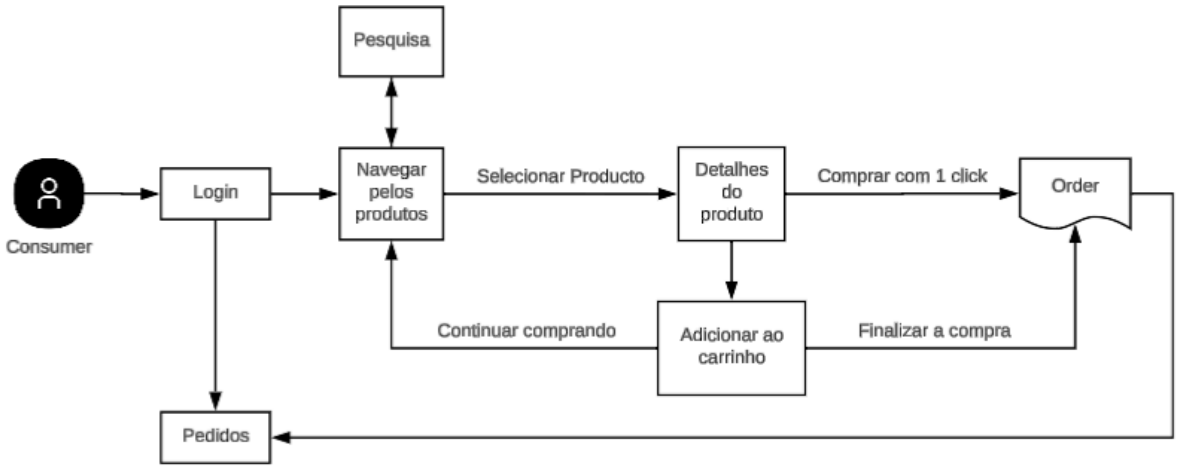
Agentes

No projeto desenvolvido foram considerados dois tipos de Agentes diferentes: O Consumidor (Consumer) e o Vendedor (Seller). Ambas representações são classes derivadas de Usuário (User).

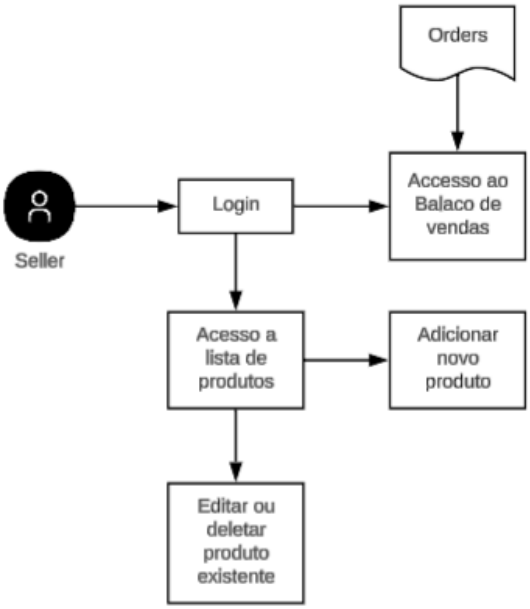


A diferenciação entre o tipo de conta é definida ao cadastrar novo usuário na página de signUp. Internamente ambos os usuários são registrados em users.json, com o atributo do tipo “type” definido como “seller” ou “consumer”.

Fluxograma de uso do Consumidor



Fluxograma de uso do Vendedor



APIs

User’s APIs

Estes endpoints lidam com o registro, autenticação e gerenciamento de usuários.

- **POST /users/register**
    - **Uso:** Para criar uma nova conta de usuário (seja consumer ou seller).
    - **Corpo da Requisição:** JSON contendo os dados do novo usuário (email, senha, tipo, etc.).
    - **Exemplo de Uso:** Quando um novo usuário preenche o formulário de registro em register.html e clica em "Criar Conta".
- **POST /users/login**
    - **Uso:** Para autenticar um usuário existente.
    - **Corpo da Requisição:** JSON contendo o email e a senha do usuário.
    - **Exemplo de Uso:** Quando um usuário preenche o formulário de login em login.html e clica em "Entrar". O backend valida as credenciais e, se bem-sucedido, retorna informações do usuário (como ID e tipo) para serem armazenadas no sessionStorage do frontend para sessões subsequentes.
- **GET /users/{id}**
    - **Uso:** Para obter os detalhes de um usuário específico pelo seu ID.
    - **Parâmetros de Path:** {id} (o ID do usuário).
    - **Exemplo de Uso:** Pode ser usado internamente pelo backend para verificar detalhes de usuários, ou em uma futura página de perfil do usuário no frontend.
- **GET /users**
    - **Uso:** Para obter uma lista de todos os usuários registrados no sistema.
    - **Exemplo de Uso:** Geralmente para fins administrativos ou de depuração.
- **PUT /users/{id}**
    - **Uso:** Para atualizar os dados de um usuário existente.
    - **Parâmetros de Path:** {id} (o ID do usuário a ser atualizado).
    - **Corpo da Requisição:** JSON com os dados atualizados do usuário.
    - **Exemplo de Uso:** Em uma futura página de "Configurações de Perfil" onde o usuário pode alterar seus dados.

Product's APIs

Estes endpoints lidam com o gerenciamento e a listagem de produtos.

- **GET /products**
    - **Uso:** Para obter uma lista de todos os produtos disponíveis no sistema. Pode aceitar parâmetros de consulta para busca por tags ou categoria.
    - **Parâmetros de Query (Opcional):** ?tags={tag1},{tag2} ou ?category={categoryName} para filtrar produtos.
    - **Exemplo de Uso:** Na página inicial (home.html) para exibir todos os produtos, ou quando o usuário usa a barra de pesquisa para filtrar produtos.
- **GET /products/{id}**
    - **Uso:** Para obter os detalhes de um produto específico pelo seu ID.
    - **Parâmetros de Path:** {id} (o ID do produto).

    - **Exemplo de Uso:** Em uma futura página de detalhes de produto.
- **POST /products**
    - **Uso:** Para adicionar um novo produto ao sistema.
    - **Corpo da Requisição:** JSON contendo os detalhes do novo produto (título, descrição, preço, estoque, etc.).
    - **Cabeçalho Necessário:** X-User-Id (o ID do vendedor que está adicionando o produto).
    - **Exemplo de Uso:** Quando um vendedor preenche o formulário em addProduct.html e clica em "Adicionar Produto".
- **PUT /products/{id}**
    - **Uso:** Para atualizar os detalhes de um produto existente.
    - **Parâmetros de Path:** {id} (o ID do produto a ser atualizado).
    - **Corpo da Requisição:** JSON com os dados atualizados do produto.
    - **Cabeçalho Necessário:** X-User-Id (o ID do vendedor que é proprietário do produto).
    - **Exemplo de Uso:** Quando um vendedor edita um produto na página manageProducts.html.
- **DELETE /products/{id}**
    - **Uso:** Para remover um produto do sistema.
    - **Parâmetros de Path:** {id} (o ID do produto a ser removido).
    - **Cabeçalho Necessário:** X-User-Id (o ID do vendedor que é proprietário do produto).
    - **Exemplo de Uso:** Quando um vendedor clica em "Excluir" ao lado de um produto na página manageProducts.html.
- **GET /products/seller/{sellerId}**
    - **Uso:** Para obter uma lista de todos os produtos cadastrados por um vendedor específico.
    - **Parâmetros de Path:** {sellerId} (o ID do vendedor).
    - **Cabeçalho Necessário:** X-User-Id (o ID do vendedor logado, para garantir que ele só veja seus próprios produtos).
    - **Exemplo de Uso:** Na página manageProducts.html, onde o vendedor gerencia seus produtos.

Order’s APIs

Estes endpoints lidam com a criação e visualização de pedidos para consumidores e vendedores.

- **GET /orders**
    - **Uso:** Para obter uma lista de todos os pedidos no sistema.
    - **Exemplo de Uso:** Geralmente para fins administrativos ou de depuração.
- **GET /orders/{id}**

    - **Uso:** Para obter os detalhes de um pedido específico pelo seu ID.
    - **Parâmetros de Path:** {id} (o ID do pedido).
    - **Exemplo de Uso:** Pode ser usado internamente ou para exibir detalhes de um pedido específico.
- **GET /orders/consumer/{consumerId}**
    - **Uso:** Para obter uma lista de todos os pedidos feitos por um consumidor específico.
    - **Parâmetros de Path:** {consumerId} (o ID do consumidor).
    - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor logado, para garantir que ele só veja seus próprios pedidos).
    - **Retorno:** List<OrderSummaryDTO> (contém um resumo do pedido e detalhes dos produtos comprados).
    - **Exemplo de Uso:** Na página consumerOrders.html, onde o consumidor visualiza seu histórico de compras.
- **POST /orders**
    - **Uso:** Para criar um ou mais novos pedidos após uma compra (checkout). Esta API **recebe os itens do carrinho do frontend**.
    - **Corpo da Requisição:** JSON contendo uma lista de BuyRequestDTOs, onde cada DTO especifica o productId e a quantity do item a ser comprado.
    - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor que está fazendo a compra).
    - **Exemplo de Uso:** Quando o consumidor finaliza a compra de itens na página cart.html (o JavaScript coleta os itens do carrinho e os envia para este endpoint).
- **PUT /orders/{id}**
    - **Uso:** Para atualizar os detalhes ou o status de um pedido existente.
    - **Parâmetros de Path:** {id} (o ID do pedido a ser atualizado).
    - **Corpo da Requisição:** JSON com os dados atualizados do pedido.
    - **Exemplo de Uso:** Pode ser usado por um vendedor para atualizar o status de um pedido (ex: de PENDING para COMPLETED).
- **DELETE /orders/{id}**
    - **Uso:** Para remover um pedido do sistema.
    - **Parâmetros de Path:** {id} (o ID do pedido a ser removido).
    - **Exemplo de Uso:** Geralmente para fins administrativos.
- **GET /orders/seller**
    - **Uso:** Para obter uma lista de pedidos que contêm produtos de um vendedor específico.
    - **Cabeçalho Necessário:** X-User-Id (o ID do vendedor logado).
    - **Retorno:** List<SellerOrderDTO> (contém um resumo do pedido e apenas os itens vendidos por aquele vendedor).
    - **Exemplo de Uso:** Na página de gerenciamento de pedidos do vendedor.

Cart’s APIs

Estes endpoints lidam com a manipulação do carrinho de compras de um consumidor no lado do servidor.

- **POST /cart/add**
  - **Uso:** Adiciona uma quantidade específica de um produto ao carrinho de um consumidor.
  - **Corpo da Requisição:** JSON contendo {"id": productId, "quantity": quantityToAdd}.
  - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor logado).
  - **Exemplo de Uso:** Quando um usuário clica em "Adicionar ao Carrinho" em uma página de produto (home.html ou productDetail.html). O JavaScript enviaria uma requisição para este endpoint.
- **PUT /cart/set-quantity**
  - **Uso:** Define a quantidade total de um produto específico no carrinho do consumidor. Pode ser usado para atualizar a quantidade ou remover um item (definindo a quantidade para 0).
  - **Corpo da Requisição:** JSON contendo {"id": productId, "quantity": newTotalQuantity}.
  - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor logado).
  - **Exemplo de Uso:** Na página do carrinho (cart.html), quando o usuário altera a quantidade de um item usando um seletor numérico, ou clica em um botão de "remover" que define a quantidade para zero.
- **DELETE /cart/remove/{productId}**
  - **Uso:** Remove completamente um produto específico do carrinho do consumidor.
  - **Parâmetros de Path:** {productId} (o ID do produto a ser removido).
  - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor logado).
  - **Exemplo de Uso:** Na página do carrinho (cart.html), quando o usuário clica em um ícone de "lixeira" para remover um item.
- **GET /cart**
  - **Uso:** Obtém todos os produtos atualmente no carrinho de um consumidor, incluindo detalhes completos do produto (título, preço, imagem, etc.).
  - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor logado).
  - **Retorno:** List<Map<String, Object>> (uma lista de mapas, onde cada mapa representa um item do carrinho com seus detalhes).
  - **Exemplo de Uso:** Na página cart.html, para carregar e exibir o conteúdo atual do carrinho do usuário.
- **DELETE /cart/clear**
  - **Uso:** Limpa todos os produtos do carrinho de um consumidor.
  - **Cabeçalho Necessário:** X-User-Id (o ID do consumidor logado).
  - **Exemplo de Uso:** Na página cart.html, se houver um botão "Limpar Carrinho", ou após o checkout bem-sucedido (embora o POST /orders já devesse lidar com a limpeza do carrinho após a criação do pedido).