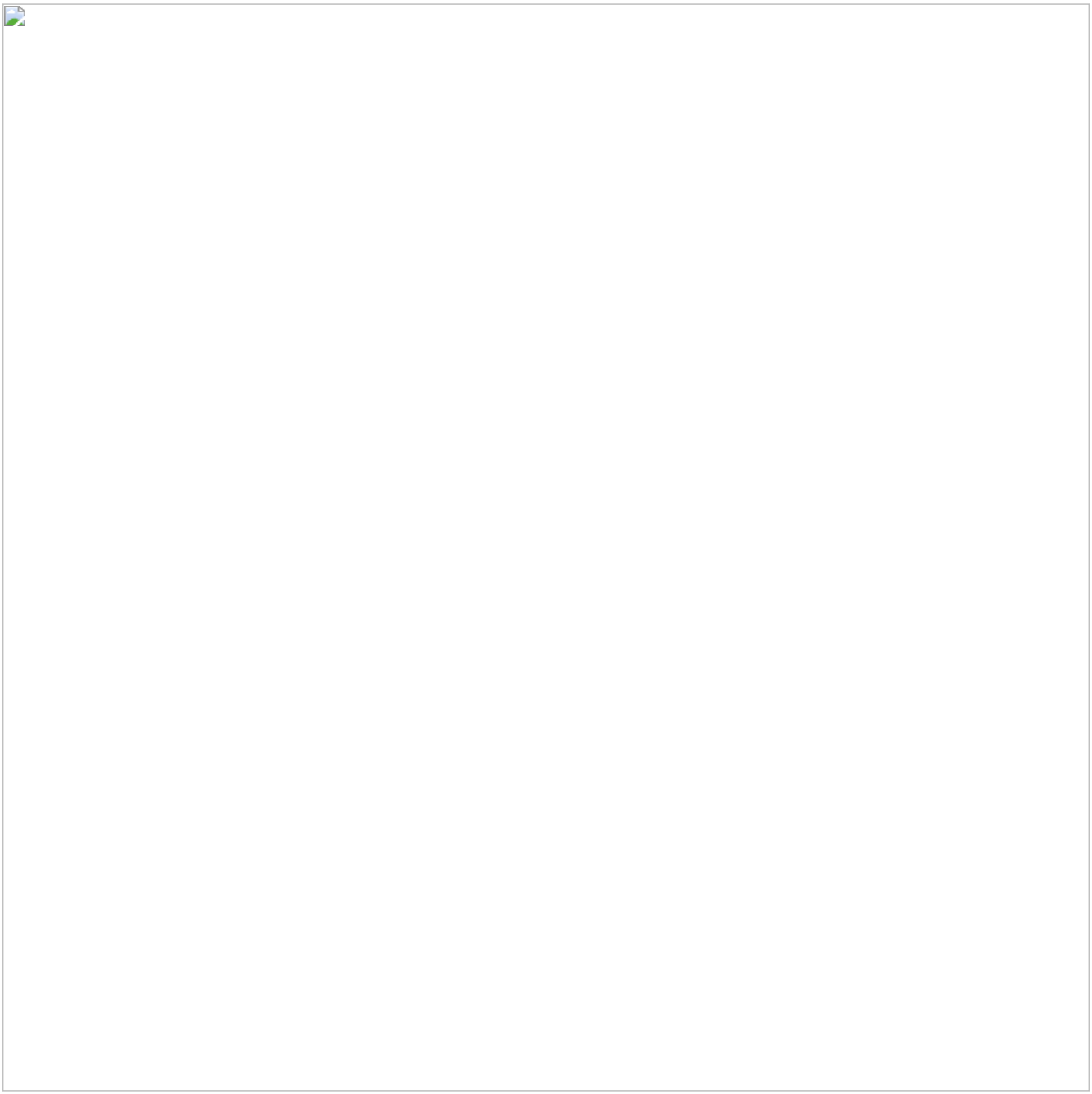
Prática 3 - API de Pedidos

▼ 1) Objetivo

Implementar um microserviço para criar pedidos e integrar com o `api-produto` para gerenciar o estoque.



Nesta prática vamos criar um microserviço de Pedidos, considerando que:

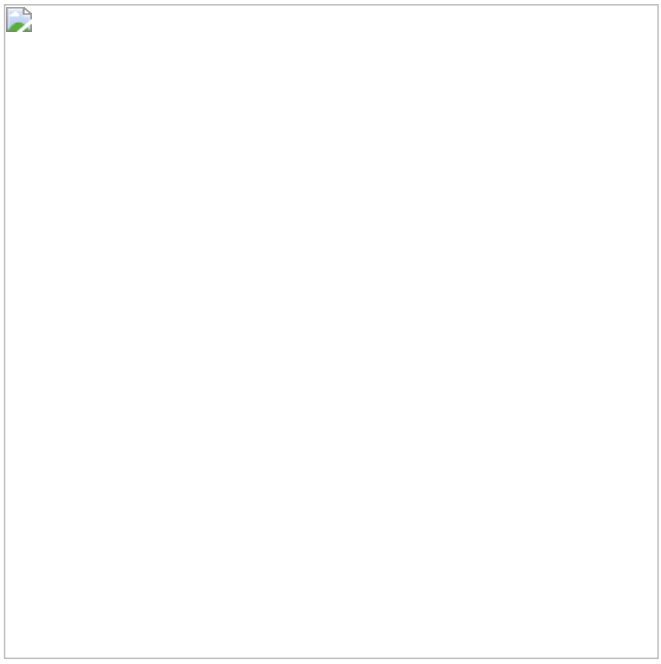
Um Pedido pode conter vários ItemPedido.

Cada ItemPedido referencia um Produto.

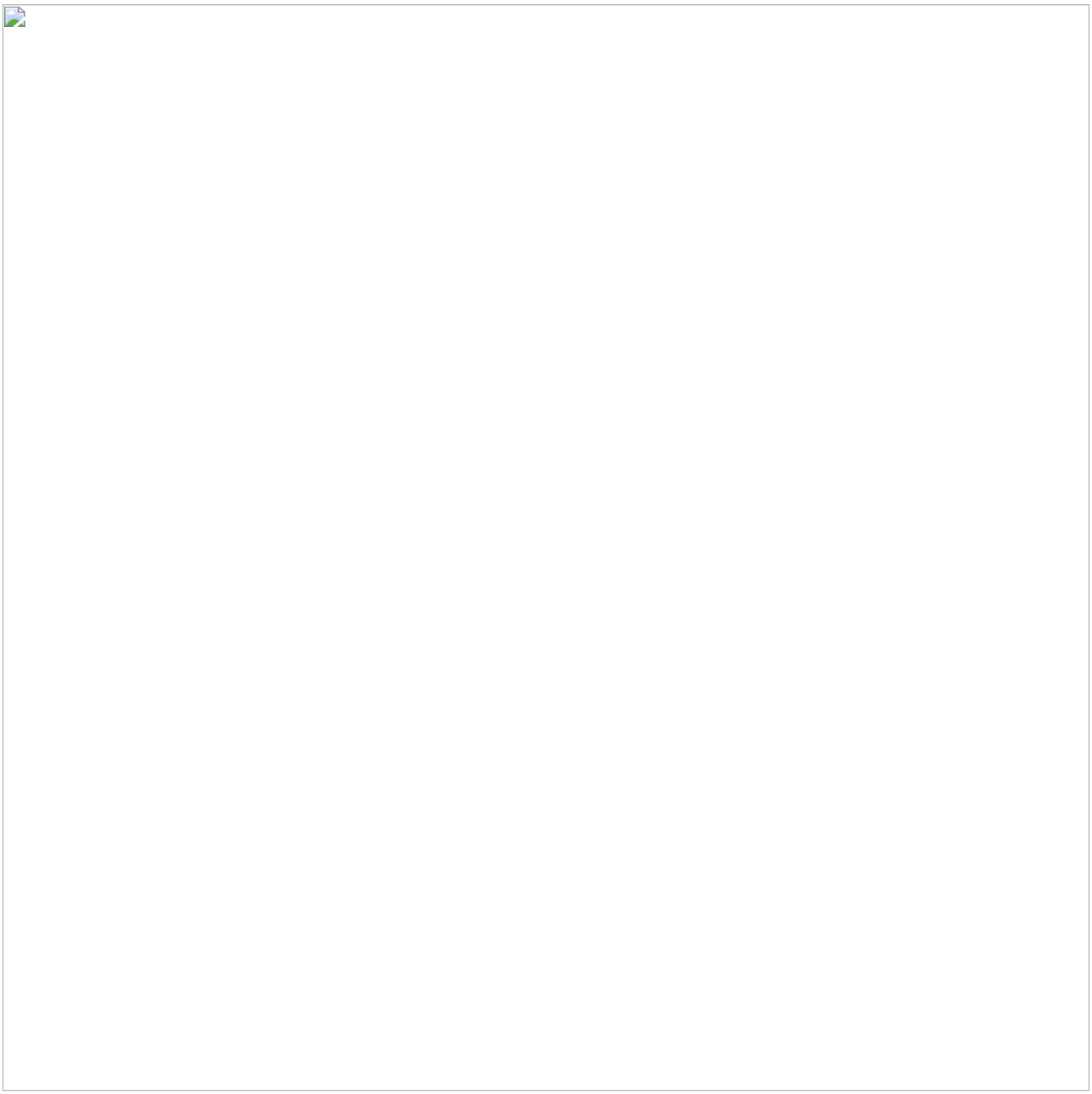
Estrutura

api-produtos
Entidade: Produto

api-pedidos
Entidades: Pedido e ItemPedido



▼ 2) Modificações em api-produto
Duas modificações serão feitas na api-produtos.



Passo 1: Crie um pacote `dtos` e adicione o `Record` abaixo:

```
public record ProdutoDTO(
        Long id,
        String description,
        Integer quantity,
        Double price,
        String category
        ) {}
```

Passo 2: Adicione os novos endpoints na classe `ProdutoController`

GET /produtos/{id}: Para obter os detalhes de um produto.

POST /produtos/estoque/baixa: Para atualizar o estoque após confirmar o pedido.

```
// Este endpoint já existe!
// Foi modificado para retornar um ProdutoDTO
@GetMapping(path = "/{id}")
public ResponseEntity<ProdutoDTO> getOne(@PathVariable(name = "id") Long idProduto){
    Produto produtoEncontrado = this.repository.findById(idProduto).orElse(null);
    if (produtoEncontrado == null)
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
    else{
        ProdutoDTO produtoDTO = new ProdutoDTO(
                produtoEncontrado.getId(),
                produtoEncontrado.getDescription(),
                produtoEncontrado.getQuantity(),
                produtoEncontrado.getPrice(),
                produtoEncontrado.getCategory()
        );
        return ResponseEntity.status(HttpStatus.OK).body(produtoDTO);
    }
}

// Endpoint para executar a baixa de estoque para vários produtos
@PostMapping("/estoque/baixa")
public ResponseEntity<Boolean> atualizarEstoque(@RequestBody List<ProdutoDTO> produtos){
    try {
        executarBaixa(produtos);
    } catch (Exception e){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(false);
    }
    return ResponseEntity.ok(true);
}

// Método para executar a baixa como uma transação no banco de dados
@Transactional
public void executarBaixa(List<ProdutoDTO> produtos){

    for (ProdutoDTO item : produtos) {
        // Verificando se produto existe
        Produto produto = this.repository.findById(item.id()).orElse(null);
        if (produto == null)
                throw new IllegalArgumentException("Produto não encontrado.");
        // Verifica se produto tem estoque suficiente
        int novaQuantidade = produto.getQuantity() - item.quantity();
        if (novaQuantidade < 0)
                throw new IllegalArgumentException("Produto com estoque insuficiente.");
    }

    // Faz a atualização dos produtos.
    for (ProdutoDTO item : produtos) {
        Produto produto = this.repository.findById(item.id()).orElse(null);
        int novaQuantidade = produto.getQuantity() - item.quantity();
        produto.setQuantity(novaQuantidade);
        this.repository.save(produto);
    }

}
```

Teste os novos endpoints:

localhost:8080/produtos/1

localhost:8080/produtos/estoque/baixa

```
// Payload para dar baixa nos itens 3 e 4, em 1 e 2 unidades do produto.
[
    {"id": 3, "quantity": 1},
    {"id": 4, "quantity": 2}
]
```

▼ 3) Criar projeto api-pedido

Passo 1:

Para criar o projeto acesse o site: https://start.spring.io/

Passo 2: configure o projeto de acordo com o descrito abaixo.

Project: Maven

Language: Java

Spring Boot: deixe a versão selecionada por padrão.

Group: br.edu.utfpr.nome_aluno

Artifact: api-pedido

Package name: deixe o padrão.

Java: no mínimo versão 17.

Passo 3: adicione as dependências

Spring Web

Lombok

H2

Spring Data JPA

OpenFeign

Passo 4: gere o arquivo .zip do projeto.

Clique em GENERATE e salve o arquivo no seu computador.

Descompacte o arquivo e use o IntelliJ para abrir a pasta do projeto.

Passo 5: configure a porta da api-pedido para 8081, no arquivo `src/main/resources/application.properties`

```
server.port=8081
```

▼ 4) Configurando H2

Passo 1:

No arquivo `src/main/resources/application.properties`, adicione as configurações para habilitar o H2 e definir o modo de funcionamento (por exemplo, em memória):

```
# Nome do banco de dados em memória
# spring.datasource.url=jdbc:h2:mem:pedidos_db
# Nome do banco de dados em arquivo
spring.datasource.url=jdbc:h2:file:./pedidos_db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.platform=h2
# Habilita o console do H2
spring.h2.console.enabled=true
# Caminho do console
spring.h2.console.path=/h2-console
# Atualiza o schema automaticamente (create, none)
spring.jpa.hibernate.ddl-auto=update
# Exibe SQL gerado
spring.jpa.show-sql=true
```

A configuração acima cria um arquivo pedidos_db.mv.db no diretório raiz da aplicação.

Passo 2:

Iniciar a aplicação e acessar o console do H2 em `http://localhost:8080/h2-console`.

Use as seguintes credenciais:

JDBC URL: `jdbc:h2:file:./pedidos_db`

Username: `sa`

Password: (deixe vazio)

▼ 5) Criando as entidades Pedido e ItemPedido

Entidade Pedido

Código da classe Pedido:

```
package br.edu.utfpr.aluno.api_pedido.models;
// imports ...

@Entity
@Getter
@Setter
public class Pedido {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String customerName;
    private String status; // CONFIRMADO ou CANCELADO

    @OneToMany(mappedBy = "pedido", cascade = CascadeType.ALL)
    private List<ItemPedido> items;
}
```

Passo 1:

Crie um pacote chamado models dentro do pacote api-pedido.

Caminho do pacote: br.edu.utfpr.aluno.api_pedido.models

Passo 2:

Crie a classe Pedido dentro do pacote models e adicione o código acima.

▼ Explicação

`@OneToMany(mappedBy = "pedido", cascade = CascadeType.ALL)`: Especifica que `Pedido` tem um relacionamento **1-N** com a entidade `ItemPedido`.
A propriedade `mappedBy` indica que o campo `pedido` em `ItemPedido` é a chave estrangeira.

O `cascade = CascadeType.ALL` permite que operações no `Pedido` (como persistência ou remoção) sejam propagadas para seus itens.

Entidade ItemPedido

Código da classe ItemPedido:

```
package br.edu.utfpr.aluno.api_pedido.models;
// imports ...

@Entity
@Getter
@Setter
public class ItemPedido {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Long productId;
    private Integer quantity;
    private Double price;

    @ManyToOne
    @JoinColumn(name = "pedido_id")
    @JsonIgnore // Interromper ciclo ao carregar dados
    private Pedido pedido;
}
```

Passo 1:

Crie a classe ItemPedido dentro do pacote models e adicione o código acima.

▼ Explicação

`@ManyToOne`: Define o relacionamento **muitos-para-um** com a entidade `Pedido`, indicando que muitos itens pertencem a um único pedido.
`@JoinColumn(name = "pedido_id")`: Especifica a coluna `pedido_id` no banco de dados, que representa a chave estrangeira ligando `ItemPedido` a `Pedido`.

▼ 6) Criando as interfaces Repository
Passo 1: Crie um novo pacote chamado repositories.

Passo 2: Crie as interfaces abaixo.

```
package br.edu.utfpr.api_pedido.repositories;

import br.edu.utfpr.api_pedido.model.Pedido;
import org.springframework.data.jpa.repository.JpaRepository;

// Use a IDE para criar a interface, deixe que a IDE gere os imports e package
public interface PedidoRepository extends JpaRepository<Pedido, Long> { }

package br.edu.utfpr.api_pedido.repositories;

import br.edu.utfpr.api_pedido.model.ItemPedido;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ItemPedidoRepository extends JpaRepository<ItemPedido, Long> {}
```

Execute o projeto para verificar se banco de dados foi criado.

▼ 7) Criando DTOs (Data Transfer Objects)
O cliente da API vai enviar os dados de pedidos por meio de DTOs.

Passo 1: Crie um novo pacote chamado dtos.

Passo 2: Crie os records abaixo.

```
public record PedidoDTO(String customerName, List<ItemPedidoDTO> items) {}

public record ItemPedidoDTO(Long productId, Integer quantity) {}
```

▼ Explicação

No Java, DTOs podem ser implementados com `records` **(Java 16).**

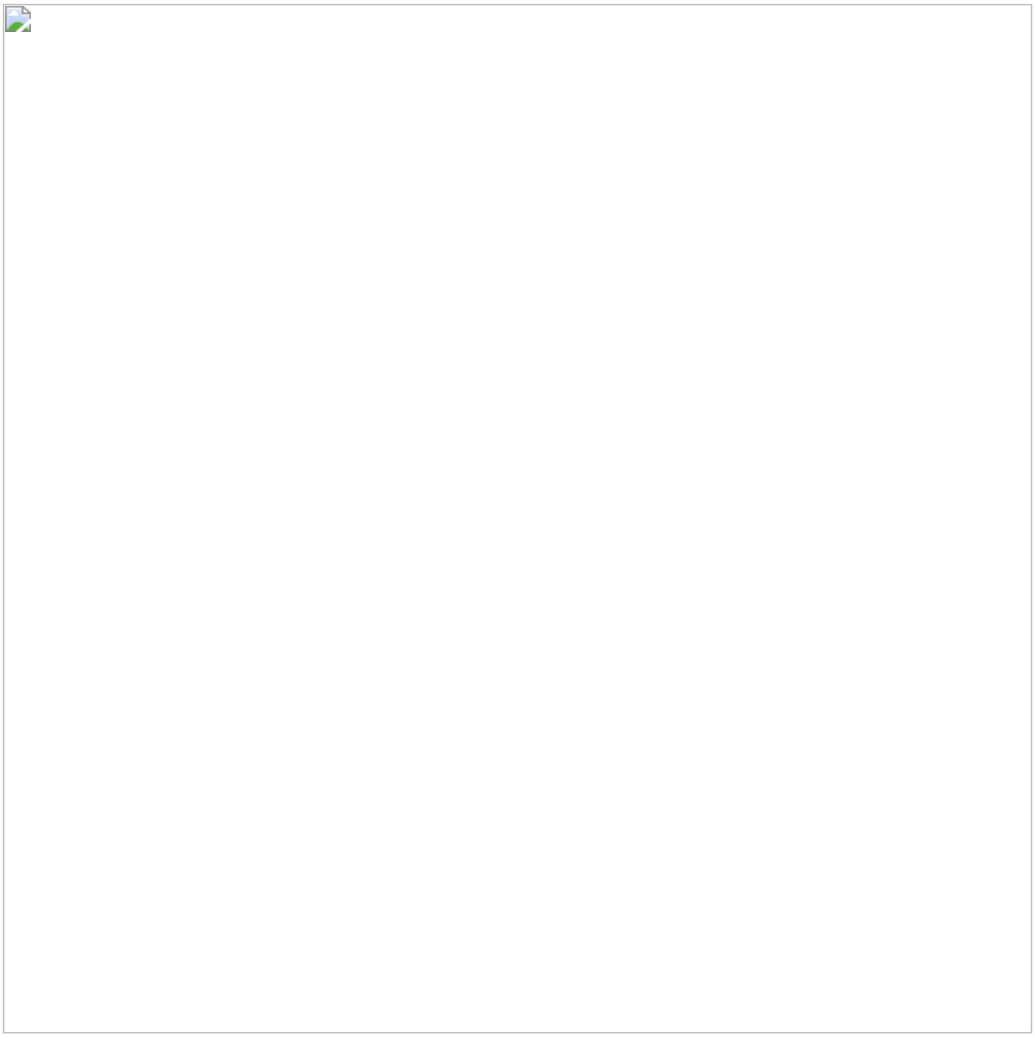
Um `record` é um tipo especial de classe, projetado para armazenar dados de forma concisa e imutável.
Ele reduz o boilerplate associado à criação de classes simples que apenas contêm dados.

Benefícios

Clareza: Facilita a compreensão de quais dados estão sendo enviados e recebidos nas requisições.
Segurança: Evita a exposição de dados sensíveis ou internos do sistema.
Manutenção: Permite alterar a estrutura de dados das APIs sem alterar a lógica de negócio interna.

▼ 8) Criando PedidoService

O
`PedidoService` é a camada de serviço responsável pela lógica de negócios relacionada ao gerenciamento de pedidos e a interação com o microserviço de produtos para verificação e atualização de estoque.



Fonte: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5

Passo 1: Crie um novo pacote chamado `services`.

Passo 2: Crie a classe abaixo.

```
package br.edu.utfpr.api_pedido.services;
// imports...

@Service
public class PedidoService {
    // injetando o repository para persistir pedidos no H2
    private PedidoRepository pedidoRepository;

    public PedidoService(PedidoRepository pedidoRepository) {
        this.pedidoRepository = pedidoRepository;
    }

    // Método para adicionar um Pedido no H2
    public Pedido createPedido(PedidoDTO pedidoDTO){
        // Cria pedido
        Pedido pedido = new Pedido();
        pedido.setCustomerName(pedidoDTO.customerName());
        pedido.setStatus("PENDENTE");

        // Cria lista de itens do pedido
        List<ItemPedido> itens = new ArrayList<>();

        // Processa a lista de itens DTO
        for (ItemPedidoDTO itemDTO : pedidoDTO.items()){

            ItemPedido itemPedido = new ItemPedido();
            itemPedido.setProductId(itemDTO.productId());
            itemPedido.setQuantity(itemDTO.quantity());
            itemPedido.setPrice(0.0);
            itemPedido.setPedido(pedido); // Relaciona o ItemPedido com o Pedido
            itens.add(itemPedido);
        }

        // Relaciona o Pedido com os ItemPedido
        pedido.setItems(itens);

        return this.pedidoRepository.save(pedido);
    }

    // Recuperar todos os pedidos
    public List<Pedido> getAll(){
        return this.pedidoRepository.findAll();
    }

}
```

`@Service`:

> Essa anotação indica que a classe `PedidoService` é um **componente de serviço** do Spring.
>
> Ela permite que o Spring registre essa classe como um bean gerenciado pelo container, responsável por conter a lógica de negócios.

▼ 9) Criando PedidoController
Passo 1: Crie um novo pacote chamado controllers.

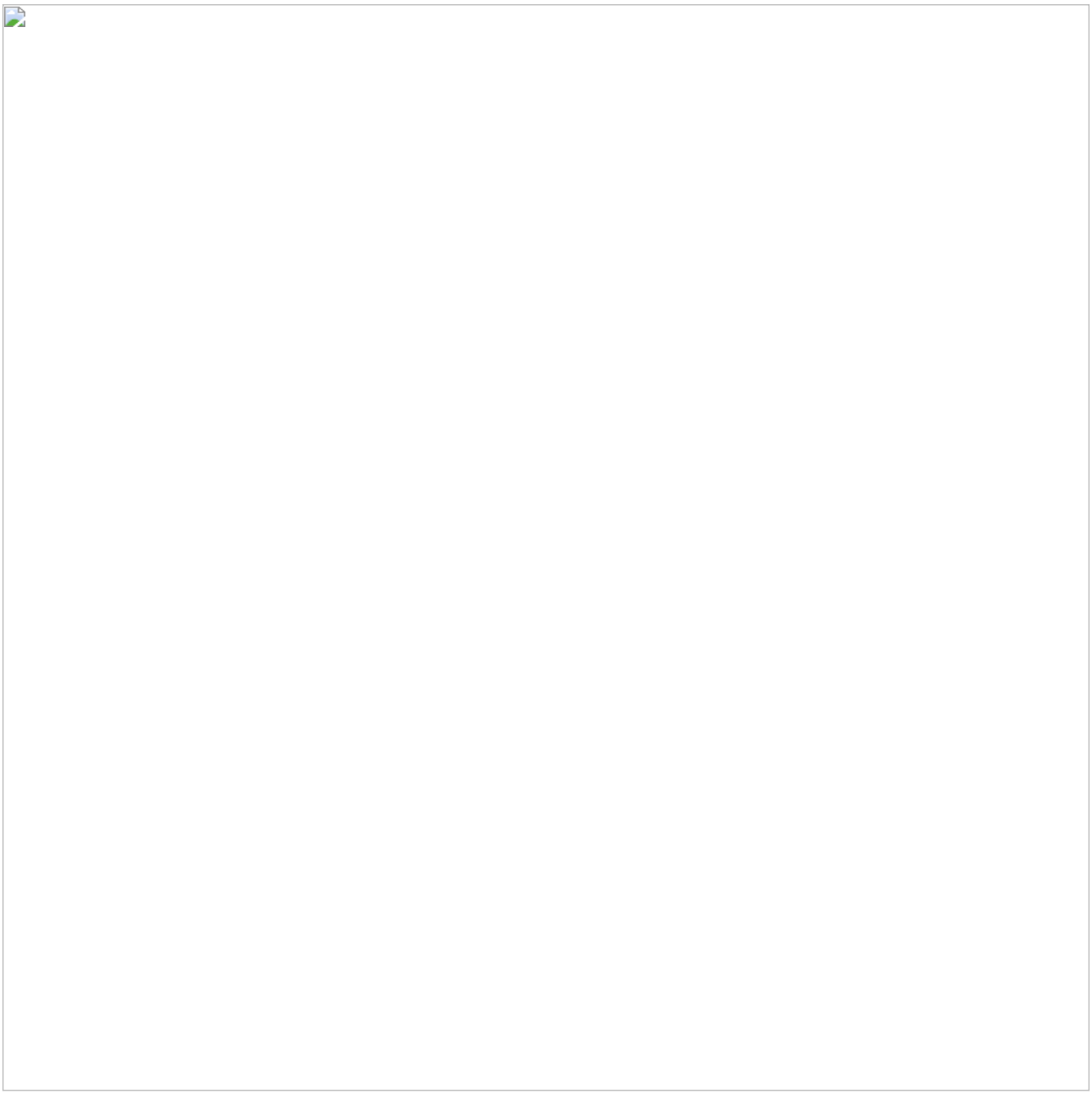
Passo 2: Crie a classe abaixo.

```
@RestController
@RequestMapping("/pedidos")
public class PedidoController {

    private PedidoService pedidoService;

    public PedidoController(PedidoService pedidoService) {
        this.pedidoService = pedidoService;
    }

    @PostMapping
                public ResponseEntity<Pedido> createPedido(@RequestBody PedidoDTO pedidoDTO){
        Pedido pedido = pedidoService.criaPedido(pedidoDTO);
        if (pedido != null)
            return ResponseEntity.ok(pedido);
        else
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
    }

    @GetMapping
    public ResponseEntity<List<Pedido>> getAll(){
        return ResponseEntity.ok(this.pedidoService.getAll());
    }

}
```

Teste a API enviando o JSON abaixo para localhost:8080/pedidos

```
// Criar um pedido com dois itens
{
    "customerName": "João",
    "items": [
        {"productId": 1, "quantity": 1},
        {"productId": 2, "quantity": 1}
    ]
}
```

▼ 10) Usando Feign para Comunicação entre Microserviços



Passo 1:
Habilite o Feign Client na sua aplicação, anotando a classe principal da aplicação.

```
@SpringBootApplication
@EnableFeignClients // Adicione esta anotação
public class ApiPedidoApplication {
	public static void main(String[] args) {
		SpringApplication.run(ApiPedidoApplication.class, args);
	}
}
```

Passo 2: No pacote `dtos` crie o `Record` abaixo:

```
public record ProdutoDTO(
	Long id,
	String description,
	Integer quantity,
	Double price,
	String category
	) {}
```

Passo 3: Crie uma nova interface no pacote `services` conforme abaixo.

```
@FeignClient(name="api-produto", url="localhost:8080")
public interface ProdutoFeignClient {

    @GetMapping("/produtos/{id}")
    ProdutoDTO getProdutoById(@PathVariable Long id);

    @PostMapping("/produtos/estoque/baixa")
    Boolean atualizarEstoque(@RequestBody List<ProdutoDTO> itens);

}
```

A **interface** **`ProdutoFeignClient`** é usada para facilitar a comunicação entre o microserviço `api-pedido` e o microserviço `api-produto` via HTTP, sem precisar implementar manualmente chamadas REST.

O Feign é usado para criar um cliente declarativo, tornando a comunicação entre microserviços mais simples e legível.

Passo 4: Modifique `PedidoService` para chamar a api-produto.

```
@Service
public class PedidoService {
    // injetando o repository para persistir pedidos no H2
    private PedidoRepository pedidoRepository;
    private ProdutoFeignClient produtoFeignClient;

    public PedidoService(PedidoRepository pedidoRepository,
                         ProdutoFeignClient produtoFeignClient) {
        this.pedidoRepository = pedidoRepository;
        this.produtoFeignClient = produtoFeignClient;
    }

    // Método para adicionar um Pedido no H2
    public Pedido createPedido(PedidoDTO pedidoDTO){
        // Cria pedido
        Pedido pedido = new Pedido();
        pedido.setCustomerName(pedidoDTO.customerName());
        pedido.setStatus("PENDENTE");

        // Cria lista de itens do pedido
        List<ItemPedido> itens = new ArrayList<>();

        // Processa a lista de itens DTO
        for (ItemPedidoDTO itemDTO : pedidoDTO.items()){

            // Chama a api-produto
            ProdutoDTO produtoEstoque = produtoFeignClient.getProdutoById(itemDTO.productId());

            // O produto não existe, interrompe a criação do pedido
            if (produtoEstoque == null) return null;

            // Não há estoque suficiente, interrompe a criação do pedido.
            if (produtoEstoque.quantity() - itemDTO.quantity() < 0){
                return null;
            }

            ItemPedido itemPedido = new ItemPedido();
            itemPedido.setProductId(itemDTO.productId());
            itemPedido.setQuantity(itemDTO.quantity());
            itemPedido.setPrice(produtoEstoque.price()); // Atualizar price!
            itemPedido.setPedido(pedido); // Relaciona o ItemPedido com o Pedido
            itens.add(itemPedido);
        }

        // Relaciona o Pedido com os ItemPedido
        pedido.setItems(itens);

        return this.pedidoRepository.save(pedido);
    }

    // Recuperar todos os pedidos
    public List<Pedido> getAll(){
        return this.pedidoRepository.findAll();
    }

}
```

Passo 5: teste o novo endpoint:

```
// Criar um pedido com dois itens
{
    "customerName": "João",
    "items": [
        {"productId": 1, "quantity": 1},
        {"productId": 2, "quantity": 1}
    ]
}

GET // Recuperar todos os pedidos
localhost:8081/pedidos
```

▼ 11) Endpoint de confirmação de pedido

O objetivo do endpoint é confirmar ou cancelar o pedido.

Se não há estoque suficiente, o pedido tem status CANCELADO.

Se há estoque, o pedido tem status CONFIRMADO.

Passo 1: crie um endpoint para confirmar o pedido criado.

```
@Service
public class PedidoService {

    // ...

    public Pedido confirmaPedido(Long pedidoId){
        Pedido pedido = this.pedidoRepository.findById(pedidoId).orElse(null);

        // Não encontrou pedido
        if (pedido == null) return null;

        // Cria uma lista de ProdutoDTO para enviar para api-produto baixar estoque
        List<ProdutoDTO> produtoList = new ArrayList<>();
        for (ItemPedido itemPedido : pedido.getItems()){
            produtoList.add(new ProdutoDTO(
                    itemPedido.getProductId(),
                    null,
                    itemPedido.getQuantity(),
                    null,
                    null));
        }

        // Envia lista para api-produto
        //  se ok, pedido CONFIRMADO
        //  se não foi possível baixa em um item, pedido CANCELADO
        if (produtoFeignClient.atualizarEstoque(produtoList)){
            pedido.setStatus("CONFIRMADO");
        } else {
            pedido.setStatus("CANCELADO");
        }

        this.pedidoRepository.save(pedido);
        return pedido;
    }

    // ...

}
```

Passo 2: Modifique `PedidoController` para adicionar novo endpoint.

```
@GetMapping("/{id}/confirmacao")
public ResponseEntity<Pedido> confirmarPedido(@PathVariable("id") Long pedidoId){
    Pedido pedido = pedidoService.confirmaPedido(pedidoId);
    if (pedido != null) return ResponseEntity.ok(pedido);
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
}
```

Passo 3: teste os endpoints de PedidoController

```
// Criar um pedido com dois itens
{
    "customerName": "João",
    "items": [
        {"productId": 1, "quantity": 1},
        {"productId": 2, "quantity": 1}
    ]
}

GET // Confirmar um pedido
localhost:8081/pedidos/1/confirmacao

GET // Recuperar todos os pedidos
localhost:8081/pedidos
```