

GABARITO COMPLETO - Repositories e Services no Projeto Delivery

Preparação: Entidade Cliente

```
package com.deliverytech.entity;
```

```
import jakarta.persistence.*;
```

```
import lombok.Data;
```

```
import java.util.List;
```

```
@Entity
```

```
@Data
```

```
public class Cliente {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    private String nome;
```

```
    private String email;
```

```
    private String telefone;
```

```
    private String endereco;
```

```
    private boolean ativo;
```

```
    @OneToMany(mappedBy = "cliente")
```

```
    private List<Pedido> pedidos;
```

```
}
```

Entidade Restaurante

```
package com.deliverytech.entity;
```

```
import jakarta.persistence.*;
```

```
import lombok.Data;
```

```

import java.math.BigDecimal;

import java.util.List;

@Entity
@Data
public class Restaurante {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;

    private String categoria;

    private String endereco;

    private String telefone;

    private BigDecimal taxaEntrega;

    private boolean ativo;

    @OneToMany(mappedBy = "restaurante")
    private List<Produto> produtos;

    @OneToMany(mappedBy = "restaurante")
    private List<Pedido> pedidos;
}

```

Entidade Pedido

```

package com.deliverytech.entity;

import com.deliverytech.enums.StatusPedido;
import jakarta.persistence.*;
import lombok.Data;

import java.math.BigDecimal;

```

```

import java.time.LocalDateTime;

import java.util.List;


@Entity
@Data
public class Pedido {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;


    private LocalDateTime dataPedido;
    private String enderecoEntrega;
    private BigDecimal subtotal;
    private BigDecimal taxaEntrega;
    private BigDecimal valorTotal;


    @Enumerated(EnumType.STRING)
    private StatusPedido status;


    @ManyToOne
    @JoinColumn(name = "cliente_id")
    private Cliente cliente;


    @ManyToOne
    @JoinColumn(name = "restaurante_id")
    private Restaurante restaurante;


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

```

Entidade Produto

```

package com.deliverytech.entity;

import jakarta.persistence.*;
import lombok.Data;

import java.math.BigDecimal;

@Entity
@Data
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    private String categoria;
    private boolean disponivel;

    @ManyToOne
    @JoinColumn(name = "restaurante_id")
    private Restaurante restaurante;
}

```

Entidade ItemPedido

```

package com.deliverytech.entity;

import jakarta.persistence.*;
import lombok.Data;

import java.math.BigDecimal;

```

@Entity

@Data

```
public class ItemPedido {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    private int quantidade;  
    private BigDecimal precoUnitario;  
    private BigDecimal subtotal;  
  
    @ManyToOne  
    @JoinColumn(name = "pedido_id")  
    private Pedido pedido;  
  
    @ManyToOne  
    @JoinColumn(name = "produto_id")  
    private Produto produto;  
}
```

Enums StatusPdido

package com.deliverytech.enums;

```
public enum StatusPedido {  
    PENDENTE,  
    CONFIRMADO,  
    PREPARANDO,  
    SAIU_PARA_ENTREGA,  
    ENTREGUE,  
    CANCELADO  
}
```

Atividade 1: Configuração dos Repositories

ClienteRepository - Implementação Completa

```
package com.deliverytech.delivery.repository;

import com.deliverytech.delivery.entity.Cliente;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface ClienteRepository extends JpaRepository<Cliente, Long> {

    // Buscar cliente por email (método derivado)
    Optional<Cliente> findByEmail(String email);

    // Verificar se email já existe
    boolean existsByEmail(String email);

    // Buscar clientes ativos
    List<Cliente> findByAtivoTrue();

    // Buscar clientes por nome (contendo)
    List<Cliente> findByNomeContainingIgnoreCase(String nome);

    // Buscar clientes por telefone
    Optional<Cliente> findByTelefone(String telefone);
```

```

// Query customizada - clientes com pedidos
@Query("SELECT DISTINCT c FROM Cliente c JOIN c.pedidos p WHERE c.ativo = true")
List<Cliente> findClientesComPedidos();

// Query nativa - clientes por cidade
@Query(value = "SELECT * FROM clientes WHERE endereco LIKE %:cidade% AND ativo =
true",
    nativeQuery = true)
List<Cliente> findByCidade(@Param("cidade") String cidade);

// Contar clientes ativos
@Query("SELECT COUNT(c) FROM Cliente c WHERE c.ativo = true")
Long countClientesAtivos();
}

```

RestauranteRepository - Implementação Completa

```

package com.deliverytech.delivery.repository;

import com.deliverytech.delivery.entity.Restaurante;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;

@Repository
public interface RestauranteRepository extends JpaRepository<Restaurante, Long> {

```

```
// Buscar por nome
```

```
Optional<Restaurante> findByNome(String nome);
```

```
// Buscar restaurantes ativos
```

```
List<Restaurante> findByAtivoTrue();
```

```
// Buscar por categoria
```

```
List<Restaurante> findByCategoriaAndAtivoTrue(String categoria);
```

```
// Buscar por nome contendo (case insensitive)
```

```
List<Restaurante> findByNomeContainingIgnoreCaseAndAtivoTrue(String nome);
```

```
// Buscar por avaliação mínima
```

```
List<Restaurante> findByAvaliacaoGreaterThanOrEqualToAndAtivoTrue(BigDecimal avaliacao);
```

```
// Ordenar por avaliação (descendente)
```

```
List<Restaurante> findByAtivoTrueOrderByAvaliacaoDesc();
```

```
// Query customizada - restaurantes com produtos
```

```
@Query("SELECT DISTINCT r FROM Restaurante r JOIN r.produtos p WHERE r.ativo = true")
```

```
List<Restaurante> findRestaurantesComProdutos();
```

```
// Buscar por faixa de taxa de entrega
```

```
@Query("SELECT r FROM Restaurante r WHERE r.taxaEntrega BETWEEN :min AND :max AND  
r.ativo = true")
```

```
List<Restaurante> findByTaxaEntregaBetween(@Param("min") BigDecimal min,  
@Param("max") BigDecimal max);
```

```
// Categorias disponíveis
```

```
@Query("SELECT DISTINCT r.categoria FROM Restaurante r WHERE r.ativo = true ORDER BY  
r.categoria")
```

```
List<String> findCategoriasDisponiveis();
```



```
}
```

ProdutoRepository - Implementação Completa

```
package com.deliverytech.delivery.repository;
```

```
import com.deliverytech.delivery.entity.Produto;
```

```
import com.deliverytech.delivery.entity.Restaurante;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
import org.springframework.data.jpa.repository.Query;
```

```
import org.springframework.data.repository.query.Param;
```

```
import org.springframework.stereotype.Repository;
```

```
import java.math.BigDecimal;
```

```
import java.util.List;
```

```
@Repository
```

```
public interface ProdutoRepository extends JpaRepository<Produto, Long> {
```

```
    // Buscar produtos por restaurante
```

```
    List<Produto> findByRestauranteAndDisponivelTrue(Restaurante restaurante);
```

```
    // Buscar produtos por restaurante ID
```

```
    List<Produto> findByRestauranteIdAndDisponivelTrue(Long restauranteId);
```

```
    // Buscar por categoria
```

```
    List<Produto> findByCategoriaAndDisponivelTrue(String categoria);
```

```
    // Buscar por nome contendo
```

```
    List<Produto> findByNomeContainingIgnoreCaseAndDisponivelTrue(String nome);
```

```
    // Buscar por faixa de preço
```

```
List<Produto> findByPrecoBetweenAndDisponivelTrue(BigDecimal precoMin, BigDecimal precoMax);
```

```
// Buscar produtos mais baratos que um valor
```

```
List<Produto> findByPrecoLessThanEqualAndDisponivelTrue(BigDecimal preco);
```

```
// Ordenar por preço
```

```
List<Produto> findByDisponivelTrueOrderByPrecoAsc();
```

```
List<Produto> findByDisponivelTrueOrderByPrecoDesc();
```

```
// Query customizada - produtos mais vendidos
```

```
@Query("SELECT p FROM Produto p JOIN p.itensPedido ip " +
```

```
    "GROUP BY p ORDER BY COUNT(ip) DESC")
```

```
List<Produto> findProdutosMaisVendidos();
```

```
// Buscar por restaurante e categoria
```

```
@Query("SELECT p FROM Produto p WHERE p.restaurante.id = :restaurantId " +
```

```
    "AND p.categoria = :categoria AND p.disponivel = true")
```

```
List<Produto> findByRestauranteAndCategoria(@Param("restaurantId") Long restaurantId,  
    @Param("categoria") String categoria);
```

```
// Contar produtos por restaurante
```

```
@Query("SELECT COUNT(p) FROM Produto p WHERE p.restaurante.id = :restaurantId AND  
p.disponivel = true")
```

```
Long countByRestaurantId(@Param("restaurantId") Long restaurantId);
```

```
}
```

PedidoRepository - Implementação Completa

```
package com.deliverytech.delivery.repository;
```

```
import com.deliverytech.delivery.entity.Pedido;
```

```
import com.deliverytech.delivery.entity.Cliente;
```

```
import com.deliverytech.delivery.entity.StatusPedido;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
```

```
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.List;
```

```
@Repository
```

```
public interface PedidoRepository extends JpaRepository<Pedido, Long> {
```

```
    // Buscar pedidos por cliente
```

```
    List<Pedido> findByClienteOrderByDataPedidoDesc(Cliente cliente);
```

```
    // Buscar pedidos por cliente ID
```

```
    List<Pedido> findByClientIdOrderByDataPedidoDesc(Long clientId);
```

```
    // Buscar por status
```

```
    List<Pedido> findByStatusOrderByDataPedidoDesc(StatusPedido status);
```

```
    // Buscar por número do pedido
```

```
    Pedido findByNumeroPedido(String numeroPedido);
```

```
    // Buscar pedidos por período
```

```
    List<Pedido> findByDataPedidoBetweenOrderByDataPedidoDesc(LocalDateTime inicio,
LocalDateTime fim);
```

```
    // Buscar pedidos do dia
```

```
    @Query("SELECT p FROM Pedido p WHERE DATE(p.dataPedido) = CURRENT_DATE ORDER BY
p.dataPedido DESC")
```

```
    List<Pedido> findPedidosDodia();
```

```

// Buscar pedidos por restaurante

@Query("SELECT p FROM Pedido p WHERE p.restaurante.id = :restaurantId ORDER BY
p.dataPedido DESC")

List<Pedido> findByRestaurantId(@Param("restaurantId") Long restaurantId);

// Relatório - pedidos por status

@Query("SELECT p.status, COUNT(p) FROM Pedido p GROUP BY p.status")

List<Object[]> countPedidosByStatus();

// Pedidos pendentes (para dashboard)

@Query("SELECT p FROM Pedido p WHERE p.status IN ('PENDENTE', 'CONFIRMADO',
'PREPARANDO') " +

    "ORDER BY p.dataPedido ASC")

List<Pedido> findPedidosPendentes();

// Valor total de vendas por período

@Query("SELECT SUM(p.valorTotal) FROM Pedido p WHERE p.dataPedido BETWEEN :inicio
AND :fim " +

    "AND p.status NOT IN ('CANCELADO')")

BigDecimal calcularVendasPorPeriodo(@Param("inicio") LocalDateTime inicio,

    @Param("fim") LocalDateTime fim);
}

```



Atividade 2: Implementação dos Services



ClienteService - Implementação Completa

```

package com.deliverytech.delivery.service;

import com.deliverytech.delivery.entity.Cliente;
import com.deliverytech.delivery.repository.ClienteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

```

```
import java.util.List;
import java.util.Optional;

@Service
@Transactional

public class ClienteService {

    @Autowired
    private ClienteRepository clienteRepository;

    /**
     * Cadastrar novo cliente
     */
    public Cliente cadastrar(Cliente cliente) {
        // Validar email único
        if (clienteRepository.existsByEmail(cliente.getEmail())) {
            throw new IllegalArgumentException("Email já cadastrado: " + cliente.getEmail());
        }

        // Validações de negócio
        validarDadosCliente(cliente);

        // Definir como ativo por padrão
        cliente.setAtivo(true);

        return clienteRepository.save(cliente);
    }

    /**
     * Buscar cliente por ID
```

```

*/
@Transactional(readOnly = true)
public Optional<Cliente> buscarPorId(Long id) {
    return clienteRepository.findById(id);
}

/**
 * Buscar cliente por email
 */
@Transactional(readOnly = true)
public Optional<Cliente> buscarPorEmail(String email) {
    return clienteRepository.findByEmail(email);
}

/**
 * Listar todos os clientes ativos
 */
@Transactional(readOnly = true)
public List<Cliente> listarAtivos() {
    return clienteRepository.findByAtivoTrue();
}

/**
 * Atualizar dados do cliente
 */
public Cliente atualizar(Long id, Cliente clienteAtualizado) {
    Cliente cliente = buscarPorId(id)
        .orElseThrow(() -> new IllegalArgumentException("Cliente não encontrado: " + id));

    // Verificar se email não está sendo usado por outro cliente
    if (!cliente.getEmail().equals(clienteAtualizado.getEmail()) &&

```

```
        clienteRepository.existsByEmail(clienteAtualizado.getEmail())) {  
            throw new IllegalArgumentException("Email já cadastrado: " +  
clienteAtualizado.getEmail());  
        }  
    }
```

```
        // Atualizar campos  
        cliente.setNome(clienteAtualizado.getNome());  
        cliente.setEmail(clienteAtualizado.getEmail());  
        cliente.setTelefone(clienteAtualizado.getTelefone());  
        cliente.setEndereco(clienteAtualizado.getEndereco());  
  
        return clienteRepository.save(cliente);  
    }
```

```
/**  
 * Inativar cliente (soft delete)  
 */  
public void inativar(Long id) {  
    Cliente cliente = buscarPorId(id)  
        .orElseThrow(() -> new IllegalArgumentException("Cliente não encontrado: " + id));  
  
    cliente.inativar();  
    clienteRepository.save(cliente);  
}
```

```
/**  
 * Buscar clientes por nome  
 */  
@Transactional(readOnly = true)  
public List<Cliente> buscarPorNome(String nome) {  
    return clienteRepository.findByNomeContainingIgnoreCase(nome);  
}
```

```

    }

    /**
     * Validações de negócio
     */
    private void validarDadosCliente(Cliente cliente) {
        if (cliente.getNome() == null || cliente.getNome().trim().isEmpty()) {
            throw new IllegalArgumentException("Nome é obrigatório");
        }

        if (cliente.getEmail() == null || cliente.getEmail().trim().isEmpty()) {
            throw new IllegalArgumentException("Email é obrigatório");
        }

        if (cliente.getNome().length() < 2) {
            throw new IllegalArgumentException("Nome deve ter pelo menos 2 caracteres");
        }
    }
}

```

RestauranteService - Implementação Completa

```

package com.deliverytech.delivery.service;

import com.deliverytech.delivery.entity.Restaurante;
import com.deliverytech.delivery.repository.RestauranteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;

```



```
import java.util.List;

import java.util.Optional;

@Service
@Transactional

public class RestauranteService {

    @Autowired

    private RestauranteRepository restauranteRepository;

    /**
     * Cadastrar novo restaurante
     */
    public Restaurante cadastrar(Restaurante restaurante) {
        // Validar nome único
        if (restauranteRepository.findByNome(restaurante.getNome()).isPresent()) {
            throw new IllegalArgumentException("Restaurante já cadastrado: " +
restaurante.getNome());
        }

        validarDadosRestaurante(restaurante);

        restaurante.setAtivo(true);

        return restauranteRepository.save(restaurante);
    }

    /**
     * Buscar por ID
     */
    @Transactional(readOnly = true)
    public Optional<Restaurante> buscarPorId(Long id) {
```

```

        return restauranteRepository.findById(id);
    }

    /**
     * Listar restaurantes ativos
     */
    @Transactional(readOnly = true)
    public List<Restaurante> listarAtivos() {
        return restauranteRepository.findByAtivoTrue();
    }

    /**
     * Buscar por categoria
     */
    @Transactional(readOnly = true)
    public List<Restaurante> buscarPorCategoria(String categoria) {
        return restauranteRepository.findByCategoriaAndAtivoTrue(categoria);
    }

    /**
     * Atualizar restaurante
     */
    public Restaurante atualizar(Long id, Restaurante restauranteAtualizado) {
        Restaurante restaurante = buscarPorId(id)
            .orElseThrow(() -> new IllegalArgumentException("Restaurante não encontrado: " + id));

        // Verificar nome único (se mudou)
        if (!restaurante.getNome().equals(restauranteAtualizado.getNome()) &&
            restauranteRepository.findByNome(restauranteAtualizado.getNome()).isPresent()) {
            throw new IllegalArgumentException("Nome já cadastrado: " +
                restauranteAtualizado.getNome());
        }
    }

```

```

    }

    restaurante.setNome(restauranteAtualizado.getNome());
    restaurante.setCategoria(restauranteAtualizado.getCategoria());
    restaurante.setEndereco(restauranteAtualizado.getEndereco());
    restaurante.setTelefone(restauranteAtualizado.getTelefone());
    restaurante.setTaxaEntrega(restauranteAtualizado.getTaxaEntrega());

    return restauranteRepository.save(restaurante);
}

/**
 * Inativar restaurante
 */
public void inativar(Long id) {
    Restaurante restaurante = buscarPorId(id)
        .orElseThrow(() -> new IllegalArgumentException("Restaurante não encontrado: " + id));

    restaurante.setAtivo(false);
    restauranteRepository.save(restaurante);
}

private void validarDadosRestaurante(Restaurante restaurante) {
    if (restaurante.getNome() == null || restaurante.getNome().trim().isEmpty()) {
        throw new IllegalArgumentException("Nome é obrigatório");
    }

    if (restaurante.getTaxaEntrega() != null &&
        restaurante.getTaxaEntrega().compareTo(BigDecimal.ZERO) < 0) {
        throw new IllegalArgumentException("Taxa de entrega não pode ser negativa");
    }
}

```

```
}  
}
```

✅ ProdutoService - Implementação Completa

```
package com.deliverytech.delivery.service;  
  
import com.deliverytech.delivery.entity.Produto;  
import com.deliverytech.delivery.entity.Restaurante;  
import com.deliverytech.delivery.repository.ProdutoRepository;  
import com.deliverytech.delivery.repository.RestauranteRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.math.BigDecimal;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional  
public class ProdutoService {  
  
    @Autowired  
    private ProdutoRepository produtoRepository;  
  
    @Autowired  
    private RestauranteRepository restauranteRepository;  
  
    /**  
     * Cadastrar novo produto  
     */  
    public Produto cadastrar(Produto produto, Long restaurantId) {
```

```
    Restaurante restaurante = restauranteRepository.findById(restaurantId)
        .orElseThrow(() -> new IllegalArgumentException("Restaurante não encontrado: " +
restaurantId));
```

```
    validarDadosProduto(produto);
```

```
    produto.setRestaurante(restaurante);
```

```
    produto.setDisponivel(true);
```

```
    return produtoRepository.save(produto);
```

```
}
```

```
/**
```

```
 * Buscar por ID
```

```
 */
```

```
@Transactional(readOnly = true)
```

```
public Optional<Produto> buscarPorId(Long id) {
```

```
    return produtoRepository.findById(id);
```

```
}
```

```
/**
```

```
 * Listar produtos por restaurante
```

```
 */
```

```
@Transactional(readOnly = true)
```

```
public List<Produto> listarPorRestaurante(Long restaurantId) {
```

```
    return produtoRepository.findByRestaurantIdAndDisponivelTrue(restaurantId);
```

```
}
```

```
/**
```

```
 * Buscar por categoria
```

```
 */
```

```

@Transactional(readOnly = true)

public List<Produto> buscarPorCategoria(String categoria) {
    return produtoRepository.findByCategoriaAndDisponivelTrue(categoria);
}

/**
 * Atualizar produto
 */
public Produto atualizar(Long id, Produto produtoAtualizado) {
    Produto produto = buscarPorId(id)
        .orElseThrow(() -> new IllegalArgumentException("Produto não encontrado: " + id));

    validarDadosProduto(produtoAtualizado);

    produto.setNome(produtoAtualizado.getNome());
    produto.setDescricao(produtoAtualizado.getDescricao());
    produto.setPreco(produtoAtualizado.getPreco());
    produto.setCategoria(produtoAtualizado.getCategoria());

    return produtoRepository.save(produto);
}

/**
 * Alterar disponibilidade
 */
public void alterarDisponibilidade(Long id, boolean disponivel) {
    Produto produto = buscarPorId(id)
        .orElseThrow(() -> new IllegalArgumentException("Produto não encontrado: " + id));

    produto.setDisponivel(disponivel);
    produtoRepository.save(produto);
}

```

```

    }

    /**
     * Buscar por faixa de preço
     */
    @Transactional(readOnly = true)
    public List<Produto> buscarPorFaixaPreco(BigDecimal precoMin, BigDecimal precoMax) {
        return produtoRepository.findByPrecoBetweenAndDisponivelTrue(precoMin, precoMax);
    }

    private void validarDadosProduto(Produto produto) {
        if (produto.getNome() == null || produto.getNome().trim().isEmpty()) {
            throw new IllegalArgumentException("Nome é obrigatório");
        }

        if (produto.getPreco() == null || produto.getPreco().compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Preço deve ser maior que zero");
        }
    }
}

```

PedidoService - Implementação Completa

```

package com.deliverytech.delivery.service;

import com.deliverytech.delivery.entity.*;
import com.deliverytech.delivery.repository.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.util.List;

```

```
import java.util.Optional;
```

```
@Service
```

```
@Transactional
```

```
public class PedidoService {
```

```
    @Autowired
```

```
    private PedidoRepository pedidoRepository;
```

```
    @Autowired
```

```
    private ClienteRepository clienteRepository;
```

```
    @Autowired
```

```
    private RestauranteRepository restauranteRepository;
```

```
    @Autowired
```

```
    private ProdutoRepository produtoRepository;
```

```
    /**
```

```
     * Criar novo pedido
```

```
     */
```

```
    public Pedido criarPedido(Long clienteld, Long restauranteld) {
```

```
        Cliente cliente = clienteRepository.findById(clienteld)
```

```
        .orElseThrow(() -> new IllegalArgumentException("Cliente não encontrado: " +  
clienteld));
```

```
        Restaurante restaurante = restauranteRepository.findById(restauranteld)
```

```
        .orElseThrow(() -> new IllegalArgumentException("Restaurante não encontrado: " +  
restauranteld));
```

```
        if (!cliente.isAtivo()) {
```

```
            throw new IllegalArgumentException("Cliente inativo não pode fazer pedidos");
```



```

    }

    if (!restaurante.getAtivo()) {
        throw new IllegalArgumentException("Restaurante não está disponível");
    }

    Pedido pedido = new Pedido();
    pedido.setCliente(cliente);
    pedido.setRestaurante(restaurante);
    pedido.setStatus(StatusPedido.PENDENTE);

    return pedidoRepository.save(pedido);
}

/**
 * Adicionar item ao pedido
 */
public Pedido adicionarItem(Long pedidoid, Long produtoid, Integer quantidade) {
    Pedido pedido = buscarPorId(pedidoid)
        .orElseThrow(() -> new IllegalArgumentException("Pedido não encontrado: " +
            pedidoid));

    Produto produto = produtoRepository.findById(produtoid)
        .orElseThrow(() -> new IllegalArgumentException("Produto não encontrado: " +
            produtoid));

    if (!produto.getDisponivel()) {
        throw new IllegalArgumentException("Produto não disponível: " + produto.getNome());
    }

    if (quantidade <= 0) {
        throw new IllegalArgumentException("Quantidade deve ser maior que zero");
    }
}

```

```

    }

    // Verificar se produto pertence ao mesmo restaurante do pedido
    if (!produto.getRestaurante().getId().equals(pedido.getRestaurante().getId())) {
        throw new IllegalArgumentException("Produto não pertence ao restaurante do
pedido");
    }

    ItemPedido item = new ItemPedido();
    item.setPedido(pedido);
    item.setProduto(produto);
    item.setQuantidade(quantidade);
    item.setPrecoUnitario(produto.getPreco());
    item.calcularSubtotal();

    pedido.adicionarItem(item);

    return pedidoRepository.save(pedido);
}

/**
 * Confirmar pedido
 */
public Pedido confirmarPedido(Long pedidoid) {
    Pedido pedido = buscarPorId(pedidoid)
        .orElseThrow(() -> new IllegalArgumentException("Pedido não encontrado: " +
pedidoid));

    if (pedido.getStatus() != StatusPedido.PENDENTE) {
        throw new IllegalArgumentException("Apenas pedidos pendentes podem ser
confirmados");
    }
}

```

```

        if (pedido.getItems().isEmpty()) {
            throw new IllegalArgumentException("Pedido deve ter pelo menos um item");
        }

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

    /**
     * Buscar por ID
     */
    @Transactional(readOnly = true)
    public Optional<Pedido> buscarPorId(Long id) {
        return pedidoRepository.findById(id);
    }

    /**
     * Listar pedidos por cliente
     */
    @Transactional(readOnly = true)
    public List<Pedido> listarPorCliente(Long clienteId) {
        return pedidoRepository.findByClienteIdOrderByDataPedidoDesc(clienteId);
    }

    /**
     * Buscar por número do pedido
     */
    @Transactional(readOnly = true)
    public Optional<Pedido> buscarPorNumero(String numeroPedido) {
        return Optional.ofNullable(pedidoRepository.findByNumeroPedido(numeroPedido));
    }

```

```

    }

    /**
     * Cancelar pedido
     */
    public Pedido cancelarPedido(Long pedidoid, String motivo) {
        Pedido pedido = buscarPorId(pedidoid)
            .orElseThrow(() -> new IllegalArgumentException("Pedido não encontrado: " +
                pedidoid));

        if (pedido.getStatus() == StatusPedido.ENTREGUE) {
            throw new IllegalArgumentException("Pedido já entregue não pode ser cancelado");
        }

        if (pedido.getStatus() == StatusPedido.CANCELADO) {
            throw new IllegalArgumentException("Pedido já está cancelado");
        }

        pedido.setStatus(StatusPedido.CANCELADO);
        if (motivo != null && !motivo.trim().isEmpty()) {
            pedido.setObservacoes(pedido.getObservacoes() + " | Cancelado: " + motivo);
        }

        return pedidoRepository.save(pedido);
    }
}

```



Atividade 3: Controllers REST



ClienteController - Implementação Completa

```

package com.deliverytech.delivery.controller;

import com.deliverytech.delivery.entity.Cliente;

```

```

import com.deliverytech.delivery.service.ClienteService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;

import jakarta.validation.Valid;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/clientes")

@CrossOrigin(origins = "**")

public class ClienteController {

    @Autowired

    private ClienteService clienteService;

    /**
     * Cadastrar novo cliente
     */

    @PostMapping

    public ResponseEntity<?> cadastrar(@Valid @RequestBody Cliente cliente) {

        try {

            Cliente clienteSalvo = clienteService.cadastrar(cliente);

            return ResponseEntity.status(HttpStatus.CREATED).body(clienteSalvo);

        } catch (IllegalArgumentException e) {

            return ResponseEntity.badRequest().body("Erro: " + e.getMessage());

        } catch (Exception e) {

            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)

                .body("Erro interno do servidor");

        }

    }

}

```

```
}  
}
```

```
/**
```

```
 * Listar todos os clientes ativos
```

```
 */
```

```
@GetMapping
```

```
public ResponseEntity<List<Cliente>> listar() {  
    List<Cliente> clientes = clienteService.listarAtivos();  
    return ResponseEntity.ok(clientes);  
}
```

```
/**
```

```
 * Buscar cliente por ID
```

```
 */
```

```
@GetMapping("/{id}")
```

```
public ResponseEntity<?> buscarPorId(@PathVariable Long id) {  
    Optional<Cliente> cliente = clienteService.buscarPorId(id);  
  
    if (cliente.isPresent()) {  
        return ResponseEntity.ok(cliente.get());  
    } else {  
        return ResponseEntity.notFound().build();  
    }  
}
```

```
/**
```

```
 * Atualizar cliente
```

```
 */
```

```
@PutMapping("/{id}")
```

```
public ResponseEntity<?> atualizar(@PathVariable Long id,
```

```

        @Valid @RequestBody Cliente cliente) {

    try {

        Cliente clienteAtualizado = clienteService.atualizar(id, cliente);

        return ResponseEntity.ok(clienteAtualizado);

    } catch (IllegalArgumentException e) {

        return ResponseEntity.badRequest().body("Erro: " + e.getMessage());

    } catch (Exception e) {

        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)

            .body("Erro interno do servidor");

    }

}

```

/**

* Inativar cliente (soft delete)

*/

@DeleteMapping("/{id}")

public ResponseEntity<?> inativar(@PathVariable Long id) {

```

    try {

        clienteService.inativar(id);

        return ResponseEntity.ok().body("Cliente inativado com sucesso");

    } catch (IllegalArgumentException e) {

        return ResponseEntity.badRequest().body("Erro: " + e.getMessage());

    } catch (Exception e) {

        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)

            .body("Erro interno do servidor");

    }

}

```

/**

* Buscar clientes por nome

*/

```

@GetMapping("/buscar")

public ResponseEntity<List<Cliente>> buscarPorNome(@RequestParam String nome) {

    List<Cliente> clientes = clienteService.buscarPorNome(nome);

    return ResponseEntity.ok(clientes);

}

/**
 * Buscar cliente por email
 */
@GetMapping("/email/{email}")
public ResponseEntity<?> buscarPorEmail(@PathVariable String email) {

    Optional<Cliente> cliente = clienteService.buscarPorEmail(email);

    if (cliente.isPresent()) {

        return ResponseEntity.ok(cliente.get());

    } else {

        return ResponseEntity.notFound().build();

    }

}

}

```

PedidoController - Implementação Completa

```

package com.deliverytech.delivery.controller;

import com.deliverytech.delivery.entity.Pedido;
import com.deliverytech.delivery.entity.StatusPedido;
import com.deliverytech.delivery.service.PedidoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

```



```
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
```

```
import java.util.Optional;
```

```
@RestController
```

```
@RequestMapping("/pedidos")
```

```
@CrossOrigin(origins = "*")
```

```
public class PedidoController {
```

```
    @Autowired
```

```
    private PedidoService pedidoService;
```

```
    /**
```

```
     * Criar novo pedido
```

```
     */
```

```
    @PostMapping
```

```
    public ResponseEntity<?> criarPedido(@RequestParam Long clienteId,
```

```
                                         @RequestParam Long restaurantId) {
```

```
        try {
```

```
            Pedido pedido = pedidoService.criarPedido(clienteId, restaurantId);
```

```
            return ResponseEntity.status(HttpStatus.CREATED).body(pedido);
```

```
        } catch (IllegalArgumentException e) {
```

```
            return ResponseEntity.badRequest().body("Erro: " + e.getMessage());
```

```
        } catch (Exception e) {
```

```
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
```

```
                .body("Erro interno do servidor");
```

```
        }
```

```
    }
```

```
    /**
```

* Adicionar item ao pedido

*/

@PostMapping("/{pedidoId}/itens")

public ResponseEntity<?> adicionarItem(@PathVariable Long pedidoId,

 @RequestParam Long produtoId,

 @RequestParam Integer quantidade) {

 try {

 Pedido pedido = pedidoService.adicionarItem(pedidoId, produtoId, quantidade);

 return ResponseEntity.ok(pedido);

 } catch (IllegalArgumentException e) {

 return ResponseEntity.badRequest().body("Erro: " + e.getMessage());

 } catch (Exception e) {

 return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)

 .body("Erro interno do servidor");

 }

}

/**

* Confirmar pedido

*/

@PutMapping("/{pedidoId}/confirmar")

public ResponseEntity<?> confirmarPedido(@PathVariable Long pedidoId) {

 try {

 Pedido pedido = pedidoService.confirmarPedido(pedidoId);

 return ResponseEntity.ok(pedido);

 } catch (IllegalArgumentException e) {

 return ResponseEntity.badRequest().body("Erro: " + e.getMessage());

 } catch (Exception e) {

 return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)

 .body("Erro interno do servidor");

 }

```
}
```

```
/**
```

```
 * Buscar pedido por ID
```

```
 */
```

```
@GetMapping("/{id}")
```

```
public ResponseEntity<?> buscarPorId(@PathVariable Long id) {
```

```
    Optional<Pedido> pedido = pedidoService.buscarPorId(id);
```

```
    if (pedido.isPresent()) {
```

```
        return ResponseEntity.ok(pedido.get());
```

```
    } else {
```

```
        return ResponseEntity.notFound().build();
```

```
    }
```

```
}
```

```
/**
```

```
 * Listar pedidos por cliente
```

```
 */
```

```
@GetMapping("/cliente/{clienteId}")
```

```
public ResponseEntity<List<Pedido>> listarPorCliente(@PathVariable Long clienteId) {
```

```
    List<Pedido> pedidos = pedidoService.listarPorCliente(clienteId);
```

```
    return ResponseEntity.ok(pedidos);
```

```
}
```

```
/**
```

```
 * Buscar pedido por número
```

```
 */
```

```
@GetMapping("/numero/{numeroPedido}")
```

```
public ResponseEntity<?> buscarPorNumero(@PathVariable String numeroPedido) {
```

```
    Optional<Pedido> pedido = pedidoService.buscarPorNumero(numeroPedido);
```

```

        if (pedido.isPresent()) {
            return ResponseEntity.ok(pedido.get());
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}

/**
 * Atualizar status do pedido
 */
@PutMapping("/{pedidoId}/status")
public ResponseEntity<?> atualizarStatus(@PathVariable Long pedidoId,
                                       @RequestParam StatusPedido status) {

    try {
        Pedido pedido = pedidoService.atualizarStatus(pedidoId, status);
        return ResponseEntity.ok(pedido);
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().body("Erro: " + e.getMessage());
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body("Erro interno do servidor");
    }
}

/**
 * Cancelar pedido
 */
@PutMapping("/{pedidoId}/cancelar")
public ResponseEntity<?> cancelarPedido(@PathVariable Long pedidoId,
                                       @RequestParam(required = false) String motivo) {

```

```

try {
    Pedido pedido = pedidoService.cancelarPedido(pedidoId, motivo);
    return ResponseEntity.ok(pedido);
} catch (IllegalArgumentException e) {
    return ResponseEntity.badRequest().body("Erro: " + e.getMessage());
} catch (Exception e) {
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
        .body("Erro interno do servidor");
}
}
}
}

```

Atividade 4: Testes e Validação

Arquivo data.sql - Dados de Exemplo

-- Dados de exemplo para testes

-- Arquivo: src/main/resources/data.sql

-- Inserir clientes

```

INSERT INTO clientes (nome, email, telefone, endereco, data_cadastro, ativo) VALUES
('João Silva', 'joao@email.com', '(11) 99999-1111', 'Rua A, 123 - São Paulo/SP', NOW(), true),
('Maria Santos', 'maria@email.com', '(11) 99999-2222', 'Rua B, 456 - São Paulo/SP', NOW(),
true),
('Pedro Oliveira', 'pedro@email.com', '(11) 99999-3333', 'Rua C, 789 - São Paulo/SP', NOW(),
true);

```

-- Inserir restaurantes

```

INSERT INTO restaurantes (nome, categoria, endereco, telefone, taxa_entrega, avaliacao, ativo)
VALUES
('Pizzaria Bella', 'Italiana', 'Av. Paulista, 1000 - São Paulo/SP', '(11) 3333-1111', 5.00, 4.5, true),
('Burger House', 'Hamburgueria', 'Rua Augusta, 500 - São Paulo/SP', '(11) 3333-2222', 3.50, 4.2,
true),

```

```
('Sushi Master', 'Japonesa', 'Rua Liberdade, 200 - São Paulo/SP', '(11) 3333-3333', 8.00, 4.8, true);
```

```
-- Inserir produtos
```

```
INSERT INTO produtos (nome, descricao, preco, categoria, disponivel, restaurante_id) VALUES
```

```
-- Pizzaria Bella
```

```
('Pizza Margherita', 'Molho de tomate, mussarela e manjeriç o', 35.90, 'Pizza', true, 1),
```

```
('Pizza Calabresa', 'Molho de tomate, mussarela e calabresa', 38.90, 'Pizza', true, 1),
```

```
('Lasanha Bolonhesa', 'Lasanha tradicional com molho bolonhesa', 28.90, 'Massa', true, 1),
```

```
-- Burger House
```

```
('X-Burger', 'Hamb rguer, queijo, alface e tomate', 18.90, 'Hamb rguer', true, 2),
```

```
('X-Bacon', 'Hamb rguer, queijo, bacon, alface e tomate', 22.90, 'Hamb rguer', true, 2),
```

```
('Batata Frita', 'Porç o de batata frita crocante', 12.90, 'Acompanhamento', true, 2),
```

```
-- Sushi Master
```

```
('Combo Sashimi', '15 pe as de sashimi variado', 45.90, 'Sashimi', true, 3),
```

```
('Hot Roll Salm o', '8 pe as de hot roll de salm o', 32.90, 'Hot Roll', true, 3),
```

```
('Temaki Atum', 'Temaki de atum com cream cheese', 15.90, 'Temaki', true, 3);
```

```
-- Inserir pedidos de exemplo
```

```
INSERT INTO pedidos (numero_pedido, data_pedido, status, valor_total, observacoes, cliente_id, restaurante_id) VALUES
```

```
('PED1234567890', NOW(), 'PENDENTE', 54.80, 'Sem cebola na pizza', 1, 1),
```

```
('PED1234567891', NOW(), 'CONFIRMADO', 41.80, '', 2, 2),
```

```
('PED1234567892', NOW(), 'ENTREGUE', 78.80, 'Wasabi   parte', 3, 3);
```

```
-- Inserir itens dos pedidos
```

```
INSERT INTO itens_pedido (quantidade, preco_unitario, subtotal, pedido_id, produto_id) VALUES
```

```
-- Pedido 1 (Jo o - Pizzaria Bella)
```

```
(1, 35.90, 35.90, 1, 1), -- Pizza Margherita
```

(1, 28.90, 28.90, 1, 3), -- Lasanha

-- Pedido 2 (Maria - Burger House)

(1, 22.90, 22.90, 2, 5), -- X-Bacon

(1, 18.90, 18.90, 2, 4), -- X-Burger

-- Pedido 3 (Pedro - Sushi Master)

(1, 45.90, 45.90, 3, 7), -- Combo Sashimi




(1, 32.90, 32.90, 3, 8); -- Hot Roll

Collection Postman - Exemplos de Testes

Exemplos de requisições para testar no Postman/Insomnia:

- **Cadastrar Cliente POST** <http://localhost:8080/clientes> Content-Type: application/json
 - {
 - "nome": "Ana Costa",
 - "email": "ana@email.com",
 - "telefone": "(11) 99999-4444",
 - "endereco": "Rua D, 321 - São Paulo/SP"
 - }
- **Listar Clientes GET** <http://localhost:8080/clientes>
- **Buscar Cliente por ID GET** <http://localhost:8080/clientes/1>
- **Criar Pedido POST** <http://localhost:8080/pedidos?clienteId=1&restaurantId=1>
- **Adicionar Item ao Pedido POST**
<http://localhost:8080/pedidos/1/itens?produtoId=1&quantidade=2>
- **Confirmar Pedido PUT** <http://localhost:8080/pedidos/1/confirmar>
- **Listar Pedidos por Cliente GET** <http://localhost:8080/pedidos/cliente/1>
- **Atualizar Status do Pedido PUT**
<http://localhost:8080/pedidos/1/status?status=PREPARANDO>







Respostas esperadas:

-  **201 (Created)** para criações
-  **200 (OK)** para consultas e atualizações
-  **404 (Not Found)** para recursos não encontrados








-  **400 (Bad Request)** para dados inválidos

Checklist de Validação Final








Repositories

-  ClienteRepository com métodos customizados
-  RestauranteRepository com consultas por categoria
-  ProdutoRepository com filtros por restaurante
-  PedidoRepository com consultas por cliente e status
-  Uso correto de @Query quando necessário
-  Métodos derivados implementados corretamente




Services

-  ClienteService com validações de email único
-  RestauranteService com regras de negócio
-  ProdutoService com controle de disponibilidade
-  PedidoService com cálculos e mudanças de status
-  Injeção de dependência configurada (@Autowired)
-  Tratamento de exceções implementado
-  Métodos transacionais (@Transactional)

Controllers

-  ClienteController com CRUD completo
-  PedidoController com criação e gestão de pedidos
-  Endpoints REST funcionando corretamente
-  Validações de entrada (@Valid)
-  Respostas HTTP adequadas (200, 201, 400, 404)
-  Tratamento de erros nos controllers
-  @CrossOrigin configurado

Testes e Validação

-  Aplicação inicializa sem erros
-  Dados de exemplo carregados corretamente
-  Endpoints testados via Postman/Insomnia

- ✓ Console H2 acessível e funcional
- ✓ Logs de SQL visíveis no console

✓ Documentação

- ✓ README.md atualizado com instruções
- ✓ Comentários nos códigos principais
- ✓ Estrutura do projeto organizada