Regras de Negócio Gerais

1. Cadastro de Empresa:
   * Apenas uma empresa pode ser cadastrada no sistema (uma instância única de empresa).
   * Todos os dados obrigatórios, como nome, cidade, província e setor, devem ser fornecidos no momento do cadastro.
2. Cadastro de Usuário:
   * Um usuário deve estar associado a uma empresa (companyID obrigatório).
   * Não é permitido duplicar o número de identificação fiscal (cedula) de um usuário.
   * Apenas usuários ativos (ativo = TRUE) podem realizar operações como vendas ou pedidos de compra.
3. Cadastro de Fornecedores:
   * O número de identificação fiscal do fornecedor (cedula) deve ser único.
   * O sistema deve permitir apenas fornecedores com todas as informações de contato preenchidas (telefone, cidade, província, etc.).
   * Fornecedores só podem ser inativados se não houver pedidos pendentes associados.

Regras para Operações de Estoque

1. Gestão de Estoque:
   * Todo item deve ser registrado em uma localização específica de estoque (rua e prateleira).
   * O estoque não pode ser atualizado com valores negativos de peso ou quantidade.
   * A data de entrada no estoque (data\_entrada) deve ser menor ou igual à data de saída (data\_saida), se preenchida.
2. Entrada e Saída de Itens:
   * Ao registrar um pedido de compra, os itens adquiridos devem ser automaticamente adicionados ao estoque.
   * A venda de itens deve reduzir a quantidade disponível no estoque.
   * Não é permitido registrar vendas para itens com quantidade insuficiente no estoque.

Regras para Pedidos

1. Pedidos de Compra:
   * Um pedido de compra deve estar associado a um fornecedor registrado.
   * O valor total do pedido de compra deve incluir o desconto aplicado (se houver).
   * Apenas usuários com permissões específicas podem registrar pedidos de compra.
2. Pedidos de Venda:
   * Um cliente deve ser registrado para criar um pedido de venda.
   * Cada pedido de venda deve ter pelo menos um item vinculado.
   * O valor final de um pedido de venda deve incluir o desconto aplicado (se houver).
   * Pagamentos pendentes associados a um cliente impedem novos pedidos de venda.

Regras de Pagamento

1. Pagamentos de Fornecedores:
   * Um pagamento de fornecedor deve estar associado a um pedido de compra.
   * O valor do pagamento não pode exceder o valor total do pedido de compra.
   * Apenas pagamentos realizados podem atualizar o status de um pedido para "completo".
2. Pagamentos de Clientes:
   * Um pagamento de cliente deve estar vinculado a um pedido de venda.
   * Clientes com pagamentos pendentes não podem fazer novos pedidos até regularizarem a situação.
   * O sistema deve registrar a forma de pagamento (transferência, boleto, etc.) obrigatoriamente.

Regras para Relatórios e Logística

1. Relatórios de Vendas e Estoque:
   * Apenas gestores podem gerar relatórios de vendas e estoque.
   * Relatórios devem incluir dados consolidados, como valor total de vendas por período e itens mais vendidos.
2. Logística de Entrega:
   * Toda venda deve ter um registro logístico vinculado à tabela cliente\_logistica.
   * O status da entrega deve ser atualizado assim que o pedido for entregue.
   * Dados de endereço e contato do cliente são obrigatórios para gerar um registro logístico.

Regras de Validação e Segurança

1. Validação de Dados:
   * Todos os campos obrigatórios de cada tabela devem ser validados antes de qualquer inserção ou atualização.
   * Dados duplicados, como números fiscais (cedula), são proibidos.
2. Acesso e Permissões:
   * Apenas administradores podem gerenciar usuários, fornecedores e estoque.
   * Clientes têm acesso apenas aos seus próprios pedidos e pagamentos.
   * O sistema deve registrar logs de todas as operações críticas (cadastro, atualizações e exclusões).

1. Camadas do Sistema

O sistema deve seguir o modelo de arquitetura em camadas:

* Controller: Gerencia as requisições HTTP e mapeia endpoints REST.
* Service: Contém a lógica de negócios, aplicando as validações necessárias.
* Repository (DAO): Interage diretamente com o banco de dados usando JPA ou consultas SQL.
* Model: Representa as entidades do banco de dados.

2. Validações na Camada Service

A camada de serviço é responsável por validar as operações:

* Validação de Dados:
  + Campos obrigatórios não podem ser nulos ou vazios.
  + Formatos de e-mail, telefone e CPF/CNPJ devem ser validados.
* Regras de Dependência:
  + Verificar se o cliente existe antes de registrar pedidos de venda.
  + Garantir que o fornecedor está ativo antes de registrar pedidos de compra.
* Consistência do Estoque:
  + Quantidades solicitadas não podem exceder o estoque disponível.

3. Segurança

* Autenticação e autorização devem ser implementadas usando JWT (JSON Web Token):
  + Administrador: Acesso total ao sistema.
  + Usuário Funcional: Pode registrar vendas, gerenciar pagamentos e visualizar estoque.
  + Cliente: Acesso restrito às informações relacionadas aos seus pedidos.
* O backend deve usar Spring Security para proteger os endpoints.

4. Endpoints REST

Os endpoints seguem os princípios RESTful:

* Exemplo de endpoints:
  + POST /empresa → Cadastrar empresa.
  + POST /usuario → Cadastrar usuário.
  + GET /usuario/{id} → Consultar detalhes de um usuário.
  + POST /pedido-compra → Registrar pedido de compra.
  + POST /pedido-venda → Registrar pedido de venda.
  + PUT /estoque/{id} → Atualizar quantidade em estoque.
  + GET /relatorios/vendas → Gerar relatório de vendas.

Regras e Validações no Java

1. Validações com Anotações

Use anotações do Bean Validation para validar os campos de entrada no backend:

* Exemplo de validações:

java

@Entity

public class Usuario {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Nome não pode ser nulo")

@Size(min = 3, max = 40, message = "Nome deve ter entre 3 e 40 caracteres")

private String nome;

@Email(message = "E-mail inválido")

private String email;

@Pattern(regexp = "\\d{11}", message = "CPF deve conter 11 dígitos")

private String cpf;

@NotNull(message = "A empresa associada é obrigatória")

@ManyToOne

private Empresa empresa;

}

2. Regras de Estoque

* Atualizar o estoque ao registrar uma venda:

java

public void atualizarEstoque(Item item, int quantidadeVendida) {

if (item.getQuantidade() < quantidadeVendida) {

throw new IllegalArgumentException("Quantidade insuficiente em estoque");

}

item.setQuantidade(item.getQuantidade() - quantidadeVendida);

itemRepository.save(item);

}

3. Regras de Pagamento

* Garantir que pagamentos sejam registrados corretamente:

java

public void registrarPagamento(Pagamento pagamento) {

if (pagamento.getValorPago() <= 0) {

throw new IllegalArgumentException("O valor do pagamento deve ser maior que zero");

}

Cliente cliente = clienteRepository.findById(pagamento.getClienteId())

.orElseThrow(() -> new EntityNotFoundException("Cliente não encontrado"));

pagamento.setStatus("Pago");

pagamentoRepository.save(pagamento);

}

4. Gerenciamento de Pedidos

* Criar pedidos de venda:

java

public PedidoVenda criarPedidoVenda(PedidoVenda pedidoVenda) {

Cliente cliente = clienteRepository.findById(pedidoVenda.getClienteId())

.orElseThrow(() -> new EntityNotFoundException("Cliente não encontrado"));

if (cliente.temPagamentosPendentes()) {

throw new IllegalArgumentException("Cliente possui pagamentos pendentes");

}

pedidoVenda.setDataVenda(LocalDate.now());

return pedidoVendaRepository.save(pedidoVenda);

}

Exemplo de Implementação Backend no Spring Boot

Controller para Pedido de Venda:

java

@RestController

@RequestMapping("/pedido-venda")

public class PedidoVendaController {

@Autowired

private PedidoVendaService pedidoVendaService;

@PostMapping

public ResponseEntity<PedidoVenda> criarPedido(@RequestBody PedidoVendaDTO pedidoVendaDTO) {

PedidoVenda pedido = pedidoVendaService.criarPedido(pedidoVendaDTO);

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

}

}

Serviço para Pedido de Venda:

java

@Service

public class PedidoVendaService {

@Autowired

private PedidoVendaRepository pedidoVendaRepository;

@Autowired

private ClienteRepository clienteRepository;

public PedidoVenda criarPedido(PedidoVendaDTO pedidoVendaDTO) {

Cliente cliente = clienteRepository.findById(pedidoVendaDTO.getClienteId())

.orElseThrow(() -> new EntityNotFoundException("Cliente não encontrado"));

if (cliente.temPagamentosPendentes()) {

throw new IllegalArgumentException("Cliente possui pagamentos pendentes");

}

PedidoVenda pedidoVenda = new PedidoVenda();

pedidoVenda.setCliente(cliente);

pedidoVenda.setDataVenda(LocalDate.now());

pedidoVenda.setValor(pedidoVendaDTO.getValor());

pedidoVenda.setDesconto(pedidoVendaDTO.getDesconto());

return pedidoVendaRepository.save(pedidoVenda);

}

}