# Sistema de Controle de Vendas e Estoque

## Documentação Técnica e Manual do Usuário

---

### Índice



---

## Visão Geral

O Sistema de Controle de Vendas e Estoque é uma aplicação web desenvolvida em Java com Spring Boot e Thymeleaf, projetada para gerenciar de forma eficiente as operações de vendas e controle de estoque de uma empresa.

### Objetivos do Sistema

- **Gestão de Produtos**: Cadastro, edição e controle de produtos

- **Controle de Estoque**: Monitoramento de quantidades, alertas de estoque baixo
- **Gestão de Vendas**: Criação e acompanhamento de pedidos
- **Gestão de Clientes**: Cadastro e manutenção de informações de clientes
- **Gestão de Usuários**: Controle de acesso com diferentes perfis
- **Dashboard Analítico**: Visão geral das operações e indicadores

### Características Principais

- Interface web responsiva e intuitiva
- Sistema de autenticação e autorização
- Relatórios e dashboards em tempo real
- Controle de estoque com alertas automáticos
- Gestão completa do ciclo de vendas
- Arquitetura modular e escalável

---

## Arquitetura do Sistema

### Padrão Arquitetural

O sistema segue o padrão **MVC (Model-View-Controller)** com as seguintes camadas:

#### Model (Modelo)

- **Entidades JPA**: Representam as tabelas do banco de dados
- **Enums**: Definem valores constantes como status e perfis
- **DTOs**: Objetos de transferência de dados quando necessário

#### View (Visão)

- **Templates Thymeleaf**: Páginas HTML dinâmicas
- **CSS/JavaScript**: Estilização e interatividade
- **Bootstrap**: Framework CSS para responsividade

#### Controller (Controlador)

- **Controllers REST**: Gerenciam requisições HTTP
- **Services**: Lógica de negócio
- **Repositories**: Acesso aos dados

### Estrutura de Pacotes

```
com.sistema.vendasestoque/
├── config/          # Configurações do sistema
├── controller/      # Controladores REST
├── entity/          # Entidades JPA
├── enums/           # Enumerações
├── repository/      # Repositórios de dados
├── service/         # Serviços de negócio
└── VendasEstoqueApplication.java
```

---

## Tecnologias Utilizadas

### Backend

- **Java 17**: Linguagem de programação
- **Spring Boot 3.2.0**: Framework principal
- **Spring Data JPA**: Persistência de dados
- **Spring Web**: Desenvolvimento web
- **Spring Validation**: Validação de dados
- **Hibernate**: ORM (Object-Relational Mapping)

### Frontend

- **Thymeleaf**: Template engine
- **Bootstrap 5.3.0**: Framework CSS
- **jQuery 3.7.0**: Biblioteca JavaScript
- **Font Awesome**: Ícones
- **CSS3/HTML5**: Tecnologias web padrão

### Banco de Dados

- **H2 Database**: Banco em memória para desenvolvimento
- **Suporte para MySQL/PostgreSQL**: Para produção

### Ferramentas de Build

- **Maven**: Gerenciamento de dependências
- **Spring Boot DevTools**: Desenvolvimento ágil

---

## Instalação e Configuração

### Pré-requisitos

- Java 17 ou superior
- Maven 3.6 ou superior
- IDE de sua preferência (IntelliJ IDEA, Eclipse, VS Code)

### Passos de Instalação

1. **Clone ou extraia o projeto** `bash cd sistema-vendas-estoque`
2. **Compile o projeto** `bash mvn clean compile`
3. **Execute a aplicação** `bash mvn spring-boot:run`
4. **Acesse o sistema**
5. URL: http://localhost:8080
6. O sistema será redirecionado automaticamente para o dashboard

### Configuração do Banco de Dados

#### Desenvolvimento (H2)

O sistema vem configurado com H2 para desenvolvimento. As configurações estão em `application.properties`:

```
# Configurações do H2
spring.datasource.url=jdbc:h2:mem:vendasestoque
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Console H2 (apenas desenvolvimento)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

#### Produção (MySQL)

Para usar MySQL em produção, adicione as seguintes configurações:

```
# Configurações do MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/vendasestoque
spring.datasource.username=seu_usuario
spring.datasource.password=sua_senha
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA/Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

### Dados Iniciais

O sistema inclui um inicializador de dados (`DataInitializer`) que cria: - 3 usuários (admin, gerente, vendedor) - 3 clientes - 8 produtos com estoque - 4 pedidos de exemplo

---

## Funcionalidades

### 1. Dashboard

- Visão geral das vendas do dia e mês

- Indicadores de pedidos pendentes
- Alertas de estoque baixo
- Estatísticas de produtos e clientes
- Ações rápidas para operações comuns

## 2. Gestão de Produtos

- **Cadastro**: Nome, descrição, preço, código de barras, categoria
- **Listagem**: Visualização em cards com filtros
- **Edição**: Atualização de informações
- **Status**: Ativação/desativação de produtos
- **Categorização**: Organização por categorias

## 3. Controle de Estoque

- **Consulta**: Visualização de quantidades atuais
- **Movimentação**: Entrada e saída de produtos
- **Alertas**: Estoque baixo e zerado
- **Histórico**: Rastreamento de movimentações
- **Relatórios**: Análises de estoque

## 4. Gestão de Vendas

- **Pedidos**: Criação e acompanhamento
- **Status**: Pendente, confirmado, em preparação, entregue
- **Itens**: Adição de produtos aos pedidos
- **Cálculos**: Valor total automático
- **Observações**: Notas adicionais

## 5. Gestão de Clientes

- **Cadastro**: Informações pessoais e de contato
- **CPF/CNPJ**: Suporte para pessoa física e jurídica

- **Endereço**: Informações de entrega
- **Histórico**: Pedidos do cliente

### 6. Gestão de Usuários

- **Perfis**: Admin, Gerente, Vendedor
- **Permissões**: Controle de acesso por perfil
- **Status**: Ativação/desativação
- **Segurança**: Senhas criptografadas

---

# Manual do Usuário

## Acesso ao Sistema

1. Abra o navegador e acesse: http://localhost:8080
2. O sistema redirecionará automaticamente para o dashboard

## Navegação Principal

### Menu Superior

- **Dashboard**: Página inicial com indicadores
- **Vendas**: Gestão de pedidos e relatórios
- **Estoque**: Controle de estoque e movimentações
- **Cadastros**: Produtos, clientes e usuários

## Operações Básicas

### Cadastrar Produto

1. Acesse **Cadastros > Produtos**
2. Clique em **Novo Produto**
3. Preencha as informações obrigatórias

4. Clique em **Salvar**

### Controlar Estoque

1. Acesse **Estoque > Consultar Estoque**
2. Visualize as quantidades atuais
3. Use **Movimentar** para adicionar/remover itens
4. Configure quantidades mínimas

### Criar Pedido

1. Acesse **Vendas > Novo Pedido**
2. Selecione o cliente
3. Adicione produtos ao pedido
4. Confirme o pedido

### Gerenciar Clientes

1. Acesse **Cadastros > Clientes**
2. Use **Novo Cliente** para cadastrar
3. Preencha informações de contato
4. Salve o cadastro

## Funcionalidades Avançadas

### Filtros e Busca

- Use os campos de busca para localizar registros
- Aplique filtros por categoria, preço, status
- Combine múltiplos filtros para refinar resultados

### Relatórios

- Acesse relatórios através dos menus específicos
- Visualize gráficos e estatísticas

- Exporte dados quando necessário

### Alertas de Estoque

- Monitore produtos com estoque baixo
- Configure quantidades mínimas
- Receba alertas automáticos

---

# Estrutura do Banco de Dados

## Entidades Principais

### Usuários

```
CREATE TABLE usuarios (
    usuario_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    senha VARCHAR(255) NOT NULL,
    perfil VARCHAR(20) NOT NULL,
    ativo BOOLEAN DEFAULT TRUE,
    data_criacao TIMESTAMP,
    data_atualizacao TIMESTAMP
);
```

### Clientes

```
CREATE TABLE clientes (
    cliente_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    telefone VARCHAR(20),
    cpf_cnpj VARCHAR(20) UNIQUE,
    endereco VARCHAR(255),
    ativo BOOLEAN DEFAULT TRUE,
    data_criacao TIMESTAMP,
    data_atualizacao TIMESTAMP
);
```

### Produtos

```sql
CREATE TABLE produtos (
    produto_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    preco DECIMAL(10,2) NOT NULL,
    codigo_barras VARCHAR(50) UNIQUE,
    categoria VARCHAR(50),
    ativo BOOLEAN DEFAULT TRUE,
    data_criacao TIMESTAMP,
    data_atualizacao TIMESTAMP
);
```

### Estoque

```sql
CREATE TABLE estoque (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    produto_id BIGINT NOT NULL,
    quantidade_atual INTEGER DEFAULT 0,
    quantidade_minima INTEGER DEFAULT 0,
    data_atualizacao TIMESTAMP,
    FOREIGN KEY (produto_id) REFERENCES produtos(produto_id)
);
```

### Pedidos

```sql
CREATE TABLE pedidos (
    pedido_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    data DATE NOT NULL,
    cliente_id BIGINT NOT NULL,
    usuario_id BIGINT NOT NULL,
    valor_total DECIMAL(10,2) DEFAULT 0,
    status VARCHAR(20) NOT NULL,
    observacoes VARCHAR(500),
    data_criacao TIMESTAMP,
    data_atualizacao TIMESTAMP,
    FOREIGN KEY (cliente_id) REFERENCES clientes(cliente_id),
    FOREIGN KEY (usuario_id) REFERENCES usuarios(usuario_id)
);
```

#### Itens do Pedido

```
CREATE TABLE itens_pedido (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    pedido_id BIGINT NOT NULL,
    produto_id BIGINT NOT NULL,
    quantidade INTEGER NOT NULL,
    preco_unitario DECIMAL(10,2) NOT NULL,
    subtotal DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (pedido_id) REFERENCES pedidos(pedido_id),
    FOREIGN KEY (produto_id) REFERENCES produtos(produto_id)
);
```

### Relacionamentos

- **Usuários** ↔ **Pedidos**: Um usuário pode criar vários pedidos
- **Clientes** ↔ **Pedidos**: Um cliente pode ter vários pedidos
- **Produtos** ↔ **Estoque**: Relacionamento 1:1
- **Pedidos** ↔ **Itens do Pedido**: Um pedido pode ter vários itens
- **Produtos** ↔ **Itens do Pedido**: Um produto pode estar em vários itens

---

## API e Endpoints

### Controladores Principais

#### HomeController

- `GET /` - Página inicial (redireciona para dashboard)
- `GET /dashboard` - Dashboard principal

#### ProdutoController

- `GET /produtos` - Listar produtos
- `GET /produtos/novo` - Formulário de novo produto
- `POST /produtos` - Salvar produto
- `GET /produtos/{id}` - Visualizar produto
- `GET /produtos/{id}/editar` - Formulário de edição

- `PUT /produtos/{id}` - Atualizar produto
- `POST /produtos/{id}/ativar` - Ativar produto
- `POST /produtos/{id}/desativar` - Desativar produto

**EstoqueController**

- `GET /estoque` - Consultar estoque
- `GET /estoque/baixo` - Produtos com estoque baixo
- `GET /estoque/zerado` - Produtos com estoque zerado
- `POST /estoque/{produtoId}/adicionar` - Adicionar ao estoque
- `POST /estoque/{produtoId}/remover` - Remover do estoque

**PedidoController**

- `GET /pedidos` - Listar pedidos
- `GET /pedidos/novo` - Formulário de novo pedido
- `POST /pedidos` - Criar pedido
- `GET /pedidos/{id}` - Visualizar pedido
- `POST /pedidos/{id}/confirmar` - Confirmar pedido

**ClienteController**

- `GET /clientes` - Listar clientes
- `GET /clientes/novo` - Formulário de novo cliente
- `POST /clientes` - Salvar cliente
- `GET /clientes/{id}` - Visualizar cliente
- `GET /clientes/{id}/editar` - Formulário de edição

**UsuarioController**

- `GET /usuarios` - Listar usuários
- `GET /usuarios/novo` - Formulário de novo usuário
- `POST /usuarios` - Salvar usuário

- `GET /usuarios/{id}` - Visualizar usuário
- `POST /usuarios/{id}/ativar` - Ativar usuário
- `POST /usuarios/{id}/desativar` - Desativar usuário

---

## Considerações de Segurança

### Validação de Dados

- Validação no lado servidor usando Bean Validation
- Sanitização de entradas do usuário
- Validação de tipos e formatos

### Controle de Acesso

- Sistema de perfis (Admin, Gerente, Vendedor)
- Controle de permissões por funcionalidade
- Sessões seguras

### Proteção de Dados

- Senhas não são armazenadas em texto plano
- Validação de CPF/CNPJ
- Prevenção contra SQL Injection através de JPA

### Recomendações para Produção

1. Implementar autenticação robusta (Spring Security)
2. Usar HTTPS para todas as comunicações
3. Configurar backup automático do banco de dados
4. Implementar logs de auditoria
5. Configurar rate limiting
6. Usar banco de dados dedicado (MySQL/PostgreSQL)

---

# Manutenção e Suporte

## Logs do Sistema

Os logs são gerenciados pelo Spring Boot e podem ser configurados em `application.properties`:

```
# Configuração de logs
logging.level.com.sistema.vendasestoque=DEBUG
logging.file.name=logs/sistema.log
logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %msg%n
```

## Backup e Recuperação

- **Desenvolvimento**: Dados em memória (H2)
- **Produção**: Configurar backup automático do banco
- **Arquivos**: Backup dos arquivos de configuração

## Monitoramento

- Use Spring Boot Actuator para monitoramento
- Configure health checks
- Monitore uso de memória e CPU

## Atualizações

1. Teste em ambiente de desenvolvimento
2. Faça backup completo
3. Execute migrações de banco se necessário
4. Deploy em produção
5. Verifique funcionamento

### Solução de Problemas Comuns

#### Erro de Conexão com Banco

- Verifique configurações em `application.properties`
- Confirme se o banco está rodando
- Valide credenciais de acesso

#### Problemas de Performance

- Analise queries SQL nos logs
- Verifique índices no banco de dados
- Monitore uso de memória

#### Erros de Validação

- Verifique dados de entrada
- Confirme regras de negócio
- Analise logs de erro

---

## Conclusão

O Sistema de Controle de Vendas e Estoque foi desenvolvido seguindo as melhores práticas de desenvolvimento Java e Spring Boot, oferecendo uma solução completa e escalável para gestão empresarial.

### Principais Benefícios

- **Eficiência**: Automatização de processos manuais
- **Controle**: Visibilidade completa das operações
- **Escalabilidade**: Arquitetura preparada para crescimento
- **Usabilidade**: Interface intuitiva e responsiva
- **Manutenibilidade**: Código bem estruturado e documentado

## Próximos Passos

- Implementação de relatórios avançados
- Integração com sistemas externos
- Módulo de compras e fornecedores
- App mobile para vendedores
- Sistema de notificações

---

**Desenvolvido com Spring Boot e Thymeleaf**
**Versão 1.0 - 2024**