HoStore - Sistema ERP para Lojas TCG

Java 17+ | Build Maven | License MIT | Status Em Desenvolvimento

> **Sistema de Gestão Empresarial (ERP) especializado para lojas de Trading Card Games (TCG)**.
> HoStore oferece controle completo de vendas, estoque, financeiro e relatórios para lojas físicas de Pokémon, Magic, Yu-Gi-Oh!, Digimon, One Piece e muito mais.

Índice

Visão Geral

HoStore é um **ERP desktop** desenvolvido em **Java 17** com interface gráfica moderna e responsiva. Ele foi criado especificamente para atender as necessidades de lojas físicas especializadas em Trading Card Games, oferecendo:

- ☑ **Gestão de Estoque Inteligente** - Cadastro especializado por tipo de produto
- ☑ **Vendas Rápidas** - Interface otimizada para venda em segundos
- ☑ **Controle Financeiro** - Fluxo de caixa, contas a pagar/receber
- ☑ **Fiscal Integrado** - Emissão de documentos fiscais (NFC-e, NFe)
- ☑ **Relatórios Completos** - Dashboards e exportação em PDF/Excel
- ☑ **APIs de TCG** - Integração com dados de Pokémon, Magic, Yu-Gi-Oh!, Digimon, One Piece
- ☑ **Backup Automático** - Proteção de dados com sincronização
- ☑ **Auditoria Completa** - Registro de todas as ações

Funcionalidades Principais

Módulo de Vendas

- Criar vendas com carrinho dinâmico

- Adicionar/remover produtos em tempo real
- Aplicar descontos por item ou total
- Calcular automaticamente troco e taxas
- Parcelamento inteligente com datas configuráveis
- Múltiplas formas de pagamento (Dinheiro, Cartão, PIX, Transferência)
- Emissão de comprovante em PDF ou impressão direta
- Devolução de produtos com reintegração ao estoque
- Estorno de vendas com reversão de movimentações
- Histórico completo com auditoria

📦 Módulo de Estoque

- **Categorias Especializadas**:
  - Cartas (Pokémon TCG, Magic, Yu-Gi-Oh!, etc.)
  - Boosters e Booster Boxes
  - Decks prontos
  - Elite Trainer Boxes (ETBs)
  - Acessórios (Sleeves, Playmats, dados)
  - Produtos alimentícios
- Busca avançada por nome, categoria e faixa de preço
- Alertas de estoque baixo (<5 unidades)
- Movimentação de estoque rastreada
- Pedidos de compra integrados
- Entrada de produtos com nota fiscal
- Exclusão com histórico de auditoria

💰 Módulo Financeiro

- **Contas a Pagar**: Registrar, parcelas, pagamentos
- **Contas a Receber**: Acompanhar vendas parceladas
- **Crédito de Loja**: Gerenciar créditos de clientes
- **Plano de Contas**: Classificação fiscal e contábil
- **Relatórios Financeiros**: Fluxo de caixa, resultados por período
- **Impostos**: Integração com cálculos de ICMS, IPI, PIS, COFINS

📊 Módulo de Relatórios

- Dashboard com KPIs principais
- Vendas por período (dia, mês, ano)
- Produtos mais vendidos
- Clientes com mais compras
- Análise de margem por produto
- Exportação em PDF e Excel
- Resumo de estoque por categoria

👥 Módulo de Clientes

- Cadastro de clientes com CPF/CNPJ

- Histórico de compras
- Saldo de crédito
- Dados para entrega
- Preferências de contato

👤 Sistema de Usuários

- Login com autenticação
- Controle de permissões por função
- Auditoria de ações por usuário
- Backup e restauração de dados

🎴 Integração com TCGs

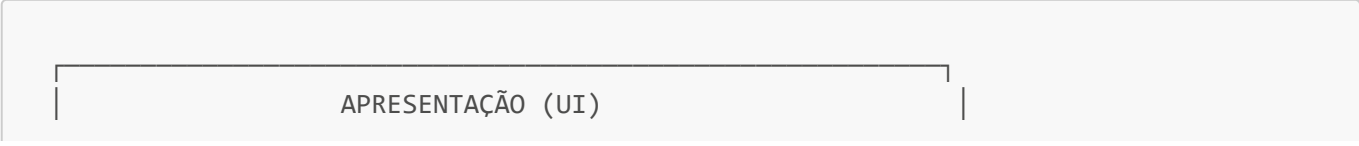
- **Sincronização de Dados**: Atualiza automaticamente coleções e sets
- **APIs Suportadas**:
  - **Pokémon TCG**: Todos os sets e cartas
  - **Magic**: Scryfall API (todos os sets)
  - **Yu-Gi-Oh!**: YGOPRODeck (todos os cards)
  - **Digimon**: digimoncard.io API
  - **One Piece**: optcgapi.com
- Cache local para performance offline

🏗️ Arquitetura e Tecnologias

Stack Tecnológico

Componente	Tecnologia	Versão
Linguagem	Java	17+
Build	Maven	3.8.0+
UI	Swing + FlatLaf	3.6
Banco de Dados	SQLite	3.42.0
PDF	Apache PDFBox	3.0.2
JSON	Gson	2.10.1
Excel	Apache POI	5.2.3
CSV	OpenCSV	5.7.1
HTTP Client	Java HTTP Client	17+

Estrutura de Camadas



```
│  Swing + FlatLaf | Dialogs | Painéis | Componentes      │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│               CONTROLADORES (Controller)                │
│  VendaController | EstoqueController | ProdutoController│
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│                 SERVIÇOS (Service)                      │
│  VendaService | EstoqueService | DocumentoFiscalService │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│              ACESSO A DADOS (DAO)                       │
│  VendaDAO | EstoqueDAO | ClienteDAO | ProdutoDAO        │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│            BANCO DE DADOS (SQLite)                      │
│  Tabelas de Vendas, Estoque, Clientes, Financeiro       │
└─────────────────────────────────────────────────────────┘
```

Padrões de Projeto

- **MVC (Model-View-Controller)**: Separação clara de responsabilidades
- **DAO (Data Access Object)**: Acesso padronizado ao banco
- **Service Layer**: Lógica transacional e regras de negócio
- **Factory Pattern**: Criação de objetos complexos
- **Singleton**: Gerenciamento de conexões e sessão
- **Observer**: Atualização de UI em tempo real

💻 Requisitos do Sistema

Requisitos Mínimos

- **Sistema Operacional**: Windows 10+, macOS 10.15+, Linux (Ubuntu 20.04+)
- **Java**: JDK 17 ou superior
- **RAM**: 2 GB mínimo
- **Armazenamento**: 500 MB livres
- **Resolução**: 1024x768 mínimo (recomendado 1920x1080)

Requisitos para Desenvolvimento

- **Java**: JDK 17+
- **Maven**: 3.8.0+
- **IDE**: VS Code, IntelliJ IDEA, Eclipse
- **Git**: 2.30+

Instalação

1. Clonar o Repositório

```
git clone https://github.com/oDuPrado/HoStore.git
cd HoStore
```

2. Compilar com Maven

```
# Compilar e executar testes
mvn clean compile

# Gerar JAR executável com todas as dependências
mvn clean package
```

3. Executar a Aplicação

Opção A: Usando Maven

```
mvn exec:java@run
```

Opção B: Executar JAR diretamente

```
java -jar target/HoStore-1.0.0-jar-with-dependencies.jar
```

Opção C: Executar diretamente (IDE)

- Abra o projeto na IDE
- Execute a classe `app.Main`

4. Primeiro Uso

1. O sistema criará automaticamente o banco de dados SQLite (`hostore.db`)
2. Banco será inicializado com tabelas padrão
3. Você será direcionado para login

- **Usuário padrão**: `admin`
- **Senha padrão**: `admin` (recomenda-se alterar)

Como Usar

Fluxo de Vendas Típico

1. **Abrir Nova Venda**
   - Clique em `Vendas` → `Nova Venda`
   - Selecione o cliente ou crie novo
2. **Adicionar Produtos**
   - Busque o produto pelo nome ou ID
   - Defina quantidade e preço (padrão do estoque)
   - Clique `Adicionar ao Carrinho`
3. **Aplicar Descontos (Opcional)**
   - Por item: edite o valor
   - Total: campo desconto no topo
4. **Confirmar Venda**
   - Revise o resumo
   - Selecione forma de pagamento
   - Configure parcelamento se necessário
   - Clique `Finalizar`
5. **Gerar Comprovante**
   - PDF será gerado automaticamente
   - Opção para imprimir diretamente

Fluxo de Estoque

1. **Novo Produto**
   - `Estoque` → `Novo Item`
   - Selecione categoria
   - Preencha dados específicos da categoria
   - Configure preço de custo e venda
2. **Entrada de Produtos**
   - Quando receber nota fiscal
   - `Estoque` → `Entrada de Produtos`
   - Vincule com pedido de compra (se houver)
   - Confirme quantidades
3. **Consultar Estoque**
   - Painel principal mostra resumo visual
   - Filtros por categoria, faixa de preço, disponibilidade
   - Alertas para estoque baixo

Módulos do Sistema

/src/main/java/

API - Integrações externas

```
api/
├── CardGamesApi.java          # Integração com APIs de TCGs
└── PokeTcgApi.java            # Pokémon TCG específico
```

App - Inicialização

```
app/
└── Main.java                  # Ponto de entrada, splash screen, inicialização
```

Controller - Controladores

```
controller/
├── EstoqueController.java     # Gerencia operações de estoque
├── ProdutoEstoqueController.java
├── VendaController.java       # Gerencia carrinho e vendas
└── [...]
```

DAO - Acesso a Dados (50+ classes)

```
dao/
├── CartaDAO.java              # CRUD de cartas individuais
├── BoosterDAO.java            # CRUD de boosters
├── ClienteDAO.java            # CRUD de clientes
├── VendaDAO.java              # CRUD de vendas
├── EstoqueDAO.java            # Consultas complexas de estoque
├── DocumentoFiscalDAO.java    # Nota fiscal
├── ContaPagarDAO.java         # Contas a pagar
├── ContaReceberDAO.java       # Contas a receber
├── [...50+ DAOs]
```

Factory - Criação de Objetos

```
factory/
└── VendaFactory.java          # Factory para criar vendas
```

Model - Modelos de Dados (60+ classes)

```
model/
├── CartaModel.java            # Representação de carta
├── BoosterModel.java          # Representação de booster
├── VendaModel.java            # Representação de venda
├── ClienteModel.java          # Representação de cliente
├── EstoqueModel.java
├── ProdutoEstoqueDTO.java     # DTO para performance
├── [...60+ Models]
```

Service - Lógica de Negócio (26 classes)

```
service/
├── VendaService.java          # Transações de venda
├── EstoqueService.java        # Gerenciamento de estoque
├── DocumentoFiscalService.java # Emissão fiscal
├── VendaDevolucaoService.java # Processamento de devoluções
├── EstornoService.java        # Processamento de estornos
├── CartaService.java          # Operações de cartas
├── ClienteService.java        # Operações de clientes
├── PedidoCompraService.java   # Gerenciamento de pedidos
├── PlanoContaService.java     # Contas contábeis
├── RelatoriosService.java     # Geração de relatórios
├── SessaoService.java         # Gerenciamento de sessão
├── [...26 Services]
```

UI - Interface Gráfica

```
ui/
├── TelaPrincipal.java         # Janela principal
├── ajustes/                   # Configurações, usuários, backup
├── clientes/                  # Gerenciamento de clientes
├── comandas/                  # Sistema de comandas (multi-mesa)
├── dash/                      # Dashboards e KPIs
├── dialog/                    # Diálogos gerais
├── estoque/                   # Gerenciamento de estoque
│   ├── dialog/                # Diálogos específicos
│   └── painel/                # Painéis de estoque
├── financeiro/                # Contas e relatórios
├── relatorios/                # Relatórios avançados
└── venda/                     # Vendas e carrinho
    ├── dialog/                # Diálogos de venda
    └── painel/                # Painel de vendas
```

Util - Utilitários

```
util/
├── DB.java                    # Gerenciador de banco de dados
├── BackupUtils.java           # Backup e restauração
├── PDFGenerator.java          # Geração de PDFs
├── LogService.java            # Sistema de logs
├── FormatUtils.java           # Formatação de dados
└── [...Utils]
```

🔧 Funcionalidades Detalhadas

1. Sistema de Vendas

Estados da Venda

Estado	Editar	Excluir	Estoque	Descrição
Aberta	☑	☑	❌	Em construção, sem impacto
Fechada	❌	❌	☑	Finalizada, impacto no estoque
Estornada	❌	❌	🔄	Revertida, estoque restaurado

Fluxo de Vendas

```
1. Criar Venda
   ↓
2. Selecionar Cliente
   ↓
3. Adicionar Itens ao Carrinho
   ↓
4. Aplicar Descontos (opcional)
   ↓
5. Revisar e Confirmar
   ↓
6. Selecionar Pagamento
   ↓
7. Configurar Parcelamento (se necessário)
   ↓
8. Finalizar e Gerar Comprovante
   ↓
9. Estoque é automaticamente atualizado
```

Métodos de Pagamento

- 💵 Dinheiro
- 💳 Cartão de Crédito/Débito
- 📱 PIX

- 🏦 Transferência Bancária
- 💳 Múltiplas formas na mesma venda

Funcionalidades Avançadas

- **Parcelamento Inteligente**: Cálculo automático de juros
- **Descontos Flexíveis**: Por item, total, percentual ou fixo
- **Histórico Completo**: Rastreamento de todas as alterações
- **Estorno Autorizado**: Necessita credenciais admin
- **Reabertura de Venda**: Somente por administrador

---

2. Gestão de Estoque

Categorias Suportadas

Categoria	Especificidades	Exemplos
Cartas	Set, número, raridade, condição	Charizard #4, Pikachu #25
Boosters	Quantidade de packs, tipo	Booster Box x36, Booster x10
Decks	Lista de cartas, estratégia	Pikachu Starter, Blastoise Control
ETBs	Conteúdo específico	Charizard ETB, Pikachu ETB
Acessórios	Marca, tipo, cor	Sleeves Deck Protector, Playmat
Alimentício	Validade, temperatura	Refrigerante, Chips

Operações de Estoque

1. **Cadastro**: Entrada inicial de produtos
2. **Entrada**: Recebimento de compras
3. **Saída**: Automaticamente por venda
4. **Movimentação**: Ajustes e perdas
5. **Exclusão**: Com histórico (nunca apagado)

Alertas Automáticos

- ⚠ Estoque baixo (< 5 unidades)
- ⚠ Sem estoque (quantidade 0)
- ⚠ Validade próxima (15 dias)

Dashboards de Estoque

- Total em unidades
- Total em valor (custo/venda)
- Produtos por categoria
- Estoque mínimo recomendado
- PMZ (Preço Médio Ponderado)

3. Módulo Financeiro

Contas a Pagar

```
Funcionalidades:
├── Cadastro de despesas
├── Vencimentos com lembretes
├── Múltiplas parcelas
├── Juros e multas
├── Registro de pagamentos
├── Histórico por fornecedor
└── Relatórios de fluxo
```

Contas a Receber

```
Funcionalidades:
├── Vendas parceladas
├── Acompanhamento de parcelas
├── Geração de boletos
├── Notificações de vencimento
├── Pré-pagamento
├── Cancelamento de parcelas
└── Relatórios por cliente
```

Crédito de Loja

```
Funcionalidades:
├── Saldo de crédito por cliente
├── Uso na venda (abate)
├── Recargas de crédito
├── Histórico de movimentação
└── Relatórios de crédito em uso
```

Plano de Contas

- Integração com código NCM
- Classificação fiscal (CSOSN, CFOP)
- Natureza de operação
- Histórico de contas

---

4. Documentos Fiscais

Tipos de Documentos

Tipo	Sigla	Uso
Nota Fiscal Eletrônica	NFe	Venda B2B
Nota Fiscal ao Consumidor	NFC-e	Venda B2C (balcão)
Cupom Fiscal	CF	PDV simplificado
Nota de Devolução	NFd	Devolução de mercadoria

Campos Obrigatórios

- ☑ Cliente (CPF/CNPJ)
- ☑ Itens (produto, quantidade, preço)
- ☑ Totalizadores (base, imposto, total)
- ☑ Pagamento (forma, valor)
- ☑ CFOP (Código Fiscal de Operação)
- ☑ CSOSN (Código de Situação)

Geração de Comprovantes

- PDF com QR Code
- Impressão direta em impressora térmica
- XML conforme padrão NF-e
- Assinatura digital (compatível)

---

5. Relatórios e Dashboards

Dashboard Principal

```
┌─────────────────────────────────────────┐
│  HoStore - Dashboard                    │
├─────────────────────────────────────────┤
│ Vendas Hoje: R$ 2.500,00   (↑ 15%)     │
│ Estoque: 1.250 unidades   (↑ 3%)      │
│ Clientes Ativos: 245       (↑ 8%)      │
│ Caixa Disponível: R$ 5.000              │
├─────────────────────────────────────────┤
│ Produtos Mais Vendidos     │ Contas     │
│ 1. Booster Escarlata       │ A Pagar    │
│ 2. Pikachu V               │ R$ 3.200   │
│ 3. Charizard EX            │ A Receber  │
│    ...                     │ R$ 1.500   │
└─────────────────────────────────────────┘
```

Relatórios Disponíveis

1. **Vendas**

- Por período (dia, mês, ano)
- Por cliente
- Por produto
- Análise de margem

2. **Estoque**
    - Movimentação
    - Validade
    - Custo x Venda
    - ABC (Curva)
3. **Financeiro**
    - Fluxo de caixa
    - Contas a pagar
    - Contas a receber
    - Resultado do período
4. **Clientes**
    - Ranking de vendas
    - Histórico de compras
    - Análise de crédito
    - Tendências

Exportação

- 📊 **Excel**: Planilhas formatadas, gráficos
- 📄 **PDF**: Relatórios profissionais com logo
- 📋 **CSV**: Para importação em sistemas externos
- 🖨️ **Impressão**: Direto para impressora

📊 Estrutura de Dados

Tabelas Principais (SQLite)

```
-- Vendas
tabelas_venda
├── vendas
├── vendas_itens
├── vendas_pagamentos
├── vendas_devolucao
└── vendas_auditoria

-- Estoque
tabelas_estoque
├── cartas
├── boosters
```

```
├── decks
├── etb
├── acessorios
├── produtos
├── estoque (view consolidada)
└── movimentacao_estoque

-- Clientes
tabelas_cliente
├── clientes
└── cliente_endereco

-- Fiscal
tabelas_fiscal
├── documento_fiscal
├── documento_fiscal_itens
├── documento_fiscal_pagamentos
├── ncm
├── cfop
└── csosn

-- Financeiro
tabelas_financeiro
├── contas_pagar
├── contas_receber
├── credito_loja
├── plano_contas
└── banco

-- Sistema
tabelas_sistema
├── usuarios
├── sessao
├── logs_auditoria
└── configuracoes
```

Exemplo: Tabela de Vendas

```
CREATE TABLE vendas (
    id TEXT PRIMARY KEY,
    cliente_id TEXT NOT NULL,
    data_venda DATETIME DEFAULT CURRENT_TIMESTAMP,
    total_bruto REAL NOT NULL,
    total_desconto REAL DEFAULT 0.0,
    total_liquido REAL NOT NULL,
    forma_pagamento TEXT,
    status TEXT CHECK(status IN ('aberta', 'fechada', 'estornada')),
    criado_por TEXT,
    criado_em DATETIME DEFAULT CURRENT_TIMESTAMP,
    alterado_por TEXT,
    alterado_em DATETIME,
```

```
    FOREIGN KEY(cliente_id) REFERENCES clientes(id)
);

CREATE TABLE vendas_itens (
    id TEXT PRIMARY KEY,
    venda_id TEXT NOT NULL,
    produto_id TEXT NOT NULL,
    quantidade INTEGER NOT NULL,
    preco_unitario REAL NOT NULL,
    desconto REAL DEFAULT 0.0,
    subtotal REAL NOT NULL,
    FOREIGN KEY(venda_id) REFERENCES vendas(id)
);
```

🌐 APIs Integradas

1. Pokémon TCG API

```
// Sincronizar todos os sets
String setsJson = PokeTcgApi.listarSetsPokemon();

// Obter cartas de um set específico
String cardsJson = PokeTcgApi.listarCardsPorSet("sv01");

// Cache automático em ./data/cache/pokemontcg_sets.json
```

Endpoint: `https://api.pokemontcg.io/v2/`

2. Magic: The Gathering (Scryfall)

```
// Listar sets
String setsJson = CardGamesApi.listarSetsMagic();

// Buscar cartas por set
String cardsJson = CardGamesApi.listarCardsMagicPorSet("sld");

// Cache: ./data/cache/magic_sets.json
```

Endpoint: `https://api.scryfall.com/`

3. Yu-Gi-Oh! (YGOPRODeck)

```
// Listar todos os sets
String setsJson = CardGamesApi.listarSetsYgo();

// Cartas por set
```

```
String cardsJson = CardGamesApi.listarCardsYgoPorSet("Metal Raiders");

// Cache: ./data/cache/yugioh_sets.json
```

Endpoint: `https://db.ygoprodeck.com/api/v7/`

4. Digimon Card Game

```
// Todas as cartas
String allCards = CardGamesApi.listarCardsDigi();

// Cache: ./data/cache/digimon_all_cards.json
```

Endpoint: `https://digimoncard.io/api-public/`

5. One Piece TCG

```
// Listar sets
String setsJson = CardGamesApi.listarSetsOnepiece();

// Cartas por set
String cardsJson = CardGamesApi.listarCardsOnepiecePorSet("OP01");

// Cache: ./data/cache/onepiece_sets.json
```

Endpoint: `https://optcgapi.com/api/`

Política de Cache

- ☑ Cache local em `./data/cache/`
- ☑ Sincronização automática diária
- ☑ Fallback offline automático
- ☑ Compressão de dados
- ☑ Limite de requisições (rate limiting)

⌨ Atalhos de Teclado

Geral

Atalho	Ação
`Ctrl+N`	Nova venda
`Ctrl+S`	Salvar
`Ctrl+P`	Imprimir

Atalho	Ação
`Ctrl+O`	Abrir/Buscar
`Esc`	Fechar diálogo/Cancelar
`F5`	Atualizar/Recarregar

Estoque

Atalho	Ação
`F2`	Focar busca
`F3`	Focar tabela
`Del`	Excluir selecionado
`Ctrl+E`	Nova entrada
`Ctrl+M`	Movimentação

Vendas

Atalho	Ação
`Ctrl+V`	Abrir novo produto
`Ctrl+D`	Aplicar desconto
`Ctrl+F`	Finalizar
`Tab`	Próximo campo
`Shift+Tab`	Campo anterior

🛠️ Desenvolvimento

Setup Ambiente

```
# 1. Clonar
git clone https://github.com/oDuPrado/HoStore.git
cd HoStore

# 2. Instalar dependências
mvn clean install

# 3. Verificar estrutura
ls -la src/main/java/

# 4. Executar testes
mvn test
```

```
# 5. Build
mvn clean package
```

Estrutura de Branches

```
main (produção)
├── develop (desenvolvimento)
│   ├── feature/vendas-v2
│   ├── feature/fiscal-integrado
│   ├── bugfix/estoque-calculo
│   └── hotfix/login-issue
```

Convenções de Código

```
// Nomes de classes (PascalCase)
public class VendaNovaDialog { }
public class EstoqueService { }

// Nomes de métodos (camelCase)
public void adicionarItem() { }
public List<VendaModel> listarVendas() { }

// Nomes de variáveis (camelCase)
private String clienteId;
private double totalLiquido;

// Constantes (UPPERCASE)
private static final String DB_PATH = "./data/hostore.db";
private static final int MAX_RETRIES = 3;

// Comentários para métodos complexos
/**
 * Finaliza venda com validações e transação atômica.
 *
 * @param venda Objeto de venda a ser finalizado
 * @param itens Lista de itens da venda
 * @return ID da venda criada
 * @throws SQLException Se erro no banco
 * @throws IllegalArgumentException Se validação falhar
 */
public int finalizarVenda(VendaModel venda, List<VendaItemModel> itens) { }
```

Testing

```
# Executar todos os testes
mvn test
```

```
# Teste específico
mvn test -Dtest=VendaServiceTest

# Com cobertura
mvn test jacoco:report
```

Gerando Documentação

```
# JavaDoc
mvn javadoc:javadoc

# Site completo
mvn site
```

Roadmap

Version 1.0.0 (Atual)

- [x] Módulo de vendas básico
- [x] Gestão de estoque
- [x] Clientes simples
- [x] Relatórios PDF
- [x] Integração Pokémon TCG

Version 1.1.0 (Próxima)

- [ ] Fiscal integrado (NFC-e)
- [ ] Múltiplos usuários e permissões
- [ ] Dashboard avançado com gráficos
- [ ] Sistema de comandas (multi-mesa)
- [ ] Integração Magic the Gathering

Version 1.2.0

- [ ] Sistema de Franquia (multi-loja)
- [ ] Sincronização em nuvem
- [ ] App mobile (vendedor)
- [ ] PDV integrado
- [ ] Geolocalização de estoque

Version 2.0.0

- [ ] Reescrever UI em JavaFX moderno
- [ ] API REST para integrações
- [ ] Analytics avançado (Machine Learning)

- [ ] Suporte a múltiplos idiomas
- [ ] Aplicativo web (versão lightweightt)

🤝 Contribuindo

Quer contribuir? Ótimo! Siga os passos:

1. **Fork** o repositório
2. **Crie** uma branch para sua feature (`git checkout -b feature/MinhaFeature`)
3. **Commit** suas mudanças (`git commit -m 'Adiciona MinhaFeature'`)
4. **Push** para a branch (`git push origin feature/MinhaFeature`)
5. **Abra** um Pull Request

Diretrizes

- Seguir convenções de código (ver acima)
- Adicionar testes para novas features
- Atualizar documentação
- Manter compatibilidade com Java 17+

📝 Licença

Este projeto é licenciado sob a **MIT License** - veja o arquivo LICENSE para detalhes.

📞 Suporte e Contato

Problemas Conhecidos

1. **Cache de API muito grande**: Limpar pasta `./data/cache/` se ocupar >500MB
2. **Performance em estoque > 10k itens**: Considerar paginar resultados
3. **Impressão em alguns drivers**: Testar USB direto

FAQ

P: Como faço backup? R: Automático diariamente em `./data/backup/`. Manual via `Menu → Ajustes → Backup`.

P: Posso usar múltiplas lojas? R: Não na v1.0. Está previsto para v2.0 com sistema de franquia.

P: Qual é o limite de produtos? R: Sem limite oficial. Testado com ~50k itens. Performance aceitável com índices corretos.

P: É seguro? Salva dados na nuvem? R: SQLite local por padrão. Nuvem será opcional na v1.1 com criptografia.

👨‍💻 Autores

- **oDuPrado** - Desenvolvedor Principal
  - GitHub: @oDuPrado

🙏 Agradecimentos

- Apache Software Foundation (POI, PDFBox, Commons)
- Formdev (FlatLaf)
- Community Java & Open Source

Última atualização: Janeiro 2026 | **Versão**: 1.0.0 | **Status**: Em Desenvolvimento Ativo

> 💡 **Dica**: Acesse a documentação técnica completa em `./Estrturas/` para detalhes de arquitetura.

🗂️ Índice de Arquivos Documentação

- 📦 Módulo de Estoque - Documentação técnica detalhada
- 🛒 Módulo de Vendas - Fluxos e regras de negócio
- 🔧 Estrutura do Projeto - Árvore completa de arquivos

Desenvolvido com ❤️ para a comunidade TCG