MAICOSOFT

Sistema de Gestão Empresarial
para Construção Civil

Documentação do Projeto

Equipe:

Denise Korgiski de Oliveira

João Vinicius Fonseca Diniz

Luiz Antonio Pereira Lima

Leandro Nicolas Silveira Gonçalves

2º semestre 2025 - Fatec Itapetininga

5 ADS noturno

SUMÁRIO

SUMÁRIO (CONTINUAÇÃO)

SUMÁRIO (CONTINUAÇÃO)

SUMÁRIO (CONTINUAÇÃO)

SUMÁRIO (CONTINUAÇÃO)

1. EQUIPE DE DESENVOLVIMENTO

João Vinícius Fonseca Diniz

Líder do Projeto

Gestão, Planejamento, Organização

Denise Korgiski de Oliveira

Dev Full Stack / DBA

Backend, Banco de Dados, Arquitetura

Luiz Antonio Pereira Lima

Dev Front-end

Interface, UX/UI, Responsividade

Leandro Nicolas Silveira Gonçalves

QA Specialist

Testes, Qualidade, Validação

2. PROBLEMA E SOLUÇÃO

2.1 Problema Identificado

A Maicosoft enfrenta desafios na gestão de seus processos comerciais, incluindo:

- Controle manual e descentralizado de vendas
- Dificuldade no acompanhamento de clientes e histórico de compras
- Gestão inadequada de cupons promocionais
- Falta de relatórios consolidados para tomada de decisão
- Processos operacionais lentos e suscetíveis a erros

2.2 Solução Proposta

Sistema web integrado que oferece:

- **Gestão Completa de Clientes:** Cadastro, edição e histórico detalhado
- **Controle de Vendas:** Registro e acompanhamento de todas as transações
- **Sistema de Cupons:** Criação e validação de promoções
- **Dashboard Analítico:** Visualização em tempo real de indicadores
- **Relatórios Gerenciais:** Exportação de dados para análise
- **Gestão de Usuários:** Controle de acesso e perfis

> **Nota:** Para informações detalhadas sobre **Requisitos Funcionais e Não-Funcionais**, **Justificativas das Escolhas Tecnológicas**, **Casos de Uso**, **Diagramas UML** e **Personas**, consulte a **Seção 7 (páginas 59-84)** deste documento, que apresenta a análise completa de Engenharia de Software e Regras de Negócio do sistema.

3. CONFIGURAÇÃO DOCKER E WSL

O projeto utiliza **Docker** para containerização, garantindo portabilidade e consistência entre ambientes de desenvolvimento e produção. Para Windows, utilizamos o **WSL (Windows Subsystem for Linux)** para evitar o uso do Docker Desktop.

3.1 Instalação do Windows Terminal

Primeiro, instale o Windows Terminal para uma melhor experiência com linha de comando:

> **Link de Download:**
> https://apps.microsoft.com/detail/9n0dx20hk701

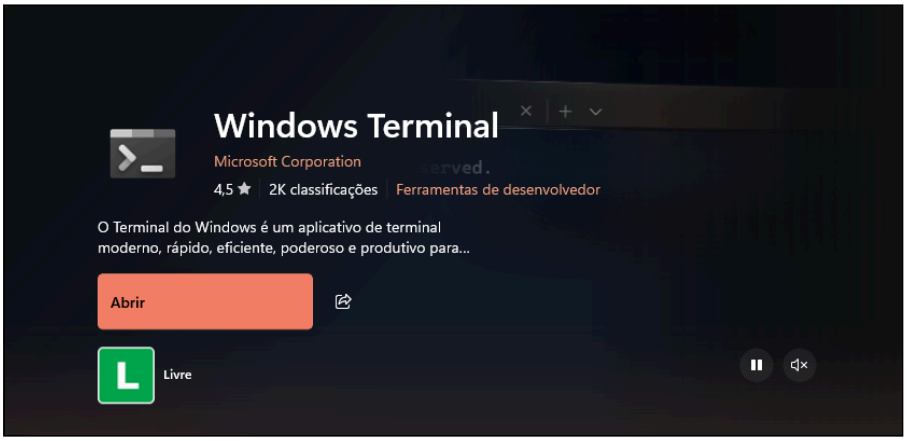


Figura 1: Windows Terminal instalado

3.2 Instalação do WSL

O WSL permite executar um ambiente Linux diretamente no Windows, sendo essencial para desenvolvimento com Docker sem overhead do Docker Desktop.

Passo 1: Instalar WSL

```
wsl --install
```

Passo 2: Instalar Ubuntu

```
wsl --install -d Ubuntu
```

Passo 3: Definir Ubuntu como padrão

```
wsl --set-default Ubuntu
```

> **Documentação Oficial:**
> https://learn.microsoft.com/pt-br/windows/wsl/install

3.2 Instalação do WSL (continuação)

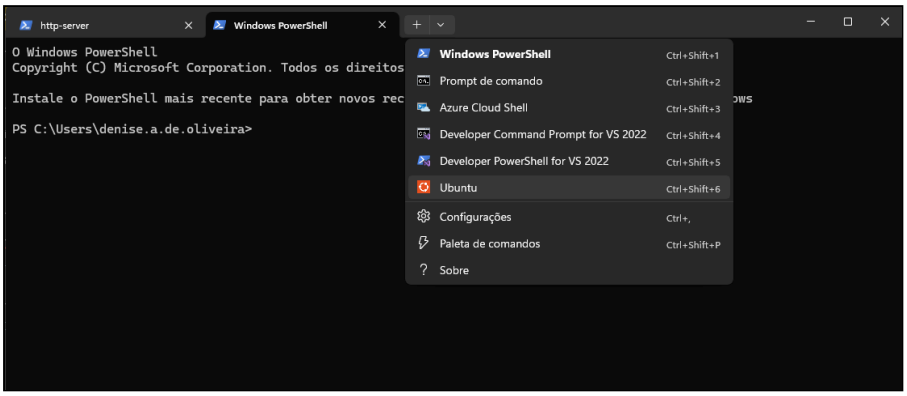


Figura 2: Acessando Ubuntu via Windows Terminal

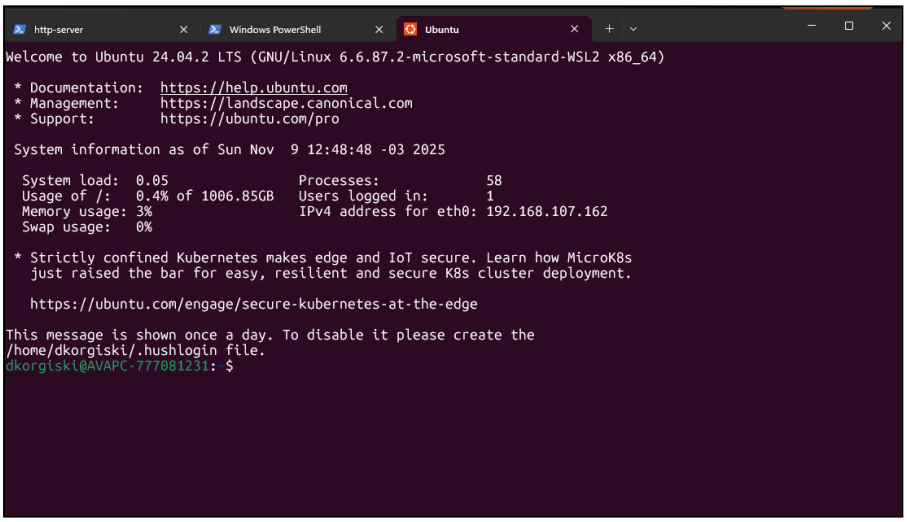


Figura 3: Ambiente Ubuntu rodando no WSL

3.3 Configuração do Ambiente Linux

3.3.1 Instalar GIT no Ubuntu

Após abrir o Ubuntu no Windows Terminal, instale o GIT:

```
# Atualizar lista de pacotes
sudo apt-get update

# Instalar GIT
sudo apt-get install git-all
```

3.3.2 Instalar Docker no Ubuntu

Siga o guia oficial de instalação do Docker no Ubuntu:

> **Documentação Oficial:**
> https://docs.docker.com/engine/install/ubuntu/

3.3.2 Instalar Docker no Ubuntu (continuação)

```
# Adicionar repositorio Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -

echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg

# Instalar Docker
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plug

# Permitir usar Docker sem sudo
sudo usermod -aG docker $USER

# Reiniciar para aplicar permissões
exit
# Abra o terminal novamente
```

> **Importante:** Após executar o comando `usermod`, é necessário reiniciar o terminal ou fazer logout/login para que as permissões sejam aplicadas.

3.4 Executando o Projeto com Docker

3.4.1 Estrutura Docker do Projeto

O projeto possui uma stack completa com 3 containers:

Container	Imagem	Porta	Descrição
postgres	postgres:15-alpine	5432	Banco de dados PostgreSQL
backend	Custom (Spring Boot)	8090	API REST Java
frontend	nginx:alpine	3000	Servidor web Nginx

Controle de Versão: O projeto utiliza **Git** como sistema de controle de versionamento, hospedado no GitHub (`https://github.com/nisze/MaicoSoft.git`). O Git permite rastreamento de todas as alterações do código, trabalho colaborativo da equipe e deployment automatizado via `git clone`.

3.4.2 Executando o Projeto (Deploy Completo)

Para executar o projeto em uma máquina nova (primeira execução), use o seguinte comando único:

> **Comando de Deploy Completo (Primeira Execução):**
> Este comando funciona em qualquer máquina Linux nova, criando toda estrutura necessária automaticamente.

```
mkdir -p ~/projetos && cd ~/projetos && \
rm -rf MaicoSoft 2>/dev/null ; \
git clone https://github.com/nisze/MaicoSoft.git && \
cd MaicoSoft && \
docker compose up -d --build && \
echo "Aguardando 30 segundos para inicialização..." && sleep 30 && \
echo -e "\n========================================" && \
echo -e "       STATUS DOS CONTAINERS" && \
echo -e "========================================\n" && \
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}" && \
echo -e "\n========================================" && \
echo -e "         TESTANDO LOGIN API" && \
echo -e "========================================\n" && \
curl -X POST http://localhost:8090/api/users/login \
  -H "Content-Type: application/json" \
  -d '{"codigoAcesso":"ADM001","senha":"123456"}' && \
echo -e "\n\n========================================" && \
echo -e "        ACESSO AO SISTEMA" && \
echo -e "========================================" && \
echo -e "\nAcesse no navegador:" && \
echo -e "http://$(hostname -I | awk '{print $1}'):3000\n" && \
echo -e "Credenciais:" && \
echo -e "Código: ADM001" && \
echo -e "Senha: 123456\n" && \
echo -e "========================================"
```

3.4.2 Script Automatizado Completo

O script automatizado (visto na seção anterior) executa todas estas etapas:

1. Cria a pasta `~/projetos` (se não existir)
2. Remove diretório do projeto anterior (se existir)
3. Clona o repositório do GitHub
4. Entra na pasta do projeto
5. Constrói e inicia todos os containers (postgres, backend, frontend)
6. Aguarda 30 segundos para inicialização completa
7. Exibe status formatado dos containers
8. Testa autenticação via API REST
9. Mostra URL de acesso e credenciais do sistema

> **Credenciais de Acesso:**
> **Código de Acesso:** ADM001
> **Senha:** 123456

3.4.3 Configuração de Comunicação entre Containers

Para garantir o correto funcionamento da aplicação no ambiente Docker, é necessário ajustar a configuração da URL da API no frontend. Esta mudança permite a comunicação adequada entre os containers.

Alteração Necessária no Código

Arquivo: `frontend/js/config.js`

Configuração Antiga (não funciona no Docker):

```
production: {
    API_BASE_URL: '/api', // URL relativa
    ENABLE_MOCK: false,
    DEBUG: false
}
```

Configuração Correta (para Docker):

```
production: {
    API_BASE_URL: 'http://localhost:8090/api', // URL completa
    ENABLE_MOCK: false,
    DEBUG: false
}
```

3.4.3 Configuração de Comunicação entre Containers (continuação)

Por que esta mudança é necessária?

- **Isolamento de Containers:** Cada container Docker tem seu próprio namespace de rede
- **Comunicação via localhost:** O frontend (porta 3000) acessa o backend (porta 8090) através do host
- **Proxy Nginx:** O servidor Nginx do frontend precisa saber exatamente onde encontrar a API
- **URLs Relativas não funcionam:** A URL `/api` seria interpretada como `http://localhost:3000/api`, que não existe

> **Importante:** Esta configuração é específica para ambiente Docker local. Em produção com domínio próprio, você pode usar URLs relativas ou o domínio configurado.

3.4.4 Inicialização Simplificada

Para usuários que preferem uma abordagem mais direta, basta clonar o repositório e executar o Docker Compose:

```
# Passo 1: Clonar o repositório
git clone https://github.com/nisze/MaicoSoft.git
cd MaicoSoft

# Passo 2: Iniciar com Docker Compose
docker compose up -d --build
```

O que este comando faz:

- **Cria 3 containers:** PostgreSQL, Backend (Spring Boot), Frontend (Nginx)
- **Compila o backend:** Maven baixa dependências e gera o .jar
- **Configura o banco:** Flyway executa migrations criando tabelas
- **Inicia serviços:** Backend (8090), Frontend (3000), Postgres (5432)

3.4.5 Comandos Docker Individuais

Se preferir executar os comandos separadamente:

```
# Iniciar todos os containers
docker compose up -d

# Ver status dos containers
docker ps

# Ver logs de um container específico
docker logs -f maiconsoft-backend

# Parar todos os containers
docker compose down

# Parar e remover volumes (apaga dados)
docker compose down -v

# Reconstruir containers
docker compose up -d --build

# Acessar terminal de um container
docker exec -it maiconsoft-postgres bash

# Testar API de login
curl -X POST http://localhost:8090/api/users/login \
  -H "Content-Type: application/json" \
  -d '{"codigoAcesso":"ADM001","senha":"123456"}'
```

3.4.6 Resultado da Execução

Após executar o comando de deploy, você verá a seguinte saída:

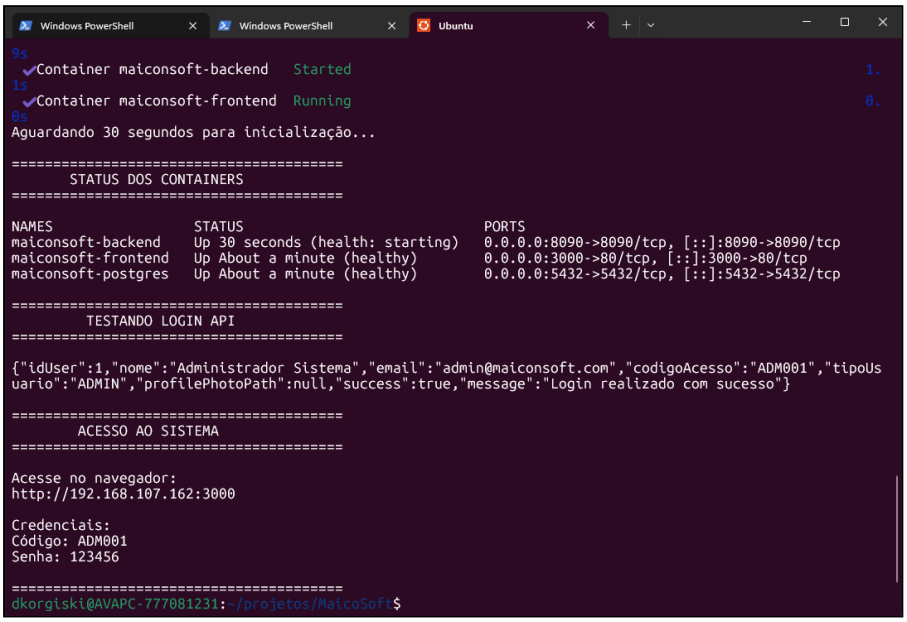


Figura 4: Execução completa do deploy com Docker

> **Verificações de Sucesso:**
>
> - ✅ 3 containers rodando (postgres, backend, frontend)
> - ✅ Status "healthy" para todos os containers
> - ✅ Teste de login retorna `"success":true`
> - ✅ URL de acesso exibida no final

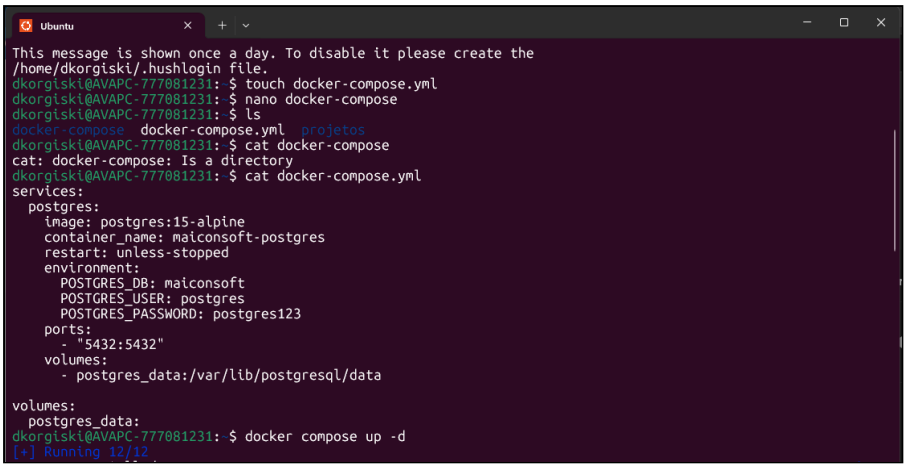


Figura 5: Container do banco de dados em execução

Scripts Prontos (Opcional)

Scripts Prontos: O projeto inclui scripts automatizados na pasta `scripts/` para facilitar o gerenciamento:

- `start.bat` - Inicia todos os containers
- `stop.bat` - Para todos os containers
- `logs.bat` - Mostra logs em tempo real
- `status.bat` - Exibe status e recursos
- `rebuild.bat` - Reconstrói os containers
- `reset.bat` - Reset completo do sistema

5. BANCO DE DADOS

4.1 PostgreSQL

O projeto utiliza **PostgreSQL 15** como sistema de gerenciamento de banco de dados relacional. PostgreSQL é um dos bancos de dados open-source mais robustos e confiáveis, oferecendo recursos avançados como transações ACID, suporte a JSON, replicação e alta performance.

Principais Características:

- **ACID Compliance:** Garantia de integridade transacional
- **Alta Performance:** Otimizado para operações complexas
- **Extensível:** Suporte a tipos de dados customizados
- **Confiável:** Sistema de backup e recuperação robusto
- **Open Source:** Sem custos de licenciamento

Configuração Docker:

```
postgres:
  image: postgres:15-alpine
  container_name: maiconsoft-postgres
  environment:
    POSTGRES_DB: maiconsoft_db
    POSTGRES_USER: maiconsoft_user
    POSTGRES_PASSWORD: maiconsoft_pass
  ports:
    - "5432:5432"
  volumes:
    - postgres_data:/var/lib/postgresql/data
```

4.2 Flyway Migration

O projeto utiliza **Flyway** para controle de versão do banco de dados. Flyway permite gerenciar e aplicar migrações de forma automatizada e versionada, garantindo que todos os ambientes (desenvolvimento, teste, produção) possuam a mesma estrutura de banco de dados.

Vantagens do Flyway:

- Versionamento automático do schema
- Migrations aplicadas apenas uma vez
- Rastreabilidade completa de mudanças
- Rollback facilitado em caso de erros
- Integração nativa com Spring Boot

Estrutura de Migrations:

```
src/main/resources/db/migration/
├── V1__Initial_Schema.sql
├── V2__Insert_Initial_Data.sql
├── V3__Remove_Security.sql
├── V4__Update_User_Data.sql
├── V5__Fix_User_Data.sql
├── V6__Add_User_Timestamps.sql
├── V7__Add_User_Auth_Fields.sql
├── V8__Popular_Dados_Exemplo.sql
├── V9__Add_Profile_Photo_Column.sql
├── V10__Remove_TipoUsuario_Column.sql
├── V11__Add_Profile_Photo_Column.sql
├── V12__Add_User_Active_Column.sql
└── V13__Add_Test_Users.sql
```

4.3 Modelo de Dados

O banco de dados foi modelado seguindo os princípios de normalização e boas práticas de design relacional, garantindo integridade referencial e evitando redundância de dados.

Principais Tabelas:

Tabela	Descrição	Relacionamentos
`users`	Usuários do sistema	roles, vendas, clientes
`roles`	Perfis de acesso	users
`clientes`	Cadastro de clientes	users, vendas
`vendas`	Registro de vendas	users, clientes, vendas_itens
`vendas_itens`	Itens de cada venda	vendas, servicos
`servicos`	Serviços oferecidos	vendas_itens
`cupons`	Cupons promocionais	vendas

Relacionamentos e Chaves Estrangeiras:

O sistema utiliza chaves estrangeiras (Foreign Keys) para garantir a integridade referencial entre as tabelas, impedindo inconsistências de dados.

`users` ↔ `roles`

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `users.id_role` → `roles.id_role`
- **Descrição:** Cada usuário possui um único perfil de acesso (ADMIN, DIRETOR, FUNCIONARIO, VENDEDOR)
- **Exemplo:** O usuário ADM001 possui `id_role = 1` (ADMIN)

`clientes` ↔ `users`

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `clientes.id_usuario_cadastro` → `users.id_user`
- **Descrição:** Registra qual usuário cadastrou cada cliente no sistema
- **Comportamento:** Campo opcional (pode ser NULL)

`vendas` ↔ `users`

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `vendas.id_usuario` → `users.id_user`
- **Descrição:** Identifica qual usuário realizou a venda
- **Integridade:** ON DELETE SET NULL (se usuário for deletado, venda mantém histórico)

vendas ↔ ***clientes***

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `vendas.id_cliente` → `clientes.id_cliente`
- **Descrição:** Vincula cada venda a um cliente específico
- **Integridade:** Campo obrigatório (NOT NULL)

vendas ↔ ***cupons***

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `vendas.id_cupom` → `cupons.id_cupom`
- **Descrição:** Aplica cupom de desconto à venda (opcional)
- **Comportamento:** Campo opcional (pode ser NULL se não houver desconto)

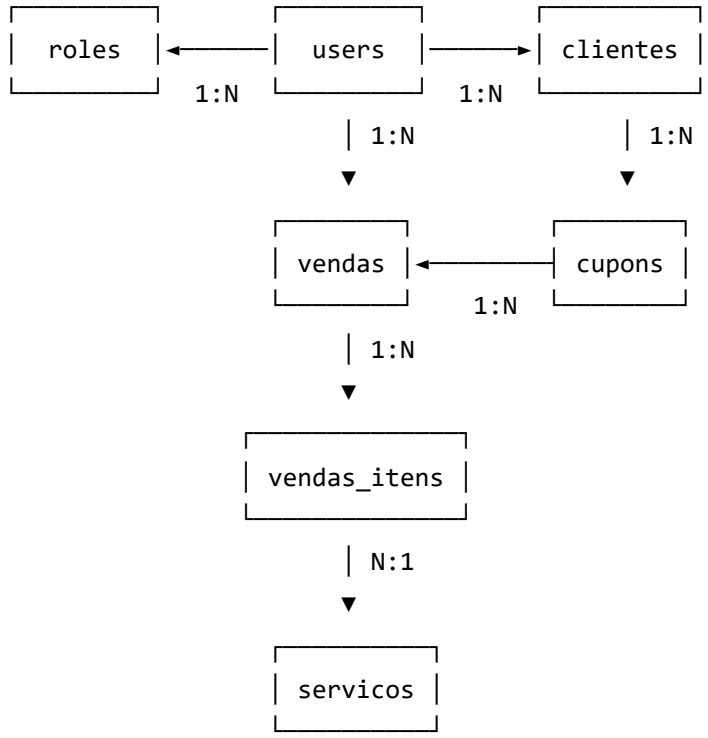
vendas_itens ↔ ***vendas***

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `vendas_itens.id_venda` → `vendas.id_venda`
- **Descrição:** Tabela de itens que compõem cada venda
- **Integridade:** ON DELETE CASCADE (se venda for deletada, itens também são removidos)

vendas_itens ↔ ***servicos***

- **Relacionamento:** Muitos-para-Um (N:1)
- **Chave Estrangeira:** `vendas_itens.id_servico` → `servicos.id_servico`
- **Descrição:** Identifica qual serviço foi vendido em cada item
- **Dados Armazenados:** Quantidade, preço unitário, subtotal

Diagrama de Relacionamento Simplificado:



6. BACKEND - API SPRING BOOT

5.1 Spring Boot 3.5.5

O backend foi desenvolvido utilizando **Spring Boot 3.5.5**, o framework Java mais popular para desenvolvimento de aplicações enterprise. Spring Boot simplifica a configuração e permite desenvolvimento rápido com convenções inteligentes.

Principais Dependências:

Tecnologia	Versão	Finalidade
Spring Boot	3.5.5	Framework base
Spring Data JPA	3.5.5	ORM e persistência
Spring Web	3.5.5	REST API
Spring Boot Mail	3.5.5	Envio de emails
Spring Security Crypto	3.5.5	BCrypt para hash de senhas
PostgreSQL Driver	Runtime	Conexão com PostgreSQL
Flyway Core	10.x	Migrations de banco
Flyway PostgreSQL	10.x	Suporte PostgreSQL para Flyway

Principais Dependências (continuação):

Tecnologia	Versão	Finalidade
H2 Database	Runtime	Banco em memória (dev)
Thymeleaf	3.5.5	Templates de email
Lombok	Latest	Redução de boilerplate
SpringDoc OpenAPI	2.8.9	Documentação Swagger/OpenAPI
Caffeine	3.2.2	Cache de alta performance
Logstash Logback	7.4	Logs estruturados em JSON

Detalhamento das Principais Dependências:

Spring Boot Mail + Thymeleaf: Utilizados em conjunto para o sistema de envio de emails HTML. O Thymeleaf processa os templates HTML localizados em `resources/templates/email/`, permitindo emails personalizados para boas-vindas de clientes, notificações de vendas e recuperação de senha.

Caffeine Cache: Implementado especificamente para armazenar tokens de recuperação de senha de forma temporária e segura. Os tokens expiram automaticamente após um período configurado, garantindo segurança no processo de reset de senha.

SpringDoc OpenAPI: Gera automaticamente a documentação interativa da API no formato Swagger/OpenAPI, acessível via `/swagger-ui.html`. Facilita testes e integração com a API.

Logstash Logback: Estrutura os logs da aplicação em formato JSON, facilitando análise, monitoramento e integração com ferramentas de observabilidade como ELK Stack (Elasticsearch, Logstash, Kibana).

H2 Database: Banco de dados em memória utilizado para testes e desenvolvimento rápido, permitindo desenvolvimento sem necessidade de PostgreSQL rodando localmente.

5.2 Arquitetura em Camadas

O backend segue a arquitetura em camadas, separando responsabilidades e facilitando manutenção, testes e escalabilidade.

Estrutura de Pacotes:

```
com.faculdae.maiconsoft_api/
├── MaiconsoftApiApplication.java  # Classe principal
├── controllers/       # Endpoints REST
│   ├── ClienteController.java
│   ├── CupomController.java
│   ├── DashboardController.java
│   ├── FileUploadController.java
│   ├── HealthController.java
│   ├── PasswordResetController.java
│   ├── UserController.java
│   └── VendaController.java
├── services/          # Lógica de negócio
│   ├── CodigoAcessoService.java
│   ├── DashboardService.java
│   ├── audit/         # Auditoria
│   ├── cliente/       # Serviços de cliente
│   ├── cupom/         # Serviços de cupom
│   ├── email/         # Envio de emails
│   ├── external/      # Integrações externas (ViaCEP)
│   ├── user/          # Serviços de usuário
│   └── venda/         # Serviços de venda
├── repositories/      # Acesso a dados (JPA)
│   ├── ClienteRepository.java
│   ├── CupomRepository.java
│   ├── DashboardLogRepository.java
│   ├── UserRepository.java
│   ├── UserRoleRepository.java
│   └── VendaRepository.java
```

Estrutura de Pacotes (continuação):

```
com.faculdae.maiconsoft_api/ (continuação)
├── entities/           # Entidades JPA
│   ├── Cliente.java
│   ├── Cupom.java
│   ├── DashboardLog.java
│   ├── Pagamento.java
│   ├── Servico.java
│   ├── User.java
│   ├── UserRole.java
│   └── Venda.java
├── dto/                # Data Transfer Objects
│   ├── DashboardMetricsDTO.java
│   ├── LoginRequestDTO.java
│   ├── LoginResponseDTO.java
│   ├── UserResponseDTO.java
│   ├── auth/           # DTOs de autenticação
│   ├── cliente/        # DTOs de cliente
│   ├── cupom/          # DTOs de cupom
│   ├── user/           # DTOs de usuário
│   ├── venda/          # DTOs de venda
│   └── viacep/         # DTOs de integração ViaCEP
├── specification/      # Filtros e buscas avançadas
│   └── ClienteSpecification.java
└── config/             # Configurações
    ├── CorsConfig.java
    ├── EmailConfig.java
    ├── GlobalExceptionHandler.java
    ├── RateLimitConfig.java
    ├── RateLimitFilter.java
    ├── RestTemplateConfig.java
    ├── SecurityConfig.java
    ├── SimpleCorsFilter.java
    ├── SwaggerConfig.java
    └── WebConfig.java
```

5.3 Organização da Arquitetura

Por que essa estrutura de pacotes?

A organização do projeto segue o padrão **Package by Feature** combinado com **Separação de Responsabilidades (SoC)**. Esta abordagem traz diversos benefícios:

- **Manutenibilidade:** Cada camada tem responsabilidade bem definida, facilitando localização e correção de bugs
- **Escalabilidade:** Novos módulos podem ser adicionados sem impactar código existente
- **Testabilidade:** Camadas isoladas permitem testes unitários independentes
- **Modularização:** Services organizados em subpacotes (cliente, venda, email) facilitam trabalho em equipe
- **Clean Architecture:** Controllers não conhecem detalhes de persistência, Services não conhecem detalhes HTTP

Spring Data JPA Specifications

O projeto utiliza **Spring Data JPA Specifications** para implementar buscas dinâmicas e filtros complexos de forma type-safe e reutilizável.

Specifications são baseadas no padrão **Criteria API** do JPA e permitem construir queries de forma programática, evitando SQL nativo e proporcionando:

- **Type Safety:** Erros detectados em tempo de compilação
- **Reutilização:** Specifications podem ser combinadas com operadores AND/OR
- **Flexibilidade:** Queries dinâmicas baseadas em parâmetros opcionais

Como funciona a busca no banco de dados?

Ao invés de criar múltiplos métodos `get` no Repository para cada combinação possível de filtros (ex: `findByNome()`, `findByCpf()`, `findByNomeAndCpf()`, etc.), o Specification permite criar **uma única busca personalizada** que aceita parâmetros opcionais.

O Spring Data JPA converte as Specifications em queries SQL automaticamente, aplicando apenas os filtros que foram fornecidos. Por exemplo:

- Se apenas o **nome** for informado → `SELECT * FROM clientes WHERE nome LIKE '%valor%'`
- Se apenas o **CPF** for informado → `SELECT * FROM clientes WHERE cpf = 'valor'`
- Se **ambos** forem informados → `SELECT * FROM clientes WHERE nome LIKE '%valor%' AND cpf = 'valor'`
- Se **nenhum** for informado → `SELECT * FROM clientes` (retorna todos)

Exemplo de uso do Specification:

```
// ClienteSpecification.java
public class ClienteSpecification {

    public static Specification buscarPorNome(String nome) {
        return (root, query, criteriaBuilder) -> {
            if (nome == null || nome.isEmpty()) {
                return criteriaBuilder.conjunction();
            }
            return criteriaBuilder.like(
                criteriaBuilder.lower(root.get("nome")),
                "%" + nome.toLowerCase() + "%"
            );
        };
    }
```

Exemplo de uso do Specification (continuação):

```
    public static Specification buscarPorCpf(String cpf) {
        return (root, query, criteriaBuilder) -> {
            if (cpf == null || cpf.isEmpty()) {
                return criteriaBuilder.conjunction();
            }
            return criteriaBuilder.equal(root.get("cpf"), cpf);
        };
    }
}

// Uso no Service
Specification spec = Specification
    .where(buscarPorNome(nome))
    .and(buscarPorCpf(cpf));

List clientes = clienteRepository.findAll(spec);
```

Documentação Oficial:
Spring Data JPA Specifications
https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html

5.4 Principais Endpoints da API

A API REST fornece endpoints para todas as operações do sistema, seguindo os princípios RESTful e retornando dados em formato JSON.

Endpoints de Usuários:

Método	Endpoint	Descrição
POST	/api/users/login	Autenticação de usuário
GET	/api/users	Listar todos os usuários
GET	/api/users/{id}	Buscar usuário por ID
POST	/api/users	Criar novo usuário
PUT	/api/users/{id}	Atualizar usuário
DELETE	/api/users/{id}	Deletar usuário

Endpoints de Clientes:

Método	Endpoint	Descrição
GET	/api/clientes	Listar todos os clientes
GET	/api/clientes/{id}	Buscar cliente por ID
POST	/api/clientes	Criar novo cliente
PUT	/api/clientes/{id}	Atualizar cliente
DELETE	/api/clientes/{id}	Deletar cliente

Endpoints de Vendas:

Método	Endpoint	Descrição
GET	/api/vendas	Listar todas as vendas
GET	/api/vendas/{id}	Buscar venda por ID
POST	/api/vendas	Registrar nova venda
PUT	/api/vendas/{id}	Atualizar venda
DELETE	/api/vendas/{id}	Deletar venda

Endpoints de Cupons:

Método	Endpoint	Descrição
GET	/api/cupons	Listar todos os cupons
GET	/api/cupons/{id}	Buscar cupom por ID
POST	/api/cupons	Criar novo cupom
PUT	/api/cupons/{id}	Atualizar cupom
DELETE	/api/cupons/{id}	Deletar cupom
POST	/api/cupons/validar	Validar código de cupom

Outros Endpoints:

Categoria	Endpoint	Descrição
Dashboard	/api/dashboard/metrics	Métricas gerais do sistema
Dashboard	/api/dashboard/logs	Logs de atividades
Upload	/api/file/upload	Upload de arquivos
Password	/api/password/reset-request	Solicitar reset de senha
Password	/api/password/reset	Resetar senha com token

5.7 Regras de Negócio

O backend implementa diversas regras de negócio essenciais para o funcionamento do sistema de gestão. Abaixo estão detalhadas as principais funcionalidades e seus fluxos de processamento.

5.7.1 Processamento de Vendas

O sistema processa vendas com validação completa de dados e aplicação automática de cupons de desconto quando aplicável. O método `save()` do `VendaService` implementa o fluxo completo:

Passo 1 - Validação de Cliente:

```
// VendaService.java - linha 57
Cliente cliente = clienteService.findEntityById(requestDTO.clienteId());
```

O método `findEntityById()` do `ClienteService` busca o cliente no banco e lança exceção se não encontrado:

```
// ClienteService.java
Cliente cliente = clienteRepository.findById(id)
    .orElseThrow(() -> new EntityNotFoundException(
        "Cliente não encontrado com ID: " + id));
```

Passo 2 - Validação de Cupom:

```
// VendaService.java - linha 67
if (!cupomService.podeUsarCupom(requestDTO.cupomCodigo())) {
    throw new RuntimeException(
        "Cupom não pode ser usado: inativo, expirado ou limite atingido"
    );
}
```

O método `podeUsarCupom()` verifica 3 condições:

```
// CupomService.java - linha 272
// 1. Verificar se cupom está ATIVO
if (!"ATIVO".equals(cupom.getStatus())) {
    return false;
}

// 2. Verificar se não expirou
if (cupom.getValidade().isBefore(LocalDate.now())) {
    return false;
}

// 3. Verificar limite de usos
if (cupom.getMaxUsos() != null && usosAtuais >= cupom.getMaxUsos()) {
    return false;
}
```

Passo 3 - Cálculo de Valores:

```
// VendaService.java - linha 76
BigDecimal valorBruto = requestDTO.valorBruto();
BigDecimal valorDesconto = calcularDesconto(valorBruto, cupom);
BigDecimal valorTotal = valorBruto.subtract(valorDesconto);
```

5.7.2 Envio de Comprovante por E-mail

Após registro da venda, o sistema envia automaticamente um e-mail de notificação. O processo utiliza Thymeleaf para templates HTML:

Implementação no VendaService:

```
// VendaService.java - linha 115
try {
    emailService.enviarNotificacaoNovaVenda(
        venda.getUsuarioCadastro().getNome(),
        cliente.getNomeFantasia(),
        vendaSalva.getValorTotal().doubleValue()
    );
    log.info("Email de notificação enviado");
} catch (Exception e) {
    log.error("Erro ao enviar email: {}", e.getMessage());
    // Não falha a venda por causa do email
}
```

EmailService com Template Thymeleaf:

```
// EmailService.java - linha 93
Context context = new Context();
context.setVariable("nomeCliente", nomeCliente);
context.setVariable("codigoCliente", codigoCliente);

// Processar template HTML
String htmlContent = templateEngine.process(
    "email/cliente-welcome", context
);

// Criar mensagem MIME para HTML
MimeMessage mimeMessage = mailSender.createMimeMessage();
MimeMessageHelper helper = new MimeMessageHelper(
    mimeMessage, true, "UTF-8"
);

helper.setFrom(fromEmail);
helper.setTo(emailCliente);
helper.setSubject("Bem-vindo à Maiconsoft!");
helper.setText(htmlContent, true); // true = HTML content

mailSender.send(mimeMessage);
```

Características: Envio assíncrono (não bloqueia resposta da API), tratamento de exceções (venda não falha se e-mail falhar), logs detalhados para auditoria, templates HTML responsivos com Thymeleaf.

5.7.3 Sistema de Cupons

Gerenciamento completo de cupons com validações múltiplas e controle de uso:

Validação Completa do Cupom:

```
// CupomService.java - método podeUsarCupom()
public boolean podeUsarCupom(String cupomCodigo) {
    try {
        Cupom cupom = findByCodigo(cupomCodigo);

        // 1. Verificar status ATIVO
        if (!"ATIVO".equals(cupom.getStatus())) {
            return false;
        }

        // 2. Verificar validade
        if (cupom.getValidade() != null &&
            cupom.getValidade().isBefore(LocalDate.now())) {
            return false;
        }

        // 3. Verificar limite de usos
        int usosAtuais = cupom.getUsosAtual() != null ?
            cupom.getUsosAtual() : 0;
        if (cupom.getMaxUsos() != null &&
            usosAtuais >= cupom.getMaxUsos()) {
            return false;
        }

        return true;
    } catch (Exception e) {
        return false;
    }
}
```

Incremento de Uso do Cupom:

```
// VendaService.java - após salvar venda
if (cupom != null) {
    try {
        cupomService.incrementarUsoCupom(cupom.getCodigo());
        log.info("Uso do cupom {} incrementado", cupom.getCodigo());
    } catch (Exception e) {
        log.error("Erro ao incrementar cupom: {}", e.getMessage());
        // Não falha a venda por causa do cupom
    }
}
```

Desativação Automática de Cupons Expirados:

```
// CupomService.java - executado ao listar cupons
private void desativarCuponsExpirados() {
    LocalDate hoje = LocalDate.now();
    List<Cupom> cuponsExpirados =
        cupomRepository.findByStatusAndValidadeBefore("ATIVO", hoje);

    if (!cuponsExpirados.isEmpty()) {
        cuponsExpirados.forEach(cupom -> {
            log.info("Desativando cupom expirado: {}", cupom.getCodigo());
            cupom.setStatus("INATIVO");
        });
        cupomRepository.saveAll(cuponsExpirados);
    }
}
```

5.7.4 Upload de Arquivos

Sistema de upload seguro com validações e armazenamento em diretório:

Validação e Upload de Arquivo:

```
// FileUploadController.java
@PostMapping("/upload")
public ResponseEntity<?> uploadFile(
    @RequestParam("file") MultipartFile file
) {
    // Validar tipo de arquivo
    String contentType = file.getContentType();
    if (!contentType.startsWith("image/")) {
        return ResponseEntity.badRequest()
            .body("Apenas imagens são permitidas");
    }

    // Validar tamanho (5MB)
    if (file.getSize() > 5 * 1024 * 1024) {
        return ResponseEntity.badRequest()
            .body("Arquivo excede 5MB");
    }

    // Gerar nome único
    String fileName = UUID.randomUUID() + "_" +
        file.getOriginalFilename();

    // Salvar no diretório
    Path uploadPath = Paths.get("uploads/profiles/");
    Files.createDirectories(uploadPath);

    Path filePath = uploadPath.resolve(fileName);
    Files.copy(file.getInputStream(), filePath,
        StandardCopyOption.REPLACE_EXISTING);

    // Atualizar banco de dados
    user.setProfilePhotoUrl("/uploads/profiles/" + fileName);
    userRepository.save(user);

    return ResponseEntity.ok(Map.of("url", filePath.toString()));
}
```

5.7.5 Recuperação de Senha

Fluxo seguro com tokens temporários armazenados em cache Caffeine:

Passo 1 - Geração e Armazenamento de Token:

```
// PasswordResetService.java
@PostMapping("/reset-request")
public ResponseEntity<?> requestPasswordReset(
    @RequestParam String email
) {
    User user = userRepository.findByEmail(email)
        .orElseThrow(() -> new RuntimeException(
            "Usuário não encontrado"
        ));

    // Gerar token UUID único
    String token = UUID.randomUUID().toString();

    // Armazenar em cache Caffeine (expira em 1 hora)
    passwordResetTokenCache.put(token, email);

    // Enviar email com link
    emailService.enviarTokenResetSenha(
        email, user.getNome(), token
    );

    return ResponseEntity.ok("Email enviado com sucesso");
}
```

Passo 2 - Validação de Token e Reset:

```
// PasswordResetService.java
@PostMapping("/reset")
public ResponseEntity<?> resetPassword(
    @RequestParam String token,
    @RequestParam String newPassword
) {
    // Buscar email no cache pelo token
    String email = passwordResetTokenCache.getIfPresent(token);

    if (email == null) {
        return ResponseEntity.badRequest()
            .body("Token inválido ou expirado");
    }

    User user = userRepository.findByEmail(email)
        .orElseThrow(() -> new RuntimeException(
            "Usuário não encontrado"
        ));

    // Hashear nova senha com BCrypt
    String hashedPassword = passwordEncoder.encode(newPassword);
    user.setSenha(hashedPassword);
    userRepository.save(user);

    // Remover token do cache (uso único)
    passwordResetTokenCache.invalidate(token);

    return ResponseEntity.ok("Senha alterada com sucesso");
}
```

Configuração do Cache Caffeine:

```
// CacheConfig.java
@Bean
public Cache<String, String> passwordResetTokenCache() {
    return Caffeine.newBuilder()
        .expireAfterWrite(1, TimeUnit.HOURS) // Expira em 1 hora
        .maximumSize(1000)
        .build();
}
```

5.7.6 Dashboard e Métricas

Agregação de dados para visualização gerencial:

- **Total de Vendas:** Valor total e quantidade de vendas no período
- **Clientes Ativos:** Quantidade de clientes com compras recentes
- **Cupons Ativos:** Cupons válidos disponíveis para uso
- **Top Produtos:** Produtos mais vendidos com quantidade
- **Vendas por Período:** Gráficos de evolução temporal
- **Logs de Sistema:** Últimas 100 atividades registradas

5.7.7 Resumo Funcional do Backend

O backend do sistema MaicoSoft API foi desenvolvido para fornecer uma base robusta de gestão comercial com as seguintes capacidades principais:

- **Autenticação e Autorização:** Sistema seguro com roles (ADMIN/USER)
- **Gestão de Clientes:** CRUD completo com busca e filtros
- **Processamento de Vendas:** Registro detalhado com itens e cálculos automáticos
- **Sistema de Cupons:** Descontos configuráveis com controle de uso
- **Notificações por E-mail:** Comprovantes e recuperação de senha
- **Upload de Arquivos:** Gestão de fotos de perfil
- **Dashboard Analítico:** Métricas e KPIs em tempo real
- **Auditoria e Logs:** Rastreamento completo de operações

7. FRONTEND - INTERFACE WEB

6.1 Tecnologias Frontend

O frontend foi desenvolvido como uma aplicação web SPA (Single Page Application) utilizando tecnologias web modernas, garantindo responsividade e ótima experiência do usuário.

Stack de Tecnologias:

Tecnologia	Versão	Finalidade
HTML5	-	Estrutura das páginas
CSS3	-	Estilização e layout
JavaScript (ES6+)	-	Lógica e interatividade
Nginx	Alpine	Servidor web
Chart.js	4.4.1	Gráficos e dashboards
Font Awesome	6.5.1	Ícones

Características:

- **Design Responsivo:** Adapta-se a diferentes tamanhos de tela
- **SPA:** Navegação fluida sem recarregamento de página
- **API Integration:** Comunicação via Fetch API
- **Validação de Formulários:** Feedback em tempo real
- **Notificações:** Sistema de alertas e mensagens

6.2 Estrutura de Arquivos

O frontend está organizado de forma modular, separando estilos, scripts e páginas para facilitar manutenção e escalabilidade.

Organização de Pastas:

```
frontend/
├── pages/              # Páginas HTML
│   ├── login.html
│   ├── dashboard.html
│   ├── clientes.html
│   ├── vendas.html
│   ├── cupons.html
│   ├── relatorios.html
│   ├── usuarios.html
│   └── perfil.html
├── css/                # Folhas de estilo
│   ├── style.css
│   ├── login.css
│   ├── dashboard.css
│   ├── cliente.css
│   ├── navigation.css
│   ├── notifications.css
│   └── responsive.css
├── js/                 # Scripts JavaScript
│   ├── config.js
│   ├── app.js
│   ├── login.js
│   ├── dashboard.js
│   ├── cliente.js
│   ├── ajax-utils.js
│   └── user-profile.js
├── images/             # Imagens e ícones
└── assets/             # Recursos estáticos
```

6.3 Nginx como Servidor Web

O Nginx foi escolhido como servidor web por sua alta performance, baixo consumo de recursos e capacidade de atuar como reverse proxy para o backend.

Configuração Nginx:

```
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # Servir arquivos estáticos
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Proxy reverso para API
    location /api/ {
        proxy_pass http://backend:8090/api/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Cache de arquivos estáticos
    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}
```

> **Vantagens do Nginx:**
>
> - Alta performance para conteúdo estático
> - Baixo consumo de memória
> - Reverse proxy integrado
> - Suporte a HTTPS
> - Balanceamento de carga

6.4 Páginas e Funcionalidades

O sistema é composto por diversas páginas, cada uma com funcionalidades específicas e controle de acesso baseado no perfil do usuário.

6.4.1 Página de Login

Arquivo: `login.html`

Funcionalidades:

- Autenticação com código de acesso e senha
- Validação de campos em tempo real
- Toggle de visualização de senha
- Botões de login rápido para demonstração
- Redirecionamento automático baseado no perfil
- Armazenamento seguro de sessão (localStorage)

6.4.2 Dashboard

Arquivo: `dashboard.html`

Acesso: ADMIN, DIRETOR

Funcionalidades:

- **KPIs Principais:** Total de vendas, clientes ativos, cupons em uso
- **Gráficos:** Vendas por período usando Chart.js
- **Top Produtos:** Lista dos produtos mais vendidos
- **Atividades Recentes:** Últimas operações do sistema
- **Ações Rápidas:** Botões para cadastro rápido
- **Refresh Automático:** Atualização periódica dos dados

6.4.3 Gestão de Clientes

Arquivo: `clientes.html`

Acesso: Todos os usuários autenticados

Funcionalidades:

- **Cadastro:** Formulário completo com validações (CPF/CNPJ, CEP, Email)
- **Listagem:** Tabela paginada com todos os clientes
- **Busca:** Filtro por nome, código, CPF/CNPJ
- **Edição:** Modal para atualização de dados
- **Exclusão:** Com confirmação de segurança
- **Responsivo:** Design adaptável para mobile

6.4.4 Gestão de Vendas

Arquivo: `vendas.html`

Acesso: ADMIN, DIRETOR, VENDEDOR

Funcionalidades:

- **Nova Venda:** Seleção de cliente, produtos e aplicação de cupons
- **Cálculo Automático:** Valores, descontos e totais
- **Histórico:** Lista de todas as vendas
- **Filtros:** Por período, cliente, status
- **Envio de Comprovante:** Via email
- **Impressão:** Geração de PDF da venda

6.4.5 Gestão de Cupons

Arquivo: `cupons.html`

Acesso: ADMIN, DIRETOR

Funcionalidades:

- **Criação:** Cupons com desconto percentual ou valor fixo
- **Configurações:** Validade, valor mínimo, limite de uso
- **Status:** Ativo, Inativo, Expirado
- **Monitoramento:** Acompanhamento de uso dos cupons
- **Desativação:** Manual ou automática por expiração

6.4.6 Gestão de Usuários

Arquivo: `usuarios.html`

Acesso: ADMIN

Funcionalidades:

- **Cadastro:** Novos usuários com perfil e permissões
- **Perfis Disponíveis:** ADMIN, DIRETOR, FUNCIONARIO, VENDEDOR
- **Ativação/Desativação:** Controle de acesso ao sistema
- **Edição:** Atualização de dados e senha
- **Listagem:** Todos os usuários com status

6.4.7 Relatórios

Arquivo: `relatorios.html`

Acesso: ADMIN, DIRETOR

Funcionalidades:

- **Vendas por Período:** Filtro customizado de datas
- **Estatísticas:** Métricas de performance
- **Top Clientes:** Clientes com maior volume de compras
- **Análise de Cupons:** Efetividade dos descontos
- **Exportação:** Relatórios em PDF e Excel
- **Gráficos:** Visualizações com Chart.js

6.4.8 Perfil do Usuário

Arquivo: `perfil.html`

Acesso: Todos os usuários

Funcionalidades:

- **Dados Pessoais:** Visualização e edição
- **Foto de Perfil:** Upload de imagem
- **Alteração de Senha:** Com validação de senha atual
- **Informações da Conta:** Perfil, data de cadastro

6.5 Arquitetura JavaScript

O frontend utiliza JavaScript ES6+ com arquitetura modular orientada a objetos, separando responsabilidades em classes e serviços especializados.

6.5.1 Estrutura de Arquivos JavaScript

```
js/
├── config.js           # Configurações da API e ambiente
├── app.js              # Gerenciador principal e permissões
├── login.js            # Autenticação de usuários
├── dashboard.js        # Métricas e KPIs
├── cliente.js          # CRUD de clientes
├── ajax-utils.js       # Cliente HTTP e utilitários
└── user-profile.js     # Gestão de perfil
```

6.5.2 Classes Principais

LoginManager (login.js):

```
class LoginManager {
    constructor() {
        this.isLoggingIn = false;
        this.init();
    }

    async handleLogin(codigoAcesso, senha) {
        const response = await fetch('/api/users/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ codigoAcesso, senha })
        });

        const data = await response.json();
        if (data.success) {
            localStorage.setItem('maiconsoft_user',
                JSON.stringify(data.user));
            this.redirectByRole(data.user.perfil);
        }
    }
}
```

DashboardManager (dashboard.js):

```
class DashboardManager {
    constructor() {
        this.apiClient = new ApiClient();
        this.authService = new AuthService();
        this.currentPeriod = '30d';
    }

    async loadDashboardData() {
        const metrics = await this.apiClient.get('/dashboard/metrics');
        this.updateKPIs(metrics);
        this.renderCharts(metrics);
    }
}
```

6.5.3 ApiClient - Cliente HTTP

Classe utilitária para comunicação com o backend, centralizando headers, tratamento de erros e conversão de dados.

```
class ApiClient {
    constructor() {
        this.baseURL = CONFIG.API_BASE_URL; // '/api'
    }

    async get(endpoint) {
        const response = await fetch(
            `${this.baseURL}${endpoint}`, {
            headers: this.getHeaders()
        });
        return this.handleResponse(response);
    }

    async post(endpoint, data) {
        const response = await fetch(
            `${this.baseURL}${endpoint}`, {
            method: 'POST',
            headers: this.getHeaders(),
            body: JSON.stringify(data)
        });
        return this.handleResponse(response);
    }

    getHeaders() {
        const headers = {
            'Content-Type': 'application/json'
        };

        const token = this.getAuthToken();
        if (token) {
            headers['Authorization'] = `Bearer ${token}`;
        }

        return headers;
    }
```

```
    async handleResponse(response) {
        if (!response.ok) {
            if (response.status === 401) {
                this.handleUnauthorized();
            }
            throw new Error(`HTTP ${response.status}`);
        }
        return await response.json();
    }
}
```

6.5.4 AuthService - Gerenciamento de Sessão

```
class AuthService {
    isAuthenticated() {
        const user = localStorage.getItem('maiconsoft_user');
        return user !== null;
    }

    getUser() {
        const userData = localStorage.getItem('maiconsoft_user');
        return userData ? JSON.parse(userData) : null;
    }

    hasPermission(permission) {
        const user = this.getUser();
        return user && user.permissions.includes(permission);
    }

    logout() {
        localStorage.removeItem('maiconsoft_user');
        window.location.href = 'login.html';
    }
}
```

6.6 Sistema de Permissões

O sistema implementa controle granular de acesso baseado em perfis e permissões específicas para cada operação.

6.6.1 Definição de Permissões

```
// config.js
const PERMISSIONS = {
    CLIENTES: {
        VIEW: 'clientes:view',
        CREATE: 'clientes:create',
        EDIT: 'clientes:edit',
        DELETE: 'clientes:delete',
        EXPORT: 'clientes:export'
    },
    VENDAS: {
        VIEW: 'vendas:view',
        CREATE: 'vendas:create',
        EDIT: 'vendas:edit',
        DELETE: 'vendas:delete',
        APPROVE: 'vendas:approve'
    },
    USUARIOS: {
        VIEW: 'usuarios:view',
        CREATE: 'usuarios:create',
        EDIT: 'usuarios:edit',
        DELETE: 'usuarios:delete'
    },
    RELATORIOS: {
        VIEW: 'relatorios:view',
        EXPORT: 'relatorios:export'
    }
};
```

6.6.2 Perfis e Permissões

Perfil	Permissões
ADMIN	Acesso total ao sistema, todas as permissões
DIRETOR	Dashboard, Vendas, Clientes, Relatórios, Cupons
FUNCIONARIO	Clientes (CRUD), Vendas (view)
VENDEDOR	Clientes (view), Vendas (create, view)

6.6.3 Verificação em Tempo de Execução

```
// Exemplo de uso em clientes.js
if (authService.hasPermission(PERMISSIONS.CLIENTES.CREATE)) {
    btnNovoCliente.style.display = 'block';
} else {
    btnNovoCliente.style.display = 'none';
}

// Proteção de rotas
function checkPageAccess() {
    const user = authService.getUser();
    const allowedRoles = ['admin', 'diretor'];

    if (!allowedRoles.includes(user.perfil.toLowerCase())) {
        window.location.href = 'dashboard.html';
    }
}
```

6.7 Integração com Backend

O frontend se comunica com a API REST do Spring Boot através do Nginx que atua como reverse proxy.

6.7.1 Configuração de Ambiente

```
// config.js
const CONFIG = {
    API_BASE_URL: '/api',  // Proxy do Nginx
    ENABLE_MOCK: false,
    DEBUG: true
};

// Nginx faz o redirecionamento:
// Frontend: http://localhost:3000
// Chamada: fetch('/api/clientes')
// Nginx redireciona para: http://backend:8090/api/clientes
```

6.7.2 Exemplos de Integração

Login de Usuário:

```
const response = await fetch('/api/users/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        codigoAcesso: 'ADM001',
        senha: '123456'
    })
});

const data = await response.json();
// Resposta: { success: true, user: {...}, token: '...' }
```

Criar Nova Venda:

```
const vendaData = {
    idCliente: 1,
    itens: [
        { produto: 'Cimento 50kg', quantidade: 10, valor: 35.00 }
    ],
    codigoCupom: 'DESC10'
};

const response = await fetch('/api/vendas', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(vendaData)
});

const result = await response.json();
// Backend aplica cupom e retorna venda com desconto calculado
```

Buscar Clientes:

```
const response = await fetch('/api/clientes');
const clientes = await response.json();

// Preencher tabela
clientes.forEach(cliente => {
    const row = `
        <tr>
            <td>${cliente.codigo}</td>
            <td>${cliente.razaoSocial}</td>
            <td>${cliente.email}</td>
            <td>
                <button onclick="editarCliente(${cliente.idCliente})">
                    Editar
                </button>
            </td>
        </tr>
    `;
    tbody.innerHTML += row;
});
```

6.8 Componentes e Padrões

6.8.1 Sistema de Notificações

Feedback visual para ações do usuário:

```
function showNotification(message, type = 'success') {
    const notification = document.createElement('div');
    notification.className = `notification ${type}`;
    notification.textContent = message;

    document.body.appendChild(notification);

    setTimeout(() => {
        notification.classList.add('show');
    }, 100);

    setTimeout(() => {
        notification.classList.remove('show');
        setTimeout(() => notification.remove(), 300);
    }, 3000);
}

// Uso:
showNotification('Cliente cadastrado com sucesso!', 'success');
showNotification('Erro ao salvar dados', 'error');
```

6.8.2 Validação de Formulários

```
function validateCPF(cpf) {
    cpf = cpf.replace(/[^\d]/g, '');
    if (cpf.length !== 11) return false;

    // Algoritmo de validação do CPF
    let sum = 0;
    for (let i = 0; i < 9; i++) {
        sum += parseInt(cpf.charAt(i)) * (10 - i);
    }
    // ... resto da validação
    return true;
}
```

```
function validateEmail(email) {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
}
```

6.8.3 Loading State

```
function showLoading(element) {
    element.disabled = true;
    element.innerHTML =
        '<span class="spinner"></span> Carregando...';
}

function hideLoading(element, originalText) {
    element.disabled = false;
    element.innerHTML = originalText;
}

// Uso durante requisição
const btn = document.getElementById('btnSalvar');
showLoading(btn);

try {
    await apiClient.post('/clientes', data);
    showNotification('Cliente salvo!', 'success');
} finally {
    hideLoading(btn, 'Salvar');
}
```

6.8.4 Modal Reutilizável

```
function openModal(modalId) {
    const modal = document.getElementById(modalId);
    modal.style.display = 'flex';
    document.body.style.overflow = 'hidden';
}

function closeModal(modalId) {
    const modal = document.getElementById(modalId);
    modal.style.display = 'none';
    document.body.style.overflow = 'auto';
}
```

3. REQUISITOS E ENGENHARIA DE SOFTWARE

7.1 Requisitos Funcionais (RF)

ID	Descrição	Prioridade
RF01	O sistema deve permitir autenticação de usuários com código de acesso e senha	Alta
RF02	O sistema deve permitir cadastro completo de clientes (Pessoa Física e Jurídica)	Alta
RF03	O sistema deve validar CPF/CNPJ único para cada cliente	Alta
RF04	O sistema deve integrar com ViaCEP para preenchimento automático de endereço	Média
RF05	O sistema deve permitir registrar vendas associadas a clientes	Alta
RF06	O sistema deve permitir aplicação de cupons de desconto em vendas	Média
RF07	O sistema deve controlar validade, limite de uso e status dos cupons	Média

7.1 Requisitos Funcionais (RF) - Continuação

ID	Descrição	Prioridade
RF08	O sistema deve enviar notificações por e-mail após registro de venda	Média
RF09	O sistema deve permitir upload e gerenciamento de comprovantes de venda	Média
RF10	O sistema deve permitir upload e gerenciamento de fotos de perfil de usuários	Baixa
RF11	O sistema deve emitir dashboard com métricas de vendas, clientes e cupons	Alta
RF12	O sistema deve gerar relatórios de vendas por período	Média
RF13	O sistema deve permitir busca avançada com múltiplos filtros (Specifications)	Média
RF14	O sistema deve controlar permissões baseadas em roles (ADMIN, DIRETOR, FUNCIONÁRIO, VENDEDOR)	Alta
RF15	O sistema deve permitir recuperação de senha via e-mail	Média

7.2 Requisitos Não-Funcionais (RNF)

ID	Descrição	Categoria
RNF01	O sistema deve proteger dados sensíveis com criptografia bcrypt	Segurança
RNF02	O sistema deve usar autenticação com armazenamento seguro de sessão (localStorage) e validação server-side	Segurança
RNF03	O sistema deve ser responsivo e acessível em dispositivos móveis	Usabilidade
RNF04	O sistema deve responder em menos de 3 segundos para 95% das requisições	Performance
RNF05	O sistema deve ser compatível com Chrome, Firefox, Edge e Safari	Compatibilidade
RNF06	O banco de dados deve suportar transações ACID	Confiabilidade
RNF07	O sistema deve usar versionamento de schema com Flyway	Manutenibilidade
RNF08	O sistema deve ser escalável horizontalmente com Docker	Escalabilidade
RNF09	O sistema deve ter disponibilidade de 99% em horário comercial	Disponibilidade
RNF10	O sistema deve registrar logs detalhados de operações críticas	Auditoria

7.3 Justificativas das Escolhas Tecnológicas

Spring Boot (Backend)

- **Produtividade:** Configuração rápida com Spring Boot Starter e convenções inteligentes
- **Ecossistema robusto:** JPA/Hibernate para persistência, BCryptPasswordEncoder para criptografia de senhas
- **Comunidade ativa:** Grande volume de documentação e suporte
- **Arquitetura em camadas:** Separação clara entre Controller, Service, Repository

PostgreSQL (Banco de Dados)

- **Open source:** Sem custos de licenciamento
- **Confiabilidade:** Suporte robusto a transações ACID
- **Performance:** Otimizações avançadas para consultas complexas
- **Escalabilidade:** Suporte a índices, particionamento e replicação
- **Extensibilidade:** Tipos de dados avançados (JSON, Arrays)

HTML/CSS/JavaScript Vanilla (Frontend)

- **Simplicidade:** Sem dependências externas ou build tools
- **Performance:** Carregamento rápido sem frameworks pesados
- **Aprendizado:** Didático para entendimento dos fundamentos web
- **Compatibilidade:** Funciona em qualquer navegador moderno

Docker (DevOps)

- **Portabilidade:** Ambiente padronizado em qualquer máquina
- **Isolamento:** Cada serviço em container separado
- **Escalabilidade:** Fácil replicação e orquestração

7.3.1 Flyway (Migrations)

O Flyway é uma ferramenta de migração de banco de dados que permite versionar e gerenciar mudanças no schema do banco de dados de forma automática e controlada.

Principais Benefícios:

- **Versionamento:** Controle de mudanças no schema do banco
- **Rastreabilidade:** Histórico completo de alterações
- **Automação:** Execução automática ao iniciar aplicação
- **Consistência:** Garante que todos os ambientes tenham o mesmo schema
- **Rollback seguro:** Possibilidade de reverter mudanças quando necessário

Estrutura de Migrations no Projeto:

As migrations estão localizadas em `src/main/resources/db/migration/` e seguem o padrão de nomenclatura:

- `V1__Initial_Schema.sql` - Schema inicial do banco
- `V2__Insert_Initial_Data.sql` - Dados iniciais
- `V3__Remove_Security.sql` - Ajustes de segurança
- E assim sucessivamente...

Execução: O Flyway executa automaticamente todas as migrations pendentes ao iniciar a aplicação Spring Boot, garantindo que o banco de dados esteja sempre atualizado com a versão mais recente do schema.

7.4 Casos de Uso Principais

UC01 - Autenticar Usuário (Parte 1)

Ator Principal: Usuário do sistema (qualquer role)

Pré-condições: Usuário possui código de acesso e senha válidos cadastrados no sistema

Fluxo Principal:

1. Usuário acessa a tela de login (login.html)
2. Sistema exibe formulário de autenticação
3. Usuário preenche código de acesso (formato case-insensitive) e senha
4. Sistema valida credenciais no banco de dados usando BCrypt para comparar senha
5. Sistema retorna dados do usuário (ID, nome, email, código, role, foto de perfil)
6. Sistema armazena dados no localStorage
7. Sistema redireciona para dashboard apropriado conforme role do usuário

UC01 - Autenticar Usuário (Parte 2 - Fluxos Alternativos)

Fluxo Alternativo A - Credenciais Inválidas:

1. No passo 4, sistema identifica credenciais incorretas
2. Sistema exibe mensagem "Código de acesso ou senha inválidos"
3. Sistema mantém usuário na tela de login

Fluxo Alternativo B - Usuário Inativo:

1. No passo 4, sistema identifica usuário com status inativo (ativo=false)
2. Sistema exibe mensagem "Usuário desativado. Contate o administrador"
3. Sistema bloqueia acesso e não permite login

Pós-condições: Usuário autenticado com dados armazenados no localStorage e acesso ao sistema conforme role (ADMIN, DIRETOR, FUNCIONARIO, VENDEDOR)

UC02 - Cadastrar Cliente (Parte 1)

Ator Principal: Usuário autenticado (qualquer role)

Pré-condições: Usuário autenticado no sistema

Fluxo Principal:

1. Usuário acessa módulo de clientes
2. Sistema exibe lista de clientes cadastrados
3. Usuário clica em "Novo Cliente"
4. Sistema exibe formulário de cadastro
5. Usuário preenche dados obrigatórios (Código, Loja, Razão Social, Tipo, CPF/CNPJ)
6. Usuário preenche CEP
7. Sistema consulta ViaCEP e preenche automaticamente Endereço, Bairro, Cidade, Estado
8. Usuário preenche dados complementares (Telefone, Email, etc.)
9. Usuário clica em "Salvar"
10. Sistema valida unicidade do CPF/CNPJ
11. Sistema salva cliente no banco de dados
12. Sistema exibe mensagem de sucesso
13. Sistema atualiza lista de clientes

UC02 - Cadastrar Cliente (Parte 2 - Fluxos Alternativos)

Fluxo Alternativo A - CPF/CNPJ Duplicado:

1. No passo 10, sistema identifica CPF/CNPJ já cadastrado
2. Sistema exibe mensagem "CPF/CNPJ já cadastrado no sistema"
3. Sistema mantém usuário no formulário com dados preenchidos

Fluxo Alternativo B - CEP Inválido:

1. No passo 7, ViaCEP retorna erro (CEP não encontrado)
2. Sistema exibe mensagem "CEP não encontrado"
3. Sistema permite preenchimento manual do endereço

Pós-condições: Cliente cadastrado no sistema e disponível para associação com vendas

UC03 - Registrar Venda com Cupom (Parte 1)

Ator Principal: Vendedor, Funcionário, Diretor ou Admin

Pré-condições: Usuário autenticado, cliente cadastrado, cupom ativo disponível

Fluxo Principal:

1. Usuário acessa módulo de vendas
2. Sistema exibe lista de vendas e botão "Nova Venda"
3. Usuário clica em "Nova Venda"
4. Sistema exibe formulário de venda
5. Usuário busca e seleciona cliente
6. Sistema carrega dados do cliente selecionado
7. Usuário preenche número do orçamento e valor bruto
8. Usuário seleciona cupom de desconto (opcional)
9. Sistema valida cupom (ativo, não expirado, dentro do limite de uso)
10. Sistema calcula desconto aplicado (percentual ou valor fixo)
11. Sistema calcula valor total (valor bruto - desconto)
12. Usuário preenche observações e seleciona status
13. Usuário clica em "Salvar Venda"
14. Sistema incrementa contador de uso do cupom
15. Sistema envia e-mail de notificação para vendedor
16. Sistema salva venda no banco de dados
17. Sistema exibe mensagem de sucesso

UC03 - Registrar Venda com Cupom (Parte 2 - Fluxos Alternativos)

Fluxo Alternativo A - Cupom Expirado:

1. No passo 9, sistema identifica cupom com data de validade vencida
2. Sistema exibe mensagem "Cupom expirado"
3. Sistema bloqueia aplicação do cupom
4. Usuário pode prosseguir sem cupom ou selecionar outro

Fluxo Alternativo B - Limite de Uso Atingido:

1. No passo 9, sistema identifica que uso atual ≥ máximo de usos
2. Sistema exibe mensagem "Cupom esgotado (limite atingido)"
3. Sistema bloqueia aplicação do cupom

Fluxo Alternativo C - Valor Mínimo Não Atingido:

1. No passo 10, sistema verifica que valor bruto < valor mínimo do cupom
2. Sistema exibe mensagem "Valor mínimo R$ X.XX não atingido"
3. Sistema não aplica desconto

Pós-condições: Venda registrada, cupom incrementado, e-mail enviado, dados auditados

7.5 Diagramas UML

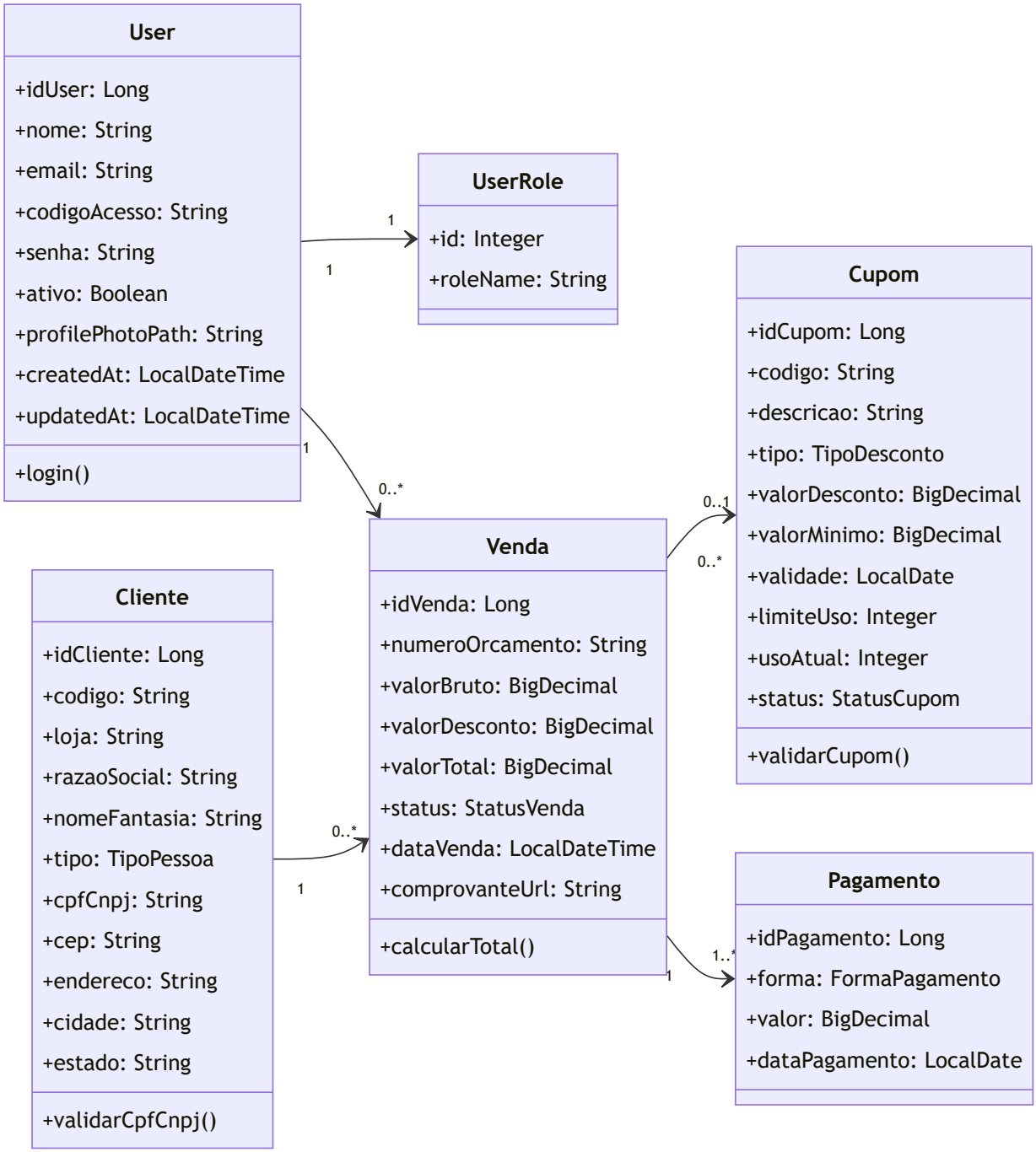
7.5.1 Diagrama de Classes

O diagrama de classes representa as principais entidades do sistema e seus relacionamentos. As classes modeladas refletem a estrutura do banco de dados e a camada de domínio da aplicação:

- **User:** Representa usuários do sistema com autenticação e roles
- **UserRole:** Define os níveis de acesso (ADMIN, DIRETOR, FUNCIONARIO, VENDEDOR)
- **Cliente:** Entidade para clientes (Pessoa Física ou Jurídica)
- **Venda:** Registros de vendas com relacionamentos múltiplos
- **Cupom:** Cupons de desconto com validação e controle de uso
- **Pagamento:** Formas de pagamento associadas às vendas

Relacionamentos:

- User possui um UserRole (1:1)
- User registra várias Vendas (1:N)
- Cliente participa de várias Vendas (1:N)
- Venda pode ter um Cupom aplicado (N:1, opcional)
- Venda possui uma ou mais formas de Pagamento (1:N)


User
+idUser: Long
+nome: String
+email: String
+codigoAcesso: String
+senha: String
+ativo: Boolean
+profilePhotoPath: String
+createdAt: LocalDateTime
+updatedAt: LocalDateTime
+login()
UserRole
+id: Integer
+roleName: String
Cupom
+idCupom: Long
+codigo: String
+descricao: String
+tipo: TipoDesconto
+valorDesconto: BigDecimal
+valorMinimo: BigDecimal
+validade: LocalDate
+limiteUso: Integer
+usoAtual: Integer
+status: StatusCupom
+validarCupom()
Venda
+idVenda: Long
+numeroOrcamento: String
+valorBruto: BigDecimal
+valorDesconto: BigDecimal
+valorTotal: BigDecimal
+status: StatusVenda
+dataVenda: LocalDateTime
+comprovanteUrl: String
+calcularTotal()
Cliente
+idCliente: Long
+codigo: String
+loja: String
+razaoSocial: String
+nomeFantasia: String
+tipo: TipoPessoa
+cpfCnpj: String
+cep: String
+endereco: String
+cidade: String
+estado: String
+validarCpfCnpj()
Pagamento
+idPagamento: Long
+forma: FormaPagamento
+valor: BigDecimal
+dataPagamento: LocalDate
1
1
1
0..*
0..1
0..*
0..*
1
1..*
1

7.5.2 Diagrama de Sequência - Registrar Venda com Cupom

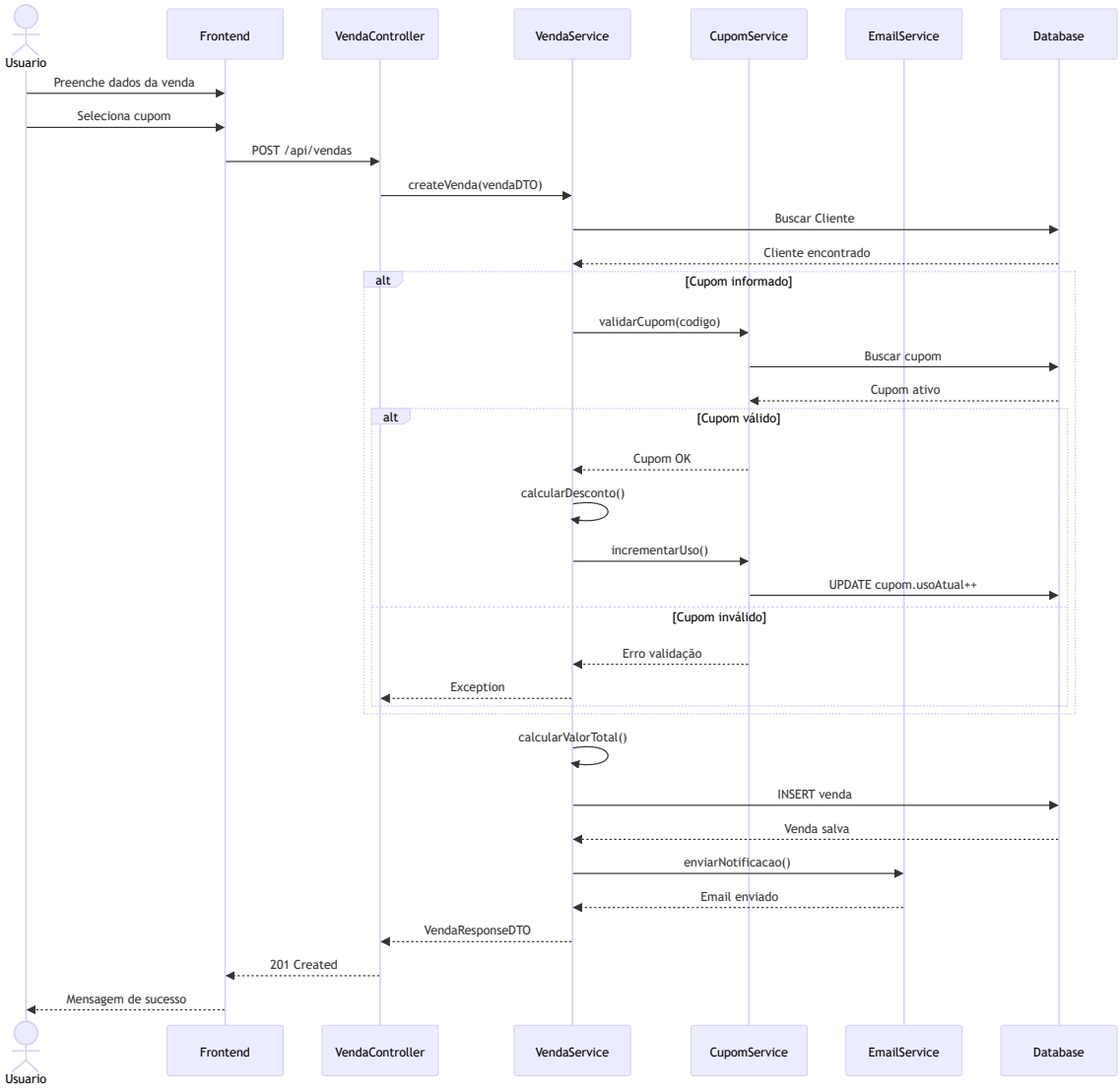
O diagrama de sequência ilustra o fluxo de comunicação entre os componentes do sistema durante o processo de registro de uma venda com aplicação de cupom de desconto.

Participantes:

- **Usuario:** Ator que interage com o sistema
- **Frontend:** Interface JavaScript (vendas.js)
- **VendaController:** REST Controller (@PostMapping /api/vendas)
- **VendaService:** Camada de lógica de negócio
- **CupomService:** Serviço de validação e gestão de cupons
- **EmailService:** Serviço de envio de notificações por email
- **Database:** PostgreSQL via JPA/Hibernate

Fluxo Principal:

1. Usuário preenche dados da venda e seleciona cupom (opcional)
2. Frontend envia requisição POST para API
3. Sistema valida cliente e cupom (se informado)
4. Calcula desconto e valor total
5. Incrementa contador de uso do cupom
6. Salva venda no banco de dados
7. Envia email de notificação
8. Retorna confirmação ao usuário


Usuario
Frontend
VendaController
VendaService
CupomService
EmailService
Database
Preenche dados da venda
Seleciona cupom
POST /api/vendas
createVenda(vendaDTO)
Buscar Cliente
Cliente encontrado
alt
[Cupom informado]
validarCupom(codigo)
Buscar cupom
Cupom ativo
alt
[Cupom válido]
Cupom OK
calcularDesconto()
incrementarUso()
UPDATE cupom.usoAtual++
[Cupom inválido]
Erro validação
Exception
calcularValorTotal()
INSERT venda
Venda salva
enviarNotificacao()
Email enviado
VendaResponseDTO
201 Created
Mensagem de sucesso

7.7 Personas do Sistema

As personas representam perfis fictícios baseados em usuários reais do sistema MaicoSoft. Cada persona foi desenvolvida considerando as necessidades, objetivos e desafios específicos de cada tipo de usuário, auxiliando na compreensão dos requisitos e na validação das funcionalidades implementadas.

7.7.1 Metodologia de Criação

As personas foram criadas a partir de:

- **Análise de roles do sistema:** ADMIN, DIRETOR, FUNCIONARIO, VENDEDOR
- **Casos de uso mapeados:** Autenticação, gestão de clientes, registro de vendas
- **Contexto empresarial:** Pequenas e médias empresas de varejo
- **Requisitos funcionais:** Necessidades identificadas durante o levantamento

7.7.2 Estrutura das Personas

Cada persona contém:

- **Dados demográficos:** Nome, idade, cargo, experiência
- **Perfil profissional:** Responsabilidades e contexto de trabalho
- **Objetivos:** O que busca alcançar com o sistema
- **Dores e frustrações:** Problemas que enfrenta no dia a dia
- **Necessidades do sistema:** Funcionalidades prioritárias
- **Cenários de uso:** Situações típicas de interação

👨‍💼 *Perfil: Carlos Silva*

Idade: 32 anos

Cargo: Gerente de TI

Experiência: 8 anos em gestão de sistemas

Formação: Tecnologia da Informação

Role no Sistema: ADMIN

Frequência de Uso: Diária

Nível Técnico: Avançado

Local: São Paulo, SP

Perfil Profissional

Carlos é responsável pela infraestrutura tecnológica da empresa, gerenciando sistemas, usuários e garantindo a segurança dos dados. Trabalha diretamente com a diretoria para implementar soluções que melhorem a eficiência operacional.

Objetivos

- Garantir que todos os usuários tenham acesso adequado ao sistema
- Gerenciar perfis de acesso e permissões de forma eficiente
- Monitorar a saúde do sistema e prevenir problemas
- Facilitar a recuperação de senhas de forma segura
- Manter auditoria completa de ações críticas

Dores e Frustrações

- Perder tempo com solicitações manuais de reset de senha
- Dificuldade em rastrear quem fez alterações em dados críticos
- Falta de visibilidade sobre uso e performance do sistema
- Processos complexos para criar novos usuários

👨‍💼 *Carlos Silva (Continuação)*

Necessidades do Sistema

- ✅ Gestão completa de usuários (CRUD)
- ✅ Sistema de recuperação de senha automatizado
- ✅ Logs de auditoria detalhados
- ✅ Geração automática de códigos de acesso
- ✅ Controle de status ativo/inativo de usuários

Cenário de Uso Típico

"Toda segunda-feira pela manhã, Carlos acessa o sistema para verificar novos funcionários admitidos. Ele cria os usuários rapidamente usando o formulário do MaicoSoft, que gera automaticamente códigos de acesso seguros. Quando algum colaborador esquece a senha, Carlos pode enviar um token de recuperação com apenas 2 cliques, sem precisar resetar manualmente."

Impacto no Sistema MaicoSoft

As necessidades de Carlos impulsionaram o desenvolvimento de:

- **PasswordResetService:** Sistema automatizado com tokens temporários via Caffeine Cache
- **UserService:** CRUD completo com geração automática de códigos de acesso
- **Controle de Roles:** UserRole entity para gestão granular de permissões
- **BCrypt:** Criptografia robusta para proteção de senhas

👩‍💼 Persona 2: Maria Santos - Diretora Comercial

Idade: 45 anos

Cargo: Diretora Comercial

Experiência: 20 anos em vendas

Formação: Administração de Empresas

Role no Sistema: DIRETOR

Frequência de Uso: Diária

Nível Técnico: Intermediário

Local: Rio de Janeiro, RJ

Perfil Profissional

Maria lidera a equipe comercial e é responsável por definir estratégias de vendas, acompanhar metas e analisar o desempenho da equipe. Precisa de visão consolidada dos resultados e capacidade de tomar decisões rápidas baseadas em dados.

Objetivos

- Visualizar métricas de vendas em tempo real
- Acompanhar performance da equipe comercial
- Identificar tendências de vendas e clientes
- Criar campanhas promocionais com cupons de desconto
- Gerar relatórios gerenciais para reuniões

Dores e Frustrações

- Dificuldade em consolidar dados de múltiplas fontes
- Falta de visibilidade sobre efetividade de promoções
- Tempo excessivo para gerar relatórios manuais
- Impossibilidade de acessar dados fora do escritório

👩‍💼 Persona 2: Maria Santos (Continuação)

Necessidades do Sistema

- ✅ Dashboard com KPIs visuais (vendas, ticket médio, total clientes)
- ✅ Gestão completa de cupons de desconto
- ✅ Relatórios de vendas por período
- ✅ Filtros avançados para análise de dados
- ✅ Acesso via navegador de qualquer dispositivo

Cenário de Uso Típico

"Antes da reunião semanal com a diretoria, Maria acessa o dashboard do MaicoSoft no tablet. Em segundos, visualiza o total de vendas da semana, ticket médio e número de novos clientes. Percebe que um cupom de desconto específico teve baixa adesão e decide criar uma nova campanha com condições mais atrativas, configurando limite de uso e validade diretamente no sistema."

Impacto no Sistema MaicoSoft

As necessidades de Maria impulsionaram o desenvolvimento de:

- **DashboardService:** Métricas em tempo real com agregações SQL otimizadas
- **CupomService:** Gestão completa com validação de validade, limite de uso e status
- **Specifications:** Filtros dinâmicos para análise flexível de dados
- **Design Responsivo:** Interface acessível em desktop, tablet e mobile

👷 Persona 3: João Oliveira - Assistente Administrativo

Idade: 28 anos

Cargo: Assistente Administrativo

Experiência: 5 anos em atendimento

Formação: Gestão Comercial

Role no Sistema: FUNCIONARIO

Frequência de Uso: Diária

Nível Técnico: Básico

Local: Belo Horizonte, MG

Perfil Profissional

João trabalha no atendimento ao cliente e back-office, sendo responsável por cadastrar novos clientes, atualizar informações cadastrais e auxiliar o time de vendas com documentação e registros administrativos.

Objetivos

- Cadastrar clientes de forma rápida e sem erros
- Manter base de dados atualizada e organizada
- Reduzir retrabalho com validações automáticas
- Atender clientes com agilidade
- Facilitar o trabalho da equipe de vendas

Dores e Frustrações

- Digitar endereços completos manualmente
- Cadastrar CPF/CNPJ duplicados por engano
- Dificuldade em encontrar clientes na base
- Interface complexa que requer muito treinamento

👷 Persona 3: João Oliveira (Continuação)

Necessidades do Sistema

- ✅ Formulário de cadastro intuitivo e simples
- ✅ Integração com ViaCEP para preenchimento automático de endereço
- ✅ Validação de CPF/CNPJ único em tempo real
- ✅ Busca avançada de clientes com múltiplos filtros
- ✅ Mensagens de erro claras e orientadoras

Cenário de Uso Típico

"João recebe a ligação de um novo cliente interessado em fazer compras. Enquanto conversa ao telefone, abre o MaicoSoft e inicia o cadastro. Ao digitar o CEP, o sistema preenche automaticamente endereço, bairro, cidade e estado via ViaCEP. Quando insere o CPF, o sistema valida e confirma que é único. Em menos de 2 minutos, o cliente está cadastrado e João pode passar o código para a equipe de vendas."

Impacto no Sistema MaicoSoft

As necessidades de João impulsionaram o desenvolvimento de:

- **Integração ViaCEP:** ClienteController endpoint externo para busca automática de endereços
- **Validação Unique:** @Column(unique=true) no CPF/CNPJ com tratamento de exceção DataIntegrityViolation
- **Specifications Pattern:** Filtros dinâmicos para busca avançada de clientes
- **Interface Intuitiva:** Design responsivo com validações front-end e feedback visual

👩‍💼 Persona 4: Ana Costa - Vendedora

Idade: 26 anos

Cargo: Vendedora

Experiência: 3 anos em vendas

Formação: Marketing

Role no Sistema: VENDEDOR

Frequência de Uso: Diária

Nível Técnico: Básico

Local: Curitiba, PR

Perfil Profissional

Ana trabalha diretamente com vendas, realizando atendimento presencial e telefônico. É responsável por elaborar orçamentos, fechar vendas, aplicar cupons promocionais e enviar comprovantes aos clientes. Sua meta é maximizar conversão e ticket médio.

Objetivos

- Registrar vendas rapidamente durante atendimento
- Aplicar cupons de desconto para aumentar conversão
- Enviar comprovantes profissionais aos clientes
- Consultar histórico de compras do cliente
- Bater metas mensais de vendas

Dores e Frustrações

- Perder tempo com cálculos manuais de desconto
- Aplicar cupons vencidos ou esgotados por engano
- Dificuldade em anexar comprovantes de pagamento
- Não receber notificação automática sobre a venda

👩‍💼 Persona 4: Ana Costa (Continuação)

Necessidades do Sistema

- ✅ Formulário de venda com cálculo automático de total
- ✅ Validação inteligente de cupons (validade, limite, valor mínimo)
- ✅ Upload de comprovantes de pagamento
- ✅ Notificação por email após registro da venda
- ✅ Busca rápida de clientes durante atendimento

Cenário de Uso Típico

"Ana está atendendo uma cliente que possui um cupom de 15% de desconto. Ela acessa o módulo de vendas no MaicoSoft, busca a cliente pelo nome, insere o valor do orçamento (R$ 1.200,00) e digita o código do cupom. O sistema valida automaticamente a validade do cupom, verifica que o valor mínimo de R$ 500 foi atingido, calcula o desconto (R$ 180) e exibe o valor final (R$ 1.020). Após confirmar o pagamento e anexar o comprovante, o sistema incrementa o contador de uso do cupom e envia um email de confirmação para Ana e para a cliente."

Impacto no Sistema MaicoSoft

As necessidades de Ana impulsionaram o desenvolvimento de:

- **VendaService:** Lógica complexa de validação de cupons com múltiplas regras de negócio
- **EmailService:** Notificações automáticas pós-venda com template HTML profissional
- **File Upload:** Sistema de upload e armazenamento de comprovantes em diretório seguro
- **Cálculos Automatizados:** Backend processa valorBruto - valorDesconto = valorTotal em transação ACID

7.8 Análise Comparativa das Personas

Aspecto	Carlos (Admin)	Maria (Diretor)	João (Funcionário)	Ana (Vendedor)
Foco Principal	Gestão técnica	Análise estratégica	Operação cadastral	Execução de vendas
Nível Técnico	Avançado	Intermediário	Básico	Básico
Funcionalidade Crítica	Gestão de usuários	Dashboard e relatórios	Cadastro de clientes	Registro de vendas
Frequência de Uso	Diária (manutenção)	Diária (análise)	Diária (operação)	Diária (operação)
Prioridade UX	Eficiência	Visibilidade	Simplicidade	Agilidade

7.7.3 Impacto das Personas no Desenvolvimento

A definição das personas orientou decisões importantes no projeto MaicoSoft:

- **Interface Intuitiva:** Considerando que João e Ana têm nível técnico básico, a interface foi desenvolvida com foco em simplicidade e feedback visual claro.
- **Dashboard Analítico:** As necessidades de Maria por visão estratégica justificaram o desenvolvimento de um dashboard com KPIs visuais e gráficos em tempo real.
- **Automações Inteligentes:** A dor de João com digitação manual de endereços levou à integração com ViaCEP. A frustração de Ana com cálculos manuais resultou na validação automática de cupons.
- **Segurança e Auditoria:** As preocupações de Carlos com rastreabilidade motivaram a implementação de logs detalhados e controle de acesso baseado em roles.
- **Sistema de Recuperação de Senha:** A sobrecarga de Carlos com resets manuais justificou o desenvolvimento do sistema automatizado com tokens temporários via email.

7.9 Validação das Personas

As personas foram validadas através de:

- **Mapeamento de casos de uso (UC01, UC02, UC03):** Cada persona foi associada a casos de uso específicos documentados neste projeto, garantindo que suas necessidades estejam refletidas nos fluxos funcionais do sistema.
- **Alinhamento com requisitos funcionais (RF01-RF15):** As necessidades das personas foram cruzadas com os requisitos funcionais para verificar cobertura completa e evitar lacunas no desenvolvimento.
- **Análise de fluxos reais de trabalho em empresas de varejo:** Os cenários de uso típico foram baseados em processos comuns do setor comercial, garantindo realismo e aplicabilidade prática.
- **Verificação de consistência com as roles do sistema (ADMIN, DIRETOR, FUNCIONARIO, VENDEDOR):** Cada persona foi mapeada para uma role específica do sistema, assegurando que as permissões de acesso e funcionalidades disponíveis correspondem às responsabilidades profissionais de cada perfil.

> **Resultado da Validação:** As quatro personas cobrem todos os níveis de acesso do sistema e representam os principais usuários-alvo do MaicoSoft. A diversidade de perfis (técnico avançado a básico) garantiu que o design da interface e a arquitetura do sistema atendam a diferentes necessidades de usabilidade.

8. BIBLIOGRAFIA E REFERÊNCIAS

8.1 Tecnologias Core

Java e Spring Framework

- **Java Development Kit (JDK):**
  https://www.oracle.com/java/
- **Spring Boot Documentation:**
  https://spring.io/projects/spring-boot
- **Spring Data JPA:**
  https://spring.io/projects/spring-data-jpa
- **Spring Boot Mail:**
  https://docs.spring.io/spring-boot/docs/current/reference/html/io.html#io.email

Banco de Dados

- **PostgreSQL Official Documentation:**
  https://www.postgresql.org/docs/
- **Flyway Database Migrations:**
  https://flywaydb.org/documentation
- **H2 Database Engine:**
  https://www.h2database.com/html/main.html

Containerização

- **Docker Official Documentation:**
  https://docs.docker.com/
- **Docker Compose Documentation:**
  https://docs.docker.com/compose/

8.2 Frontend e Servidor Web

Tecnologias Web

- **HTML5 Specification:**
  https://html.spec.whatwg.org/
- **CSS3 Documentation:**
  https://www.w3.org/Style/CSS/
- **JavaScript MDN Web Docs:**
  https://developer.mozilla.org/en-US/docs/Web/JavaScript
- **Nginx Official Documentation:**
  https://nginx.org/en/docs/
- **Chart.js Documentation:**
  https://www.chartjs.org/docs/latest/

8.3 Dependências e Ferramentas

Build e Gerenciamento

- **Apache Maven:**
  https://maven.apache.org/guides/index.html
- **Git Version Control:**
  https://git-scm.com/doc
- **GitHub:**
  https://docs.github.com/

8.4 Padrões e Metodologias

- **RESTful API Design:**
  https://restfulapi.net/
- **MVC Pattern:**
  https://www.oracle.com/technical-resources/articles/javase/mvc.html
- **Repository Pattern:**
  https://martinfowler.com/eaaCatalog/repository.html
- **UML Diagrams Guide:**
  https://www.uml-diagrams.org/
- **Personas in UX Design:**
  https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them

8.5 DevOps e Infraestrutura

- **Windows Subsystem for Linux (WSL):**
  https://learn.microsoft.com/en-us/windows/wsl/

8.6 Bibliotecas Java

- **Lombok Project:**
  https://projectlombok.org/
- **Thymeleaf Template Engine:**
  https://www.thymeleaf.org/documentation.html
- **SpringDoc OpenAPI (Swagger):**
  https://springdoc.org/
- **Caffeine Cache:**
  https://github.com/ben-manes/caffeine

8.4 Padrões e Metodologias

- **RESTful API Design:**
  https://restfulapi.net/
- **MVC Pattern:**
  https://www.oracle.com/technical-resources/articles/javase/mvc.html
- **Repository Pattern:**
  https://martinfowler.com/eaaCatalog/repository.html
- **UML Diagrams Guide:**
  https://www.uml-diagrams.org/
- **Personas in UX Design:**
  https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them

8.5 Segurança e Boas Práticas

- **OWASP Top 10:**
  https://owasp.org/www-project-top-ten/
- **BCrypt Password Hashing:**
  https://en.wikipedia.org/wiki/Bcrypt
- **CORS (Cross-Origin Resource Sharing):**
  https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

8.6 Repositório do Projeto

- **MaicoSoft GitHub Repository:**
  https://github.com/nisze/MaicoSoft