Especificação Técnica e Funcional do Sistema Simple

1. Visão Geral do Sistema

O **Simple** é um sistema de gestão de pedidos de serviços municipais desenvolvido em Java com Spring Boot 3.2.0. O sistema permite o gerenciamento completo do ciclo de vida de solicitações de serviços municipais, desde o cadastro inicial até a conclusão, com funcionalidades específicas para diferentes perfis de usuários:

- **Cidadãos**: Podem solicitar serviços e acompanhar o status de suas solicitações
- **Atendentes**: Responsáveis pelo cadastro e triagem inicial dos pedidos
- **Gestores**: Monitoram o fluxo de pedidos e geram relatórios
- **Técnicos**: Executam os serviços solicitados
- **Fiscais**: Verificam a qualidade dos serviços executados
- **Administradores**: Gerenciam configurações e usuários do sistema

O sistema foi projetado para ser robusto, seguro e escalável, seguindo as melhores práticas de desenvolvimento com Spring Boot e padrões de arquitetura modernos.

2. Arquitetura Detalhada

2.1 Visão Geral da Arquitetura

O Simple segue uma arquitetura em camadas bem definida, baseada no padrão MVC (Model-View-Controller) adaptado para APIs REST. Conforme ilustrado no Diagrama de Arquitetura, o sistema está organizado nas seguintes camadas:

- **Camada de Apresentação (Controllers)**: Recebe as requisições HTTP, valida os dados de entrada e delega o processamento para os serviços.
- **Camada de Negócio (Services)**: Implementa a lógica de negócio da aplicação, orquestrando as operações entre as diferentes entidades.
- **Camada de Persistência (Repositories)**: Responsável pela persistência e recuperação de dados no banco de dados.
- **Camada de Domínio (Entities e DTOs)**: Representa as entidades do banco de dados e os objetos de transferência de dados.
- **Camada de Segurança**: Implementa autenticação baseada em JWT e autorização baseada em roles.

2.2 Componentes do Sistema

Conforme ilustrado no Diagrama de Componentes, o sistema está organizado nos seguintes módulos funcionais:

1. **Módulo de Autenticação e Autorização**:
    - Responsável pelo controle de acesso ao sistema
    - Implementa autenticação baseada em JWT

- Gerencia usuários e perfis
- Controla permissões baseadas em roles

2. **Módulo de Gestão de Pedidos**:
   - Núcleo do sistema
   - Gerencia o ciclo de vida dos pedidos de serviços
   - Implementa fluxos de trabalho para processamento de pedidos
   - Controla status e etapas dos pedidos
3. **Módulo de Gestão de Cidadãos**:
   - Gerencia o cadastro de cidadãos
   - Associa cidadãos a pedidos
   - Mantém histórico de pedidos por cidadão
4. **Módulo de Configuração do Sistema**:
   - Gerencia tipos de serviços e categorias
   - Configura parâmetros do sistema
   - Gerencia perfis de acesso
5. **Módulo de Favoritos**:
   - Permite aos usuários marcar serviços como favoritos
   - Facilita o acesso rápido a serviços frequentemente utilizados
6. **Infraestrutura**:
   - Componentes de suporte como tratamento de exceções
   - Configuração de CORS
   - Documentação da API com Swagger

2.3 Fluxo de Dados

O Diagrama de Fluxo de Dados ilustra como os dados fluem através do sistema para os principais casos de uso:

1. **Fluxo de Autenticação**:
   - O usuário envia credenciais (email e senha)
   - O sistema valida as credenciais e gera um token JWT
   - O token é utilizado em requisições subsequentes para autorização
2. **Fluxo de Criação de Pedido**:
   - O usuário autenticado envia dados do pedido
   - O sistema valida os dados e cria um novo pedido
   - Um código de acompanhamento é gerado automaticamente
   - O pedido é armazenado no banco de dados
3. **Fluxo de Atualização de Status**:
   - O usuário autorizado solicita a atualização do status de um pedido
   - O sistema valida a permissão e atualiza o status
   - Se o status for "CONCLUIDO", a data de conclusão é atualizada

2.4 Modelo de Dados

O Diagrama ER ilustra as principais entidades do sistema e seus relacionamentos:

1. **Usuario**: Representa os usuários do sistema com diferentes perfis

2. **Perfil**: Representa os perfis de acesso dos usuários
3. **Cidadao**: Representa os cidadãos que solicitam serviços
4. **Pedido**: Representa as solicitações de serviços
5. **StatusPedido**: Representa os diferentes estados de um pedido
6. **EtapaProcesso**: Representa as etapas do fluxo de processamento de um pedido
7. **TipoServico**: Categoriza os diferentes tipos de serviços oferecidos
8. **CategoriaServico**: Agrupa tipos de serviços relacionados
9. **Favorito**: Permite aos usuários marcar serviços como favoritos

3. Tecnologias Utilizadas

3.1 Backend

- **Linguagem de Programação**: Java 17
- **Framework**: Spring Boot 3.2.0
- **Gerenciador de Dependências**: Maven
- **Persistência**: Spring Data JPA / Hibernate
- **Segurança**: Spring Security com JWT (JSON Web Token)
- **Documentação da API**: SpringDoc OpenAPI (Swagger)
- **Validação**: Spring Validation (Bean Validation)
- **Utilitários**: Lombok, Hypersistence Utils

3.2 Banco de Dados

- **SGBD**: PostgreSQL
- **Migrations**: Flyway (opcional, não identificado explicitamente no código)

3.3 Infraestrutura

- **Containerização**: Docker
- **Servidor de Aplicação**: Tomcat (embutido no Spring Boot)

3.4 Ferramentas de Desenvolvimento

- **IDE**: Não especificada, mas compatível com qualquer IDE Java moderna (IntelliJ IDEA, Eclipse, VS Code)
- **Controle de Versão**: Git (inferido pela estrutura do projeto)
- **Automação de Build**: Scripts shell (build.sh, run.sh)

4. Requisitos Funcionais

4.1 Módulo de Autenticação e Autorização

- **RF-01**: O sistema deve permitir o registro de novos usuários com informações básicas (nome, email, senha)
- **RF-02**: O sistema deve autenticar usuários através de email e senha
- **RF-03**: O sistema deve gerar tokens JWT para usuários autenticados

- **RF-04**: O sistema deve validar tokens JWT em requisições protegidas
- **RF-05**: O sistema deve controlar o acesso a recursos baseado em perfis de usuário
- **RF-06**: O sistema deve permitir a atualização de informações de usuário
- **RF-07**: O sistema deve registrar o último acesso do usuário

4.2 Módulo de Gestão de Pedidos

- **RF-08**: O sistema deve permitir a criação de novos pedidos de serviço
- **RF-09**: O sistema deve gerar automaticamente um código de acompanhamento para cada pedido
- **RF-10**: O sistema deve permitir a consulta de pedidos por diferentes critérios (ID, código, cidadão, usuário)
- **RF-11**: O sistema deve permitir a atualização do status de um pedido
- **RF-12**: O sistema deve calcular automaticamente a data prevista de conclusão baseada no tipo de serviço
- **RF-13**: O sistema deve registrar a data de conclusão quando o pedido for finalizado
- **RF-14**: O sistema deve permitir a atribuição de prioridade a pedidos
- **RF-15**: O sistema deve permitir a adição de observações a pedidos

4.3 Módulo de Gestão de Cidadãos

- **RF-16**: O sistema deve permitir o cadastro de cidadãos com informações pessoais
- **RF-17**: O sistema deve permitir a consulta de cidadãos por diferentes critérios
- **RF-18**: O sistema deve permitir a atualização de informações de cidadãos
- **RF-19**: O sistema deve associar cidadãos a pedidos
- **RF-20**: O sistema deve permitir a consulta de pedidos por cidadão

4.4 Módulo de Configuração do Sistema

- **RF-21**: O sistema deve permitir o cadastro de tipos de serviços
- **RF-22**: O sistema deve permitir o cadastro de categorias de serviços
- **RF-23**: O sistema deve permitir a associação de tipos de serviços a categorias
- **RF-24**: O sistema deve permitir a ativação/desativação de tipos de serviços
- **RF-25**: O sistema deve permitir a configuração de prazos estimados para tipos de serviços
- **RF-26**: O sistema deve permitir a configuração de valores para tipos de serviços

4.5 Módulo de Favoritos

- **RF-27**: O sistema deve permitir que usuários marquem tipos de serviços como favoritos

- **RF-28**: O sistema deve permitir a consulta de favoritos por usuário
- **RF-29**: O sistema deve permitir a remoção de favoritos

5. Requisitos Não Funcionais

5.1 Segurança

- **RNF-01**: O sistema deve armazenar senhas de forma segura utilizando algoritmo BCrypt
- **RNF-02**: O sistema deve implementar autenticação stateless baseada em JWT
- **RNF-03**: O sistema deve implementar controle de acesso baseado em roles
- **RNF-04**: O sistema deve implementar proteção contra ataques CSRF
- **RNF-05**: O sistema deve implementar configuração de CORS para permitir requisições de origens específicas
- **RNF-06**: O sistema deve implementar validação de dados de entrada para prevenir injeção de dados maliciosos

5.2 Desempenho

- **RNF-07**: O sistema deve responder a requisições em tempo aceitável (< 1 segundo para operações simples)
- **RNF-08**: O sistema deve suportar múltiplos usuários simultâneos
- **RNF-09**: O sistema deve implementar paginação para consultas que retornam grandes volumes de dados

5.3 Disponibilidade

- **RNF-10**: O sistema deve estar disponível 24/7, com tempo de inatividade planejado mínimo
- **RNF-11**: O sistema deve ser resiliente a falhas, recuperando-se automaticamente quando possível

5.4 Escalabilidade

- **RNF-12**: O sistema deve ser projetado para escalar horizontalmente
- **RNF-13**: O sistema deve ser containerizado para facilitar a implantação em ambientes de nuvem

5.5 Manutenibilidade

- **RNF-14**: O sistema deve seguir padrões de código e arquitetura que facilitem a manutenção
- **RNF-15**: O sistema deve ser modular, com componentes bem definidos e baixo acoplamento
- **RNF-16**: O sistema deve implementar tratamento de exceções consistente

5.6 Usabilidade

- **RNF-17**: O sistema deve fornecer mensagens de erro claras e informativas
- **RNF-18**: O sistema deve fornecer documentação da API através do Swagger
- **RNF-19**: O sistema deve implementar validação de dados com mensagens de erro específicas

6. Descrição Detalhada das APIs

6.1 API de Autenticação

6.1.1 POST /auth/login **Descrição**: Autentica um usuário e retorna um token JWT.

Requisição:

```
{
  "email": "usuario@exemplo.com",
  "senha": "senha123"
}
```

Resposta (200 OK):

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Resposta (401 Unauthorized):

```
{
  "timestamp": "2023-01-01T12:00:00",
  "status": 401,
  "error": "Unauthorized",
  "message": "Credenciais inválidas",
  "path": "/auth/login"
}
```

6.1.2 POST /auth/register **Descrição**: Registra um novo usuário no sistema.

Requisição:

```
{
  "nome": "João Silva",
  "email": "joao.silva@exemplo.com",
  "senha": "senha123",
  "perfilId": 2
}
```

Resposta (201 Created):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "nome": "João Silva",
  "email": "joao.silva@exemplo.com",
  "perfil": "ATENDENTE",
  "ativo": true,
  "criadoEm": "2023-01-01T12:00:00"
}
```


6.2 API de Pedidos

6.2.1 GET /pedidos **Descrição**: Retorna todos os pedidos (paginados).

Parâmetros de Consulta: - `page`: Número da página (default: 0) - `size`: Tamanho da página (default: 20) - `sort`: Campo para ordenação (default: criadoEm,desc)

Resposta (200 OK):


```
{
  "content": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "codigoAcompanhamento": "PED123456",
      "tipoServico": "Reparo de Iluminação Pública",
      "cidadao": "João Silva",
      "usuarioCriacao": "Maria Atendente",
      "usuarioResponsavel": "Pedro Técnico",
      "etapaAtual": "Análise Técnica",
      "status": "EM_ANDAMENTO",
      "dataInicio": "2023-01-01T10:00:00",
      "dataPrevisao": "2023-01-05T10:00:00",
      "dataConclusao": null,
      "observacoes": "Poste com lâmpada queimada",
      "valorTotal": 150.00,
      "origem": "PRESENCIAL",
      "prioridade": 2,
      "criadoEm": "2023-01-01T10:00:00"
    }
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 20,
    "sort": {
      "sorted": true,
      "unsorted": false,
```

```
      "empty": false
    },
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "totalElements": 1,
  "totalPages": 1,
  "last": true,
  "size": 20,
  "number": 0,
  "sort": {
    "sorted": true,
    "unsorted": false,
    "empty": false
  },
  "numberOfElements": 1,
  "first": true,
  "empty": false
}
```


6.2.2 GET /pedidos/{id} **Descrição**: Retorna um pedido específico pelo ID.

Parâmetros de Path: - `id`: ID do pedido (UUID)

Resposta (200 OK):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "codigoAcompanhamento": "PED123456",
  "tipoServico": "Reparo de Iluminação Pública",
  "cidadao": "João Silva",
  "usuarioCriacao": "Maria Atendente",
  "usuarioResponsavel": "Pedro Técnico",
  "etapaAtual": "Análise Técnica",
  "status": "EM_ANDAMENTO",
  "dataInicio": "2023-01-01T10:00:00",
  "dataPrevisao": "2023-01-05T10:00:00",
  "dataConclusao": null,
  "observacoes": "Poste com lâmpada queimada",
  "valorTotal": 150.00,
  "origem": "PRESENCIAL",
  "prioridade": 2,
  "criadoEm": "2023-01-01T10:00:00"
}
```

6.2.3 GET /pedidos/codigo/{codigo} **Descrição**: Retorna um pedido pelo código de acompanhamento (endpoint público).

Parâmetros de Path: - `codigo`: Código de acompanhamento do pedido

Resposta (200 OK):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "codigoAcompanhamento": "PED123456",
  "tipoServico": "Reparo de Iluminação Pública",
  "status": "EM_ANDAMENTO",
  "dataInicio": "2023-01-01T10:00:00",
  "dataPrevisao": "2023-01-05T10:00:00",
  "dataConclusao": null
}
```

6.2.4 POST /pedidos **Descrição**: Cria um novo pedido.

Requisição:

```
{
  "cidadaoId": "550e8400-e29b-41d4-a716-446655440000",
  "tipoServicoId": 1,
  "observacoes": "Poste com lâmpada queimada",
  "origem": "PRESENCIAL",
  "prioridade": 2
}
```

Resposta (201 Created):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "codigoAcompanhamento": "PED123456",
  "tipoServico": "Reparo de Iluminação Pública",
  "cidadao": "João Silva",
  "usuarioCriacao": "Maria Atendente",
  "usuarioResponsavel": "Maria Atendente",
  "etapaAtual": "Registro",
  "status": "NOVO",
  "dataInicio": "2023-01-01T10:00:00",
  "dataPrevisao": "2023-01-05T10:00:00",
  "dataConclusao": null,
  "observacoes": "Poste com lâmpada queimada",
  "valorTotal": 150.00,
  "origem": "PRESENCIAL",
  "prioridade": 2,
  "criadoEm": "2023-01-01T10:00:00"
}
```

6.2.5 PATCH /pedidos/{id}/status/{statusId} **Descrição**: Atualiza o status de um pedido.

Parâmetros de Path: - `id`: ID do pedido (UUID) - `statusId`: ID do novo status

Resposta (200 OK):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "codigoAcompanhamento": "PED123456",
  "tipoServico": "Reparo de Iluminação Pública",
  "cidadao": "João Silva",
  "usuarioCriacao": "Maria Atendente",
  "usuarioResponsavel": "Pedro Técnico",
  "etapaAtual": "Execução",
  "status": "EM_ANDAMENTO",
  "dataInicio": "2023-01-01T10:00:00",
  "dataPrevisao": "2023-01-05T10:00:00",
  "dataConclusao": null,
  "observacoes": "Poste com lâmpada queimada",
  "valorTotal": 150.00,
  "origem": "PRESENCIAL",
  "prioridade": 2,
  "criadoEm": "2023-01-01T10:00:00"
}
```


6.3 API de Cidadãos

6.3.1 GET /cidadaos **Descrição**: Retorna todos os cidadãos (paginados).

Parâmetros de Consulta: - `page`: Número da página (default: 0) - `size`: Tamanho da página (default: 20) - `sort`: Campo para ordenação (default: nome,asc)

Resposta (200 OK):


```
{
  "content": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "nome": "João Silva",
      "cpf": "123.456.789-00",
      "email": "joao.silva@exemplo.com",
      "telefone": "(11) 98765-4321",
      "endereco": "Rua das Flores, 123",
      "bairro": "Centro",
      "cidade": "São Paulo",
      "estado": "SP",
```

```
      "cep": "01234-567",
      "criadoEm": "2023-01-01T10:00:00"
    }
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 20,
    "sort": {
      "sorted": true,
      "unsorted": false,
      "empty": false
    },
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "totalElements": 1,
  "totalPages": 1,
  "last": true,
  "size": 20,
  "number": 0,
  "sort": {
    "sorted": true,
    "unsorted": false,
    "empty": false
  },
  "numberOfElements": 1,
  "first": true,
  "empty": false
}
```


6.3.2 POST /cidadaos **Descrição**: Cria um novo cidadão.

Requisição:


```
{
  "nome": "João Silva",
  "cpf": "123.456.789-00",
  "email": "joao.silva@exemplo.com",
  "telefone": "(11) 98765-4321",
  "endereco": "Rua das Flores, 123",
  "bairro": "Centro",
  "cidade": "São Paulo",
  "estado": "SP",
  "cep": "01234-567"
}
```

Resposta (201 Created):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "nome": "João Silva",
  "cpf": "123.456.789-00",
  "email": "joao.silva@exemplo.com",
  "telefone": "(11) 98765-4321",
  "endereco": "Rua das Flores, 123",
  "bairro": "Centro",
  "cidade": "São Paulo",
  "estado": "SP",
  "cep": "01234-567",
  "criadoEm": "2023-01-01T10:00:00"
}
```


6.4 API de Tipos de Serviço

6.4.1 GET /tipos-servicos

Descrição: Retorna todos os tipos de serviço.

Resposta (200 OK):


```
[
  {
    "id": 1,
    "nome": "Reparo de Iluminação Pública",
    "descricao": "Serviço para reparo de postes e lâmpadas de iluminação pública",
    "categoria": "Infraestrutura",
    "prazoEstimado": 5,
    "valorBase": 150.00,
    "ativo": true,
    "criadoEm": "2023-01-01T10:00:00"
  }
]
```


6.4.2 POST /tipos-servicos

Descrição: Cria um novo tipo de serviço.

Requisição:


```
{
  "nome": "Reparo de Iluminação Pública",
  "descricao": "Serviço para reparo de postes e lâmpadas de iluminação pública",
  "categoriaId": 1,
  "prazoEstimado": 5,
  "valorBase": 150.00
}
```


Resposta (201 Created):

```
{
  "id": 1,
  "nome": "Reparo de Iluminação Pública",
  "descricao": "Serviço para reparo de postes e lâmpadas de iluminação pública",
  "categoria": "Infraestrutura",
  "prazoEstimado": 5,
  "valorBase": 150.00,
  "ativo": true,
  "criadoEm": "2023-01-01T10:00:00"
}
```


6.5 API de Favoritos

6.5.1 GET /favoritos **Descrição**: Retorna os favoritos do usuário logado.

Resposta (200 OK):


```
[
  {
    "id": 1,
    "tipoServico": {
      "id": 1,
      "nome": "Reparo de Iluminação Pública",
      "descricao": "Serviço para reparo de postes e lâmpadas de iluminação pública",
      "categoria": "Infraestrutura",
      "prazoEstimado": 5,
      "valorBase": 150.00,
      "ativo": true
    },
    "criadoEm": "2023-01-01T10:00:00"
  }
]
```


6.5.2 POST /favoritos **Descrição**: Adiciona um tipo de serviço aos favoritos.

Requisição:


```
{
  "tipoServicoId": 1
}
```


Resposta (201 Created):


```
{
  "id": 1,
  "tipoServico": {
    "id": 1,
    "nome": "Reparo de Iluminação Pública",
    "descricao": "Serviço para reparo de postes e lâmpadas de iluminação pública",
```

```
    "categoria": "Infraestrutura",
    "prazoEstimado": 5,
    "valorBase": 150.00,
    "ativo": true
  },
  "criadoEm": "2023-01-01T10:00:00"
}
```


7. Modelo de Dados

7.1 Entidades Principais

7.1.1 Usuario

Campo	Tipo	Descrição
id	UUID	Identificador único do usuário
nome	String	Nome completo do usuário
email	String	Email do usuário (único)
senha	String	Senha do usuário (criptografada)
perfil	Perfil	Perfil de acesso do usuário
ativo	Boolean	Indica se o usuário está ativo
ultimoAcesso	LocalDateTime	Data e hora do último acesso
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.2 Perfil

Campo	Tipo	Descrição
id	Long	Identificador único do perfil
nome	String	Nome do perfil
codigo	String	Código do perfil (ROLE_XXX)
descricao	String	Descrição do perfil
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.3 Cidadao

Campo	Tipo	Descrição
id	UUID	Identificador único do cidadão
nome	String	Nome completo do cidadão
cpf	String	CPF do cidadão (único)
email	String	Email do cidadão
telefone	String	Telefone do cidadão

Campo	Tipo	Descrição
endereco	String	Endereço do cidadão
bairro	String	Bairro
cidade	String	Cidade
estado	String	Estado (UF)
cep	String	CEP
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.4 Pedido

Campo	Tipo	Descrição
id	UUID	Identificador único do pedido
codigoAcompanhamento	String	Código para acompanhamento público
tipoServico	TipoServico	Tipo de serviço solicitado
cidadao	Cidadao	Cidadão solicitante
usuarioCriacao	Usuario	Usuário que criou o pedido
usuarioResponsavel	Usuario	Usuário responsável pelo pedido
etapaAtual	EtapaProcesso	Etapa atual do processo
status	StatusPedido	Status atual do pedido
dataInicio	LocalDateTime	Data e hora de início
dataPrevisao	LocalDateTime	Data e hora prevista para conclusão
dataConclusao	LocalDateTime	Data e hora de conclusão
observacoes	String	Observações sobre o pedido
valorTotal	BigDecimal	Valor total do serviço
origem	String	Origem do pedido (PRESENCIAL, ONLINE, TELEFONE)
prioridade	Integer	Nível de prioridade (1-5)
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.5 StatusPedido

Campo	Tipo	Descrição
id	Long	Identificador único do status
nome	String	Nome do status
codigo	String	Código do status
descricao	String	Descrição do status
cor	String	Cor associada ao status (para UI)
ordem	Integer	Ordem de exibição
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.6 TipoServico

Campo	Tipo	Descrição
id	Long	Identificador único do tipo de serviço
nome	String	Nome do tipo de serviço
descricao	String	Descrição do tipo de serviço
categoria	CategoriaServico	Categoria do serviço
prazoEstimado	Integer	Prazo estimado em dias
valorBase	BigDecimal	Valor base do serviço
ativo	Boolean	Indica se o tipo de serviço está ativo
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.7 CategoriaServico

Campo	Tipo	Descrição
id	Long	Identificador único da categoria
nome	String	Nome da categoria
descricao	String	Descrição da categoria
icone	String	Ícone associado à categoria
cor	String	Cor associada à categoria
ordem	Integer	Ordem de exibição
criadoEm	LocalDateTime	Data e hora de criação
atualizadoEm	LocalDateTime	Data e hora da última atualização

7.1.8 Favorito

Campo	Tipo	Descrição
id	Long	Identificador único do favorito
usuario	Usuario	Usuário que favoritou
tipoServico	TipoServico	Tipo de serviço favoritado
criadoEm	LocalDateTime	Data e hora de criação

7.2 Relacionamentos

- **Usuario - Perfil**: Muitos para um (N:1)
- **Pedido - Cidadao**: Muitos para um (N:1)
- **Pedido - TipoServico**: Muitos para um (N:1)
- **Pedido - StatusPedido**: Muitos para um (N:1)
- **Pedido - Usuario (criação)**: Muitos para um (N:1)
- **Pedido - Usuario (responsável)**: Muitos para um (N:1)
- **TipoServico - CategoriaServico**: Muitos para um (N:1)

- **Favorito - Usuario**: Muitos para um (N:1)
- **Favorito - TipoServico**: Muitos para um (N:1)

8. Mecanismos de Segurança

8.1 Autenticação

O sistema utiliza autenticação baseada em JWT (JSON Web Token):

1. **Processo de Login**:
   - O usuário envia credenciais (email e senha)
   - O sistema valida as credenciais usando Spring Security
   - Se válidas, gera um token JWT com informações do usuário e permissões
   - O token é retornado ao cliente para uso em requisições subsequentes
2. **Validação de Token**:
   - Cada requisição a endpoints protegidos deve incluir o token JWT no header Authorization
   - O filtro `JwtAuthenticationFilter` intercepta a requisição e valida o token
   - Se válido, configura o contexto de segurança do Spring com as informações do usuário
   - Se inválido, a requisição é rejeitada com status 401 (Unauthorized)
3. **Refresh Token**:
   - O sistema implementa refresh tokens para renovação da sessão sem necessidade de reautenticação
   - Quando o token principal expira, o cliente pode usar o refresh token para obter um novo token

8.2 Autorização

O sistema implementa controle de acesso baseado em roles (RBAC):

1. **Perfis de Acesso**:
   - ADMINISTRADOR: Acesso total ao sistema
   - GESTOR: Acesso a relatórios e monitoramento
   - ATENDENTE: Cadastro e atendimento inicial
   - TECNICO: Execução de serviços
   - FISCAL: Verificação de qualidade
2. **Controle de Acesso a Endpoints**:
   - Nível de classe: Anotações `@PreAuthorize` nos controladores
   - Nível de método: Anotações `@PreAuthorize` em métodos específicos
   - Exemplo: `@PreAuthorize("hasAnyRole('ADMINISTRADOR', 'GESTOR')")`
3. **Endpoints Públicos**:
   - `/auth/**`: Endpoints de autenticação
   - `/v3/api-docs/**`, `/swagger-ui/**`: Documentação da API

- `/pedidos/codigo/**`: Consulta pública de pedidos por código
- `/configuracoes`: Configurações públicas do sistema

8.3 Proteção de Dados

1. **Armazenamento Seguro de Senhas**:
   - Utilização do algoritmo BCrypt para hash de senhas
   - Configurado através do `PasswordEncoder` do Spring Security
2. **Proteção contra CSRF**:
   - Desabilitado para APIs REST stateless, pois o JWT já fornece proteção
3. **Configuração de CORS**:
   - Implementação de filtro CORS para permitir requisições de origens específicas
   - Configuração de headers, métodos e credenciais permitidos
4. **Validação de Dados**:
   - Validação de dados de entrada usando Bean Validation
   - Tratamento de exceções para validação de dados

8.4 Auditoria

1. **Registro de Acessos**:
   - Registro da data e hora do último acesso de cada usuário
   - Atualizado automaticamente durante o processo de login
2. **Registro de Alterações**:
   - Campos `criadoEm` e `atualizadoEm` em todas as entidades
   - Atualizados automaticamente através de `@CreatedDate` e `@LastModifiedDate`

9. Configuração e Implantação

9.1 Requisitos de Sistema

- Java 17 ou superior
- PostgreSQL 12 ou superior
- Docker (opcional, para containerização)

9.2 Configuração do Ambiente

9.2.1 Configuração do Banco de Dados O sistema utiliza PostgreSQL como banco de dados. A configuração é feita no arquivo `application.yml`:

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/simple
    username: postgres
    password: postgres
    driver-class-name: org.postgresql.Driver
```

```yaml
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
        format_sql: true
    show-sql: true
```

9.2.2 Configuração de Segurança A configuração de segurança JWT é feita no arquivo `application.yml`:

```yaml
application:
  security:
    jwt:
      secret-key: 404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970
      expiration: 86400000 # 1 dia
      refresh-token:
        expiration: 604800000 # 7 dias
```

9.3 Processo de Build

O sistema utiliza Maven para gerenciamento de dependências e build. O processo de build é automatizado através do script `build.sh`:

```bash
#!/bin/bash

# Verificar versão do Java
java_version=$(java -version 2>&1 | awk -F '"' '/version/ {print $2}')
required_version="17"

if [[ $java_version != $required_version* ]]; then
    echo "Erro: Java $required_version é necessário, mas a versão atual é $java_version"
    exit 1
fi

# Compilar o projeto
echo "Compilando o projeto..."
mvn clean package -DskipTests

if [ $? -eq 0 ]; then
    echo "Build concluído com sucesso!"
else
    echo "Erro durante o build."
    exit 1
fi
```

9.4 Implantação

9.4.1 Implantação Tradicional O sistema pode ser executado diretamente como uma aplicação Spring Boot através do script `run.sh`:

```
#!/bin/bash

# Verificar se o JAR existe
if [ ! -f "target/simple-0.0.1-SNAPSHOT.jar" ]; then
    echo "JAR não encontrado. Executando build..."
    ./build.sh
fi

# Executar a aplicação
echo "Iniciando a aplicação..."
java -jar target/simple-0.0.1-SNAPSHOT.jar
```

9.4.2 Implantação com Docker O sistema pode ser containerizado usando Docker. O `Dockerfile` está configurado da seguinte forma:

```
FROM openjdk:17-jdk-slim AS build

WORKDIR /app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

RUN ./mvnw package -DskipTests

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY --from=build /app/target/simple-0.0.1-SNAPSHOT.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Para construir e executar o container:

```
# Construir a imagem
docker build -t simple-app .

# Executar o container
docker run -p 8080:8080 -e SPRING_DATASOURCE_URL=jdbc:postgresql://host.docker.internal:5432
```

10. Integrações com Sistemas Externos

Não foram identificadas integrações diretas com sistemas externos no código analisado. O sistema Simple é autocontido, sem chamadas a APIs externas ou serviços de terceiros.

No entanto, a arquitetura do sistema permite a fácil adição de integrações através de:

10.1 Possíveis Pontos de Integração

1. **Integração com Sistemas de Geolocalização**:
   - Para localização de endereços de serviços
   - Para cálculo de rotas para técnicos
2. **Integração com Sistemas de Notificação**:
   - Para envio de SMS ou emails aos cidadãos
   - Para notificações push em aplicativos móveis
3. **Integração com Sistemas de Pagamento**:
   - Para processamento de pagamentos de serviços
   - Para emissão de boletos ou notas fiscais
4. **Integração com Sistemas de BI**:
   - Para exportação de dados para análise
   - Para geração de dashboards e relatórios avançados

10.2 Implementação de Integrações Futuras

Para implementar integrações futuras, recomenda-se:

1. **Criação de Clientes HTTP**:
   - Utilização de RestTemplate ou WebClient do Spring
   - Implementação de clientes Feign para APIs REST
2. **Implementação de Mensageria**:
   - Utilização de RabbitMQ ou Kafka para comunicação assíncrona
   - Implementação de listeners para processamento de mensagens
3. **Criação de Adaptadores**:
   - Implementação do padrão Adapter para isolar o código de integração
   - Criação de interfaces para permitir múltiplas implementações
4. **Configuração de Circuit Breakers**:
   - Utilização de Resilience4j ou Spring Cloud Circuit Breaker
   - Implementação de fallbacks para garantir resiliência