

# Sistema de gestão de candidaturas para alojamento

Joaquim Segunda - 56396

13 de novembro de 2025

## 1 Introdução

O presente documento descreve a arquitetura e o funcionamento de um **Sistema Distribuído de Gestão de Candidaturas a Alojamento**, projetado para otimizar o processo de alocação e administração de vagas de habitação. O objetivo primário do sistema é centralizar a informação de alojamentos disponíveis e automatizar o ciclo de vida das candidaturas, desde a submissão inicial até à aceitação ou rejeição final.

## 2 Estrutura do projeto

A arquitetura lógica do sistema é decomposta nas seguintes camadas essenciais:

- **Camada de Modelo (model/)**: Contém as estruturas de dados (POJOs) que representam as entidades de negócio: **Alojamento**, **Candidato** e **Candidatura**. Define também os estados possíveis para cada entidade através de **Enums**.
- **Camada de Persistência (repository/)**: Responsável pela comunicação direta com a base de dados. Utiliza JDBC para operações *CRUD* (Create, Read, Update, Delete) no **PostgreSQL**, garantindo a integridade dos dados e as transações.
- **Camada de Serviço (service/)**: Implementa a **lógica de negócio**. Aqui ocorrem as validações de dados (e.g., unicidade do email do candidato, transições válidas de estado de uma candidatura), agindo como intermediário entre o servidor e os repositórios.
- **Camada de Apresentação/Comunicação (server/)**: Contém o **Server principal**, que gera as conexões de rede, injeta as dependências de Serviço nos **ClientHandler** e processa os comandos recebidos.

## 3 Arquitetura de Comunicação Cliente-Servidor

### 3.1 Protocolo de Comunicação

O sistema implementa uma arquitetura cliente-servidor baseada em sockets TCP/IP, utilizando o porto 12345 para comunicação. A comunicação segue um protocolo textual simples com os seguintes elementos:

- **Mensagens do Cliente:** Comandos numéricos (1-4) para usuários normais, strings para administradores.
- **Respostas do Servidor:** Estruturadas como TIPO|CONTEÚDO, onde TIPO pode ser:
  - SUCESSO - Operação bem-sucedida.
  - ERRO - Erro na operação.
  - SAIR - Encerramento de sessão.
- **Delimitador:** Linha vazia indica fim da transmissão.

### 3.2 Diagrama de Sequência - Fluxo Normal

```

Cliente Normal           Servidor
|                         |
|--- CONECTA ----->|
|<-- MENU -----|
|--- "1" ----->|
|<-- "Nome:" -----|
|--- "João Silva" ---->|
|<-- "Email:" -----|
|--- "joao@email.com"-->|
|<-- LINHA VAZIA -----|
|<-- SUCESSO|... -----|

```

### 3.3 Diagrama de Sequência - Admin

```

Cliente Admin           Servidor
|                         |
|--- "ADMIN" ----->|
|<-- MENU_ADMIN -----|
|--- "LISTAR_CANDIDATURAS" ->|
|<-- LISTA -----|
|<-- LINHA VAZIA -----|

```

## 4 Limitações do Cliente Administrativo

O cliente administrativo atual apresenta várias limitações significativas:

### 4.1 1. Autenticação Inexistente

**Problema:** Qualquer cliente pode enviar "ADMIN" e obter acesso.

**Solução:** Implementar sistema de credenciais com username/password.

## **4.2 2. Comandos Complexos para Administradores**

Problema: Administradores precisam memorizar sintaxe complexa:  
REGISTAR\_ALOJAMENTO|Nome|Cidade|Capacidade.

Solução: Interface interativa similar ao usuário normal.

## **4.3 3. Funcionalidades em Falta**

Funcionalidades críticas não implementadas:

- Suspender/reativar candidatos.
- Estatísticas do sistema.
- Gestão de capacidade dos alojamentos.
- Histórico de alterações.

## **4.4 4. Feedback Insuficiente**

Problema: Mensagens de erro genéricas.

Exemplo: "ERRO\_DB|Falha na operação".

Solução: Mensagens específicas e sugestões de correção.

# **5 Conclusão do Projeto**

## **5.1 Sucessos Implementados**

O projeto demonstra competência nas seguintes áreas:

### **5.1.1 Arquitetura de Comunicação**

- **Sockets TCP/IP:** Implementação robusta da comunicação de rede.
- **Protocolo Textual:** Fácil de debuggar e expandir.
- **Concorrência:** Suporte a múltiplos clientes simultâneos.
- **Separação de Responsabilidades:** MVC com Services e Repositories.

### **5.1.2 Funcionalidades de Usuário**

- **Registo Completo:** Fluxo interativo com validações.
- **Consulta de Estado:** Informação detalhada das candidaturas.
- **Listagem:** Alojamentos disponíveis com filtros.
- **Validações:** Email, números, estados válidos.

### 5.1.3 Base de Dados

#### 5.1.4

sectionComandos de Configuração

Os comandos listados abaixo devem ser executados sequencialmente no terminal `psql` (ou ferramenta gráfica), assumindo acesso inicial como super-utilizador (`postgres`).

## 1. Criação de Utilizador e Base de Dados

```
CREATE USER alojamento_admin WITH PASSWORD '12345';
CREATE DATABASE alojamento_db OWNER alojamento_admin;
\c alojamento_db
```

## 2. Criação dos Tipos ENUM Personalizados

```
-- ENUMs para a entidade Candidatura
CREATE TYPE estado_candidatura_type AS ENUM ('SUBMETIDA', 'EM_ANALISE',
                                              'ACEITE', 'REJEITADA', 'FINALIZADA');

-- ENUMs para a entidade Alojamento
CREATE TYPE estado_alojamento_type AS ENUM ('ATIVO', 'EM_OBRAS',
                                              'FECHADO');

-- ENUMs para a entidade Candidato
CREATE TYPE estado_candidato_type AS ENUM ('ATIVO', 'SUSPENSO',
                                              'REMOVIDO');
CREATE TYPE sexo_type AS ENUM ('MASCULINO', 'FEMININO', 'OUTRO');
```

## 3. Criação das Tabelas

```
-- Tabela CANDIDATO
CREATE TABLE candidato (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    telefone VARCHAR(50),
    sexo sexo_type NOT NULL,
    curso VARCHAR(255),
    data_registro DATE NOT NULL,
    estado estado_candidato_type NOT NULL
);

-- Tabela ALOJAMENTO
CREATE TABLE alojamento (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    cidade VARCHAR(255) NOT NULL,
    capacidade INTEGER NOT NULL CHECK (capacidade > 0),
    active BOOLEAN NOT NULL,
    estado estado_alojamento_type NOT NULL
);

-- Tabela CANDIDATURA
```

```

CREATE TABLE candidatura (
    id SERIAL PRIMARY KEY,
    alojamento_id INTEGER NOT NULL REFERENCES alojamento(id),
    candidato_id INTEGER NOT NULL REFERENCES candidato(id),
    data_candidatura DATE NOT NULL,
    estado.estado_candidatura_type NOT NULL,
    UNIQUE (alojamento_id, candidato_id)
);

```

- **Enums:** Estados consistentes entre Java e PostgreSQL.
- **Transações:** Operações atómicas quando necessário.
- **Prepared Statements:** Prevenção de SQL injection.

## 5.2 Lições Aprendidas

### 5.2.1 Desafios Técnicos Superados

1. Sincronização Cliente-Servidor
  - Problema: Buffer com dados residuais
  - Solução: Limpeza de buffer e delimitadores claros
2. Gestão de Estados
  - Problema: Transições inválidas de estado
  - Solução: Validações no service layer
3. Concorrência
  - Problema: Race conditions na base de dados
  - Solução: Verificações de existência antes de inserções

## 5.3 Recomendações para Melhoria

### 5.3.1 Prioritárias

1. Sistema de Autenticação: JWT ou sessões para administradores.
2. Interface Admin Interativa: Menu numérico similar ao usuário.
3. Logging: Sistema de logs para auditoria.
4. Validações Avançadas: Capacidade disponível, conflitos de datas.

### 5.3.2 Secundárias

1. API REST: Para integração com frontend web.
2. Base de Dados Connection Pool: Melhor performance.
3. Cache: Para dados frequentemente acedidos.
4. Documentação API: OpenAPI/Swagger.

## 5.4 Avaliação Final

O projeto representa uma **implementação sólida** dos conceitos fundamentais de sistemas distribuídos, demonstrando:

- Compreensão profunda de comunicação por sockets.
- Arquitetura em camadas bem estruturada.
- Gestão adequada de erros e exceções.
- Design extensível e manutenível.

A principal área de melhoria reside no **cliente administrativo**, que necessita de uma reformulação completa para se tornar verdadeiramente utilizável em ambiente de produção.

**Conclusão:** Projeto academicamente bem-sucedido, com base técnica sólida para expansão futura.