Sistemas Distribuídos

Licenciatura em Engenharia Informática

1º Trabalho

Sistema Distribuído de Gestão

André Gonçalves, 58392 | André Zhan, 58762

Departamento de Informática
Universidade de Évora
12 de Novembro de 2025

Conteúdo

1 Introdução

Como parte do trabalho prático de Sistemas Distribuídos, os alunos devem escolher e modelar um domínio de aplicação real e relevante, aplicando conceitos de comunicação remota e arquitetura cliente-servidor. Para este projeto foi desenvolvido um sistema distribuído para gestão de uma Biblioteca Digital, permitindo o registo e aprovação de utilizadores, catálogo de livros, empréstimos e histórico de operações. Este domínio foi escolhido por ser simples de compreender, mas suficientemente rico para aplicar e demonstrar os conceitos aprendidos de sistemas distribuídos.

2 Base de Dados: Modelação, Entidades e Justificações

A base de dados foi desenvolvida com o objetivo de garantir armazenamento persistente, controlo de acessos e rastreabilidade das operações realizadas pelos utilizadores, características estas essenciais de sistemas distribuídos. A solução solicitada foi PostgreSQL, e a sua integração com Java através de JDBC.

2.1 Entidades Criadas e Justificações

Para implementar o sistema distribuído para gestão de uma Biblioteca Digital que satisfaça os requisitos mínimos do sistema apresentados no enunciado, foram criadas três entidades principais: **utilizadores**, **livros** e **emprestimos**. A seguir apresentam-se as suas funções e justificações.

2.1.1 Tabela *utilizadores*

Esta tabela armazena toda a informação relativa aos utilizadores do sistema e representa uma das duas entidades principais.
Campos:

- `id SERIAL` — identificador único e chave primária;
- `nome` e `email VARCHAR` — definidos como `NOT NULL`, sendo o `email` único;
- `data_nascimento DATE` — atributo opcional;
- `estado_operacional VARCHAR` — pode ser: 'aguarda_aprovacao', 'ativo', 'suspenso', 'aguarda_suspensao' e 'bloqueado';
- `estado_admin VARCHAR` — indica se o utilizador está aprovado para usar o sistema;
- `motivo_suspensao VARCHAR` — indica o motivo pelo qual o utilizador foi suspenso;

- `justificacao_remocao VARCHAR` — justificação da remoção da suspensão e reativação do utilizador;
- `suspensoes_count INT` — contador utilizado para bloqueios automáticos.

Justificações:

- o atributo **email** foi definido como `UNIQUE` de forma a evitar utilizadores duplicados;
- o utilizador pode ficar suspenso por decisão de um administrador ou se devolver livros danificados;
- o **contador de suspensões** possibilita a alteração automática do estado operacional do utilizador para "bloqueado", quando atinge 3 suspensões.

2.1.2 Tabela *livros*

Esta tabela contém os livros disponibilizados na Biblioteca Digital e representa a segunda entidade principal.
Campos:

- `id SERIAL` — identificador único e chave primária;
- `titulo` e `autor VARCHAR` — definidos como `NOT NULL`;
- `categoria VARCHAR` — atributo opcional;
- `estado_operacional VARCHAR` — pode ser: 'aguarda_aprovacao', 'disponível', 'emprestado' e 'manutencao';
- `estado_admin VARCHAR` — indica se o livro está aprovado no sistema.

Justificações:

- não definimos nenhum atributo como `UNIQUE` considerando que pode haver várias cópias do mesmo livro em uma biblioteca;

2.1.3 Tabela *emprestimos*

Regista todos os empréstimos realizados no sistema e representa a operação principal entre as duas entidades mencionadas anteriormente.
Campos:

- `id SERIAL` — identificador único e chave primária;
- `utilizador_id INT` — chave estrangeira para *utilizadores*(id);

- `livro_id INT` — chave estrangeira para *livros*(id);
- `data_emprestimo DATE` — definido como `NOT NULL` e indica quando o empréstimo foi iniciado;
- `data_devolucao DATE` - indica quando o livro foi devolvido;
- `estado VARCHAR` — pode ser: 'ativo', 'concluido', 'devolucao_pendente' e 'suspenso';
- `motivo_negacao VARCHAR` - indica o motivo pelo qual a devolução foi recusada pelo administrador.

Justificações:

- as **chaves estrangeiras** mencionadas foram definidas para associar o utilizador e livro respetivos ao empréstimo;
- a **data_devolucao** não toma nenhum valor quando o empréstimo é iniciado mas sim quando o utilizador inicia a devolução;

2.2 Fluxo dos Estados Operacionais

2.2.1 Utilizadores

- **aguarda_aprovacao**: Estado inicial do utilizador após o seu registo.
- **ativo**: Estado após aprovação administrativa ou após a aprovação de um pedido de remoção de suspensão, permitindo ao utilizador interagir com o servidor.
- **suspenso**: Cada vez que o utilizador entra neste estado, o seu contador de suspensões é incrementado. Este estado pode ser definido manualmente pelo administrador ou automaticamente pelo sistema, caso uma devolução de livro seja negada. O utilizador fica impedido de realizar qualquer atividade e necessita de efetuar um pedido de remoção da suspensão, que deverá ser autorizado pelo administrador.
- **aguarda_suspensao**: Estado operacional em que o utilizador se encontra após submeter um pedido de remoção de suspensão.
- **bloqueado**: Estado automático após o utilizador atingir 3 suspensões no sistema. Também pode ser definido manualmente pelo administrador ou automaticamente caso o pedido de remoção da suspensão seja negado. O utilizador fica definitivamente impedido de interagir com o sistema, podendo apenas regressar após uma nova aprovação administrativa.

2.2.2 Livros

- **aguarda_aprovacao**: Estado inicial do livro após o seu registo.

- **disponivel**: Estado após aprovação administrativa ou após devolução aceite pelo administrador. O livro fica disponível para interação com o servidor.
- **emprestado**: Estado do livro após a realização de um empréstimo por um utilizador. Durante este estado, não pode ser emprestado novamente.
- **manutencao**: Estado do livro durante o processo de devolução. Enquanto estiver neste estado, o livro não pode ser emprestado.

2.2.3 Empréstimos

- **ativo**: Estado inicial do empréstimo após a sua criação.
- **devolucao_pendente**: Estado do empréstimo após ser feito um pedido de devolução de um livro.
- **concluido**: Estado do empréstimo se o administrador aprovar a devolução do livro.
- **suspenso**: Estado do empréstimo caso a devolução do livro seja negada.

3 Arquitetura Geral do Sistema e Comunicação Cliente–Servidor

O sistema desenvolvido segue o modelo **cliente–servidor**, onde um servidor central é responsável pela lógica principal da aplicação e pela gestão da base de dados PostgreSQL. Os clientes comunicam com o servidor de formas distintas, consoante o seu tipo (geral ou admin).

3.1 Arquitetura Geral do Sistema

- **Cliente Geral** — utilizado para registos de entidades, consultas e operações básicas. Comunica com o servidor através de chamadas remotas usando **Java RMI**.
- **Cliente Administrador** — ferramenta de administração do sistema. Comunica com o servidor através de **sockets TCP** com objetos Java serializados. Suporta ações administrativas como aprovar utilizadores, aprovar livros, suspensões, desbloqueios e alterações de estado/dados.
- **Servidor Central** — concentra a lógica do sistema e acede à base de dados PostgreSQL.

Esta divisão permite separar responsabilidades e introduzir dois estilos distintos de comunicação distribuída, como solicitado no enunciado.

3.2 Comunicação Cliente–Servidor

A comunicação foi implementada recorrendo a duas tecnologias complementares: **RMI** e **sockets**.

3.2.1 Justificação da utilização de RMI

A escolha de **Java RMI** para o Cliente Geral deve-se a várias vantagens:

- Permite invocar métodos remotos como se fossem locais, reduzindo complexidade;
- Facilita o transporte automático de objetos sem necessidade de gerir protocolos manualmente;
- Garante tipagem estática e tratamento de exceções ao nível da linguagem Java;
- Simplifica a implementação de operações mais frequentes e de alto nível, típicas do cliente normal.

Assim, RMI foi mais adequado para chamadas bem estruturadas e repetitivas, como registar utilizadores, consultar livros ou pedir empréstimos.

3.2.2 Justificação da utilização de Sockets

A comunicação por **sockets** foi utilizada para o Cliente Administrador, com envio e receção de objetos serializados. A opção justifica-se por:

- Maior controlo sobre a comunicação e formato dos dados transmitidos;
- Menos sobrecarga comparada ao RMI, adequado para eventos administrativos pontuais;
- Permite construir protocolos específicos e ampliar funcionalidades sem dependência do registry RMI;
- Atende ao requisito do enunciado de utilizar um segundo método de comunicação distribuída.

O administrador envia pedidos através de `Socket`, o servidor processa o objeto recebido e responde com outro objeto serializado.

3.2.3 Vantagens da combinação RMI + Sockets

A utilização de ambos os modelos traz benefícios importantes:

- Demonstra diferentes abordagens de comunicação distribuída.
- Permite comparar um modelo orientado a chamadas remotas (**RMI**) com um modelo orientado a transmissão de dados (**sockets**).
- Aumenta a flexibilidade da arquitetura — um cliente pode evoluir sem afetar o outro.
- Casos de uso distintos ficam melhor organizados e mais seguros.

No geral, o sistema beneficia da simplicidade do RMI para operações do dia-a-dia e do controlo manual dos sockets para funcionalidades administrativas sensíveis.

3.2.4 Desafio na Implementação de Duas Comunicações Distintas

Durante o desenvolvimento, surgiu um desafio relevante: o servidor precisava atender simultaneamente pedidos vindos do **RMI** e pedidos recebidos via **sockets**. Inicialmente, o servidor apenas registava a interface RMI e aguardava chamadas remotas, o que impedia o uso paralelo de sockets, pois não havia um processo dedicado a ouvir conexões TCP.
A solução adotada foi criar uma **thread adicional no servidor** responsável exclusivamente por:

1. Abrir um `ServerSocket` num porto configurado;
2. Ficar permanentemente em `listen()`;
3. Aceitar novas ligações de clientes administradores;
4. Criar uma nova **thread** por pedido, garantindo concorrência;
5. Receber e enviar objetos serializados.

Desta forma, o servidor passou a suportar **dois canais ativos em simultâneo**: um para RMI e outro para sockets, sem bloquear a execução e sem interferência entre eles. Este mecanismo foi essencial para cumprir os requisitos da aplicação e garantir que ambos os clientes podem operar ao mesmo tempo, tal como num ambiente real.

4 Funcionalidades dos Clientes

Além de implementarmos todas as funcionalidades obrigatórias solicitadas no enunciado, implementámos outras mais de forma a complementar o sistema e com o objetivo de demonstrar a nossa criativade.

4.1 Cliente Geral

4.1.1 Principais

Utilizador

- **registarUtilizador**: Regista um novo utilizador na base de dados com nome, email e data de nascimento, deixando o estado operacional da entidade em *aguarda_aprovacao*. Apenas são permitidos emails que não existam previamente na base de dados.
- **apagarUtilizadorSeSemEmprestimos**: Permite apagar um utilizador através do seu ID caso não tenha nenhum empréstimo com o estado *ativo* ou *devolucao_pendente*. Caso existam empréstimos ativos, é retornada uma mensagem de erro.
- **consultarEstadoUtilizador**: Consulta o estado operacional do utilizador.

Livros

- **registarLivro**: Regista um livro na base de dados com título, autor e categoria, com estado operacional inicial *aguarda_aprovacao* e estado administrador *nao_aprovado*.
- **listarLivrosDisponiveis**: Retorna uma lista de livros disponíveis, ou seja, com estado operacional *disponivel* e estado administrativo *aprovado*. Caso a lista esteja vazia, é apresentada uma mensagem informando que não existem livros disponíveis.
- **consultarEstadoLivro**: Permite consultar o estado operacional de um livro através do seu ID.
- **consultarHistoricoLivro**: Permite consultar todo o histórico de empréstimos de um livro.

Empréstimos

- **realizarEmprestimo**: Permite ao utilizador realizar um empréstimo caso o mesmo esteja aprovado/ativo. Caso o estado operacional do utilizador seja *aguarda_aprovacao*, *suspenso* ou *bloqueado*, é retornado um erro. O livro também deve estar *aprovado* e *disponivel*. Apenas são necessários o ID do utilizador e o ID do livro.
- **listarEmprestimosAtivosPorUtilizador**: Retorna a lista de todos os empréstimos ativos de um utilizador através do seu ID.
- **consultarEstadoEmprestimo**: Permite ao utilizador consultar o estado de um empréstimo.

4.1.2 Adicionais

- **sugerirLivrosPorTitulo**: Dado o título de um livro, sugere outros livros com o mesmo autor ou categoria.
- **realizarDevolucao**: Permite realizar a devolução de um livro através do ID do livro a devolver. O estado do empréstimo muda para *devolucao_pendente* e o estado do livro para *manutencao*. A devolução fica pendente de aprovação administrativa.
- **pedirRemoverSuspensao**: Caso seja realizada uma ação que exija o ID de um utilizador com estado *suspenso*, a ação é bloqueada. O utilizador pode, então, fazer um pedido de remoção da sua suspensão ao administrador, indicando uma justificação.

—

4.2 Cliente Administrador

4.2.1 Principais

Utilizador

- **listarUtilizadoresPorEstado**: Retorna a lista de utilizadores filtrada pelo seu estado administrativo (*aprovado* ou *nao_aprovado*).
- **aprovarUtilizador**: Permite aprovar um utilizador, alterando o seu estado operacional para *ativo* e o seu estado administrativo para *aprovado*. Também limpa o motivo de suspensão e o contador de suspensões.
- **alterarEstadoUtilizador**: Permite alterar o estado operacional do utilizador para *ativo*, *suspenso* ou *bloqueado*.
- **alterarDadosUtilizador**: Permite alterar os dados básicos do utilizador (nome, email, data de nascimento). Caso um campo seja deixado vazio, o valor atual é mantido.

Livros

- **listarLivrosPorEstado**: Retorna uma lista de livros filtrada pelo estado administrativo (*aprovado* ou *nao_aprovado*).
- **aprovarLivro**: Permite aprovar um livro, alterando o seu estado operacional para *disponivel* e o estado administrativo para *aprovado*, permitindo que seja emprestado.
- **alterarEstadoLivro**: Permite alterar o estado operacional de um livro para *disponivel*, *emprestado* ou *manutencao*.

- **alterarDadosLivro**: Permite alterar os dados básicos do livro (título, autor, categoria). Caso um campo seja deixado vazio, o valor atual é mantido.
- **consultarHistoricoLivro**: Retorna o histórico completo de empréstimos de um livro.

4.2.2 Adicionais

- **gerirDevolucoes**: Retorna a lista de empréstimos com estado *devolucao_pendente*, permitindo ao administrador aceitar ou rejeitar a devolução pretendida. Caso seja aceite, o empréstimo é concluído e o livro volta a estar *disponivel*. Caso seja rejeitada, o empréstimo passa a *suspenso*, o livro continua em *manutencao* e o utilizador é suspenso (incrementando o contador de suspensões). É também solicitado ao administrador um motivo para a rejeição.
- **gerirPedidosRemocaoSuspensao**: Retorna a lista de utilizadores com estado *aguarda_suspensao*, permitindo ao administrador aceitar ou negar a remoção da suspensão. Se o pedido for aceite, o utilizador volta a poder interagir com o sistema. Caso contrário, o utilizador é automaticamente bloqueado e precisa de nova aprovação para voltar a interagir com o sistema.

5 Configuração da Base de Dados PostgreSQL

Para que o sistema possa ser executado corretamente, é necessário configurar previamente a base de dados PostgreSQL, criar o utilizador de acesso, atribuir permissões e carregar as tabelas que suportam o funcionamento da aplicação.

5.1 Instalação do PostgreSQL

Caso ainda não esteja instalado, o PostgreSQL pode ser obtido em:
`https://www.postgresql.org/download/`

5.2 Criar a Base de Dados

Depois de instalado, abrir uma consola PostgreSQL e executar:

```
CREATE DATABASE biblioteca;
```

Esta base de dados será utilizada pelos clientes e pelo servidor para armazenar utilizadores, livros e empréstimos.

5.3 Criar um utilizador com permissões

O servidor Java acede à base de dados através das credenciais indicadas no ficheiro `config/db.properties`:

```
db.url=jdbc:postgresql://localhost:5432/biblioteca
db.user=admin
db.password=admin123
```

Por isso, deve ser criado um utilizador PostgreSQL com este nome e palavra-passe:

```
CREATE USER admin WITH PASSWORD 'admin123';
```

Dar permissões sobre a base de dados:

```
GRANT ALL PRIVILEGES ON DATABASE biblioteca TO admin;
```

5.4 Importar as Tabelas (script.sql)

O projeto inclui um script SQL em:

```
db/script.sql
```

Este ficheiro contém a criação de todas as tabelas necessárias:

- utilizadores
- livros
- emprestimos

Para importar o script via terminal:

```
psql -U admin -d biblioteca -f db/script.sql
```

Após este passo, todas as tabelas ficam criadas com os respetivos campos, restrições, estados e chaves estrangeiras.

5.5 Verificar se tudo foi criado corretamente

No terminal PostgreSQL:

```
\c biblioteca;
\dt;
```

Deverão aparecer três tabelas essenciais:

```
utilizadores
livros
emprestimos
```

5.6 Configuração de Acesso no Servidor Java

O acesso do Java ao PostgreSQL é configurado no ficheiro:

`config/db.properties`

Este ficheiro deve permanecer na pasta `config` do projeto, uma vez que a classe `Database.java` o carrega automaticamente no arranque do servidor.

5.7 Driver JDBC

Na pasta `lib/` encontra-se o driver PostgreSQL necessário para o Java comunicar com a base de dados:

`lib/postgresql-42.7.3.jar`

O `Makefile` já inclui automaticamente este ficheiro no classpath, pelo que não é necessária configuração adicional.

5.8 Conclusão e Utilização do Sistema

Após estes passos, a base de dados encontra-se totalmente configurada e pronta para ser utilizada. Para usar o sistema basta compilar todas as classes:

```
make
```

De seguida executar o servidor:

```
make run-server
```

Por fim executar qualquer cliente:

```
make run-cliente-geral
make run-cliente-admin
```