UNIVERSIDADE DE ÉVORA
CURSO DE ENGENHARIA INFORMÁTICA
2024/2025

# Sistemas Distribuídos

## → Relatório ←

Sistema de Gestão de Aluguer de Automóveis

**Docente: José Saias**
Miguel Aleixo - 51653
Pedro Freire- 52215

## Objetivos do Trabalho

Com este trabalho pretende-se desenvolver uma aplicação distribuída para simular um sistema de gestão de aluguer de veículos. O sistema deverá permitir o registo de veículos, gestão do estado de aluguer e realizar consultas e aprovações remotamente. A aplicação deverá incluir um servidor e dois clientes: um geral (para consultas e registos) e um administrativo (para gestão e aprovações), acessíveis remotamente para verificar e atualizar as informações dos veículos.

## Desenvolvimento

➔ **Base de Dados:**

Foi implementada uma base de dados com 3 tabelas; **cliente, veículo, aluguer** que foram criadas da seguinte forma:

```
create table cliente(name varchar(100),
idNumber int not null ,
numeroTele int,
clienteID int primary key);

create table veiculo(matricula varchar(6),
 modelo varchar(50),
  tipo varchar(50),
   localizacao varchar(100),
    estadoAluguer varchar(100),
     estadoAdmin boolean,
      veiculoID serial primary key);


create table aluguer(veiculoID integer references
veiculo(veiculoID),
 clienteID integer references cliente(clienteID),
 valor float ,
 dataInicio varchar(10),
  duracaoPrev float,
  aluguerID serial primary key);
```

## ➔ Classes

Utilizou-se a interface **RMI (Remote Method Invocation)** para estabelecer a ligação entre o servidor e os clientes no sistema desenvolvido. A implementação inicial incluiu a definição das classes **Veículo**, **Aluguer** e **Cliente**, cada uma contendo os métodos de acesso (getters e setters) necessários para a manipulação dos dados.

Em seguida, procedeu-se à criação e configuração da base de dados no **PostgreSQL**, onde foram estabelecidas as tabelas correspondentes. Implementou-se também a classe **ConexaoBD**, cuja função é permitir operações de inserção e pesquisa na base de dados, bem como estabelecer a ligação entre o servidor e o sistema de gestão de dados, viabilizando essas operações.

Para a estruturação do sistema, definiram-se as classes Servidor, **RemoteObject** e **RemoteObjectImpl**, que em conjunto formam a base do projeto. A comunicação entre essas classes é essencial para o funcionamento do sistema. A classe Servidor é responsável por inicializar o objeto remoto (**RemoteObjectImp**l), que implementa a interface **RemoteObject**, e configurar a conexão com a base de dados. O objeto remoto contém as funções que gerem a base de dados, as quais são implementadas na classe **ConexaoBD**.

Por fim, desenvolveram-se as classes **ClienteServer** e **ClienteAdmin**, que, utilizando as operações implementadas na **ConexaoBD** e os objetos remotos definidos nas respectivas classes, permitem a realização das funcionalidades propostas no projeto. Essas funcionalidades incluem o registo de clientes, veículos e alugueres, a busca de veículos por meio de filtros específicos e, no caso da **ClienteAdmin**, a aprovação de veículos para aluguer.

## Compilação e Execução do código

Para compilar o código, utilizamos o seguinte comando, que compila todos os arquivos .java, usando a biblioteca PostgreSQL localizada em resources/postgresql.jar, e coloca os arquivos compilados na pasta build:

- java -cp resources/postgresql.jar -d build *.java

Inicia o registro RMI (Remote Method Invocation) na porta 9000, especificando a pasta build como o classpath:

- rmiregistry -J-classpath -Jbuild 9000

Para iniciar a classe do servidor, use o comando abaixo. Ele especifica o driver JDBC do PostgreSQL (org.postgresql.Driver) e executa o servidor na porta 9000, configurado para a base de dados trabalho1, com o user user1 e a senha umaPas:

- java -cp build:resources/postgresql.jar -Djdbc.drivers=org.postgresql.Driver servidor 9000 localhost trabalho1 user1 umaPass

Para o cliente comunicar com o servidor, inicializamos a classe clienteServer usando o comando abaixo, que conecta o cliente ao servidor RMI na porta 9000:

- java -cp build:resources/postgresql.jar clienteServer localhost 9000

Finalmente, para aceder funcionalidades administrativas, iniciamos a classe clienteAdmin, que também se conecta ao servidor na porta 9000:

- java -cp build:resources/postgresql.jar clienteAdmin localhost 9000

## Conclusão

Este trabalho permitiu-nos aplicar as bases teóricas aprendidas nas aulas teóricas e uma compreensão mais detalhada dos principais conceitos da cadeira de Sistemas Distribuídos.