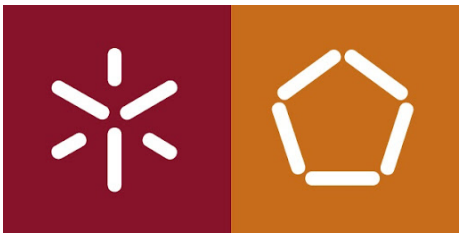

**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

# Sistemas Distribuídos

## Trabalho prático

**Dezembro 2024**

Desenvolvido por:

Rodrigo Ferreira - a104531
Rodrigo Fernandes - a104175
Diogo Esteves - a104004
André Ribeiro - 104436

# Índice

# 1 Introdução

Este relatório apresenta o desenvolvimento de um serviço de armazenamento de dados partilhado, onde a informação é mantida num servidor e acedida remotamente. A implementação segue a arquitetura cliente-servidor, utilizando sockets TCP para comunicação. O projeto teve como objetivo implementar um armazenamento de dados chave-valor acessível por múltiplos clientes simultaneamente, incorporando funcionalidades básicas e avançadas conforme descrito no enunciado.

# 2 Arquitetura do Sistema

O sistema segue uma arquitetura modular dividida em três componentes principais:

- **Biblioteca *ClientHandler***: Implementa o protocolo para comunicação com o servidor, fornecendo métodos para operações como put, get, multiPut e multiGet.
- **Servidor**: Mantém o armazenamento chave-valor em memória, processa pedidos de clientes de forma concorrente e garante a atomicidade das operações.
- **Interface do Cliente**: Oferece uma interface de linha de comandos para interação com o sistema.
- ***ClientBatch***: Permite executar multiplos pedidos ao servidor para testes

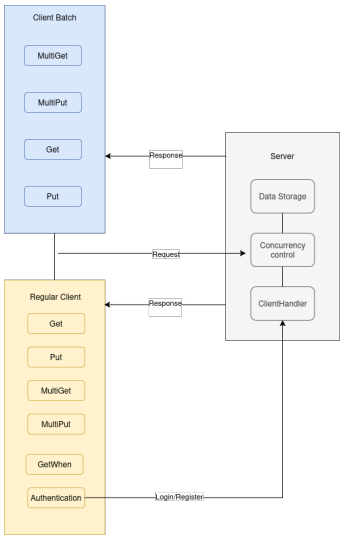


Figure 1: Arquitetura do Sistema de Armazenamento de Dados Distribuído.

### 2.1 Design do Protocolo

A comunicação entre o cliente e o servidor utiliza um protocolo binário personalizado baseado nas classes `DataInputStream` e `DataOutputStream`. A tabela abaixo resume os comandos suportados:

| **Comando** | **Descrição** |
|---|---|
| CMD_PUT | Armazena ou atualiza um par chave-valor. |
| CMD_GET | Recupera o valor de uma chave. |
| CMD_MULTIPUT | Armazena múltiplos pares chave-valor de forma atómica. |
| CMD_MULTIGET | Recupera valores para um conjunto de chaves. |
| CMD_LOGIN | Autentica um utilizador. |
| CMD_REGISTER | Regista um novo utilizador. |
| CMD_GETWHEN | Realiza uma leitura condicional. |
| CMD_EXIT | Encerra a conexão. |

Table 1: Comandos do Protocolo de Comunicação

## 3 Implementação

### 3.1 Implementação do Servidor

O servidor gere múltiplos clientes simultaneamente utilizando threads. Os aspetos principais incluem:

- **Autenticação**: Os utilizadores devem registar-se e autenticar-se antes de aceder ao serviço.
- **Controlo de Concorrência**: O servidor utiliza locks e condições para gerir o acesso simultâneo aos recursos partilhados de forma segura.
- **Armazenamento de Dados**: Os pares chave-valor são armazenados numa estrutura de dados em memória, segura para uso em múltiplas threads.
- **Operações Atómicas**: As operações de escrita e leitura são implementadas de forma a garantir a atomicidade.

### 3.2 Implementação do Cliente Batch

O cliente batch foi projetado para realizar operações automáticas e repetitivas com o servidor. As principais funcionalidades incluem:

- **MultiPut**: Envio simultâneo de múltiplos pares chave-valor, com confirmação de sucesso ou falha para cada operação.
- **MultiGet**: Recuperação de valores para um conjunto de chaves, exibindo as chaves encontradas e notificando quando nenhuma é localizada.
- **Put e Get**: Suporte para operações simples, inserindo ou recuperando valores individualmente.
- **Execução em Loop**: Repetição automática de operações com limite configurável de iterações, garantindo testes prolongados e avaliação de desempenho.

## 4 Funcionalidades

### 4.1 Funcionalidades Básicas

- **Autenticação e Registo**: Os utilizadores autenticam-se com um nome de utilizador e uma palavra-passe.
- **Put e Get**: Suporte para inserção e recuperação de um único par chave-valor.
- **MultiPut e MultiGet**: Gerem operações em lote de forma atómica.

### 4.2 Funcionalidades Avançadas

- **Leituras Condicionais**: Implementa a operação `getWhen`, bloqueando o pedido até que as condições sejam satisfeitas.

## 5 Conclusão

Este projeto implementou com sucesso um sistema robusto de armazenamento de dados distribuído, atingindo todos os objetivos propostos. O design garante escalabilidade, atomicidade e gestão eficiente de concorrência, fornecendo uma base fiável para melhorias futuras. Algumas melhorias potenciais incluem:

- **Persistência de Dados**: Implementar um mecanismo para armazenamento persistente de dados aumentaria a fiabilidade do sistema e garantiria a sobrevivência das informações após reinícios do servidor.

- **Melhoria de Escalabilidade**: Expandir o sistema para suportar um maior número de clientes concorrentes e cargas de trabalho mais elevadas. Para isso pode-se recorrer a estratégias como uma *ThreadPool* para melhorar a eficiência na utilização das *Threads* no servidor
- **Funcionalidades Avançadas**: Adicionar suporte a tipos de dados e operações mais complexas poderia ampliar a aplicabilidade do sistema.