**Universidade do Minho**
Escola de Engenharia
Licenciatura em Engenharia Informática

# Sistemas Distribuídos

## Projeto Prático: Sistema de Gestão de Séries Temporais

**Grupo 9**

| **Número** | **Nome** |
|---|---|
| A106902 | Francisco Martins |
| A107293 | Hugo Soares |
| A107372 | Nuno Rebelo |
| A106807 | Marco Sèvegrand |

Braga, 2 de janeiro de 2026

# 1 Introdução

Este projeto implementa um motor de base de dados especializado no armazenamento e processamento eficiente de séries temporais de vendas. O sistema permite que múltiplos clientes registem eventos de vendas em tempo real, recuperem agregações estatísticas (quantidade, volume, média, máximo) e recebam notificações de padrões específicos. O foco principal incidiu na conciliação de três desafios: suportar volumes massivos de dados através de persistência em disco, manter excelente performance através de cache em memória, e garantir a integridade dos dados num ambiente multi-utilizador altamente concorrente.

# 2 Arquitetura do Sistema

O sistema foi desenhado em camadas: um servidor multi-threaded que recebe pedidos de clientes, delegando o processamento a um conjunto de threads de trabalho, enquanto a receção de conexões nunca é interrompida. A persistência segue um modelo de separação entre dados voláteis (dia corrente em RAM) e histórico (ficheiros em disco). No cliente, uma biblioteca abstrai completamente a complexidade da rede, oferecendo uma interface síncrona enquanto gere assincronia de respostas em background através de um demultiplexer baseado em tags.

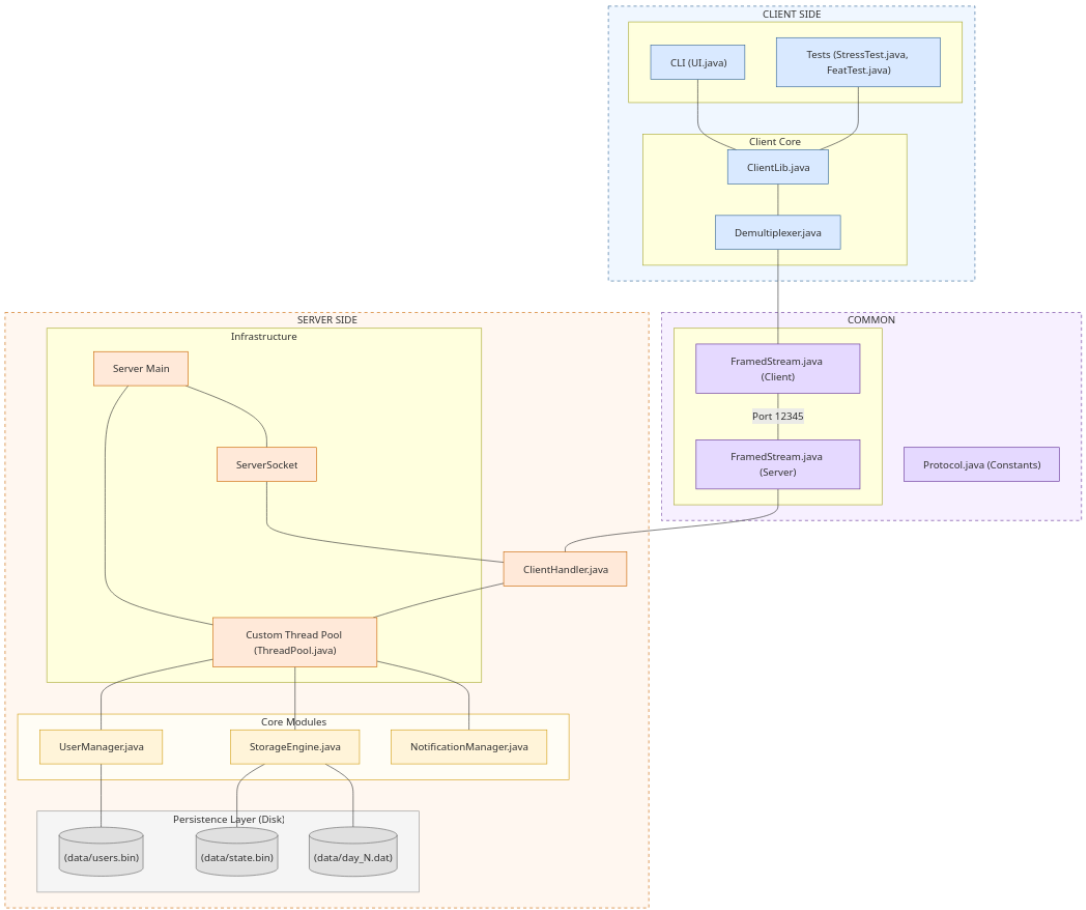


Figura 1: Diagrama da arquitetura do sistema e fluxo de mensagens entre componentes.

# 3 Implementação dos Requisitos

## 3.1 Autenticação e Gestão de Sessão

O sistema exige autenticação para qualquer operação excepto registo e login. A autenticação é centralizada no `UserManager`, que utiliza um mapa protegido por mutex para manter o registo de utilizadores. Os dados de utilizadores são persistidos em ficheiro binário após cada novo registo, assegurando que as contas sobrevivem reinicializações do servidor.

## 3.2 Registo de Eventos e Transição Temporal

Os eventos do dia corrente residem em buffer volátil, otimizado para inserção rápida. A operação `NEW_DAY` simula a progressão do tempo: fecha o dia atual, persiste-o em disco, incrementa o contador de dias e notifica o sistema de notificações. Esta operação é atómica do ponto de vista dos clientes, garantindo consistência na transição de estado.

## 3.3 Agregações Estatísticas e Cache Lazy

As operações de agregação seguem o modelo lazy: o cálculo é adiado até ao pedido do cliente. O sistema mantém uma cache de segundo nível que armazena resultados parciais por dia e por produto. Em vez de re-processar ficheiros históricos a cada consulta, o sistema combina resultados em cache, reduzindo drasticamente o I/O em pedidos repetidos. Isto permite que o servidor escale a consultas frequentes sem degradação de performance.

## 3.4 Filtragem Eficiente de Dados

Para a listagem de eventos históricos, o protocolo utiliza compressão por dicionário: nomes de produtos repetidos são substituídos por identificadores inteiros no fluxo de dados transmitido. Esta técnica é especialmente eficaz em séries temporais com milhares de eventos para o mesmo produto, reduzindo significativamente o consumo de largura de banda sem perder informação.

## 3.5 Notificações Bloqueantes com Timeout Implícito

As notificações requerem que a thread do cliente fique suspensa até que o padrão seja satisfeito. O sistema utiliza primitivas de sincronização (`Condition.await()`) para este fim. Uma decisão crítica de projeto foi o tratamento da transição de dias: se o dia termina enquanto uma thread aguarda uma notificação, a condição é invalidada e o cliente recebe uma resposta negativa, cumprindo o comportamento de tempo real exigido.

## 3.6 Ligação TCP Única com Multiplexagem

O requisito de manter uma única ligação TCP por cliente, suportando múltiplos pedidos concorrentes, é satisfeito através de um protocolo de tags. O cliente aloca uma etiqueta única para cada pedido e aguarda a resposta correspondente. Um demultiplexer no cliente redireciona

respostas da rede para a thread correta. Esta arquitetura elimina problemas de bloqueio em cabeça de linha, permitindo que pedidos rápidos não fiquem retidos atrás de agregações pesadas.

### 3.7 Persistência com Gestão de Memória

O motor de armazenamento respeita dois parâmetros: $S$ (número máximo de séries em RAM) e $D$ (janela de dias históricos). Uma cache LRU implementada através de `LinkedHashMap` garante que apenas as $S$ séries mais consultadas ocupam memória. Séries excedentes são processadas diretamente do disco sob demanda. O sistema também limpa automaticamente dados que saem da janela de retenção de $D$ dias, mantendo o uso de disco sob controlo.

## 4 Validação e Teste

A validação funcional confirmou que o sistema gere corretamente notificações simultâneas e consecutivas, bem como a transição entre dias. O teste de carga submeteu o servidor a 25.000 inserções simultâneas de eventos concorrentes. A concordância perfeita entre os contadores locais dos testes e os valores devolvidos pelo servidor demonstra a fiabilidade do sistema sob carga e a ausência de condições de corrida.

## 5 Utilização de Ferramentas de IA

Conforme especificado no enunciado, o uso de IA foi limitado a tarefas periféricas que não constituem o núcleo das competências de Sistemas Distribuídos avaliadas. A interface de utilizador (`UI.java`) foi gerada para acelerar a construção de um cliente interativo, assim como os testes (`FeatTest.java` e `StressTest.java`) foram gerados para validar automaticamente as funcionalidades implementadas. Todas as modificações foram validadas manualmente e integradas após verificação de conformidade com o protocolo.

## 6 Conclusão

O sistema desenvolvido cumpre integralmente os requisitos de eficiência e escalabilidade. A arquitetura modular permite que cada componente (armazenamento, autenticação, notificações) evolua independentemente. As técnicas empregadas—cache LRU, agregação lazy, demultiplexing no cliente, compressão de dados—são mecanismos consolidados na engenharia de sistemas distribuídos que garantem performance adequada mesmo perante volumes de dados massivos e elevada concorrência.