Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Sistemas Distribuídos

Trabalho Prático: Base de Dados para Séries Temporais

Grupo X

Número	Nome
Axxxxx	Francisco Martins
Axxxxx	Hugo Soares
Axxxxx	Nuno Rebelo
A106807	Marco Sèvegrand

Braga, 9 de janeiro de 2026

1 Introdução

Este relatório detalha o desenvolvimento de um serviço distribuído para o registo e agregação de eventos de vendas em séries temporais. O sistema foi concebido para suportar múltiplos clientes concorrentes através de sockets TCP, utilizando um protocolo binário otimizado. O foco principal da implementação incide na eficiência da gestão de memória e disco, bem como na correta sincronização de pedidos concorrentes.

2 Arquitetura e Fluxo do Programa

O sistema segue um modelo cliente-servidor clássico, mas introduz camadas de abstração para lidar com a concorrência e a persistência de dados.

2.1 Fluxo de Operação

1. **Inicialização**: O servidor inicia os componentes de armazenamento e autenticação, carregando o estado anterior de ficheiros binários.
2. **Conexão**: Um cliente liga-se via TCP. O servidor cria uma thread dedicada (`ClientHandler`) para ler do socket.
3. **Pedidos**: O cliente (via `UI`) envia pedidos através da `ClientLib`. Para suportar concorrência no mesmo socket, o `Demultiplexer` gere as etiquetas (*tags*) de cada mensagem.
4. **Processamento**: No servidor, a leitura do socket é separada do processamento lógico através de um `ExecutorService` (Pool de threads), permitindo que um único cliente submeta múltiplos pedidos sem ser bloqueado por operações lentas.
5. **Resposta**: O servidor devolve a resposta com a mesma *tag*, que o cliente usa para desbloquear a thread que aguardava o resultado.

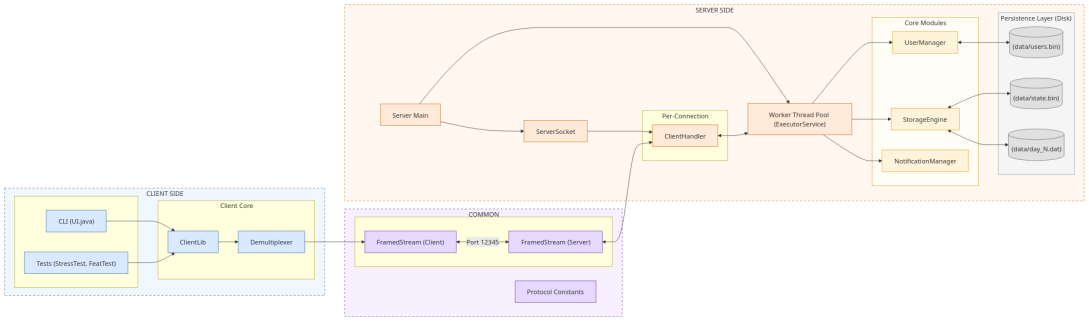

Figura 1: Diagrama da arquitetura do sistema e fluxo de mensagens entre componentes.

3 Implementação por Requisitos

3.1 1. Autenticação e Registo do Utilizador

Implementação: Realizada na classe `UserManager`. Os utilizadores são armazenados num `HashMap` protegido por um `ReentrantLock`.

Decisões: Conforme exigido, o servidor rejeita qualquer comando se a flag `authenticated` for falsa. A persistência é feita num ficheiro `users.bin`.

3.2 2. Registo de Eventos e Novo Dia

Implementação: A classe `StorageEngine` mantém uma lista `currentEvents` em memória. O comando `newDay` move estes eventos para um ficheiro `day_X.dat`.
Decisões: A decisão de usar `DataOutputStream` com `BufferedOutputStream` visou minimizar o número de acessos físicos ao disco, seguindo as diretrizes do enunciado para eficiência.

3.3 3. Agregação de Informação (Lazy)

Implementação: O método `aggregate` processa apenas os dias solicitados. Utiliza-se a `aggCache` para guardar resultados parciais por par dia/produto.
Decisões: O cálculo só ocorre no primeiro pedido (*lazy loading*). Os resultados são mantidos em cache e descartados quando saem da janela de interesse D, poupando recursos de CPU.

3.4 4. Filtragem com Serialização Compacta

Implementação: No `ClientHandler.handleFilter`, é construído um dicionário de nomes de produtos únicos presentes na série temporal.
Decisões: O envio de um dicionário inicial seguido de índices inteiros cumpre o requisito de reduzir o tráfego de rede em listas extensas de eventos repetitivos.

3.5 5. Notificações de Ocorrências

Implementação: A classe `NotificationManager` orquestra o bloqueio de threads usando `Condition`.
Decisões: Esta abordagem permite a espera passiva eficiente. As threads são acordadas por `signalAll()` quando ocorre uma venda ou quando o dia é encerrado.

3.6 6. Suporte a Clientes Multi-threaded

Implementação: O `Demultiplexer` desacopla a receção de dados da sua utilização, associando cada resposta à sua thread de origem.
Decisões: Esta estrutura garante que operações longas no servidor não bloqueiam a fluidez de outros pedidos do mesmo cliente.

3.7 7. Persistência e Limites de Memória (S e D)

Implementação: A cache `loadedSeries` utiliza a política LRU para limitar o número de ficheiros em RAM.
Decisões: O limite S é respeitado rigorosamente; dados extraídos do disco para agregações fora da cache são processados de forma volátil se a cache estiver na sua capacidade máxima.

4 Makefile e Execução

A `Makefile` automatiza a gestão do projeto, desde a criação de diretórios até à execução de testes.

4.1 Comandos Disponíveis

- `make`: Compila todos os componentes e prepara as pastas de dados.
- `make run-server`: Lança o serviço de backend.
- `make run-client-cli`: Lança a consola interativa de utilizador.
- `make run-feat-test`: Executa testes funcionais às notificações.
- `make run-stress-test`: Valida a integridade sob carga concorrente.
- `make clean`: Limpa o ambiente de desenvolvimento.

5 Utilização de Ferramentas de IA

Declara-se a utilização de IA para auxílio na redação técnica deste relatório, na estruturação dos comentários Javadoc e na definição inicial dos cenários de teste de stress. Toda a lógica de sincronização foi validada manualmente pelo grupo.

6 Conclusão

O sistema desenvolvido responde a todos os desafios do enunciado, garantindo persistência, concorrência e uma gestão de recursos otimizada para o processamento de séries temporais de grande escala.

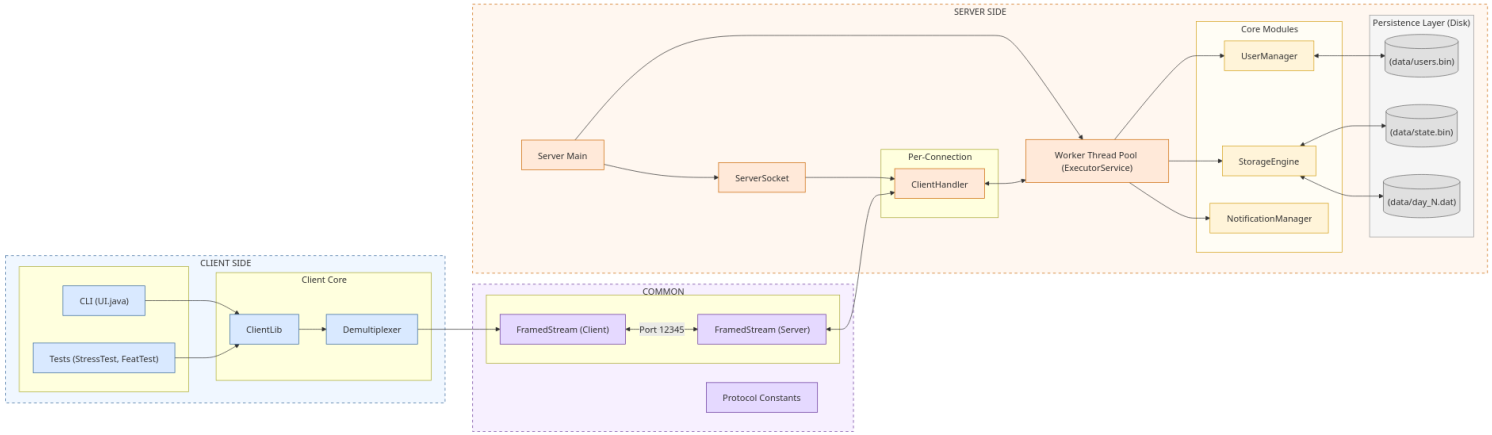

Figura 2: Diagrama Detalhado da Arquitetura (Vista Panorâmica)