Sistemas Distribuídos — Trabalho Prático
Base de dados para séries temporais

Engenharia Informática
Universidade do Minho

2025/2026

Informações gerais

- Cada grupo deve ser constituído por 4 elementos, obrigatoriamente inscritos no *eLearning* até 14 de novembro de 2025.
- Deve ser entregue o código fonte e um relatório de até 6 páginas (A4, 11pt) no formato PDF (a capa não é contabilizada neste limite).
- O código fonte e o relatório devem ser inteiramente da autoria dos membros do grupo sem contribuição de terceiros, obtidas online ou utilizando ferramentas de IA, a menos que explicitamente assinaladas e autorizadas pelos docentes.
- O trabalho deve ser entregue até às 23:59 do dia 9 de janeiro de 2026 no *eLearning*.
- A apresentação do trabalho é obrigatória e será em dia e hora a agendar posteriormente por cada grupo.
- Cada grupo deve organizar a sua apresentação de forma a que todos os elementos participem espontaneamente, o que constitui um fator de avaliação.

Resumo

Neste projeto pretende-se a implementação de um serviço de registo de eventos em séries temporais e de agregação de informação, relativos a venda de produtos, em que a informação é mantida num servidor e acedida remotamente. Clientes interagem com o servidor através de sockets TCP, de forma a inserir e consultar informação. O servidor atende clientes concorrentemente e armazena a informação. Assuma que todas as operações se referem apenas aos D dias anteriores, sendo D um parâmetro de inicialização do servidor.

Valorizam-se estratégias que diminuam a contenção e minimizem o número de threads acordadas. O relatório deverá descrever a arquitetura e protocolos do sistema, bem como justificar as principais decisões de desenho assumidas na elaboração do projeto. O serviço deverá suportar as seguintes funcionalidades.

1 Autenticação e registo do utilizador

- Registo e autenticação de utilizadores, dado o seu nome e palavra passe.
- Sempre que um utilizador desejar interagir com o serviço, deverá estabelecer uma conexão e ser autenticado pelo servidor. O servidor não deverá processar qualquer pedido de um cliente que não esteja autenticado, exceto os próprios pedidos de autenticação/registo.

2 Registo de eventos

- Operação para adicionar um evento à série temporal do dia corrente, dado o nome do produto, quantidade, e preço de venda.

Implemente também uma operação no servidor para começar um novo dia, para simular a passagem do tempo. Esta poderá ser usada internamente no servidor ou ser disponibilizada para um cliente de administração. Poderá ser usada nomeadamente por código de teste do serviço, invocada a diferentes ritmos conforme os cenários de teste.

3 Agregação de informação

Estas operações dizem respeito a dias anteriores, já concluídos (ou seja, excluindo o dia corrente). Como estas envolvem algum processamento, e podem dizer respeito a um pequeno subconjunto de produtos e intervalos de dias, implemente todas estas agregações de um modo *lazy, on demand*. Ou seja, percorrendo uma série temporal relativa a um dado dia e agregando a informação relativa ao produto, da primeira vez que é solicitada uma agregação para o produto, fazendo *caching* do resultado no servidor, para ser usado em agregações futuras. A informação em *cache* deve ser descartada quando já não for relevante.

- Quantidade de vendas. Devolve a quantidade de unidades de um produto p vendidas nos últimos d dias anteriores. ($1 \leq d \leq D$).
- Volume de vendas. Devolve o valor total das vendas de um produto p nos últimos d dias anteriores. ($1 \leq d \leq D$).
- Preço médio de venda. Devolve o preço médio de venda de um produto p nos últimos d dias anteriores. ($1 \leq d \leq D$).
- Preço máximo de venda. Devolve o preço máximo de venda de um produto p nos últimos d dias anteriores. ($1 \leq d \leq D$).

4 Filtrar eventos de uma série temporal

- Devolve a lista de eventos relativa aos produtos pertencentes a um conjunto c, do dia anterior d. ($1 \leq d \leq D$).

Como a lista de eventos a devolver ao cliente é potencialmente grande e com nomes de produtos aparecendo repetidos, deverá ser implementada uma serialização de dados eficiente, que represente a lista de forma compacta.

5 Notificação de ocorrências

- Vendas simultâneas. Reportar mal tenham sido vendidos dois produtos específicos, p_1 e p_2, no dia corrente, devolvendo `true` nesse caso. A operação deve ficar bloqueada até tal acontecer, ou até o dia terminar, devolvendo `false` nesse caso.
- Vendas consecutivas. Reportar assim que tenham sido efetuadas n vendas consecutivas de um mesmo produto no dia corrente, devolvendo o nome do produto. A operação deve ficar bloqueada até tal acontecer, ou até o dia terminar, devolvendo `null` nesse caso.

6 Suporte a clientes *multi-threaded*

- O cliente deverá poder ter várias *threads* concorrentemente a submeter pedidos ao servidor. Um pedido que demore mais tempo a ser servido não deverá impedir outros pedidos que o cliente submeta concorrentemente de serem servidos.

7 Persistência de séries temporais e limite destas em memória

Admita que cada série pode ter biliões de eventos e não é viável ter todas as séries dos D dias anteriores em memória. Persista a série de cada dia para disco e mantenha em memória, para além do dia corrente, no máximo S séries, com $S < D$ um parâmetro de inicialização do servidor. Quando é requisitada uma agregação que exige informação que não está em cache, esta pode processar uma série que esteja em memória, ou ler uma série do disco. Neste caso, se já estiverem S séries em memória a serem processadas, a informação lida do disco deve ir sendo processada e descartada ao longo da agregação de modo a não exceder o limite S. Uma série que não esteja a ser processada poderá ser descartada para dar lugar a outra.

Programa Servidor

Programa servidor deve ser implementado em Java, usando *threads* e *sockets* TCP, mantendo a informação relevante para suportar as funcionalidades acima descritas, receber conexões e input dos clientes, bem como fazer chegar a estes a informação pretendida.

Biblioteca do cliente

Deverá ser disponibilizada uma biblioteca (conjunto de classes e interfaces) que proporcione o acesso à funcionalidade do serviço descrita acima. Esta biblioteca deve ser independente da interface com o utilizador e deverá ser escrita em Java usando *threads* e *sockets* TCP.

Interface do utilizador

Deverá ser também disponibilizada uma interface com o utilizador que permita interagir com o serviço através da biblioteca cliente. Esta interface deverá também ser escrita em Java e tem como único objetivo a interação com o serviço para pequenos testes e durante a apresentação do trabalho.

Avaliação de desempenho

De modo a perceber o desempenho do serviço, conceba diferentes cenários de testes, incluindo cargas de trabalho com diferentes tipos de operações, testes de escalabilidade (i.e., verificar o desempenho do sistema com o aumento do número de clientes a submeter pedidos) e de robustez (i.e., verificar o que acontece quando um cliente não consome as respostas que lhe são enviadas). O relatório deve incluir uma reflexão sobre os resultados dos testes efetuados relacionando-os com as decisões de desenho que tomou.

Requisitos

Na implementação devem ser satisfeitos os seguintes requisitos:

- Para cada cliente, deve haver apenas uma única conexão com o servidor.

- O protocolo de comunicação deverá ser num formato binário, através de código desenvolvido no trabalho, podendo recorrer apenas a `Data[Input|Output]Stream`.
- Relativamente à comunicação, serialização, e concorrência, devem ser usadas apenas as APIs do Java que foram apresentadas e usadas na resolução dos guiões práticos. Não podem usar outras APIs do Java ou pacotes de software adicionais.

Utilização de ferramentas de IA

O objetivo deste trabalho é saber como aplicar e explicar em detalhe as primitivas e técnicas que fazem parte do programa da UC e não apenas saber que devem ou quando devem ser aplicadas. O código gerado por ferramentas de IA relativo a estas primitivas e técnicas não é por isso considerado uma resolução válida do trabalho.

O uso de ferramentas de IA para reproduzir a mesma solução várias vezes ou para gerar código que não faça parte dos objetivos fundamentais da UC, por exemplo, na construção de testes, não é desencorajado, mas nesse caso a resolução considerada é o conjunto de *prompts* usados e a justificação de como o código gerado foi validado. O código gerado só por si não é considerado.

Em caso de dúvida devem consultar os docentes.