Exercício - Sistema de transporte de aplicativo

A atividade prática consiste na implementação, em Java com RMI, de um sistema distribuído chamado RideMatch RMI, que simula o funcionamento básico de um aplicativo de transporte semelhante ao Uber. O objetivo é permitir que passageiros solicitem corridas e que motoristas registrados sejam alocados automaticamente conforme disponibilidade, prioridade e localização, de forma concorrente e tolerante a falhas.

O sistema deve ser composto por um servidor despachante e clientes que representam motoristas e passageiros. O despachante é responsável por gerenciar as requisições de corrida, manter o registro de motoristas disponíveis e atribuir cada solicitação a um motorista adequado. Os motoristas e passageiros se comunicam com o despachante via chamadas remotas RMI, e o motorista também deve possuir um serviço de retorno (callback) para ser notificado sobre novas corridas ou cancelamentos.

O funcionamento geral é o seguinte: motoristas registram-se no servidor informando seus dados e passam ao estado “disponível”. Passageiros solicitam corridas indicando local de partida, destino e prioridade (VIP ou padrão). O despachante recebe essas requisições e as insere em uma fila de prioridade, considerando primeiro as corridas VIP e, em seguida, as padrão. Para cada requisição, o servidor deve procurar o motorista mais próximo, considerando a distância euclidiana entre os pontos. Em caso de empate, deve escolher o motorista disponível há mais tempo. O servidor deve tentar atribuir a corrida em até três segundos; se não conseguir, a solicitação expira.

Quando um motorista é selecionado, o servidor envia uma notificação de atribuição através do callback remoto. O motorista tem até dois segundos para aceitar a corrida. Caso o callback falhe ou o tempo se esgote, o servidor deve automaticamente reatribuir a corrida a outro motorista disponível. Após aceitar, o motorista deve sinalizar o início e a conclusão da corrida, voltando então ao estado de disponível. Passageiros também podem cancelar corridas antes que sejam aceitas, e nesse caso o servidor deve notificar o motorista, se já houver atribuição.

A implementação deve usar ExecutorService para gerenciar múltiplas threads de correspondência (matching), ReentrantLock ou Semaphore para proteger as estruturas de dados compartilhadas e evitar condições de corrida, e mecanismos de timeout para respeitar os prazos estabelecidos. Os dados de motoristas, passageiros e corridas podem ser mantidos em memória, e o servidor deve registrar logs informando a criação de requisições, alocações bem-sucedidas, falhas e reatribuições. A robustez é essencial: o sistema deve continuar funcionando mesmo que callbacks falhem, motoristas fiquem offline ou threads sejam interrompidas.

Você foi contrato para implementar os métodos da inteface remota da classe ServicoDespachante no servidor RMI da classe ServidorDespachante.

1) processarMatching(RequisicaoCorrida requisicao)

2) encontrarMelhorMotorista(RequisicaoCorrida requisicao)

3) notificarMotorista(String motoristaId, String atribuicaoId)

4) tratarFalhaConfirmacao(String atribuicaoId)