Universidade do Minho
Escola de Engenharia

Sistemas Distribuidos
Relatório do Trabalho Prático - Grupo 30
2022/2023
Repositório (GitHub)

Afonso Xavier Cardoso Marques 94940
Vicente de Carvalho Castro 91677
José Diogo Lopes Faria 95255
Pedro Calheno Pinto 87983




Índice

1 Introdução

Este trabalho consistiu no desenvolvimento de um programa que faz a gestão de uma frota de trotinetes elétricas, semelhante às aplicações das empresas de mobilidade Lime e Bird. Permite que vários utilizadores consultem uma lista customizada de localizações de trotinetes que estejam disponiveis para reserva.

Um utilizador tem também a possibilidade de ser recompensado por percorrer certas distâncias. Para além disso, pode escolher se quer ser notificado destas recompensas ou não.

O programa consiste num servidor que é capaz de comunicar com vários clientes que atuam em concorrência e que tem sempre guardado informação sobre a localização das trotinetes. Cada cliente comunica com o servidor através de sockets TCP.

Tal como é dito no enunciado, o mapa da nossa aplicação consiste numa grelha NxN pelo que as diferentes localizações no mapa são identificadas por um par de números inteiros.

2 Classes Usadas

O primeiro passo para a resolução do enunciado foi a criação de classes que fossem apropriadas ao problema em questão. Segue-se a listagem das mesmas:

2.1 Cliente

Um cliente é uma entidade capaz de interagir com o servidor. Um cliente pode ou não ter associado a si uma conta. Sempre que o programa é iniciado, o cliente tem a opção de criar uma nova conta ou usar uma que já esteja guardada no servidor.

Quando um cliente se autentica, os seus dados são enviados para o servidor para validação, isto é, verifica se o nome de utilizador de facto existe e se a palavra passe introduzida corresponde. Também vai verificar se não está a ser usado por outro cliente. Caso os dados sejam validados, o cliente passa a poder interagir com o programa.

Se escolher criar uma nova conta, ser-lhe-á pedido o username e password e estes serão adicionados à lista de contas do programa. Como o foco do projeto não é a gestão da base de dados, decidimos que esta lista ficaria guardada em memória e não em disco, pelo que toda a informação armazenada é perdida sempre que o servidor é reiniciado.

Em ambos os casos, o servidor informa o cliente relativamente ao sucesso/insucesso da operacão escolhida.

2.2 Servidor

O servidor tem como principal função receber os pedidos dos clientes através de um socket TCP. É aqui que são armazenadas as contas que estão associadas ao serviço bem como o mapa onde estão posicionadas as trotinetes. Para atingir isto, o servidor faz uso das classes Contas e Mapa.

O servidor é sempre inicializado com quatro contas. Novas contas podem ser adicionadas mas serão perdidas quando o servidor é reinicializado. O mesmo acontece com o mapa que é inicializado sempre com os mesmos parâmetros e é alterado conforme os pedidos dos clientes; quando reinicia o servidor, as alteraçoes ao estado do mapa também são perdidas.

Também cabe ao servidor ter a função de notificar os clientes sobre as recompensas que estes adquiriram. A opção de ser notificado por ser desativado pelo cliente.

Como o servidor serve vários clientes, é necessário que tanto as estruturas de dados do servidor como as classes do qual faz uso tenham controlo de concorrência. Na classe de Contas apenas necessitamos de um lock, mais especificamente um ReentrantReadWriteLock, pois os métodos desta classe são de leitura e escrita numa estrutura de dados (neste caso um Map). Na classe Mapa temos dois locks: um ReentrantReadWriteLock "lock- mapa" e um ReentrantLock "lock-geral". O "lock-mapa" é usado maioritariamente nas operações de listagem nomeadamente

a listagem das trotinetes e das recompensas disponiveis naquele momento. O "lock geral" é usado quando não é prático ter um ReentrantReadWriteLock, como por exemplo na reserva das trotinetes e deslocação.

2.3 Mapa

O Mapa é o espaço "virtual" onde se encontram as trotinetes. Aqui temos um *HashMap* para armazenar as trotinetes em que a *key* corresponde ao ID individual da trotinete; um *HashMap* com as várias posições e um *HashMap* para guardar as trotinetes que são reservadas.

O mapa suporta vários tamanhos, fazendo uso de uma variavél de instâcia N ao qual é atribuido um valor inteiro, no entanto decidimos que o mapa deve ser iniciado sempre com N=20, ficando com uma dimensão de 20x20.

A classe mapa possui dois locks, um ReentrantLock denominado "lock geral" e um ReentrantReadWriteLock denominado "lock mapa" que usamos em operações onde apenas vamos fazer uma leitura do estado do mapa.

Foi decidido fazer a geração das recompensas nesta classe, criando para esse efeito o método *geraRecompensas (List<Mapa.Posicao> A)* que para cada posição da lista de posições que recebe cria uma recompensa que é colocada numa lista de recompensas. No final essa lista é devolvida.

2.4 Trotinetes

A classe Trotinete serve meramente para representar as trotinetes como objetos com os quais podemos interagir. Cada trotinete tem um número de identificação e uma posição no mapa. Possui também um estado de ocupação representado por um valor booleano onde se este for falso é porque está ocupada e se for verdadeiro é porque está livre e pode ser reservada e deslocada.

Cada trotinete é inicializado com uma posição e estado pré-definidos, cujos valores são alterados conforme os pedidos dos clientes.

2.5 Recompensa

Para levar a uma boa distribuição das trotinetes pelo mapa, existe um sistema de recompensas, em que utilizadores são premiados por levaram trotinetes estacionadas num local A para um local B. Em cada momento existe uma lista de recompensas, atualizada ao longo do tempo, sendo cada recompensa identificada por um par origem-destino, com um valor de recompensa associado, calculado em função da distância a percorrer.

Para gerar um local A foi criado um método *geraLocalA(List<Trotinete> livres)* que recebe a lista de trotinetes livres e verifica percorrendo dois ciclos, se a posição da primeira trotinete está a uma distância menor que a variável *DISTANCIA* (à qual foi atribuido o valor de 2) da segunda trotinete. Faz break do ciclo quando a condição é verificada e retorna a posição da primeira trotinete.

Para geral um local B foi criado um método *geraLocalB(List<Trotinete> trotinetes)* onde se inicializa uma posição do Mapa a *null* a que se dá o nome de *rand*. Já dentro de um ciclo *while* atribuiem-se dois valores aleatórios (usamos a biblioteca *java.util.Random*) ao primeiro e segundo valor do tuplo textitMapa.Posicao. De seguida inicia-se um novo ciclo que percorre a lista de trotinetes e verifica se a posição gerada anteriormente em *rand* está a uma distância maior que a variável *DISTANCIA* de uma trotinete livre qualquer. Se a condição se verificar retorna a posição *rand* atual, se não se verificar cria-se uma posição *rand* nova com valores diferentes e repete-se o ciclo que percorre as trotinetes.

Para calcular o valor de uma recompensa foi criado o método *formulaValor(Mapa.Posicao inicio, Mapa.Posicao fim)* que calcula a distância do local A ao local B. A partir desse valor, cria-se uma condição em que se a distância for maior que 10 o valor atribuido à recompensa será 100, se a distância for menor que 10 o valor atribuido será 50. O método devolve o valor da recompensa.

3 Conexão

A conexão entre o cliente e o servidor é assegurada por três classes: Connection, Demultiplexer e Frame.

3.1 Frame

A classe Frame serve para encapsular as mensagens trocadas entre cliente e servidor, associando a cada frame uma tag, o username do cliente a quem é destinada, e um array de bytes com a mensagem que se pretende enviar. A tag serve para garantir que a mensagem é entregue no sitio certo, pelo que decidimos atribuir uma tag especifica para cada serviço disponiblizado pela aplicação.

Por exemplo se um cliente quiser fazer a autenticação será criada uma frame com tag 0 ou se quiser listar as trotinetes livres é usada uma frame com tag 2.

O uso do username do cliente na frame não é muito necessário mas facilita as operações de autenticação e registo de contas permitindo cortar o número de frames enviadas de duas para apenas uma.

3.2 Connection

A classe Connection é aquela que trata do envio e receção de mensagens, que são encapsuladas em frames, fazendo uso do socket a ela associado.

No fundo atua como a ponte entre o cliente e o servidor que permite a passagem de mensagens entre os dois lados, usando para esse efeito DataInputStreams e DataOutputStreams. Contem os métodos de *send()* e *receive()* e faz uso dos *locks* "sendLock" e "receiveLock":

- "sendLock" especifico para o método send();

- "receiveLock" especifico para o método receive();

3.3 Demultiplexer

Temos ainda a classe de Demultiplexer que atua apenas do lado do cliente. Tendo em conta que um cliente pode ter vários threads e que estes devem ter de ficar à espera que uma mensagem seja recebida, o Demultiplexer assegura que isso acontece, recorrendo a métodos de *await()* e *signal()*. Basicamente coloca os threads a "dormir" enquanto a mensagem que querem receber não chega, e envia um sinal para "acordar" quando finalmente chega.

4 Conclusão

Este projeto permitiu-nos consolidar os conhecimentos que fomos adquirindo ao longo do semestre nas aulas de Sistemas Distribuídos. A aplicação que desenvolvemos permitiu-nos compreender melhor conceitos como locks, sockets e threads, e a sua utilidade no mundo real.

Conseguimos implementar com sucesso as funcionaldades relativas à autenticação e registo de clientes bem como a listagem de trotinetes livres e a reserva e deslocaçao de trotinetes pelo mapa. Destacamos ainda a aplicação de regras de exclusão mútua sempre que era necessário e a forma robusta como tratamos a conexão entre cliente e servidor que vai de acordo com aquilo que foi lecionado nas aulas práticas.

As maiores dificuldades que sentimos foram na implementação das recompensas e das notificações. De facto desenvolvemos métodos que permitem fazer a gestão das recompensas, tal como já foi mencionado neste relatorio, mas por motivos que não conseguimos averiguar, a implementação dos mesmos não está a funcionar como esperávamos. As notificações foram a maior de todas as dificuldades e infelizmente não conseguimos implementar nada relativo a esta componente do projeto.

Tendo em conta os dois parágrafos anteriores acreditamos que, apesar das adversidades, conseguimos ter um projeto positivo.