Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de
Sistemas Distribuídos

Ano Letivo de 2022/2023

Relatório do Projeto
Implementação de uma plataforma de gestão de uma frota de trotinetes elétricas

Grupo 42

Cláudio Alexandre Freitas Bessa	A97063
Rafael dos Anjos Arêas	A86817
Carlos Gustavo Silva Pereira	A96867
José Fernando Monteiro Martins	A97903

10 de janeiro de 2023

Index

1 Introdução

Iniciamos este documento, notificando de algumas dificuldades que enfrentamos e que acabaram por limitar o resultado final. O projeto possui duas versões, cada uma em sua diretoria, tituladas de *src1* e *src2*.

Na diretoria *src1* está uma versão totalmente funcional do projeto, contendo controlo de concorrência e exclusão mútua que se torna inutilizado, uma vez que a aplicação não permite a utilização de vários clientes em simultâneo. Esta versão, contudo, possui alguns *bugs* por corrigir, sendo eles na atualização de trotinetes e recompensas pelo mapa e não cumpre a 100% os requisitos relativos às notificações.

Já na diretoria *src2* está uma versão não funcional do projeto, que acreditamos que siga uma linha de raciocínio muito mais perto do que é solicitado pelos requisitos, uma vez estar possibilitada ao uso de *multi−threading*. O problema nesta versão, creiamos nós, é relativo às conexões entre cliente e servidor onde não por *deadlock* mas por outro motivo após a primeira iteração a conexão bloqueia não permitindo a continuação da utilização da aplicação. Seguimos o relatório, justificando e argumentando, supondo a finalização do projeto, na versão contida na *src2* uma vez que esta, em resultado final, demonstraria um maior conjunto de conhecimento após o lecionamento desta UC.

Sistemas distribuídos consistem em múltiplos sistemas interconectados que trabalham juntos para realizarem tarefas comuns ou diferentes. Podem estar localizados em diferentes lugares geográficos e mesmo assim se comunicar através do estabelecimento de ligações através de diferentes protocolos, como por exemplo, a *TCP/IP*. Com isso, os sistemas distribuídos podem ser usados para realizar várias tarefas, como o processamento de dados de forma concorrente, enviando e recebendo dados, fazendo compartilhamento de arquivos, execução de sistemas em nuvem e etc. No mundo atual, muitas das aplicações que são utilizadas no dia-a-dia tem a implementação sobre sistemas distribuídos, tais como:

- Sistemas de mensagens: os utilizadores podem enviar mensagens para outros utilizadores do sistema através de um servidor central.
- Aplicações de distribuição de arquivos: os usuários podem baixar arquivos de outros usuários do sistema através de um servidor central.
- Jogos *online*: os jogadores podem se conectar a um servidor central para jogar com outros jogadores em tempo real.
- etc...

Com isso, é perceptível que existe uma grande importância no conhecimento e na criação desses sistemas, que pode ser criado a partir de esquemas similares a estes, tais como:

- Definição do objetivo do sistema distribuído e a implementação de como os computadores do sistema devem se comunicar e trabalhar juntos para atingir esse objetivo.
- Criação de um servidor que será responsável por gerenciar o sistema e comunicar-se com os outros computadores(clientes).

- Implementação de medidas de segurança para proteger o sistema e os dados nele contidos. Isso poderá incluir autenticação de usuários, criptografia de dados e outras medidas de segurança da rede.
- Mecanismos que garantem a exclusão mútua, que é a propriedade que garante que dois processos não acedem simultaneamente a um recurso partilhado.

Logo, para intensivar o conhecimento sobre o conteúdo lecionado, foi proposto um sistema que faça a implementação de uma plataforma de gestão de uma frota de trotinetes elétricas, sob a forma de um par cliente-servidor em Java utilizando *sockets*. *sockets* são interfaces de programação de aplicativos (APIs) que permitem que os programas se comuniquem através de uma rede *threads* são unidades de execução em um programa em que, cada uma é uma sequência independente de instruções que pode ser executada de forma paralela a outras *threads* fazendo com que possam ser executados em simultâneo, aumentando consequentemente a sua eficiência.

2 Caso de Estudo

2.1 Descrição do problema

Para este projeto é pedido a implementação de uma plataforma de gestão de uma frota de trotinetes elétricas, utilizando a linguagem de programação *Java* para criar um par Cliente-Servidor, utilizando *sockets* e *Threads*.

A plataforma tem como essência permitir que os usuários possam reservar trotinetes para o seu uso e poder estacionar elas em diferentes locais, podendo também receber recompensas/premiações se os usuários levarem a trotinete estacionadas de um local A para um local B determinado, levando assim uma boa distribuição das trotinetes pelo mapa.

O mapa é dividido em uma grade de N X N localidades, com N=20, onde as coordenadas geográficas são pares discretos de índices. A distância entre duas localidades é determinada pela norma l1, também conhecida como distância de Manhattan. Um exemplo de um mapa gerado no sistema seria o seguinte:

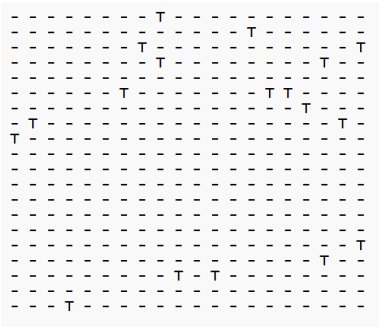

Figura 1: Exemplo de um Mapa gerado no sistema.

2.2 Formulação do sistema

Foi analisado que para a formulação do sistema o serviço deveria ter que suportar algumas funcionalidades, tais elas são:

- Autenticação e registo de utilizador, 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.
- UmalLista dos locais onde existem trotinetes livres, até uma distância fixa $D(\mathrm{D} = 2)$ de um determinado local.
- Uma lista das recompensas com origem até uma distância fixa $D(\mathrm{D} = 2)$ de um determinado local, dada por pares origem-destino.
- Reserva de uma trotinete livre, o mais perto possível de determinado local, limitado uma distância fixa D. O servidor deverá responder com o local e um código de reserva, ou código de insucesso, caso tal não seja possível.
- Estacionamento de uma trotinete dando o código de reserva e o local. O servidor deve informar o cliente do custo da viagem, em função do tempo passado desde a reserva e da distância percorrida. Caso a viagem corresponda a uma recompensa, é informado do valor da recompensa. A aplicabilidade da recompensa deve ser avaliada no estacionamento, de acordo com a lista de recompensas em vigor nesse momento
- Um cliente pedir para ser notificado quando apareçam recompensas com origem a menos de uma distância fixa D de determinado local. As notificações poderão ser enviadas muito mais tarde, devendo entretanto o cliente poder prosseguir com outras operações. O cliente poderá cancelar os pedidos de notificação

A aplicação deve suportar um sistema de recompensa, que deve correr em *background*, avaliando a distribuição das trotinetes que estão livre no mapa. A recompensa gerada para movimentar uma trotinete de um dado local A para o local B deve ser somente para quando há mais do que uma trotinete livre em A e não há nenhuma livre num raio D de B. Toda vez que algum usuário reservar uma trotinete ou estacioná-la o sistema deve reavaliar a situação das trotinetes no mapa.

3 Descrição da Solução Proposta para o Caso de Estudo

Fazendo uma contextualização *a priori* da nossa solução proposta para cada objeto em particular, na nossa aplicação de trotinetes utilizamos um total de 3 *threads*. O módulo Servidor possui duas del

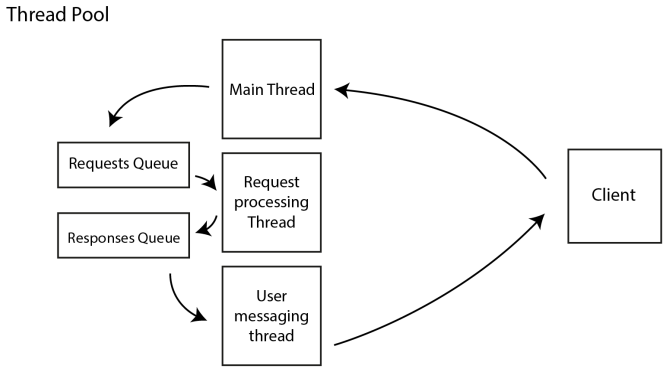

Figura 2: Fluxo de uma mensagem na aplicação

O **Cliente** inicializa na sua *main thread* que envia um pedido para o servidor. Este por sua vez terá uma *thread* a ler a *request pool*, *queue* onde é recebido os pedidos vindos do cliente. Após isso, esta primeira *thread* interpreta o pedido e redireciona para a *queue* existente na *response pool*. Seguidamente, solicita outra *thread* para ler a resposta da *response pool* e finalmente responde ao cliente. Finalizando o cliente utiliza a *thread* que já estava a funcionar, uma vez que esta está sempre ativa em *background* pronta a receber algum tipo de notificação, para ler a resposta enviada do **Servidor** e reiniciará o seu fluxo, isto se aplicável.

3.1 Servidor

Não sendo a primeira vez que nos deparávamos com uma conexão servidor/cliente já tínhamos algumas bases idealizadas que poderíamos utilizar. Para este servidor da nossa aplicação, necessitaria obviamente do conhecimento total de todo o sistema, i.e conhecimento do estado atual do mapa, isso incluindo saber as posições e o estado das trotinetes, ter o registo dos clientes da aplicação e identificar e notificar os clientes, que pretendem receber notificações, que existem recompensas disponíveis numa distância de 2.

O servidor guardando como variáveis o mapa, incluindo todas as suas variáveis internas, as suas recompensas e as contas dos utilizadores que após a sua ligação associa a mesma por cada cliente a uma *thread* com esse propósito. Essa *thread* irá inicializar um objeto que titulamos de *ProcessRequest* que não é nada mais nada menos como um *worker* que efetuará as atividades que o cliente escolher. Isto seguindo o exemplo do **ServerWithWorkers** estudado nas aulas teórico−práticas.

3.2 Cliente

O cliente, utilizando o conhecimento obtido durante as aulas práticas, efetua conexões *TCP* com o servidor e onde vai escolhendo o que fazer na aplicação. O cliente tem o conhecimento total do menu, sendo que apenas é ativado com a notificação recebida do servidor. Neste cliente utilizamos uma espécie de ligação utilizando **TaggedConnections** uma vez que os sockets são identificados pelo nome do usuário, enviando a restante respetiva mensagem.

3.3 Mapa

O mapa foi construído de forma a ser uma matriz de células, tendo este dimensões 20 por 20. Considerou-se esta como a melhor opção, sendo que assim poder-se-ia ter uma maior facilidade de acesso e identificação de células, sendo que cada uma poderia não passar de isto mesmo (uma célula) ou ter uma trotinete contida na mesma.

3.3.1 Célula

Uma célula é um elemento do mapa, todas estas têm um identificador próprio, as coordenadas correspondentes, e podem ou não conter uma trotinete na mesma. Esta classe é importante para termos forma de ter um mapa povoado com e sem trotinetes e conseguirmos originar viagens e recompensas. De forma a que o projeto funcione de acordo com o expectável.

3.3.2 Trotinete

A Trotinete é uma classe herdada de célula, sendo que esta, para além dos atributos herdados, tem um *boolean* que indica a disponibilidade da mesma e uma *Recompensa* que em caso de ter uma recompensa associada, indica a mesma e as suas características que serão descritas posteriormente.

3.4 Recompensas

Um dos requisitos das recompensas era estas serem geradas em uma *thread* específica e serem atualizadas na mesma. De tal modo, isso foi realizado, uma vez que no pleno método *main* do servidor essa *thread* é invocada e posteriormente dentro da interação com o cliente caso sejam efetuadas recompensas e/ou aconteçam outros métodos que de alguma forma alterem a situação das recompensas em si, essa *thread* é invocada.

Já a nível da classe **Recompensa** em si, esta possui uma célula inicial e uma célula final que definem onde se encontra a trotinete que tem a respetiva recompensa e o respetivo novo local.

4 Menu Interativo

O Menu Interativo é a face do projeto. Ele permite a escolha e visualização das diferentes funcionalidades implementadas pela equipa no desenvolvimento do projeto. Logo no início quando o sistema arranca o usuário terá 3 opções de escolha que são as de fazer *login*, *registro* ou *sair*, tais como na seguinte imagem:

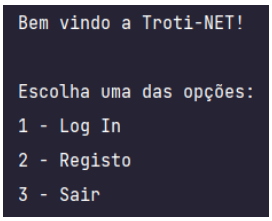

Figura 3: Menu inicial do sistema.

Logo após a autenticação do usuário no sistema ele deverá indicar a sua localização no mapa dando a sua coordenada x e y, como mostra na figura a seguir:

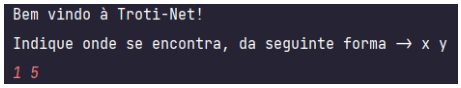

Figura 4: Exemplo de um utilizador a fornecer a sua posição

Depois do utilizador ser devidamente autenticado e ter as suas coordenadas já definidas no sistema, ele terá 4 opções de utilização do sistema com as 2 principais sendo a de reservar uma trotinete, desligar/ligar as notificações de recompensas. A última opção é a de simplesmente sair do sistema. Elas são representadas da seguinte forma:

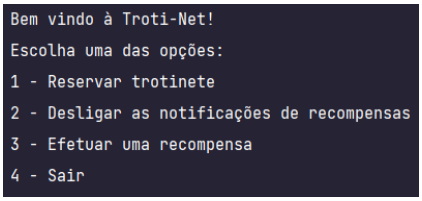

Figura 5: Menu após utilizadores encontrarem-se logados

Quando o cliente escolhe a opção de reservar uma trotinete o servidor procura no mapa as trotinetes que estão livres num raio de 2 segundo a fórmula de distância de *manhattan*, enviando para o utilizador uma lista de trotinetes em que ele pode escolher para reservar. Logo após a trotinete ser selecionada, é gerado um código de reserva. Em seguida deve indicar o local onde a trotinete será estacionada. Um exemplo desse processo é o seguinte:

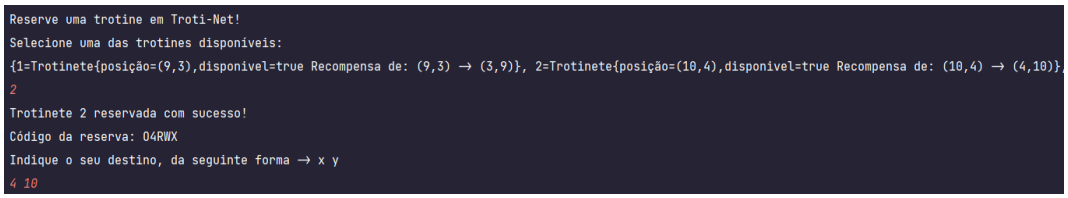

Figura 6: Exemplo de reserva de uma trotinete feita por um usuário.

Se o cliente escolher a opção de efetuar uma recompensa, ele receberá uma lista de trotinetes que estão livres num raio de 2 segundo a fórmula de distância de *manhattan* para escolher qual ele pretende realizar uma recompensa. Sabendo as posições de trotinetes livres que existem e que tem uma recompensa, o utilizador volta para o menu e escolhe a opção de reservar uma trotinete, dando as coordenada da trotine livre da sua escolha que tem uma recompensa. A seguinte figura mostra um exemplo deste processo:

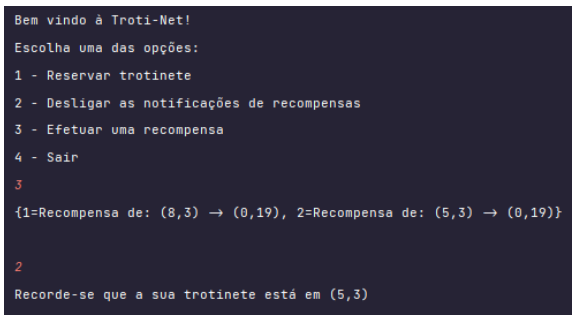

Figura 7: Exemplo da escolha de trotinetes com recompensas.

5 Discussão das decisões tomadas

5.1 Mapa ser uma matriz

Decidiu-se que o mapa seria uma matriz porque isso facilitaria o acesso às células a partir das suas coordenadas. Assim, a partir das coordenadas das mesmas iríamos ter acesso a todas e poderíamos mais facilmente fazer a gestão das mesmas, nomeadamente adicionar ou remover uma trotinete de uma certa célula. A apresentação das trotinetes e das células disponíveis é feita por uma conversão desta matriz num "mapa".

5.2 Gerar recompensas

As recompensas são geradas caso no mapa sejam identificadas 2 ou mais trotinetes em um círculo de raio 2. Após serem sinalizadas, todas essas trotinetes possuem uma recompensa ativa que uma seja realizada. Após essa realização, existirá uma atualização de recompensas que se baseia da seguinte forma: se no mesmo círculo permanecer um conjunto de 2 ou mais trotinetes, permanecerá as recompensas, caso contrário as recompensas deixarão de existir.

5.3 Trotinete herda uma célula

Como é visível no código enviado em anexo, na nossa aplicação o objeto **Trotinete** são uma extensão da classe **Célula**. Isto acontece devido a existir momentos durante a execução, o servidor pretende aceder a uma trotinete, e.g alterar, adicionar a outra célula, remover de uma célula, etc. Seguindo esta linha de raciocínio, efetuaríamos variadas duplicações de código se também tivéssemos que aceder às células do mapa e efetuar comparações para verificar se os objetos eram os mesmos. Por esse motivo as classes são herdadas uma da outra, podendo assim minimizar o código escrito e alterando duas instâncias de uma só vez.

5.4 Célula só poder ter 1 trotinete

Foi decidido que a cada célula só suportaria uma trotinete, pois visualmente ficaria mais "legível" para os utilizadores, sendo que assim teriam a capacidade de ver o mapa 1:1, algo que se existisse a opção do mapa ter mais do que uma trotinete na mesma coordenada, não seria nesta escala. Tendo isto em conta a adaptação da geração de recompensas, que foi acima descrita, e a cardinalidade das trotinetes foram as maiores mudanças uma vez reivindicar esta decisão.

6 Conclusão

Durante este semestre fomos providos das bases teóricas de vários conceitos para a execução e implementação deste trabalho prático como por exemplo métodos de comunicação sobre um par de cliente/servidor a partir de *sockets* com uma comunicação *TCP/IP*, um conceito importante que é o de exclusão mútua e também o conceito de *threads*, um recurso que permite que um processo execute várias tarefas de forma concorrente.Porém, o desenvolvimento deste projeto guarneceu-nos pelo facto de podermos explorar tais conceitos lecionados, sendo uma mais valia para tal.

Apesar de alguns contratemos na implementação do *multi−multithreading* conseguimos ultrapassar os *bugs* obtidos. Deparamos nos também com bastantes conflitos uma vez que a representação das matrizes são do tipo coluna/linha e não linha/coluna que acabaram por ser corrigidos. Com isto, acrescentamos ao nosso portfólio mais bagagem para aplicações que em *runtime* são acedidas por múltiplos clientes, algo que cada vez mais é necessário que seja implementado algum tipo de controlo.

7 Bibliografia

Todo o tipo de utilização de recursos bibliográficos encontra-se na plataforma digital da universidade, uma vez que apenas foram utilizadas ferramentas lecionas, dadas e/ou criadas nas aulas práticas desta UC.

Nota: Não ensejável para bibliografia uma vez que para aceder a tais recursos é necessário uma chave de autenticação.