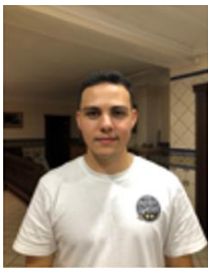
Sistemas Distribuídos

Trabalho Prático (Gestão de frota)

Universidade do Minho

Licenciatura em Engenharia Informática

Trabalho realizado pelo grupo 40 composto por:

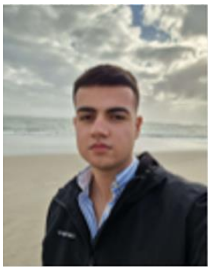


- A95442 -

André Oliveira Gonçalves

- A95323 -

Henrique Ribeiro Fernandes



- A95641 -

João Pedro Moreira Brito

Ano Letivo 2022/2023

Submetido a: 09/01/2023

Índice

1. Introdução

No âmbito da unidade curricular de Sistemas Distribuídos, foi desenvolvido um projeto tendo em vista 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* e *threads*. O objetivo do serviço é garantir aos utilizadores reservar e estacionar em diferentes locais. O mapa é uma grelha de N x N locais, neste caso optou-se por uma grelha 20 x 20, sendo as coordenadas geográficas pares discretos de índices, para calcular a distância entre dois locais considera-se a distância Manhattan.

De forma a garantir uma boa distribuição das trotinetes pelo mapa, existe um sistema de recompensas, em que os utilizadores são premiados por levarem trotinetes estacionadas num local A para um local B, onde no local A existe mais que uma trotinete livre e no local B não há nenhuma num raio D=2. Neste caso, o grupo optou por premiar os utilizadores com 20% de desconto no preço da viagem realizada, cada recompensa é identificada por um par origem-destino.

Este serviço também garante um sistema de notificações, que pode ser ativado e desativado pelos utilizadores, para que estes recebam notificações quando existem recompensas a menos de uma distância fixa D=2 de determinado local.

2. Arquitetura da Plataforma

2.1. Servidor

O servidor inicializa um *socket* na porta 1100, garantindo a troca de mensagens com o cliente, este *socket* é conhecido por todos os clientes. Assim, quando uma conexão é estabelecida, sempre que um cliente realizar um *request*, este será respondido pelo servidor. O servidor utiliza, também, métodos e dados armazenados em memória na classe App, de forma a satisfazer os pedidos realizados pelos utilizadores.

2.2 App

A classe App é onde fica armazenado o mapa com as trotinetes, em uma matriz. Esta classe possui diversos métodos essenciais para o servidor realizar as funcionalidades presentes no programa.

2.3 Cliente

O cliente conecta-se ao servidor através do *socket* criado com a porta 1100. Neste programa todos os clientes necessitam possuir uma conta associada, por isso, quando esta conexão é estabelecida, é apresentado um menu principal ao cliente, onde o mesmo pode fazer login, registar-se ou abandonar o programa. Se escolher uma das duas primeiras opções, e realizá-las com sucesso, este é direcionado para um segundo menu onde tem acesso às diversas funcionalidades presentes no programa. Cada uma destas opções, presentes nestes menus, levam à criação de uma *thread* de execução.

2.4 Comunicação Cliente ⟹ Servidor

O cliente para efetuar grande parte das funcionalidades, presentes no programa, precisa de introduzir alguns dados necessários. Estes dados são enviados para o servidor através do método send criado na classe Demultiplexer, que por sua vez, chama o método, de mesmo nome, da classe TaggedConnection. Para que o servidor possa diferenciar o tipo de mensagem enviada, os dados no *socket* são encaminhados juntamente com uma *tag*, onde cada *tag* corresponde a uma funcionalidade diferente.

2.5 Comunicação Servidor ⟹ Cliente

O servidor, para enviar os resultados obtidos com a execução das funcionalidades, recorre ao método send da classe TaggedConnection, que escreve no *socket* a *tag* e os dados. O método start da classe Demultiplexer permite organizar as respostas recebidas no cliente, num buffer que junta todos os dados recebidos, tendo em conta a *tag*, numa fila de espera. Este método executa a thread responsável por interromper todas as mensagens enviadas pelo servidor para o *socket*, assim, o método *receive* da classe Demultiplexer é invocado quando existe uma *thread* do cliente que pretende receber os dados. Quando esta fila se encontra vazia, a *thread* fica à espera que cheguem os dados a partir do método await.

3. Funcionalidades

- A autenticação e/ou o registo de um utilizador, dado o seu nome e palavra passe, são sempre necessários quando o utilizador deseja interagir com o serviço e possui uma conexão estabelecida. Esta funcionalidade é garantida pelos método trataFazerLogin() e trataRegistar() presentes na classe TextUI, os dados introduzidos pelo cliente são verificados no servidor.
- A listagem dos locais onde existem trotinetes livres, até uma distância fixa D=2 de um determinado local inserido pelo cliente, encontra-se no método trotinetes_livres(int x, int y) da classe App que recebe as coordenadas inseridas pelo cliente e retorna a lista.
- A listagem das recompensas com origem até uma distância fixa D=2 de um determinado local inserido pelo cliente, é garantida pelo método recompensas_na_areas(int X,int Y) da classe App.
- A reserva de uma trotinete livre, o mais perto possível de um determinado local indicado pelo cliente, limitado a uma distância D=2, é realizada pelo método reserva_trotinete(int x, int y) na classe App.
- O estacionamento de uma trotinete dando o código de reserva e o local, é feito com a ajuda do método reserva_trotinete da classe e o servidor informa o cliente do custo da viagem, sendo esta normal ou uma de recompensa, com a ajuda do método clientes da classe Servidor.
- A funcionalidade de um cliente pedir para ser notificado, ou desligar as notificações, para quando aparecem recompensas com origem a menos de uma distância fixa D=2 de determinado local é realizada pelo método clientes da classe Servidor, com o auxílio do método espera_notificações da mesma classe.

4. Conclusão

Em suma, a realização deste projeto permitiu-nos consolidar os conceitos de controlo da concorrência, programação com *sockets* e outros conceitos abordados no âmbito da unidade curricular de Sistemas Distribuídos.

Enquanto grupo, pensamos que tenhamos conseguido concretizar o projeto da forma correta, pois conseguimos implementar todas as funcionalidades pedidas no enunciado, tendo em conta, também, que utilizamos estratégias para diminuir a contenção e minimizar o número de *threads* acordadas.