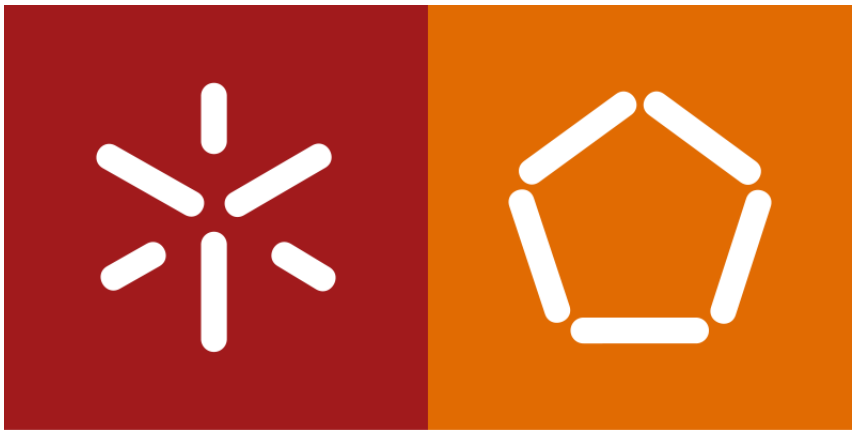

Universidade do Minho

Escola de Engenharia

Relatório

Trabalho Prático de Sistemas Distribuídos 22/23

Grupo 1:
96547 Rodrigo Freitas
95076 Pedro Oliveira
96794 Ricardo Oliveira
94595 João Pedro Cardoso

Professor:
José Pereira

Contents

1 Introdução

Sistemas distribuídos têm se tornado cada vez mais importantes na computação moderna, com aplicações que vão desde a gestão de redes em larga escala até a coordenação da execução de tarefas complexas. Neste projeto, nosso objetivo foi projetar um sistema distribuído para gerir trotinetes elétricos numa grelha NxN.

Um dos principais desafios que enfrentamos foi o desenvolvimento de um sistema de recompensas que incentiva o uso responsável dos scooters elétricos. Também precisávamos de considerar o uso de threads para garantir a eficiência e eficácia do sistema. Através da nossa pesquisa e análise, conseguimos criar um projeto abrangente que aborda competentemente essas questões.

Nas próximas páginas, vamos aprofundar nos detalhes de nosso projeto, destacando as principais características e componentes que o tornam uma solução sólida para os requisitos e funcionalidades pedidos no enunciado deste trabalho prático. Sendo os mesmos:

Funcionalidades

1. Autenticação e registo, via nome e palavra-passe
2. Listagem dos locais onde existem trotinetes livres, até uma distância fixa.
3. Listagem das recompensas até distância fixa de um local, dado por pares origem-destino.
4. Reserva de uma trotinete livre, limitado a uma distânia fixa
5. Estacionamento da trotinete dado o código da reserva e local
6. Notificação de recompensa com origem a menos de uma distância fixa D de um local.

Requisitos

1. Cada cliente estabelece uma única conexão para o servidor, onde passam pedidos, respostas e notificações.
2. Cliente e servidor comunicarão num formato binário, recorrendo apenas a Data[Input|Output]Stream.
3. Threads do servidor escreverão em apenas um socket, e apenas por threads associadas ao mesmo.
4. O gerador de recompensas deve correr no servidor em thread(s) específica(s), e nunca em threads associadas a clientes.

2 Funcionalidades

2.1 Autenticação e registo

2.1.1 Registo

O registo de utilizadores é feito pelo UserDB.java. Os mesmos, após registados, serão guardados numa Collection denomimada *users* do tipo *Map<String,User>*, sendo String o respetivo ID incremental.

1. addUser recebe um objeto u da classe *User* e faz um *getName()*
2. Adquire um reentrant write lock.
3. Username pertence à DB ?

Sim ? Liberta o lock e retorna -1

Não ? Atribui um ID seguindo a regra de incrementação, acrescenta-o à DB e autentica-se automaticamente. Só depois liberta o lock.

2.1.2 Autenticação

1. login recebe um username e password.
2. Adquire um reentrant write lock.
3. Username pertence à DB ?

Sim ? Liberta o lock e retorna false.

Não ? Cria um User u e faz getUsername(), liberta o lock e retorna o método login do User u, usando a password como parâmetro.

4. Se username e password estão corretos a função devolverá true, caso contrário false.

O método logoff segue uma metodologia semelhante, diferindo no final, em vez de devolver alguma variável, muda ou não o valor de setLoggedIn para false.

2.2 Listagem dos Locais

O principal método responsável por esta funcionalidade é o *getavailable*.

Este recebe a posição do usuário e retorna a lista de todas as posições onde se encontrem trotinetes disponíveis para uso.

Este método encontra-se na classe UserAPI.java, que é responsável pela API do usuário, ou seja, de todos os métodos que o mesmo poderá chamar e assim utilizar as funcionalidades do sistema. Primeiramente, um frame é criado e enviado pelo demultiplexer. Em seguida, quando recebermos o frame de volta, iremos verificar se recebemos uma resposta com a informação necessária. Se sim, criamos uma lista aonde serão guardadas as posições desejadas que serão retornadas posteriormente.

2.3 Listagem das Recompensas

O *checkrewards* é o principal responsável por fornecer a funcionalidade de recompensas ao nosso user. Esse método, localizado no UserAPI.java, recebe a posição atual do user e retorna uma lista completa de recompensas disponíveis, incluindo os detalhes da localização de início e término da viagem geradora de recompensa, bem como o desconto aplicável.

Para aceder essas informações, primeiro criamos e enviamos um frame para o demultiplexer. Quando recebemos a resposta, verificamos se ela contém os dados desejados e, de seguida, criamos uma lista para armazenar essas informações sobre as recompensas, que serão retornadas para o user.

2.4 Reserva de trotinete livre

Para garantir a reserva de um scooter, iniciamos o processo com o método *reservescooter*, que recebe a posição do usuário e se encarrega de preparar a reserva. Isso é realizado através da criação de um frame que é enviado pelo demultiplexer. Quando recebemos a resposta, verificamos se ela contém a informação necessária e, caso positivo, criamos a reserva atribuindo-lhe um código único.

Finalmente, armazenamos essa reserva junto com as outras, organizando-as pelo código correspondente para garantir a facilidade de acesso e gestão

2.5 Estacionamento de trotinete

Antes de tudo, recuperamos a reserva correspondente usando o código fornecido pelo usuário e finalizamos o percurso utilizando o método setEnd. Além de encerrar a reserva, esse método nos permite obter a data e hora da finalização do percurso. Em seguida, utilizamos o método *parkscooter* para criar e enviar um frame pelo demultiplexer. Quando recebemos a resposta, verificamos se ela contém a informação necessária e, caso positivo, retornamos o preço final do percurso ao usuário para que ele possa tomar conhecimento.

2.6 Notificação de recompensa

O usuário tem a opção de ativar ou desativar as notificações para qualquer localização de sua escolha, utilizando o método setnotifications. Esse método é responsável por realizar a alteração solicitada pelo usuário. Para isso, criamos e enviamos um frame para o demultiplexer. Quando recebemos a resposta, verificamos se ela contém a informação necessária e confirmamos se a modificação foi efetuada com êxito. As notificações são enviadas ao usuário através do NotificacationHandler.java e do RewardHandler.java.

Enquanto o programa está em execução, verificamos periodicamente se há novas notificações a serem enviadas. Para fazer isso, recuperamos a lista de recompensas com o método getListRewards e, caso hajam novas disponíveis, criamos um frame com todas essas informações e as enviamos para o cliente através do método send, que é responsável por serializar o pedido e enviá-lo para o usuário

3 Requisitos

3.1 Cliente estabelece única conexão com servidor

O sistema cliente-servidor implementado permite que cada cliente estabeleça uma única conexão para o servidor e use essa conexão para enviar pedidos, receber respostas e receber notificações. Isso permite que o servidor gerencie o tráfego de rede e as solicitações dos clientes de maneira eficiente, além de permitir que os clientes enviem solicitações e recebam respostas em tempo real. Para isso, quando o cliente é executado, usando o clientMain.java, uma UserAPI é criada, onde cada cliente tem uma Connection, com o seu respetivo socket, e um Demultiplexer.

Assim, através do menu, o cliente poderá enviar pedidos de queries ao servidor que irá tratá-lo e retornar uma resposta. De seguida, uma thread é criada nesse cliente, que será utilizada para receber e apresentar as notificações. Essa thread estará sempre a correr durante a execução do programa. Desse modo, sempre que uma nova notificação, enviada pelo servidor, chegar ao cliente, ela será apresentada imediatamente.

3.2 Escrita de threads do server

A comunicação entre o cliente e o servidor é gerida, pelas classes Connection.java e Frame.java. Antes do envio ser realizado, a mensagem passará pelo processo serialize e só depois será de fato enviado. Quando uma mensagem é recebida, existe o processo contrário, onde a mensagem passará pelo processo deserialize antes de ser processada.

Serialize é o processo de converter um objeto em uma sequência de bytes antes de ser transmitida através de rede. Deserialize é o processo inverso, que consiste em converter uma sequência de bytes em um objeto. A serialização é útil porque permite transmitir objetos completos com todos os seus campos e relacionamentos. É uma forma eficiente de comunicação entre clientes e servidores, pois os dados são transmitidos de um lado para o outro de forma rápida, compacta e abstrata.

Além disso, o formato de dados binário é independente da plataforma, o que significa que os dados podem ser lidos e interpretados corretamente independentemente do sistema operacional, do hardware, da linguagem e do middleware usado pelo cliente ou servidor. As desvantagens é que se torna mais difícil de realizar debug, comparando com formato de texto por exemplo. Além disso, é um tipo de mensagem frágil e que requer cuidado ao utilizar.

3.3 Threads do gerador de recompensas

Neste programa usamos, maioritariamente read-write locks, que é um mecanismo de sincronização que permitem que múltiplos threads leiam um recurso compartilhado ao mesmo tempo, mas permite que apenas um thread escreva de cada vez, sendo úteis em situações em que a leitura de um recurso é muito mais comum do que a escrita, pois permitem que várias threads leiam o recurso ao mesmo tempo, o que pode melhorar o desempenho do sistema.

O que neste caso se encaixa perfeitamente no que nós desejamos, uma vez que, dentro do servidor, vários clientes desejam consultar os recursos, porém a quantidade de clientes que desejam realizar alterações é bastante menor. Quando o servidor é iniciado, um Server Socket é criado. De seguida, sempre que um cliente deseja se conectar ao servidor, esse socket lhe é partilhado, pois será utilizado pelo servidor para o envio de mensagens ao cliente. Além do mais, uma thread é criada para a execução e gestão do cliente.

4 Conclusão

Ao longo deste projeto, tivemos a oportunidade de explorar os desafios e a importância de projetar e implementar corretamente um sistema distribuído.

Na realização do mesmo, ficou evidente a relevância de sistemas distribuídos em geral, que são amplamente utilizados nma variedade de aplicações e desempenham um papel fundamental na operação eficiente de redes em larga escala. É crucial compreender os desafios e considerações envolvidos na projeção de sistemas distribuídos para garantir sua eficácia e escalabilidade.

Este projeto proporcionou a oportunidade única de aplicar os conceitos aprendidos sobre sistemas distribuídos num contexto mais prático, o que foi uma experiência enriquecedora e valiosa. Ao final, podemos dizer com confiança que saímos daqui com uma compreensão aprofundada da teoria e da prática da unidade curricular de sistemas distribuídos.