FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Googol: Motor de pesquisa de páginas Web

Sistemas Distribuídos 2022/2023

Meta 1

Licenciatura em Engenharia Informática

Trabalho realizado por:
Bruno Eduardo Machado Sequeira nº 2020235721
Rui Pedro Capelas Santos nº 2020225542

Índice:

Introdução:

Neste projeto foi-nos pedido que criássemos um motor de pesquisa de páginas Web, contendo um subconjunto de funcionalidades semelhantes aos serviços Google, Bing e DuckDuckGo, incluindo indexação automática e busca. O sistema terá todas as informações relevantes sobre as páginas, como por exemplo, o URL, o título, uma citação de texto. Após o utilizador realizar uma busca de uma palavra / várias palavras, obtém a lista de páginas que contenham as palavras pesquisadas. Os utilizadores também podem sugerir URLs para estes serem indexados pelo sistema. Partindo desses URLs, o sistema deve indexar recursivamente todas as ligações encontradas em cada página.

Com este projeto iremos aplicar os nossos conhecimentos para programar um sistema de pesquisa de páginas Web com uma arquitetura cliente-servidor. Utilizar sockets Multicast para comunicação entre Servidores (neste caso, entre Downloaders e Barrels), seguir um modelo Multi Thread para desenhar os servidores, criar uma camada de acesso aos dados usando Java RMI, e garantir a disponibilidade da aplicação através de redundância com failover.

Nota:
Para que possa executar o projeto, tem um tutorial no “README.md” na pasta do projeto.

Arquitetura de Software:

A aplicação consiste em 5 programas: Downloaders, Index Storage Barrels, Search Module, Clientes, Queue_url.

Os downloaders são as componentes que obtêm as páginas Web em paralelo, analisando-as com o jsoup e envia através de Multicast para todos os Index Storage Barrel. (MULTICAST MAIS ABAIXO)

O index Storage Barrel é o servidor central que possui todos os dados da aplicação, este que recebe através do multicast enviados pelos Downloaders. Sendo que todos os index Storage Barrel têm de conter a mesma informação entre eles.

Search Module é a componente visível pelos clientes. O cliente comunica com este utilizando RMI.

Os Clientes é o cliente RMI usado pelos utilizadores para aceder às funcionalidades da aplicação. Tendo uma UI simples e invoca os métodos remotos no servidor RMI (Search module).

Queue Url é uma componente que guarda os URLS encontrados pelos Downloaders na forma de uma fila. Sendo este o primeiro programa a ser executado.

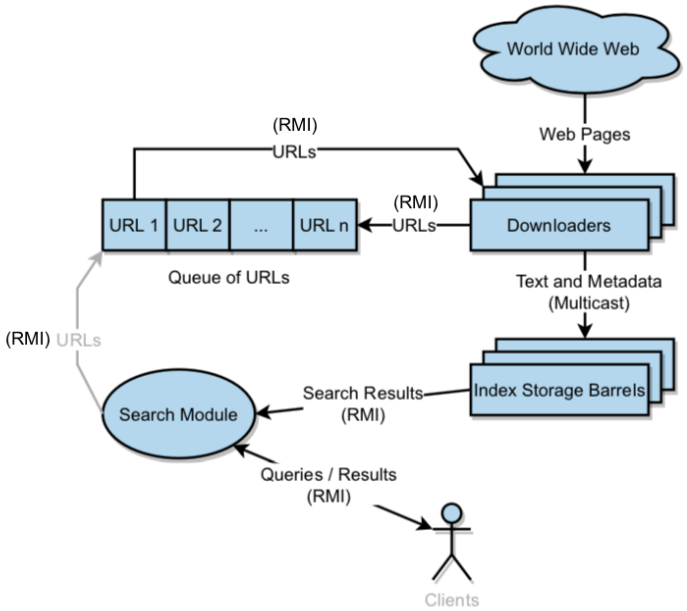

Figura 1 - Arquitetura da aplicação

Para armazenar todos os dados utilizamos uma Base de Dados. Esta que será explicada na página 7.

As ligações entre as componentes/programas:

- **Cliente - Search Module:** utilizamos RMI, tal como pedido no enunciado, o cliente pode chamar métodos do servidor, por exemplo, indexar um url, fazer uma pesquisa. Implementamos um mecanismo que permite que um objeto remoto invoque um método num objeto local do cliente, ou seja, callBack em RMI. O seu funcionamento básico é o cliente registra um objeto remoto do servidor RMI, e em seguida o servidor RMI (Searche Module) chama um método em um objeto do cliente que implementa uma determinada interface (InterfaceClienteServer). A razão pela sua implementação é enviar as estatísticas para o cliente sem que este peça. E para verificar se o cliente ainda está ativo.

- **Downloaders - Index Storage Barrels:** Esta ligação é através de um protocolo de comunicação Multicast.

- **Search Module - Index Storage Barrels:** Tal como pedido, utilizamos RMI, sendo que o Barrel ao conectar-se com a porta, chama um método do SearcheModule(registerClient), enviando-lhe dois parâmetros, o seu identificador e a sua interface para que este através do CallBack possa invocar métodos do cliente. Estes métodos são “Connected()”, método que retorna true se estiver ativo, “ProcuraToken()”, este que vai procurar através da pesquisa todos os urls que contenham os tokens, “SendHash()” este método é invocado quando um outro barrel se liga a meio e este barrel é o que tem mais informação, passando lhe a HashMap que tem as informações que recebeu através do multicast dos Downloaders. E “listPage()” retorna uma String com todos os urls que apontam para o que o utilizador pediu.

- **Downloaders - Queue url:** Para a ligação destes programas utilizamos o RMI. Visto que estávamos mais familiarizados com

este mecanismo de comunicação optamos pela sua utilização, poderíamos ter utilizado o TCP e UDP, mas consideramos que seria mais simples e mais seguro a utilização do RMI. Utilizamos novamente CallBack para esta ligação para que a Queue possa verificar se o Downloader está ativo, sendo importante para atualizarmos a estatística, utilizando a interface (InterfaceDownloaders). Os Downloaders podem chamar métodos da Queue sendo estes “coloca()”, este tal como o nome indica coloca um novo url na Fila, e “retira()”, que retira da fila do url para que este possa analisar. (RMI - Server é a Queue_url, e o RMI - Cliente são os Downloaders)

- **Queue Url - Search Module:** Tal como na alínea de cima, utilizamos o RMI pelas mesmas razões. Utilizamos a Queue_url como o RMI - Server, e o Search Module como RMI - Cliente) visto que o Search Module chama métodos da Queue_url e o Queue_url não chama métodos do Search Module. O search module pode chamar os seguintes métodos: “info()” este que retorna um arraylist de interfaces de Downloaders ativos, “coloca()”, que referimos acima, “atualizaNumeroBarrels()” este que envia para a Queue_url a informação acerca do número de Barrels ativos.

Base de Dados:

Para guardar toda a informação necessária para o funcionamento do programa, decidimos usar uma base de dados, neste caso, uma base de dados postgreSQL, isto porque na nossa opinião, é mais simples e mais fácil de usar, principalmente porque estamos a lidar com grandes quantidades de dados. Para além disso, achamos que nos facilita na obtenção de informações necessárias do que os ficheiros de texto/objetos.

A base de dados usada é muito simples, consiste em 5 tabelas:

- token_url: serve para armazenar as informações de cada token, isto é, o url onde foi encontrado, o contador de pesquisas e o barrel onde está armazenado.
- url_url: guarda o url e o url que aponta para ele.
- url_info: guarda as informações de cada url, isto é, o título e a citação de cada url.
- info_client: guarda os dados de acesso dos clientes.
- Queue_url: guarda os urls que estão na fila para serem processados.

Para além das tabelas, criamos uns indexes para ajudarem na procura de informações:

- CREATE INDEX ON token_url (token1,barrel);
- CREATE INDEX ON url_url (url1);
- CREATE INDEX ON url_url (url2);
- CREATE INDEX ON url_info (url);

Queue_url:

A Queue conecta-se à base de dados.
Utilizamos uma Queue correspondente a ConcurrentLinkedQueue, isto porque é thread safe o que ajuda a reduzir erros e bugs, é mais confiável, e tem maior escalabilidade.

Este após estar conectado, seleciona todos os URLs que ainda não foram analisados pelos Downloaders colocando-os assim na Queue.

Após ter a informação toda na fila, este cria duas ligações RMI:

1. Porta 7005: ligação com os Downloaders.
2. Porta 7003: ligação com o Search Module

Seguindo assim, chama-se um método “Verifica()” que a cada 2 segundos verifica quantos Downloaders estão ativos.
Verifica através de um método para cada clientes “Connected()” sendo que no caso do Downloader/Barrel/Cliente não está ativo este irá detectar uma Exception “RemoteException”, sendo que este irá verificar se tem algum url que lhe foi atribuído e que não foi totalmente executado, no caso de este não ter sido executado irá novamente para a Queue.

NOTA:
A Queue possui 2 ArrayLists:

- DownloadersOnPorta: Os identificadores dos downloaders ativos que possui.
- Downloaders: As interfaces dos downloaders ativos que possui.

Os métodos da Queue retira() e coloca() ambos são “syncronized” para que possa controlar o acesso a essa função.

Search Module:

O Search Module quando executado irá se ligar à Base de Dados.

Este irá se conectar à Queue_url , pelo RMI através da porta 7003 utilizando “LocateRegistry.getRegistry(7003).lookup(“QD”)”.

Após se conectar à Queue_url este cria duas novas ligações RMI:

1. Porta 7001: ligação dos clientes.
2. Porta 1099, com Naming “ServerObject”: ligação dos barrels.

Seguindo assim, chama um método que a cada 2 segundos verifica quantos Downloaders, Barrels e Clientes estão ativos.

Verifica através de um método para cada clientes “Connected()” sendo que no caso do Downloader/Barrel/Cliente não está ativo este irá detectar uma Exception “ConnectException” retirando-o assim do Arraylist correspondente, e chamará os métodos dos clientes com a nova informação acerca das estatísticas.

NOTA:

O Search Module possui 5 ArrayLists:

- Downloads: As interfaces mais recentes que este possui.
- Download2: As interfaces antigas que possui.
- Barrels: As interfaces dos barrels que estão ativos.
- BarrelsID: São os identificadores dos Barrels ativos.
- Clientes: As interfaces dos clientes ativos.

Existe uma ligação entre as duas primeiras interfaces, sendo que o search module tem um método que a cada 2 segundos verifica quantos Downloaders estão ativos, sendo que quando recebe a nova informação compara com a antiga, se for diferente este chama um método do cliente

Downloaders:

O downloader quando executa conecta-se à Base de Dados. Conecta-se à Queue se o identificador já estiver ligado à Queue o programa é encerrado.

Após passar a verificação, este a cada 10 segundos vai verificar se o número de barrels é diferente de 0, isto pelo facto de que só pode analisar um site quando algum barrel estiver ativo para que guarde a informação.

Retira um url e coloca na BD que o url é seu e que ainda não foi executado.

Após processar, envia por multicast para os barrels.

Se a meio, não existir nenhum barrel ativo este volta a colocar na Queue e dá update na BD que não foi executado e que o url já não é seu.

Pode ocorrer uma situação em que o no início podem existir por exemplo 2 barrels, mas depois existem 3 este irá enviar novamente as últimas mensagens de novo para o multicast.

NOTA:

O Downloader possui as seguintes componentes:

- Lista de StopWords.
- ligação ao multicast Server.

Index Storage Barrels:

O index storage barrel quando executa conecta-se à Base de Dados. Conecta-se com o Search module, se o valor retornado for -1, significa que existe um barrel com esse id, então o programa termina.

Senão irá verificar quantos URLs ele processou, com esse número multiplica por 2, visto que a informação guardada nas hash Tables dos outros Barrels é de (id,token),(id2,url). Se o valor for diferente de 0, são executadas duas threads, uma que vai buscar a informação a um outro index storage barrel que está ativo (a pedido do Search module), e outra que entra no multicast.

Se o valor for igual a 0, apenas entra no multicast.

Temos duas situações que temos de ir diretamente à base de dados:

Se o barrel for completamente novo.

Se o barrel não for novo, mas os que estão ativos não têm a informação que este necessita.

Nas outras situações, pedimos ao Search Module que vá a todos os Barrels que estão ativos e verifique que o primeiro identificador da HashMap que recebeu é o mesmo nº de urls que o index storage barrel processou 2 vezes. Se não existir nenhum aí vai buscar diretamente a informação à base de dados.

NOTA:

O index storage barrel possui as seguintes componentes.:

- HashMap com a informação que recebe a partir do Multicast.
- ligação ao multicast cliente.

Cliente

O cliente quando executa irá estabelecer uma ligação por RMI com o search module.

É lhe apresentado algumas opções, login, registrar ou continuar como convidado.

No caso de optar pelo login é lhe pedido um username e uma password, que será verificada na Base de Dados (search module).

No caso de optar por se registrar é lhe pedido um username e uma password, que será verificada na Base de Dados (search module), se não existir é criado com sucesso, e está logado automaticamente.

Um utilizador sem estar logado não tem alguns acessos, como por exemplo poder ver os links que estão a apontar para o url da pesquisa que realizou. Já um utilizador logado tem esse acesso, e como tem poder ver os links que estão associados de um url em específico (funcionalidade acrescida).

NOTA:

O cliente possui as seguintes componentes.:

- ArrayList de Barrels Ativos.
- ArrayList de Downloaders ativos.

Utilizamos estas componentes para apresentar as estatísticas.

Funcionamento da componente Multicast:

Tal como foi pedido no enunciado, implementamos o Multicast para envio da informação dos Downloaders para os Barrels, é uma excelente técnica de comunicação em rede que permite enviar um único pacote de dados para vários destinatários que estejam ligados ao mesmo grupo. É um método bastante mais eficiente, do que enviar uma cópia para cada barrel.

O Downloader conecta-se à ligação do servidor Multicast, enviando a sua informação para uma certa porta, e endereço IP. No nosso caso, o ip é de 224.3.2.1 e porta 4321.

Os Downloaders após receber um url, estes repartem a informação e a partir de um StringBuilder cria a String com a informação acerca dos tokens.

Basicamente, a partir de um link este envia 2 mensagens (todas strings) de uma só vez.

A primeira string que envia corresponde ao tamanho da mensagem que irá enviar mais o tipo de informação. Por exemplo, uma mensagem com 1454 de tamanho, que corresponde aos tokens. Este irá enviar: “1454 | TOKEN”.

A string que irá corresponder a esse tamanho terá como estrutura:

titulo | citacao | url ; token1 ; token2 ; token3 ; token4 ; token5;

Quando o downloader necessitar de enviar uma mensagem acerca de URLs associados este irá enviar novamente duas mensagens.

Uma com tamanho mais uma String acerca do tipo. Usando o exemplo de cima ilustrado: “1454 | URL”

A string que irá corresponder a esse tamanho terá como estrutura:

url ; url1 ; url2 ; url3 ; url4 ; url5; url6; url7;

Para que os Barrels recebam a informação dos Downloaders estes necessitam de criar um socket, e entrar no grupo e a partir desse terá acesso a todas as informações que os Downloaders enviam.
Sendo que estes lêem duas mensagens .

Recebem o pacote com o socket.receive(). Lê a mensagem e verifica se é um “TOKEN” ou um “URL”.

Com isso este irá dividir a informação utilizando split(), trim().

Se a mensagem for TOKEN, vai à posição 0 do array resultante do split de “;” separa o título, a citação e o url com o split de “ | “ .
Após aplicar essa separação, irá colocar na sua base de dados todos os tokens que estão associados a esse url.

Se a mensagem for URL, este igualmente à posição 0 do array resultante do split de “;” e retira o url, e a partir desse irá colocar na tabela correspondente da base de dados os urls todos que estão associados.

Para que não haja perda de dados, por exemplo, se um barrel se desligar a meio da colocação da informação na base de dados, então criamos uma transação e damos commit no final, ou seja, o Barrel coloca a informação toda ou não coloca nenhuma.

Finalizando colocamos a mensagem que recebemos (multicast) e adicionamos ao seu HashMap, incrementando assim o valor do contador.

Esse hashMap poderá vir a ajudar a que outro barrel recupere os dados que perdeu.

Funcionamento da componente RMI:

RMI é um mecanismo de comunicação entre processos em Java que permite que um objeto possa invocar métodos de outro objeto.

Neste projeto implementamos o RMI para comunicação do cliente com o Search Module, o Search Module com Barrels e a Queue com os Downloaders.

Implementamos o callback porque tínhamos situações em que o objeto remoto precisava de notificar o objeto local sobre algum resultado (atividade, estatísticas), e a comunicação deve ser iniciada pelo objeto local, no caso do cliente search module, tal como referimos acima, este necessita de atualizar as estatísticas ao cliente, portanto notifica o objeto local.

Apresentamos a razão da utilização do callback do RMI:

- A sua utilização permitiu-nos fazer os pedidos que necessitamos, tal como o cliente pedir ao Search module a pesquisa (RMI normal) e este pedir a um Barrel aleatório ativo o pedido que o cliente necessita.(callback).

- Search Module com os barrels, este necessitava de chamar métodos do cliente (barrel) para verificar se este estava ativo, fazer pedidos da hash, pedidos de pesquisa etc.

- A Queue com Downloaders necessita de saber se os Downloaders estão ativos, logo também tem o uso de callback para que possa saber se este ainda está a ser executado.

Portanto, para uso de callback, teríamos duas interfaces, uma default do servidor em que o cliente tinha acesso (RMI normal sem callback), e uma no cliente para que o servidor tivesse acesso para atualizar dados, verificar se ainda estava ativo etc.(call back do RMI).

Funcionamento do FailOver:

Para que não haja perda de dados, como por exemplo, URL's que não são processados "desaparecerem", tokens que não são guardados tivemos de implementar certas "seguranças":

Downloaders:

- No caso de este receber um url, e ao processar o url, se este se desligar a meio, o url que tentou processar é reenviado para a Queue. Resolvemos esse problema através da informação na base de dados, o programa Queue ao saber que este não está ativo, através da Exceção "RemoteException", este irá verificar se o Downloader deixou algum url por processar, no caso "executed" ser false, e o barrel ser o id deste próprio. No caso de não ter o url processado, este é enviado de novo para a Queue e para a Base de Dados.

Barrels:

- Após o barrel receber a mensagem por multicast, se este a meio da colocação fosse abaixo, iria perder o resto dos tokens. Portanto para garantir que este não perdia, ou seja, não colocava tudo na base de dados,(não recuperando depois), criamos uma transação e só dá commit no final quando este já tiver as "queries" todas prontas executando-as numa só vez, não perdendo assim nenhum token.

Distribuição de Tarefas:

No início do projeto, o Rui Santos ficou encarregado pela criação dos Downloaders, da Queue. O Bruno Sequeira ficou encarregado pela criação dos clientes, do Search Module e pelos Barrels.

Reunimo-nos e fizemos em conjunto o multicast, pelo facto de acharmos que conseguimos ter melhores ideias e implementações, e podendo ter melhores resultados.

Após o multicast estar funcional, e os Barrels receberem bem as informações que os Downloaders enviavam por multicast, dividimos os pontos do enunciado que nos foi pedido:

Pontos	Descrição	**Responsável**
Ponto 1:	Indexar novo URL	**Bruno Sequeira**
Ponto 2:	Indexar recurs. todos os URLs encontrados	**Rui Santos**
Ponto 3:	Pesquisar páginas que contenham um conjunto de termos	**Rui Santos**
Ponto 4:	Resultados de pesquisa ordenados por importância	**Bruno Sequeira**
Ponto 5:	Consultar lista de páginas com ligação para uma página específica	**Bruno Sequeira**
Ponto 6:	Página de administração atualizada em tempo rea	**Rui Santos/ Bruno Sequeira**

Testes efetuados:

- Client side:

Teste 1	Login
Objetivo	Um utilizador deve conseguir dar login pelo terminal
Resultado obtido	Quando o cliente pede para dar login é lhe solicitado para inserir o username e a password. Após inserir o que é pedido o login é feito com sucesso.
Resultado	Passou

Teste 2	Registo
Objetivo	Deve ser possível registrar um novo utilizador pelo terminal
Resultado obtido	Quando um novo cliente pede para dar registar é-lhe solicitado para inserir o username e a password. Após inserir o que é pedido o registo é feito com sucesso.
Resultado	Passou

Teste 3	Indexação de um novo Url
Objetivo	Deve ser possível um cliente adicionar um novo url para indexar
Resultado obtido	O cliente seleciona a opção para indexar um novo url (2), indica esse url e ele é adicionado a queue com sucesso
Resultado	Passou

Teste 4	Pesquisa de um token 1
Objetivo	Um cliente pode pesquisar todos os sites onde aparece um certo token
Resultado obtido	O cliente seleciona a opção de pesquisar tokens (1) e aparecem todos os sites onde aparece aquele token
Resultado	Passou

Teste 5	Pesquisa de um token 2
Objetivo	Se um cliente inserir um token que ainda não foi encontrado, aparece uma mensagem a dizer que não há informação sobre esse token.
Resultado obtido	O cliente seleciona a opção de pesquisar tokens (1) e aparece a mensagem “Sem resultados encontrados!!”
Resultado	Passou

Teste 6	Pesquisa de um token 3
Objetivo	Se um cliente inserir um token para pesquisa e não houver barrels disponíveis recebe uma mensagem a informar disso
Resultado obtido	O cliente seleciona a opção de pesquisar tokens (1) e aparece a mensagem”Barrels Indisponíveis!!”
Resultado	Passou

Teste 7	Lista de páginas com ligação para uma pagina 1
Objetivo	Um cliente logado pode pedir para listar as páginas com ligação a uma página que ele indicar
Resultado obtido	O cliente seleciona a opção de listagem de páginas (4) e aparece a lista de links que apontam para o link indicado, se não houver nenhum aparece a mensagem ”Sem resultados”.
Resultado	Passou

Teste 8	Lista de páginas com ligação para uma pagina 2
Objetivo	Para qualquer link resultante da pesquisa de um token, deve ser possível a um cliente logado listar as páginas que apontam para ele.
Resultado obtido	O cliente faz a pesquisa de um token e aparece a listagem dos links numerados. Depois o cliente pode colocar o número do link que quer que seja mostrada a listagem e esta aparece.
Resultado	Passou

Teste 9	Lista de páginas com ligação para uma pagina 3
Objetivo	Um cliente não logado não pode pedir para listar as páginas com ligação a uma página que ele indicar
Resultado obtido	Um cliente não logado não consegue visualizar a opção de listagem de páginas com ligação para outra.
Resultado	Passou

Teste 10	Lista de páginas com ligação para uma pagina 4
Objetivo	Para qualquer link resultante da pesquisa de um token, não deve ser possível a um cliente não logado listar as páginas que apontam para ele.
Resultado obtido	O cliente faz a pesquisa de um token e aparece a listagem dos links numerados. Depois o cliente mesmo que indique o número do link não consegue ver a listagem dos links que apontam para ele.
Resultado	Passou

Teste 11	Consulta das estatísticas do servidor
Objetivo	Um cliente pode pedir para consultar as estatísticas do servidor
Resultado obtido	O cliente seleciona a opção Estatísticas (3) e aparecem o número de downloaders e barrels com os seus ip's e portas. Para além disso aparece também o top 10 de tokens mais pesquisados
Resultado	Passou

- Server side:

Teste 12	Mandar um barrel abaixo e reiniciá-lo
Objetivo	Quanto um barrel vai abaixo e é, mais tarde, reiniciado, deve ser atualizado com a informação perdeu enquanto foi abaixo.
Resultado obtido	O barrel recebe com sucesso a informação perdida enquanto foi abaixo
Resultado	Passou

Teste 13	O programa vai abaixo - queue
Objetivo	Nesta situação, a queue deve conseguir ir buscar os urls que faltavam processar, quando o programa voltar ao ativo.
Resultado obtido	Quando o programa volta, a queue consegue com sucesso ir à base de dados buscar os urls por processar.
Resultado	Passou

Teste 14	Barrel entra a meio da mensagem multicast
Objetivo	Um barrel, mesmo que entre a meio da mensagem multicast dos downloaders para os barrels, deve receber tudo o que ficou para trás
Resultado obtido	Quando o barrel entra a meio da mensagem multicast, recebe tudo o que ficou para trás.
Resultado	Passou

Teste 15	Barrels vão abaixo -> Comportamento dos downloaders
Objetivo	Se durante o funcionamento do programa, os barrels forem todos abaixo, os downloaders devem parar e aguardar pela chegada de um barrel
Resultado obtido	Quando os barrels vão abaixo os downloaders aguardam a chegada de barrels para continuarem a trabalhar.
Resultado	Passou

Teste 16	Entrada de um barrel novo(sem info nenhuma associada a ele)
Objetivo	Se entrar um barrel novo, ele deve ir buscar à bd toda a informação que os outros barrels tinham
Resultado obtido	Quando o barrel novo entra, vai à bd buscar a informação que precisa.
Resultado	Passou

Teste 17	Restart de um barrel
Objetivo	Se um barrel for abaixo e reiniciar e se, ou é o unico ativo, ou os restantes barrels ativos não têm toda a informação que precisa nas hashmaps deles, vai à bd buscar o que precisa.
Resultado obtido	Quando o barrel entra, vai à bd buscar a informação que precisa.
Resultado	Passou

Teste 18	Barrel vai abaixo a meio de pedido de cliente
Objetivo	Se um barrel vai abaixo enquanto está a responder a um pedido de um cliente, esse pedido deve ser transferido para outro barrel ativo.
Resultado obtido	Quando um barrel vai abaixo, o pedido do cliente é transferido para outro cliente.
Resultado	Passou

Teste 19	Verificar a entrada de um barrel
Objetivo	Quando queremos adicionar um barrel, temos de lhe fornecer um id, por isso, temos de garantir que não há dois barrels ativos com o mesmo id.
Resultado obtido	Quando é adicionado um barrel, se tiver um id igual a um dos barrels já existentes ativos é lhe negada a entrada.
Resultado	Passou

Teste 20	Verificar a entrada de um downloader
Objetivo	Quando queremos adicionar um downloader, temos de lhe fornecer um id, por isso, temos de garantir que não há dois downloaders ativos com o mesmo id.
Resultado obtido	Quando é adicionado um downloader, se tiver um id igual a um dos downloader já existentes ativos é lhe negada a entrada.
Resultado	Passou

Conclusão:

Após realização da meta 1, ambos tivemos a mesma opinião, o projeto é bastante interessante e engraçado, visto que é tem uma ligação ao mundo real, como por exemplo a google, uma aplicação que usamos no nosso dia-a-dia, e ver que conseguimos implementar uma versão dessa plataforma fez com que tivéssemos mais vontade.

Tivemos dificuldade nas novas tecnologias, por exemplo, a comunicação por RMI era completamente nova, então tivemos de pesquisar as funcionalidades que este tem para que pudéssemos implementar de forma correta. Tivemos alguma dificuldade, pelo facto de que ao longo do projeto os barrels não possuíam a mesma informação entre eles, apesar de já estar resolvido.

Ao nível da carga de trabalho, como foi bem dividido em termos de horas foram o expectável tivemos cerca de 30h cada elemento.

Concluindo, no nosso ponto de vista, este projeto permitiu-nos que as nossas capacidades, em criar uma camada de acesso aos dados usando Java RMI, usar sockets Multicast, programar um sistema pesquisa de páginas Web com uma arquitetura cliente-servidor, seguir um modelo multi thread para desenhar os servidores, e garantir a disponibilidade da aplicação aumentaram.

Biografia:

https://www.facebook.com/DEI.UC/
https://gist.github.com/sebleier/554280
https://gist.github.com/alopes/5358189
https://www.uc.pt/identidadevisual/Marcas_UC_submarcas
Materiais da Cadeira.