FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

departamento de engenharia informática

Sistemas Distribuídos

GOOGOL: MOTOR DE PESQUISA DE PÁGINAS WEB

(META 2)

2022/2023

Trabalho realizado por:

- **Tomás Bernardo Martins Dias nº 2020215701**
- **Henrique Miguel da Silva Costa nº 2020214120**

Índice

Introdução

Na primeira meta deste trabalho prático, foi pedido que fosse desenvolvido um motor de busca de páginas Web, com um conjunto de funcionalidades semelhantes aos serviços google.com, bing.com. Este motor de busca teria de possuir indexação automática (Web Crawler) e busca (Search Engine).

Na segunda meta, foi requerido que fosse desenvolvido um frontend Web para a aplicação Googol, e para isso foi utilizada a framework Spring boot, e a ferramenta Thymeleaf.

Ao longo deste relatório, vamos explicar detalhadamente a arquitetura de software, o funcionamento do servidor, os testes que realizamos à plataforma e o que implementamos para assegurar fiabilidade e consistência da nossa aplicação.

Arquitetura de software

No que toca à arquitetura num geral, a mesma se encontra praticamente semelhante à da meta1 sendo apenas adicionadas algumas componentes relativas à segunda meta (Figura 1).

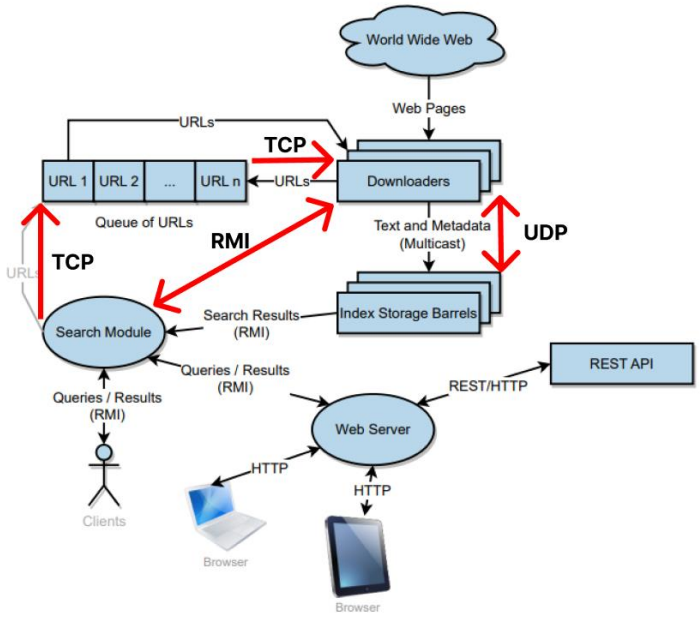

Figura 1 - Arquitetura do Software

O software, é composto por 5 controllers (ControllerHackerNews, GreetingController, IndexController, SearchController, ApontadosController), um Bean (WebServerRMI), as resources com as templates das webpages.

As páginas HTML estão na pasta 'resources/templates' onde estão presentes as templates referentes à homepage, resultados da pesquisa, indexação de urls, links apontados, HackerNews homepage, indexação por utilizador ou palavras HackerNews.

O WebServerRMI e o ControllerHackerNews serão discutidos em tópicos mais à frente. Mas numa visão mais geral o WebServerRMI contem a ligação RMI ao servidor RMI (SearchModule), como também os métodos responsáveis pela resposta aos pedidos do utilizador (Figura 2).

```java
private ServerInterface w;

public WebServerRMI() throws RemoteException {
    connection();
}

public void connection() throws RemoteException {
    try {
        this.w = (ServerInterface) LocateRegistry.getRegistry( port: 4000).lookup( name: "WEBSERVER");
    } catch (NotBoundException e) {
        e.printStackTrace();
    }
}
```

Figura 2 - Contrutor + função connection() que possibilita a invocação dos métodos.

O ControllerHackerNews, como o nome sugere, é responsável por todas as páginas que estão ligadas a REST API HackerNews.

O controller GreetingController é apenas responsável por mapear a página de homepage. No método redirect() existe um mapeamento com a requisição do método GET lida com as requisições no URL raiz (/) e as rediriciona para a homepage. Para alem deste método existe o método homepage que faz o mapeamento com a requisição do método GET e devolve a homepage.

Os controllers SearchController, IndexController, ApontadosController tem todos as suas classes com o atributo WebServerRMI (Figura 3), sendo este o atributo utilizado para comunicar com o servidor RMI.

```java
private WebServerRMI server;

public SearchController(WebServerRMI webserver){
    this.server = webserver;
}
```

Figura 3- Contrutor do SearchController.

O controller SearchController é responsável pelas páginas das pesquisas. No método search() existe um mapeamento com a requisição do método GET, quando se pressiona o botão da homepage que redireciona para o "/search". Esta função tem como parâmetros requisitados a pesquisa, ou seja, o input que o cliente inseriu, e as páginas, sendo o default value a primeira página a ser visualizada por isso, o seu valor é igual a 1.

Neste método são invocadas as funções do WebServer num_paginas() que recebe como parâmetro o input e a função getLinks() que recebe como parâmetro tanto o input com o número de páginas retornado pela função anterior. Este último método é o que retorna os resultados da pesquisa. No final, são passados ao model todos os atributos necessários para a visualização da página.

O controller IndexController é responsável pela página de indexação de URL´s. No método indexarUrl, existe um mapeamento com requisição do método GET, para fazer o mapeamento da página de indexação. No método indexarUrlPost é utilizada a requisição do método POST, uma vez que é necessário enviar os dados inseridos para o SearchModule, sendo o seu mapeamento feito para a página anteriormente referida.

Este apresenta apenas um parâmetro requerido, o input inserido pelo utilizador, ou seja, URL a ser indexado. Em caso de sucesso o utilizador recebe uma mensagem a informar dá indexação.

O controller ApontadosController, é responsável pela página de link que apontam para a página pretendida. No método getApont(), existe um mapeamento com requisição do método GET que é acionado quando é pressionado o botão da pagina links que redireciona para o "/linksapt". Este método tem com parâmetro requerido, a pesquisa por links indexados que o cliente deseja efetuar.

Neste método é invocada a função do WebServer getlinksAp() que recebe como parâmetro o input inserido pelo utilizador e retorna a lista de links apontados. No final, são passados ao model todos os atributos necessários para a visualização da página.

Integração Spring Boot com o servidor RMI

Para fazer a integração entre o servidor RMI e o Spring Boot foi criado um Bean, que cria e retorna uma instância da classe WebServerRMI, que será utilizada nos controllers para a comunicação com o servidor.

Esta classe apresenta como atributo a interface ServerInterface (a interface do SearchModule). O seu construtor chama o método connect(), responsável por estabelecer a conexão entre o WebServerRMI e o SearchModule via RMI, no objeto remoto "WEBSERVER" e porto 4000. Isto viabiliza a conexão entre ambos e a invocação de métodos da interface do servidor (do SearchModule) por parte do WebServerRMI.

Esta classe apresenta ainda os métodos getlinksAp() que recebe como parâmetro uma String, o URL para o qual queremos a lista de links que apontam para ele.

O metodo getlinks() que recebe como parâmetros uma String, os termos da pesquisa, e recebe também como String o numero de paginas.

O método num_paginas() auxilia as pesquisas e recebe como parâmetro uma String, o termos da pesquisa feita e retorna o numero de paginas que essa pesquisa irá possuir.

Por ultimo temos o metodo index() que recebe como parâmetro uma String, o url a ser indexado na Queue do Servidor.

Todos estes métodos usam o método que o cliente na meta 1 usava para comunicar com o servidor. Este método sofreu algumas alterações, passando a retornar um ArrayList de Strings em vez de uma String. Também a forma de processar informação teve de ser alterada e otimizada.

Integração Spring Boot com o serviço REST

Para realizar a integração do Spring Boot com o serviço Rest, de início, foi necessário adicionar algumas dependências ao ficheiro “pom.xml”.

O controller HackerNewsController, através dos métodos HackerNews() e HackerNewsUser(), quando o cliente clica nos botões disponíveis consoante a ação desejada, redirecionando-o para a página em que pode efetuar as suas indexações do HackerNews, indexar por palavras, indexar por utilizador, respetivamente.

No método getHackerNews(), utiliza-se a requisição Get uma vez que é pretendido buscar dados à API e o mapeamento é feito para "/indexarHackerNews". É efetuada uma conexão (Figura 4) às topstories com o método Get (buscar informação), obtendo-se informação que posteriormente necessita de ser convertida num JSONArray.

```
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");
connection.setDoOutput(true);
connection.setInstanceFollowRedirects(false);
connection.setRequestProperty("Accept", "application/json");
connection.setRequestProperty("User-agent", "Internet Explorer");
```

Figura 4 - Conexão à API.

Com este Array, que contem os id's das stories, é de novo efetuada uma ligação sendo que o url agora é direto à user storie, recorrendo-se ao mesmo filtro de informação para se obter, neste caso JSONObject's que possui o Objeto de cada user storie.

Com isto, verifica-se se o objeto possui texto e url, pega-se no texto e dá-se split para comparar palavra a palavra com o input do cliente e verificar se é de interesse indexar determinado url. Caso seja afirmativo, envia-se o Url coma função index().

Também, apenas se realiza a pesquisa até ao Top100 uma vez que, se fosse para todas user stories, tornar-se-ia computacionalmente muito mais pesado (Figura 5).

```
String id;
String url_start = "https://hacker-news.firebaseio.com/v0/item/";
String url_end = ".json?print=pretty";
for (int i = 0; i< 101; i++){
    id = json.get(i).toString();

    String url_full = url_start + id + url_end;
    connection = (HttpURLConnection) new URL(url_full).openConnection();
```

Figura 5 - Apenas verifica a informação nas primeiras 100 user stories.

No método getHackerNewsUser(), utiliza-se a requisição Get da mesma forma referida anteriormente, e o mapeamento é feito para "/indexarHackerNewsUser", sendo que a estrutura do código é semelhante à função anterior, deferindo em poucos aspetos.

Nesta, a ligação inicial é realizada ao username que o cliente inseriu, sendo que se este não existir, obtém-se um null, possibilitando informar o cliente que o username que ele inseriu não se encontra na API.

Caso o username exista, pega-se nas user stories do mesmo (parâmetro: "submitted" do objeto), separando os id's e realizando outra conexão para obter os dados de tais sories e as indexar, caso objetos contenham url.

Testes Realizados

Testes da página "/homepage"

	Cliente acede ao website
Resultado Esperado	O cliente acede ao website e é lhe apresentada a homepage
Resultado Obtido	O cliente acede ao website e é lhe apresentada a homepage
Conclusão	Passou

	Cliente clica no botão "Pesquisa Googol" sem nenhum input
Resultado Esperado	Redirecionamento para página de resultados da pesquisa e está vazia.
Resultado Obtido	Entrou na página e encontrava-se vazia.
Conclusão	Passou

	Cliente clica no botão "Pesquisa Googol" com input
Resultado Esperado	É redirecionado para a página de resultados de pesquisa e tem os resultados relativos à pesquisa.
Resultado Obtido	Entrou na página e obteve os resultados da pesquisa.
Conclusão	Passou

	Cliente clica em "Indexar Url"
Resultado Esperado	Cliente é redirecionado para a página em que pode indexar url
Resultado Obtido	Redirecionamento efetuado com sucesso
Conclusão	Passou

	Cliente clica no botão "HackerNews"
Resultado Esperado	Cliente redirecionado para a página de operações HackerNews
Resultado Obtido	Cliente está na página de operações HackerNews
Conclusão	Passou

	Cliente clica em "Links apontados"
Resultado Esperado	Cliente é redirecionado para a página em que pode submeter um URL e ver os links que apontam para este.
Resultado Obtido	Redirecionamento efetuado com sucesso
Conclusão	Passou

	Cliente clica no botão "Login"
Resultado Esperado	Cliente é redirecionado para a página de login
Resultado Obtido	Cliente conseguiu aceder à página de login
Conclusão	Passou

	Cliente clica no botão "Login"
Resultado Esperado	Cliente é redirecionado para a página de login
Resultado Obtido	Cliente conseguiu aceder à página de login
Conclusão	Passou

Testes da página "/search"

	Cliente pode clicar no botão para escolher a página de resultados
Resultado Esperado	Consegue navegar nas páginas e ver os links que possuem.
Resultado Obtido	Consegue navegar nas páginas e ver os links.
Conclusão	Passou

	Cliente consegue navegar nas páginas (número páginas) sem perder dados.
Resultado Esperado	Consegue navegar nas páginas para frente e trás com os dados consistentes.
Resultado Obtido	Consegue navegar para a frente e para trás e ver links sem perder dados e estão consistentes.
Conclusão	Passou

	Cliente quer sair da página de pesquisa
Resultado Esperado	Cliente clica em "Voltar" e volta para a página inicial
Resultado Obtido	Redirecionamento efetuado com sucesso
Conclusão	Passou

Testes da página "/indexarUrl"

	Cliente insere um URL e indexa-o
Resultado Esperado	Cliente consegue indexar URL e recebe mensagem a dizer se foi sucedido
Resultado Obtido	Cliente indexa e recebe mensagem.
Conclusão	Passou

Testes da página "/HackerNews"

	Cliente clica no botão "User HackerNews"
Resultado Esperado	Cliente redirecionado para a página de indexar por username
Resultado Obtido	Cliente está na página de indexar por username
Conclusão	Passou

	Cliente clica no botão "Palavras HackerNews"
Resultado Esperado	Cliente redirecionado para a página de indexar por palavras
Resultado Obtido	Cliente está na página de indexar por palavras
Conclusão	Passou

Testes da página "/HackerNewsUser"

	Cliente clica no botão "Index HackerNews"
Resultado Esperado	Cliente consegue indexar url e recebe mensagem a dizer se foi sucedido ou não
Resultado Obtido	Cliente indexa e recebe mensagem.
Conclusão	Passou

Testes da página "/HackerNewsSearch"

	Cliente clica no botão "Search HackerNews"
Resultado Esperado	Cliente consegue indexar url e recebe mensagem a dizer se foi sucedido ou não
Resultado Obtido	Cliente indexa e recebe mensagem.
Conclusão	Passou

Testes da página “/links”

	Cliente clica no botão "Pesquisar Ligações” com um input
Resultado Esperado	Cliente na página redirecionado, obtém os links apontados pelo que submeteu
Resultado Obtido	Cliente conseguiu obter os links na página
Conclusão	Passou

Conclusão

No início deste projeto na meta 1 foi pedido que fosse desenvolvido um motor de busca garantindo a aplicação através de redundância com FailOver.

Ao longo da meta 2 foram finalizadas as funcionalidades e mecanismo de Failover, como os resultados das pesquisas agrupados 10 em 10, para cada página ligações conhecidas que apontam para essa página, as 10 pesquisas mais comuns realizadas pelos utilizadores, recuperação dos dados do Barrel.

Para alem da finalização da meta 1 foram cumpridos alguns objetivos como a integração da REST API HackerNews e do Servidor RMI (SearchModule).

Em suma tendo em conta os resultados dos testes apresentados acima podemos reforçar afirmar que no geral o resultado foi positivo tendo maioria dos testes passado com distinção.

Referências

- Material fornecido pelos docentes
- https://www.baeldung.com/spring-remoting-rmi
- https://spring.io/guides/gs/serving-web-content/
- https://getbootstrap.com/
- https://www.w3schools.com/js/