



UNIVERSIDADE D  
**COIMBRA**

# **Googol: Motor de pesquisa de páginas Web**

Sistemas Distribuídos - Meta 2


Licenciatura em Engenharia Informática  
2022/2023

30 de março de 2024

# Autores


**João Moreira**

 joaomoreira@student.dei.uc.pt

 2020230563

**Tomás Pinto**

 tomaspinto@student.dei.uc.pt

 2020224069

# Índice

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Arquitetura</b>	<b>3</b>
2.1	Models . . . . .	4
2.1.1	Search . . . . .	4
2.1.2	UrlModel . . . . .	4
2.1.3	Loginp . . . . .	4
2.2	HackerNewsItemRecord . . . . .	4
2.3	HackerNewsUserRecord . . . . .	4
2.4	Views . . . . .	4
2.4.1	Pesquisa por Termos . . . . .	4
2.4.2	Resultados da Pesquisa por Termos . . . . .	5
2.4.3	Consultar estatísticas . . . . .	6
2.4.4	Registo . . . . .	6
2.4.5	Resultado do Registo . . . . .	7
2.4.6	Login . . . . .	8
2.4.7	Logout . . . . .	8
2.4.8	Pointed Links . . . . .	9
2.4.9	Resultados do Pointed Links . . . . .	9
2.4.10	Indexação das top 10 stories do Hacker News . . . . .	10
2.4.11	Indexação das stories de um utilizador do Hacker News . . . . .	11
2.5	Controllors . . . . .	12
<b>3</b>	<b>Integração SpringBoot - Server RMI</b>	<b>13</b>
<b>4</b>	<b>WebSockets</b>	<b>13</b>
<b>5</b>	<b>Serviço REST</b>	<b>13</b>
<b>6</b>	<b>Testes</b>	<b>15</b>

# 1. Introdução

Para a meta 2 do projeto foi nos pedido que criássemos um *frontend* para a nossa aplicação *Googol*, que permita o acesso à plataforma de qualquer dispositivo de uma forma mais fácil. Para alcançar esse objetivo, será utilizado o servidor *RMI* desenvolvido na primeira meta do projeto. Dessa forma, os utilizadores da aplicação web terão acesso as mesmas funcionalidades disponíveis anteriormente, como indexação de *URLs* e pesquisa de páginas.

Na implementação do *frontend* da nossa aplicação, foram usadas diversas tecnologias, tais como: *SpringBoot*, *API Rest* do *Hacker News* e *Web Sockets*.

# 2. Arquitetura

Em termos de Arquitetura, são adicionados alguns componentes aos já desenvolvidos na primeira meta.

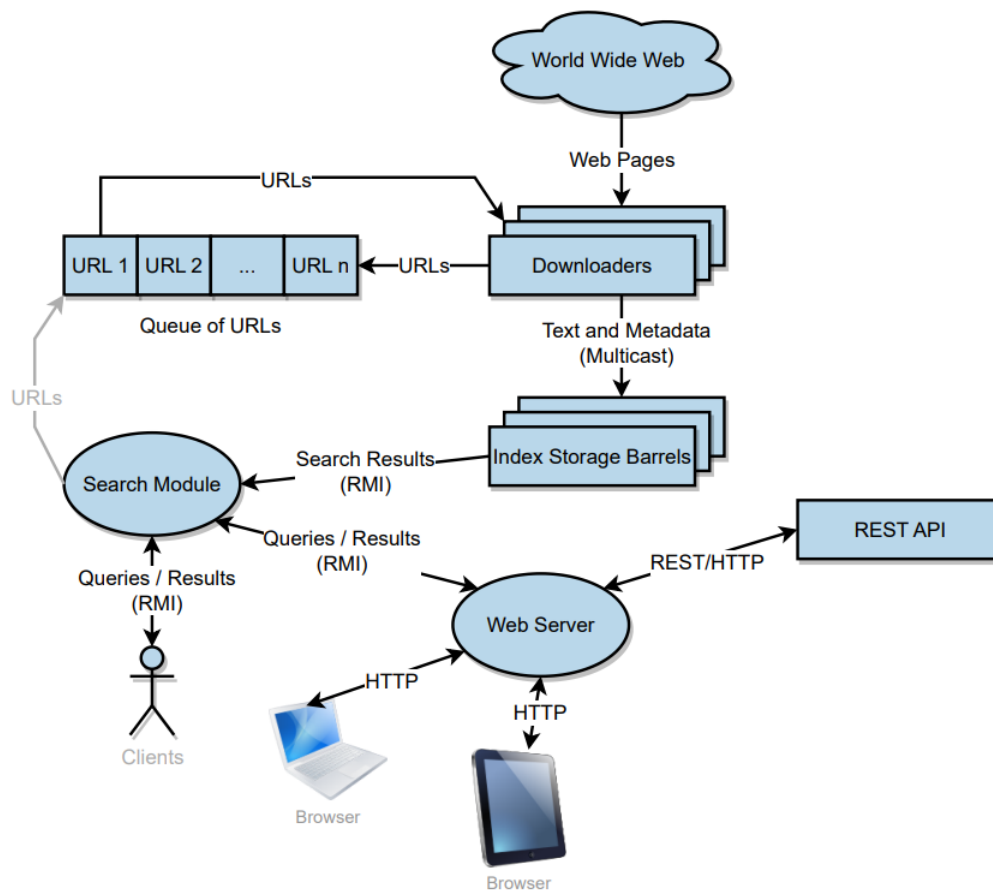


Figura 1: Estrutura do projeto

Um dos novos elementos é um servidor *Web*, que se irá conectar via *RMI* como cliente ao servidor *Search Module* já desenvolvido, substituindo/incorporando as funções dos clientes utilizados anteriormente.

Os utilizadores da aplicação Web irão se conectar ao servidor HTTP através de *web browsers* nos seus dispositivos.

É feita a Integração com um serviço REST, neste caso a utilização da API do Hacker News, que visa desempenhar duas funcionalidades, sendo elas, a indexação das top 10 stories da plataforma Hacker News e a indexação de todas as stories de um dado utilizador registado na mesma.

## 2.1. Models

### 2.1.1. Search

O model Search é responsável por representar uma pesquisa realizada na aplicação. Ele possui uma propriedade principal chamada search, que é uma string que armazena o termo de pesquisa fornecido pelo utilizador.

### 2.1.2. UrlModel

O model UrlModel é responsável por representar um URL de qualquer recurso da aplicação. Possui três propriedades principais: url, title e paragraph, que armazenam respectivamente a URL, o título e o parágrafo relativos à página associada com tal url. Esta classe é importante na representação das páginas apresentadas nas funcionalidade de Pointed links e pesquisa de paginas por termos. O objeto passado na comunicação via rmi do servidor rmi para o servidor web é proveniente desta classe.

### 2.1.3. Loginp

O model Loginp é responsável por representar as informações de registo de um utilizador na aplicação. Ele possui duas propriedades principais: username e password, que são strings que armazenam o nome de utilizador e a password do mesmo, respetivamente.

## 2.2. HackerNewsItemRecord

O modelo HackerNewsItemRecord é utilizado para representar uma story do Hacker News. Contém várias propriedades que armazenam informações sobre a story, como o seu ID, tipo, autor, data de criação, texto, URL, pontuação, título e outros atributos relevantes. O atributo URL será o mais importante uma vez que temos como objetivo extrair os URLs das stories para fazer indexação.

## 2.3. HackerNewsUserRecord

O modelo HackerNewsUserRecord é usado para representar um user do Hacker News. Contém várias propriedades que armazenam informações sobre o user, como a data de criação da conta, o ID do user, o karma (pontuação) do user e uma lista de itens submetidos pelo mesmo, sendo este ultimo atributo o mais importante, para obtenção da lista de stories para a função de indexação.

## 2.4. Views

### 2.4.1. Pesquisa por Termos

O Gogool permite ao utilizador fazer uma pesquisa. Para isso, basta aceder à página principal

*[http : //gogool.com : 8080/search](http://gogool.com:8080/search)*

onde terá a oportunidade de colocar os seus termos de pesquisa, ou então partir para outras possibilidades da nossa aplicação, como pode ser visível na imagem a baixo.

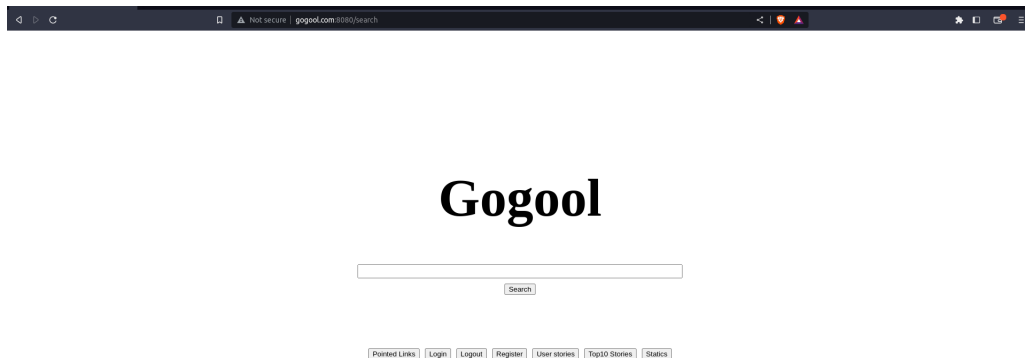


Figura 2: Estrutura do projeto

Esta vista simplesmente guarda as Hiperligações para as outras atividades no ficheiro *HTML* correspondente, cujos os pedidos serão invocados nos métodos do controlador.

## 2.4.2. Resultados da Pesquisa por Termos

Se na primeira vista o utilizador decidir fazer uma pesquisa, via *Spring boot* o utilizador será redirecionado para o *endpoint*

*/search – result*

e terá um resultado semelhante ao que se encontra na figura a baixo. Uma pesquisa com um resultado interessante terá sempre vários *URLs* com os seus respetivos títulos e o seu primeiro paragrafo.

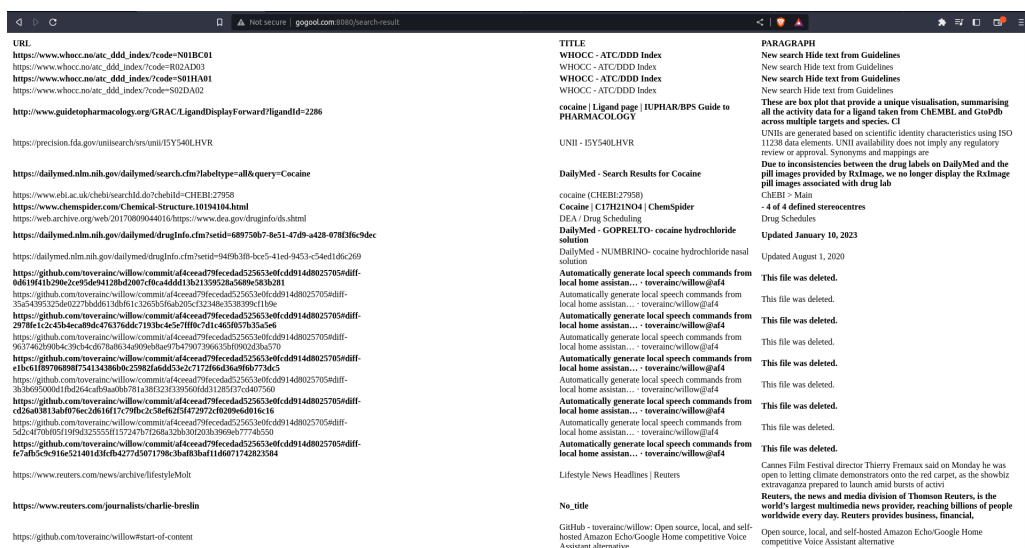


Figura 3: Estrutura do projeto

No final da página existe uma hiperligação para a página principal.

Na vista anterior, obtemos a pesquisa do utilizador através do *Spring boot* e neste *endpoint* enviamos a pesquisa ao *SearchModel* por via *RMI*, cujo devolve-nos uma lista com todos os *URLs*, onde por fim listamos os resultados.

### 2.4.3. Consultar estatísticas

Pode-se acessar às estatísticas pelo *endpoint*

/

ou então pelo botão *Statics* presente na página principal. Como neste *endpoint* é estabelecida uma ligação via *web wocket* a cada cinco segundos iremos receber as novas atualizações sem ter a necessidade de fazer *refresh* à página.

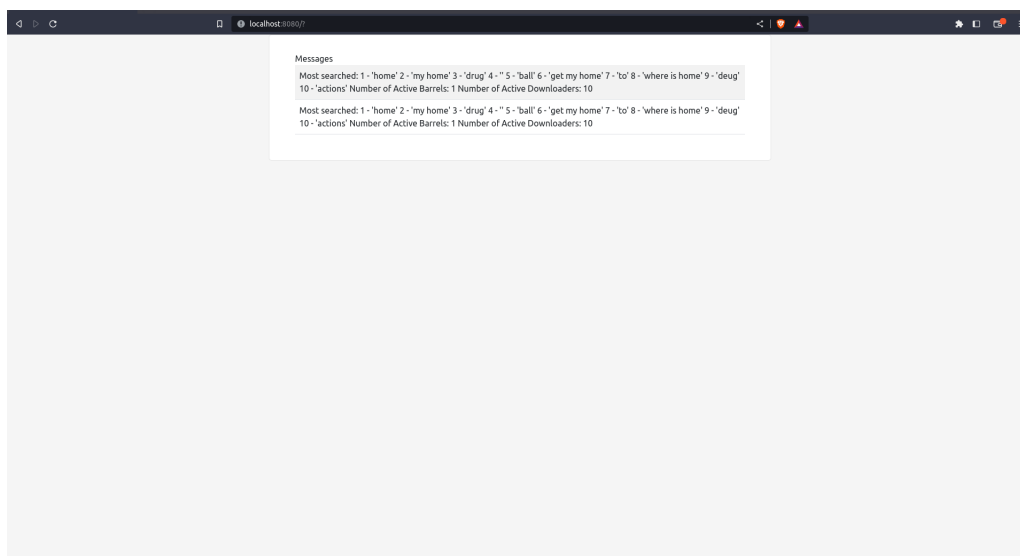


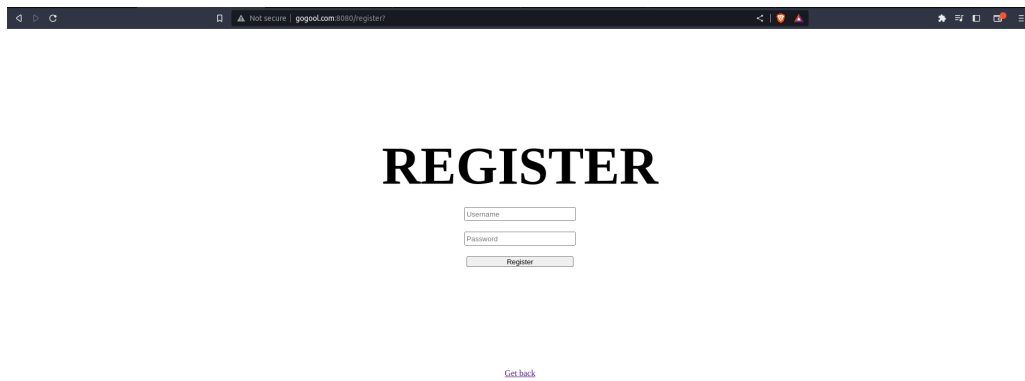
Figura 4: Estrutura do projeto

### 2.4.4. Registo

Para o utilizador registar-se basta dirigir-se à página principal e clicar no botão: *Register* e será redirecionado para a página com o *endpoint*

/register

.



REGISTER

Username

Password

Register

[Get back](#)

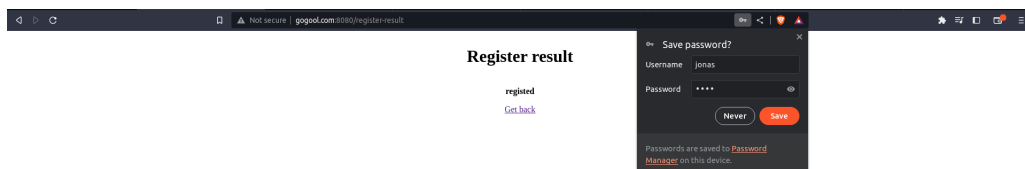
Figura 5: Estrutura do projeto

Esta vista é muito simples e serve simplesmente como *forms* para a próxima vista: Resultado do Registo.

### 2.4.5. Resultado do Registo

O resultado do registo é feito no *endpoint*

*/register – result*



Register result

registered

[Get back](#)

Save password?

Username Jonas

Password \*\*\*\*

Never Save

Passwords are saved to Password Manager on this device.

Figura 6

Aqui limitamos-nos a guardar toda a informação recebida da vista anterior em disco. Os dados ficam armazenados em disco por parte do servidor *web* (em ficheiros de texto).



### 2.4.6. Login

Para o utilizador fazer o *login* basta dirigir-se à página principal e clicar no botão: *Login* e será redirecionado para a página com o *endpoint*

*/login*

.

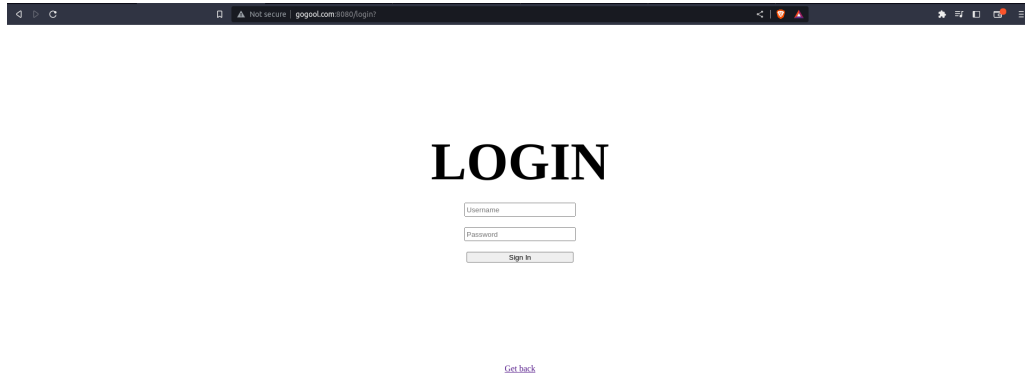


Figura 7

Usufruímos do *Spring Security Web* para efetuar o registo. Quando o servidor arranca, ele lê do ficheiro de texto (*users.txt*) todas as credencias e define as *roles* dos utilizadores e os *endpoints* que cada um tem acesso. No nosso caso, todos os utilizadores têm as mesmas permissões. Para aceder à página de pesquisa de *URLs* apontados, os utilizadores têm de ter realizado o *login*.

### 2.4.7. Logout

Não existe propriamente um *endpoint* para o *logout*, pois contamos com a ajuda do *Spring Security Web* para isso e simplesmente deixamos uma hiperligação

*/logout*

na página principal, cuja trata de tudo por defeito.

Outra opção para o *logout* é desligar e voltar a ligar o *web-browser*.

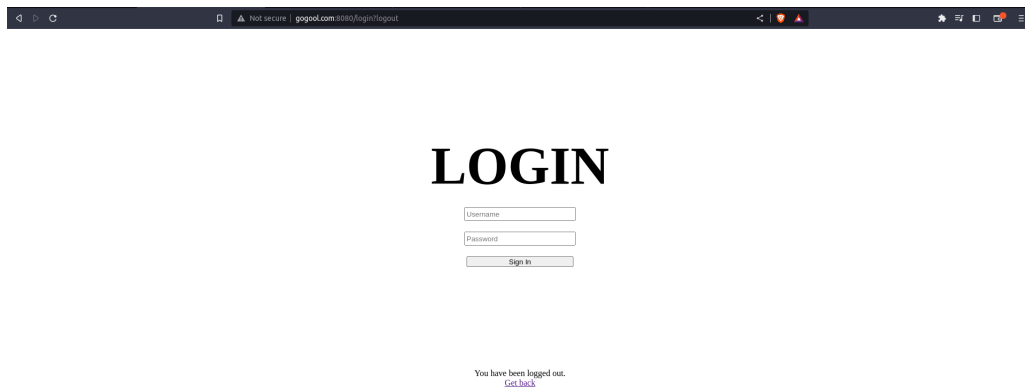


Figura 8

Quando o utilizador efetua o *logout* ele é redirecionado para a página de *login*.

### 2.4.8. Pointed Links

Para obter *Pointed Links* basta aceder através do *endpoint*

*/pointed – links*

ou então ir pela hiperligação disponível na página principal.

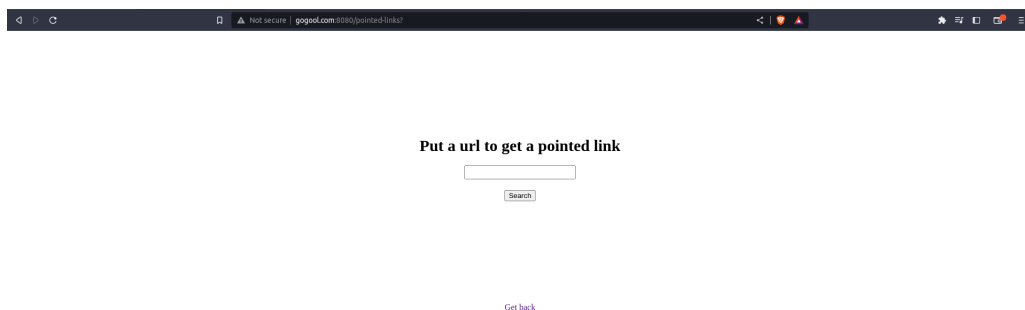


Figura 9

Como para aceder a este *endpoint* é requisito ter o *login* efetuado, no caso de o utilizador não ter o *login* realizado, será redirecionado para a página de *login*.

### 2.4.9. Resultados do Pointed Links

Depois de o utilizador submeter um *URL* será redirecionado para o *endpoint*

*/pointed – links – result*

e terá um resultado como o que se segue, uma lista dos URLs que apontam para o introduzido.



Figura 10

#### 2.4.10. Indexação das top 10 stories do Hacker News

Para proceder À indexação das top 10 stories do Hacker News basta aceder através do *endpoint*

*/list*

ou então ir pelo botão "Top10 Stories" disponível na página principal.

### Top 10 Hacker News Stories

#### URL

<https://boulderbeat.news/2023/05/12/controlled-burn-rules/>  
<https://worldsensorium.com/the-regenerating-power-of-big-basins-redwoods/>  
<https://retrocomputermuseum.co.uk/>  
<https://med.stanford.edu/news/all-news/2023/05/depression-reverse-brain-signals.html>  
<https://smyachenkov.com/posts/book-review-the-staff-engineers-path/>  
<https://www.aidemos.info/summarize-any-article-as-a-dialog-between-two-people-with-bing-chat/>  
<https://web.archive.org/web/20080220175358/http://www.andrew-turnbull.net/tech/windows95.html>  
<https://github.com/madprops/blog/blob/main/docs/timers.md>  
<https://kotaku.com/mtg-aftermath-leaks-pinkertons-wotc-magic-the-gathering-1850368923>  
<https://rekt.network>  
[Home](#)

Figura 11

A página lista os 10 urls que irão ser indexados.

### 2.4.11. Indexação das stories de um utilizador do Hacker News

Para proceder à indexação das stories de um utilizador registado na plataforma do Hacker News basta aceder através do *endpoint*

*/user*

ou então ir pelo botão "User Stories" disponível na página principal.

## Enter HackerNews Username



Username:

Submit

[Home](#)

Figura 12

Após a introdução do username do utilizador, vai ser feito redirecionamento para o *endpoint*

*/user – urls*

onde serão listados todos os urls das paginas que irão ser indexadas, correspondentes as stories do utilizador.

## User's Submitted Stories

**URL**

<https://gubern.net/search>  
<https://www.cs.dartmouth.edu/~doug/tinysort.html>  
<https://ultimateelectronicsbook.com/>  
[Home](#)

Figura 13

## 2.5. Controllers

O nosso controlador é contruido na class *DemoControler* com a anotação *@Controler*.

No controler fica o mapeamento de muitos dos *endpoints* criados nesta fase do projeto, cujos servem-se de algumas anotações: *@GetMapping*, *@PostMapping* e *@Schedule*. Segue a seguinte tabela com todos os *endpoint* e as suas respectivas anotações.

<i>Endpoint</i>	<i>Anotação</i>
/	@Schedule
/search	@GetMapping
/search-result	@PostMapping
/pointed-links	@GetMapping
/pointed-links-result	@PostMapping
/login	@GetMapping
/register	@GetMapping
/register-result	@PostMapping
/user	@GetMapping
/user-urls	@PostMapping
/list	@GetMapping

Tabela 1: Tabela de mapeamentos com as respetivas anotações.

Para podermos fazer a integração destes métodos com o *backend* desenvolvido na meta passada, foi necessária a injeção de código através da anotação *@Autowired*, cuja possibilita guardar o objeto remoto que estabelece a conexão entre o *Web Server* com o *SearchModel*. Mais detalhes sobre como a ligação foi estabelecida será explicado na próxima secção. Esta anotação foi usada também para guardar a ligação estabelecida pelo *websocket*.

Uma outra anotação usada foi o *@Schedule* para que o método responsável por enviar a informação via *web socket* se repita tempos em tempos.

### 3. Integração SpringBoot - Server RMI

Uma das prioridades do projeto foi a integração do Web server criado nesta meta com o trabalho feito na meta passada de modo a que conseguíssemos fazer uso de todas as funcionalidades desenvolvidas antes através da nova interface Web.

A conexão foi feita entre o Web server SpringBoot e o server RMI da primeira meta (search module) através de uma ligação via RMI, onde o web server vai atuar como cliente do servidor RMI. A conexão é estabelecida no método `init()` (anotado com `@PostConstruct`) da classe `DemoApplication`, que é executado automaticamente durante a inicialização da aplicação Spring-Boot.

No método `init()`, foi criado um Registry usando o endereço do servidor RMI (`RMI_HOST`) e a porta correspondente (`RMI_PORT`). Em seguida, fez-se o `lookup(RMI_NAME)` no registro para obter uma referência ao objeto remoto identificado pelo nome `RMI_NAME`, sendo ele no nosso caso "LoginService". Do lado do servidor RMI é criado o registo através do método "createRegistry" e é utilizado o método `rebind` para associar o objeto remoto ao nome "LoginService" no registro, ficando assim pronto a receber conexões.

Com a conexão RMI estabelecida, do lado do Web server, o objeto remoto é injetado no `loginService` utilizando a anotação `@Autowired`. Com isto a aplicação chama os métodos do objeto remoto de forma transparente, como se estivessem a ser executados localmente. Esta abordagem simplifica a interação com o servidor RMI, permitindo aceder aos seus métodos para satisfazer as funcionalidades da aplicação Gogool.

### 4. WebSockets

Para que pudéssemos atualizar a página de estatísticas automaticamente sem o utilizador fazer um *refresh*, ou seja, sem um pedido novo, resolvemos esse problema recorrendo aos *web sockets*.

Para isso, criamos a *class* `WebSocketConfig` para criar os *endpoints* ao qual os utilizadores podem estabelecer conexão: `/my-websocket` e a *root* onde começam as subscrições: `/topic/...`

No controlador é definida o *endpoint* onde todos os utilizadores subscritos iram receber de cinco em cinco segundos as novas informações.

No HTML temos um elemento de eventos, para que quando o utilizador entra no *endpoint*

/

uma função no *javascript* seja ativada. Esta é responsável de fazer a conexão e subscrição aos *endpoints* corretos. Assim, o utilizador simplesmente entra na vista e fica apto a receber notificações

### 5. Serviço REST

Um ponto importante do projeto foi a integração da aplicação com a API do Hacker News realizada por meio de serviços REST, permitindo que a aplicação se comunique e consuma os dados atualizados em tempo real disponibilizados pelo Hacker News. Essa integração foi essencial para realizar as funcionalidades de indexação de páginas.

A comunicação com a API do Hacker News (funções `hackerNewsTopStories` e `getUserStories`) foi realizada através da biblioteca `RestTemplate`, que é uma classe fornecida pelo Spring Framework para simplificar a integração de serviços RESTful em aplicações Java, facilitando o envio de requisições HTTP para a API. O `RestTemplate` foi utilizado para realizar GETs para a API do Hacker News. O método `restTemplate.getForObject(url, responseType)` foi usado para obter objetos JSON da API e convertê-los automaticamente em instâncias de classes correspondentes (models). Ao receber estes objetos conseguimos obter as informações sobre as top stories do momento na plataforma e todas as stories de certos utilizadores, que são necessárias para as funcionalidades de indexação de páginas.

## 6. Testes

Na secção seguinte enumeramos os testes feitos à aplicação. Em cada teste especificamos o procedimento da funcionalidade, bem como o que é esperado que resulte do teste. Colocamos para cada "Aceite" caso a nossa aplicação cubra tal comportamento e "Rejeitado" caso contrário.

<b>Pesquisar páginas que contenham um conjunto de termos</b>	
Procedimento	No endpoint /search fazer pesquisa e submeter.
Esperado	São mostradas todas as páginas (url, título e paragrafo) que contenham os termos introduzidos.
Resultado	Aceite

Tabela 2: Testes de pesquisa

<b>Registar-se (Sign up)</b>	
Procedimento	No endpoint /register ou pela ligação na Home, introduzir utilizador e password.
Esperado	É feito o registo de um utilizador com as credenciais introduzidas.
Resultado	Aceite

Tabela 3: Teste de registo

<b>Fazer login</b>	
Procedimento	No endpoint /login ou pela ligação na Home, introduzir utilizador e password.
Esperado	É feito o login de um utilizador com as credenciais introduzidas.
Resultado	Aceite

Tabela 4: Teste de login

<b>Fazer logout</b>	
Procedimento	Aceder ao endpoint /logout ou prosseguir a ligação "logout" na Home
Esperado	É feito o logout de um utilizador e é feito o redirecionamento para a página de log in.
Resultado	Aceite

Tabela 5: Teste de logout



	<b>Consultar lista de páginas com ligação para uma página específica</b>
Procedimento	No endpoint /pointed-links ou pela ligação na Home, introduzir o url da pagina pretendida caso esteja com o login feito.
Esperado	E esperado uma lista de todas as páginas (url, titulo e parágrafo) que apontem para a pagina introduzida.
Resultado	Aceite

Tabela 6: Testes de consulta de ligações

	<b>Consultar informações sobre o sistema e 10 pesquisas mais feitas por utilizadores.</b>
Procedimento	Aceder ao endpoint / .
Esperado	É esperado uma lista com as estatísticas (nº de Downloader e Barrels) e uma lista das pesquisas mais feitas na aplicação com atualização em tempo real.
Resultado	Aceite

Tabela 7: Testes de consulta de estatísticas

	<b>Indexar as top 10 stories do Hacker News</b>
Procedimento	Aceder ao endpoint /list ou seguir a ligação "Top 10 Stories"na Home .
Esperado	É apresentada uma lista dos 10 urls correspondentes às top10 stories do Hacker News, e a indexação dos mesmos é feita.
Resultado	Aceite

Tabela 8: Testes de Indexar *URLs*

	<b>Indexar as stories de um user do Hacker News</b>
Procedimento	Aceder ao endpoint /user ou seguir a ligação "User Stories"na Home e introduzir o username pretendido.
Esperado	É apresentada uma lista dos urls correspondentes stories do utilizador do Hacker News introduzido, e a indexação dos mesmos é feita.
Resultado	Aceite

Tabela 9: Testes de Indexar *URLs*