Sistemas Distribuídos

Licenciatura em Engenharia Informática - 2023/24

Projeto

Googol: Motor de pesquisa de páginas Web

Trabalho realizado por:

- Miguel Ângelo Ferreira Miranda, nº 2021212100
- Diogo Borges Simões, nº 2021225264
- Rodrigo Oliveira de Sá, nº 2021213188

Conteúdo

Introdução

O projeto "Googol" tem como objetivo desenvolver um motor de pesquisa de páginas web, oferecendo funcionalidades semelhantes aos principais serviços de busca, como Google e Bing. O sistema inclui a indexação automática de páginas web através de um web crawler e a capacidade de realizar buscas eficientes. Proporciona informações detalhadas sobre cada página indexada, como URL, título e uma citação de texto.

Além disso, o Googol permite que os utilizadores sugiram URLs para serem indexados e realizem pesquisas, retornando resultados ordenados por relevância. A interface web, desenvolvida com Spring Boot e seguindo a arquitetura MVC, permite o acesso fácil a partir de qualquer dispositivo conectado à internet. O projeto também integra funcionalidades de comunicação assíncrona em tempo real usando WebSockets e conecta-se a APIs externas, como Hacker News e OpenAI, para enriquecer as funcionalidades de pesquisa.

Esta abordagem visa criar uma experiência de pesquisa robusta e interativa, alinhando-se com as melhores práticas de desenvolvimento web moderno.

Estrutura Geral do Projeto:

Para a meta 2 do projeto, foi-nos solicitado que desenvolvêssemos uma interface frontend para a nossa aplicação Googol, de modo a facilitar o acesso à plataforma a partir de qualquer dispositivo. Para atingir este objetivo, utilizaremos o servidor RMI desenvolvido na primeira meta do projeto. Assim, os utilizadores da aplicação web terão acesso às mesmas funcionalidades anteriormente disponíveis, como a indexação de URLs e a pesquisa de páginas.

Na implementação do frontend da nossa aplicação, recorremos a diversas tecnologias, incluindo SpringBoot, a API Rest do Hacker News e Web Sockets.

Pacote **org.googled.engine.frontend**:

Este é o pacote principal do projeto web, onde estão agrupadas todas as classes relacionadas com a interface do utilizador, incluindo controladores, modelos e configurações.

Pacote **controllers**:

Contém todos os controladores responsáveis por lidar com pedidos HTTP e fornecer respostas apropriadas. Cada controlador está associado a uma rota específica.

Pacote **api**:

Inclui classes que fazem chamadas a APIs externas ou serviços para obter os dados necessários para a aplicação.

Pacote **config**:

Contém classes de configuração para o projeto, como configurações do Spring MVC, configurações do WebSocket, etc.

Ficheiro **Application.java**:

É a classe principal que inicia a aplicação Spring Boot. Aqui são feitas configurações específicas, como desligar o banner de inicialização e o log de informações de inicialização.

Diretório **resources**:

Contém recursos estáticos, como ficheiros CSS, imagens e scripts JavaScript.

Tecnologias Utilizadas:

- **Spring Boot:** É o framework utilizado para criar a aplicação web, fornecendo uma configuração padrão e simplificando o desenvolvimento.
- **Spring MVC:** É o módulo do Spring Framework utilizado para desenvolver aplicações web. Ele fornece a estrutura MVC para o projeto.
- **Thymeleaf:** É um mecanismo de template para aplicações web baseadas em Java. É usado para renderizar as páginas HTML com dados dinâmicos.
- **Bootstrap:** É um framework front-end utilizado para criar interfaces de utilizador responsivas e elegantes.
- **WebSocket:** É um protocolo de comunicação bidirecional, que permite a comunicação em tempo real entre o cliente e o servidor. É utilizado para atualizar os dados estatísticos em tempo real na página de estatísticas.

Arquitetura Detalhada:

Camada de Controlo:

Contém os controladores que lidam com as requisições HTTP e gerenciam a lógica de negócios. Os controladores incluem:

- SearchController: Responsável por processar as solicitações de pesquisa, tanto para pesquisas normais como para pesquisas no Hacker News. Interage com o serviço de backend através do BackendRMIClient para obter os resultados da pesquisa.
- StatsController: Controla as estatísticas do sistema, enviando estatísticas periodicamente para um endpoint WebSocket para atualização em tempo real.

Camada de Serviço (APIs):

Esta camada contém classes responsáveis por fazer chamadas a APIs externas ou serviços internos para obter os dados necessários para a aplicação.

- **Componentes:**
  - **BackendRMIClient:** Cliente para se comunicar com o backend do sistema via RMI.
  - **HackerNewsRestClient:** Cliente para interagir com a API do Hacker News e obter histórias relevantes.
  - **OpenAIRestClient:** Cliente para se comunicar com uma API externa para gerar análises de texto.

Camada de Modelo:

Contém as classes `HackerNewsItemRecord` e `HackerNewsUserRecord`, que representam os dados das histórias e dos utilizadores do Hacker News, respectivamente. Estas classes são utilizadas para mapear os dados recebidos em formato JSON para objetos Java.

HackerNewsItemRecord:

Esta classe representa um registo de um item do Hacker News.

- @JsonIgnoreProperties(ignoreUnknown = true): Esta anotação indica que quaisquer propriedades desconhecidas (não mapeadas) no JSON devem ser ignoradas ao deserializar o objeto Java.
- Os campos nesta classe correspondem aos diferentes atributos de um item do Hacker News, como id, type, by (autor), time (data de criação), text (texto do item), url (URL da história), score (pontuação), e assim por diante.
- Este registo é usado para mapear os dados recebidos do Hacker News em objetos Java para facilitar o processamento e manipulação desses dados no backend.

HackerNewsUserRecord:

Esta classe representa um registo de um utilizador do Hacker News.

- @JsonIgnoreProperties(ignoreUnknown = true): Tal como no HackerNewsItemRecord, esta anotação indica que quaisquer propriedades desconhecidas devem ser ignoradas ao deserializar o objeto Java.
- Os campos nesta classe representam os diferentes atributos de um utilizador do Hacker News, como id (nome de utilizador), created (data de criação da conta), karma (pontuação do utilizador), about (descrição do utilizador) e submitted (histórias, votações e comentários submetidos pelo utilizador).
- Este registo é usado para mapear os dados do utilizador do Hacker News em objetos Java para facilitar o processamento e manipulação desses dados no backend.

Camada de Visualização:

Contém os ficheiros HTML que definem a aparência das páginas da web.

- Ficheiros como search.html, hacker-search.html, stats.html, user-stories.html, etc., são usados para renderizar as diferentes páginas da web com base nas solicitações do utilizador.

Camada WebSocket:

Esta camada é responsável pela comunicação em tempo real entre o cliente e o servidor. Ela é usada para atualizar os dados estatísticos na página de estatísticas sem a necessidade de recarregar a página.

- **Componentes:**
  - **stats.js:** Script JavaScript responsável por se conectar ao WebSocket e receber atualizações de estatísticas em tempo real.

Serviços e Componentes Utilitários:

- BackendRMIClient: Um cliente RMI (Remote Method Invocation) que comunica com o serviço de backend para obter resultados de pesquisa e indexar URLs.
- SimpMessagingTemplate: Uma instância do Spring Messaging Template usada para enviar mensagens para um tópico WebSocket para atualizar as estatísticas em tempo real.
- RestTemplate: Uma classe do Spring utilizada para fazer chamadas HTTP para serviços externos, como o Hacker News API e o serviço de análise Gemini API.

Configuração do Spring Boot:

- Application.java: A classe principal da aplicação Spring Boot que configura e inicia a aplicação, definindo configurações como a ausência de banner de inicialização, log de inicialização e configurações de agendamento.

Interações entre Camadas:

- Os controladores recebem os pedidos HTTP dos utilizadores e coordenam a lógica de negócios apropriada.
- Eles acessam os serviços da camada de serviço para buscar dados ou executar operações específicas.
- Os serviços da camada de serviço, por sua vez, podem fazer chamadas a APIs externas ou acessar recursos internos para obter os dados necessários.
- Os dados são então passados para os controladores, que os encaminham para as visualizações.
- As visualizações usam Thymeleaf para renderizar os dados e exibi-los aos utilizadores.
- Além disso, a camada WebSocket permite a comunicação bidirecional em tempo real entre o cliente e o servidor, facilitando a atualização dinâmica de dados na interface do utilizador.

Fluxo de Comunicação:

1. **Pedido do Utilizador:**
    - Um utilizador faz um pedido HTTP ao servidor web, por exemplo, uma pesquisa ou uma solicitação para indexar uma URL.
2. **Processamento no Servidor Web:**
    - O controlador correspondente no servidor web recebe o pedido e, se necessário, interage com os serviços da camada de serviço.
3. **Comunicação com a Gateway:**
    - Se o pedido requer acesso à gateway, o serviço apropriado na camada de serviço, como BackendRMIClient, é chamado.
    - O BackendRMIClient estabelece uma conexão com a gateway e faz uma chamada para executar a operação desejada, como pesquisar no índice de pesquisa ou indexar uma URL.
4. **Processamento na Gateway:**
    - A gateway recebe a chamada e executa a operação solicitada, como buscar dados no índice de pesquisa ou adicionar uma URL ao índice.
    - Se necessário, a gateway pode acessar recursos ou serviços internos para concluir a operação.
5. **Resposta para o Servidor Web:**
    - A resposta da gateway é retornada ao BackendRMIClient, que a encaminha para o controlador correspondente no servidor web.
6. **Resposta ao Utilizador:**
    - O controlador no servidor web utiliza os dados recebidos para gerar uma resposta adequada ao utilizador, que é então enviada de volta ao navegador do utilizador como uma página HTML ou outro tipo de resposta.

BackendRMIClient

A classe BackendRMIClient é fundamental para a interação remota entre o cliente e o gateway, utilizando a tecnologia RMI (Remote Method Invocation). Esta classe permite que o cliente se conecte ao gateway, envie consultas e receba respostas, tudo de forma remota e eficiente.

Esta classe carrega propriedades a partir de ficheiros de configuração para estabelecer a conexão RMI com o gateway.

A BackendRMIClient implementa métodos para enviar consultas de pesquisa (searchQuery), obter URLs que apontam para uma URL específica (getPointingURLs) e indexar URLs (index). Também implementa o método updateStats, que recebe atualizações de estatísticas do gateway e as armazena como uma string JSON. Utilizámos a biblioteca SLF4J para registar as atividades do cliente, facilitando a depuração e monitorização.

A BackendRMIClient garante uma comunicação robusta e eficiente com o gateway, suportando operações críticas do sistema.

WebSocketConfig

A configuração é implementada numa classe Java localizada no pacote org.googled.engine.frontend.config. A classe é anotada com @Configuration e @EnableWebSocketMessageBroker, indicando que é uma configuração Spring e ativando o suporte ao WebSocket, respetivamente.

A configuração do broker de mensagens é feita através do método configureMessageBroker, onde um broker simples é habilitado com o prefixo "/topic" para permitir que os clientes se inscrevam em tópicos de mensagens. Além disso, é definido o prefixo "/stats" para direcionar mensagens do cliente para métodos de manipulação de mensagens no servidor.

O método registerStompEndpoints é responsável por registar o ponto de extremidade STOMP, permitindo que os clientes se conectem à aplicação via WebSocket. O ponto de extremidade é configurado em "/stats", e o suporte a SockJS é habilitado para garantir compatibilidade com navegadores que não suportam nativamente o WebSocket.

Esta configuração possibilita a comunicação em tempo real entre a aplicação Spring Boot e clientes web, proporcionando uma experiência interativa e dinâmica.

Controllers

FailController

Este código Java define um controlador Spring responsável por lidar com erros na aplicação. Aqui está uma explicação detalhada:

1. **Declaração do Pacote e Imports**:
   - O código está contido no pacote org.googled.engine.frontend.controllers.
   - São importadas as classes necessárias do Spring Framework para definir um controlador e lidar com erros.
2. **Declaração da Classe do Controlador**:
   - Esta classe é um controlador do Spring, indicado pela anotação @Controller, o que significa que ela trata pedidos HTTP.
   - Além disso, implementa a interface ErrorController, que permite lidar com erros globais na aplicação.
3. **Método handle**:
   - Este método é mapeado para a rota "/error", o que significa que será invocado sempre que ocorrer um erro na aplicação.
   - Ele retorna uma string que representa o nome da página de erro a ser exibida ao utilizador.

Este controlador é responsável por direcionar o fluxo da aplicação para uma página de erro específica sempre que ocorrer um erro durante a execução. Isso é essencial para garantir uma experiência de utilizador consistente e informativa em caso de falhas.

RootController

Este código Java define um controlador Spring responsável por lidar com solicitações à página inicial da aplicação. Aqui está uma explicação detalhada:

1. **Declaração da Classe do Controlador**:
   - Esta classe é um controlador do Spring, indicado pela anotação @Controller, o que significa que ele trata pedidos HTTP.
   - Não há implementação adicional, mas a classe é documentada para descrever a sua finalidade.
2. **Método redirect**:
   - Este método está mapeado para a URL raiz "/" através da anotação @GetMapping("/"), o que significa que será invocado quando o utilizador aceder à página inicial.
   - O método retorna uma string que representa o redirecionamento para a página de pesquisa ("/search").

Este controlador tem como objetivo redirecionar o utilizador da página inicial para a página de pesquisa da aplicação. Isso é útil para direcionar os utilizadores para a funcionalidade principal da aplicação logo após o acesso inicial.

SearchController

Este código Java define um controlador Spring responsável por gerir as operações na página de pesquisa de uma aplicação.

1. Declaração do Pacote e Imports

- O código está contido no pacote org.googled.engine.frontend.controllers.
- São importadas várias classes do Spring Framework e outras bibliotecas necessárias para a implementação do controlador e suas funcionalidades.

2. Declaração da Classe do Controlador

- A classe é anotada com @Controller, indicando que é um controlador Spring que trata pedidos HTTP.
- Um logger é definido para registar informações e erros da aplicação.

3. Método search

- Este método lida com pedidos GET à URL /search.
- Se uma consulta (query) for fornecida, o método processa a pesquisa. Dependendo do parâmetro hackerNewsSearch, a pesquisa pode ser realizada no Hacker News ou no backend da aplicação.
- Para pesquisas no Hacker News, o método searchTopStoriesOnHackerNews é chamado.
- Para pesquisas no backend, o método chama o cliente RMI para obter os resultados, que são então paginados e adicionados ao modelo.
- Se não houver uma consulta, o método simplesmente retorna a página principal de pesquisa.

4. Método searchTopStoriesOnHackerNews

- Este método realiza uma pesquisa nas principais histórias do Hacker News.
- Obtém as IDs das histórias principais a partir de um endpoint do Hacker News.
- Para cada história, verifica se o título contém a consulta fornecida e, se sim, adiciona a URL e o título da história ao mapa de resultados.
- Os URLs encontrados são indexados usando o cliente RMI.

5. Método generateAnalysis

- Este método gera uma análise da consulta usando a API Gemini.
- Constrói e envia uma requisição HTTP POST para a API, processa a resposta JSON e extrai o conteúdo gerado.
- Adiciona a análise gerada ao modelo.

6. Método indexUrl

- Este método lida com pedidos POST à URL /index.
- Indexa um URL fornecido no backend RMI e adiciona o resultado ao modelo.

7. Método getPageLinks

- Este método lida com pedidos GET à URL /page-links.
- Obtém os URLs que apontam para um URL específico e adiciona-os ao modelo.

8. Método fetchHackerNewsStories

- Este método lida com pedidos POST à URL /fetch-hackernews-stories.
- Obtém as histórias submetidas por um utilizador específico do Hacker News, indexa-as e adiciona as URLs das histórias ao modelo.

Este controlador é essencial para gerir a funcionalidade de pesquisa da aplicação, integrando resultados de várias fontes, como o backend da aplicação e o Hacker News, e proporcionando uma experiência de utilizador rica e informativa.

StatsController

Este código Java define um controlador Spring responsável por gerir a visualização e transmissão de estatísticas em tempo real na aplicação. A seguir, uma explicação detalhada em Português de Portugal para o relatório:

1. Declaração do Pacote e Imports

- O código está contido no pacote org.googled.engine.frontend.controllers.
- São importadas várias classes do Spring Framework, incluindo suporte para agendamento de tarefas e mensagens WebSocket, além de classes específicas do projeto.

2. Declaração da Classe do Controlador

- A classe é anotada com @Controller, indicando que é um controlador Spring que trata pedidos HTTP.

- É também anotada com @EnableScheduling para permitir a execução de tarefas agendadas.
- Um logger é definido para registar informações e erros da aplicação.

3. Injecção de Dependências

- A classe usa a anotação @Autowired para injetar uma instância de SimpMessagingTemplate, que é usada para enviar mensagens aos clientes WebSocket.

4. Método stats

- Este método lida com pedidos GET à URL /stats.
- Retorna o nome da vista "stats", que é a página onde as estatísticas serão exibidas.

5. Método sendStats

- Este método é anotado com @Scheduled(fixedRate = 5000), o que significa que será executado a cada 5 segundos.
- Dentro deste método, uma simulação de obtenção de estatísticas é realizada. Normalmente, este método chamaria um cliente RMI para obter estatísticas reais do backend.
- As estatísticas obtidas são registadas e enviadas para o endpoint WebSocket /topic/stats usando messagingTemplate.convertAndSend.
- Em caso de erro durante a obtenção das estatísticas, o erro é registado no logger.

Funcionalidade Geral

Este controlador é responsável por gerir e enviar estatísticas em tempo real para os clientes conectados via WebSocket. Através do método agendado sendStats, ele obtém periodicamente as estatísticas do backend e as transmite para todos os clientes subscritos no endpoint /topic/stats. Isso é crucial para fornecer uma atualização contínua e em tempo real das métricas da aplicação, melhorando a interatividade e a experiência do utilizador.

Application

Este controlador é responsável por iniciar a aplicação Spring Boot. Através do método main, a aplicação é configurada para não exibir o banner padrão do Spring Boot e para não logar informações detalhadas de inicialização, proporcionando um início mais silencioso. Quando a aplicação é iniciada, o Spring Boot faz a configuração automática dos componentes e inicia a aplicação com base nas configurações definidas.

Main.js

Este código JavaScript é responsável por estabelecer uma conexão WebSocket utilizando a biblioteca STOMP para receber e exibir estatísticas em tempo real numa página web. A seguir, uma explicação detalhada em Português de Portugal para o relatório:

Funcionalidade do Código

1. **Variável webSocketStompClient**:
    - Declara uma variável global webSocketStompClient para armazenar o cliente STOMP sobre SockJS.
2. **Função connectWebsocket**:
    - Esta função estabelece a conexão WebSocket e subscreve-se ao tópico /topic/stats para receber atualizações.

Passos na Função connectWebsocket

1. **Criação do Objeto SockJS**:
    - var socket = new SockJS('/stats'); cria um novo objeto SockJS que se conecta ao endpoint /stats.
2. **Inicialização do Cliente STOMP**:
    - webSocketStompClient = Stomp.over(socket); cria um cliente STOMP sobre a conexão SockJS.
3. **Conexão ao WebSocket**:
    - webSocketStompClient.connect({}, function (frame) {...}); estabelece a conexão WebSocket. Quando a conexão é bem-sucedida, a função de callback é executada.
4. **Subscrição ao Tópico**:
    - Dentro do callback, webSocketStompClient.subscribe('/topic/stats', function (message) {...}); subscreve-se ao tópico /topic/stats. Cada vez que uma mensagem é recebida neste tópico, a função de callback é executada.
5. **Tratamento das Mensagens Recebidas**:
    - console.log('Websocket got stats: ' + message.body); regista a mensagem recebida na consola.
    - var stats = JSON.parse(message.body); converte o corpo da mensagem JSON num objeto JavaScript.
    - O conteúdo do elemento com id stats é limpo.
    - São criados e adicionados novos elementos ao DOM para exibir as estatísticas:
        - **Data da Última Atualização**: Um parágrafo com a data da última atualização.
        - **Top Ten Queries**: Um título e uma lista não ordenada das dez principais consultas.
        - **Número de Barris Conectados**: Um parágrafo com o número de barris conectados.

Conclusão

Este código estabelece uma conexão WebSocket com o servidor, subscreve-se a um tópico específico para receber atualizações de estatísticas e atualiza a interface do utilizador com os dados recebidos. Esta funcionalidade é crucial para fornecer informações em tempo real aos utilizadores da aplicação, melhorando a interatividade e a experiência do utilizador.

Integração de Spring Boot com o servidor RMI

A integração do Spring Boot com o servidor RMI no nosso código é feita principalmente na classe SearchController. Aqui está uma explicação passo a passo da integração:

Cliente RMI:

- No método search(), criamos uma instância de um cliente RMI, BackendRMIClient, para interagir com o servidor RMI.
- Este cliente RMI é usado para fazer chamadas remotas aos métodos do servidor RMI, como searchQuery(), index(), e getPoitingURLs().

Registo RMI:

- No cliente RMI, o método searchQuery() comunica com o servidor RMI para realizar a pesquisa e obter os resultados.
- O método index() é utilizado para enviar uma URL ao servidor RMI para indexação.
- O método getPoitingURLs() é utilizado para obter URLs apontadas pela URL fornecida.

Integração através de Instanciação Direta:

- No código fornecido, não usamos a anotação @Autowired para injetar o cliente RMI. Em vez disso, ele é instanciado diretamente onde necessário, como no método search().
- Esta abordagem simplifica a integração, já que não precisamos configurar explicitamente a injeção de dependência através de anotações.

Comunicação Transparente:

- A comunicação com o servidor RMI é tratada de forma transparente, como se os métodos estivessem a ser executados localmente.
- Isto simplifica a interação com o servidor RMI, permitindo-nos chamar os seus métodos remotos diretamente nas classes de controlo onde são necessários.

Tratamento de Exceções:

- No código, também lidamos com exceções que podem ocorrer durante a comunicação com o servidor RMI, como falhas na conexão ou erros de processamento no servidor.

Em resumo, a integração do Spring Boot com o servidor RMI no nosso código é feita principalmente através da instanciação direta de um cliente RMI nas classes de controlo relevantes, como SearchController. Isto permite que a nossa aplicação Spring Boot comunique com o servidor RMI de forma transparente, facilitando o acesso aos seus métodos remotos para atender às funcionalidades da aplicação Googled.

Integração de WebSockets com Spring Boot e RMI

Para permitir uma comunicação em tempo real entre o servidor e o cliente, implementámos a integração de WebSockets com Spring Boot e RMI. Isto possibilita atualizar dinamicamente a interface do utilizador do cliente com informações em tempo real provenientes do servidor.

No lado do servidor, configurámos a comunicação WebSocket usando Spring Boot. A configuração é feita na classe WebSocketConfig, que está anotada com @Configuration e implementa WebSocketMessageBrokerConfigurer. Nesta configuração, habilitámos um corretor de mensagens simples usando config.enableSimpleBroker("/topic") e definimos o prefixo de destino da aplicação como "/stats" usando config.setApplicationDestinationPrefixes("/stats"). Além disso, registámos o ponto de extremidade do STOMP (Simple Text Oriented Messaging Protocol) usando registry.addEndpoint("/stats").withSockJS(), permitindo que os clientes se conectem ao servidor através deste endpoint.

No lado do cliente, usámos JavaScript para estabelecer uma conexão WebSocket e receber atualizações em tempo real. Primeiro, criámos um objeto SockJS que representa a conexão WebSocket com o endpoint definido no servidor. Em seguida, usámos o cliente Stomp.js para nos conectarmos ao servidor WebSocket e subscrevermos no tópico /topic/stats. Quando uma mensagem é recebida neste tópico, processamos os dados e atualizamos dinamicamente a interface do utilizador com as estatísticas recebidas.

No servidor, para enviar periodicamente estatísticas aos clientes conectados, implementamos o StatsController. Este controlador é responsável por enviar estatísticas periodicamente para os clientes conectados por meio do WebSocket. Está marcado com @Controller e @EnableScheduling, o que permite a execução de tarefas agendadas. No método sendStats, anotado com @Scheduled(fixedRate = 5000), as estatísticas são recuperadas (neste caso, estáticas) e enviadas aos clientes conectados ao endpoint WebSocket /topic/stats usando o SimpMessagingTemplate.

Esta integração permite uma comunicação bidirecional em tempo real entre o servidor e o cliente, permitindo que a interface do utilizador do cliente seja atualizada dinamicamente conforme as alterações ocorrem no servidor.

Integração de REST WebServices no Projeto

No lado do servidor, os endpoints RESTful são definidos nas classes de controlador, que estão anotadas com @RestController ou @Controller. Cada método mapeado com anotações como @GetMapping, @PostMapping, etc., representa um endpoint que processa pedidos HTTP específicos.

Por exemplo, o método search no SearchController é mapeado para o endpoint /search com o método HTTP GET. Ele recebe parâmetros de consulta, como query, page e hackerNewsSearch, e retorna os resultados da pesquisa com base nesses parâmetros.

O método indexUrl no mesmo controlador é mapeado para o endpoint /index com o método HTTP POST. Ele recebe um URL como parâmetro e executa uma operação específica, como indexar esse URL no backend.

Para consumir esses endpoints do lado do cliente, podes usar bibliotecas HTTP como o RestTemplate ou o WebClient fornecidos pelo Spring. Por exemplo, no método fetchHackerNewsStories, um RestTemplate é usado para fazer uma solicitação GET para a API do Hacker News e obter os detalhes das histórias de um usuário.

Além disso, a integração de WebSockets, conforme explicado anteriormente, também é uma forma de comunicação bidirecional entre o cliente e o servidor, que complementa a integração de REST WebServices, permitindo a transmissão assíncrona de dados em tempo real.

Esta abordagem de integração de REST WebServices oferece uma maneira eficaz de criar uma API para o teu projeto, permitindo a comunicação entre diferentes partes do sistema e serviços externos de forma padronizada e escalável.

Testes

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

Pesquisar páginas que contenham um conjunto de termos

Procedimento No endpoint /search fazer pesquisa e submeter.

Esperado São mostradas todas as páginas (url, titulo e paragrafo) que contenham os termos introduzidos.

Resultado Aceite

Consultar lista de páginas com ligação para uma página específica

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

Consultar informações sobre o sistema e 10 pesquisas mais feitas por utilizadores.

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

Indexar as top 10 stories do Hacker News

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

Indexar as stories de um user do Hacker News

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