Relatório Final: Sistemas Distribuídos - 2024/2025

Bruna Dewes - uc2024243221@student.uc.pt
Cecília Quaresma - uc2024245307@student.uc.pt
Heloísa Centenaro - uc2024246775@student.uc.pt

1. Introdução

Este relatório visa apresentar uma visão completa e detalhada do projeto sobre um mini motor de busca, que envolveu o desenvolvimento de um sistema responsável por indexar páginas da web, armazenar suas informações e buscar os dados coletados.

O principal objetivo deste documento é demonstrar o processo de desenvolvimento, baseado em uma arquitetura distribuída, onde foi utilizada a linguagem Java, conexões RMI, Spring Boot, Websockets e integração com APIs externas como a do Hacker News e OpenRouter, além de conter as decisões técnicas do grupo, as demais etapas e estruturas do sistema, que nos permitiu chegar ao resultado final.

Link do repositório: **https://github.com/cqcoding/SistemasDistr/tree/main.**

2. Arquitetura de Software

A arquitetura escolhida para o projeto segue o modelo cliente-servidor distribuído, com múltiplos componentes que se comunicam entre si através das chamadas remotas via RMI (Remote Method Invocation) e as funcionalidades do framework Spring Boot para a interface web, Websockets para fornecer as atualizações em tempo real aos usuários, HackerNews e integração de API REST.

2.1 Componentes Principais:

Cliente.java: Interface simples de linha de comando que permite ao usuário fazer buscas, enviar URLs para serem indexadas e visualizar estatísticas do sistema. Foi essencial para testar funcionalidades e sem a necessidade de interface.

GatewayServer.java: Recebe requisições tanto da interface web quanto do cliente e distribui essas requisições para os demais servidores (Barrels).

SearchController.java: Faz a ponte entre a web e os serviços internos usando RMI.

Barrels.java: Servidores responsáveis por indexar e armazenar os conteúdos das páginas. Eles mantêm os índices invertidos, que permitem realizar buscas eficientes e também organizam filas de URLs a serem processadas.

Downloader.java: Baixam as páginas da web e extraem o conteúdo relevante. Essas informações são enviadas aos Barrels para serem armazenadas e indexadas.

RegistrarBarrels.java e Servidor.java: Componentes que registram os Barrels no sistema e fazem a conexão direta com o servidor.

3. Organização do Código

A fim de manter o projeto bem estruturado e facilitar a manutenção, organizamos o código em pacotes conforme a responsabilidade de cada classe:

- **com**: Contém as classes principais como, por exemplo, GatewayServer, Barrels, Downloader, Cliente, RegistrarBarrels e Servidor.
- **com.api**: Controladores da interface web, desenvolvidos com Spring Boot.
- **com.api.services**: Serviços auxiliares, como o responsável pelas atualizações em tempo real.
- **com.api.config**: Configurações do Spring Boot e WebSockets.
- **resources/templates**: Arquivos HTML
- **resources/static**: Estilos CSS e scripts JavaScript.
- **resources/config.properties**: Arquivo com informações de configuração do endereço do servidor.

4. Comunicação Distribuída com RMI

A comunicação entre os diferentes módulos do sistema foi feita com Java RMI, o que nos permitiu trabalhar com vários processos de forma coordenada.

4.1 Mecanismo de Integração

Criamos interfaces remotas (InterfaceGatewayServer e InterfaceBarrel) que definem os métodos disponíveis para comunicação. Cada servidor se registra no RMI Registry, permitindo que os outros componentes encontrem e se comuniquem com ele dinamicamente.

4.2 Integração com Spring Boot

A interface web, via Spring Boot, também participa desse sistema distribuído. O SearchController se conecta ao Gateway utilizando Naming.lookup() e executa chamadas remotas com base nas configurações definidas no arquivo config.properties.

5. Interface Web com Spring Boot e Thymeleaf

A criação de uma interface web foi fundamental para tornar o sistema mais acessível e onde foi utilizado o framework Spring Boot para montar um servidor HTTP e o Thymeleaf para gerar páginas HTML dinâmicas.

5.1 Páginas Web

search.html: Página principal para buscar termos e enviar novas URLs.

statistics.html: Mostra estatísticas de forma dinâmica, como o número de pesquisas feitas e os barrels ativos.

relacoes.html: Exibe a lista de backlinks de uma URL, ou seja, quem está apontando para ela.

hackernews.html: Apresenta dados da integração com a API do Hacker News.

5.2 SearchController

Controlador desenvolvido para mapear as rotas HTTP para métodos que, por sua vez, se comunicam com o Gateway via RMI. Assim, conseguimos unir a web ao sistema distribuído por trás.

6. WebSockets e Atualizações em Tempo Real

6.1 Configuração

Criamos uma configuração Websocket personalizada com **STOMP**, que define um endpoint /ws e um tópico /topic/statistics. Isso permite que as estatísticas do sistema sejam atualizadas em tempo real.

- **WebSocketConfig.java**: Define os endpoints e tópicos disponíveis.
- **RealTimeUpdateService.java**: Usa SimpMessagingTemplate para enviar mensagens com dados atualizados aos navegadores conectados.

6.2 Lado do Cliente

Usamos bibliotecas como **SockJS** e **STOMP.js** para permitir que o navegador se conecte ao Websocket. Sempre que novas estatísticas são geradas, a página é atualizada.

7. Consumo e Exposição de APIs REST

7.1 API REST Interna

Disponibilizamos endpoints REST como /search, /index-url, /relacoes e /statistics, que podem ser acessados tanto pela interface web quanto por qualquer outro cliente HTTP.

7.2 Integrações com APIs Externas

OpenRouter API: Integramos essa API para gerar análises contextuais das URLs indexadas, com base no termo pesquisado pelo usuário. Utilizamos a biblioteca Unirest para fazer as requisições HTTP e exibir os resumos retornados.

Hacker News API: Implementamos uma integração com o Hacker News para exibir as principais notícias e informações de itens relevantes. Para isso, usamos o RestTemplate do Spring. A integração permite fazer buscas nas 50 Top Stories do site, criando um novo mecanismo para indexar diferentes URLs que serão armazenadas em arquivo .txt para persistência.

8. Processos, Threads e Concorrência

8.1 Processos Distribuídos

Cada módulo do sistema (Gateway, Barrels, Downloaders e Cliente) roda como um processo separado, podendo ser executado em diferentes máquinas.

8.2 Uso de Threads

GatewayServer.java: Lança threads para monitorar a disponibilidade dos Barrels.

Downloader.java: Usa um ExecutorService para realizar múltiplos downloads simultâneos, otimizando o tempo de processamento.

9. Fluxo de Funcionamento

Indexação de URLs: O usuário submete uma URL → Gateway envia para os Barrels → Página é baixada → Conteúdo é indexado. O usuário também pode fazer uma requisição na página das HackerNews, que faz a busca e já envia diretamente a URL ao barrel caso retorne a palavra desejada.

Busca de Relações: Sistema analisa os backlinks a partir de arquivos de links de saída.

Análises Semânticas: A OpenRouter API gera resumos sobre os conteúdos encontrados.

Atualizações Dinâmicas: Estatísticas são atualizadas em tempo real por Websockets.

10. Decisões Técnicas

Tecnologia	Justificativa
Java + RMI	Simplicidade na comunicação entre os componentes distribuídos.
Spring Boot	Agilidade no desenvolvimento da interface e RESTful APIs.
Thymeleaf	Permite páginas HTML dinâmicas com integração ao Spring.
WebSockets	Atualizações dos dados em tempo real para disponibilizar aos usuários.
REST + APIs	Expansão do sistema com recursos inteligentes e dados externos.

11. Testes

ID	DESCRIÇÃO	PRÉ-CONDIÇÕES	PASSOS	RESULTADO ESPERADO	STATUS
T01	Conexão com o BarrelServer	Ambiente RMI configurado. BarrelServer em execução.	1. Aplicação cliente tenta obter uma referência ao stub RMI do BarrelServer. 2. Invocar um método simples para ver se conecta.	Conexão estabelecida sem exceções (ex: RemoteException, NotBoundException).	PASS
T02	indexar_URL	BarrelServer conectado e ativo. URL "https://example.com" acessível e válida.	1. Aplicação cliente envia a URL "https://example.com" para o BarrelServer via	A URL "https://example.com" deve ser indexada sem erros e uma mensagem de confirmação deve ser exibida. O conteúdo da	PASS

			método indexar_URL.	URL é processado e armazenado no índice do Barrel.	
T03	pesquisar	BarrelServer/GatewayServer conectado e ativo. Palavra "guerra" previamente indexada com URLs associadas.	1. Aplicação cliente envia a palavra "guerra" para o método de pesquisa. 2. Esperar a lista de resultados.	A pesquisa deve retornar uma lista de SearchResult contendo URLs relevantes. A lista pode estar vazia se "guerra" não estiver indexada.	PASS
T04	estaAtivo	BarrelServer/GatewayServer em execução.	1. Invocar o método estaAtivo() no servidor RMI.	Retorna true se o servidor estiver ativo e respondendo.	PASS
T06	sincronizarDados	Pelo menos dois BarrelServers (B1, B2) ativos. Mecanismo de sincronização implementado. Dados diferentes em B1 e B2.	1. Acionar o processo de sincronização de dados entre B1 e B2. 2. Após a sincronização, verificar os dados em ambos os Barrels.	Dados sincronizados corretamente entre B1 e B2.	PASS
T07	Conexão com GatewayServer	Ambiente RMI configurado. GatewayServer em execução e acessível.	1. Aplicação cliente/Barrel tenta obter uma referência ao stub RMI do GatewayServer. 2. Verificar se a referência não é nula.	A conexão deve ser estabelecida sem lançar exceções de RemoteException ou NotBoundException.	PASS
T08	Pesquisar uma palavra não indexada	Sistema online. Pelo menos um Barrel ativo. Palavra "jgvjgvjccgcjcj" não indexada.	1. Abrir a interface de busca (search.html). 2. Inserir "jgvjgvjccgcjcj" no campo de pesquisa. 3. Clicar no botão "Pesquisar".	A interface exibe uma mensagem como "Sem resultados para essa busca". A pesquisa pode ser registada em pesquisasFrequentes.txt.	PASS
T09	Pesquisar com um Barrel inativo	Sistema online. Barrels registrados. 1 barrel é torna-se inativo.	1. Abrir search.html. 2. Pesquisar uma palavra. 3. Desligar 1 barrel. 4. Repetir a pesquisa.	Deve retornar os mesmos resultados nas duas vezes, visto que todos os barrels tem o mesmo conteúdo.	PASS
T10	Adicionar um novo Barrel ao sistema manualmente	Sistema online com N Barrels.	1. Iniciar uma nova instância de Barrels.java. 2. Garantir que o novo Barrel se registra no GatewayServer. 3. Realizar pesquisas.	Novas pesquisas são distribuídas também para o novo Barrel. Se o Barrel tiver dados, contribui para os resultados. 'statistics.html' deve mostrar N+1 Barrels ativos. O sistema continua a funcionar sem interrupções.	PASS
T11	Remover um Barrel do sistema manualmente	Sistema online com N Barrels (Mais que 1).	1. Desligar um Barrel (ou simular falha). 2. Realizar pesquisa. 3. Verificar estatísticas	Pesquisas continuam a funcionar (pois todos os barrels tem o mesmo conteúdo). Página de estatísticas mostra os barrels ativos.	PASS
T12	Indexar uma URL que não	Sistema online. Downloader e Barrels	1. Indexar "htp://exemplo".	A URL não é adicionada a urlsIndexados.txt. Downloader	PASS

	está nos padrões (ex: malformada)	ativos.		solta uma exception.	
T13	Indexar uma URL que não está nos padrões (ex: conteúdo não HTML)	Sistema online. Downloader e Barrels ativos.	1. Indexar uma URL que aponta para uma imagem (.jpg) diretamente.	O Downloader vai soltar uma exceção e a URL não vai ser adicionada a urlsIndexados.txt.	PASS
T14	Colocar uma URL válida na funcionalidade de "indexar URL"	Sistema online. Interface com campo para submeter URL para indexação.	1. Inserir uma URL válida e ainda não indexada. 2. Clicar em "Indexar".	A interface exibe uma mensagem de "URL enviada para indexação". Após algum tempo, a URL e seu conteúdo (termos) são indexados por um Barrel. A URL aparece em urlsIndexados.txt. A pesquisa por termos da página retorna resultados.	PASS
T15	Pesquisar uma palavra que tenha uma URL relevante encontrada no Hacker News.	Sistema online. Palavra com URL relevante no Hacker News.	1. Pesquisar uma palavra com URL relevante no Hacker News.	Retorna na lista de resultados a url do hacker news.	PASS
T16	Ao fazer uma pesquisa ver uma análise contextualizada	Sistema online. API_KEY funcional.	1. Fazer uma pesquisa e ver análise contextualizada logo abaixo dos resultados.	Análise contextualizada aparecer logo abaixo dos resultados da busca, sem erros de API_KEY.	PASS (desde que a KEY esteja funcional)

12. Considerações Finais

Desenvolver este projeto foi uma experiência muito desafiadora, mas de muitos aprendizados também, pois conseguimos aplicar conceitos importantes de sistemas distribuídos, programação concorrente, desenvolvimento web e integração de API, que até então só havíamos visto na teoria. Por isso foi pensado e desenvolvido com foco em modularidade e integração de tecnologias, também para oferecer um sistema eficiente e uma interface interativa e responsiva.

Além disso, a documentação, aliada à estrutura e à organização dos códigos, possibilita o entendimento das soluções utilizadas.