Relatório SD

Arquitetura de software:

Backend:

Contém classes e métodos utilizados na Meta1, que estão responsáveis por todas as funcionalidades principais do sistema. Não foram feitas alterações a esta estrutura, porém foi alterado o método *search()*, para suportar algumas funcionalidades da Meta2. Foi modificado para suportar paginação, incluindo dois novos parâmetros: *int page,* que está responsável por enviar o número atual da página, para que o frontend possa apresentar resultados agrupados em grupos de 10, devolvendo apenas os resultados da página em questão; *boolean isPagination,* é responsável por verificar se a ação no frontend é ou não uma mudança de página, de modo que, não haja um incremento incorreto no número de pesquisas por palavra.

Frontend:

Para integrar uma página web com as funcionalidades da meta anterior, foram criados Controllers, Models e Templates. Para tornar mais simples a compreensão do código, os Controllers estão organizados por funcionalidades (pesquisa, indexação, APIs, Websockets).

- *HomeController* - Define a página principal, que inclui o menu principal com acesso a todas as funcionalidades do sistema.
- *AddUrlController* - É responsável pela indexação de urls. Verifica se o input é um link válido e procede à sua indexação através de RMI, utilizando a função *putNew()* da *Gateway*. Esta ação envia um trigger para atualizar as estatísticas em tempo real (via WebSockets), uma vez que altera o tamanho dos Barrels.
- *SearchController* - Utilizado para receber uma ou várias palavras como input, recorrendo ao método *search()* da *Gateway* através de chamada RMI. Esta função envia a query, o número atual da página (default = 1), e indica se é uma mudança de página (default = false).
  Os resultados são recebidos por página e apresentados em grupos de 10, contendo o título e uma breve descrição.
  É enviado um trigger para atualizar as estatísticas em tempo real (via WebSockets), uma vez que pode alterar o top10 de pesquisas.
  Para além dos resultados da pesquisa, é apresentada uma análise, no topo da página, utilizando a AI, *Gemini,* que, através da query, os títulos e descrições dos resultados gera um breve texto acerca do conteúdo enviado.
- *LinksToController* - Semelhante ao anterior, este Controller recebe um link de input, e recebe os links que apontam para ele, utilizando a função *getPagesLinkingTo()* da Gateway, através de RMI. Os resultados são exibidos utilizando a mesma configuração que os resultados do método *search().*
- *Top10Controller*- É responsável por apresentar as 10 páginas que possuem mais ligações de outras páginas, utilizando a função *getTop10PagesByLinks(*) da Gateway, através de chamada RMI. Após receber o resultado desse método, mostramos o top10 de forma simples na página web.

- *WebSocketStatsController* - Utilizamos a função da Gateway getFormattedStatistics(), através de chamada RMI, para receber o top10 pesquisas mais frequentes, a quantidade de barrels existentes, e respetivo tamanho, bem como o tempo médio de resposta às pesquisas. Esta informação é atualizada em tempo real, através de WebSockets, no qual gera uma página sensível aos triggers de atualização de estatísticas (de modo não periódico), sem ser necessário dar refresh à página. O funcionamento dos WebSockets é detalhado mais adiante neste relatório. O trigger é chamado em métodos de procura e indexação, através da chamada da função *checkForUpdates(),* de modo a atualizar os valores das estatísticas bem como indicar a data e hora da última atualização. Implementamos também a função *getStats()* para que sejam geradas estatísticas quando o utilizador abre a página pela primeira vez.
- *HackerNewsController* – Integra a API pública do Hacker News para obter as principais histórias (top stories) relacionadas com a pesquisa efetuada pelo utilizador. A interface inclui, na página de resultados de pesquisa, um botão *Search Hacker News*, que permite ao utilizador consultar se existem top stories relevantes associadas ao/s termo/s pesquisado/s. Esta ação gera uma nova página com os resultados correspondentes às top stories encontradas. Caso existam URLs disponíveis, o utilizador pode optar por indexá-las diretamente no sistema, recorrendo ao método putNew() da classe Gateway, através de chamada RMI. O funcionamento detalhado da API do Hacker News, bem como o processo de indexação via Gateway é detalhado mais adiante neste relatório.
- *OpenAIController* - É responsável por produzir uma análise com base na query, e nas descrições dos links obtidos. Sempre que é realizada uma pesquisa, através do SearchController, a AI Gemini gera automaticamente um resumo do conteúdo apresentado. Para orientar a elaboração da análise, é utilizada uma prompt personalizada, concebida para adaptar a resposta ao contexto dos resultados. Caso o utilizador navegue para outra página de resultados, é gerada uma nova análise, atendendo aos novos resultados. O funcionamento desta API é detalhado mais adiante neste relatório.

Templates

Ao longo do projeto, são utilizados templates .html para apresentar os resultados processados pelos Controllers. Nestes, os dados são disponibilizados aos templates através da definição de atributos, como por exemplo: model.addAttribute("stats", formattedStats). Este método permite um fluxo simples e dinâmico na utilização das variáveis, facilitando a integração entre a lógica da aplicação e a camada de apresentação. De seguida, indicamos quais os templates utilizados por cada um dos controllers:

- *SearchController():* search.html, results.html
- *HomeController()*: index.html
- *AddUrlController()*: add-url.html
- *LinksToController()*: search-linksto.html, links-results.html
- *Top10Controller():* top10.html
- *WebSocketStatsController(): webstats.html*
- *HackerNewsController():* hackernews.html

Models

Para dar suporte aos Controllers, foram criadas estruturas de dados específicas para os diferentes serviços utilizados:

- WebSockets (Stats): armazena as estatísticas atualizadas do sistema, incluindo a data da última atualização e os dados formatados.
- API do HackerNews: inclui as classes IndexRequest, IndexResponse e HackerNewsStory. A primeira é responsável por solicitar a indexação de uma história; a segunda representa a resposta após a operação de indexação; e a terceira representa a notícia do HackerNews, contendo os campos necessários, bem como o método matchesQuery(), que permite verificar se o termo de pesquisa aparece no título ou no conteúdo da história.
- Serviço de análise com AI (AnalysisRequest): estrutura utilizada para enviar pedidos de análise, contendo a query e a respetiva descrição (text).

Integração do SpringBoot com Servidor RMI

Para permitir que o frontend (SpringBoot) comunique com o backend (implementado com RMI), criamos a classe *BackendClient* que é responsável por estabelecer a conexão com o servidor RMI. Através do endereço IP e da porta, utilizamos o método *LocateRegistry.getRegistry()* e *lookup()* com "Gateway" para obter a referência e guardá-la na variável *GatewayInterface gateway*, e através da função *getGateway*(), disponibilizamos essa instância aos restantes componentes do Spring (como os *Controllers*), possibilitando assim a invocação de métodos remotos. Deste modo, para aceder aos métodos da gateway nos Controllers através chamada RMI, basta ir buscar a instância do backendClient no construtor e utilizar a função getGateway() para aceder aos métodos da Gateway (ex: currentStats = backendClient.getGateway().getFormattedStatistics();). Assim, garantimos um sistema escalável, mantendo uma comunicação eficiente entre o frontend (web) e o backend (RMI).

Integração de WebSockets e RMI

O Googol utiliza WebSockets (via STOMP) para comunicação em tempo real entre o frontend (Spring Boot) e o cliente (web), utilizando RMI para interação entre o frontend e o backend. Esta abordagem permite atualizações em tempo real de estatísticas sem necessidade de refresh constante.

É configurado um endpoint para conexão inicial ('/stats-websocket') e um broker de mensagens ('/topic/stats') no WebSocketConfig. O cliente conecta-se ao endpoint e inscreve-se em /topic/stats e, o controller conecta-se via RMI utilizando a instância devolvida pelo método getGateway() do backendClient. Quando os dados são atualizados no backend (pesquisas, tempo de resposta e indexações), o frontend usa WebSocket para mostrar as mudanças em tempo real ao cliente. Isto deve-se ao facto de o WebSocketStatsController obter as estatísticas via RMI, através do método getFormattedStatistics(), e enviá-las para /topic/stats, permitindo que todos os clientes subscritos (ou seja, com a página aberta no navegador) as recebam automaticamente. Garantimos que as atualizações não são feitas de forma períodica, mas sim através de triggers (statsController.checkForUpdates()), depois de pesquisas e indexação de links, dado que influenciam o tamanho dos Barrels, os tempos de resposta e as pesquisas mais comuns.

Integração de REST WebServices

O projeto utiliza REST Web Services para interagir com APIs externas, combinando-as com a arquitetura existente. Esta integração permite funcionalidades como indexação de histórias do Hacker News e geração de análises contextualizadas com recurso à inteligência artificial, através do modelo Gemini.

O HackerNewsController() faz chamadas HTTP à Hacker News API utilizando RestTemplate, para obter IDs das top stories (‘/topstories.json’) e procurar os detalhes de cada storie (‘/item/{id}.json’). Os dados recebidos são convertidos em objetos HackerNewsStory através de Gson. As histórias que possuam um URL e contenham a query no seu título ou texto, podem ser indexadas através do método putNew() da gateway, através de conexão RMI (as stories que não tiverem url são apresentadas na página, mas não podem ser indexadas). Para isso são utilizados os endpoints ‘/hackernews/index’ (recebe IDs de stories e indexa-as) e ‘/hackernews’ (lista as stories filtradas por query).

Por outro lado, o Controller OpenAI envia resultados das pesquisas para a Gemini API via HttpClient. A prompt de pesquisa é formada por uma breve indicação, juntamente com a query, e as respetivas descrições da página, sendo que, cada página terá uma análise diferente, devido a essas diferentes informações. O endpoint ‘/generate-analysis’ retorna o resumo gerado pelo modelo AI, que é exibido no topo da página, antes dos resultados.

Testes de software

Indexar novo URL introduzido por utilizador	pass
Indexar iterativamente ou recursivamente todos os URLs encontrados	pass
Pesquisar páginas que contenham um conjunto de palavras	pass
Páginas ordenadas por número de ligações recebidas de outras páginas	pass
Consultar lista de páginas com ligações para uma página específica	pass
Resultados de pesquisa paginados de 10 em 10	pass
WebSockets	
Top 10 de pesquisas mais comuns atualizado em tempo real	pass
Lista de barrels ativos com os respetivos tamanhos do índice e tempo de resposta	pass
APIs REST 16	
Indexar URLs de fonte externa (p.ex., HackerNews) contendo os termos da pesquisa	pass
Gerar um texto contextualizado com IA (API REST de OpenAI, Gemini, Ollama, etc.)	pass