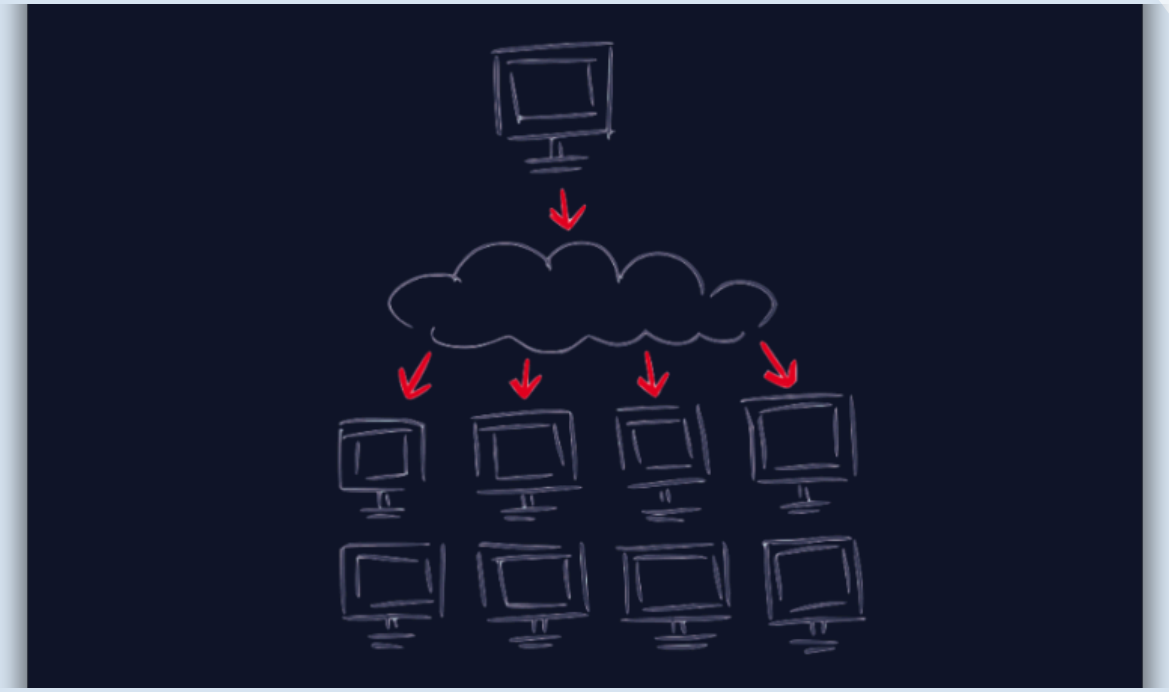
1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

dei departamento de engenharia informática

GOOGOL

Motor de pesquisa distribuído



16 de Maio de 2025

Diogo Costa, nº 2022213506
José Frota, nº 2022214661
Luana Carolina Reis, nº 2022220606

ÍNDICE

01. INTRODUÇÃO

O projeto *Googol* consiste num motor de pesquisa desenvolvido seguindo os princípios fundamentais dos sistemas distribuídos modernos, com uma arquitetura baseada em RMI/RPC (*backend*) e Spring Boot (*web development*) . Foi concebido para oferecer funcionalidades completas de indexação e pesquisa *web*, com especial ênfase na tolerância a falhas e processamento paralelo. A arquitetura adotada reflete uma abordagem prática aos desafios de distribuição de dados, balanceamento de carga e recuperação após falhas, mantendo ao mesmo tempo uma *interface* simples para o utilizador final.
O sistema é composto por quatro componentes principais:

Componente	Descrição
Gateway	Servidor intermédio
Barrels	Armazenamento do índice invertido, replicado
Downloaders	*Crawlers* que processam páginas *web*
WebClient	Interface *web* do utilizador

02. ARQUITETURA DO SISTEMA WEB DISTRIBUÍDO

Funcionamento

O sistema *Googol* organiza-se nestes quatro componentes principais, que comunicam entre si através de Java RMI, formando uma arquitetura do tipo cliente-servidor multicamada.

A *Gateway* serve como ponto central de coordenação, recebendo todos os pedidos dos clientes e distribuindo-os pelos diversos *Barrels*.

Estes últimos armazenam o índice invertido, particionado alfabeticamente em duas metades: A-M e N-Z, com cada partição tendo réplicas idênticas, de forma a garantir redundância.

Os *Downloaders* operam como *workers* paralelos, responsáveis pelo *crawling* efetivo das páginas *web*. Utilizam a biblioteca "Jsoup" para extrair não apenas o conteúdo textual, mas também os metadados mais relevantes, como títulos e descrições, bem como todos os *hyperlinks* presentes em cada página.

O Web*Client* oferece uma interface única e funcional, permitindo tanto a pesquisa por termos/URLs como a administração básica do sistema.

A comunicação entre estes componentes segue um padrão bem definido. Quando um utilizador adiciona uma nova URL através do *WebClient*, este pedido é encaminhado para a *Gateway*, que por sua vez coloca o URL numa *queue* central. Os *Downloaders*, a trabalhar em paralelo, consomem desta fila, processam as páginas correspondentes e atualizam os *Barrels* através de chamadas RMI.

Um mecanismo de *reliable multicast* garante que todas as réplicas de cada partição recebam as mesmas atualizações.

O projeto adota ainda uma arquitetura distribuída modular baseada no padrão MVC (*Model-View-Controller*), projetada para garantir escalabilidade, resiliência e separação clara de responsabilidades. No núcleo dos sistema, o *WebController* atua como controlador principal, coordenando requisições HTTP e intermediando a comunicação assíncrona com o *backend* através do gateway RMI (*IGoogolGateway*).

1. *Gateway*

- Recebe pedidos de *Client* (pesquisas, adição de URLs).
- Distribui pesquisas pelos *Barrels* (A-M ou N-Z), por *Round-Robin*.
- Implementa *failover*: se um *Barrel* falhar, usa o redundante.

2. *Barrels* (A-M e N-Z)

- Armazenam o índice invertido (HashMap<String, Set<Page>>).
- Replicação via *reliable multicast* (atualizações são propagadas entre *peers*).
- Partição do índice, onde cada *Barrel* cobre metade do alfabeto, A-M ou N-Z.

3. *Downloaders*

- Processam URLs em paralelo, usando *jsoup*.
- Extraem texto, *links* e metadados (título, descrição).
- Atualizam os B*arrels* via RMI.

4. ***WebClient (Constituição)***

- **Página Principal / *Index***
  - Pesquisas por palavras.
  - Pesquisa por URLs (verificar *links* que apontam para ele próprio).
  - Adicionar URLs à *queue* para serem processados.
  - Pesquisa por termos da "Hacker News".
- **Páginas de pesquisa**
  - Mostra os resultados (links, descrição, título) de cada página.
  - API do "Hugging Face" para breve descrição sobre a pesquisa.
- **Página de estatísticas**
  - Apresenta estatísticas periodicamente e de imediato, quando se faz uma pesquisa por palavra.

Integração do sistema Web

- **Camada de Visualização**: Utiliza templates "Thymeleaf" (exemplos: index.html, search.html) para renderização dinâmica das páginas descritas acima.

- **Comunicação em Tempo Real**: Implementa *WebSockets* (configurados via *WebSocketConfig*) para notificações instantâneas de estatísticas e resultados de pesquisas, complementados por APIs REST para interações síncronas.

- **Integração Externa:** Conecta-se a serviços como o "Hacker News" via *RestTemplate*, garantindo processamento assíncrono de feeds.

03. INTEGRAÇÃO DE SPRING BOOT COM SERVIDOR RMI

Integração Spring Boot e Serviços RPC/RMI

A interoperabilidade entre *Spring Boot* e o serviço RPC/RMI é implementada através de uma camada de abstração que une a interface *web* aos módulos distribuídos do *backend*. O *WebController* inicializa uma conexão segura com o *Gateway* RMI (*IGoogolGateway*), atuando como intermediário para operações remotas como "searchWord" e "putNew".

Mecanismo de Comunicação

- **Invocações remotas:** Métodos do *gateway* RMI são encapsulados em chamadas assíncronas, garantindo que bloqueios não afetem a responsividade da *interface*.
- **Endpoints REST Híbridos:** Rotas como "/processTopStories" combinam o consumo de APIs externas (via *RestTemplate*) com a invocação de métodos RMI, permitindo processar e redistribuir dados (ex.: feeds do "Hacker News") para o armazenamento.
- **Gestão de ciclo de vida:** O *Spring Boot* gere automaticamente a reconexão em falhas de rede e a serialização/deserialização de objetos, simplificando a lógica de integração.

04. INTEGRAÇÃO DE WEBSOCKETS COM SPRING BOOT

A integração de *WebSockets* com *Spring Boot* no projeto foi concebida para oferecer atualizações em tempo real aos clientes e também periódicas, tornando a interface *web* dinâmica e responsiva. A configuração dos *WebSockets* é centralizada na classe *WebSocketConfig*, que habilita a comunicação baseada no protocolo STOMP e define *endpoints* como "/my-websocket", permitindo que clientes estabeleçam conexões persistentes com o servidor.

A primeira mensagem obtida por parte do W*ebSocket* é realizada por uma função "fetchStats" presente no ficheiro "app.js". Esta função envia uma mensagem vazia para sinalizar ao servidor *web* que pretende mostrar as estatísticas mais recentes enviadas pelo *MessagingController*. As seguintes mensagens podem ter origem de técnicas de *fetch* como "Poll" e "Long Poll", sendo respetivamente provenientes desse mesmo *MessagingController*, que envia mensagens quando pedidas pelo servidor (no "app.js" define-se com um período de 5 segundos) ou pelo *WebController* que, após uma pesquisa, envia a mensagem imediatamente.

Ao processar essas mensagens, utiliza o gateway RMI (*IGoogolGateway*) para obter estatísticas atualizadas do sistema, recorrendo a chamadas remotas ao *backend*. Os dados recuperados são então transmitidos para todos os clientes conectados, por meio do tópico "/topic/stats", garantindo que todas as *interfaces* recebam as informações mais recentes de forma instantânea.

Essa arquitetura tira partido do modelo de publicação/assinatura dos *WebSockets*, permitindo que múltiplos clientes sejam notificados simultaneamente sobre alterações no estado do sistema, como resultados de pesquisa ou estatísticas de uso.

05. INTEGRAÇÃO DE REST WEB SERVICES

A integração de *WebServices* REST no projeto é fundamental para garantir a comunicação eficiente tanto com serviços externos quanto para expor funcionalidades internas do *backend* de forma padronizada.

Utilizando o *Spring Boot*, o sistema implementa *endpoints* REST, como o "/processTopStories" no *WebController*, que utiliza o cliente *RestTemplate* para enviar pedidos HTTP (POST) a APIs externas ("Hacker News").

A API de IA é integrada no projeto por meio de uma chamada REST no arquivo "analysis.js", que utiliza o modelo "llama-3.3-70b", hospedado na "Hugging Face", para gerar análises contextuais baseadas nos resultados de pesquisa.

A integração de *WebServices* REST no projeto, utilizando *Spring Boot* e clientes como *RestTemplate*, torna o sistema distribuído mais eficiente e preparado para diferentes plataformas.

06. FUNCIONALIDADES IMPLEMENTADAS

Funcionalidade	Descrição
Indexação de URLs	*Clients* adicionam URLs manualmente; *Downloaders* processam-nos.
Pesquisa por termos	Devolve páginas que contêm todas as palavras pesquisadas.
Ordenação por *PageRank*	Resultados ordenados pelo número de links recebidos.
Consulta de *links* para uma página	Lista todas as páginas que apontam para um URL específico.
Estatísticas em tempo real	Mostra tamanho da fila, *Barrels*, tempo médio de resposta e *top* 10 de pesquisas.
Processar *Top Stories* do "Hacker News"	Na sequência de uma pesquisa, vão-se buscar URLs às *top stories* do "Hacker News" que contêm os termos pesquisados.
Análise Contextual	Análise textual baseada nos termos da pesquisa e nas citações curtas dos resultados da pesquisa, através de API "Hugging Face".

07. INTERFACE WEB

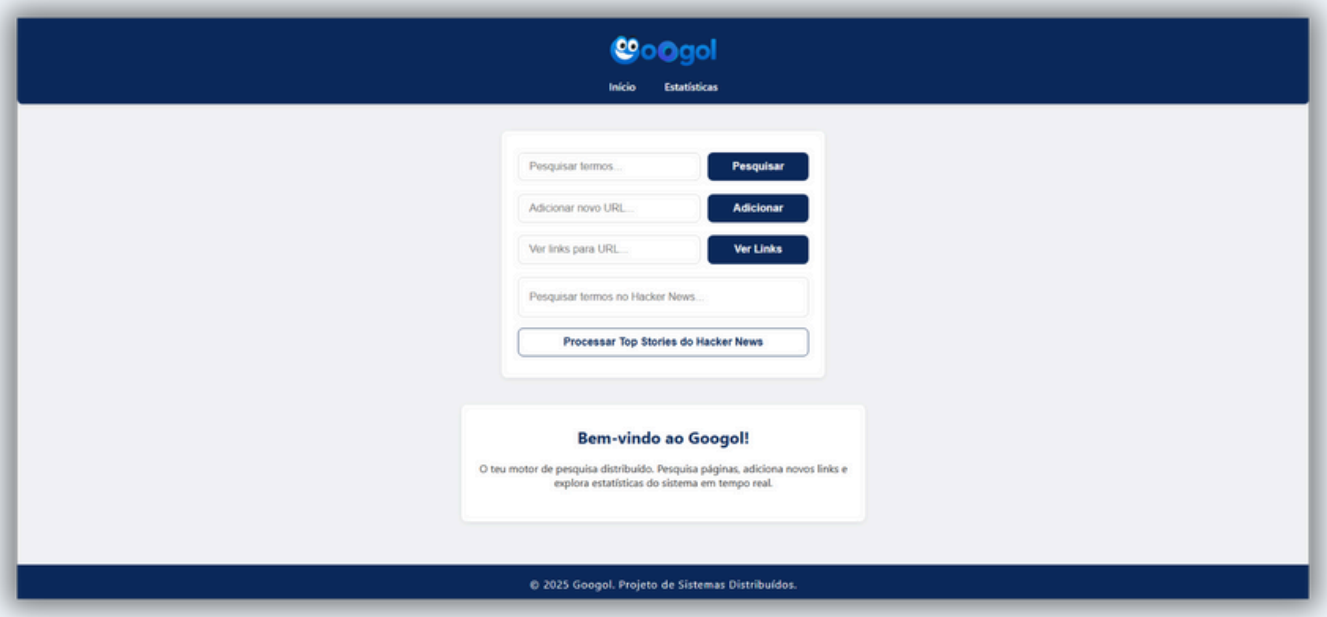


Fig. 1 - Página inicial do “Googol”

URL added successfully: https://pt.wikipedia.org/wiki/Wikip%C3%A9dia:P%C3%A1gina_principal

Fig. 2 - Adição de URL para indexação

Processed URLs matching the query: [https://openai.com/index/introducing-codex/, https://economics.mit.edu/news/assuring-accurate-research-record]

Fig. 3 - Processamento de top stories no “Hacker News”

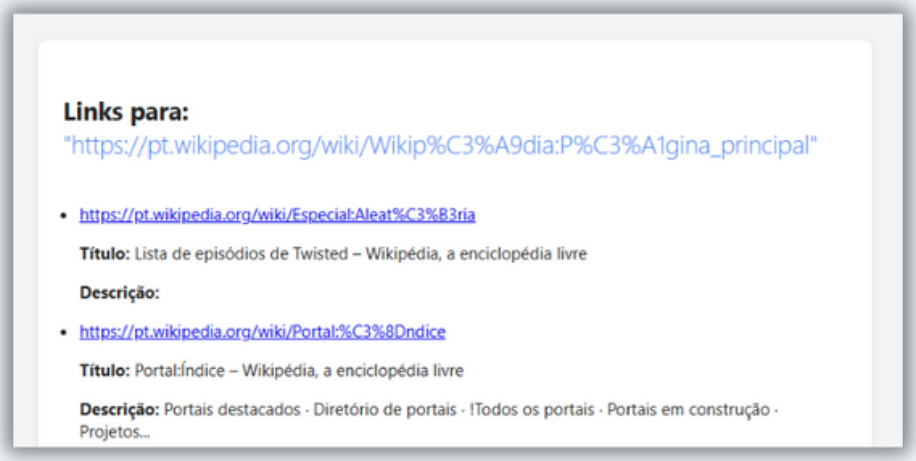


Fig. 4 - Ver *links* para um URL

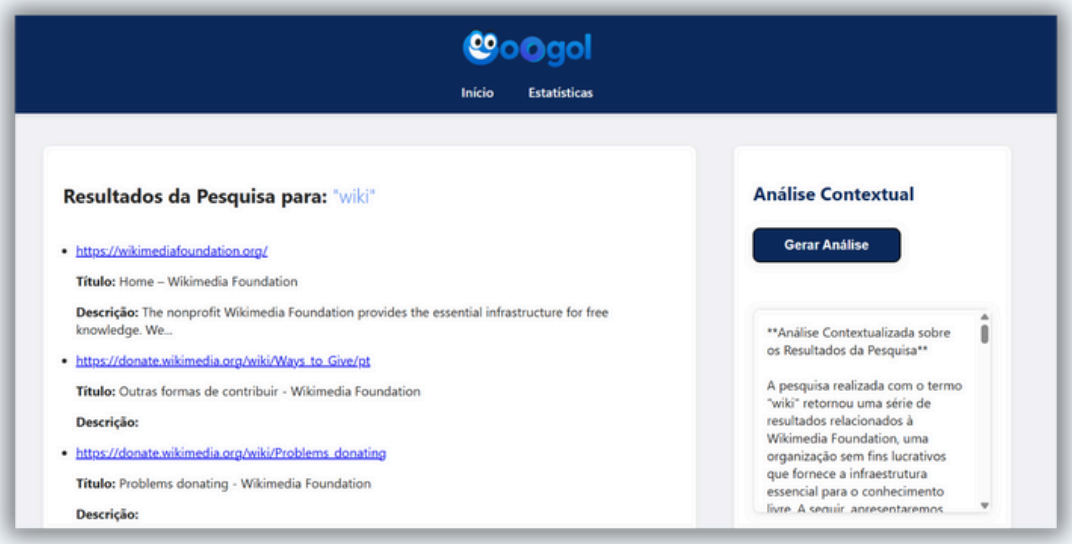


Fig. 5 - Resultados de pesquisa e análise contextual

Fig. 6 - Estatísticas do sistema, em tempo real

08. DISTRIBUIÇÃO DE TAREFAS

Meta 1

Diogo Costa	Indexação, Partição e Redundância
José Frota	Indexação, Páginas Redirecionadas e Stats
Luana	Indexação, Pesquisa de termos e PageRank

Meta 2

Diogo Costa	"Hacker News" e Https
José Frota	Endpoints *WebController* e *WebSocket*
Luana	API "Hugging Face" e *frontend*

Nota: Todos os elementos do grupo testaram e foram ajustando a solução.

09. TESTES REALIZADOS

Resquisitos Funcionais

Cenário

- Adicionar URL manualmente
- Pesquisa com múltiplos termos
- Falha de um *Barrel* AM
- Downloader processa *links*
- Recuperação após *crash*

Resultado

- URL é indexado e aparece em pesquisas ✅
- Devolve interseção de páginas relevantes ✅
- *Gateway* usa *Barrel* AM2 sem interrupção ✅
- *Links* são extraídos e adicionados à fila ✅
- *Barrel* recarrega índice do ficheiro ✅

WebSockets

Cenário

- *Top* 10 pesquisas em tempo real
- Lista de *Barrels* Ativos
- Partição dos índices

Resultado

- Atualizado periodicamente ou de imediato ✅
- Atualizado periodicamente ✅
- Índices particionados com os seus valores ✅

APIs REST

Cenário

- Indexar URLs de fonte externa
- Gerar um texto com IA

Resultado

- "Hacker News" indexa segundo termos ✅
- Texto gerado pela API com sucesso ✅

10. DIAGRAMA DE COMPONENTES

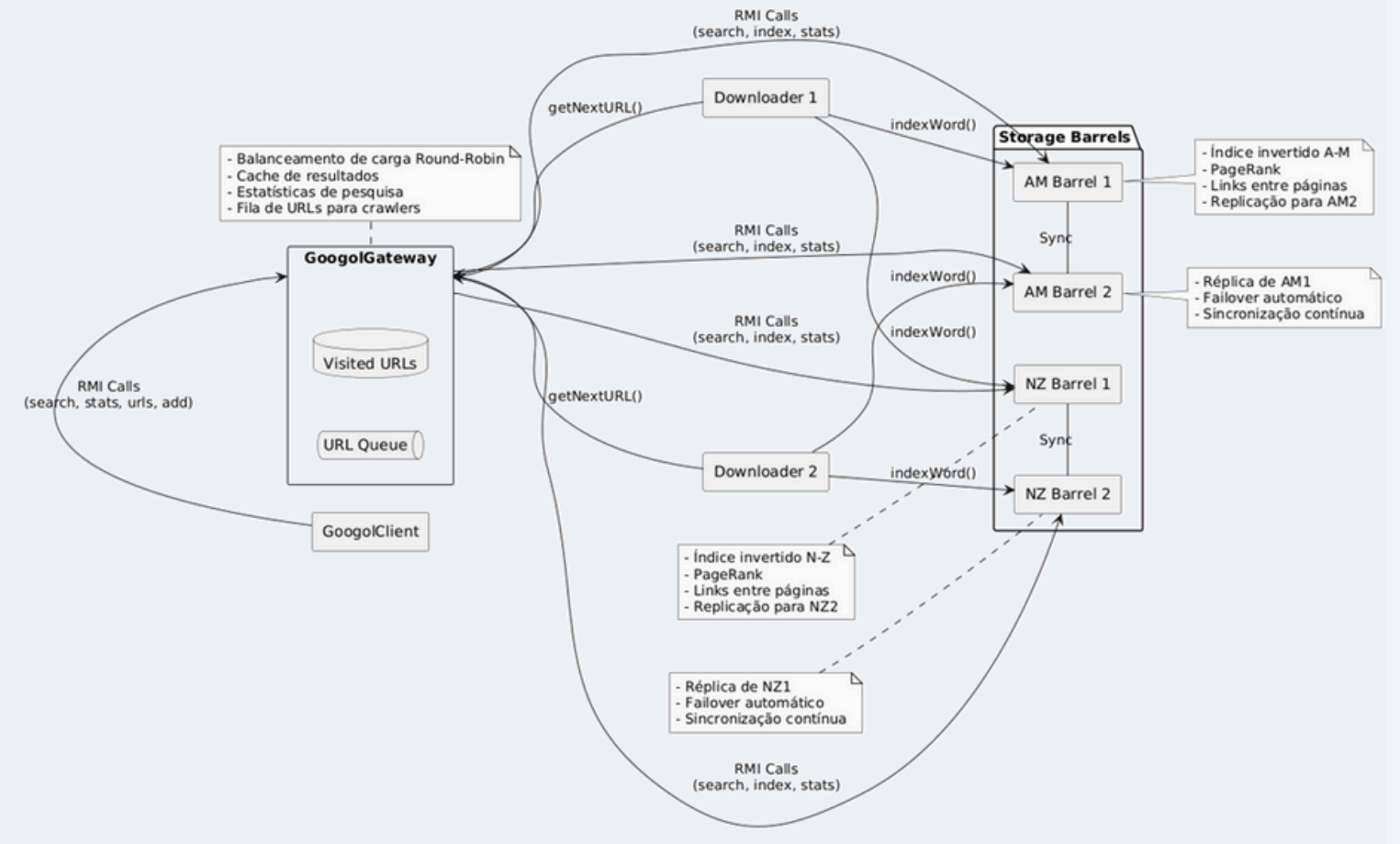


Fig. 7 - Diagrama do sistema implementado na meta 1

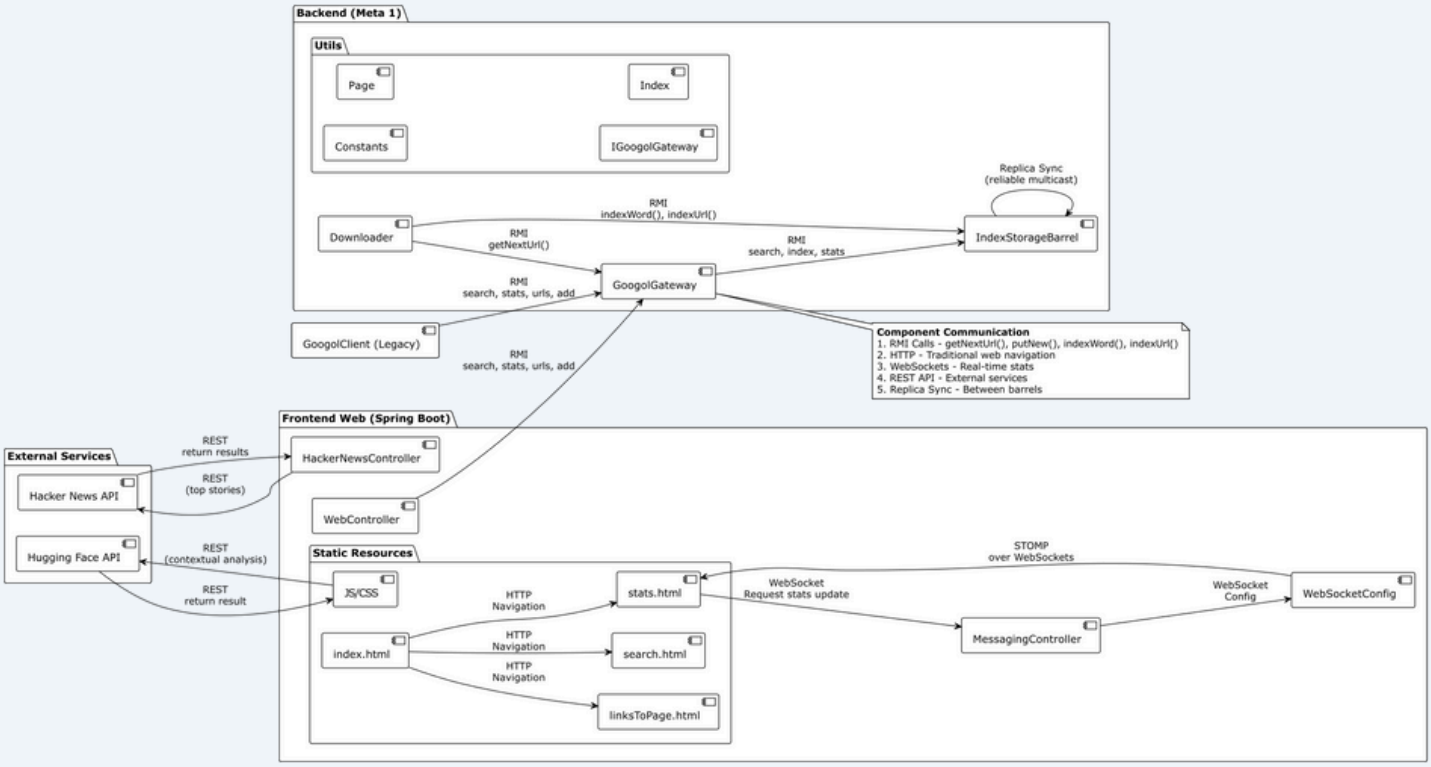


Fig. 8 - Diagrama do sistema implementado na meta 2

11. CONCLUSÕES

Depois da primeira meta, o nosso sistema distribuído estava consolidado e pronto para ser reproduzido num servidor *Web*. No final desta meta 2, podemos dizer que estamos satisfeitos com o resultado alcançado.

Aplicámos conceitos novos de REST API, e comunicação com *webSockets*. Os diversos *endpoints* que existem na nossa *web app*, provenientes tanto da API da "Hugging Face", da "Hacker News" e dos *endpoints* originais, foram implementados com sucesso e apresentados na nossa *web app*.

Nos *webSockets*, as mensagens foram também bem implementadas, enviando de forma periódica ou direta. Ambas as técnicas de "Poll" e "Long Poll" foram usadas.