Googol: Motor de pesquisa de páginas Web

Sistemas Distribuídos 2024/25 — Meta 2

Resumo

Este projeto tem como objetivo criar um motor de pesquisa de páginas Web. Pretende-se que tenha funcionalidades semelhantes aos serviços Google.com, Qwant.com e Ecosia.org, incluindo indexação automática (*Web crawler*) e busca (*search engine*). O sistema deverá apresentar diversas informações relevantes sobre cada página, tais como o URL, o título, uma citação de texto e outras que considere importantes. Ao realizar uma busca, obtém-se a lista de páginas que contenham as palavras pesquisadas ordenada por relevância. Os utilizadores podem sugerir URLs para serem indexados pelo sistema. Partindo desses URLs, o sistema deve construir o índice recursivamente/iterativamente para todas as ligações encontradas em cada página.

1 Objetivos do projeto

No final do projeto cada estudante deverá ter:

- Desenvolvido uma interface Web para a aplicação Googol.
- Integrado a interface Web com a aplicação desenvolvida na primeira meta.
- Aplicado tecnologias de programação para a Web, *e.g.*, Spring Boot ou FastAPI.
- Seguido a arquitetura MVC para desenvolvimento Web.
- Aplicado WebSockets para comunicar assincronamente com os clientes.
- Integrado a aplicação com serviços REST externos.

2 Visão geral

Nesta segunda meta do projeto, os estudantes irão criar um *frontend* Web para a aplicação Googol. Esta nova interface irá possibilitar que os utilizadores acedam ao serviço a partir de quase qualquer dispositivo com Internet no planeta, sem necessitar de instalação de software cliente. Como a interoperabilidade é um requisito muito importante, utilizadores Web deverão aceder à mesma informação que os utilizadores na aplicação desktop. Para tal, o servidor Web deverá usar o servidor RPC/RMI desenvolvido na Meta 1.

Os utilizadores deverão ter as mesmas funcionalidades através da aplicação Web que estavam disponíveis na Meta 1. Portanto, a interface Web deverá indexar URLs, pesquisar páginas, etc. Como o aspeto interativo é muito importante na Web, a aplicação deverá mostrar algumas alterações em tempo real, nomeadamente as informações gerais sobre o sistema (downloaders e barrels ativos, etc.). Como os utilizadores estão cada vez mais exigentes, técnicas menos robustas como meta-refresh e iframes ocultas não serão consideradas.

Finalmente, as aplicações de hoje em dia são integradas entre si para se conseguir funcionalidade mais rica. Através de APIs REST, irão integrar a vossa aplicação com a API do Hacker News e da OpenAI (ou alternativas equivalentes à escolha). Na sequência de uma pesquisa, deverá ser possível ir buscar URLs às "top stories" do Hacker News que contenham os termos pesquisados no Googol.

3 Arquitetura

A Figura 1 mostra a arquitetura geral do projeto. Os elementos adicionados em relação à Meta 1 referem-se à segunda meta do projeto. O servidor Web deverá ligar-se por RPC/RMI ao servidor de dados, garantindo a interoperabilidade com os clientes da primeira meta.

Devem desenvolver uma aplicação Web que corra num servidor HTTP e que atue como um cliente do servidor RPC/RMI. Os utilizadores irão usar Web browsers para se ligarem ao servidor Web e pedirem páginas através do protocolo HTTP. Para melhorar a experiência de utilização poderão ponderar fazer alguns dos pedidos via AJAX em vez de carregar a página toda. A comunicação em tempo real para o browser deverá ser construída usando WebSockets.

4 Interface Web

Pretende-se criar uma aplicação Web que disponibilize as mesmas funcionalidades da Meta 1 através da Internet. Usando uma arquitetura MVC, deverão implementar os seguintes requisitos funcionais usando Spring Boot com Thymeleaf (Java) ou FastAPI (Python):

1. **Indexar novo URL.** Um utilizador deve poder introduzir manualmente um URL para ser indexado. Esse URL será então visitado (assim que possível) por um Downloader (*Web crawler*) e ficará associado, no índice invertido, às palavras que forem encontradas no texto.

2. **Indexar recursivamente ou iterativamente todos os URLs encontrados.** O indexador automático deve visitar os URLs que forem encontrados em páginas previamente visitadas. Sugere-se a utilização de uma fila de URLs para este efeito (processo que é, em si, iterativo).

3. **Pesquisar páginas que contenham um conjunto de termos.** Qualquer utilizador deve poder realizar uma pesquisa. Para tal, o motor de busca consulta o índice

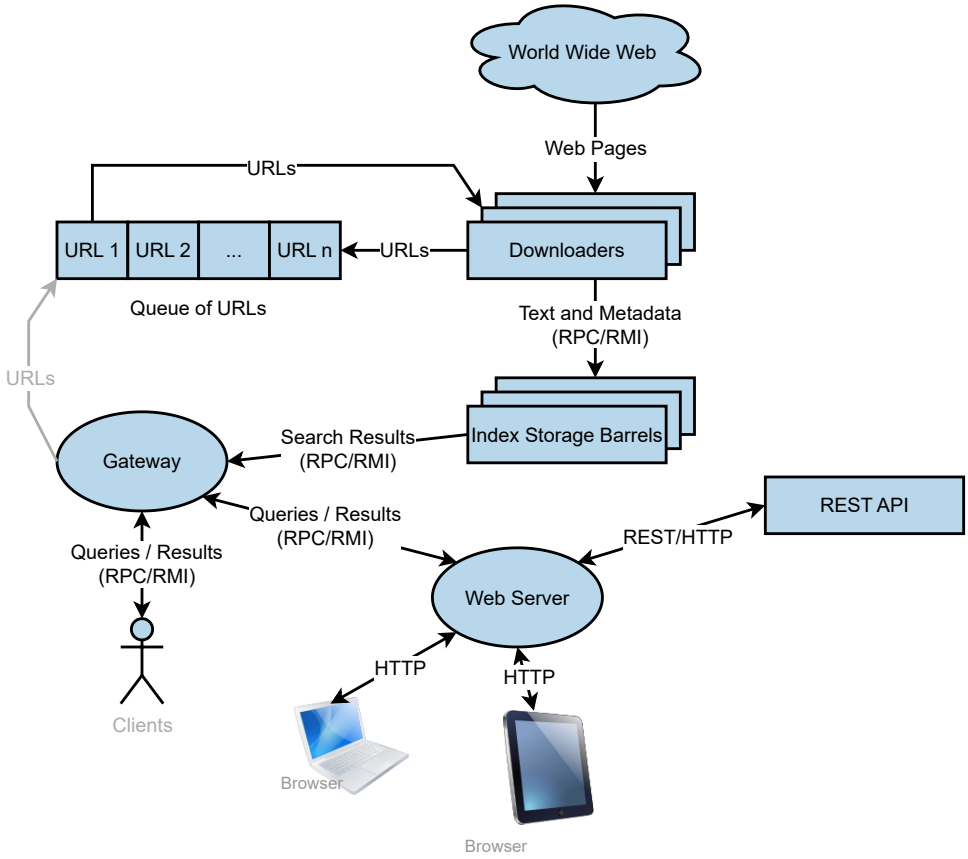

Fig. 1: Arquitetura da aplicação, incluindo os componentes da Meta 1.

invertido e apresenta a lista de páginas que contenham *todos* os termos da pesquisa. Para cada resultado da pesquisa, deve mostrar o título da página, o URL completo e uma citação curta composta por texto da página. Os resultados da pesquisa devem ser agrupados de 10 em 10.

4. **Resultados de pesquisa ordenados por importância.** Os resultados de uma pesquisa (funcionalidade anterior) devem ser apresentados por ordem de relevância. Para simplificar, considera-se que uma página é mais relevante se tiver mais ligações *vindas de* outras páginas. Assim, o indexador automático deve manter, para cada URL, a lista de outros URLs que fazem ligação para ele.

5. **Consultar lista de páginas com ligação para uma página específica.** É possível saber, para cada página, todas as ligações conhecidas que apontem para essa página. Esta funcionalidade pode estar associada à funcionalidade de pesquisa (por exemplo, uma opção associada a cada resultado).

6. **Estatísticas atualizadas em tempo real.** Todos os utilizadores têm acesso a informações gerais sobre o sistema. Esta informação será atualizada apenas quando

houver alterações (sem refrescamento periódico). Pretende-se saber o estado do sistema, designadamente as 10 pesquisas mais comuns, a lista de Barrels ativos com os respetivos tamanhos do índice, e o tempo médio de resposta a pesquisas medido pela Gateway descriminado por Barrel (em décimas de segundo).

7. **(Grupos de 3 alunos) Partição do índice.** O índice invertido está particionado em (pelo menos) duas metades: uma contendo as palavras iniciadas pelas letras A–M e outra contendo as palavras iniciadas pelas letras N–Z. É necessário agregar resultados de pesquisas que envolvam Barrels distintos.

5 Notificações em tempo real

De forma a que as páginas da aplicação sejam atualizadas em tempo real (server push), deverão usar WebSockets para fazer *push* de informação para o cliente assim que esteja disponível. Deverão usar WebSockets para:

- **Atualização de informações gerais em tempo real.** Todos os utilizadores têm acesso a informações gerais sobre o sistema (na *frontpage* ou possivelmente numa página separada). Esta informação será atualizada apenas quando houver alterações (sem refrescamento periódico). Pretende-se saber o estado do sistema, designadamente as 10 pesquisas mais comuns, a lista de Barrels ativos com os respetivos tamanhos do índice, e o tempo médio de resposta a pesquisas medido pela Gateway descriminado por Barrel (em décimas de segundo).

6 Integração com serviço REST

Este projeto deverá ser integrado com o Hacker News e com a OpenAI, ou qualquer alternativa equivalente que utilize REST. As documentações da APIs estão disponíveis em `https://github.com/HackerNews/API` e `https://platform.openai.com/docs/api-reference/chat` permitindo construir as duas funcionalidades pretendidas através de REST, designadamente:

- **Indexar URLs das top stories que contenham os termos da pesquisa.** Na sequência de uma pesquisa do Googol um utilizador deve poder solicitar a indexação dos URLs das "top stories" do Hacker News que contenham (no texto) os termos da pesquisa efetuada. Em vez das "top stories", aceita-se como alternativa que um utilizador com conta no Hacker News peça ao Googol para ir buscar todas as suas "stories" e indexar todos os seus URLs do Hacker News.

- **Gerar com a API da OpenAI uma análise contextualizada.** Usando a API de inteligência artificial generativa da OpenAI (serviço de "chat completions") deverá ser acrescentada à página de resultados do Googol uma análise textual baseada nos termos da pesquisa e/ou nas citações curtas dos resultados da pesquisa. Poderão igualmente usar uma qualquer alternativa à OpenAI, desde que seja usada uma API REST.

6.1 Relatório

Devem reservar tempo para a escrita do relatório no final do projeto, tendo em conta os passos anteriores. Devem escrever o relatório de modo a que um novo colega que se junte à equipa de desenvolvimento possa perceber a solução criada, as decisões técnicas e possa adicionar novos componentes ou modificar os que existem. O relatório pode ser inteiramente escrito em Javadoc/Docstrings no código-fonte apresentado pelos estudantes. Deve incluir:

- Arquitetura de software detalhadamente descrita. Deverá ser focada a estrutura de models, views, controllers, processos, threads e sockets usados, bem como a organização do código.
- Detalhes sobre a integração do Spring Boot / FastAPI com o Servidor RPC/RMI da primeira meta.
- Detalhes sobre a programação de WebSockets e a sua integração com o servidor RPC/RMI.
- Detalhes sobre a integração com os serviços REST.
- Descrição dos testes feitos à plataforma.

7 Entrega do projeto

O projeto deverá ser entregue na plataforma InforEstudante num arquivo ZIP contendo o código fonte completo do projeto. A ausência deste elemento impedirá a avaliação do projeto. O ficheiro ZIP deverá conter um MD/HTML/Javadoc/Sphinx/Docstrings com o relatório. O relatório deve seguir a estrutura fornecida, dado que a avaliação irá incidir sobre cada um dos pontos. Esse arquivo deverá também conter um ficheiro README.md ou .txt com toda a informação necessária para instalar e executar o projeto sem a presença dos alunos. Projetos sem informações suficientes, sem quaisquer destes elementos, que não compilem ou não executem corretamente não serão avaliados.