Personal Website Introduction

Author: Yunfei Luo

July 26, 2021

This website was mainly designed for representing the Software Engineering and open source projects that I've done or participated in.

There are reports and documentaions of the projects. Moreover, there are blogs, include the reflections and discussions on the papers of the related fields that I've read. The blogs also include some of my learning notes on the significant concepts in the engineering (application) field.

There is a micro-search-engine implemented in the backend to enable the search functionalities. Since the articles are stored structured in a retrospective way, they are easy to be retrieved and be able to concisely displayed to the readers.

The website welcome all readers who are interested in the topics of these articles.

The website was originally deployed entirely on AWS, with domain name https://yunfeiluo.com. (It will expired in 2022.). The new version of the website is hosted on GitHub Pages, with Search Engine API deployed on AWS Lambda (https://yunfeiluo.github.io).

Personal Website, Requirements

Author: Yunfei Luo

July 26, 2021

---

1. Functionalities.

1.1. Welcome page

When user navigate to the website, there is a clean-style page, with brief introduction, links to the documents, and nice pictures. There is also a footer at the bottom of the webpage, containing the linkedin, github, and my contact information. This footer will be shown in all the webpages on the website.

1.2. Navigation to documents

User click on the link with either:
i) navigate to software engineering documentations
ii) navigate to reports of open source projects
iii) navigate to blogs
iv) navigate to search area! :)
For the first three cases, the list of corresponding documents will be shown with a clean style. When user click on the title, the documents need to be shown explicitly. AND, there need to be a back button to let the user navigate back to the list page.

1.3. Search for relevant document(s)

User click into the search area. They will search with key words: blogs, reports, image process, language model, what is search engine, what is this, to be or not to be, hey what's up! ...

The retrieved list will be shown in the same form mentioned in 1.2. The search bar will keep at the top of the webpage. The user could either click the document for reading (also same scenario in 1.2.), or try another query, or close the browser.

2. Non-Functionalities.

2.1. Development Schema: Waterfall, CI/CD

Since this is a personal project, waterfall pattern is a proper way to keep the progress going forward stably and correctly. Continuous Integration enable the consistency of codes among local development branch and the remote master branch.

2.2. Front-end Tool(s): React

The render functionality of React enables the concise response the quick and temporary click actions. The JSX syntax enables the effective development, and makes the code more readable.

2.3. Back-end Tool(s): Flask, AWS

Flask is used for local testing, i.e. setup a local server, and test the interactions between front-end

and back-end.

AWS provide cloud services that enables the quick deployment. The services that will be used include: S3 bucket for server, Lambda functions for handling events, API Gateway for the communication between front-end and back-end, and Relational Database Service for indexing the articles.

2.4. Version Control: Github

A widely recognized tool for version control. Github offer clean interface to trace the issues, and push/pull requests.

Personal Website, High Level Design

Author: Yunfei Luo

July 26, 2021

---

1. Overview.

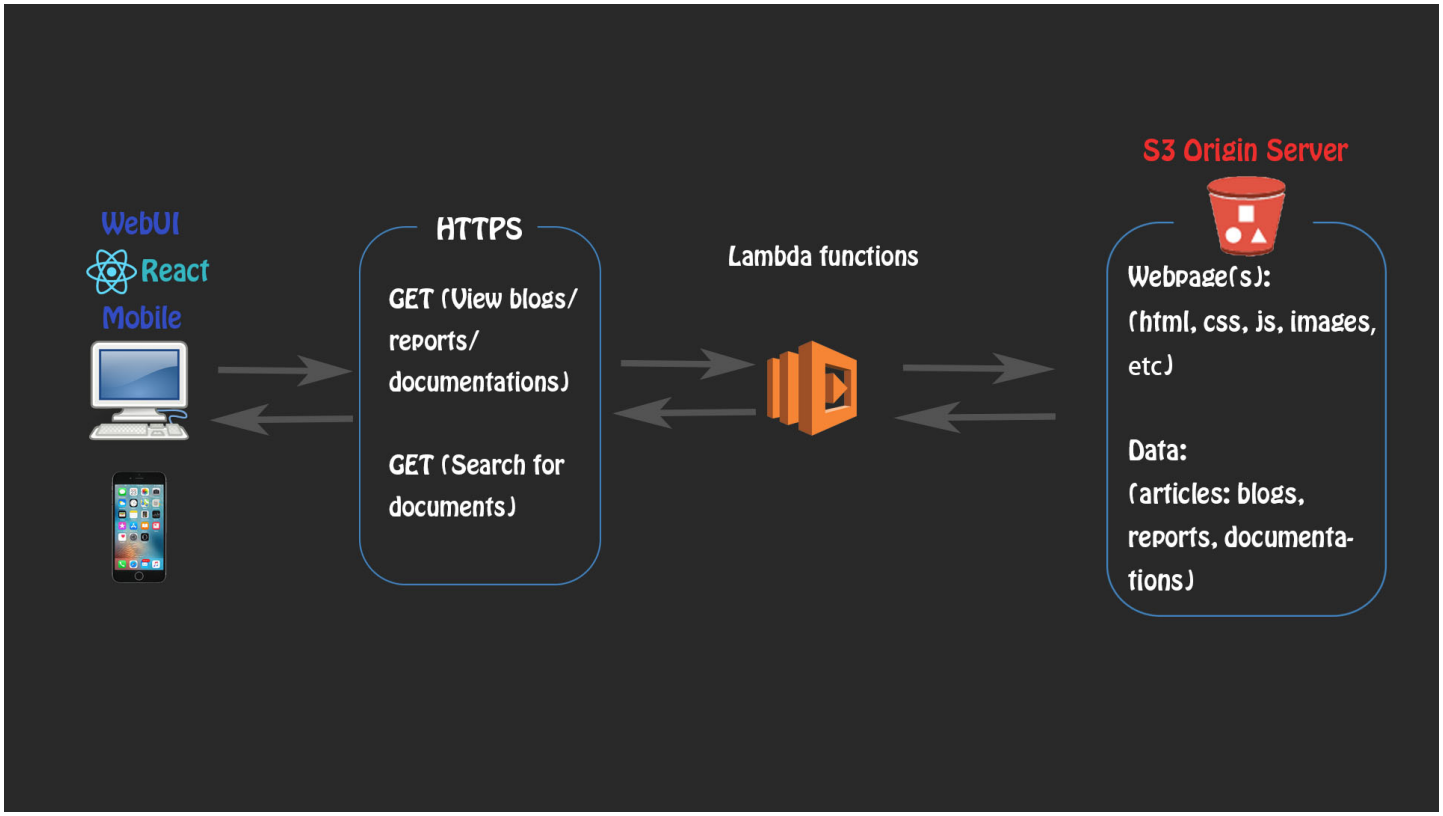

Figure 1. Entire Architecture

2. Front-end Schema.

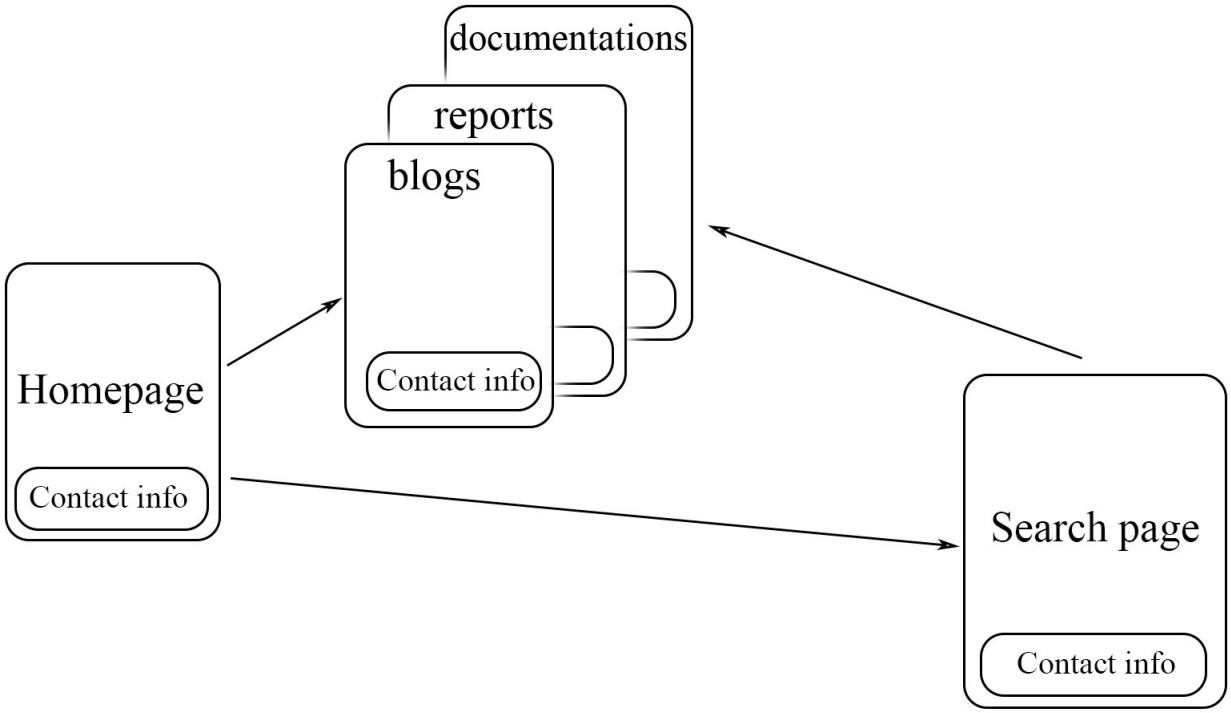

Figure 2. Frontend Architecture

Homepage is the node connecting the website to with users. Search page is the node connecting the frontend to the backend. Both Homepage and Search page navigate to the article pages.

Every pages return to the homepage.

Every pages contain Contact info in the footer.

3. Back-end Scheme.

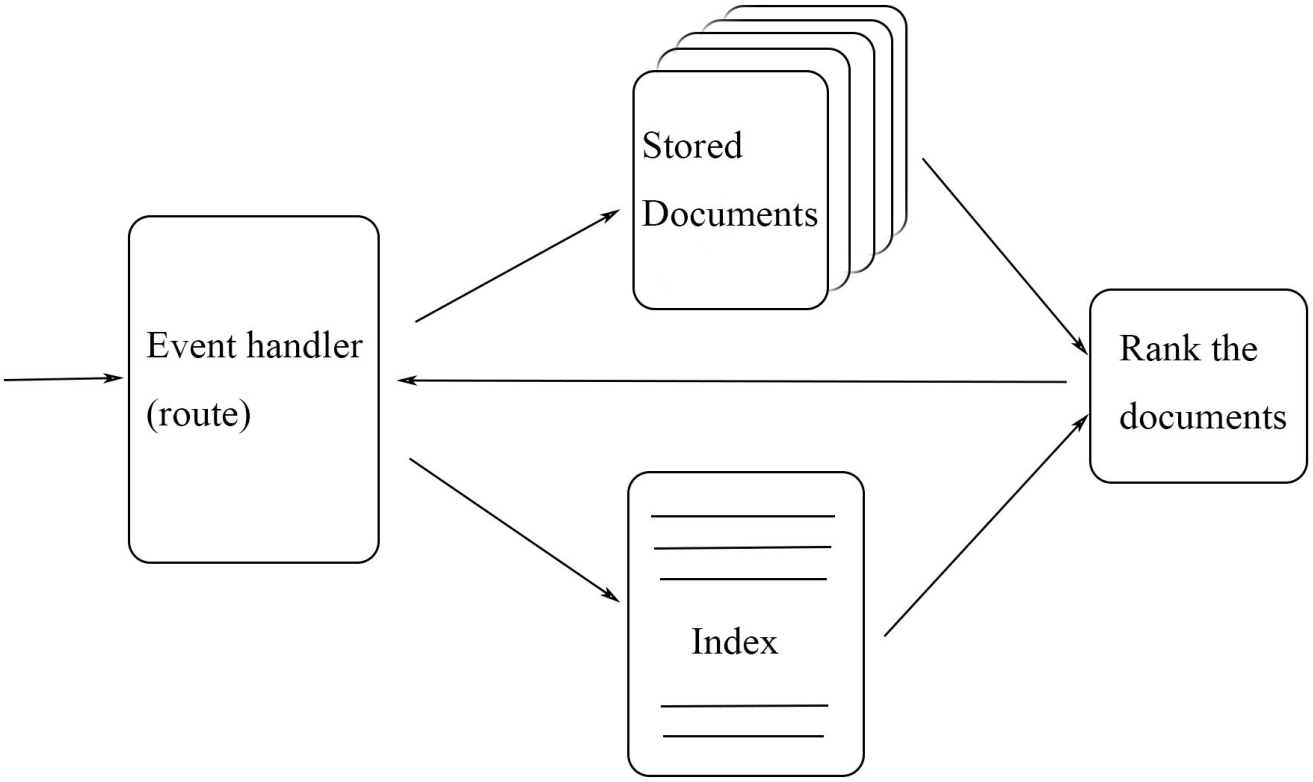

Figure 3. Backtend Architecture

Event Handler is the node receiving requests from the frontend. It will run the micro search engine, and return the result back to the frontend.

Within the micro search engine, the Store Documents and Index contains the data needed for retrieving the documents. The retrieval model rank the documents based on the stored data, and return the result back to the Event handler node.

4. Database Schema.

Article
id (num)
type (str)
title (str)
tags (list(str))
path (str)
summery (str)
docs (list(Article))

Figure 4. table for storing Article object

Each article has a unique id. Type attribute could be {blogs, reports, documentations}. Path attribute specifies where the document stored. Docs attribute points to other documents that is within the scope of project of the Article object.

Personal Website, Low Level Design, Search Engine

Author: Yunfei Luo

July 26, 2021

---

1. Overview.

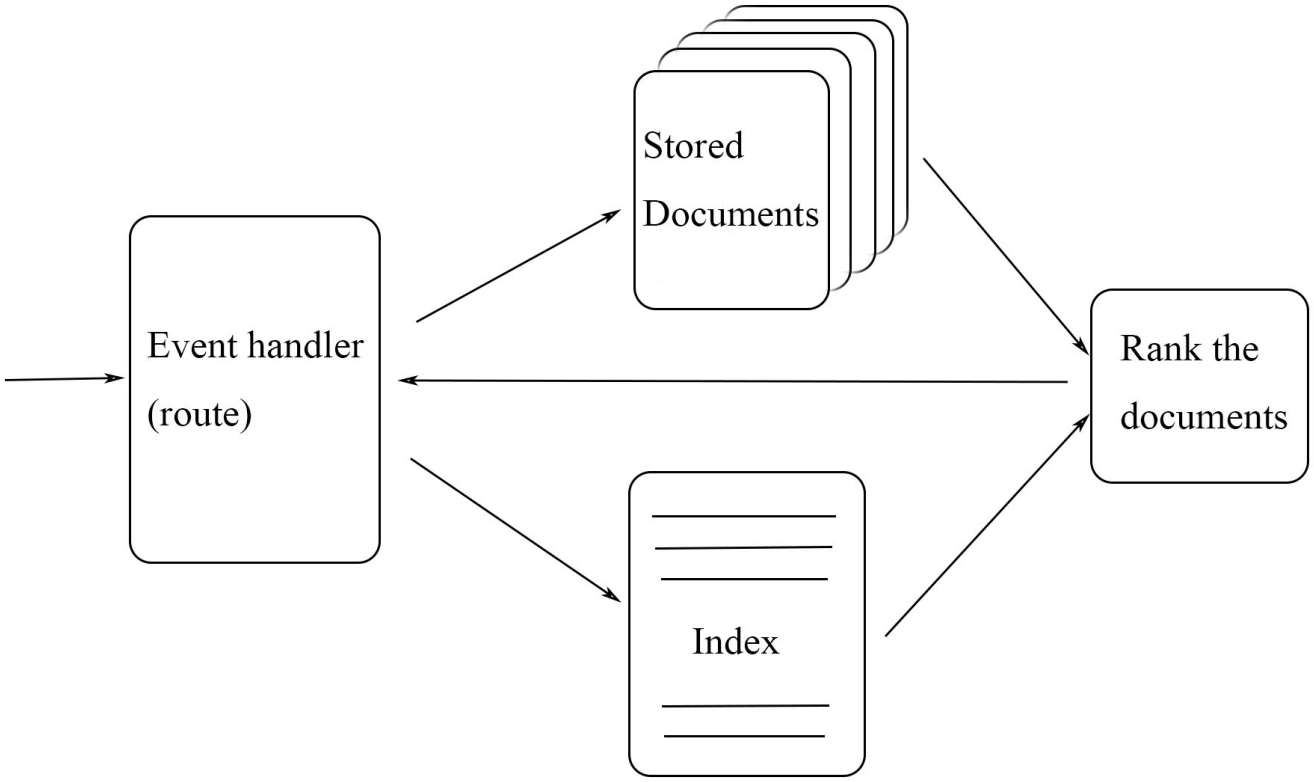

Figure 1. Micro Search Engine Architecture

2. Pre-processing.

2.1. Processing Documents.

This is independent with the interaction between frontend and backend. This process need to be done periodically, or whenever there are new articles added to the database.

The process follows steps:
i) tokenize, all letter to lower case, and remove dot between abbreviation words.
ii) stemming, use Snowball stemmer, also called Porter 2.
iii) stopword removal, remove non-significant words, use the concise list from python package: nltk.

2.2. Processing Queries.
When ever there is a query request coming from frontend, the query terms will first be pre-processed, then the querying functions will be executed. The processing steps are the same as steps for processing documents, see 2.1.

3. Indexing (Inverted Index).
Indexing will be mainly used by the retrieval model to rank the documents given the query terms. Inverted Index is a inverted list, that contains the necessary information for each stored terms. More specifically, the inverted index is defined as:

$$map : term \mapsto (map : document\_id \mapsto list(positions))$$

For example, term *learning* occur in document 1 and 3. It is the first and fifth word of document 1, and second and forth word of document 2, then the index for this term would looks like:

$$learning \mapsto \{document_1 \mapsto [1, 5], document_3 \mapsto [2, 4]\}$$

Then the term frequency in a document is $length(term.doc\_id)$, i.e. the total length of the list of positions in document with specified id. The collection frequency is the length of the list for a term.

4. Retrieval Models.
The key element for ranking the documents given the query terms. The following are the options of algorithms.

4.1. BM25. (formula information reference from: Search Engines, Information Retrieval in Practice, by W.B. Croft, D. Metzler, T. Strohman, 2015)
BM25 (BM stands for Best Match) is a well-known probabilistic retrieval model that not only take the document term frequency into account, but also consider the query term frequency. In the micro search engine, we will use the most common form of BM25, with no reference information. More specifically, the score for a document given the query terms is calculated by:

$$\sum_{i \in Q} \log(\frac{N - n_i + 0.5}{n_i + 0.5}) \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)qf_i}{k_2 + qf_i}$$

Where Q is the set of terms in the query terms, N is the total number of documents we have, n_i is the document frequency, i.e. number of documents that contain term i. f_i is the term frequency in the document, and qf_i is the term frequency in the query terms. The weighting parameters k_1, k_2, K are set empirically (by science). k_1 and k_2 determine the importance of document term frequency and query term frequency respectively. K is a normalized term, determined by:

$$K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$$

Where b is a empirical parameter, dl is the document length, and $avdl$ is the average length of all the documents we have.

We could set the magic parameters to $k_1 = 1.1, k_2 = 10, b = 0.6$.

4.2. Query Likelihood. (formula information reference from: Search Engines, Information Retrieval in Practice, by W.B. Croft, D. Metzler, T. Strohman, 2015)
Query Likelihood is a well-known probabilistic retrieval model depends on language model. The

smoothing techniques we use is Dirichlet smoothing. More specifically, we calculate the score for a document by:

$$\alpha_D P(q_i|D) + \alpha_D P(q_i|C)$$

where $P(q_i|D)$ is the probability of query term i occur in document D, and $P(q_i|C)$ is the probability of query term i occur in the entire collection C. α_D is the Dirichlet smoothing coefficient determined by $\alpha_D = \frac{\mu}{|D|+\mu}$, where μ is set empirically. The final formula is:

$$\frac{f_{q_i,D} + \mu \frac{c_{q_i}}{|C|}}{|D| + \mu}$$

where $f_{q_i,D}$ is the term frequency in document D, and c_{q_i} is the term frequency in entire collection C. We could set $\mu = 1000$.

5. Deployment

The preprocessing and indexing steps are running offline. There will be two json files generated, one is the inverted index, another one is the document index that is a map from documents' id number to their actual name.

The query time interaction is running rely on the AWS lambda function. The following are the deployment steps:

a. Create a lambda function, and upload the code to the lambda. The query can be received in the *event['query']* in the lambda handler. (Remember to hit Deploy after changing the code)

b. Create an API-Gateway trigger, as a REST request.

c. Navigate to API-Gateway, create a GET and OPTION method for the lambda function.

d. Enable CORS under Actions.

e. Click the GET method, under Method Request, URL Query String Parameters, add the variable called *query*, and check the required box.

f. Under Integration Request, Mapping Template, add *application/json* in the content type. and add

$$\{"query" : "\$input.params('query')"\}$$

In the blank field. Then hit Save.

g. Under Action, hit Deploy.