Index Search

Mini Documentation

Table of Contents

Introduction

Index search is a simple search engine that uses an inverted index to search for documents. It is a simple search engine that can be used to search for documents in a collection of documents. From the collection of documents, the search engine returns the documents that contain the search query and orders them by relevance.

Use Cases

The search engine can be used in the following scenarios:

- Searching for documents in a collection of documents
- Searching for documents in a collection of documents and ordering them by relevance

Project Structure

The project is structured in such a way that it has 3 main components:

Front End

The Front End is the user interface component responsible for interacting with the search engine. It provides a user-friendly interface to add documents to the collection and perform searches efficiently.

Concierge

This is the component that is used to interact with the search engine. It is the component that is used to add documents to the collection of documents and to search for documents in the collection of documents. The concierge uses the librarian to add documents to the collection of documents and to search for documents in the collection of documents.
This component does three main goals:

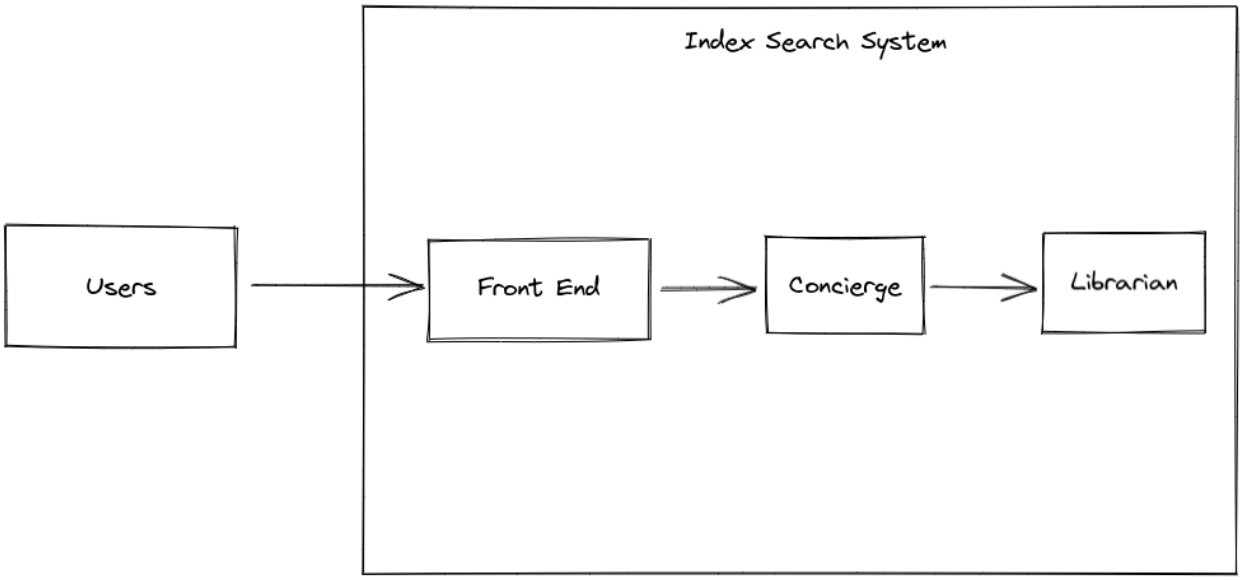
1. Accept documents (title and url of document)
2. Downloads the documents and tokenizes them
3. Sends metadata to the librarian

Librarian

This is the component that is used to create the inverted index. The inverted index is a data structure that is used to store the words in the documents and the documents that

contain the words. The indexer is used to create the inverted index from a collection of documents. This component is also used to search for documents in the collection of documents. The searcher uses the inverted index to search for documents that contain the search query and orders them by relevance.
This component has two main goals:

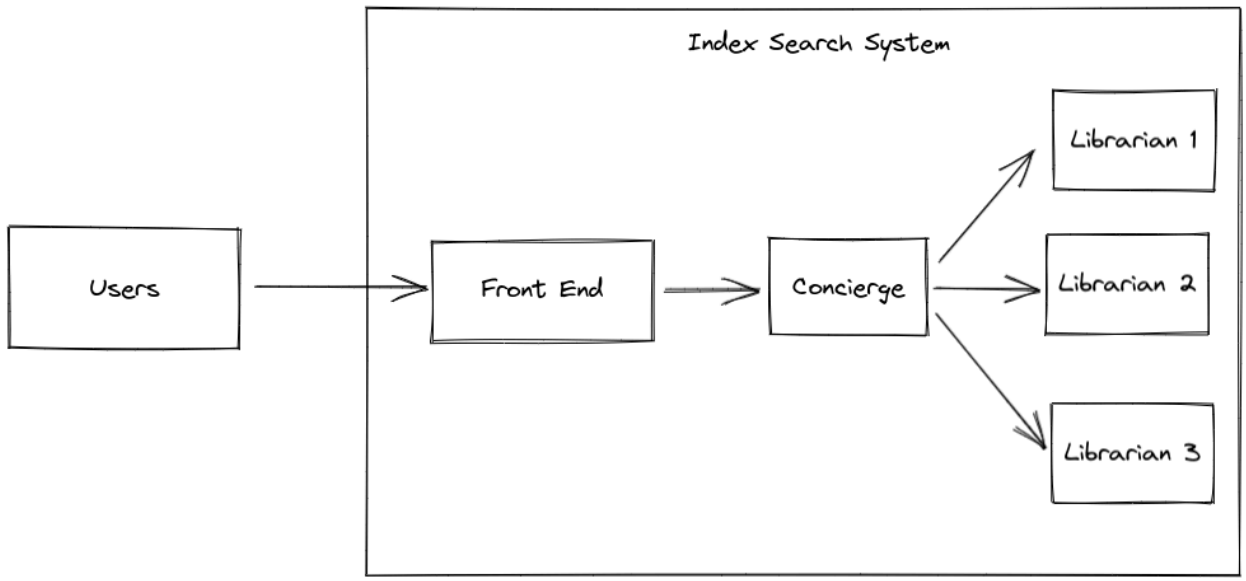
1. Accepts metadata about tokens and builds an inverted index
2. Searches for tokens in an inverted index and return results



In order to show the distributed nature of the search engine, the project is divided into two services:

- Concierge Service: This is the service that is used to interact with the search engine.
- Librarian Service: This is the service that is used to create the inverted index.
- Front End: This is the component that will be deployed on a web-server and it not really the point of attention so we will run it on the go.

And to bring about Fault Tolerance, the project is structured to have 1 concierge service and 3 librarian services. The rationale behind this is that the concierge service is the service that is used to interact with the search engine and it is the service that is most likely to be used by the users. And the librarian service is the service that is used to create the inverted index and it is the service that is most likely to be used by the concierge service. So, in order to bring about fault tolerance, the librarian service is replicated 3 times.



This structure will help us in that the load on the librarians (who will do the heavy lifting of the systems) are load balanced.

How to Use

The components / services are dockerized and there is a docker-compose.yaml file that will spawn the instances. Docker compose will instantiate the instances and in turn will notify the concierge component where to find the librarian components.

Therefore, to run the project.

1. Clone the project
2. Change directory to the root of the project
3. Install Docker
4. Install Docker compose
5. Run docker compose
6. Change directory to the root of the front-end project
7. Run the front-end project