Search Engine System (SES)

Autocomplete:

We look at the real searches that happen on SES and show common and trending ones relevant to the characters that are entered and related to your location and previous searches.

To build this sophisticated autocomplete service, we combine two powerful ideas:

- suggestion generation from database
- Most frequent user query-based suggestion generation

Search:
Our search will suggest you the list of relevant topics for your request. The Suggestion will include:

- Title
- Content (short description of the website)
- Link
- Image

Indexing:
We will process the information received from the websites and organise it in such way in order to implement super-fast response to queries. So we will implement (tokenization) hash of keywords which will be related to search information in order to easily and fast search based only on hashed keywords.

Ranking:

We will return the sorted sites the most relevant in the top. So our system will implement ranking based on following criteria:

- Best Match
- Popularity (Count of users searches)
- Quality & Usability (Count how many quality links references the page has)

List of Services

<u>Search service:</u>

- Outbound API: **GET** searching/search?word=microservices
  Receive requests on search with string parameter.
- Internal API:
  - **GET** indexing/getPages/{word}
    Make request to find the most common pages ids with this word occurrence by keywords.
  - **POST** ranking/findByIndexes with body {list of index.Id}
    Make request to receive response with most relevant pages based on previous searches requests and index best match.

Auto-complete service:

- Outbound API: **GET** auto-complete/complete?word=micro
  Receive requests on autocomplete feature with string parameter.
- Internal API:
  - **GET** indexing/getSuggestedWords/{word}
    Make request to find the most common words with the max occurrence by keywords.
  - **GET** ranking/getSuggestedSearches/{word}
    Make request to find the most common words with the max occurrence by user searches.

Indexing service

- Internal API based on Database queries:
  - GET db/sites (with pagination)
    Make request to get all info from DB after spider crawling.
  - POST db/indexes with body {id of web-site & list of keywords}
  - GET db/indexes/{word}

Ranking service

- Internal API based on Database queries:
  - GET db/searches (with pagination)
    Make request to get all user searches.
  - POST db/searche/{word}

List of technologies:

- **The Gateway (ASP.NET Core)**

The API gateway sits between the client apps and the microservices. It acts as a reverse proxy, routing requests from clients to services. It can also provide other cross-cutting features such as authentication, SSL termination, and cache.

- **Load Balancer (ASP.NET Core)**

Distributing network traffic across multiple servers effectively is called Load Balancing. To meet high volumes, it generally requires adding more servers. Load Balancer routes the client requests to the servers in an optimized way and takes care that no server is overworked. If a new server is added to the group, It will start sending requests to this new server.

- **Caching (Microsoft.Extensions.Caching.Memory)**

Our applications often call same method, again and again and fetch data from database, but sometimes, data doesn't get changed or updated in database, in that case, we can use caching to reduce database calls and gets same data directly from memory-cache.

- **Rest API (.NET Core)**
- **RPC or Event Publisher**

- **Main DB (for pages content) –> PostgreSQL**
- **DB for indexes –> MongoDB**
- **DB for searches –> MongoDB**

Architecture diagram: