Documentation
ELK Quran/Hadith search engine

Version 1.0.0

Houssam Aït Ouaziz
Ziad EL-fatih

Contents

List of Code Listings

1 Introduction

This document software manual created using LaTeX and Overleaf. It provides a nice clean format for producing this type of technical document.

It will walk you through setting up the different **Elasticsearch** indices with the provided mappings, Data loading and cleaning procedures, to the complex queries that were used as an empirical way to test the performance of both the **Hadith** and **Quran** search engine.
The tools that were mainly used in this project are **Elasticsearch**, python with certain that were put to use through jupyter notebook for dealing with data and **Elasticsearch** client and vscode was the code editor of choice when it came to building the corresponding Web app in python's framework Flask. This, hereby, will be a thorough step by step documentation to successfully running the code on your machine.

2 Setting up the environment/dependencies

First, let's start by setting up the environment and necessary packages :

- In your terminal, **cd** to the directory where requirements.txt is located(Elk-search-analysis/Elk-search-analysis).
- Activate your virtualenv.
- Run: pip install -r requirements.txt in your shell.

If the previously mentioned method works, but certain cells still don't run correctly, revert back to a new environment and use .yml file instead, use the terminal or an Anaconda Prompt for the following steps:

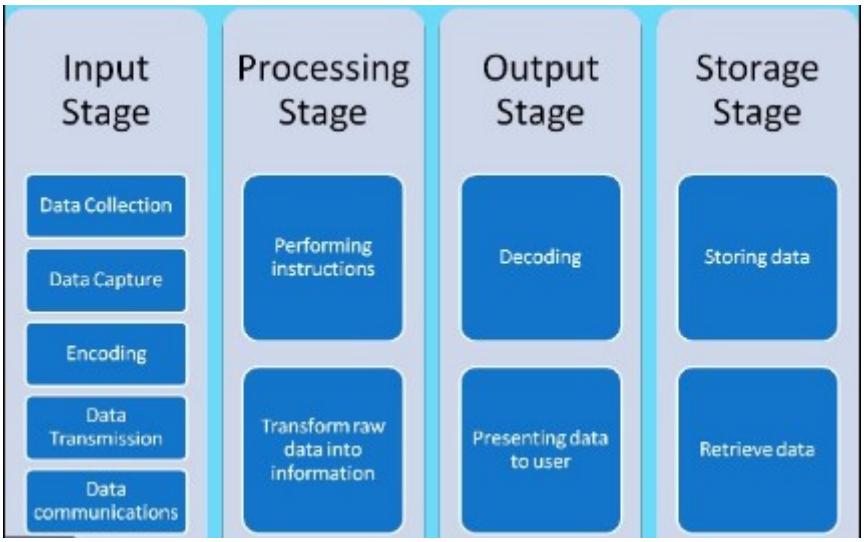
- Create the environment from the shared_environment.yml file: conda env create -f shared_environment.yml
- Activate the new environment: conda activate myenv
- Verify that the new environment was installed correctly: conda env list

3 Jupyter notebook scripts

3.1 Data processing

For the quran and tafssir data we aggregated it from different sources like tanzil project(SQL db, json files) into one JSON file format on which we will be

based for putting in place the Elasticsearch mapping, same process has been done with the Hadith data that we scraped which was in CSV format, then thoroughly cleaned before ingesting the output data into a meaningful format in a JSON file, that will be transformed and loaded into the Elasticsearch index with the provided mapping.
We processed the data according to the following cycle :



3.2 Indexing Data into Elasticsearch

This is the part where we have successfully ran the data-processing notebook, now we're on to executing the two remaining notebooks(hadith-index, quran-index) to create and feed the indices with data and be able to run all different types of queries on them.
Before we index any data we have to talk about the mapping, that acts as metadata for the data that we will index in our elasticsearch instance, also tells elasticsearch how that data is going to be read and analyzed, and what kind of analyzers and how many of them will be used for each text term of the mapping, see the code in the figure (the paths used in the token filters should be changed to config file where Elasticsearch is installed in your machine), this is just where we defined our own token filters, which include the stemmer, stopword analyzer and tokenizer that our analyzer will be based upon :

```
"analysis":{
 "filter":{
   "arabic_stop":{
     "type":"stop",
     "stopwords_path":"D:\Elastic\elasticsearch\config\merged-stopwords.txt" },
    "arabic_keywords":{
     "type":"keyword_marker",
     "keywords_path":"D:\Elastic\elasticsearch\config\keywordmarker-words.txt"
    "stop_complex_queries":{
      "type":"stop",
       "stopwords_path":"D:\Elastic\elasticsearch\config\merged-stopwords-1.txt"
   "arabic_stemmer":{
      "type":"stemmer",
     "language":"arabic" },
   "arabic_shingle":{
     "type":"shingle",
     "min_shingle_size":2,
     "max_shingle_size":3     }  },
   "analyzer":{
      "rebuilt_arabic":{
         "tokenizer":"standard",
         "filter":[
            "arabic_stop",
            "arabic_normalization",
            "arabic_keywords",
            "arabic_shingle",
            "arabic_stemmer"  ] },
       "stem_noshingle":{
         "tokenizer":"standard",
         "filter":[
            "arabic_stop",
            "arabic_normalization",
            "arabic_keywords",
            "arabic_stemmer" ] },
      "shingle_nostem":{
         "tokenizer":"standard",
         "filter":[
            "arabic_stop",
            "arabic_normalization",
            "arabic_shingle"  ]  },
      "simple": {
          "tokenizer":"standard",
         "filter":[
            "arabic_stop",
```

```
            "arabic_normalization"    ] },
        "comp_query": {
           "tokenizer":"standard",
          "filter":[
            "stop_complex_queries",
            "arabic_normalization" ]
```

Same goes for setting up the mapping of both indices, you have to change the path for the .txt files in the stopwords token filters, you can find them in the config file in the project directory, they then have to be moved to the config file in the directory where ElasticSearch is actually installed.

After putting the mapping in place, we give it as an argument for creating the index, and then process the data from the JSON file in an appropriate to be indexed by the BULK api, we used the "helpers" library in python.

4 User Interface

Make sure Flask is installed by checking

```
import flask
flask.__version__
```

If flask is not installed you should install it. Python uses pip to manage dependencies, so the command to pull Flask into our development environment is:

```
pip install Flask.
```

to run the app, cd to the directory:

```
cd /your-path-here/Elk-search-analysis/Interface
```

In windows powershell or cmd, under the same directory run the following commands:

```
set FLASK_APP=code.py
$env:FLASK_APP = "code.py"
flask run
```

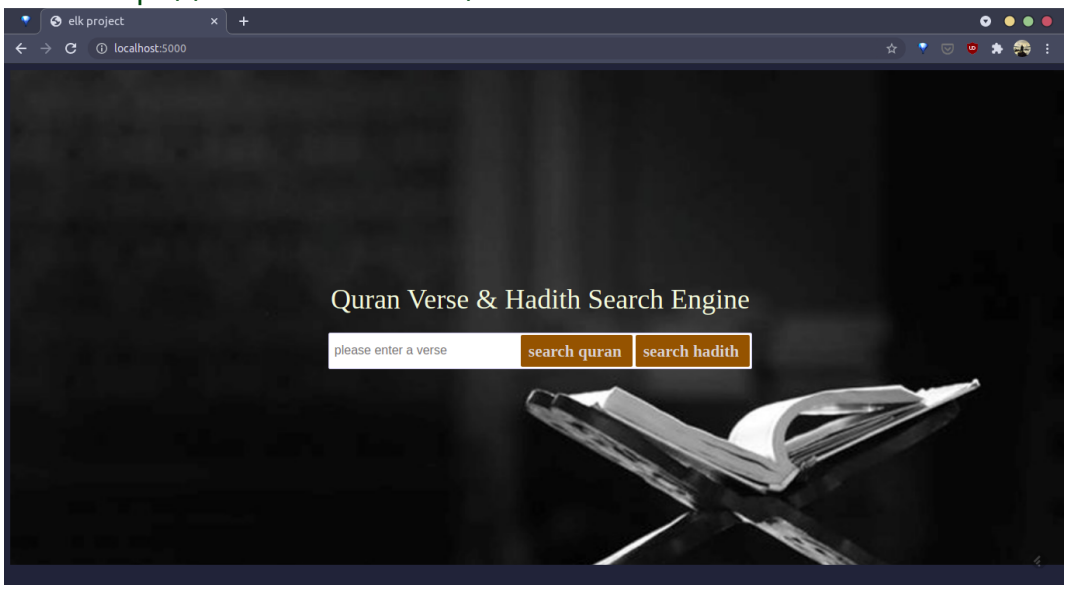
Make sure ElasticSearch server is active before running the app.

it should display something like this:

```
ziad@okcomputer:~/projects/Elk-search-analysis$ cd Interface
ziad@okcomputer:~/projects/Elk-search-analysis/Interface$ python3 code.py
 * Serving Flask app "code" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

After the server initializes it will wait for client connections. The output indicates that the server is running on IP address 127.0.0.1(localhost), which is always the address of your own computer. Network servers listen for connections on a specific port number. Since this application is running in a development environment, Flask uses the freely available port **5000**.

Now open up your web browser and enter the following URL in the address field: `http://localhost:5000/`



Navigating through the app is really intuitive, just enter what you want to search in the corresponding box and hit the desired button, you can choose whether you want to display search results from the Quran or the Hadith.