THE SCALING ON THE X-AXIS REPORT

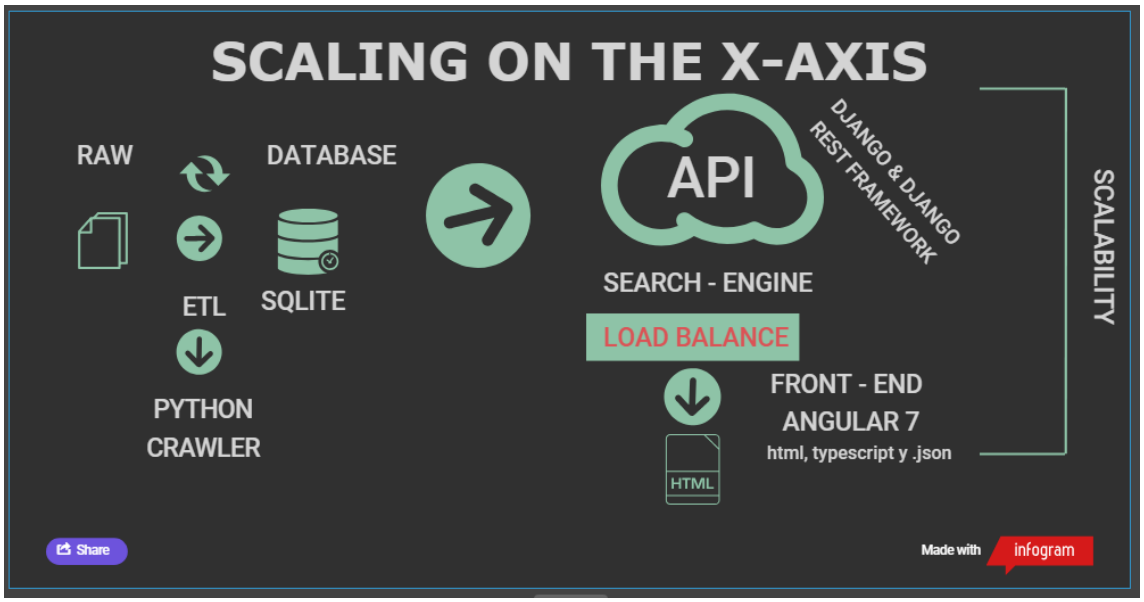
BY: EDISON LAMAR, RAFAEL GARCÍA, MIGUEL HERRANZ

DATE: 11/03/2019

This report serves as auxiliary documentation to fully understand the work done in this compulsory assignment.

For this assignment it was requested that in the face of a problem of organization of files, a specific word and / or file could be found through a specific search.
For this problem, it would be necessary to have an app that solves the dilemma proposed in an effective way. But, at the same time, it is necessary to provide it with the possibility of being modular, that is, that new future functionalities could be coupled in the design without having to modify what has already been achieved.

Well, we will start by making a brief introduction to what is going to be seen next, in broad strokes we can divide the work done in 4 parts, which will be clearly denoted. To see it more clearly, we will add a picture with the project´s design architecture:



First of all, we focused on the problem of how to organize the search and find out what you wanted to find. Then our second task was for the project to be modular; so and finally we could focus on the search logic and the Front - End that users can see. For this, within the logic of the Project, it is divided into Back - End (where we use different tools and for that reason we will subdivide them into Api and Indexation) and in Front - End.

1. Back – End

All the Back End was written in Python, however it should not suppose any kind of comprehension problem, since being an Object Oriented Language keeps the same logic of the others (excepting some cases of specific functions and the syntax in general, clearly). Still we will put several lines of code commented so that there is no doubt about the logic made.

1.1. Indexation

In order to find the logic of the Indexation you need to move to Back-End / api / engine / management / commands / etl_engine.py. In the file pointed to by this path you can find the logic that is responsible for counting words, how to get the files or their paths, insert words within them etc.
The files are extracted and their words (determined by blank spaces) are cut through the ETL process performed by Crawler.

```
Def handle(self, *args, **options):
            # r=root, d=directories, f = files
            for r, d, f in os.walk(PATH):
                for file in f:
                    file_path = self.get_file_path(r, file)
                    self.open_file(file_path)
```

This is simply the method in which the flow enters when we invoke the command. (It's like the main).

Here what we do is iterate folder by folder and file by file recursively. We access the files that is what interests us. We invoke this method to give us the absolute path of the file we access. Method that does all the logic.

```
def get_file_path(self, root, file):
            return os.path.join(root, file)
```

Method that returns the absolute path of the file x.

```
    def get_file_name(self, path):
            return os.path.basename(path)
```

Method that returns the name of the file x.

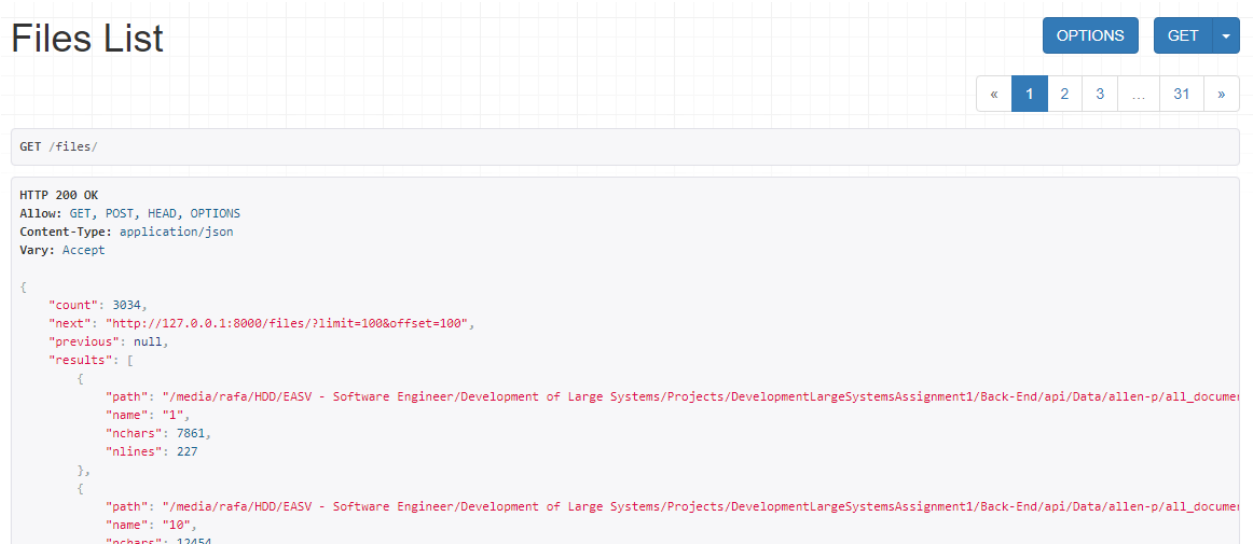
```
def open_file(self, path):
    # File properties
    lines=0
    words=0
    total_words=0
    characters=0

    # Word Properties
    words_to_insert = []

    with open(path, 'r') as all_text:
        for l in all_text:
            # Get file props.
            words=l.split()
            lines+=1
            total_words+=len(words)


            for i in words:
                # Get Words in-line
                words_to_insert.append((i, lines))
                # Get all characters per word
                characters+=len(i)

    print('Loading...\n')
    _file = self.perform_insert_file(path, self.get_file_name(path), characters, lines)
    self.perform_insert_words(_file, words_to_insert)

    print('---- File: {} ---- \n'.format(self.get_file_name(path)))
    print('Total lines: {}'.format(lines))
    print('Total Words: {}'.format(total_words))
    print('Total Characters: {}'.format(characters))
    print('-------------------------------- \n')
```

We open the file in read mode. The typical split is created to separate all the words in the file by blank spaces. The total number of words a file has is counted.

Here what we do is take all the characters that have a word and we also do the total sum of the characters that have a file

1.2. Api Rest:

Once the part of the Crawler was done, we proceeded through the Load Balance, creating a modular application was one of the initial requirements raised by the Assignment. For this task we have created an Api using Django and Django Rest Framework, in addition to that our ETL process is programmed in Python, and django is an open source web application framework written in Python. The development environment of the Api has been Visual Studio Code.



Django is a free framework which is nothing more than a collection of modules that make development easier. They are grouped together, and allow you to create applications or websites from an existing source, instead of from scratch. And Django REST framework is a powerful and flexible toolkit for building Web APIs.

In the Api we manage the data of the database in the JSON format, we pass them to the Api and it returns the values to be displayed in the Front - End

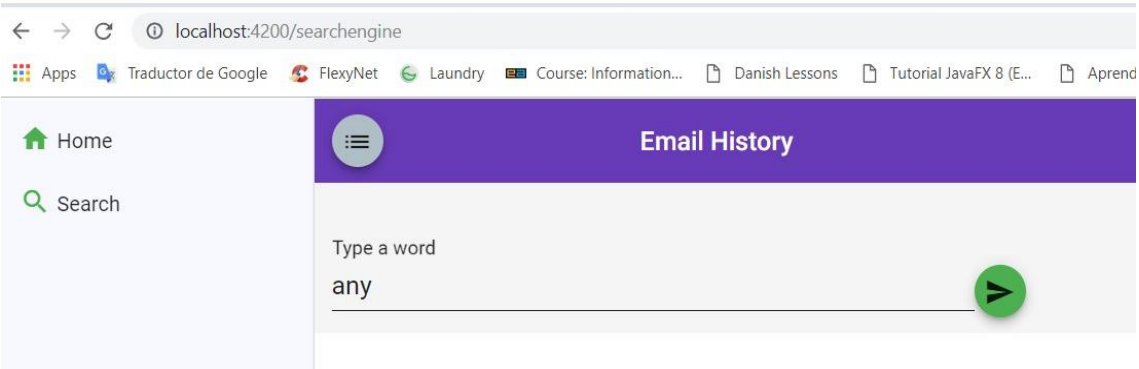
2. Front – End:

The development of the Interface has been carried out in Angular 7.

Angular is a development framework for JavaScript created by Google. The purpose of Angular is to facilitate the development of web applications and also give us tools to work with the elements of a web in a simpler and more optimal way.

Another purpose that Angular has is the complete separation between the front-end and the back-end in a web application, as in this case.

All the Front - End was developed in the WebStorm development environment, and has two modules, one common and another containing the search engine.



Finally, it is important to highlight the importance of the scalability that makes up the entire structure of the project. This application right now is designed to be 100% scalable, and in fact the next step would be this one.

You can see the rest in our Github repository in: https://github.com/edijavi/DLS-Assignment1