DATA COLLECTION ON THE INTERNET FOR SEWAGE SYSTEM CARTOGRAPHY

Project n°6 – Technical report

ARTIGUES Gabriel & LABAT Coline
Projet Industriel de Fin d'Etudes IG5
Polytech Montpellier
From 28th November 2016 to 9th February 2017

For Cart'Eaux project and Berger-Levrault company,
For the attention of Mrs DELENNE Carole, Mrs CHAHINIAN Nanée and Mr DERUELLE Laurent
Supervisor: Mrs LAURENT Anne
Examinations board composed of Mrs TOULEMONDE Gwladys and Mr CHAPELLIER Philippe

Berger Levrault
L'AVENIR EST AUX VALEURS SÛRES

IG

POLYTECH MONTPELLIER

UNIVERSITÉ DE MONTPELLIER

Table of contents

Table of figures

ACKNOWLEDGMENTS

First, we would like to express our sincere gratitude to all the Cart'Eaux project members for offering us this work on their project.

We would like to thank Carole DELENNE and Nanée CHAHINIAN for their trust, their time, their advice and their availability all along this project.

We would also thank Benjamin COMMANDRE for his availability and his help on the word analysis.

Finally, we would like to thank Anne LAURENT, our tutor, for her attentiveness, her advice and her support all along our mission.

INTRODUCTION

Cart'Eaux project aims at developing a multi-sources data collection method to merge knowledge into information in order to allow the cartography and modelling of urban wastewater and stormwater networks. It started on 20th November 2015 and it is funded by the European Union via ERDF[1] funds.

Cart'Eaux project is divided into 4 distinct parts:

1. Creation and compilation of geographical data
2. Creation of the attribute table
3. Wastewater network cartography
4. Hydraulic simulation

We are involved in action 2. Indeed, until now, relevant documents were found and retrieved manually by hydrologist experts. They saved PDF documents, html pages, pictures, etc. Then, they have to turn them into text files in order to execute a word analysis on them, and focus on it.

It will be interesting to automate this task. It will save a lot of time and energy to people. Finally, this would provide a big collection of documents related to stormwater and wastewater networks. It will enable an automated processing chain in order to perform data mining on the documents found with our solution.

This report highlights our technological choices that we made and our approach in the development of the solution. We will also provide a guide to the Cart'Eaux project members in order for them to set up the solution. Finally, we will explain in detail our code, the functions we used and the sequence of events.

[1] ERDF : European Regional Development Funds

I. TECHNOLOGICAL CHOICE

Before starting the project, we defined 3 important steps:

- Collect the results of our requests in JSON;
- Then, from the JSON, collect the documents (in PDF, html, etc.) available thanks to the links;
- Transform these documents into text documents.

To be the most efficient during the project, and in order that we do not waste time in the middle of the project in case that the chosen technology is not suitable with what happens next, we made some research on different technologies with their advantages and their drawbacks.

a. Java

> *« Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. »* ***[1]***

Java has some advantages. It is an object-oriented language, so the code can be reused easily. We can say that it is a robust language, it can manage the memory automatically and its errors. It is also a multithreaded language: it can execute several tasks at the same time. The application developed can easily run on desktop and does not need a server.

However, we can note some inconveniences for this language. First, it is an interpreted language, so the speed of the program will not be really fast. Then, the syntax can be criticized. Indeed, some formulations are inherited from C or C++ like the placement of the semicolons.

b. Java Enterprise Edition

> *« The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. Java EE extends the Java Platform, Standard Edition (Java SE). »* ***[2]***

As it is written in Java, it is possible to execute the platform code on every operating system like Linux, Microsoft Windows, and Mac OS X. JEE presents a lot of tools for issues of enterprise in terms of computer science: databases access, communications between Java applications for example, realized thanks to frameworks like Spring, Hibernate, Apache CXF, etc. It requires a server to execute applications (e.g. websites). As every type of technologies, JEE has some disadvantages. This is a mono-language so JEE can only understand Java language, and the architecture is complex, it is necessary to have a long time to learn this technology.

c. Python

> *« Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. [...] Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. »* ***[3]***

As JEE, it can be executed on many operating systems like Linux, Microsoft Windows, and Mac OS X. The inconvenience we can notice is its slowness and the error detection during the execution (not during the compilation).

However, we have found a lot of tools, provided by libraries, which can easily transform PDF documents, html source code, etc. into a text document. Moreover, we found a Python module which can be used for fetching URLs.

d. Conclusion

Thanks to this research, we have chosen to use Python language. Indeed, it allows us to use tools to convert documents into text documents. As we did not really know how much time we need to develop the solution, we have guessed it will be better to use Python.

We chose to use Python 2.7.x instead of Python 3.x because some libraries, that are essential for our program, are not compatible with Python 3.

II. PRELIMINARY CONFIGURATION AND RUNNING

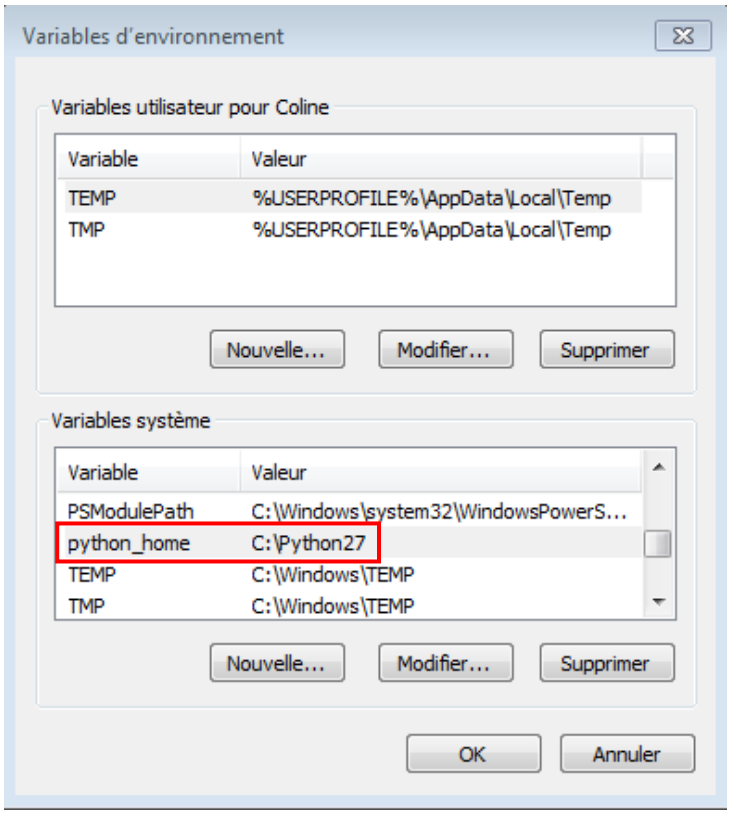
The first important step is to download Python 2.7.x from the website https://www.python.org/downloads/release/python-2713/ in order to start the installation. Then, download the folder containing:

- Our solution program (`search_python.py`)
- The keywords documents (`keywords.txt` and `type_documents.txt`)
- The `requirements.txt` file generated thanks to the command `pip freeze`
- The `get-pip.py` file.

a. Environment work creation

1. On Windows

Once Python is installed and the folder containing the solution is downloaded, you have to set up some environment variables. To achieve it, it is essential to go on "Panneau de configuration" → "Système et sécurité" → "Système". On the left menu bar, click on "Paramètres système avancés". Then, a window appears on which you have to choose "Variables d'environnement...". On the new window, it is important to add a new system variable named as the following.



(If you have changed the installation folder, you have to give the correct address.)

Then, find the system variable called `Path` and click on Edit. Add the following text to the end of the variable value: `;%python_home%\;%python_home%\Scripts`

Now, verify a successful environment variable update by opening a new command prompt window and typing `python` from any location.

```
C:\Users\Coline>python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

As we use packages to perform some actions, you have to install a command to make the package setting up easier. Download the `get-pip.py` file to a folder on your computer. On a command prompt window, navigate to the folder containing `get-pip.py` and run `python get-pip.py` to install `pip`.

For the penultimate step, after downloading the `requirements.txt` file, on a command prompt window, you have to navigate to the folder containing it and run `pip install -r requirements.txt` in order to install all the necessary packages.

Finally, the last command to run is `pip install pdfminer` after navigating to the folder containing the solution program named `search_python.py`.

2. On Mac OSX

For Mac OSX, in a command prompt, run the both commands below:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
sudo python get-pip.py
```

Finally, on a command prompt you have to navigate to the folder containing the `requirements.txt` file and run `pip install -r requirements.txt` then `pip install pdfminer` in order to install all the necessary packages.

b. Subscribing to the API

In order to perform searches on the web and retrieve the results, we use the Google Custom Search tool linked with their JSON API. First we will describe the functioning of these tools, and then we will explain in detail how to set them up.

1. Presentation

The JSON[2] Custom Search API uses REST[3], however this REST is different from the "traditional" REST. Indeed, the JSON Custom Search API provides access to a service instead of providing access to a resource. Consequently, it provides an URI[4].

The use of this API is not possible without having created a Custom Search Engine before. Google offers the possibility to create a search engine that we can configure to adjust the ranking of the results, customize the look of the search results or so that it searches only the contents of one website. Originally, it is designed to be integrated to the user's website or blog. But here, we are creating one in order to use the JSON API that lets us retrieve search results from Google Custom Search programmatically.

Here is how it works. We send an HTTP GET request to the URI to be able to retrieve results of a particular search. The format for the JSON Custom Search API URI is:

https://www.googleapis.com/customsearch/v1?key=***KEY***&cx=***CX***&q=***QUERY***

"key", "cx" and "q" are required parameters. The "key" query parameter is used to identify the application. It is an API key, we will see in detail later how to obtain it. Then we have to inform our Custom Search Engine ID ("cx") to specify on which search engine we want to perform the search. This is where the creation of our Custom Search Engine comes into play: without this parameter we cannot use the API. We will see in the next part how to get this ID. Finally, we use the "q" query parameter to specify the search expression.

Other optional parameters exist and they are of 2 types:

- API-specific parameters that are used to define properties of the search. Some are used to restrict results on a date, on a particular country, etc. They can be found here: https://developers.google.com/custom-search/json-api/v1/reference/cse/list
- Standard query parameters to define technical aspects of the request like the data format for the response for example.

If the request is well formatted and that no error happened, the server responds with a 200 OK HTTP status code and the response data in JSON is returned to the user. In this document, the user can find 3 types of information: metadata describing the search, metadata describing the custom search engine, and search results.

[2] JSON : JavaScript Object Notation
[3] REST : Representational State Transfer
[4] URI : Uniform Resource Identifier

The structure of the response body can be found in appendix 1 and a detailed description of each property is available here: https://developers.google.com/custom-search/json-api/v1/reference/cse/list#response

2. How to use Google API

Here is the procedure to follow in order to have an API key for Google Custom Search, indispensable tool to run our program.

Creation of a Google Account

First the user needs a Google account. If he does not already have one, he must follow this link https://accounts.google.com/SignUp?hl=fr , fill in the form and follow the steps.

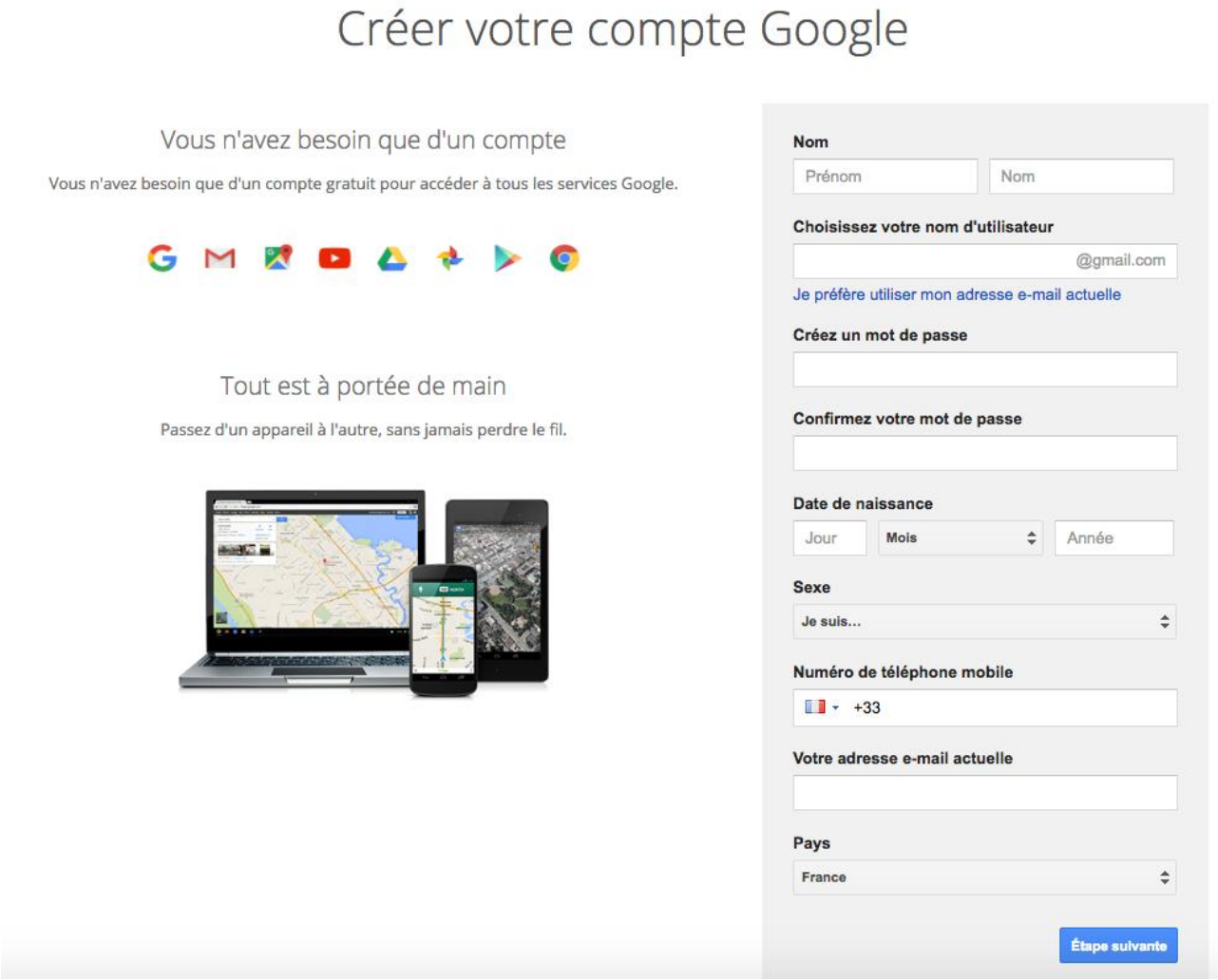


Figure 1 – Screenshot of a Google account creation form

Getting an API key

Once you are registered with your Google account, go to the Google Developer Console available through this link: https://**console.developers.google**.com/?hl=fr

You reach a web page where you can find all the APIs offered by Google. Select the Custom Search API, located in "Autres API populaires".

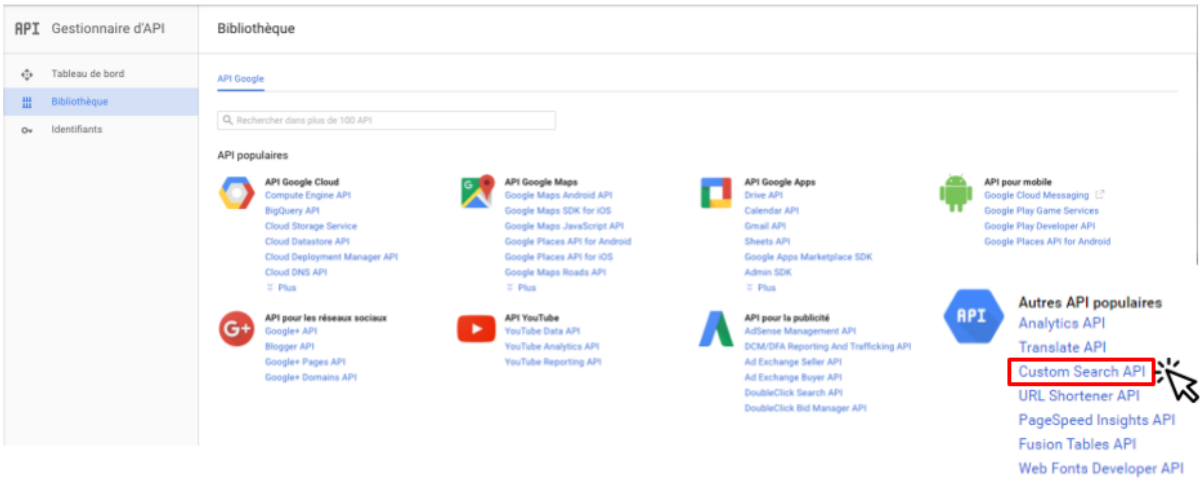


Figure 2 – Screenshot of all existing Google API

Then, the system will ask you to create a project in order to activate the APIs. Click on "Create a project".

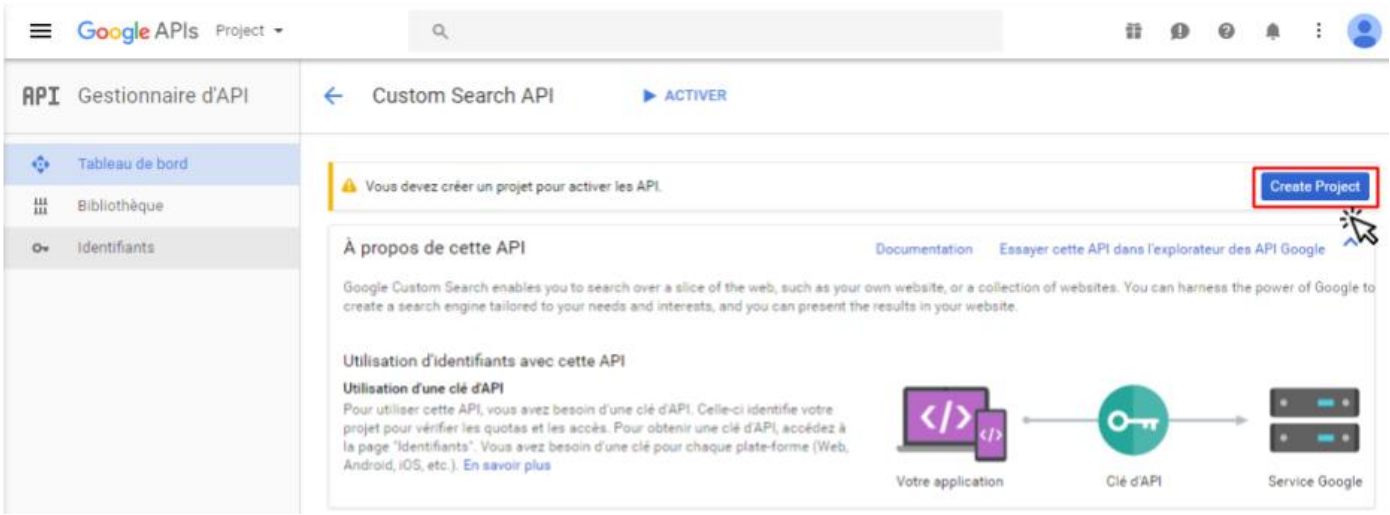


Figure 3 – Screenshot of Custom Search API page

Fill in the form, accept the conditions of use and click on “Create”. Now that the project has been created, you are able to activate the API.

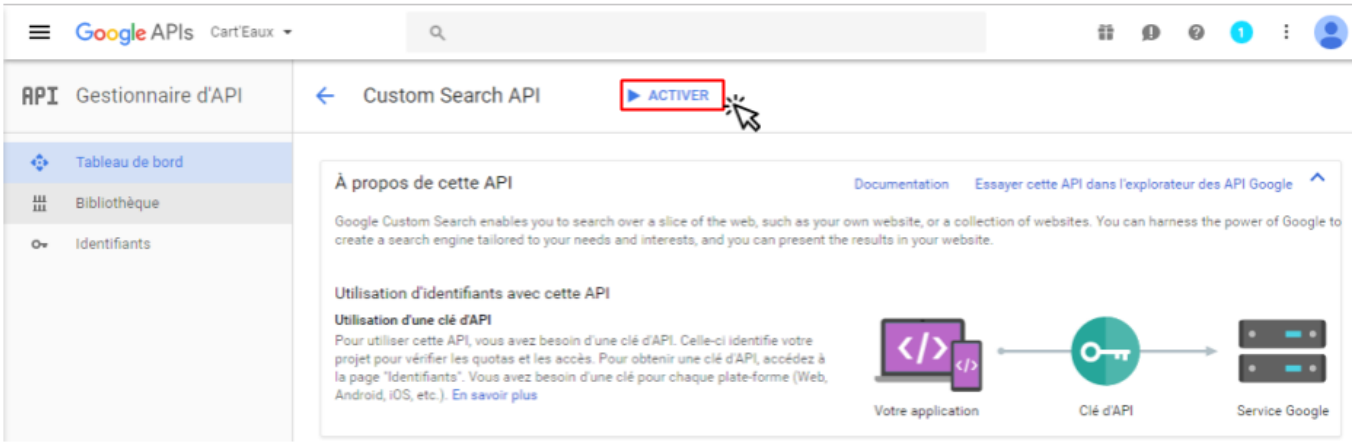


Figure 4 – Screenshot of Custom Search API page to activate the key

Once the API is activated, Google ask you to create IDs to be able to use the API. Follow the process to do so.

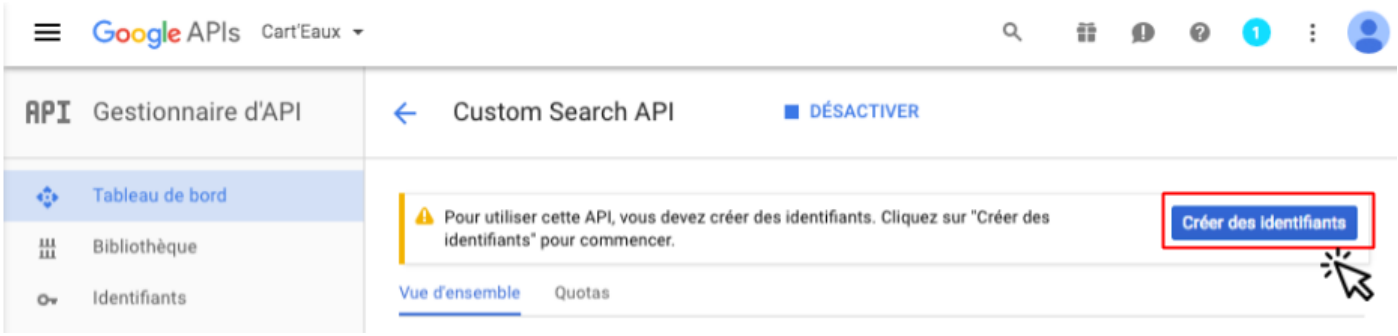


Figure 5 – Screenshot of Custom Search API page to create identifiers

Check that Custom Search API is selected and click on “De quels identifiants ai-je besoin ?”

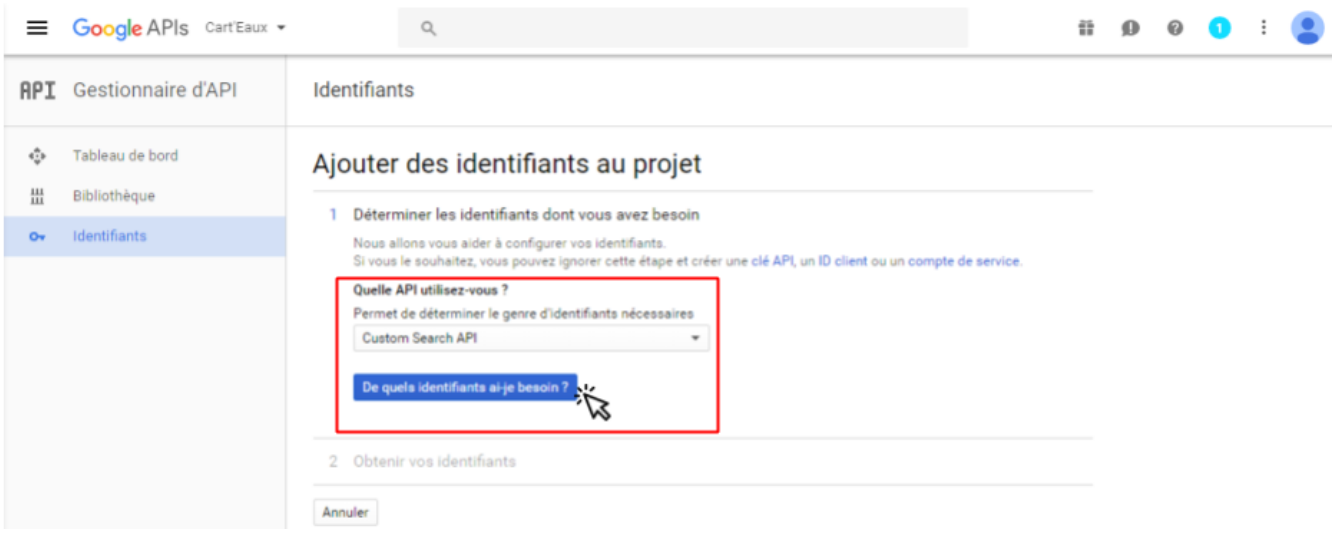


Figure 6 – Screenshot of identifiers page for Custom Search API

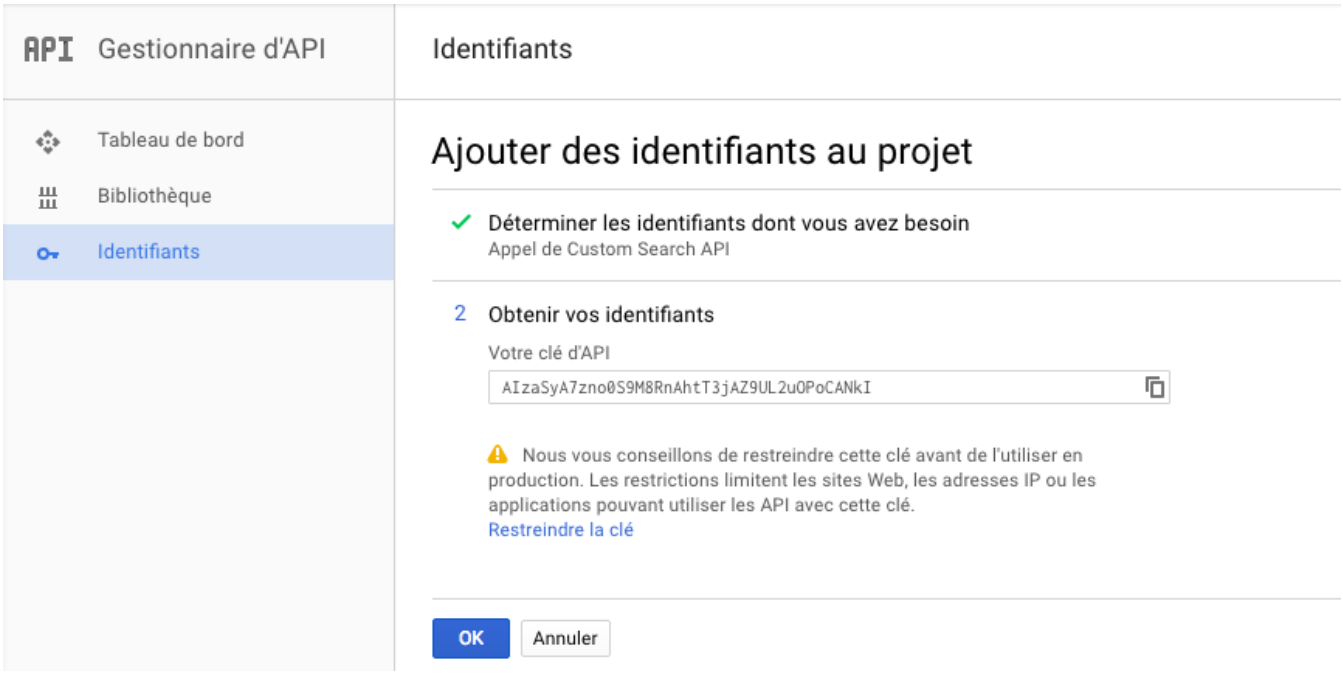


Figure 7 – Screenshot of identifiers page for API key

Now, your API key has been created. You can find it in the tab “Identifiants”.

Create a Custom Search Engine

Now, the Google Custom Search API has been activated and you have an API key. We are going to create a Custom Search Engine from which we are going to run our queries. Go to the following link: https://cse.google.com/ and add a new search engine.

Figure 8 – Screenshot of Custom Search Engine

Fill in the form like the following and click on “Créer”.

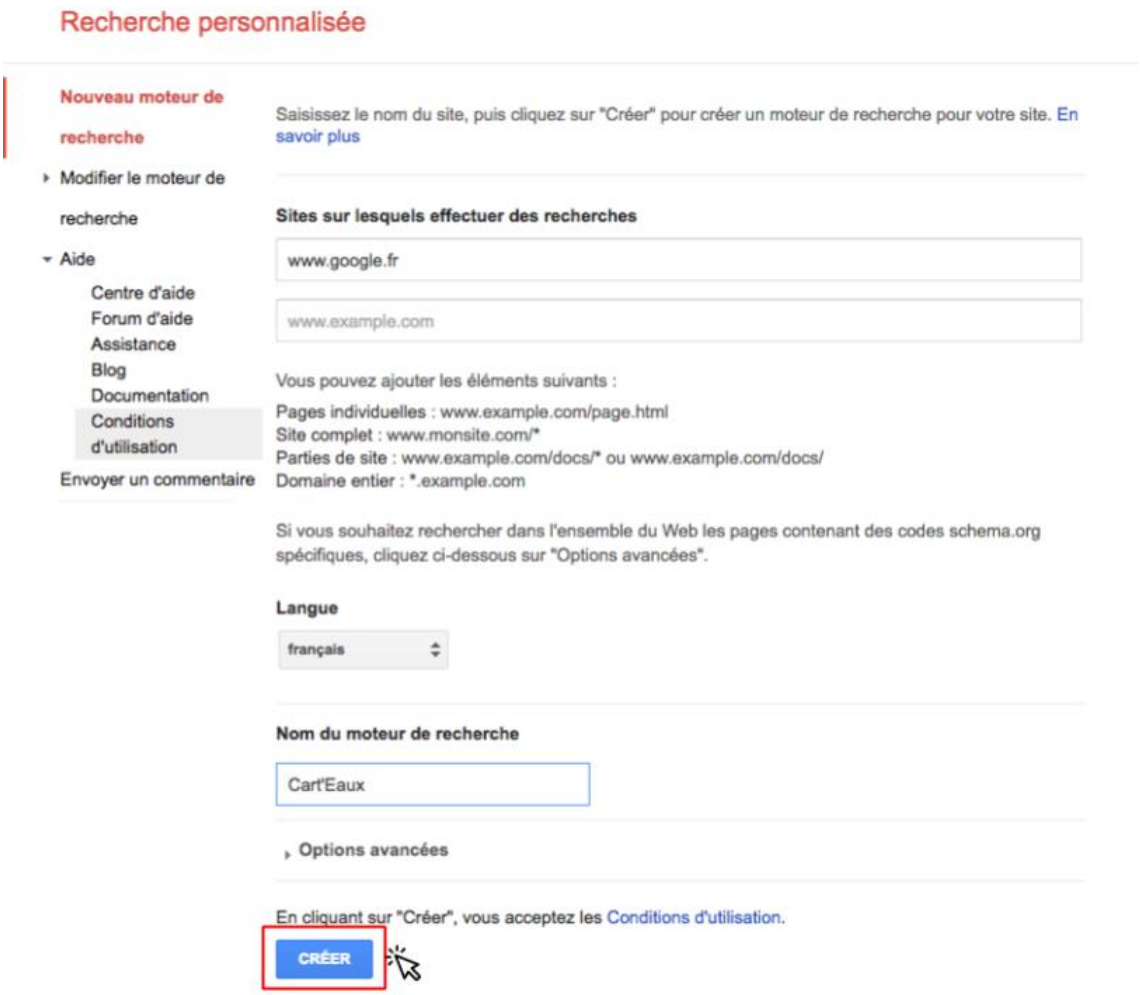


Figure 9 – Screenshot of the filled form to create your own Custom Search Engine

We are now going to configure it. For doing so, in the right menu bar, select “Modifier le moteur de recherche”, select the one that we have just created and click on “configuration”.

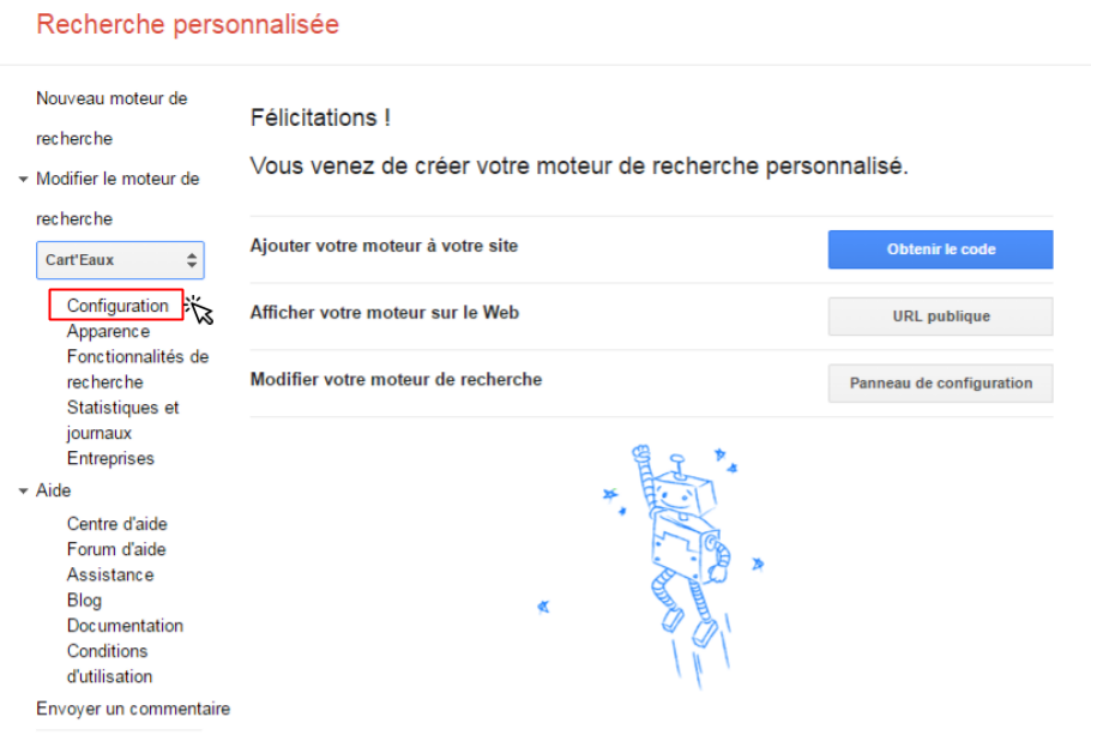


Figure 10 – Screenshot to configure your Custom Search Engine

In the part “Sites sur lesquels effectuer des recherches”, select “Chercher sur l’ensemble du web en privilégiant les sites inclus” and delete the website defined when we created the search engine (www.google.fr in our case).

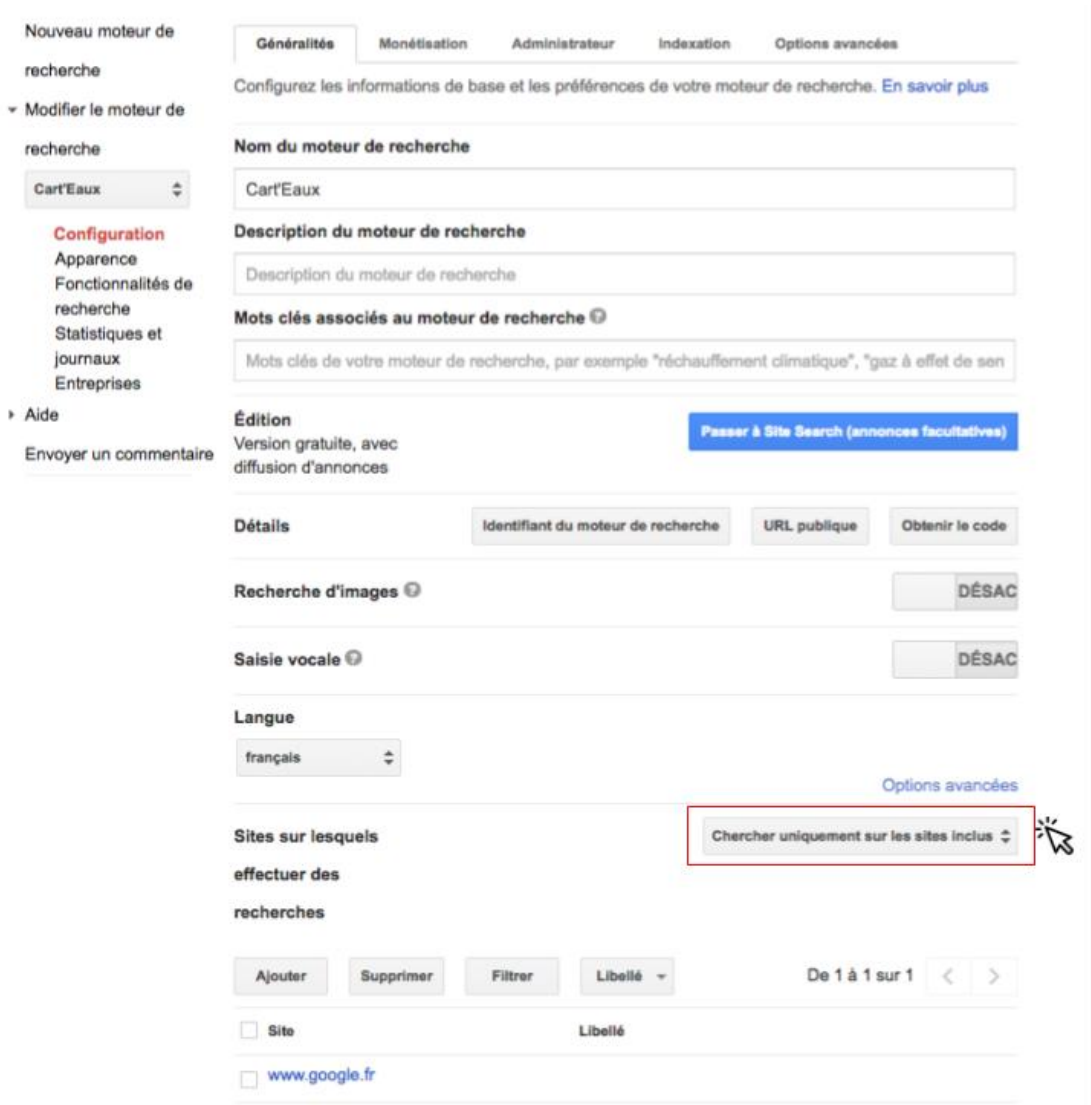


Figure 11 – Screenshot to modify and configure your Custom Search Engine

Figure 12 – Screenshot to delete websites included in your Custom Search Engine

That's it! Your Custom Search Engine is configured.

Find the parameters required to run the program

To use the JSON API some parameters are required: your API key and the ID of your search engine (cx). We are going to show you where to find this information and insert the parameters in our program.

First, open the source code of our program. You will have to paste the parameters here:

```
# User's keys required for the functioning of the program
key="YOUR_API_KEY"    # Enter here your Google API Key
cx="YOUR_SEARCH_ENGINE_ID"  # Enter here your search engine ID
```

Figure 13 – Place where included the API key and cx in the program

To find the API key, go to the Developer Console (https://console.developers.google.com/). On the right menu bar, go to "Identifiants" and you will see your API key. Copy and paste this key in the program.

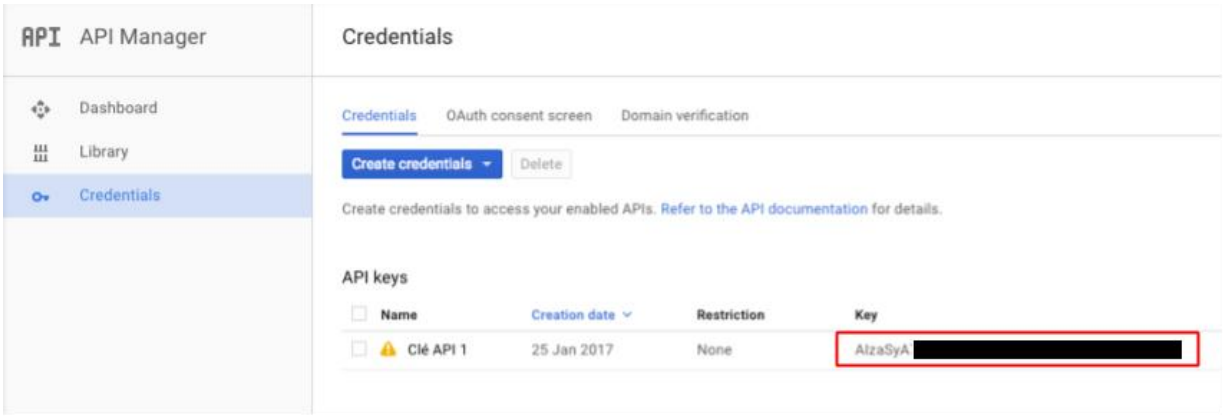


Figure 14 - Screenshot of the page where to find the API key

To find the "cx", go to your Custom Search Engine (https://cse.google.com/). Click on the engine you created before for the Cart'eaux project and click on "Identifiant du moteur de recherche". Copy and paste this ID in the program.

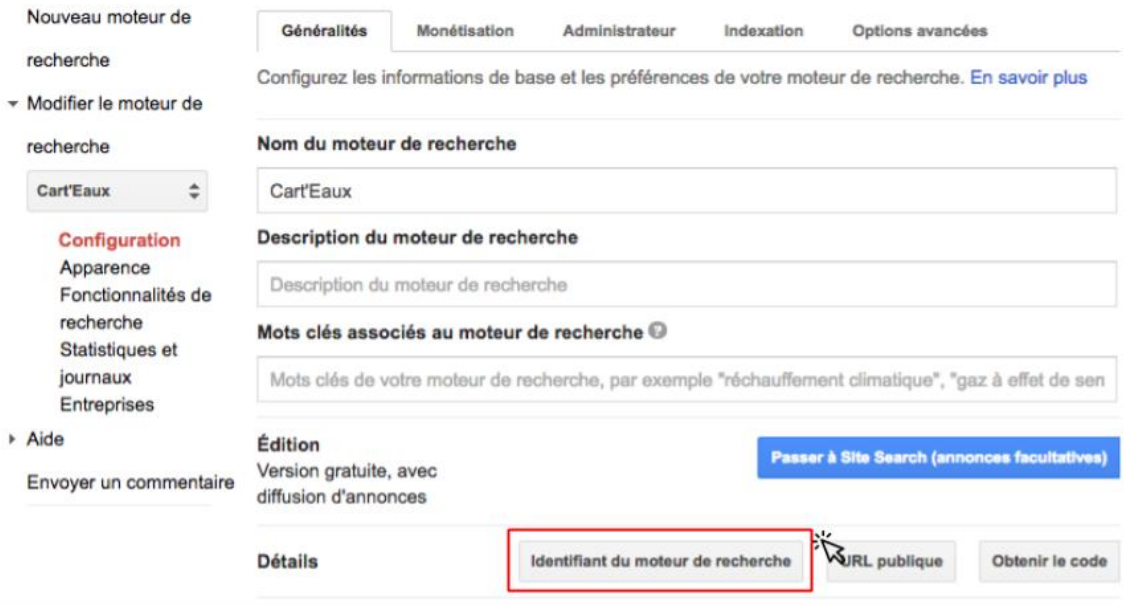


Figure 15 – Screenshot of the place where to find the cx information

Now, everything is set up to run the program.

c. Running the program

Now the configuration is done, you can run the program. To do so, open the `search_python.py` with IDLE[5] Python 2.7 (an integrated development environment for Python), and click on "Run" → "Run Module". Then the shell will open and you will have to enter the name of the city on which you want to perform the search.

[5] IDLE : Integrated DeveLopment Environment or Integrated Development and Learning Environment

III. EXPLANATION OF THE CODE

In order to explain the technical aspects of our solution and the function that have been used, we are going to examine the code of our program by chronologically following its execution.

a. Setting up

The first part of our program is a setting up: we gather needed information for what happens next.

The first parameters that we store are the Google API key and the search engine ID of the user. There will be used to perform queries through the API.

```
# User's keys required for the functioning of the program
key="YOUR_API_KEY"   # Enter here your Google API Key
cx="YOUR_SEARCH_ENGINE_ID"  # Enter here your search engine ID
```

Then we need to define on which city the user wants to run the queries. For doing that, we use the function `raw_input`. It allows us to display a message to the user, via the Python Shell, and to retrieve his answer. So here we ask him on which city he wants to perform the search and we keep its answer in a variable. Then, we replace characters like spaces and apostrophes so that it will fit to the URL encoding.

```
# The user enters the city name on which he wants to run the search
city=raw_input("Sur quelle ville voulez-vous effectuer la recherche ?")

# Replacing characters that can cause issue once inserted in the URL
city=city.replace(" ","%20")
city=city.replace("'","%27")
city=city.replace("'","%27")
```

After that, we are going to create all the directories in which the documents will be stored.

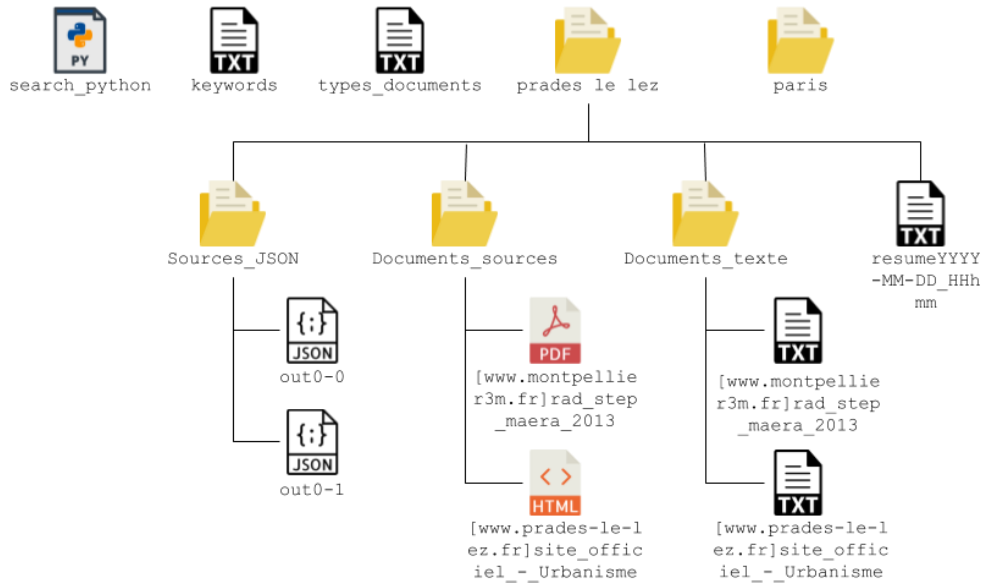


Figure 16 - Tree view of files

The first directory to be created is the one with the name of the city. Before creating it, we check if the directory already exists thanks to the function `os.path.exists(path)`. If it already exists then do nothing, else we create it with the function `os.makedirs(path)` and tell the user that it has been created, via the shell. For the other directories, we keep the same method, except that we change the path in order that the directories are created inside the first one. To do so, we create the path with the function `os.path.join(path, *paths)` that concatenate the parameters with directory separation.

```
path_city=city
if not os.path.exists(path_city):   # If the directory of the city does not exist, it creates it
    os.makedirs(path_city)
    print("Creation of directory for "+str(city))
    print("\n")

path=os.path.join(path_city,"Sources_JSON")     # Defines the path for the JSON directory
if not os.path.exists(path):    # If the JSON directory does not exist, it creates it
    os.makedirs(path)
    print("Creation of JSON directory")
    print("\n")

path2=os.path.join(path_city,"Documents_sources")   # Defines the path for the directory of source documents
if not os.path.exists(path2):   # If the directory does not exist, it creates it
    os.makedirs(path2)
    print("Creation of Documents_sources directory")
    print("\n")

path3=os.path.join(path_city,"Documents_texte")     # Defines the path for the directory of text documents
if not os.path.exists(path3):   # If the directory does not exist, it creates it
    os.makedirs(path3)
    print("Creation of Documents_texte directory")
    print("\n")
```

Finally, we are going to read the files where all the keywords required to build the queries are written. To do so in Python, we have to open our output file, do the actions on it and close the connection. This is done by the function `with open('workfile') as file:`. Once the connection is opened, we use the method `readlines()` that reads until EOF and returns a list containing the lines. In our case, we have one combination of keywords by line so we will just have to iterate on this list to get all the combinations. We do this for the 2 text documents that we have.

```
with open("keywords.txt") as keywords_file:     # Open the keywords file and read the lines
    words = keywords_file.readlines()
    print("Reading keywords.txt ...")
    print("\n")
with open("types_documents.txt") as docs_file:  # Open the type of documents file and read the lines
    docs = docs_file.readlines()
    print("Reading types_documents.txt ...")
    print("\n")
```

To finish, as the keywords from the lists will be integrated in a URL, we browse the 2 lists to replace some characters that will not be in the query.

```
for i in range(0, len(words)) : # Replacing characters that can cause issue once inserted in the URL
    words[i] = words[i].replace("\n","")
    words[i] = words[i].replace(" ", "%20")
    words[i] = words[i].replace("'", "%27")

for i in range(0, len(docs)) :  # Replacing characters that can cause issue once inserted in the URL
    docs[i] = docs[i].replace("\n","")
    docs[i] = docs[i].replace(" ", "%20")
    docs[i] = docs[i].replace("'", "%27")
```

b. Getting the JSON response from the API

Now that we have all the parameters to run the queries, we are going to build them and use the API to get the results in JSON documents.

In order to associate all the technical keywords with some type of documents, we use a nested loop. Each iteration corresponds to a different query, so each time we create a document in the JSON directory where the results will be written.

```
for i in range(0,len(words),1):     # Iterate on the keywords list
    for j in range(0, len(docs),1):     # Iterate on the type of documents
        f = open(os.path.join(path,'out' + str(i) + "-" + str(j) + '.json'),'w+')     # Open a JSON file
```

We build the query by combining the city given by the user, a line from the technical keywords document (that corresponds to a combination of 2 technical keywords) and a line from the document that contain types of documents keywords.

```
query = city + "%20" + words[i] + "%20" + docs[j]     # Modeling the query
```

Before running the query with the API, we check if the quota of 100 queries is not exceeded. If it is okay, we open the URL of the API, corresponding to the GET HTTP verb, formed with the parameters (API key, search engine ID and query). Here, the query has been run so we increase the query counter by an increment of one. We store the response in a variable thanks to the library `urllib2` and the method `urlopen(url)`. Then, we write the response in the JSON document created before. Finally, we create a text document that corresponds to the report (where all the results will be detailed). We use the system date to name the report document generated so that the user can see when he performed the search and that if he carries out another search later, it will generate a different report document. We open the connection to this document with the function `open(path, 'a')` (`'a'` opens the file for appending; any data written to the file is automatically added to the end).

```
if (cpt_query<100):     # Checking that the number of queries is not exceeded
    response = urllib2.urlopen("https://www.googleapis.com/customsearch/v1?key="+key+"&cx="+cx+"&q="+query)     # Open the URL from the API
    cpt_query = cpt_query+1     # Incrementation of the number of query
    # Printing the results in the JSON file
    html = response.read()
    f.write(html)
    f.close()
    print("JSON file created")
    print("\n")
    res=open(os.path.join(path_city,'resume'+date+'.txt'), 'a')     # Opening text file for the search report
```

c. Pause and restart of the program

In the event that the quota is exceeded, i.e. the query counter is superior or equal to 100 and we inform the user via the shell. Then we calculate at what time the quota will reboot by adding 24 hours to the system date with the function `timedelta(hours=24)`. While the system date is not superior to the reboot date, the execution is paused with the `time.sleep(1)` method. When it is okay, we notify the user, we reinitiate the query counter back to 0 and we process similarly than before to run the queries.

```
else:
    print(str(datetime.now()) + ": Number limit of queries reached. Please wait 24 hours before the program restarts ...")
    tomorrow=datetime.now() + timedelta(hours=24)
    while datetime.now()<tomorrow:
        time.sleep(1)
    print(str(datetime.now()) + ": Number of queries back to 0. Restart of the program ...")
    print("\n")
    cpt_query=0
    response = urllib2.urlopen("https://www.googleapis.com/customsearch/v1?key="+key+"&cx="+cx+"&q="+query)
```

d. Processing the results one by one

1. Reading the JSON document

For the moment, we have a JSON document listing the results of the query that we obtained thanks to the API and we have opened a connection to write the report document. So now we want to read the JSON document in order to access all of its data and write the information in the report. To do so, we open the connection with the JSON document and we load the data in a variable with the method `json.load(file)`. It deserializes the file containing the JSON document to a Python object using the conversion table opposite.

JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
true	True
false	False
null	None

Figure 17 – Conversion table JSON to Python

2. Accessing information about the query

Now all the data concerning the processed query from the JSON document can be accessed. We look how many results there are so that we can iterate on the list and check each result. We also write in the report the keywords of the query, the total number of results and the number of results displayed.

```
print("Processing new query:" + str(words[i]) + " + " + str(docs[j]))
print("\n")
with open(os.path.join(path,'out'+str(i)+"-"+str(j)+'.json')) as data_file:     # Reading the results from JSON file
    data1=json.load(data_file)
    rq = data1['queries']['request'][0]['searchTerms']      # Retrieving keywords of the query and writing it in the report
    res.write("################################################ \n")
    res.write("#############   Nouvelle requete   ############# \n")
    res.write("################################################ \n")
    res.write("\n")
    res.write("Mots clés de la requete : "+rq.encode('utf-8')+"\n")
    total_res = int(data1['queries']['request'][0]['totalResults'])     # Retrieving total number of results
    res.write("Nombre total de résultats : "+str(total_res)+"\n")
if (total_res>10):  # If the number of results is bigger than 10, we display the 10 first results else we display all the results
    nb_result = 10
else:
    nb_result = total_res
res.write("Nombre de resultats affichés : "+str(nb_result)+"\n")
```

3. Accessing details for each result

Now we are going to browse the list of results, one by one. We read the URL, the title and the snippet of the result in the JSON document and we keep this information in variables. We write the details of the result in the report document by being careful to ensure that there is no problem of encoding between the 2 documents. To ensure that, we use the method `encode('utf-8')`.

```
for k in range (0, nb_result):  # Iterating on the number of results displayed to get the details of each of them
    url = data1['items'][k]['link']
    docName = data1['items'][k]['title']
    docName = docName.replace("\"", "-")
    docName = docName.replace("/", "-")
    docName = docName.replace(" ", "_")
    snippet = data1['items'][k]['snippet']
    displayLink = data1['items'][k]['displayLink']
    print(url)
    print(docName)
    res.write("URL : "+url+"\n")
    res.write("Nom : "+docName.encode('utf-8')+"\n")
    res.write("Résumé : "+snippet.encode('utf-8')+"\n")
```

4. Identifying the file extension

From now on, we are going to prepare the storing of the documents that we found thanks to the query. We first need the format of the result so that when we will download it, we keep it under the correct file extension. To do so, we are going to look at the URL of the result and check if we can find a file extension. We use the function `os.path.splitext(path)` that splits the pathname into a pair root+extension and we look at the extension. If we find an extension like PDF, html, php or asp we write it in the report, else we can't define the result's format.

```
fileExtension = os.path.splitext(url)   # Splitting the URL to get the file format then writing it in the report
if ('fileFormat' in data1['items'][k]):
    file_format = ".pdf"
    res.write("Format : .pdf\n")
elif (".pdf" in fileExtension[1] or ".PDF" in fileExtension[1] or ".php" in fileExtension[1]
        or ".htm" in fileExtension[1] or ".asp" in fileExtension[1]):
    file_format = fileExtension[1]
    res.write("Format : "+fileExtension[1].encode('utf-8')+"\n")
else:
    file_format = "non renseigné"
    res.write("Format : non renseigné\n")
```

If the document is a PDF, then we have an additional section in the JSON document in which we can find, most of the time, a creation date and a modification date. So, in order to add this information in the report document, we check if the dates are present and we do what is necessary to extract them and write them in the report.

```
if (".pdf" in file_format or".PDF" in file_format):  # If it is a PDF we check if there is creation and modification date in the JSON doc then we write it in the report
    if ('pagemap' in data1['items'][k]):
        if ('creationdate' in data1['items'][k]['pagemap']['metatags'][0]):
            creation_date=data1['items'][k]['pagemap']['metatags'][0]['creationdate']
            if (creation_date.startswith("D:")):
                creation_date=creation_date.replace('+', ':')
                creation_date=creation_date.replace('Z', ':')
                creation_date=creation_date.split(":")
                creation_date=creation_date[1]
                date_split=[creation_date[l:l+2] for l in range(0, len(creation_date), 2)]
                try:
                    creation_date=date_split[3]+"-"+date_split[2]+"-"+date_split[0]+date_split[1]
                except:
                    creation_date="Inconnu"
                try:
                    creation_hour=date_split[4]+":"+date_split[5]+":"+date_split[6]
                except:
                    creation_hour=" "
                res.write("Date et heure de création : "+creation_date.encode("utf-8")+" "+creation_hour.encode("utf-8")+"\n")

        if ('moddate' in data1['items'][k]['pagemap']['metatags'][0]):
            modif_date=data1['items'][k]['pagemap']['metatags'][0]['moddate']
            if (modif_date.startswith("D:")):
                modif_date=modif_date.replace('+', ':')
                modif_date=modif_date.replace('Z', ':')
                modif_date=modif_date.split(":")
                modif_date=modif_date[1]
                date_split=[modif_date[l:l+2] for l in range(0, len(modif_date), 2)]
                try:
                    modif_date=date_split[3]+"-"+date_split[2]+"-"+date_split[0]+date_split[1]
                except:
                    modif_date="Inconnu"
                try:
                    modif_hour=date_split[4]+":"+date_split[5]+":"+date_split[6]
                except:
                    modif_hour=" "
                res.write("Date et heure de modification : "+modif_date.encode("utf-8")+" "+modif_hour.encode("utf-8")+"\n")
```

5. Retrieving source documents

Now that we know the format of the result, we are going to define the procedure to follow to retrieve the document, depending on the format, and convert it into text.

In the event that the result is a PDF document, we are first going to retrieve the document under the PDF format. We use the method `urllib.FancyURLopener()` with the context parameter to configure the SSL settings that are used if we make a HTTPS connection. Then we save locally the PDF document in the source directory thanks to the method `retrieve(url, path)`, by naming it with the display link and the title that we found in the JSON document. We use these 2 to ensure that we will not retrieve two different documents with the same name. As we check if the document already exists with the method `os.path.isfile(path)`, we do not download the document several times if it is found with other queries.

```
if (".pdf" in file_format or".PDF" in file_format):     # If it is a PDF we retrieve the document and convert it into text
    ctx=ssl.create_default_context()
    ctx.check_hostname=False
    ctx.verify_mode=ssl.CERT_NONE
    testfile=urllib.FancyURLopener(context=ctx)
    if not (os.path.isfile(os.path.join(path2,"["+displayLink+"]"+docName+".pdf"))):
        testfile.retrieve(url,os.path.join(path2,"["+displayLink+"]"+docName+".pdf"))
        path_to_pdf=os.path.join(path2,"["+displayLink+"]"+docName+".pdf")
        print("PDF document saved")
```

6. Converting source documents into text

After that, we use the function `convert_pdf_to_txt(path)` in a try block in case a error appears. If there is no error we create a text document and we write the converted text.

```
print("Converting into text...")
try:
    text=convert_pdf_to_txt(path_to_pdf)
except:
    text=None
if text is not None:
    f=open(os.path.join(path3,"["+displayLink+"]"+docName+".txt"),"w+")
    f.write(text)
    f.close
    print("PDF document converted into text")
    print("\n")
```

Hereafter is the function used:

```
def convert_pdf_to_txt(path):
    """ Convert a PDF document into text.
        Args:
            path: the path of the PDF document

        Returns:
            The text converted from the PDF document.

    """
    rsrcmgr = PDFResourceManager()
    retstr = StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, codec=codec, laparams=laparams)
    fp = file(path, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    password = ""
    maxpages = 0
    caching = True
    pagenos=set()
    parser = PDFParser(fp)
# Create a PDF document object that stores the document structure.
# Supply the password for initialization.
    document = PDFDocument(parser, password)
    if document.is_extractable:
        for page in PDFPage.get_pages(fp, pagenos, maxpages=maxpages, password=password,caching=caching, check_extractable=True):
            interpreter.process_page(page)

        text = retstr.getvalue()

        fp.close()
        device.close()
        retstr.close()
        return text
```

If it is another format (html, php, asp), we are first going to save the source code of the result. We use the function `urllib.urlopen(url)` to open the page, we read the source code with the `read()` method and we ensure that it is written with UTF-8 encoding. We save the source code in a file located in the source directory and we convert it into text with

`HTML2Text()`. We use the same method as before to name the files, only the extension will change.

```
if (".html" in file_format or ".htm" in file_format or ".asp" in file_format or ".aspx" in file_format):
    h=html2text.HTML2Text()
    html = urllib.urlopen(url).read()
    html=html.decode("UTF-8","ignore")
    html=html.encode("UTF-8","ignore")
    if not (os.path.isfile(os.path.join(path2,"["+displayLink+"]"+docName+".html"))):
        page=open(os.path.join(path2,"["+displayLink+"]"+docName+".html"),"w+")
        page.write(html)
        page.close()
        print("Document saved")
        result = h.handle(html.decode('utf-8'))
        f=open(os.path.join(path3,"["+displayLink+"]"+docName+".txt"),"w+")
        f.write(result.encode('utf-8'))
        f.close()
        print("Document converted into text")
        print("\n")
```

We do this for all the results of the query, then we run another query, save the results under text format and so on and so forth until we use all the combinations to build our queries.

e. Sending an email

When all the queries have been executed, we send an email to the user in order to notice him of the end of the program's execution. We attached the report document so that the user can find all the results.

```
# When the program is over, send email with the report attached
fromaddr = "your.address@gmail.com" # YOUR ADDRESS
toaddr = "your.address@gmail.com" # EMAIL ADDRESS YOU SEND TO

msg = MIMEMultipart()

msg['From'] = fromaddr
msg['To'] = toaddr
msg['Subject'] = "[Projet Cart'Eaux] - Fin du programme !" # SUBJECT OF THE EMAIL

body = "Execution du programme terminée, vous pouvez trouvez un compte rendu en pièce jointe." # TEXT YOU WANT TO SEND

msg.attach(MIMEText(body, 'plain'))

filename = 'resume'+date+'.txt' # NAME OF THE ATTACHED FILE WITH ITS EXTENSION
attachment = open(os.path.join(path_city,'resume'+date+'.txt'), "rb") # open("PATH OF THE FILE", "rb")

part = MIMEBase('application', 'octet-stream')
part.set_payload((attachment).read())
encoders.encode_base64(part)
part.add_header('Content-Disposition', "attachment; filename= %s" % filename)

msg.attach(part)

server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login(fromaddr, "PASSWORD") # server.login(fromaddr, "PASSWORD OF YOUR EMAIL ACCOUNT")
text = msg.as_string()
server.sendmail(fromaddr, toaddr, text)
server.quit()
```

CONCLUSION

The industrial project that we realized allowed us to acquire and apply a lot of knowledge in technical, methodological as well as interpersonal relationship.

The technological choice was really important for the good proceedings of the project. Choosing Python made easier our work because we could rely on a lot of tools that saved time. Moreover we saw the importance of using a language that has a large community of users because it helped us to solve errors and we have been able to use a lot of libraries developed by passionate people. We can also say that our Python program is easy to deploy thanks to the `requirements.txt` file and the `pip install` command.

We tried to run a lot of tests to face every eventuality but it was impossible for us to test it on a large period as it is supposed to perform during more than 3 months. Consequently, our program can be improved. We applied ourselves to write comments on the code and detail our approach in order that our solution can be maintained.

Finally, we have high hopes that our solution will be used by the Cart'Eaux project and permit them to save time and focus on other tasks just as important.

WEBOGRAPHY

[1] Wikipedia contributors, Java (programming language), In *Wikipedia, The Free Encyclopedia* [Online], https://en.wikipedia.org/wiki/Java_(programming_language) (read on)

[2] Wikipedia contributors, Java Platform, Enterprise edition, In *Wikipedia, The Free Encyclopedia* [Online], https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition (read on)

[3] Wikipedia contributors, Python (programming language), In *Wikipédia, The Free Encyclopedia* [Online], https://en.wikipedia.org/wiki/Python_(programming_language) (read on 3rd December 2016)

Joel SPOLSKY & Jeff ATWOOD, Stack Overflow, August 2008, available at http://stackoverflow.com/

APPENDIX

Appendix 1: JSON structure of the response body

```
{
  "kind": "customsearch#search",
  "url": {
    "type": "application/json",
    "template":
"https://www.googleapis.com/customsearch/v1?q={searchTerms}&num={count?
}&start={startIndex?}&safe={safe?}&cx={cx?}&cref={cref?}&sort={sort?}&f
ilter={filter?}&gl={gl?}&cr={cr?}&googlehost={googleHost?}&c2coff={disa
bleCnTwTranslation?}&hq={hq?}&hl={hl?}&siteSearch={siteSearch?}&siteSea
rchFilter={siteSearchFilter?}&exactTerms={exactTerms?}&excludeTerms={ex
cludeTerms?}&linkSite={linkSite?}&orTerms={orTerms?}&relatedSite={relat
edSite?}&dateRestrict={dateRestrict?}&lowRange={lowRange?}&highRange={h
ighRange?}&searchType={searchType}&fileType={fileType?}&rights={rights?
}&imgSize={imgSize?}&imgType={imgType?}&imgColorType={imgColorType?}&im
gDominantColor={imgDominantColor?}&alt=json"
  },
  "queries": {
    (key): [
      {
        "title": string,
        "totalResults": long,
        "searchTerms": string,
        "count": integer,
        "startIndex": integer,
        "startPage": integer,
        "language": string,
        "inputEncoding": string,
        "outputEncoding": string,
        "safe": string,
        "cx": string,
        "cref": string,
        "sort": string,
        "filter": string,
        "gl": string,
        "cr": string,
        "googleHost": string,
        "disableCnTwTranslation": string,
        "hq": string,
        "hl": string,
        "siteSearch": string,
        "siteSearchFilter": string,
```

```
        "exactTerms": string,
        "excludeTerms": string,
        "linkSite": string,
        "orTerms": string,
        "relatedSite": string,
        "dateRestrict": string,
        "lowRange": string,
        "highRange": string,
        "fileType": string,
        "rights": string,
        "searchType": string,
        "imgSize": string,
        "imgType": string,
        "imgColorType": string,
        "imgDominantColor": string
      }
    ]
  },
  "promotions": [
    {
      "title": string,
      "htmlTitle": string,
      "link": string,
      "displayLink": string,
      "bodyLines": [
        {
          "title": string,
          "htmlTitle": string,
          "url": string,
          "link": string
        }
      ],
      "image": {
        "source": string,
        "width": integer,
        "height": integer
      }
    }
  ],
  "context": {
    "title": string,
    "facets": [
      [
        {
          "label": string,
          "anchor": string,
```

```
          "label_with_op": string
        }
      ]
    ]
  },
  "searchInformation": {
    "searchTime": double,
    "formattedSearchTime": string,
    "totalResults": long,
    "formattedTotalResults": string
  },
  "spelling": {
    "correctedQuery": string,
    "htmlCorrectedQuery": string
  },
  "items": [
    {
      "kind": "customsearch#result",
      "title": string,
      "htmlTitle": string,
      "link": string,
      "displayLink": string,
      "snippet": string,
      "htmlSnippet": string,
      "cacheId": string,
      "mime": string,
      "fileFormat": string,
      "formattedUrl": string,
      "htmlFormattedUrl": string,
      "pagemap": {
        (key): [
          {
            (key): (value)
          }
        ]
      },
      "labels": [
        {
          "name": string,
          "displayName": string,
          "label_with_op": string
        }
      ],
      "image": {
        "contextLink": string,
        "height": integer,
```

```
        "width": integer,
        "byteSize": integer,
        "thumbnailLink": string,
        "thumbnailHeight": integer,
        "thumbnailWidth": integer
      }
    }
  ]
}
```

Résumé :

Nous avons travaillé pour le projet Cart'Eaux. Il a pour but de développer une méthode de récolte de donnée multi-sources et de fusion des connaissances en une information permettant la cartographie et la modélisation hydraulique d'un réseau d'assainissement urbain.

Nous avons développé un programme en Python afin d'automatiser la recherche et la collecte de documents en rapport avec les réseaux d'assainissement et les eaux usées pour une ville spécifique, sur le web, et de les transformer en documents texte.

Ce rapport traite de l'installation du programme, de la mise en place de l'API et d'une explication détaillée de notre solution.

Mots clés : Projet Cart'Eaux, Python, API, recherche automatisée

Abstract:

We worked for Cart'Eaux project. This project aims at developing multi-sources data collection method and merge knowledge in information that allows cartography and hydraulic design of urban purification network.

We developed a Python program in order to automate the search and the collection of documents related to the stormwater and wastewater networks of a specific city, on the web, and to transform them into text files.

This report deals with the program installation, the API setting up and a detailed explanation or our solution.

Keywords: Cart'Eaux project, Python, API, automated search