Project

Start : October, 2th, 2020
End : October, 9th, 2020

Subject

Goals

Develop an API RESTful JSON in Python to perform text mining.
Input : Text.
Output :
- List of people with information such as type, label, description, image, etc.
- Group by eventual common characteristics such as nationality, age, job, etc.

Deliveries

- English log board: explain the architecture, steps of thinking, improvement for production.
- Docker image: with READ-ME containing instructions to test the project.
- GitHub repository: source code with clear services (single responsibility) and unit tests.

Bonus :

*) L'API JSON could be encapsulated into a docker container.

**) 12 factors development (https://12factor.net/)

***) Graphic interface in JS for the user to test the API.

***) Propose cool features for the app.

Tools proposed

Entity extraction: API TextRazor.

Collecting entity information: Wikidata.

Sprint 1

While this first sprint aims to build an API quickly, I kept in mind how it could evolve, and defined the architecture accordingly.

Functionalities

Let’s clarify the functionalities that our first version should implement.

1. UC1 : Receive text analysis requests with options.
   The request should contain the text to be analyzed, and the options to configure how the analysis should be performed and define the results user is interested in.
2. UC2 :

Architecture

Controllers

This module represents the first interface with our API. It should be able to receive the requests from the user, validate the requests, handle errors and return a response.

I will start in this first sprint to implement only the « JobsController » but we imagine others such as « AuthController ».

JobsController

This sub-module is in charge managing different types of jobs the user wants to execute, such as text analysis, image analysis, voice analysis, etc.
For now I will consider only text analysis jobs, which lead to the following tasks:

- Receive text analysis requests and validate them. Request should be a POST method sending a JSON object. The JSON object should contain:
  - Required text, not exceeding a certain size (check size limit of TextRazorAPI).
  - Options, the text analysis workflow is configurable (for now at least the choice of entities extractors, researchers and aggregators) .
- Create and run a text analysis job.
  If we want to be able to process many requests simultaneously (many requests from one or many users) we should perform parallel computing with threads. Hence each request should be a new thread, but the WSGI server used by default in Bottle is non-threading HTTP server. The easiest way to overcome this is to install a multi-threaded server library like paste or cherrypy and tell Bottle to use that instead of the single-threaded server. But for now we will use the default WSGI server.

- Return a response to the user.

JobModels

As the user should be able to execute different types of jobs, this module represents this job execution.

A job is executed according to a workflow. Each step of the workflow represents a state of the job. Hence at each step/state the job will be enriched following the process of the previous step. At the end, the job returns the result.

Considering a request that needs to execute many jobs, we should perform parallel computing with threads. We implement this behavior with *concurrent.futures.ThreadPoolExecutor object.*

TextJob

The text job sub-module is in charge of all the steps of the text workflow, following the description of the goals of this API. We can define the following tasks :

- Preprocessing the text if needed.
- Extract entities from the text.
- Enrich entites with information, collected from different databases/APIs.
- Aggregate entities according to a strategy (same job, same age, etc)
- Return the result as a JSON object.
- Keep track of the information relative to the execution of the workflow. Such as the status (of the job), the starting time, etc.

TextWorkflow

This module defines all elements needed for our text analysis, such as data models, NLP algorithms, tools to target external APIs, etc.

Regarding one task (entities extraction for example), we may have many elements able to perform the task (TextRazor API or internal NLP methods). Therefor we implement the strategy design pattern, to be able to select one or another method.

Entities

According to the main goal of our API, we will manipulate Entities. Therefor we need to define a class representing this entities. From which we can inheritate an organisation class, a person class, etc. But for now we will only manage one type of entities.

We need to formalize information of entities through attributes and methods.

EntitiesExtractors

This sub-module aims to extract entities from a given text. The other tasks differs depending on the chosen strategy. For now we will only implement the tools allowing to use the TextRazor API.

EntitiesResearchers

This sub-module aims to collect information concerning the entities extracted from the given text. The other tasks differs depending on the chosen strategy. For now we will only implement the tools allowing to use the Wikidata API.

EntitiesAggregators

This sub-module aims to group entities and find common characteristics between the entities extracted from the given text and enriched with the sub-module EntitiesResearchers.

Configs

This module contains configuration files or our API.

ServerConfig

In this sub-module we store the server configuration. For instance the default value for the host and port. But moreover we setup the routes we can target within our API.

TextWorkflowConfig

As seen previously a workflow can be configured and here we store the needed configurations at the start of the API, such as default values, keys to connect to APIs, etc.

LoggersConfig

A YAML file describing the main logger of the app, the format and its handlers.

DataManagers

This module manages how the data manipulated withing the API is stored. We can have multiple types of data base depending on what we are storing or the activities, like MongoDB, ElasticSearch, etc.

Tools & Libraries

- **Bottle** : Micro web-framework for Python
- **Requests** :
- **Jsonschema** :

Questions

Input

- Comment gérer la taille du texte ?
  Définir une taille maximum ? Ou laisser libre choix ?
- Comment gérer la langue du texte ?
  L'information doit être transmis par l'utilisateur ? C'est géré automatiquement par le module qui va traiter le texte ?
- Quelles sont les bonnes pratiques d'un point de vue sécurité ?
- Comment procéder face à l'encodage du texte ? Sous quelle format l'API va recevoir le texte ? JSON ou Brut ?

Data

- Comment gérer les données manipulées au niveau de l'API ?

Controllers

- JobsController should be Singleton or static class?
  It should be Singleton ! It manages the requests of the user, and schedules the execution of the workflows. There should be only one instance with such power.