2021/
2022

Software Lab Project

WEB SERVICE APPLICATION AND CLIENT LIBRARY

BULLONI DYUMAN, NATHAN MARGNI

DATA SCIENCE AND AI, SECOND YEAR, SUPSI

COURSE RESPONSIBLE: GUIDI ROBERTO

Index

1 Abstract

During the bachelor course of Data Science in SUPSI, we have a module focussed on medium-level programming.
This project is its semester's work and evaluation, to put in practice the learned knowledges.
The idea was to create a first project to scratch, by dataset search to a simple client application that tested main function such as API connection in a RESTful service, with data properly stored in some manner.
The main topic used as context is the tourism and the implementation between hotels and point of interests in Europe.
This documentation focus on the requirements and the implementation of the tourism_api python library.
This module implements classes to communicate directly with an apposite API, built for this project.
The results are basic but solid. It misses some potential connection and interaction between our main data types, hotels, and point of interests, but achieve a functional API stored in a MongoDB collection.
Thanks to the implementation steps for this application, it will be easy to extend features and public it online.

2 Introduction

This documentation has the purpose to guide in the main decisions taken, both in the choose of a topic and the approach at the problem.

In the chapter 1, is it defined the conditions of why this project is born, as the definition of the context.
In the chapter 2 it is analysed the problem that this project had to solve with the proper details of the requirements.
Chapter 3 briefly explain the current state of the art and why it is not so relevant for this project itself.
Chapter 4 is dedicated to the implementation, how the problem was approached and what types of decisions it has been made to use proper methodologies and technologies.
Chapter 5 explains the results obtained and not obtained, in an objective way.
Chapter 6 sums up the results to confront them with the initial purpose and objectives of the project, explaining the limits and why they are not fulfilled and guidelines of what it will be required as future steps, as some suggestions to improve and expand the project.

3 Motivation and context

The work is originated from a semester's project from our bachelor's course, with the idea of using all the learned knowledges from the module in a practical application and evaluate the understanding.
The technologies are object-oriented classes, inheritance, encapsulation, web services and production of libraries to share. This documentation will be rather technical in explaining the different processes and uses of such methods.
The topic was not imposed and therefore we chose the tourism.
As we learned during the course, APIs are already an every-day technology, fundamental for every person that intends to work in the IT field or associates. Having a practical approach to the problem, this project will be a valid basis to use as reference for future applications. The connection to the database itself is based on the same concept: by scope of the project, it was sufficient to use simple plain files to store and interact with the Flask application. By using a new and just known by other courses technology, our purpose is to imitate most as possible an actual useful and usable application.

4 Problem

The problem is in this case self-created: the scope of the project was to find a topic where the knowledges could have been applied properly.
Our group chose to work on the tourism field, by managing hotels and point of interests from Europe.

4.1 Objectives and requisites

The project has 10 weeks duration, 2 of them in the winter holidays. Each week the lab session is available for working at the project.

The work is done in couple. The domain of the application is chosen internally.

The backend service must be able to:

- Read data from a data source of choice
- Provide basic read/write operations on the data source
- Provide additional useful operations on your data set, as statistics or custom queries.
- It has to be developed using the flask framework.

The client library must be thought for a developer user, to easily interact with the webservice and generate some diagrams with the data obtained.

- The project has to be managed using the provided GIT repository.
- Be object oriented.
- Manage dependencies using tool such as pip, pipenv or conda.
- Be fully unit tested.
- Be built/distributed as a wheel package.
- The number of operations has to be significative, around 15 of them with at least one example per each request method.

5 State of the art

In this project the state of the art is not considered since the scope is more practical and regarding the implementation than a proper final application that put something new on the field. They already exist many APIs or websites that manages tourism, but it was strictly forbidden to use them directly.

6 Approach to the problem

6.1 Search of Data

The first thing that it was required it was actually find out the data to use. Having chosen the domain ourselves, we had a decent idea of where to search. However, the initial ideas were discarded by lack of interesting data or impossibility to use it properly, such as not-free APIs or databases that would have required some type of complex work. As Europe was too broad as field to find every data we wanted (for points of interests), we reduced the data to only Switzerland, limiting therefore also the hotel data.

6.2 Pre-Processing

Once we found the data, a detailed hotel world dataset and another (less detailed) for the points of interest in Switzerland, we pre-processed the two datasets.

For the hotels we did the usual cleaning, handling outliers and missing values as we should, checked that all the coordinates were valid and removed all hotels outside Switzerland.

Then also for the points of interest we applied the same procedures and removed some columns, plus we found out that some points were of Hotels and therefore we decided to try merging these with those on the hotel dataset to have more information about them, to do that we firstly created a function (also present as method in the Position class) that calculate the distance between two points by transforming the coordinates in radians and then computing the distance using a formula that uses the radius of the hearth, suddenly we created an algorithm that find near hotels from one dataset to the other with a nested for loop, but for some reason the coordinates resulted not very precise, for example applying this algorithm on the first 1000 hotels and a margin of 30 meters to find near hotels the algorithm found 490 matches and 60 hotels that got more than occurrence (found more than one hotel in the range of 30 meters), trying also with others margins the results were not reasonable (too few matches of too many "one to many" matches) so we abandoned this idea and remained with the original hotel dataset, because it have more information regarding the hotels.

We had also problem on the point of interest features because it was missing the name of the city or town of that position (useful for filtering) that is present instead on the hotels data, to solve that we tried using the Geopy API for reverse geocode (get address from position), but after some test and calculations we discovered that geocoding the whole dataset it would take about 28 hours, that is a lot of time and therefore we thought that wasn't worth it so we discarded also this idea, in any case it still implementable as we structured the point of interest class to support also this additional attribute, if present.

6.3 Class structure

For our project, based on the data previously obtained, we focus in two main types of objects: hotels and point of interests.

As both are defined by a clear geographic position with minor parameters that define things as cost, fanciness (hotel stars) or categories, in the case of point of interests, we decided to use as abstract and common class what it defines a place, such as country, city, address, and coordinates.

The position is relatively useful also to determine the real distance between two places as other possible considerations, where instead things as nation and city are more etiquettes. Therefore, we decided to properly split these two main categories in two distinct classes (Position and Address) and make the composite abstract class of these two for both Hotel and PointOfInterest.

6.4 Flask

The choice of this technology was not given since it was a main requirement.
We had the time to dig more on this powerful tool for managing an API.

Having a strong class-oriented application, we opted with the implementation of single blueprints. The blueprints are managed by a central Flask app, which registers the two blueprint objects.

The application is RESTful and thanks to the libraries, easy to put in development and create the proper web service online (not scope of this project).

For both hotels and points of interests, the methods implemented are GET (single and multiple value), POST (single value), PUT (single value) and DELETE (single value).

6.5 Database

We initially thought of implementing a simple relational database, but after some consideration we noticed that that would have been a waste of the technology, since the project does not require neither particular separation (dividing addresses and positions from the hotels/point of interests, both unique, would be meaningless) or strong transaction. As a simple storage built for being used by an API, and giving the nested structure of the main classes, we opted for MongoDB, implemented by the library pymongo.

Hotel Collection example:

	{} hotelId	{} hotelName	{} address	{} position	{} price	{} stars 1
1	199097	Swiss Diamond Hotel Olivella	{"countryName": "Switzerland", "countryCode": "CH", "cityName": "Vico Morcote", "s	{"longitude": 8.92903, "latitude": 45.9333}	204	5
2	207127	A Room With A View Bed and Breakfast Montreux	{"countryName": "Switzerland", "countryCode": "CH", "cityName": "Montreux", "stree	{"longitude": 6.88983, "latitude": 46.4454}	NaN	5
3	207292	Four Seasons Hotel des Bergues Geneva	{"countryName": "Switzerland", "countryCode": "CH", "cityName": "Geneva", "street"	{"longitude": 6.14692, "latitude": 46.207}	612	5

Point of Interest Collection example:

	poiId	poiName	address	position	category	subcategory	openingHours	otherTags
1	0	Clos de l'Ombren	{"countryName": "Switzerland", "countryCode": null, "cityName": "Yv	{"longitude": 6.954822, "latitude": 46.3294888}	TRANSPORT	BUSSTOP	nan	{"bus":"yes","uic
2	1	Güterbahnhof	{"countryName": "Switzerland", "countryCode": null, "cityName": "Zü	{"longitude": 8.5196824, "latitude": 47.3804363}	TRANSPORT	BUSSTOP	nan	{"bus":"yes","ref
3	2	Nations	{"countryName": "Switzerland", "countryCode": null, "cityName": "Ge	{"longitude": 6.1396377, "latitude": 46.2227546}	TRANSPORT	BUSSTOP	nan	{"bus":"yes","uic
4	3	Morgins	{"countryName": "Switzerland", "countryCode": null, "cityName": "Mo	{"longitude": 6.8581016, "latitude": 46.2376589}	SETTLEMENTS	VILLAGE	nan	{"openGeoDB:loc_i

Both collections include a field _id, defined internally in the database and never considered for the scope of this application.

The structure of address and position is the same for both collections and use a nested approach thanks to the MongoDb. Initially we thought of using address as primary key (two places cannot have the same address) since this technology permits complete check of the object, but this would have been incompatible with the RESTful application that have to use it.

We implemented by the usage of Docker, as simple technology that not only provides simple coverage but, in case of expansion of the project, is easy to put online as a final application.

6.6 Wheel

The library is implemented with the rather new technology of wheel.

This technology uses the file setup.py collocated in code_client: there it defines the properties of the wheel package by the library setuptools and its function setup.

Our project it is simple, so the required libraries to install defined inside this file are only pandas (used for reading and transforming classes in a pandas dataframe and reverse) requests (used for communicate with the Flask application) and gmplot (used for create maps based on Google Maps to visualize our data).

The file that bdist_wheel creates from that is easy to both use to install with virtualenvironments as libraries or even put it online on systems like pypi. We discarded the option after a discussion with the responsible of the module, that clarifies that it was not really required and could have potentially created issues with versions that would have costed precious time. However, the option is still available and relatively easy to put online by that. This means that the whole application is easy to public in every part, with simple few steps.

7 Results

The results consist in a RESTful Flask application interactable both directly from requests that from the Client Class Managers (HotelManager and PoiManager).

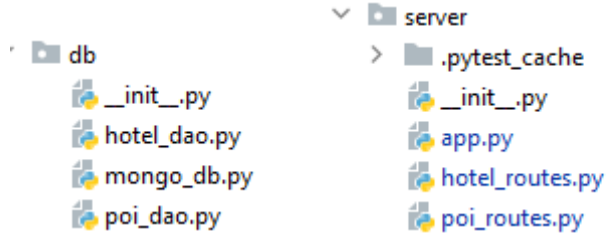
The data are successfully stored in a MongoDb database.

The project is at the moment private and stored in its GitLab repository, as directed primarily to our teacher and evaluator of the module. Public availability is not yet considered but since there is no real sensible information, in case of interest feel free to contact us by the following emails: dyuman.bulloni@student.supsi.ch, nathan.margni@student.supsi.ch

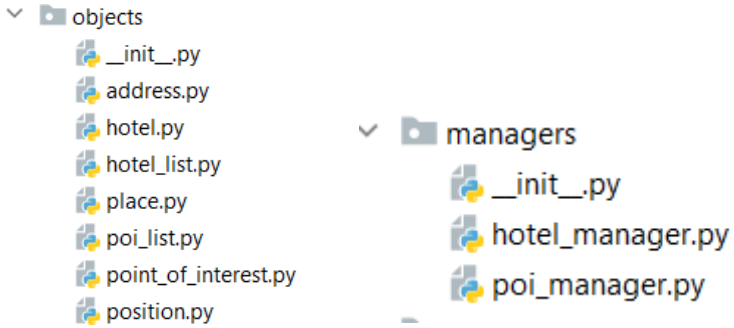
7.1 Code

This project is divided in two main sub-sections: backend and frontend.

The backend is then yet divided in two main categories, server and database.



The frontend is divided in two categories, objects and managers.



The first ones are the skeleton of the application, the instrument used for all types of internal and external passages and checks.

The managers are instead the classes intended to be the mean for the user (developer) to communicate with the web service, instead of the need of doing requests. Option that, of course, it is anyway available for them. The type of operations are limited to be the same directly in the respective routes.

7.2 Client-Server Communication

7.3.1 Hotels requests:

<url> to use for the request:

```
url = 'http://127.0.0.1:5000/hotels/'
```

(currently all managed locally)

GET:

URL	Description	Client Function with the same effect
<url>	return all the hotels stored in the object HotelList.	HotelManager.get_hotels()
<url>id	return the object Hotel corresponding the id.	HotelManager. get_hotel_by_id(hotel_id :int=)
<url>?sort_by=<param> [&direction=<value>]	Return all the hotels sorted by the <param> inserted. Direction is optional, 1 is for ascending order, -1 for descending order.	HotelManager.get_hotels(sort_by :str = <param>, direction : int=<value>)
<url>?[between=<param1>] [&start=<start>] [&end=<end>] [&sort_by=<param2>] [&direction=<value>]	Return the values with the condition of between two values. Useful for stars and price parameter (values for <param1>. sort_by function available.	HotelManager.get_hotels_by <param1>_range(start : float = <start>, end :float=<end> sort_by :str = <param2>, direction : int=<value>)
<url><params> [&sort_by=<param2>] [&direction=<value>]	Return the values that are compatible with the <params> defined. <params> is the set of the single values of the hotel. The nested ones requires the structure with '.'. Example: 'address.cityName'	HotelManager.get_hotels(params : dict = <param>) or already implemented: HotelManager.get_hotels_by • _city() • _country() • _name()

POST:

URL	Description	Client Function with the same effect
<url>	Add the Hotel object passed as payload in the json format.	HotelManager. add_hotel (hotel : Hotel)

PUT:

URL	Description	Client Function with the same effect
<url>id	Modify the Hotel object passed as payload in the json format.	HotelManager. update_hotel (hotel : Hotel)

DELETE:

URL	Description	Client Function with the same effect
<url>id	delete the object Hotel corresponding the id.	HotelManager. delete_hotel_by_id(hotel_id :int=)

7.3.2 Point of Interests requests:

```
url = 'http://127.0.0.1:5000/pois/'
```

GET:

URL	Description	Client Function with the same effect
<url>	return all the pois stored in the object PoiList.	PoiManager.get_pois()
<url>id	return the object PointOfInterest corresponding the id.	PoiManager. get_hotel_by_id(hotel_id :int=)
<url>?sort_by=<param> [&direction=<value>]	Return all the hotels sorted by the <param> inserted. Direction is optional, 1 is for ascending order, -1 for descending order.	PoiManager.get_hotels(sort_by :str = <param>, direction : int=<value>)
<url><params> [&sort_by=<param2>] [&direction=<value>]	Return the values that are compatible with the <params> defined. <params> is the set of the single values of the hotel. The nested ones requires the structure with '.'. Example: 'address.cityName'	PoiManager.get_hotels(params : dict = <param>) or already implemented: PoiManager.get_pois_by • _city() • _country() • _name() • category() • subcategory()

POST:

URL	Description	Client Function with the same effect
<url>	Add the PointOfInterest object passed as payload in the json format.	PoiManager. add_poi (poi : PointOfInterest)

PUT:

URL	Description	Client Function with the same effect
<url>id	Modify the PointOfInterest object passed as payload in the json format.	PoiManager. update_poi (poi : PointOfInterest)

DELETE:

URL	Description	Client Function with the same effect
<url>id	delete the object PointOfInterest corresponding the id.	PoiManager. delete_poi_by_id(poi_id :int=)

7.3 Not Obtained Results

7.3.1 Documented Code

As the project scope is mainly of evaluation, the cleanness of the code is still something that has to be considered as required result. The clearness is sufficient, explaining the main particular operations used, but it is something that during the work was overlooked and recovered only partially.

7.3.2 Unit test

The scope of being fully unit tested is not reached: this is attributed to a lack of understanding of the proper structure and a lack of investment of time by our part, cutting it partially for the well-being of other main features as the RESTful application.

7.3.3 Integration between the two objects

The domain of the project is the tourism but as the classes and implementation are now, hotels and point of interests are not directly linked. The usage of proper integration is still possible but left to the user. This is another cut we had to do as it was taking too much time as planning. We searched for possible solutions, such as searching point of interests near specific hotel or reverse. This feature would have required the confront of distances by the geographical coordinates. Even if the calculation is implemented, it is not used since even limiting our scope at the only Switzerland we are still left with almost 4000 hotels and around 300'000 points of interests. It is clear that repeating the calculation each time would require an enormous amount of time. We tried to think at possible solutions such as proper order in the data stored or the usage of a previously calculated parameter list, possible both by using MongoDb or alternatively and more optimized with another technology designed for the task, as Neo4J, a powerful graph database. At the end, each possible solution was considered too much time-consuming and therefore considered as possible extra for the project.

7.3.4 Interesting Demo

We initialized the process of creating an appealing way for the user to visualize the data, but we reached only an alpha version of a map, based on Google Maps.

It was originally designed also two visualizations for the hotels, such as a ordered histogram of the average costs by city, and a boxplot showing the rapport between stars of the hotel and their cost. Both of them were excluded due a poor time management, that prioritized critical features.

8 Conclusion

8.1 Personal Conclusions

The main objectives of the project are reached but not fulfilled. There was much more space for cleanness and integrating features.

This project would be a complete example of an implemented RESTful application created in Python, giving a solid infrastructure for future projects.

The technologies used were optimal for the task, but as first approach they required an unexpected greater amount of time that the one originally planned by us.

In retrospect, especially with personal past experiences of project management, the main problems that conducted to our limits were:

A lack of proper definition of functions: having defined the requirements, what we lacked was a clear limitation on our hands of the objectives. We tried to achieve many things, spending time in trying to figure out ways to obtain them. However, this distracted us from the basic steps really required and needed for the project. The search of data, their pre-processing and tentative to obtain useful information such as addresses from the point of interests with the usage of external libraries drained many time and effort from the real scope.

A lack of a proper strict timetable: we had the general timetable provided by the prof, but it was too general and not applicable to our case. Creating minor steps and putting them in for example a Gantt structure would have helped a lot. Also, we maintained only the first week a sort of daily journal. This was done as it was not required and considered a risk to keep since sometimes it would have required effort we preferred to spend elsewhere.

After this project is however clear to us that having to put down the daily progress is better in terms of understanding not only of the single work but especially of the partner's. We remained with basic guidelines and messages, having a brief meeting and explanation of the point reached at least 1-2 times per week, but this became more difficult to stick with the advent of other projects and exams overlapping.

8.2 Future developments

There is a lot of space for improvement, from cleanness of the code to excluded features already described in detail in the not obtained results.

The testing coverage part is surely one that did not have enough time dedicated during our project, and would be the first thing to cover properly, as finalising the solid basis of classes.

After that, the integration of hotels and point of interests would create a proper guide to tourism, effective domain of this project, improving the usability of the application by a lot.

As already described, searching hotels by the filter (specific city and range of cost, for example) would lead also to the nearest point of interests, with the primary information about them. This type of request, instead of the function for the user intended as developer, could be expanded to a user-friendly interface (from a simple web page o graphical user interface from Python).

Another improvement would be increasing the coverage of this project: we already disposed of hotels from the whole Europe, but for the points of interests the amount of data was simply too large to consider: we had almost 300'000 points of interests only in the relatively small Switzerland. Having large amount of data was not really a problem, but for the scope of this project it seemed not necessary. Searches by country are already implemented both by the RESTful application that the respective managers of the client library.