Project: “Organic and local restoration advice”

In this project, you play the role of a consulting company, which connects its clients, restaurateurs with producers. The goal is to allow restaurateurs to create a menu based on fresh, organic products with the best possible carbon footprint.

Your company provides its customers with an API that can be queried by each of them. This API allows customers to identify themselves and formulate their request (ingredients, reference location). Finally, when the request is correctly formulated, the client can ask the API to provide it with a list of producers.

The code created in the TDs ?? to ??, you are making great progress in this work. It should be noted that corrections for each of these TDs are available on moodle.

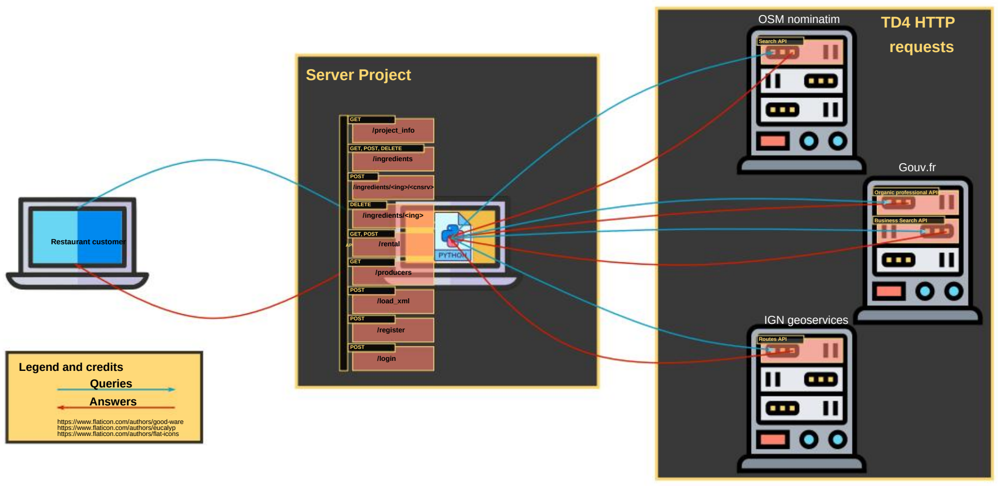

Figure 1 – Overall architecture of the project

The part you are evaluated on is labeled "Server Project" in Figure 1. It includes the server and all the processing it will need to perform to respond to client requests.

This means that the test client that you will have to create to test your server does not have to be rendered. We will only rate the server.

1 Rendering

Rendering (server) will be done via gitlab. You will need to add users to your project
—Audrey Serna
— Mathieu Loiseau
—Ludovic Moncla

You will send us an email to give us the URL of your project.

2 Specifications

To provide the requested services, your server will need to manage a number of endpoints. These endpoints will call functions which themselves will use code
written in previous TDs. All endpoints are listed below. Up to you
if you prefer to separate them — assign a function to each pair (route, http method) —
or group together certain endpoints that process the same “route”.

Your API must scrupulously follow the following specifications.

2.1 Listening port

Your server will need to listen to port 5080.

2.2 /project_info GET endpoint

This endpoint should give us all the information about your project:
— the address of the deposit
— the list of project members
— what authentication you have set up (see section 3): null, “IP” or “account”
— what data storage strategy you have put in place (see section 4): null, "serialization" or "sqlite"

Description	Returns information about the project	
Request	GET method	
	Parameters Ø	
Answer	Status 200	
	Format	JSON (dict) : {"group ":" GI3.1.4 ", „ depot ":" https :// gitlab .insa - lyon .fr/ cbd /g1 -4",„ " account " „ authentication ":"IP", /* or or null */ „ storage ":" serialization ", /* or members ":[" sqlite " or null */ {"firstname ":" John "," lastname":"Doe "}, {" first name ":" Alain ", " last name ":" Known "}, /* ... */] }

Table 1 – /project_info GET endpoint

2.3 Route /ingredients

This route can be queried with 3 different methods.

a) GET

Request	Method	GET
	Parameters	Ø
Answer	Status	200
	Description	Returns the current status of the ingredient list as entered by the customer.
	Format	JSON (dict) : {"ingredient_name1": storage time in days, ...}
	Example	{"Fresh bread":2, "Potatoes":30}

Table 2 – GET /ingredients endpoint

b) POST

Description	Replaces the ingredient list with the one sent by the customer and returns it	
Request	method	POST
	JSON parameters	(dict) : {"ingredient_name1": storage time (d), ...}
Answer	Status	200
	Format	JSON (dict) : {"ingredient_name1": storage time (d), ...}
	Example	{"Fresh bread":2, "Potatoes":30}

Table 3 – POST endpoint /ingredients

c) DELETE

Description	Clears the ingredient list and returns a text message	
Request	Method	DELETE
	Parameters	Ø
Answer	Status	200
	Format	Text: A message indicating whether the request worked

Table 4 – DELETE endpoint /ingredients

2.4 /ingredients/<ing>/<cnsrv> POST endpoint

Description	Adds an ingredient to the list and returns the current state of the list	
Request	Method	POST
	Parameters in	URL
Answer	Status	200 OK
		304 no change
	Format	JSON (dict) : {"ingredient_name1": storage time (d), ...}
	Example	{"Fresh bread":2, "Potatoes":30}

Table 5 – /ingredients/<ing>/<cnsrv> POST endpoint

2.5 /ingredients/<ing> DELETE endpoint

Description	Clears the specified ingredient from the ingredient list and returns the list status	
Request	Method	DELETE
	Parameters in	URL
Answer	Status	200 OK
		304 no change
	Format	JSON (list) : ["ingredient1","ingredient2", ...]
	Example	["Fresh bread", "Potatoes"]

Table 6 – /ingredients/<ing> DELETE endpoint

2.6 Route /rental

This route can be queried with 2 different methods.

a) GET

Description	Returns the current state of the reference address (the address where we would like to establish a restaurant)	
Request	Method	GET
	Parameters	Ø
Answer	Status	200
	Format	JSON (dict) : {"street":"address", "city":"name of the city"}
	Example	{"city":"Rennes", "street":"5, allée Geoffroy de Pontblanc"}

Table 7 – GET endpoint /location

b) POST

Description	Replaces the reference address with the one sent by the customer and returns it	
Request	Method	POST
	JSON parameters	(dict) : {"street":"address", "city":"name of the city"}
Answer	Status	200
	Format	JSON (dict) : {"street":"address", "city":"name of the city"}
	Example	{"city":"Rennes", "street":"5, allée Geoffroy de Pontblanc"}

Table 8 – POST endpoint /location

2.7 Route /producers

This route can be queried with only 1 method.

a) GET

Description	Uses input data stored on the server (with routes ingredients and location) as input data for functions created in the **TD??** and returns to the customer a producer for each commodity, as well as the distance in km which separates it of the reference address		
Request	Method	GET	
	Parameters Ø		
Answer	Status	200	OK
		400	parameters are missing
	Format	400	JSON (list) indicating missing parameters
		200	JSON (dict) associating to each product, a company, its manager's name and a distance in km
	Example	400	["location","ingredients"] or ["ingredients"] or ["location"]

		200	{
			" Fresh bread ":{
			" Company ":" MONTOIR MATHIEU ",
			" Manager ":" Mathieu Montoir ",
			" Distance " :20.6
			},
			" Potatoes ":{
			" Company ":" EMPLOYMENT SPACE - JARDINS DU BREIL ",
			" Manager ":" Unknown ",
			" Distance " :4.57
			}
			}

Table 9 – /producers GET endpoint

2.8 /load_xml POST endpoint

This endpoint allows you to send an XML file containing all the problem data (and thus handle the location and ingredients in one request).

To carry out the associated processing, you will need to adapt the parser, and use the file sending examples available on moodle: —
file_upload_server.py —
file_upload_client.py

Description	Replaces the reference address and the list of ingredients according to the data in the XML file (structured according to the schema of TD no. 3) received.	
Request	Method	POST
	Settings File Send	
Answer	Status	200 OK 400
		Bad request (no file, malformed file, etc.) text : explanation of the error / "OK"
	Format	if successful Response("Error parsing the file", status=200)
	Example	

Table 10 – /load_xml POST endpoint

3 Authentication

It will be necessary to be able to manage several users, this means that the problem data will be associated with one user and only one. To do this, we can do things in several ways (see scale): 1. use the IP address to distinguish users (problem of changing IP, or having several users from the same address — shared computer, boxes); 2. the user can be asked to create an account, then authenticate.

The first solution does not require a specific endpoint, it is the one we expect by default.

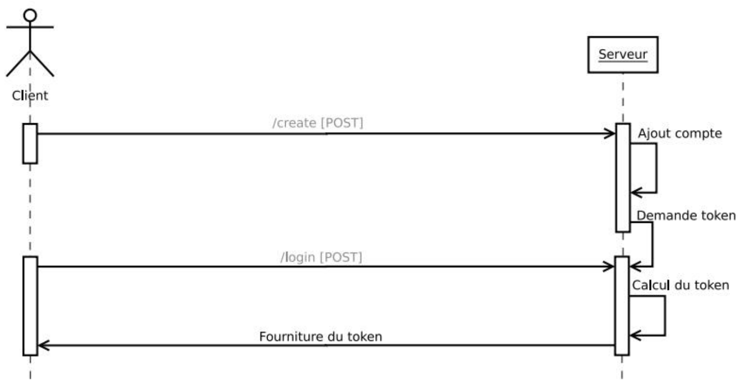

Figure 2 – Account Authentication Sequence Diagram

The second is more complex, but closer to reality (although we would use functionalities of authentications integrated into flask [1]). To process this strategy, we will follow the sequence diagram in Figure 2, this adds two endpoints and requires adjusting all the others.

3.1 Route /register POST

Description	The user chooses an account identifier. If it already exists the server returns an error, otherwise it returns an authentication token.	
Request	Method	POST
	JSON Parameters (dict) : {"login":"username", "password":"1dGZ*neo8P!"}	
Answer	Status	200 Account creation was successful
		400 Failed to create account (username existing or not provided)
	Format	200 text : random identification token in format (\d\w[%*:.-~=]){10}
		400 JSON (dict) : {"error":"explanation of the error"}
	Example	200 text : "1Rv*3Rt4c%"
		400 JSON (dict) : {"error":"user name already exists"}

Table 11 – /register POST endpoint

3.2 Route /login POST

Description	The user logs in using his login and password. If the authentication is successful, the server returns a new token. Otherwise it sends an error.	
Request	POST method	
	JSON parameters (dict) : {"login":"username", "password":"1dGZ*neo8P!"}	
Answer	Status	200 Login was successful
		401 Login failed
	Format	200 text : random identification token in format (\d\w[%*:.-~=]){10}
		400 JSON (dict) : {"error":"explanation of the error"}
	Example	200 text : "13Rt4c%Rv*"
		401 JSON (dict) : {"error":"bad login/password combination"}

Table 12 – POST /login endpoint

3.3 Consequences

Caution For groups that choose to deploy authentication, this method will have consequences on **all** other endpoints. They will all have to verify that JSON data are sent, include a "token" field and that the latter contains a value associated with a existing user account. If the token is not good, you will have to return a 401 error.

1. https://flask-login.readthedocs.io/en/latest/

4 Data storage on the server

So that users do not start from 0 each time the server is restarted. It would be relevant that it retains data. This can be done both using serialization (see TD 2) and a database (e.g. SQLite, see section 5 of TD 5).

5 Scale

The scale is indicative: the number of points associated with an endpoint may fluctuate. THE Point counts shown assume that endpoints follow specifications and work. It is possible that a functional endpoint that does not follow the specification is evaluated as 0.

The scale suggests an order for resolving the problem.

Note	Functionality according to specifications
1.5	/project_info GET
4	/ingredients GET /ingredients/<ing>/<csrv> POST /location GET
2	/POST rental
4+2	/producers GET (2 points off scale for optimizations)
1.5	/ingredients DELETE /ingredients POST
0.5	/ingredients/<ingred> DELETE
4	/load_xml POST
2	data storage strategy
0.5+2	authentication strategy (2 points off scale for authentication) /register POST /login GET

Table 13 – Indicative scale **(maximum score is 20)**