Steps:

* Review the content and plan forward
* Found on my\_vm the collection “COCO” to use as a testing bed for the API
* Decision to go for **Connexion** (<https://github.com/spec-first/connexion>), notice this is **Flask**

**First task:**

Do Spec 🡪 OpenAPI 3 [DONE! ]

**SECOND TASK:**

Change the file default\_controller.py to implement the API actions

In order to perform this step use as “asset” (api meaning) the collection already instantiated by Luca for the COCO Dataset.

**Partire ad implementare con la search**

(Idea: Copia il dataset di COCO e lavora in locale insieme a MongoDB Compass)

**NEXT ??**

* Bisogna capire l’integrazione con la TUI di Luca per riuscire ad avere accesso al Parallel File System (Consiglio Tony: Fare una repo, libreria Python, in cui si installerà su “my\_vm”, che si occupa di fare solo il lavoro di integrazione)
* Bisogna integrare S3, a patto che non sia già integrata con la TUI di Luca 🡪 S3 sarà semi integrata. Visto che il bucket s3 è su una repo dentro pfs, automatimaticamente quello caricato dentro s3 è anche caricato su pfs. Nella parte inversa il concetto non vale (bisogna fare un curl per riuscire a sincronizzare i due)
* (Bisogna generare un esempio più generale di collections per handling anche dati non immagini) ok
* Optional\* Creare una gRPC, non di sostituzione ma complementare alla REST API creata.

AUTENTICAZIONE 🡪 La RESTapi deve gestire l’autenticazione tramite un token (vedi white paper di axion)

I token dovrebbero essere ad personam, permettendo un’idnetificazione. In più per il logging, è facile importare pacchetti “logging”

Il range di IP che ha accesso si setta durante l’accesso al Tenant di Adacloud

3 N point architecture

* Upload 🡪 L’upload deve trasformare la rest in una richiesta **di upload sul BUCKET S3** (che è dentro il PFS)
* Search only 🡪 stringa SQL in INPUT + (JOB specifications??)
* *Search* (**Analize**) 🡪 stringa SQL in INPUT + FILE PYTHON DI ANALISI, Output utente 🡪conferma del successo della richiesta /// Output TUI 🡪 JSON + File Python da mettere nella cartella tmp della cartella??? (-- File Python viene tramutato in stringa) (L’unico parametro per i JOB da includere sono il numero di nodi)

**Output JSON OBJECT con SQL\_QUERY =…. PYTHON\_QUERY=…**

* Download 🡪 restituisce il file
* \*Optional CLI

Tentative Diagram of the Service

Come fare ritornare I risultati dell’analisi python garantita di TUI all’utente?

Far ritornare uno script

Lo scopo dell’UPLOAD

JSON in input da curl {

Id utente

Path su s3

*Path su file system*

Data di upload

Estensione

+ Tutti I metadati del caso per il file uploadato

}

OUTPUT su VM??:

Push su bucket s3 (Parti con il caricamento sulla VM, poi si trova il modo di fare la copia dalla VM a PFS)

Insert entry in MongoDB

Nel JSON di SEARCH trasformare dall’input Utente e mandare in ouput alla VM:

In input da curl: {

Query\_SQL: **mandatory**

Script\_python: **optional**(

N\_nodes: **optional**

}

In Output sulla VM:

{ Query\_SQL: **mandatory**

Script\_python: **optional**(

N\_nodes: **optional**

}

Lo scopo del DOWNLOAD

Input: Path/oggetto di download

Output: file al path del download

Select Images, Video 🡪

Collection MongoDB –>

{ Images:

Image\_Size

Image\_Length

Video:

Video\_Size:

Video\_Length:

}

[Discorso Luca

RestAPI chiama l’interfaccia TUI

L’interfaccia TUI manda un JOB con 2 step (1 interrogazione di api in python di mongoDB per ottenere lista di file, Data la lista si trasforma in JSON e passarla al motore che fa l’analisi OPPURE utilizzare standard In Standard Out ((programmino python dell’utente??))

I file possono essere slavati o in Place, o come Nuovi file dentro il Datalake.

L’API deve passare alla TUI una stringa (che sarà il comando SQL probabilmente) 🡪 bada bene che questa deve essere formattata in modo tale da riuscire a formare un dictionary con tutte le informazioni che devono darti.

La stringa sarà utilizzata per fare la query sul HPC

A diagram of a diagram

Description automatically generated

**Notes from morning 24/10 Discussion**

**Authentication**

We discussed also Authenticaton and Login Issues.

These will be addressed later as it is necessary to have input on Cineca

**Technical**

Is it important where the data are stored for API concerns?

Yes and no. Yes cause we can set a entry in MongoDB regarding Source: S3 or Source: PFS

No because it will be found

|  |
| --- |
| openapi: 3.0.3  info:    title: DTaaS API    description: This API handles the communication between the virtual infrastructure generated by the DTaaS codebase and the User. The goal is to allow to Search, Upload and Donwload data from a Dual S3/PFS Datalake.    version: 1.0.0  servers:    - url: http://localhost:8080/v1    paths:    /search:      get:        summary: Search COCO dataset        parameters:          - name: category            in: query            description: Category of items to search for            required: false            schema:              type: string          - name: pageSize            in: query            description: Number of items to return per page            required: false            schema:              type: integer              default: 10          - name: pageToken            in: query            description: Token for pagination            required: false            schema:              type: string        responses:          '200':            description: Successful Search            content:              application/json:                schema:                  type: array                  items:                    $ref: '#/components/schemas/Asset'    /upload:      post:        summary: Upload an item to the COCO dataset        requestBody:          required: true          content:            multipart/form-data:              schema:                type: object                properties:                  file:                    type: string                    format: binary                  metadata:                    type: string        responses:          '201':            description: Upload Successful          '400':            description: Bad Request    /download/{id}:      get:        summary: Download an item from the COCO dataset        parameters:          - name: id            in: path            required: true            schema:              type: string        responses:          '200':            description: Download Successful            content:              application/octet-stream:                schema:                  type: string                  format: binary          '404':            description: Not Found  components:    schemas:      Asset:        type: object        properties:          \_id:            type: string          id:            type: integer          path:            type: string          height:            type: integer          width:            type: integer          captured:            type: string            format: date-time          caption:            type: string          segmentation:            type: array            items:              type: number          area:            type: number          iscrowd:            type: integer          bbox:            type: array            items:              type: number |

**MOLTO IMPORTANTE IL FILE default\_controller.py**

* **Riguarda implementazione di FastAPI e scrivi quello che c’è da scrivere (tipo interazioni con mongo su tale file)**

**A group of folders with icons

Description automatically generated**

Sopra abbiamo la cartella swagger server, che è la cartella principale all’interno del folder generato automaticamente a seguito della creazione della Spec dell’API nell’editor swagger editor.

Fuori da questa cartella ci sono file per utilità di integrazione per esempio

-un file git push

- Un file Travis (Travis = TOOL per CI/CD)

E altre simili

**L’importante all’interno della cartella è CONTROLLERS,** specificamente **default\_controller.py**Ricorda che tale file è denominato default in quanto nel nostro use case abbiamo solamente un tipo di TAG dove tag è considerato il seguente (dalla guida di SWAGGER EDITOR )

|  |
| --- |
| tags:  - name: **pet**  description: Everything about your Pets  externalDocs:  description: Find out more  url: http://swagger.io  - name: **store**  description: Access to Petstore orders  externalDocs:  description: Find out more about our store  url: http://swagger.io  - name: **user** |

QUINDI TUTTE LE MODIFICHE DI CODICE VANNO FATTE **RIMPIAZZANDO IL “DO SOME MAGIC”** dentro file default\_controller.py

In generale un controller è l’entità utile a performare le CRUD operations per intenderci

Tornando alla cartella in figura

* MODELS = Serve alla validation delle request (per esempio se byte\_size\_max in model = 1000 se un utente chiede un file byte\_size = 2000 IN AUTMOATICO esce una bad request.
* TEST = Questo server a definire dei test per la api e vale la pena giocarci domani
* SWAGGER = contiene il file yaml con le definizione delle spec dell’api con in aggiunta altre specifiche
* PYCACHE= Sono dei runable file temporanei di funzione a metà tra Cache e Log (di solito aggiunto a git ignore)
* Encoder.py per fare file in json
* Utils.py per serializzare e deserielizzare il dato (CLASSE🡨🡪Dizionario 🡨🡪 JSON)

DB datalake

Collection metadata

Per le immagini di COCO train2014.zip e val2014.zip

Per i JSON ha messo mano, posso scaricarli dalla VM

LN -S 🡪 per traslare il test dalla locale alla my\_vm

Ivan 🡪 Scaricato Mongo e MongoSH in locale.

To DO: Creare in Locale da MongoCompass la collection come da collection su my\_vm

* Per fare ciò scaricare da my\_vm ai nodi di Login il file COCO.json
* Scaricare dai nodi di login a locale il file COCO.json

Scaricare i 13 GB di dati per le immagini coco, puoi eliminarle in fretta dopo aver aggiunto alcune per le prove.

To DO: Una volta che Collection è istanziata, testare le funzioni Upload e Download della API

* Lanciare l’API
* Verificare che il collegamento API e MongoDB avvenga correttamente
* Lanciare le query dall’editor swagger dopo il lancio dell’API in locale
* Verificare i risultati sono quelli desiderati.

Per quanto riguarda la SEARCH API

Deve restituire in output non un JSON ma una stringa formattata come di seguito

--query “””SELECT \* FROM MYTABLE””” --script “”import pandas as panda….”””

26/10

*Dubbi sul Download:  
Al momento si riesce a scaricare solo 1 item basandosi sul path*

*L’utente non ha possibilità di scaricare più file a meno che non siano in un .zip da cui fare il download.*

Dopo confronto tra developers, il comportamento sopradescritto risulta **OK**

Step Affrontati nel corso della giornata:

* Abilitato il Testing dell’API In Locale.
  + Creato dataset 108 immagini (random, aerei) con metadati di cocodataset
    - Attenzione! Il file COCO.json è stato modificato in modo da aver la key path puntante su i corrispettivi path dei file in locale
  + Creata collection mongodb in locale ( connection= localhost:27017 ; db=local ; collection=metadata)
    - Attenzione! Questo path è hardcoded nella soluzione e va modificato una volta importato il codice nella VM
  + Creato virtual environment python su vscode per riuscire a lanciare il server della flask api connexion
    - Attenzione! Il requirement.txt è stato modificato per assicura che pymongo sia installato
* **Modificate le API SPEC,** al fine di riuscire ad avere dei post invece che delle get
  + Il **Download** è rimasto GET input path and output the file
    - Attenzione! Solo un singolo file in donwnload! Leggi descrizione sopra per conferma necessità di un .zip
  + **Upload** è stato modificato per permettere un upload di un json\_file e di un file\_generico
    - Attenzione! al momento nessun check sul tipo di metadati caricati, i.e. non devono rispettare un format predefinito di collection
    - Attenzione! Non sono aggiunti altri metadati desiderati al file json al momento.
  + **Search** è stato modificato in **query\_and\_process** e ora corrisponde a un **post** che accetta un **file\_query e un file\_python**
    - Attenzione! Va sviluppata
* Sviluppata **download GET**
  + - Attenzione! Si punta a “/download/id”, volendo si può puntare a “/download/path”
* Sviluppata **upload POST**
  + Creato un comportamento nel caso che il metadato creato sia già presente all’interno della collection. Ovvero se si uploada un file (che è assunto, non testato dall’api, che sia correttamente puntato dal key-value pare path nel metadato) assieme ad un metadato già presente in collection, dove già presente indica che lo stesso path è presente in un elemento nella collection, allora non si aggiunge una nuova entry e si comunica allo user che c’è un replacement, vedi figura in fondo
    - Attenzione! Solo un file e un metadato viene caricato al momento. **Bisogna permettere l’upload di multipli dati e metadati in un singolo colpo,** (oppure no? Semplicemente fare un for con many API requests per ogni dato e associato metadato? ) 🡪 **Confermato con Eric che no, basta il singolo upload di dato e metadato, verà poi loopato per fare many post request nel caso di tanti dati**
    - Attenzione! **Nessun metadato aggiuntivo assegnato alla** collection (eg. User name, ora...)

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**TO DO:**

* Implementare query\_and\_process
* Migliorare l’upload con i metadati aggiuntivi
* Riguardare i punti di attenzione e discuterli con i team members
* **Gestire Update 🡪 PUT o PATCH e Delete**

Reminders:

Bisognerà gestirsi l’upload su S3 quindi implementare boto e metterlo nei requirements

Per quanto riguarda la SEARCH API

Deve restituire in output non un JSON ma una stringa formattata come di seguito

--query “””SELECT \* FROM MYTABLE””” --script “”import pandas as panda….”””

AUTENTICAZIONE 🡪 La RESTapi deve gestire l’autenticazione tramite un token (vedi white paper di axion)

I token dovrebbero essere ad personam, permettendo un’idnetificazione. In più per il logging, è facile importare pacchetti “logging”

Il range di IP che ha accesso si setta durante l’accesso al Tenant di Adacloud

OUTPUT su VM??:

Push su bucket s3 (Parti con il caricamento sulla VM, poi si trova il modo di fare la copia dalla VM a PFS)

Insert entry in MongoDB

Nel JSON di SEARCH trasformare dall’input Utente e mandare in ouput alla VM:

In input da curl: {

Query\_SQL: **mandatory**

Script\_python: **optional**(

N\_nodes: **optional**

}

In Output sulla VM:

{ Query\_SQL: **mandatory**

Script\_python: **optional**(

N\_nodes: **optional**

}

**27/10**

* **Aggiornate le SPEC** 
  + DELETE -> ok per single object dato esclusivamente il path cancella file e metadato
  + PATCH 🡪 Cambiare metodo di inserimento dati?
  + PUT 🡪 può cambiare file e metadato, o, solo file (associato a vecchio metadato)
  + **Query and process🡪 basic ok**
  + Inserire un download solo metadato? Cambiare Downlaod attuale ?

**TO DO:**

* Migliorare query\_and\_process
* Migliorare l’upload con i metadati aggiuntivi
* Riguardare i punti di attenzione e discuterli con i team members

To do futuri:

* Aggiornare la documentazione
* Sistemare con S3
* Settare collection definitiva
* Integrare TUI
* Rivedere sviluppi E4 e integrarli
* Test totale In Ambienti (Leonardo???)

**3/11/2023**

Dopo aver risistemato l’environment con centos8 sulla nuova my\_vm

Abbiamo iniziato a correggere librDL.yml e vorremmo chiedere il testing da parte di E4

**Come installare la git REPO su my\_vm🡪 installare git su my\_vm (problema infrastruttura)**

**(API) Bisogna fare il post dei file, e passare il path dei file alla tui, non passare il testo fra virgolette:**

**NO** --query “””SELECT \* FROM MYTABLE””” --script “”import pandas as panda….””” **NO**

**Updated:**

Uuid,library for creating ID

Dtaas\_tui\_server “{\”query\”: SELECT …. \”script\”: import pandas…}

Venerdì 10 Novembre

Aggiornati I requirements in modo che funzioni con python 3.9

Aggiustati i vari path e caricati i metadati con path aggiornati su mongodb

A prescindere, il **Download Non Funziona** non so perchè, i file sono accessibili (r and w)🡪 problema con gli absolute paths not being recognized, actually is the server not able to get out of its own folder to look at other resources on the machine it seems. Bada bene **quando ti ritorna il 200 non sai come scaricare i file**

**Query and Process works fine**

Non vedo i messaggi “print/return” di python che vedevo con python, come risolvere? 🡪 quando funziona li vedo,

ho problema di directories e paths come al solito.

Troubleshooting:   
“””

|  |
| --- |
| i noticed this difference in behavior: (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ pwd /home/centos/dtaas\_test\_api (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ curl -X 'GET' \ > 'http://localhost:8080/v1/download/%2Fhome%2Fcentos%2Fdtaas\_test\_api%2FpriceDetail.png' \ > -H 'accept: application/octet-stream' { "detail": "The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.", "status": 404, "title": "Not Found", "type": "about:blank" } (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ curl -X 'GET' \ > 'http://localhost:8080/v1/download/home%2Fcentos%2Fdtaas\_test\_api%2FpriceDetail.png' \ > -H 'accept: application/octet-stream' { "detail": "The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.", "status": 404, "title": "Not Found", "type": "about:blank" } (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ curl -X 'GET' \ > 'http://localhost:8080/v1/download/priceDetail.png' \ > -H 'accept: application/octet-stream' it looks as if in the first case the file was checked but not found, in the second was directly not found. am i right? this suggests the server has no freedom to access it right? given this tree structure: -priceDetail.png -datalake\_api\swagger\_server how can i make sure the server can reach other folders in the OS  ChatGPT  **ChatGPT**  Your observations about the different behaviors of the **curl** commands are insightful. Let's analyze each scenario:  **Analyzing the curl Commands**   1. **First Command (Absolute Path)**:    * URL: **'http://localhost:8080/v1/download/%2Fhome%2Fcentos%2Fdtaas\_test\_api%2FpriceDetail.png'**    * The **%2F** at the beginning represents **/**, making it an absolute path.    * The server seems to recognize the path but cannot find the file, likely due to issues with file access or path interpretation by the server. 2. **Second Command (Modified Absolute Path)**:    * URL: **'http://localhost:8080/v1/download/home%2Fcentos%2Fdtaas\_test\_api%2FpriceDetail.png'**    * Here, the absence of the initial **%2F** makes it a relative path from the server's current working directory.    * The server likely interprets this as a non-existent route, hence the immediate 404 response. 3. **Third Command (Relative Path)**:    * URL: **'http://localhost:8080/v1/download/priceDetail.png'**    * This is a relative path and would work if **priceDetail.png** is in the server's current working directory.   **Understanding Server's Directory Access**   * The server's ability to access files outside its working directory depends on the server configuration, file permissions, and potentially security policies like SELinux.   **Ensuring Server Access to Other Folders**   1. **Adjust the Server Working Directory**:    * If feasible, change the working directory of the server to a location that has access to the required files. 2. **Use Symbolic Links**:    * Create symbolic links within the server's working directory that point to the desired files in other locations.    * For example, you can create a symlink to **priceDetail.png** in the server's working directory:   bashCopy code  ln -s /home/centos/dtaas\_test\_api/priceDetail.png /path/to/server/working/dir/priceDetail.png   1. **Modify Server Code**:    * Adjust the server code to handle file paths more flexibly, possibly using whitelists of allowed directories or more complex path resolution logic. 2. **Review Security Policies**:    * If using SELinux or similar, ensure it's not restricting file access. 3. **Check File Permissions**:    * Ensure the server process has the necessary permissions to access the files. 4. **Server Configuration**:    * Depending on how the server is set up (e.g., running in a container, specific user permissions), you might need to adjust its configuration to allow access to certain directories. |

*I solve with symbolic link and using relative paths:*

ln -s /home/centos/dtaas\_test\_api/COCO\_dataset/ /home/centos/dtaas\_test\_api/datalake\_api/COCO\_dataset/

but I can also use import os

os.chdir(/home) on the main flask app (where app.run, with debug, is)

**delete non funziona per path**

Problemi con **upload:**

|  |
| --- |
| (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ curl -X 'POST' \  > 'http://localhost:8080/v1/upload' \  > -H 'accept: text/plain' \  > -H 'Content-Type: multipart/form-data' \  > -F 'file=@SINGLE\_cat\_entry\_metadata\_test.json;type=application/json' \  > -F 'json\_data=@priceDetail.png;type=image/png'  "Upload Failed: [Errno 2] No such file or directory: 'C:\\\\Users\\\\IvanGentile\\\\OneDrive - Net Service S.p.A\\\\Desktop\\\\API\_test\_files\\\\COCO\_dataset/SINGLE\_cat\_entry\_metadata\_test.json'"  (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ ls  api\_local\_install\_\_env datalake\_api queryTry.txt  COCO\_dataset priceDetail.png SINGLE\_cat\_entry\_metadata\_test.json  COCO.json pyTry.py single\_entry\_metadata\_test.json  (api\_local\_install\_\_env) [centos@my-vm dtaas\_test\_api]$ vi SINGLE\_cat\_entry\_metadata\_test.json |