## Al momento sul github cineca il codice sotto per un esempio di collection: CODICE SU VM ADESSO

|  |
| --- |
| # Importing necessary modules  from fastapi import FastAPI, HTTPException  from motor.motor\_asyncio import AsyncIOMotorClient  from fastapi.middleware.cors import CORSMiddleware  import uuid  # Instantiating FastAPI application  app = FastAPI()  # Adding CORS middleware to the application to handle Cross-Origin Resource Sharing  app.add\_middleware(  CORSMiddleware,  allow\_origins=["\*"],  allow\_credentials=True,  allow\_methods=["\*"],  allow\_headers=["\*"],  )  # Establishing connection to MongoDB on startup  @app.on\_event("startup")  async def startup\_event():  """Initializes MongoDB client and database when the server starts."""  app.state.mongodb = AsyncIOMotorClient('mongodb://localhost:27017')  app.state.db = app.state.mongodb['my\_database']  # Closing connection to MongoDB on shutdown  @app.on\_event("shutdown")  async def shutdown\_event():  """Closes MongoDB client when the server shuts down."""  app.state.mongodb.close()  # GET endpoint to fetch all items in the 'test\_collection' collection  @app.get("/items/")  async def read\_items():  """Fetches all documents in the 'test\_collection' collection."""  items = []  cursor = app.state.db['test\_collection'].find()  async for document in cursor:  # Convert ObjectId to string  document["\_id"] = str(document["\_id"])  items.append(document)  return items  # GET endpoint to fetch a specific item by its id  @app.get("/items/{item\_id}")  async def read\_item(item\_id: str):  """Fetches a single document in the 'test\_collection' collection by its 'id'."""  item = await app.state.db['test\_collection'].find\_one({"id": item\_id})  if item is None:  raise HTTPException(status\_code=404, detail="Item not found")  # Convert ObjectId to string  item["\_id"] = str(item["\_id"])  return item  # POST endpoint to create a new item in the 'test\_collection' collection  @app.post("/items/")  async def create\_item(item: dict):  """Creates a new document in the 'test\_collection' collection."""  # Add a unique id to the item  item["id"] = str(uuid.uuid4())  new\_item = await app.state.db['test\_collection'].insert\_one(item)  return {"item\_id": str(new\_item.inserted\_id)}  # PUT endpoint to update an existing item by its id  @app.put("/items/{item\_id}")  async def update\_item(item\_id: str, item: dict):  """Updates a specific document in the 'test\_collection' collection by its 'id'."""  item['id'] = item\_id  updated\_item = await app.state.db['test\_collection'].replace\_one({"id": item\_id}, item)  if updated\_item.matched\_count == 0:  raise HTTPException(status\_code=404, detail="Item not found")  return {"item\_id": item\_id}  # DELETE endpoint to delete an item by its id  @app.delete("/items/{item\_id}")  async def delete\_item(item\_id: str):  """Deletes a specific document in the 'test\_collection' collection by its 'id'."""  deleted\_item = await app.state.db['test\_collection'].delete\_one({"id": item\_id})  if deleted\_item.deleted\_count == 0:  raise HTTPException(status\_code=404, detail="Item not found")  return {"item\_id": item\_id} |

# Possibili TAGS per le COLLECTIONS MONGODB

Visto che non sappiamo che tipo di dati andremo ad utilizzare potremmo creare delle blank connection che verrano usate a seconda del tipo di file.

Per esempio: (Questi Schema Sono tutti One-to-One. Giusto? [MongoDB Schema Design Best Practices | MongoDB](https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/))

|  |  |  |
| --- | --- | --- |
| Image-like | **image\_id**, **creation\_date** | **image\_id**, **creation\_date**, **image\_size**, **image\_color**, **image\_resolution**, **image\_format** |
| Video-like | **video\_id**, **creation\_date** | **video\_id**, **creation\_date**, **video\_length**, **video\_format**, **video\_resolution**, **video\_bitrate** |
| Text-like | **text\_id**, **creation\_date** | **text\_id**, **creation\_date**, **text\_length**, **text\_language**, **text\_format**, **tags** |

Given the above We can write the APIs something like this maybe?

Pydantic Models for Request Validation

| **Data Type** | **Pydantic Model Fields** |
| --- | --- |
| Image-like | **image\_id: str**, **image\_size: Optional[str]**, **image\_color: Optional[str]** |
| Video-like | **video\_id: str**, **video\_length: Optional[str]**, **video\_format: Optional[str]** |
| Text-like | **text\_id: str**, **text\_length: Optional[str]**, **text\_language: Optional[str]** |

CRUD Endpoints

| **Data Type** | **CRUD Operation** | **API URL** | **Method** |
| --- | --- | --- | --- |
| Image-like | Create | **/images** | POST |
|  | Read | **/images/{image\_id}** | GET |
|  | Update | **/images/{image\_id}** | PUT |
|  | Delete | **/images/{image\_id}** | DELETE |
| Video-like | Create | **/videos** | POST |
|  | Read | **/videos/{video\_id}** | GET |
|  | Update | **/videos/{video\_id}** | PUT |
|  | Delete | **/videos/{video\_id}** | DELETE |
| Text-like | Create | **/texts** | POST |
|  | Read | **/texts/{text\_id}** | GET |
|  | Update | **/texts/{text\_id}** | PUT |
|  | Delete | **/texts/{text\_id}** | DELETE |

How to decide when to **Embed** and when to **Reference**?

[MongoDB Schema Design Best Practices | MongoDB](https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/)

## APPPENDIX

# Codice per i Tags Collections MongoDB

from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")

db = client["mydatabase"]

Image-like Data - MongoDB Collection

For light tagging:

python

Copy code

image\_light\_collection = db["image\_light"]

image\_light\_sample = {

"image\_id": "2u3509032",

"creation\_date": "2023-01-01"

}

image\_light\_collection.insert\_one(image\_light\_sample)

For detailed tagging:

python

Copy code

image\_detailed\_collection = db["image\_detailed"]

image\_detailed\_sample = {

"image\_id": "2u3509032",

"creation\_date": "2023-01-01",

"image\_size": "343kB",

"image\_color": "Red",

"image\_resolution": "1024x768",

"image\_format": "PNG"

}

image\_detailed\_collection.insert\_one(image\_detailed\_sample)

Video-like Data - MongoDB Collection

For light tagging:

python

Copy code

video\_light\_collection = db["video\_light"]

video\_light\_sample = {

"video\_id": "3u4509043",

"creation\_date": "2023-01-02"

}

video\_light\_collection.insert\_one(video\_light\_sample)

For detailed tagging:

python

Copy code

video\_detailed\_collection = db["video\_detailed"]

video\_detailed\_sample = {

"video\_id": "3u4509043",

"creation\_date": "2023-01-02",

"video\_length": "120s",

"video\_format": "MP4",

"video\_resolution": "1920x1080",

"video\_bitrate": "2000kbps"

}

video\_detailed\_collection.insert\_one(video\_detailed\_sample)

Text-like Data - MongoDB Collection

For light tagging:

python

Copy code

text\_light\_collection = db["text\_light"]

text\_light\_sample = {

"text\_id": "4u5509054",

"creation\_date": "2023-01-03"

}

text\_light\_collection.insert\_one(text\_light\_sample)

For detailed tagging:

python

Copy code

text\_detailed\_collection = db["text\_detailed"]

text\_detailed\_sample = {

"text\_id": "4u5509054",

"creation\_date": "2023-01-03",

"text\_length": "5000 characters",

"text\_language": "English",

"text\_format": "Markdown",

"tags": ["sample", "text"]

}

text\_detailed\_collection.insert\_one(text\_detailed\_sample)