Universal Semantic Search Project

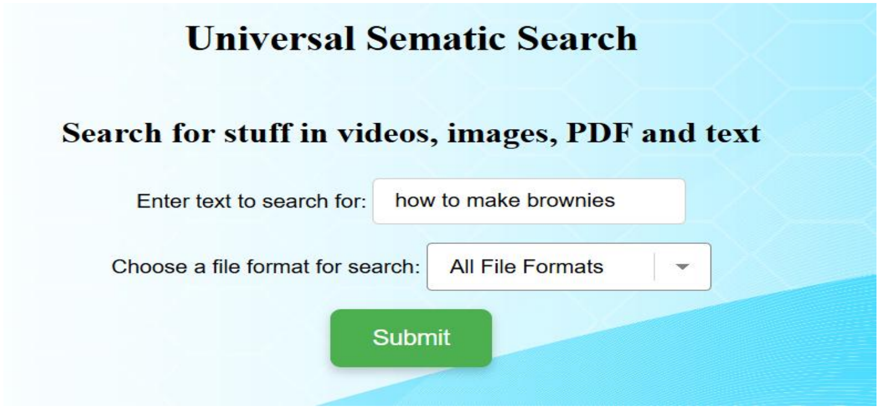
Tara Rajagopalan

Objective

- Input Query is typed into a task bar.
- Top 5 related documents will appear in results based on cosine similarity distance of vector embeddings.

How Site Works

Enter Query into Search Bar & Apply Filter, Click "Submit"



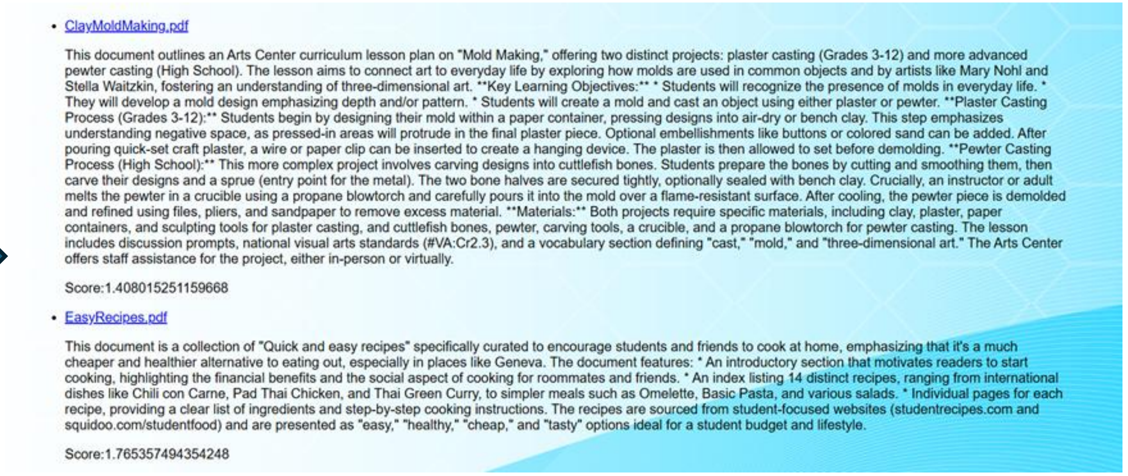
Universal Sematic Search

Search for stuff in videos, images, PDF and text

Enter text to search for: how to make brownies

Choose a file format for search: All File Formats

Most Similar Documents Show Up: Document Link, Text Summary, Cosine Distance Presented



- ClayMoldMaking.pdf

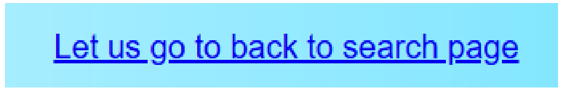
  This document outlines an Arts Center curriculum lesson plan on "Mold Making," offering two distinct projects: plaster casting (Grades 3-12) and more advanced pewter casting (High School). The lesson aims to connect art to everyday life by exploring how molds are used in common objects and by artists like Mary Nohl and Stella Waitzkin, fostering an understanding of three-dimensional art. **Key Learning Objectives:** * Students will recognize the presence of molds in everyday life. * They will develop a mold design emphasizing depth and/or pattern. * Students will create a mold and cast an object using either plaster or pewter. **Plaster Casting Process (Grades 3-12):** Students begin by designing their mold within a paper container, pressing designs into air-dry or bench clay. This step emphasizes understanding negative space, as pressed-in areas will protrude in the final plaster piece. Optional embellishments like buttons or colored sand can be added. After pouring quick-set craft plaster, a wire or paper clip can be inserted to create a hanging device. The plaster is then allowed to set before demolding. **Pewter Casting Process (High School):** This more complex project involves carving designs into cuttlefish bones. Students prepare the bones by cutting and smoothing them, then carve their designs and a sprue (entry point for the metal). The two bone halves are secured tightly, optionally sealed with bench clay. Crucially, an instructor or adult melts the pewter in a crucible using a propane blowtorch and carefully pours it into the mold over a flame-resistant surface. After cooling, the pewter piece is demolded and refined using files, pliers, and sandpaper to remove excess material. **Materials:** Both projects require specific materials, including clay, plaster, paper containers, and sculpting tools for plaster casting, and cuttlefish bones, pewter, carving tools, a crucible, and a propane blowtorch for pewter casting. The lesson includes discussion prompts, national visual arts standards (#VA:Cr2.3), and a vocabulary section defining "cast," "mold," and "three-dimensional art." The Arts Center offers staff assistance for the project, either in-person or virtually.

  Score:1.408015251159668

- EasyRecipes.pdf

  This document is a collection of "Quick and easy recipes" specifically curated to encourage students and friends to cook at home, emphasizing that it's a much cheaper and healthier alternative to eating out, especially in places like Geneva. The document features: * An introductory section that motivates readers to start cooking, highlighting the financial benefits and the social aspect of cooking for roommates and friends. * An index listing 14 distinct recipes, ranging from international dishes like Chili con Carne, Pad Thai Chicken, and Thai Green Curry, to simpler meals such as Omelette, Basic Pasta, and various salads. * Individual pages for each recipe, providing a clear list of ingredients and step-by-step cooking instructions. The recipes are sourced from student-focused websites (studentrecipes.com and squidoo.com/studentfood) and are presented as "easy," "healthy," "cheap," and "tasty" options ideal for a student budget and lifestyle.

  Score:1.765357494354248



Let us go to back to search page

Link at Bottom of Page redirects back to main page

Folder Structure

- + Flask
-     Server.jpynb
-     + Data Processing // your data processing files go here
          + GeneratingTextSummaries.ipnyb
          + projectdatatable_script.ipnyb
          + project_querySearch.ipnyb
-     + Static
-         Style.css
-         + Images   // your image files go here
-         + PDF // your PDF files go here
-         + Images // your image files go here
-         + videos // your video files go here
-     + templates // all the html files go here
-         Main.html
-         Results.html
-         Image.html
-         Video.html
-

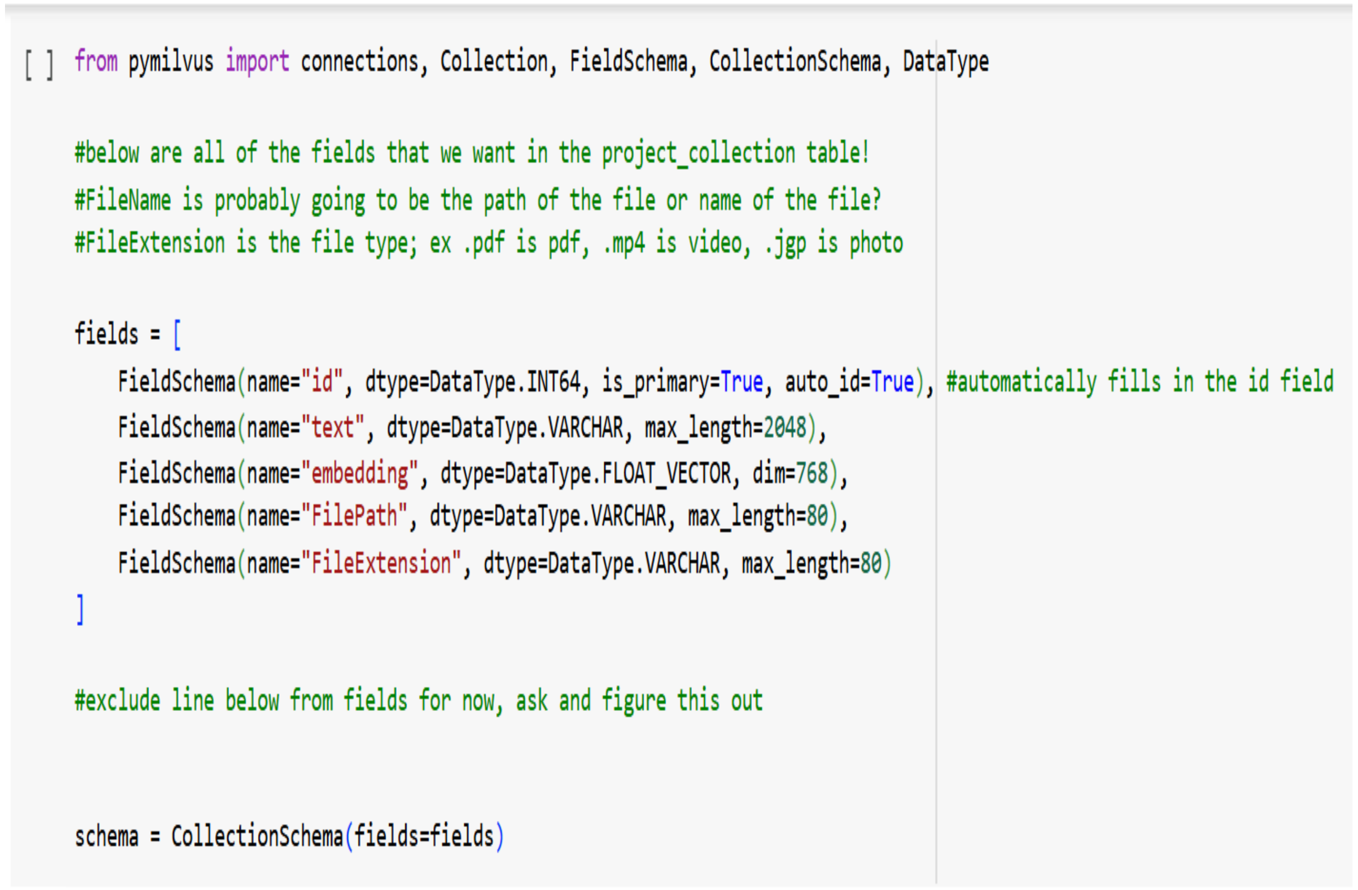
Steps

1) Build Collection Table (Milvus Database) for File Information using Milvus Client
2) Generate Text Summaries of Every File Using Gemini Client
3) Compute Vector Embeddings using pymilvus Default Embedding Function
4) Load Key information into Collection Table (Milvus Database)
5) Write a Flask App, Include Query Functions, Serve Up Site Pages.

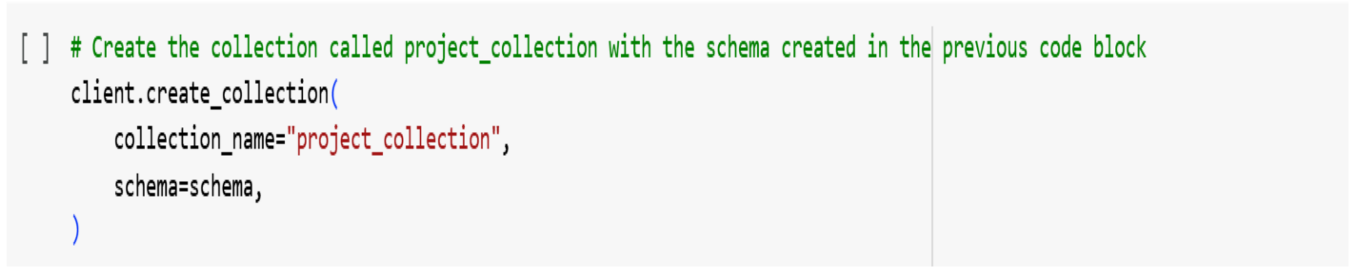
Build Collection Table

- Collection table was named "project_collection"

- **Fields included:**
- ID
- Text: Text Summary of Files
- Embedding: Vector Embeddings of Files
- FilePath: Path of File
- FileExtension: Extension of File

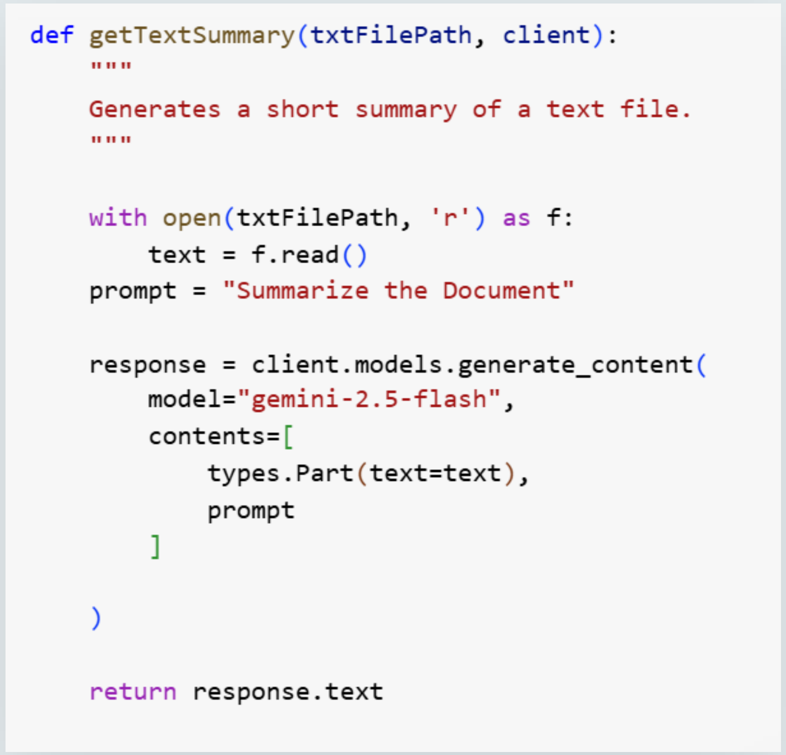


```
[ ] from pymilvus import connections, Collection, FieldSchema, CollectionSchema, DataType

    #below are all of the fields that we want in the project_collection table!
    #FileName is probably going to be the path of the file or name of the file?
    #FileExtension is the file type; ex .pdf is pdf, .mp4 is video, .jgp is photo

    fields = [
        FieldSchema(name="id", dtype=DataType.INT64, is_primary=True, auto_id=True), #automatically fills in the id field
        FieldSchema(name="text", dtype=DataType.VARCHAR, max_length=2048),
        FieldSchema(name="embedding", dtype=DataType.FLOAT_VECTOR, dim=768),
        FieldSchema(name="FilePath", dtype=DataType.VARCHAR, max_length=80),
        FieldSchema(name="FileExtension", dtype=DataType.VARCHAR, max_length=80)
    ]

    #exclude line below from fields for now, ask and figure this out


    schema = CollectionSchema(fields=fields)
```

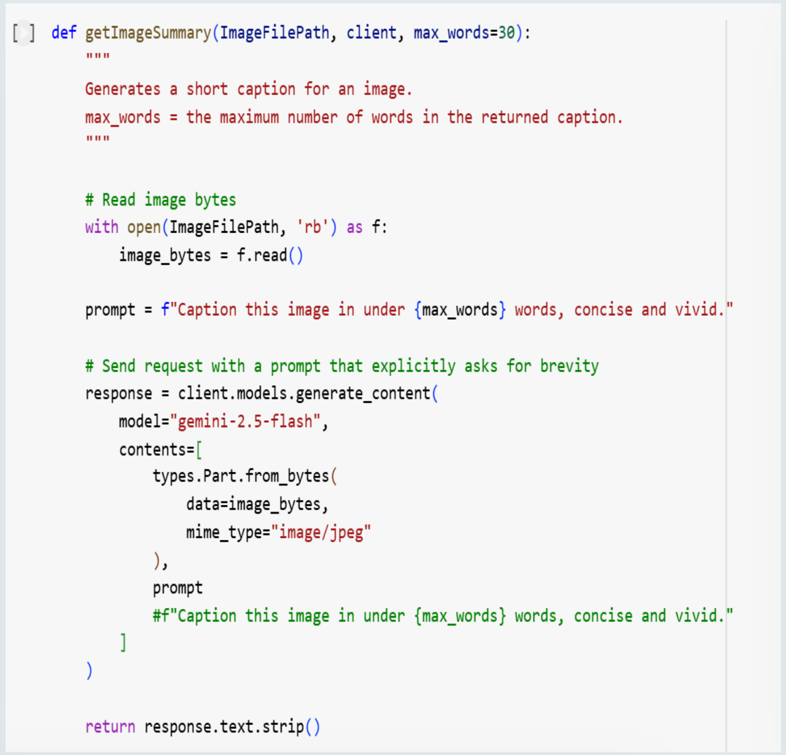


```
[ ] # Create the collection called project_collection with the schema created in the previous code block
    client.create_collection(
        collection_name="project_collection",
        schema=schema,
    )
```

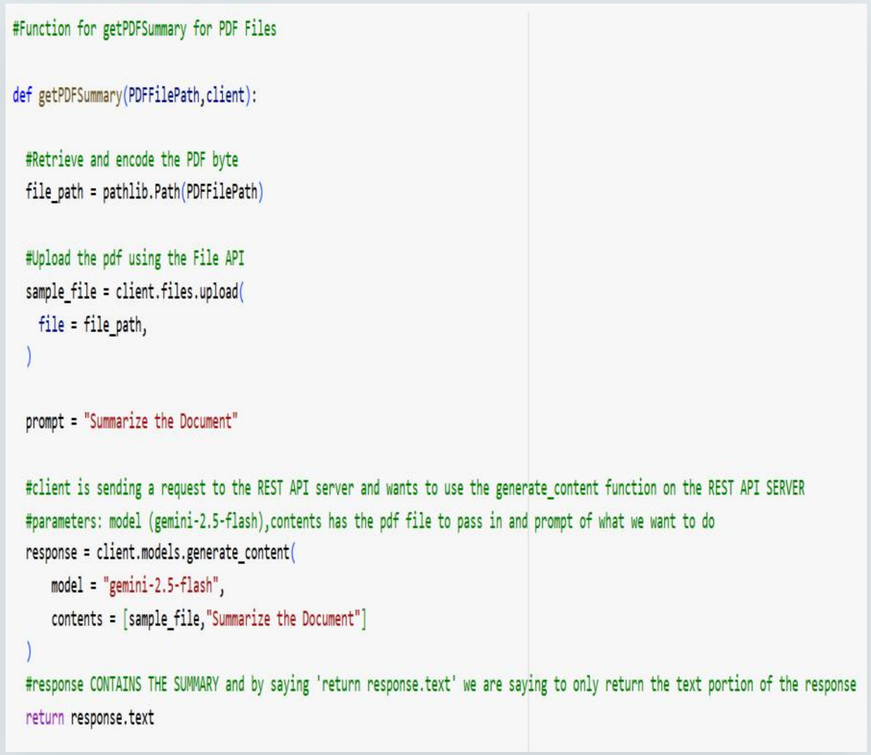
File Summarization Helper Functions



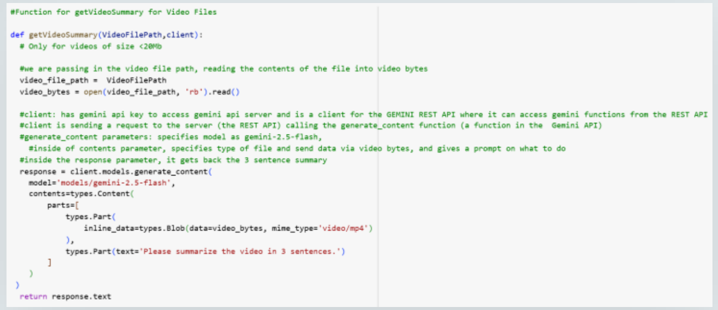
```python
def getTextSummary(txtFilePath, client):
    """
    Generates a short summary of a text file.
    """

    with open(txtFilePath, 'r') as f:
        text = f.read()
    prompt = "Summarize the Document"

    response = client.models.generate_content(
        model="gemini-2.5-flash",
        contents=[
            types.Part(text=text),
            prompt
        ]

    )

    return response.text
```



```python
[ ] def getImageSummary(ImageFilePath, client, max_words=30):
        """
        Generates a short caption for an image.
        max_words = the maximum number of words in the returned caption.
        """

        # Read image bytes
        with open(ImageFilePath, 'rb') as f:
            image_bytes = f.read()

        prompt = f"Caption this image in under {max_words} words, concise and vivid."

        # Send request with a prompt that explicitly asks for brevity
        response = client.models.generate_content(
            model="gemini-2.5-flash",
            contents=[
                types.Part.from_bytes(
                    data=image_bytes,
                    mime_type="image/jpeg"
                ),
                prompt
                #f"Caption this image in under {max_words} words, concise and vivid."
            ]
        )

        return response.text.strip()
```



```python
#Function for getPDFSummary for PDF Files

def getPDFSummary(PDFFilePath,client):

  #Retrieve and encode the PDF byte
  file_path = pathlib.Path(PDFFilePath)

  #Upload the pdf using the File API
  sample_file = client.files.upload(
    file = file_path,
  )

  prompt = "Summarize the Document"

  #client is sending a request to the REST API server and wants to use the generate_content function on the REST API SERVER
  #parameters: model (gemini-2.5-flash),contents has the pdf file to pass in and prompt of what we want to do
  response = client.models.generate_content(
      model = "gemini-2.5-flash",
      contents = [sample_file,"Summarize the Document"]
  )
  #response CONTAINS THE SUMMARY and by saying 'return response.text' we are saying to only return the text portion of the response
  return response.text
```

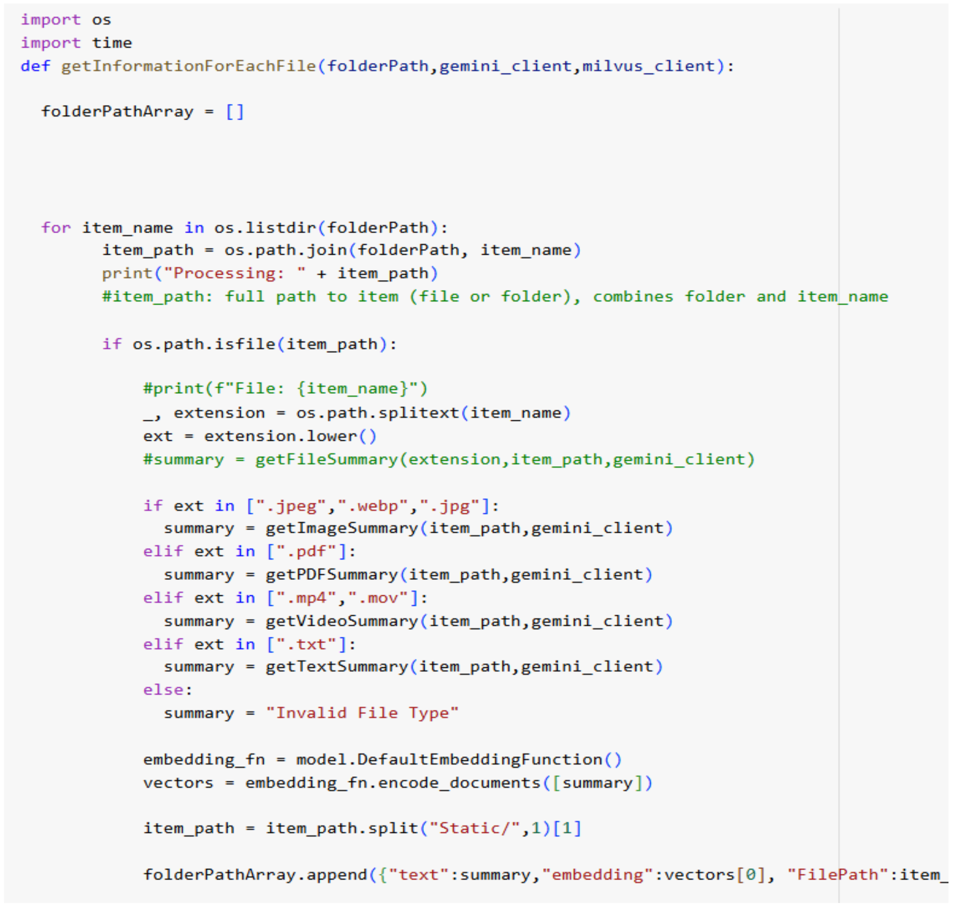


```python
#Function for getVideoSummary for Video Files

def getVideoSummary(VideoFilePath,client):
  # Only for videos of size <20Mb

  #we are passing in the video file path, reading the contents of the file into video bytes
  video_file_path =  VideoFilePath
  video_bytes = open(video_file_path, 'rb').read()

  #client: has gemini api key to access gemini api server and is a client for the GEMINI REST API where it can access gemini functions from the REST API
  #client is sending a request to the server (the REST API) calling the generate_content function (a function in the  Gemini API)
  #generate_content parameters: specifies model as gemini-2.5-flash,
    #inside of contents parameter, specifies type of file and send data via video bytes, and gives a prompt on what to do
  #inside the response parameter, it gets back the 3 sentence summary
  response = client.models.generate_content(
    model='models/gemini-2.5-flash',
    contents=types.Content(
        parts=[
            types.Part(
                inline_data=types.Blob(data=video_bytes, mime_type='video/mp4')
            ),
            types.Part(text='Please summarize the video in 3 sentences.')
        ]
    )
 )
  return response.text
```

Collecting Data for Each File

- **Shown here is the getInformationForEachFile() function:**

- A for loop is used to iterate through the Folder Path that was passed in,
- The File Extension of each file in the folder path is collected.
- Appropriate helper functions summarize the files based on file type using Gemini Client
- Embeddings are generated for each file.
- Item path information is collected.
- Each file's information is stored in a dictionaries.
- As a result, an array with dictionaries pertaining to each file is returned!

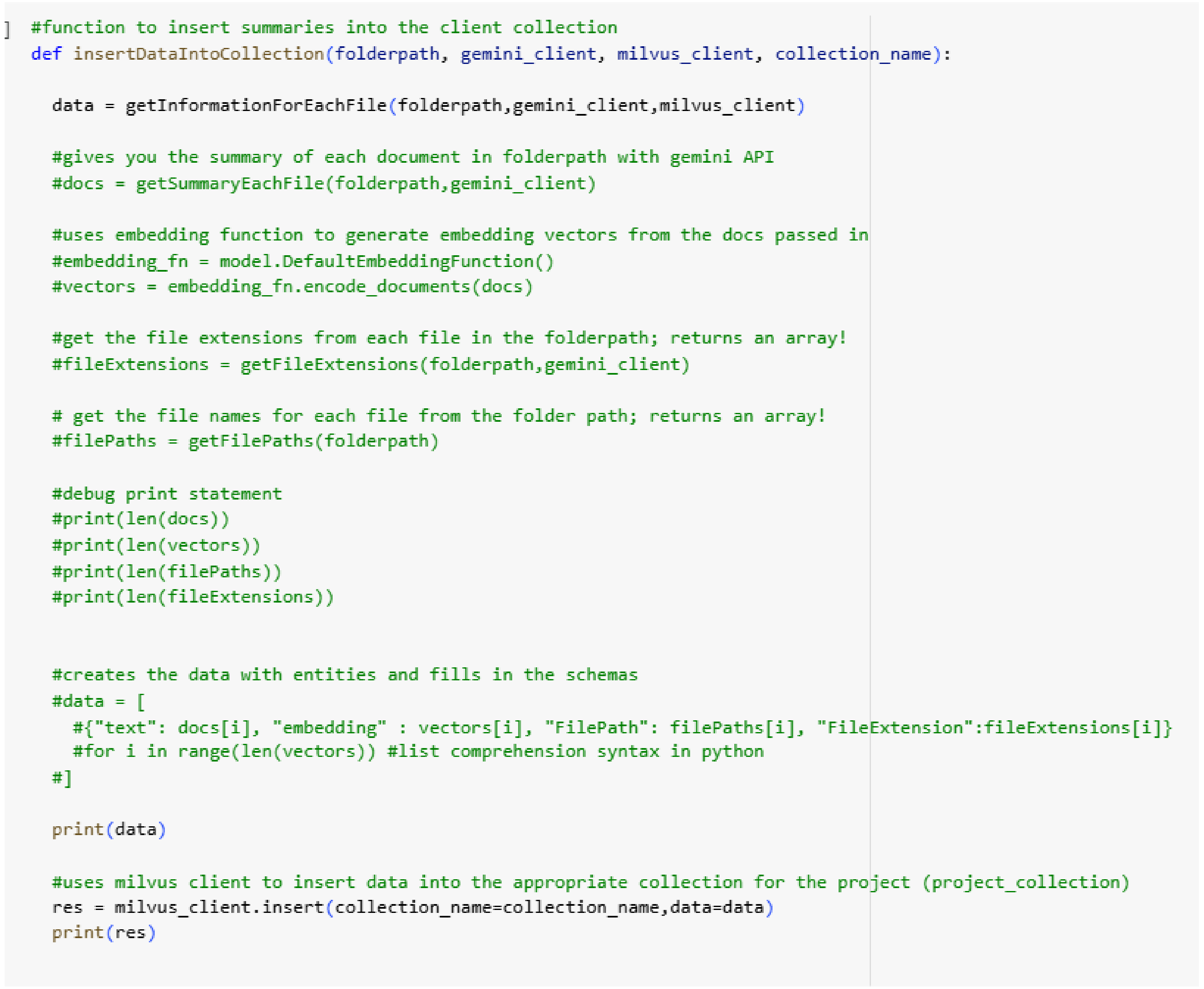


```python
import os
import time
def getInformationForEachFile(folderPath,gemini_client,milvus_client):

  folderPathArray = []



  for item_name in os.listdir(folderPath):
        item_path = os.path.join(folderPath, item_name)
        print("Processing: " + item_path)
        #item_path: full path to item (file or folder), combines folder and item_name

        if os.path.isfile(item_path):

            #print(f"File: {item_name}")
            _, extension = os.path.splitext(item_name)
            ext = extension.lower()
            #summary = getFileSummary(extension,item_path,gemini_client)

            if ext in [".jpeg",".webp",".jpg"]:
              summary = getImageSummary(item_path,gemini_client)
            elif ext in [".pdf"]:
              summary = getPDFSummary(item_path,gemini_client)
            elif ext in [".mp4",".mov"]:
              summary = getVideoSummary(item_path,gemini_client)
            elif ext in [".txt"]:
              summary = getTextSummary(item_path,gemini_client)
            else:
              summary = "Invalid File Type"

            embedding_fn = model.DefaultEmbeddingFunction()
            vectors = embedding_fn.encode_documents([summary])

            item_path = item_path.split("Static/",1)[1]

            folderPathArray.append({"text":summary,"embedding":vectors[0], "FilePath":item_
```

Load Data Into Collection Table

- **Shown here is the insertDataIntoCollection Function:**

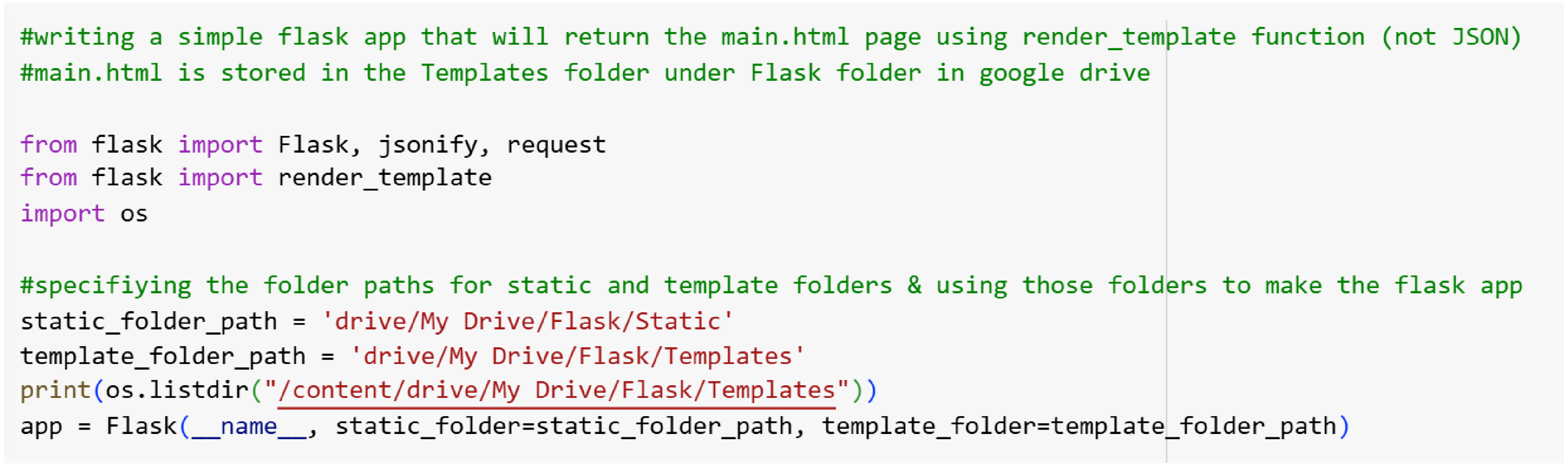
- The insertDataIntoCollection function calls the getInformationForEachFileFunction to get appropriate information for each file. Information for each file is stored in an array called “data”
-
- Milvus Client is used to insert data into project_collection table



```python
] #function to insert summaries into the client collection
  def insertDataIntoCollection(folderpath, gemini_client, milvus_client, collection_name):

    data = getInformationForEachFile(folderpath,gemini_client,milvus_client)

    #gives you the summary of each document in folderpath with gemini API
    #docs = getSummaryEachFile(folderpath,gemini_client)

    #uses embedding function to generate embedding vectors from the docs passed in
    #embedding_fn = model.DefaultEmbeddingFunction()
    #vectors = embedding_fn.encode_documents(docs)

    #get the file extensions from each file in the folderpath; returns an array!
    #fileExtensions = getFileExtensions(folderpath,gemini_client)

    # get the file names for each file from the folder path; returns an array!
    #filePaths = getFilePaths(folderpath)

    #debug print statement
    #print(len(docs))
    #print(len(vectors))
    #print(len(filePaths))
    #print(len(fileExtensions))


    #creates the data with entities and fills in the schemas
    #data = [
      #{"text": docs[i], "embedding" : vectors[i], "FilePath": filePaths[i], "FileExtension":fileExtensions[i]}
      #for i in range(len(vectors)) #list comprehension syntax in python
    #]

    print(data)

    #uses milvus client to insert data into the appropriate collection for the project (project_collection)
    res = milvus_client.insert(collection_name=collection_name,data=data)
    print(res)
```

Flask App Creation

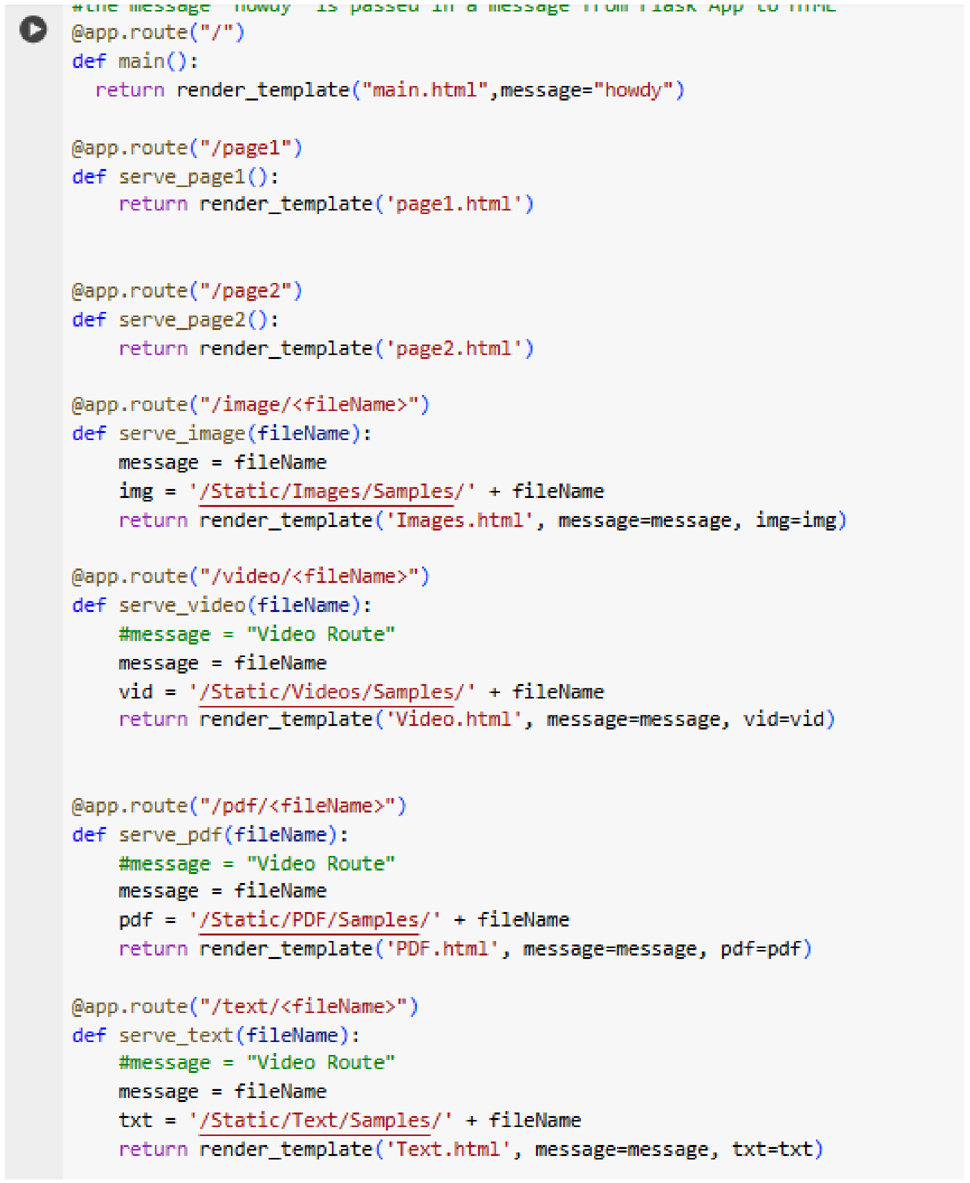
Used the Static and Template folders to create a flask app



```python
#writing a simple flask app that will return the main.html page using render_template function (not JSON)
#main.html is stored in the Templates folder under Flask folder in google drive

from flask import Flask, jsonify, request
from flask import render_template
import os

#specifiying the folder paths for static and template folders & using those folders to make the flask app
static_folder_path = 'drive/My Drive/Flask/Static'
template_folder_path = 'drive/My Drive/Flask/Templates'
print(os.listdir("/content/drive/My Drive/Flask/Templates"))
app = Flask(__name__, static_folder=static_folder_path, template_folder=template_folder_path)
```

Serve Up Site Pages and Query

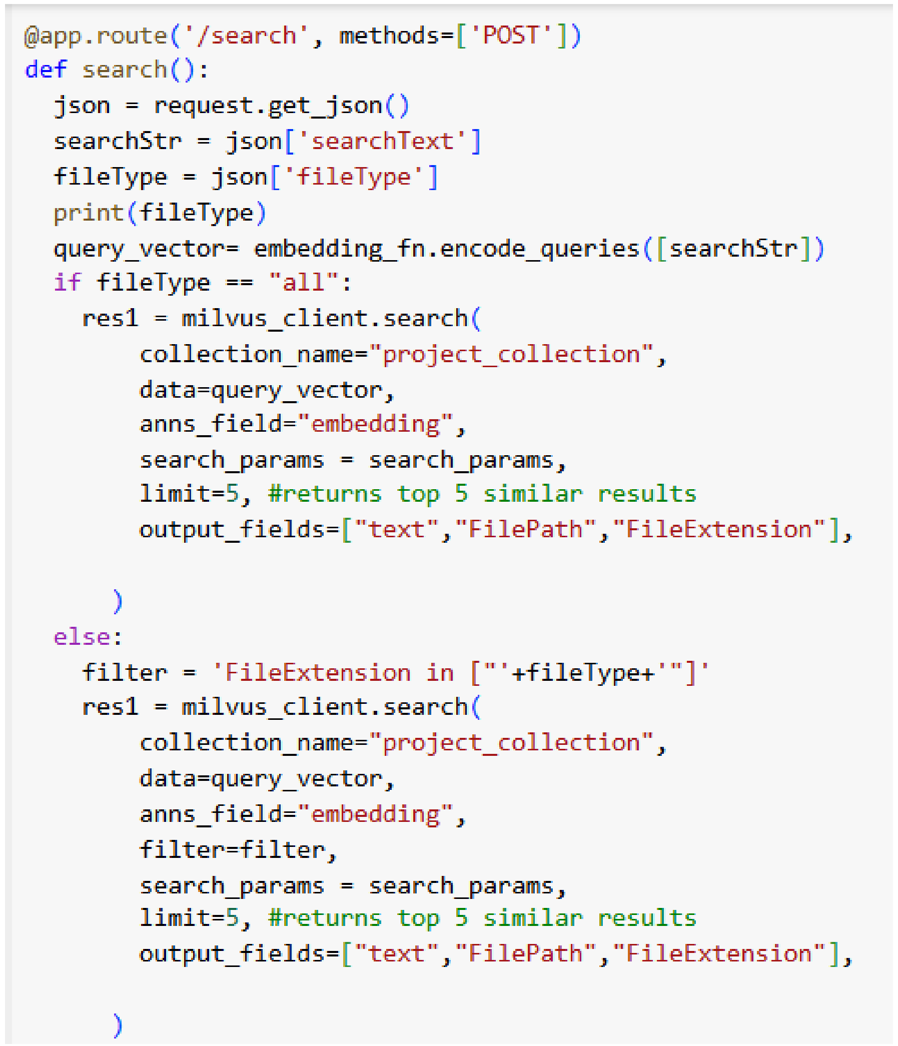
- Serves up the pages for the different files by rendering their html templates:
  - Main Page
  - Image Files
  - Video Files
  - PDF Files
  - Text Files



```
#the message "howdy" is passed in a message from Flask App to HTML
@app.route("/")
def main():
  return render_template("main.html",message="howdy")

@app.route("/page1")
def serve_page1():
    return render_template('page1.html')


@app.route("/page2")
def serve_page2():
    return render_template('page2.html')

@app.route("/image/<fileName>")
def serve_image(fileName):
    message = fileName
    img = '/Static/Images/Samples/' + fileName
    return render_template('Images.html', message=message, img=img)

@app.route("/video/<fileName>")
def serve_video(fileName):
    #message = "Video Route"
    message = fileName
    vid = '/Static/Videos/Samples/' + fileName
    return render_template('Video.html', message=message, vid=vid)


@app.route("/pdf/<fileName>")
def serve_pdf(fileName):
    #message = "Video Route"
    message = fileName
    pdf = '/Static/PDF/Samples/' + fileName
    return render_template('PDF.html', message=message, pdf=pdf)

@app.route("/text/<fileName>")
def serve_text(fileName):
    #message = "Video Route"
    message = fileName
    txt = '/Static/Text/Samples/' + fileName
    return render_template('Text.html', message=message, txt=txt)
```

Serve Up Site Pages and Query

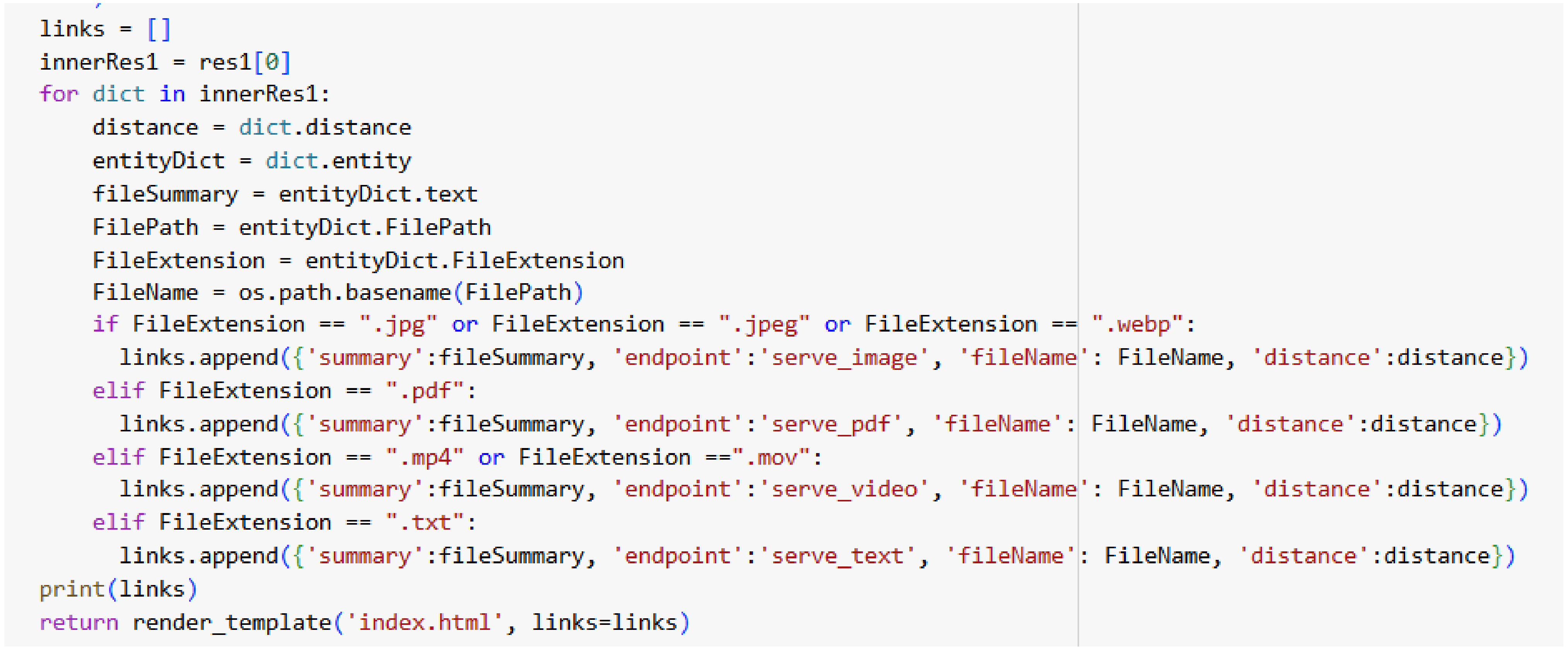
- This search function is responsible for querying and rendering the template of the resulting list.
- The querying according to the specified filter is shown here.
- The milvus client was used to query the project_collection data base.



```
@app.route('/search', methods=['POST'])
def search():
  json = request.get_json()
  searchStr = json['searchText']
  fileType = json['fileType']
  print(fileType)
  query_vector= embedding_fn.encode_queries([searchStr])
  if fileType == "all":
    res1 = milvus_client.search(
        collection_name="project_collection",
        data=query_vector,
        anns_field="embedding",
        search_params = search_params,
        limit=5, #returns top 5 similar results
        output_fields=["text","FilePath","FileExtension"],

      )
  else:
    filter = 'FileExtension in ["'+fileType+'"]'
    res1 = milvus_client.search(
        collection_name="project_collection",
        data=query_vector,
        anns_field="embedding",
        filter=filter,
        search_params = search_params,
        limit=5, #returns top 5 similar results
        output_fields=["text","FilePath","FileExtension"],

      )
```

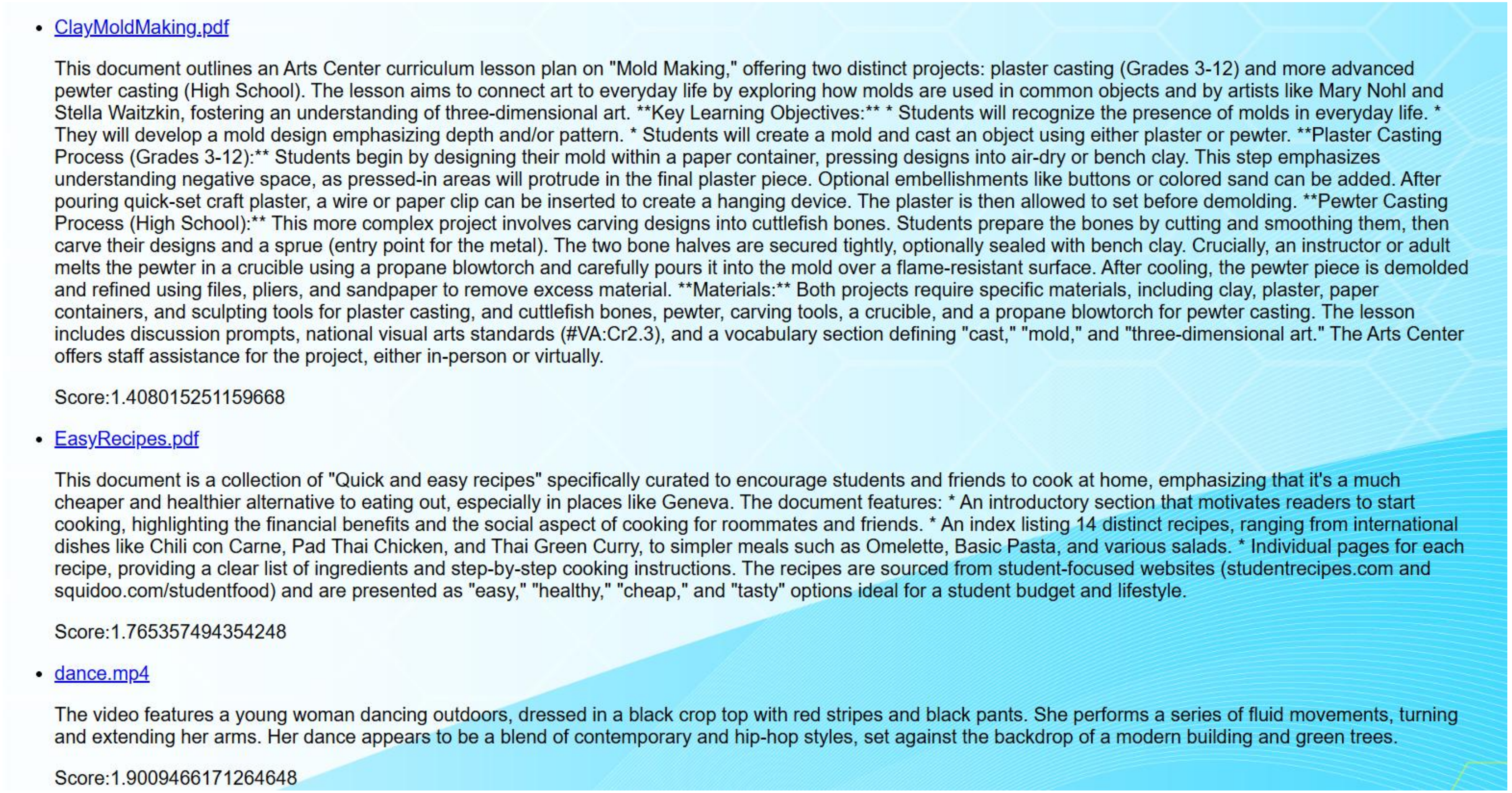
Links of Documents Displayed

- Iterates through the resulting array of dictionaries representing each file after the query
- Calls the appropriate serve function based on the File Extension
- Renders the index template to display the links of the resulting documents after querying.



```
links = []
innerRes1 = res1[0]
for dict in innerRes1:
    distance = dict.distance
    entityDict = dict.entity
    fileSummary = entityDict.text
    FilePath = entityDict.FilePath
    FileExtension = entityDict.FileExtension
    FileName = os.path.basename(FilePath)
    if FileExtension == ".jpg" or FileExtension == ".jpeg" or FileExtension == ".webp":
      links.append({'summary':fileSummary, 'endpoint':'serve_image', 'fileName': FileName, 'distance':distance})
    elif FileExtension == ".pdf":
      links.append({'summary':fileSummary, 'endpoint':'serve_pdf', 'fileName': FileName, 'distance':distance})
    elif FileExtension == ".mp4" or FileExtension ==".mov":
      links.append({'summary':fileSummary, 'endpoint':'serve_video', 'fileName': FileName, 'distance':distance})
    elif FileExtension == ".txt":
      links.append({'summary':fileSummary, 'endpoint':'serve_text', 'fileName': FileName, 'distance':distance})
print(links)
return render_template('index.html', links=links)
```

Results



- ClayMoldMaking.pdf

  This document outlines an Arts Center curriculum lesson plan on "Mold Making," offering two distinct projects: plaster casting (Grades 3-12) and more advanced pewter casting (High School). The lesson aims to connect art to everyday life by exploring how molds are used in common objects and by artists like Mary Nohl and Stella Waitzkin, fostering an understanding of three-dimensional art. **Key Learning Objectives:** * Students will recognize the presence of molds in everyday life. * They will develop a mold design emphasizing depth and/or pattern. * Students will create a mold and cast an object using either plaster or pewter. **Plaster Casting Process (Grades 3-12):** Students begin by designing their mold within a paper container, pressing designs into air-dry or bench clay. This step emphasizes understanding negative space, as pressed-in areas will protrude in the final plaster piece. Optional embellishments like buttons or colored sand can be added. After pouring quick-set craft plaster, a wire or paper clip can be inserted to create a hanging device. The plaster is then allowed to set before demolding. **Pewter Casting Process (High School):** This more complex project involves carving designs into cuttlefish bones. Students prepare the bones by cutting and smoothing them, then carve their designs and a sprue (entry point for the metal). The two bone halves are secured tightly, optionally sealed with bench clay. Crucially, an instructor or adult melts the pewter in a crucible using a propane blowtorch and carefully pours it into the mold over a flame-resistant surface. After cooling, the pewter piece is demolded and refined using files, pliers, and sandpaper to remove excess material. **Materials:** Both projects require specific materials, including clay, plaster, paper containers, and sculpting tools for plaster casting, and cuttlefish bones, pewter, carving tools, a crucible, and a propane blowtorch for pewter casting. The lesson includes discussion prompts, national visual arts standards (#VA:Cr2.3), and a vocabulary section defining "cast," "mold," and "three-dimensional art." The Arts Center offers staff assistance for the project, either in-person or virtually.

  Score:1.408015251159668

- EasyRecipes.pdf

  This document is a collection of "Quick and easy recipes" specifically curated to encourage students and friends to cook at home, emphasizing that it's a much cheaper and healthier alternative to eating out, especially in places like Geneva. The document features: * An introductory section that motivates readers to start cooking, highlighting the financial benefits and the social aspect of cooking for roommates and friends. * An index listing 14 distinct recipes, ranging from international dishes like Chili con Carne, Pad Thai Chicken, and Thai Green Curry, to simpler meals such as Omelette, Basic Pasta, and various salads. * Individual pages for each recipe, providing a clear list of ingredients and step-by-step cooking instructions. The recipes are sourced from student-focused websites (studentrecipes.com and squidoo.com/studentfood) and are presented as "easy," "healthy," "cheap," and "tasty" options ideal for a student budget and lifestyle.

  Score:1.765357494354248

- dance.mp4

  The video features a young woman dancing outdoors, dressed in a black crop top with red stripes and black pants. She performs a series of fluid movements, turning and extending her arms. Her dance appears to be a blend of contemporary and hip-hop styles, set against the backdrop of a modern building and green trees.

  Score:1.9009466171264648