Document Search Engine using Text Analysis hosted over the cloud

MSc Research Project
Cloud Computing

Vishwas Mudalahippe Shankarappa
Student ID: 21205825

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vishwas Mudalahippe Shankarappa
Student ID:	21205825
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Rejwanul Haque
Submission Due Date:	14/08/2023
Project Title:	Document Search Engine using Text Analysis hosted over the cloud
Word Count:	1243
Page Count:	8

Signature:	
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Document Search Engine using Text Analysis hosted over the cloud

Vishwas Mudalahippe Shankarappa
21205825

1 Introduction

This document outlines the configuration management for deploying and managing the Python Flask-based Web Application on an EC2 instance. The file serves as a blueprint for deploying, maintaining, and scaling the Python Flask-based web application on an EC2 instance. It encompasses various aspects, including server settings, database connections, file system configurations, background processing optimization, and more. By adhering to this configuration management, administrators can ensure consistent application behavior, seamless updates, efficient resource allocation, and reliable performance.

2 Methodology

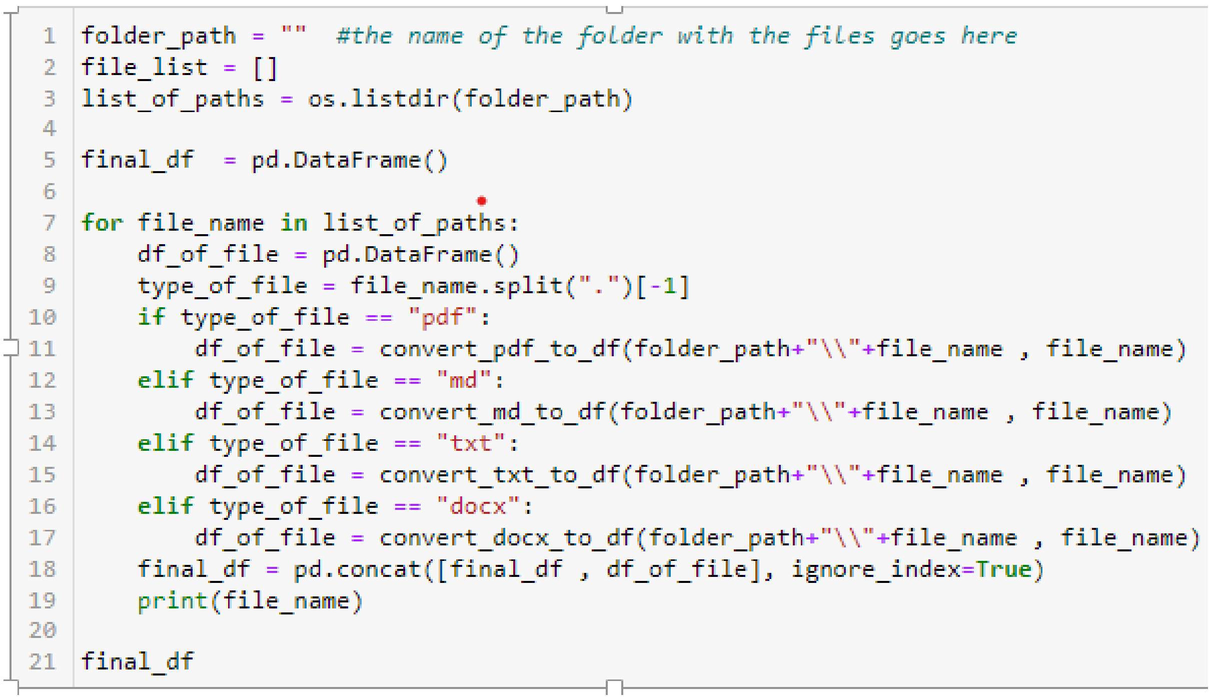
2.1 Data Acquisition

An integral part of the research is being represented by the code snippet presented below Figure 1. At this point, we are concentrating on extracting and converting papers so that we may incorporate them into our document search engine. The primary objective of this stage is to process and convert documents from various forms into a structured format appropriate for indexing and future search operations.

At this stage, the variable folder_path will be set to indicate the location of the archive containing the many papers to be studied. The iterative procedure's third stage entails searching for a file using its name within the folder's list of contents. For each file, a new DataFrame is built and given the notation df_of_file. To store the content of the structured document, this DataFrame is used.

The process of identifying the file format begins with the detection of the file's extension. This allows the system to correctly identify the document type, which is required for triggering the appropriate conversion function. There are four distinct document types that are differentiated by the sequential conditional checks: PDF, Markdown (MD), plain text (TXT), and DOCX. Invoking the proper conversion functions enables processing of each format and the effective completion of the transformation into a structure that is compatible with DataFrame. DataFrame concatenation facilitates the systematic integration of transformed DataFrames, leading to the generation of a comprehensive final_df DataFrame containing the aggregated content of all treated documents.

Notably, print statements are included in the execution of the code to provide insight into the running operation. The console updates with a report for each processed file, allowing for easy monitoring and verification of results. The culmination of our document



```python
folder_path = ""  #the name of the folder with the files goes here
file_list = []
list_of_paths = os.listdir(folder_path)

final_df  = pd.DataFrame()

for file_name in list_of_paths:
    df_of_file = pd.DataFrame()
    type_of_file = file_name.split(".")[-1]
    if type_of_file == "pdf":
        df_of_file = convert_pdf_to_df(folder_path+"\\"+file_name , file_name)
    elif type_of_file == "md":
        df_of_file = convert_md_to_df(folder_path+"\\"+file_name , file_name)
    elif type_of_file == "txt":
        df_of_file = convert_txt_to_df(folder_path+"\\"+file_name , file_name)
    elif type_of_file == "docx":
        df_of_file = convert_docx_to_df(folder_path+"\\"+file_name , file_name)
    final_df = pd.concat([final_df , df_of_file], ignore_index=True)
    print(file_name)

final_df
```

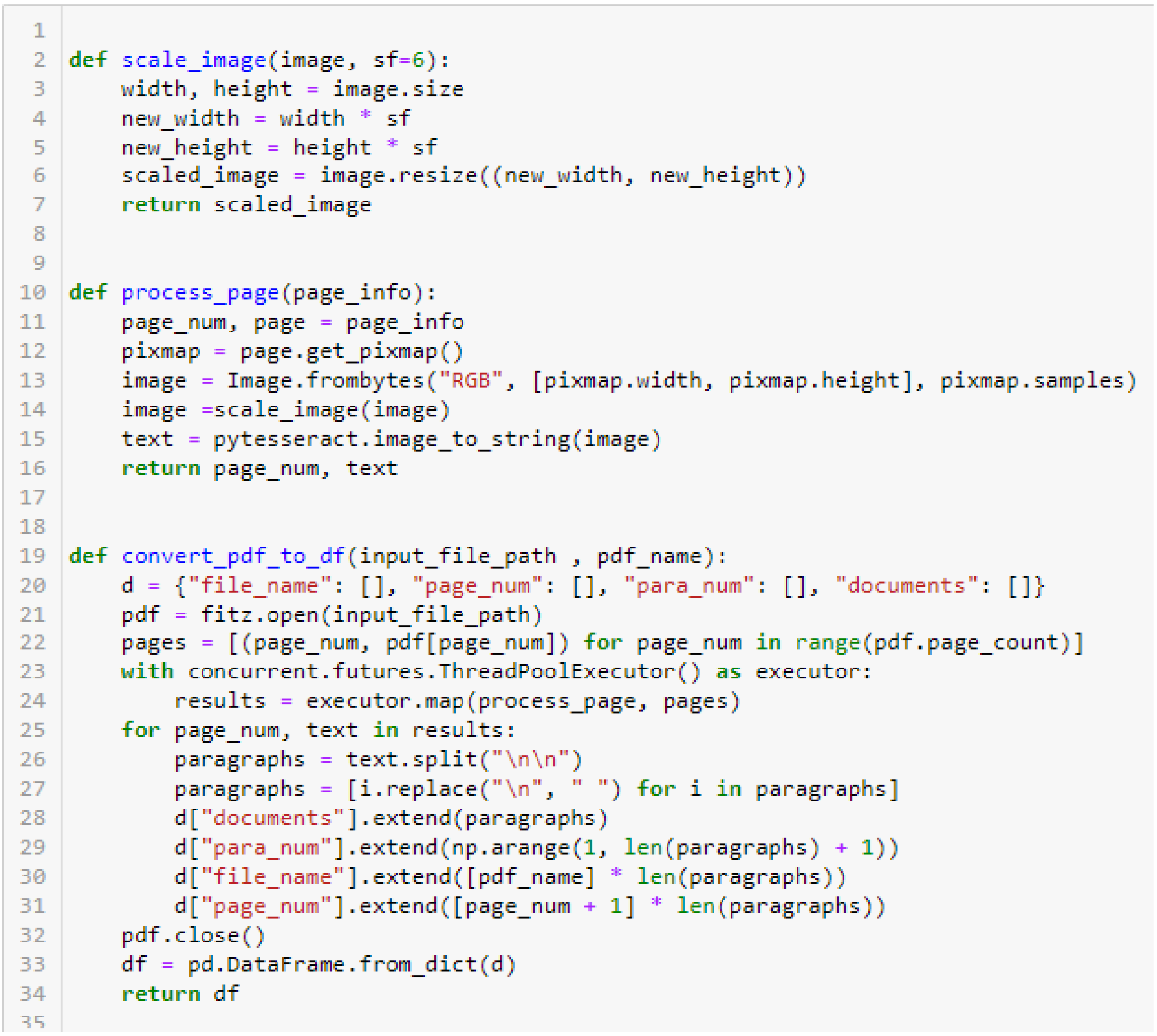
Figure 1: Data Acquisition

extraction and transformation process is the final_df DataFrame. This step is necessary preparation for the subsequent indexing and searching processes.

2.2 PDF Extraction

The following Figure 2 code sample exemplifies a critical component of the research platform that is developed. The fundamental goal of this method is to convert PDFs into structured data frames that may be used in the indexing and retrieval of documents later on. The three main functions contained in the wrapped code each make a distinct contribution to the overall conversion procedure.

- Scaling and Image Processing (scale_image): The'scale_image' function is responsible for resizing images to enhance readability and facilitate precise text extraction. The function takes an image and a scaling factor as input, calculates the new dimensions, and returns the image scaled to those dimensions.
- Text Extraction from PDF Pages (process_page): The process_page method operates on a tuple containing the page number and the page object corresponding to it, both of which are extracted from the PDF file. After the pixmap representation of the page has been extracted, it is converted into an RGB image. After that, the scale_image function is used to alter the size of the image. After that, the image will be processed by Tesseract, a very effective Optical Character Recognition (OCR) system, in order to extract textual information from it. The function returns the page number and the extracted text.



```python

def scale_image(image, sf=6):
    width, height = image.size
    new_width = width * sf
    new_height = height * sf
    scaled_image = image.resize((new_width, new_height))
    return scaled_image


def process_page(page_info):
    page_num, page = page_info
    pixmap = page.get_pixmap()
    image = Image.frombytes("RGB", [pixmap.width, pixmap.height], pixmap.samples)
    image =scale_image(image)
    text = pytesseract.image_to_string(image)
    return page_num, text


def convert_pdf_to_df(input_file_path , pdf_name):
    d = {"file_name": [], "page_num": [], "para_num": [], "documents": []}
    pdf = fitz.open(input_file_path)
    pages = [(page_num, pdf[page_num]) for page_num in range(pdf.page_count)]
    with concurrent.futures.ThreadPoolExecutor() as executor:
        results = executor.map(process_page, pages)
    for page_num, text in results:
        paragraphs = text.split("\n\n")
        paragraphs = [i.replace("\n", " ") for i in paragraphs]
        d["documents"].extend(paragraphs)
        d["para_num"].extend(np.arange(1, len(paragraphs) + 1))
        d["file_name"].extend([pdf_name] * len(paragraphs))
        d["page_num"].extend([page_num + 1] * len(paragraphs))
    pdf.close()
    df = pd.DataFrame.from_dict(d)
    return df

```

Figure 2: PDF Extraction

3 Implementation

The models created for the document search engine are first tested on Google Colaboratory, and the best model is deployed to the cloud. As part of this research, five models were created and tested, and one model is deployed on an EC2 instance.

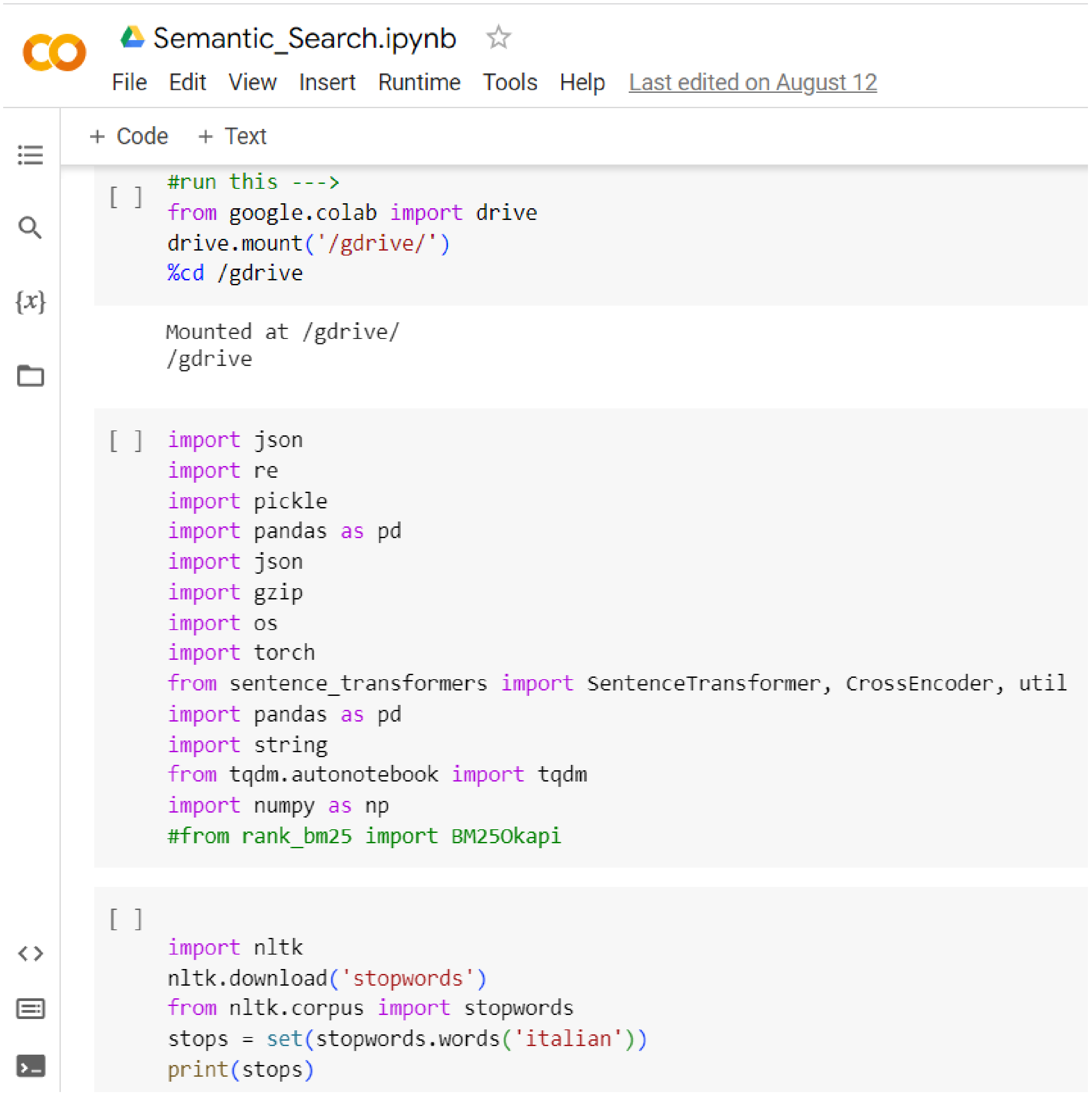
3.1 Google Colaboratory

All five models were created and tested on Google Colaboratory Figure 3, and the dataset is stored in Google Drive. These models can be run anywhere by uploading the models and dataset to Google Colaboratory.

3.2 Cloud Deployment

The TF-IDF model is deployed on an EC2 instance as a Python Flask-based Web Application. The application offers three APIs where users can upload the PDFs, extract the content, and ask queries. The APIs can be accessed through Postman. The below-mentioned steps were followed for the deployment:

- The AWS Cloud9 IDE Figure 4 is used as a workspace for the development of code and testing.
- An AWS EC2 instance is launched with ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516 AMI for deployment of the application.
- An elastic IP (18.200.13.52) is created and attached to the EC2 instance to retain the same IP even after restarting the instance and for dynamic computing.
- sudo apt update : Updates the local package repository cache on a Debian-based system.
- git clone https ://github.com/vishwasms121/document-search.git: Clones a Git repository from the specified URL.
- cd document-search/ : Changes the current directory to the ”document-search” directory.
- sudo apt install mysql-server : Installs the MySQL database server.
- sudo systemctl start mysql.service : Starts the MySQL service using systemd.
- sudo mysql : Opens the MySQL command-line client.
- mysql -u root -p : Opens the MySQL command-line client as the ”root” user with password authentication.
- CREATE DATABASE search; Creates a new database named ”search”
- USE search; Sets the current session to use the ”search” database for subsequent queries.


Semantic_Search.ipynb
File Edit View Insert Runtime Tools Help Last edited on August 12
+ Code + Text

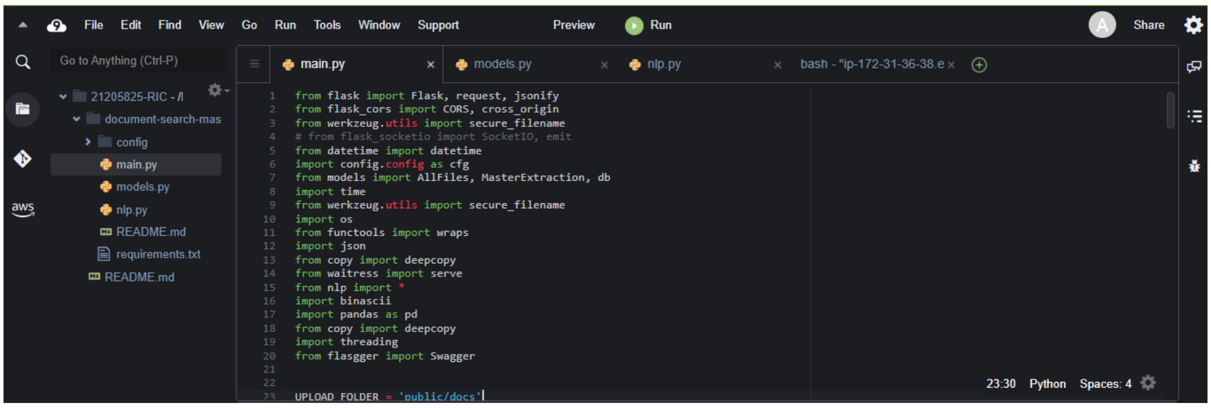
```
#run this --->
from google.colab import drive
drive.mount('/gdrive/')
%cd /gdrive
```

```
Mounted at /gdrive/
/gdrive
```

```
import json
import re
import pickle
import pandas as pd
import json
import gzip
import os
import torch
from sentence_transformers import SentenceTransformer, CrossEncoder, util
import pandas as pd
import string
from tqdm.autonotebook import tqdm
import numpy as np
#from rank_bm25 import BM25Okapi
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stops = set(stopwords.words('italian'))
print(stops)
```

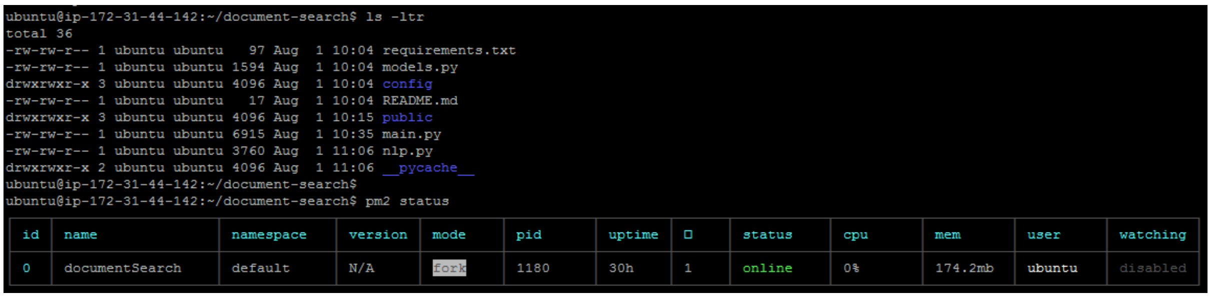
Figure 3: Sentence Encoder on Google Colaboratory



```python
from flask import Flask, request, jsonify
from flask_cors import CORS, cross_origin
from werkzeug.utils import secure_filename
# from flask_socketio import SocketIO, emit
from datetime import datetime
import config.config as cfg
from models import AllFiles, MasterExtraction, db
import time
from werkzeug.utils import secure_filename
import os
from functools import wraps
import json
from copy import deepcopy
from waitress import serve
from nlp import *
import binascii
import pandas as pd
from copy import deepcopy
import threading
from flasgger import Swagger


UPLOAD_FOLDER = 'public/docs'
```

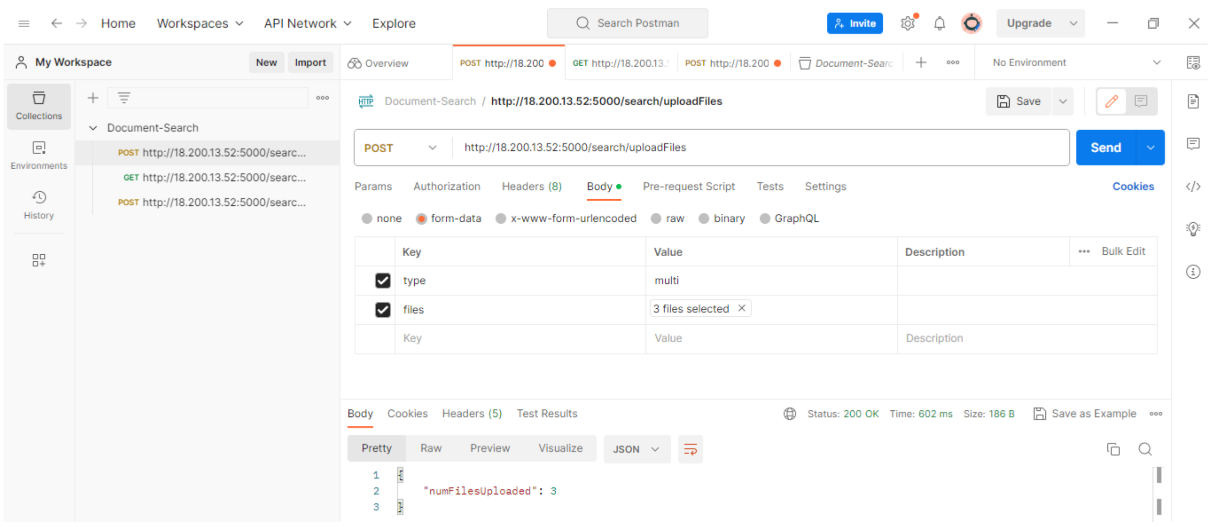
Figure 4: AWS Cloud9 environment

- CREATE TABLE ‘all_files‘ (‘id‘ int NOT NULL AUTO_INCREMENT, ‘filename‘ varchar(500) DEFAULT NULL, ‘created_date‘ datetime DEFAULT NULL, PRIMARY KEY (‘id‘)); Creates a table named "all_files" with columns for file metadata including an auto-incrementing ID, filename, and creation date.
- CREATE TABLE ‘master_extraction‘ (‘id‘ int NOT NULL AUTO_INCREMENT, ‘file_id‘ int DEFAULT NULL, ‘page_num‘ int DEFAULT NULL, ‘para_num‘ int DEFAULT NULL, ‘extracted_text‘ varchar(5000) DEFAULT NULL, ‘created_date‘ datetime DEFAULT NULL, PRIMARY KEY (‘id‘)); Creates a table named "master_extraction" to store extracted data, including an auto-incrementing ID, file ID reference, page number, paragraph number, extracted text, and creation date.
- python3 : Opens the Python 3 interpreter.
- sudo apt install python3-pip : Installs the Python 3 package manager (pip).
- pip install -r requirements.txt : Installs Python package dependencies listed in "requirements.txt".
- mkdir -p public/docs : Creates a directory named "public/docs" and any necessary parent directories.
- sudo apt install nodejs : Installs the Node.js JavaScript runtime.
- node -v: Displays the version of Node.js installed.
- sudo apt install npm : Installs the Node.js package manager (npm).
- sudo npm install pm2 -g : Installs the pm2 process manager globally using npm.
- pm2 start "python3 main.py" –name documentSearch : Starts a Python script named "main.py" using pm2 and assigns it the name "documentSearch" Figure 5
- pm2 logs : Displays the logs of running pm2 processes.
- pm2 status : Displays the status of pm2-managed processes.



```
ubuntu@ip-172-31-44-142:~/document-search$ ls -ltr
total 36
-rw-rw-r-- 1 ubuntu ubuntu   97 Aug  1 10:04 requirements.txt
-rw-rw-r-- 1 ubuntu ubuntu 1594 Aug  1 10:04 models.py
drwxrwxr-x 3 ubuntu ubuntu 4096 Aug  1 10:04 config
-rw-rw-r-- 1 ubuntu ubuntu   17 Aug  1 10:04 README.md
drwxrwxr-x 3 ubuntu ubuntu 4096 Aug  1 10:15 public
-rw-rw-r-- 1 ubuntu ubuntu 6915 Aug  1 10:35 main.py
-rw-rw-r-- 1 ubuntu ubuntu 3760 Aug  1 11:06 nlp.py
drwxrwxr-x 2 ubuntu ubuntu 4096 Aug  1 11:06 __pycache__
ubuntu@ip-172-31-44-142:~/document-search$
ubuntu@ip-172-31-44-142:~/document-search$ pm2 status
 id │ name            │ namespace   │ version │ mode    │ pid      │ uptime │ ↺    │ status    │ cpu      │ mem      │ user     │ watching
 0  │ documentSearch  │ default     │ N/A     │ fork    │ 1180     │ 30h    │ 1    │ online    │ 0%       │ 174.2mb  │ ubuntu   │ disabled
```

Figure 5: pm2 process manager

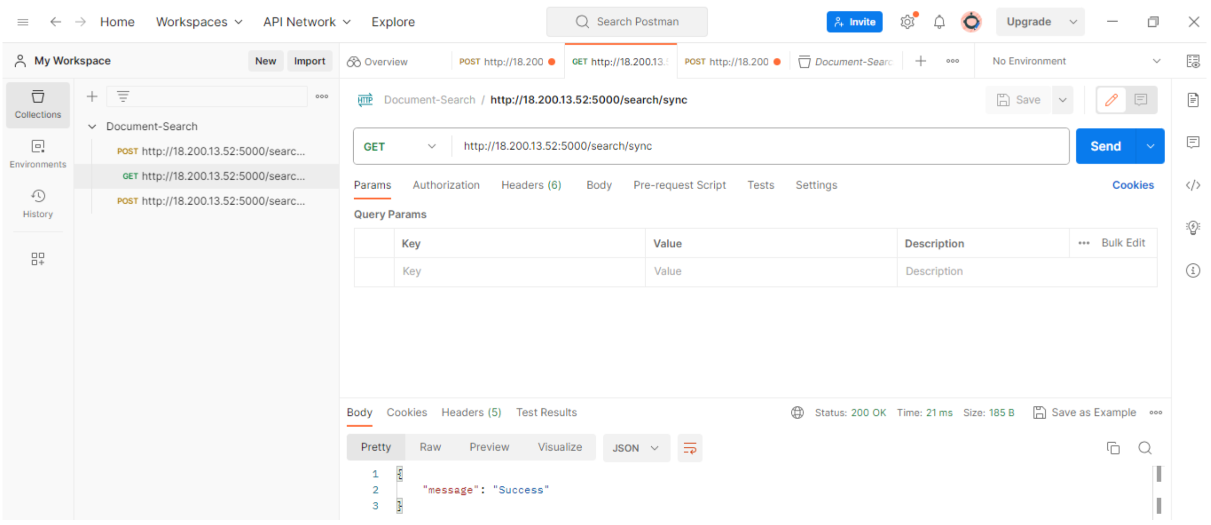


```
POST http://18.200.13.52:5000/search/uploadFiles

Body: form-data
Key     Value
type    multi
files   3 files selected

Status: 200 OK  Time: 602 ms  Size: 186 B

{
    "numFilesUploaded": 3
}
```

Figure 6: POST endpoint to upload the PDFs

4 Results

The application creates three APIs that can be accessed and tested through Postman.



Document-Search / http://18.200.13.52:5000/search/sync

GET http://18.200.13.52:5000/search/sync

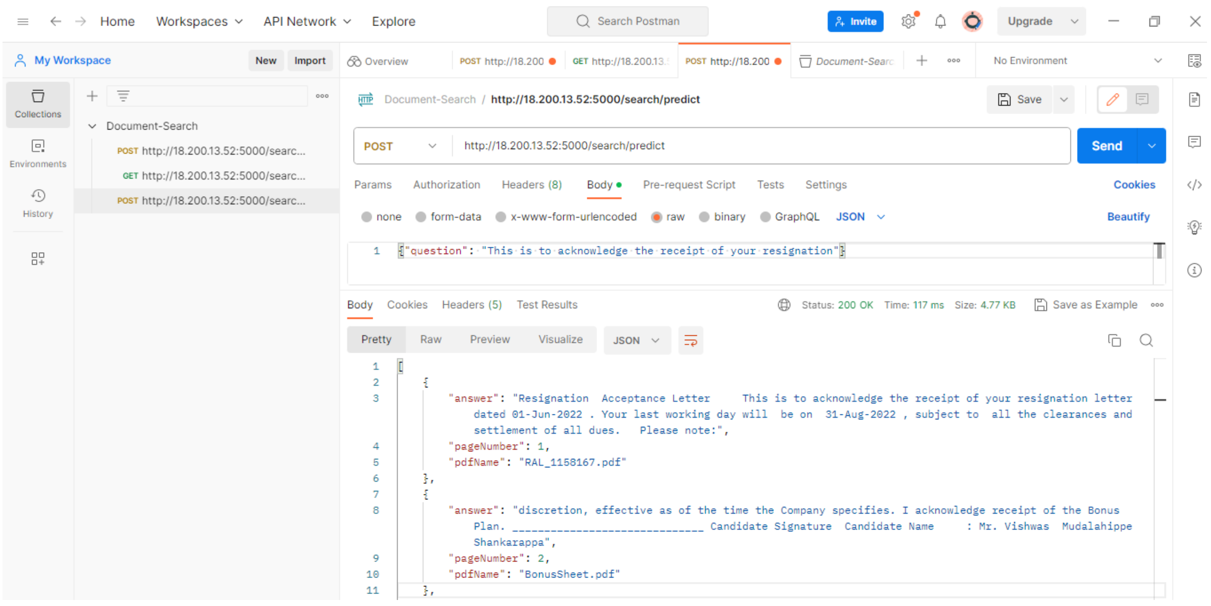
Query Params

Key	Value	Description
Key	Value	Description

Body — Status: 200 OK Time: 21 ms Size: 185 B — JSON

```
{
    "message": "Success"
}
```

Figure 7: GET endpoint to extract the PDFs



Document-Search / http://18.200.13.52:5000/search/predict

POST http://18.200.13.52:5000/search/predict

Body — raw — JSON

```
{"question": "This is to acknowledge the receipt of your resignation"}
```

Body — Status: 200 OK Time: 117 ms Size: 4.77 KB — JSON

```
[
    {
        "answer": "Resignation  Acceptance Letter     This is to acknowledge the receipt of your resignation letter
            dated 01-Jun-2022 . Your last working day will  be on  31-Aug-2022 , subject to  all the clearances and
            settlement of all dues.   Please note:",
        "pageNumber": 1,
        "pdfName": "RAL_1158167.pdf"
    },
    {
        "answer": "discretion, effective as of the time the Company specifies. I acknowledge receipt of the Bonus
            Plan. ______________________________ Candidate Signature  Candidate Name     : Mr. Vishwas  Mudalahippe
            Shankarappa",
        "pageNumber": 2,
        "pdfName": "BonusSheet.pdf"
    },
```

Figure 8: POST endpoint to ask the queries