Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2023, 10(8):146-149**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Automating PDF Report Generation Using Python and FastAPI: A Seamless Integration with Elastic Content for Real-Time Data Reporting

Pankaj Dureja

Pankaj.Dureja@gmail.com

Abstract: This research explores the automation of PDF report generation using Python, with data sourced from Elastic Content via a Python FastAPI interface. The study demonstrates how integrating Python libraries like FPDF with FastAPI allows for real-time, dynamic report creation, enhancing data-driven decision-making. The implemented solution shows potential for scalability and adaptability across various industries, particularly those needing regular, automated report generation.

Keywords: Python, PDF Report Automation, Elastic Content, FastAPI, Data Integration, FPDF, API-Driven Reporting.

1. Introduction

In an era where data-driven decisions are pivotal, generating comprehensive reports efficiently has become increasingly important. Traditional methods of report creation often involve manual efforts that are time-consuming and prone to errors. With the advent of APIs and Python's extensive library ecosystem, it's now possible to automate the entire process of report generation. This research focuses on automating the generation of PDF reports by leveraging data stored in Elastic Content, accessed through Python's FastAPI. This approach not only streamlines the process but also ensures that the reports are up-to-date and accurate, reflecting real-time data.

2. Problem Statement

Manual report generation, especially in large-scale organizations, often leads to inefficiencies, including time delays, human errors, and resource wastage. The existing methods do not scale well with increasing data volumes and complexity. Additionally, the lack of integration between data sources and reporting tools further complicates the process, making it challenging to produce timely and accurate reports. This research addresses the need for an automated solution that can generate PDF reports by seamlessly integrating with data stored in Elastic Content and accessed via FastAPI.

3. Solution Implemented

The implemented solution involves the use of Python's FPDF libraries to generate PDF reports. FastAPI serves as the interface to access and query data from Elastic Content. The solution is designed to be modular and scalable, allowing for easy customization of report formats and content. The process begins with querying the Elastic Content database via FastAPI, processing the retrieved data in Python, and then using a PDF generation library to create the report. This automated process ensures that reports are generated dynamically, based on the most recent data available in Elastic Content.

1. Data Retrieval with FastAPI: FastAPI is used to create an API that queries Elastic Content, retrieving the necessary data for the report. FastAPI's asynchronous capabilities make it well-suited for handling multiple data requests efficiently. FastAPI Router Endpoint for On-Demand Report Generation from the User Interface. Below is a sample implementation:

```python
from fastapi import APIRouter, Depends, Request
from fastapi.responses import HTMLResponse, JSONResponse
from dependencies import auth_token
from content_client import Client
from models import (
    SecurityRoleInput,
    FavouritesInput,
    SaveFavouritesInput,
    GenerateReportInput,
    SaveContentCrudInput,
    ContentCrudInput
)
from services.generate_report import save_generate_report_serv
from config import config_object

router_generate_report = APIRouter(prefix="/api/generate-report")

@router_generate_report.post("/", dependencies=[Depends(auth_token)])
async def generate_report_rout(
    request_ip: GenerateReportInput, token: str = Depends(auth_token)
):
    request_ip.user_name = token["UserName"]
    request_ip.token = token["ID"]
    if token["ID"]:
        return save_generate_report_serv(request_ip)
```

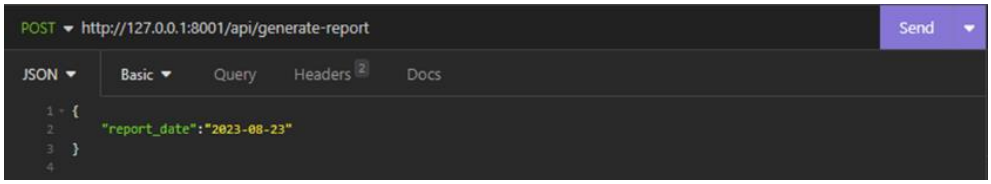
2. Processing Data in Python: Once the data is retrieved from Elastic Content, it is processed in Python. This processing might include data cleaning, filtering, aggregation, and formatting, depending on the report's requirements.

```python
from const import *
from elasticsearch import Elasticsearch
from config_helper import config_helper
from datetime import datetime, timedelta
from config import config_object

class ElasticClient:
    def __init__(self):...

    def get_connection(self):...

    def fetch_all_es_content(self, date: int, size: int):
        es = self.get_connection()

        if date > 0:
            start_date = datetime.fromtimestamp(date)
            end_date = start_date + timedelta(days=1)

            start_date_str = start_date.strftime('%Y-%m-%dT%H:%M:%S')
            end_date_str = end_date.strftime('%Y-%m-%dT%H:%M:%S')

            date_range = {
                "query": {"bool": {"must": [{"range": {"update_dt": {"gte": start_date_str, "lt": end_date_str}}}]}}}

            res = es.search(index=ES_INDEX, size=size, _source_excludes=['html', 'text', 'keywords'], body=date_range)
        else:
            sort_query = {"sort": [{"create_dt": {"order": "desc"}}]}
            res = es.search(index=ES_INDEX, size=size, _source_excludes=['html', 'text', 'keywords'], body=sort_query)

        list = []

        sources = res['hits']['hits']

        for source in sources:
            new_item = source['_source']
            new_item['id'] = source['_id']
            list.append(new_item)

        return list
```

3. PDF Report Generation with Python Libraries: The processed data is then passed to a Python library such as FPDF to generate the PDF report. These libraries provide extensive capabilities for designing and formatting reports, including adding text, tables, charts, and images. Sample Python Program.

```python
from fpdf import FPDF

    try:
        def generate_pdf_report(data, file_name):
            pdf = FPDF()
            pdf.add_page()
            pdf.set_font("Arial", size=12)

            # Title of the PDF
            pdf.set_font("Arial", 'B', 16)
            pdf.cell(200, 10, txt="Sample PDF Report", ln=True, align='C')

            # Line break
            pdf.ln(10)

            # Creating table header
            pdf.set_font("Arial", 'B', 12)
            pdf.cell(100, 10, txt="Title", border=1, align='C')
            pdf.cell(40, 10, txt="Value", border=1, align='C')
            pdf.ln()

            # Inserting data into the table
            pdf.set_font("Arial", size=12)
            for item in data:
                pdf.cell(100, 10, txt=item["title"], border=1)
                pdf.cell(40, 10, txt=str(item["value"]), border=1)
                pdf.ln()

            # Save the PDF with the given file name
            pdf.output(file_name)

        return "Success"
    except Exception as e:
        logging.error(f"Exception in running _execute_sp {e}")
        raise
```

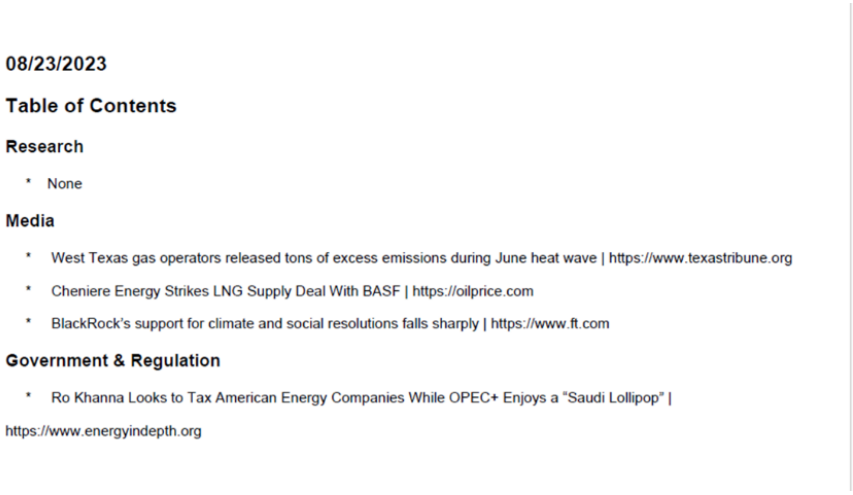
4. Automated Workflow: The entire workflow, from data retrieval to report generation, is automated, allowing reports to be generated on-demand or scheduled at regular intervals.



Report Name Generated Highlighted in Blue color.

Name	Size	Type	Last Modified	Storage Class
Research and Clips 08.08.2023.pdf	262.59 KB	Adobe Acrobat...	8/8/2023 2:51:03 PM	STANDARD
Media_08.23.2023.pdf	43.40 KB	Adobe Acrobat...	8/23/2023 2:33:49 PM	STANDARD
Gov_Rel_08.23.2023.pdf	39.39 KB	Adobe Acrobat...	8/23/2023 2:33:49 PM	STANDARD
TC_08.23.2023.pdf	28.92 KB	Adobe Acrobat...	8/23/2023 2:33:49 PM	STANDARD
Research and Clips 08.23.2023.pdf	110.85 KB	Adobe Acrobat...	8/23/2023 2:33:49 PM	STANDARD
77297153_08.24.2023.pdf	779.10 KB	Adobe Acrobat...	8/24/2023 3:23:50 PM	STANDARD
Media_08.24.2023.pdf	42.77 KB	Adobe Acrobat...	8/24/2023 3:23:51 PM	STANDARD
Gov_Rel_08.24.2023.pdf	45.27 KB	Adobe Acrobat...	8/24/2023 3:23:52 PM	STANDARD
TC_08.24.2023.pdf	28.24 KB	Adobe Acrobat...	8/24/2023 3:23:52 PM	STANDARD

Report Output Generated Using the FPDF Python Library via FastAPI Endpoint.



08/23/2023

Table of Contents

Research

* None

Media

* West Texas gas operators released tons of excess emissions during June heat wave | https://www.texastribune.org
* Cheniere Energy Strikes LNG Supply Deal With BASF | https://oilprice.com
* BlackRock's support for climate and social resolutions falls sharply | https://www.ft.com

Government & Regulation

* Ro Khanna Looks to Tax American Energy Companies While OPEC+ Enjoys a "Saudi Lollipop" | https://www.energyindepth.org

4. Potential Extended Use Cases

The approach demonstrated in this research can be extended to various other use cases, such as:

- Automating financial statements or audit reports generation.
- Generating real-time dashboards in PDF format for business intelligence.

- Creating customized reports for different stakeholders in an organization.
- Integrating with other data sources like SQL databases, REST APIs, or cloud storage to broaden the scope of data inputs.
- Enhancing the reporting process in sectors like healthcare, finance, and logistics where timely and accurate data reporting is crucial.

5. Impact

The automation of PDF report generation significantly reduces the time and effort required to produce reports. It eliminates the risk of human error, ensuring the consistency and accuracy of the reports. Additionally, by utilizing Elastic Content as the data source, organizations can ensure that their reports are always based on the latest available data, improving decision-making processes. This approach also provides a scalable solution that can grow with the organization's needs.

6. Scope

This research is focused on the automation of PDF report generation using Python and FastAPI, with Elastic Content as the primary data source. While the study primarily targets industries requiring regular report generation, the solution can be adapted and extended to various other sectors and use cases, as outlined in the potential extended use cases section. The scope also includes exploring the limitations of the current implementation and suggesting areas for future improvement.

7. Conclusion

The integration of Python with FastAPI and Elastic Content presents a powerful solution for automating the generation of PDF reports. This approach offers a robust, scalable, and efficient method for producing accurate and up-to-date reports, addressing many of the challenges associated with manual report generation. The research concludes that such automation can significantly enhance organizational efficiency, reduce errors, and provide real-time insights, making it a valuable tool for data-driven decision-making.

References

[1]. FastAPI Documentation, Available at https://fastapi.tiangolo.com/
[2]. Python PDF generation library, Available at https://pyfpdf.readthedocs.io/en/latest/
[3]. Automate the boring stuff, working with pdf and word documents, Available at https://automatetheboringstuff.com/2e/chapter15/
[4]. Python Elasticsearch Client, Available at https://elasticsearch-py.readthedocs.io/en/v8.15.0/quickstart.html#getting-documents
[5]. Python Elasticsearch Client - 2, Available at https://www.elastic.co/guide/en/elasticsearch/client/python-api/current/index.html