FastAPI-based Resume Processing Application

### Detailed Design Document for FastAPI-based Resume Processing Application

---

## \*\*1. Overview\*\*

The purpose of this document is to outline the architecture, design, and implementation details of a FastAPI-based application that accepts resumes in PDF or Word Document formats, extracts text content, and returns appropriate responses. The application will also handle errors, provide proper HTTP status codes, and will be deployed on a GCP VM with CI/CD pipelines set up.

---

## \*\*2. System Architecture\*\*

- \*\*Frontend\*\*: None (CLI/Postman or API clients for testing)

- \*\*Backend\*\*: FastAPI (for building REST APIs)

- \*\*Storage\*\*: Local filesystem or GCS Bucket (optional for storing files)

- \*\*Authentication\*\*: API Keys (simple key-based validation)

- \*\*Deployment\*\*: GCP Compute Engine VM

- \*\*CI/CD\*\*: GitHub Actions or GitLab CI/CD for building, testing, and deployment

---

## \*\*3. Requirements\*\*

### \*\*3.1 Functional Requirements\*\*

1. \*\*Upload Endpoint\*\*: An endpoint to accept a resume file (PDF or Word).

2. \*\*File Validation\*\*: Check the file format (PDF or Word) and ensure it is not corrupt.

3. \*\*Text Extraction\*\*: Extract text content from valid files.

4. \*\*Response\*\*: Return appropriate messages with status codes (200 for success, 400 for invalid input, 422 for processing errors).

### \*\*3.2 Non-functional Requirements\*\*

1. \*\*Security\*\*: API key-based authentication.

2. \*\*Performance\*\*: Handle concurrent requests efficiently.

3. \*\*Reliability\*\*: Proper error handling and logging.

4. \*\*Scalability\*\*: Should be easy to scale using GCP managed services if needed.

---

## \*\*4. API Design\*\*

### \*\*4.1 Endpoints\*\*

1. \*\*POST /upload-resume\*\*

- \*\*Description\*\*: Uploads a resume and extracts its text.

- \*\*Headers\*\*:

- `Authorization`: `API-Key {API\_KEY}`

- \*\*Request Body\*\*:

- `file`: Multipart file (PDF/Word document)

- \*\*Responses\*\*:

- \*\*200\*\*: Text content extracted successfully.

- \*\*400\*\*: Invalid file format or missing file.

- \*\*422\*\*: Unable to process the file (corrupt file).

- \*\*401\*\*: Unauthorized (invalid API key).

### \*\*Sample Request and Response\*\*

```json

POST /upload-resume

Headers: {"Authorization": "API-Key abc123"}

Response (200 OK):

{

"message": "Text extracted successfully.",

"content": "John Doe Resume Content..."

}

Response (400 Bad Request):

{

"message": "Invalid file format. Only PDF and Word documents are allowed."

}

Response (422 Unprocessable Entity):

{

"message": "File is corrupted or cannot be processed."

}

```

---

## \*\*5. Authentication\*\*

- \*\*API Key Storage\*\*: A configuration file (`config.py`) will store API keys.

- \*\*Validation\*\*: Middleware or dependency function to validate the key.

- \*\*Example\*\*:

```python

from fastapi import Depends, HTTPException, Security

from fastapi.security.api\_key import APIKeyHeader

API\_KEY = "abc123"

api\_key\_header = APIKeyHeader(name="Authorization", auto\_error=False)

def get\_api\_key(api\_key\_header: str = Security(api\_key\_header)):

if api\_key\_header != f"API-Key {API\_KEY}":

raise HTTPException(status\_code=401, detail="Invalid API key")

return api\_key\_header

```

---

## \*\*6. Text Extraction Logic\*\*

- \*\*Libraries Used\*\*:

- `python-docx`: For Word document text extraction.

- `PyMuPDF` or `pdfminer.six`: For PDF text extraction.

- \*\*Implementation\*\*:

```python

from fastapi import FastAPI, File, UploadFile, HTTPException, Depends

import os

import fitz # PyMuPDF for PDFs

import docx2txt # For Word Documents

app = FastAPI()

@app.post("/upload-resume")

async def upload\_resume(file: UploadFile = File(...), api\_key: str = Depends(get\_api\_key)):

if file.content\_type not in ["application/pdf", "application/vnd.openxmlformats-officedocument.wordprocessingml.document"]:

raise HTTPException(status\_code=400, detail="Invalid file format. Only PDF and Word documents are allowed.")

try:

if file.content\_type == "application/pdf":

pdf\_bytes = await file.read()

doc = fitz.open(stream=pdf\_bytes, filetype="pdf")

text = "\n".join([page.get\_text() for page in doc])

else:

text = docx2txt.process(file.file)

return {"message": "Text extracted successfully.", "content": text}

except Exception as e:

raise HTTPException(status\_code=422, detail="File is corrupted or cannot be processed.")

```

---

## \*\*7. CI/CD Pipeline\*\*

### \*\*7.1 Git Repository\*\*

- Repository to host all source code, including FastAPI application, Dockerfile, and configuration files.

### \*\*7.2 CI/CD Steps\*\*

1. \*\*Build Stage\*\*:

- Install dependencies.

- Run unit tests.

- Linting and formatting checks.

2. \*\*Dockerize the Application\*\*:

- Create `Dockerfile`:

```dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

```

3. \*\*Push to Container Registry\*\*:

- Push Docker image to GCP Container Registry.

4. \*\*Deploy to GCP Compute Engine\*\*:

- Create a VM instance.

- SSH into the instance and pull the latest Docker image.

- Run the container.

### \*\*Sample GitHub Actions Workflow\*\*

```yaml

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Check out repository

uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: '3.9'

- name: Install dependencies

run: pip install -r requirements.txt

- name: Run tests

run: pytest

- name: Lint and format checks

run: black --check .

deploy:

runs-on: ubuntu-latest

needs: build

steps:

- name: Deploy to GCP VM

run: |

gcloud auth activate-service-account --key-file=${{ secrets.GCP\_KEY\_FILE }}

gcloud compute ssh ${GCP\_VM\_NAME} --command="docker pull <image-url> && docker run -d -p 8000:8000 <image-url>"

```

---

## \*\*8. GCP VM Configuration\*\*

1. \*\*Create a GCP Compute Engine VM\*\*:

- Machine type: e2-standard-2

- OS: Ubuntu 24.04 LTS

2. \*\*Install Docker\*\*:

```bash

sudo apt update

sudo apt install -y docker.io

sudo systemctl enable docker

```

3. \*\*Allow HTTP/HTTPS Traffic\*\*:

- Open firewall rules to allow incoming requests on port 8000.

4. \*\*Run Docker Container\*\*:

```bash

docker run -d -p 8000:8000 <image-url>

```

---

## \*\*9. Monitoring and Logging\*\*

- Use `FastAPI` logging for API request and error logs.

- Enable GCP Stackdriver Logging and Monitoring for tracking API metrics.

---

## \*\*10. Future Enhancements\*\*

- Add support for other document formats (e.g., plain text).

- Implement JWT-based authentication for improved security.

- Scale using GKE (Google Kubernetes Engine) if traffic increases.

---

This design ensures a robust, scalable, and secure FastAPI-based resume processing service with CI/CD and deployment to GCP.

Tracker for the project –

Initial setup for environment – Git, IDE, Python, libraries, Docker

API Key authentication –

Extract test form pdf, docs – python code implementation –

|  |
| --- |
| Initial setup for designing environment – VSCode, Python, Git, docker |
| API Implementation – implement POST/upload-resume endpoint |
| Text Extraction Logic – python libraries using PDf, word files |
| Authentication Setup – Add API key authentication |
| Error Handling – add code to handle errors (invalid format, corrupt, invalid api key) |
| Dockerization – Make dockerfile |
| CI/CD Pipeline – Make git actions pipeline |
| GCP Deployment – move application to GCP Compute Engine VM |
| Testing – Complete testing, unit testing |

Initial setup for environment –

* **VS Code:** Install and install the Python extension.
* **Python:** Install Python 3.9+
* **Git:** Install and configure your Git username and email.
* **Docker:** Install Docker Desktop.
* **Postman:** Install Postman.

API implementation –

Requirements -

pip install “fastapi[standard]”

pip install fastapi uvicorn

Create main.py in put code –

from fastapi import FastAPI, File, UploadFile, HTTPException # type: ignore

app = FastAPI()

@app.post("/upload-resume")

async def upload\_resume(file: UploadFile = File(...)):

if not file:

raise HTTPException(status\_code=400, detail="No file uploaded.")

return {"filename": file.filename, "content\_type": file.content\_type}

Commands to run

* uvicorn app:main --reload
* curl -X POST \

-H "Authorization: API-Key your\_secret\_api\_key" \

-F "file=@./test\_resume.pdf" \

http://127.0.0.1:8000/upload-resume

* curl -X POST -F "file=@ /Users/bharadwajdhornala/Desktop/Final\_versions\_resume/ BharadwajD\_Resume.pdf" http://127.0.0.1:8000/upload-resume

**curl** is used Local testing ->

-X POST -> specifies POST http method

-H “Authorization: API-Key your\_secret\_api\_key” -> authorization header with API key

-F “file=@file\_path” -> sends file as multipart form data

<http://127.0.0.1:8000/upload-resume> -> API endpoints

Postman for testing ->

**2. Using Postman (GUI):**

Postman is a popular GUI tool for API testing.

* **Create a new request:** Select POST as the method.
* **Enter the URL:** http://<your-api-address>:<port>/upload-resume
* **Headers:** Add a header with the key Authorization and the value API-Key your\_secret\_api\_key.
* **Body:** Select the "form-data" option. Add a key named file and select the file type as "File". Choose your resume file.
* **Send the request:** Click the "Send" button.

Postman will display the response from your API, including the status code, headers, and body.

**3. Using requests library in Python (for automated testing or scripts):**

If you want to automate your API tests or integrate them into a testing script, you can use the requests library in Python:

Python

import requests

url = "http://127.0.0.1:8000/upload-resume" # Replace with your API URL

api\_key = "your\_secret\_api\_key" # Replace with your API key

file\_path = "./test\_resume.pdf" # Replace with the path to your file

headers = {"Authorization": f"API-Key {api\_key}"}

files = {"file": open(file\_path, "rb")}

try:

response = requests.post(url, headers=headers, files=files)

response.raise\_for\_status() # Raise an exception for bad status codes (4xx or 5xx)

print(response.json()) # Print the JSON response

except requests.exceptions.RequestException as e:

print(f"Error: {e}")

Remember to install the requests library: pip install requests

**Complete code execution –**

from fastapi import FastAPI, File, UploadFile, HTTPException, Depends # type: ignore

from fastapi.security.api\_key import APIKeyHeader # type: ignore

import fitz # PyMuPDF # type: ignore

import docx2txt # type: ignore

import io

API\_KEY = "api-123"

api\_key\_header = APIKeyHeader(name="Authorization", auto\_error=False)

app = FastAPI()

def get\_api\_key(api\_key\_header: str = Depends(api\_key\_header)):

if not api\_key\_header or api\_key\_header != f"API-Key {API\_KEY}":

raise HTTPException(status\_code=401, detail="Invalid API key")

return api\_key\_header

@app.post("/upload-resume", dependencies=[Depends(get\_api\_key)])

async def upload\_resume(file: UploadFile = File(...)):

if not file:

raise HTTPException(status\_code=400, detail="No file uploaded.")

try:

if file.content\_type == "application/pdf":

pdf\_bytes = await file.read()

doc = fitz.open(stream=pdf\_bytes, filetype="pdf")

text = "\n".join([page.get\_text() for page in doc])

elif file.content\_type == "application/vnd.openxmlformats-officedocument.wordprocessingml.document":

contents = await file.read()

text = docx2txt.process(io.BytesIO(contents))

else:

raise HTTPException(status\_code=400, detail="Invalid file format")

return {"filename": file.filename, "content": text}

except Exception as e:

print(f"Error: {e}") #Print error for debugging

raise HTTPException(status\_code=422, detail="Error processing file")

commands to run –

* uvicorn app:main –reload
* curl -X POST -H "Authorization: API-Key api-123"-F [file=@./BharadwajD\_Resume.pdf](mailto:file=@./BharadwajD_Resume.pdf) http://127.0.0.1:8000/upload-resume