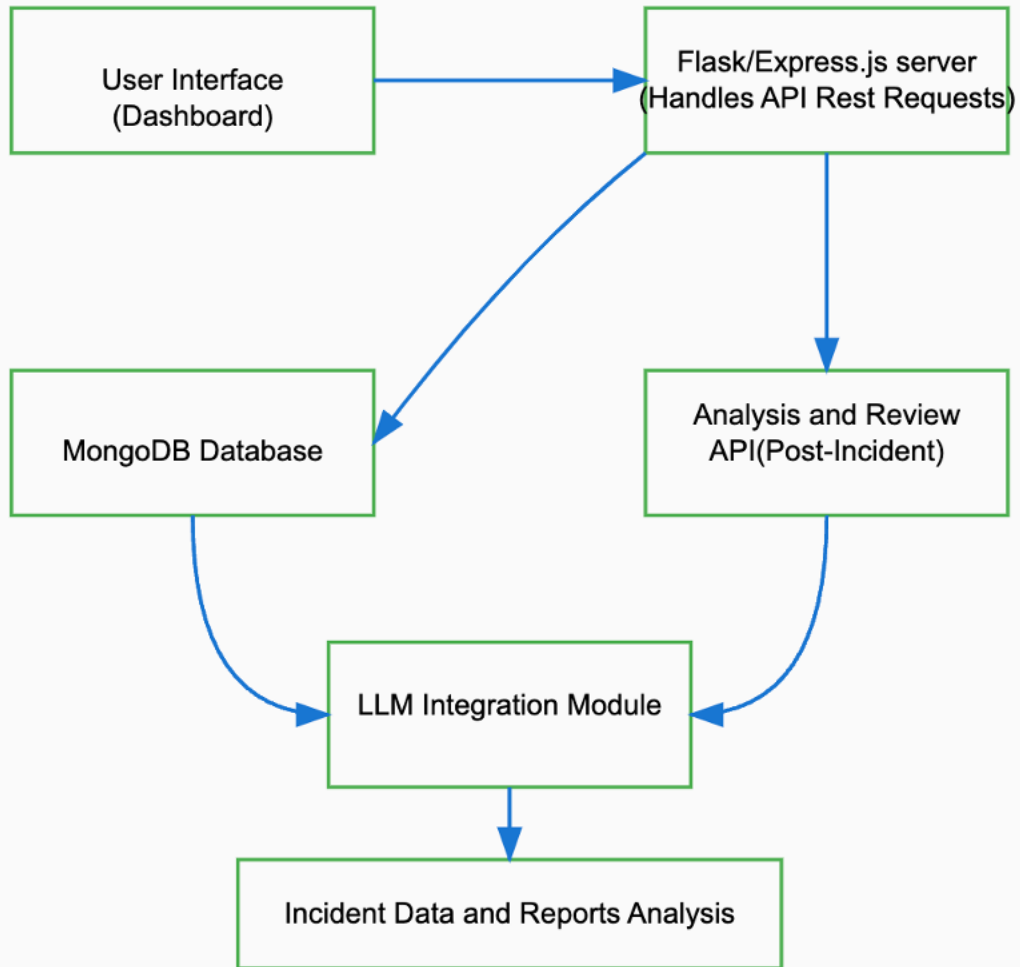


## Module 5: Crisis Management

### Architecture Diagram of Crisis Management:



## Initial Design for DB and API routes:

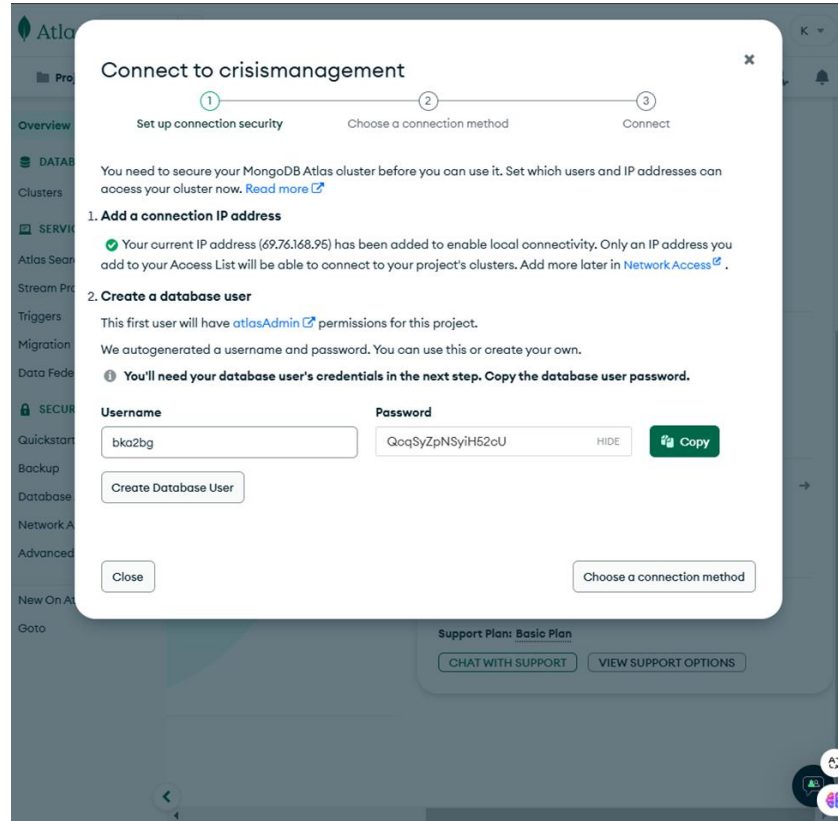
Schema of the DB:

```
class CrisisEvent(Document):
    title = StringField(required=True)
    severity = StringField(required=True, choices=['low', 'medium', 'high', 'critical'])
    status = StringField(required=True, default='active', choices=['active', 'resolved', 'archived'])
    description = StringField(required=True)
    type = StringField(required=True)
    location = StringField()
    affected_assets = ListField(StringField())
    incident_id = StringField() # Reference to incident response module
    communication_log = ListField(DictField())
    created_at = DateTimeField(default=datetime.utcnow)
    updated_at = DateTimeField(default=datetime.utcnow)
    resolution_time = IntField() # Time to resolve in minutes
```

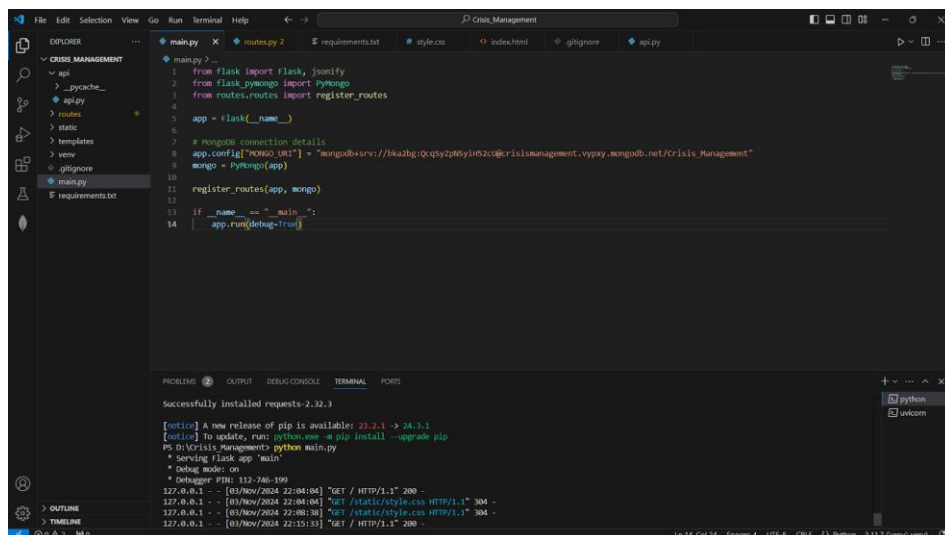
## API Endpoints Setup:

1. Get Crisis:
  - GET /api/documents- get all crisis details
  - GET /api/documents/<id> - get specific crisis details
2. Post Crisis
  - POST /api/documents/add- post crisis details

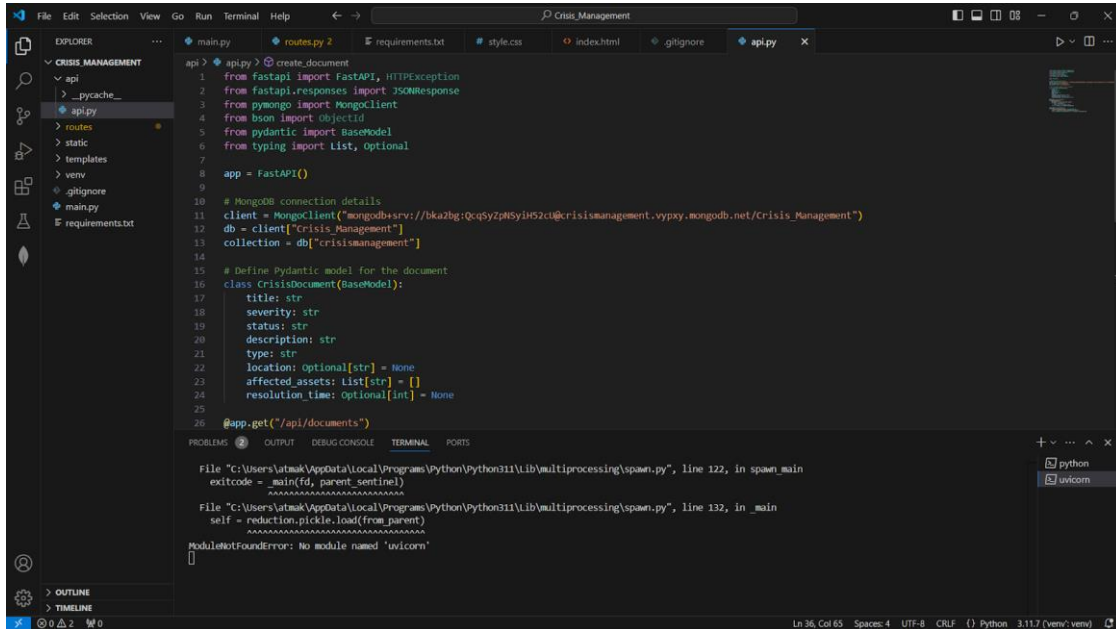
Installed MongoDB Compass app and connect using the private connection key.



Front-end code setup in VS Code:

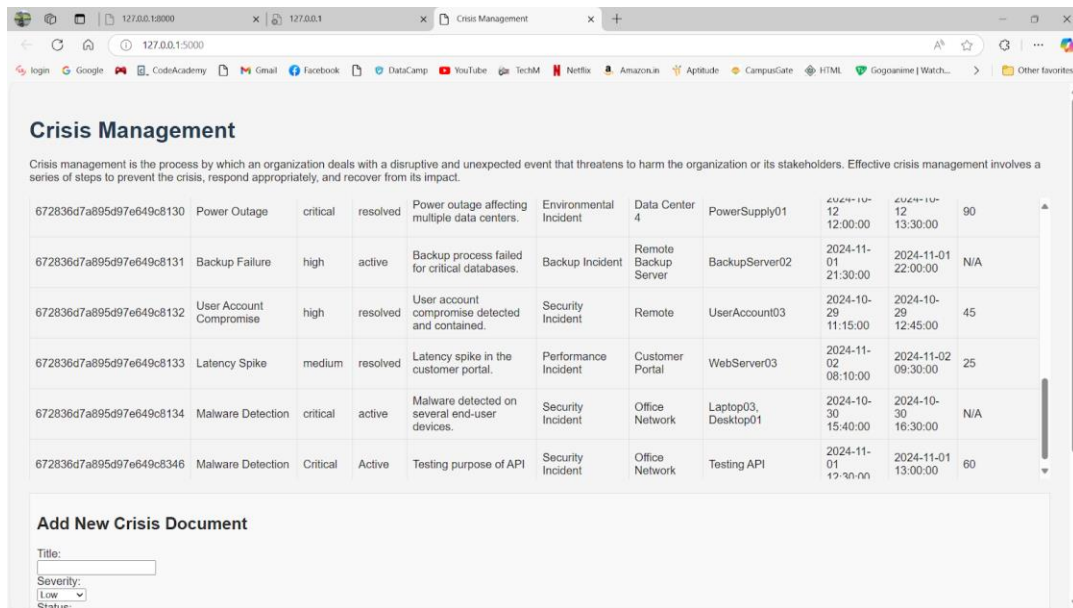


## Backend code setup in VS Code:



```
1 from fastapi import FastAPI, HTTPException
2 from fastapi.responses import JSONResponse
3 from pymongo import MongoClient
4 from bson import ObjectId
5 from pydantic import BaseModel
6 from typing import List, Optional
7
8 app = FastAPI()
9
10 # MongoDB connection details
11 client = MongoClient("mongodb+srv://bka2hg:QcqsYzP8SyjH52cU@crisismanagement.vypry.mongodb.net/Crisis_Management")
12 db = client["Crisis_Management"]
13 collection = db["crisismanagement"]
14
15 # Define Pydantic model for the document
16 class CrisisDocument(BaseModel):
17     title: str
18     severity: str
19     status: str
20     description: str
21     type: str
22     location: Optional[str] = None
23     affected_assets: List[str] = []
24     resolution_time: Optional[int] = None
25
26 @app.get("/api/documents")
27 def get_documents():
28     documents = collection.find()
29     return documents
```

UI access through localhost <http://127.0.0.1:5000>



MongoDB database data in Compass app:

The screenshot displays the MongoDB Compass application interface. On the left, a sidebar shows the database structure: 'crisismanagement.vypxy.mongodb.net' > 'Crisis\_Management' > 'crisismanagement'. The main panel shows two JSON documents from the 'crisismanagement' collection. The top document is a 'Malware Detection' incident with a 'critical' severity and 'active' status. The bottom document is a 'Security Incident' for 'Testing purpose of API' with an 'Active' status. Both documents include fields for 'title', 'severity', 'status', 'description', 'type', 'location', 'affected\_assets', 'incident\_id', 'communication\_log', 'created\_at', 'updated\_at', and 'resolution\_time'.

```
{
  "_id": {},
  "title": "Malware Detection",
  "severity": "critical",
  "status": "active",
  "description": "Malware detected on several end-user devices.",
  "type": "Security Incident",
  "location": "Office Network",
  "affected_assets": [],
  "incident_id": "IR-2024-015",
  "communication_log": [],
  "created_at": {},
  "updated_at": {},
  "resolution_time": null
}
```

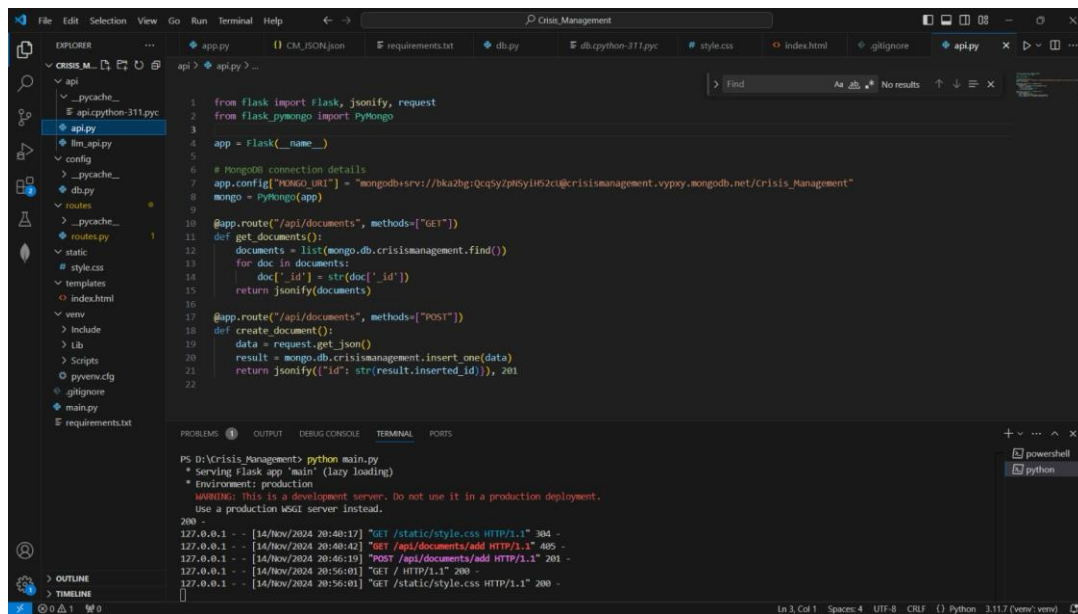
```
{
  "_id": {},
  "title": "Malware Detection",
  "severity": "Critical",
  "status": "Active",
  "description": "Testing purpose of API",
  "type": "Security Incident",
  "location": "Office Network",
  "affected_assets": [],
  "incident_id": "IR-2024-090",
  "communication_log": [],
  "created_at": {},
  "updated_at": {},
  "resolution_time": 60
}
```

## Week 3 Updates

**Create APIs for Incident Logging:** We have created the API's for performing the CRUD operations on the database. We have tested the API end points using Postman.

## API CRUD Operations

app.py



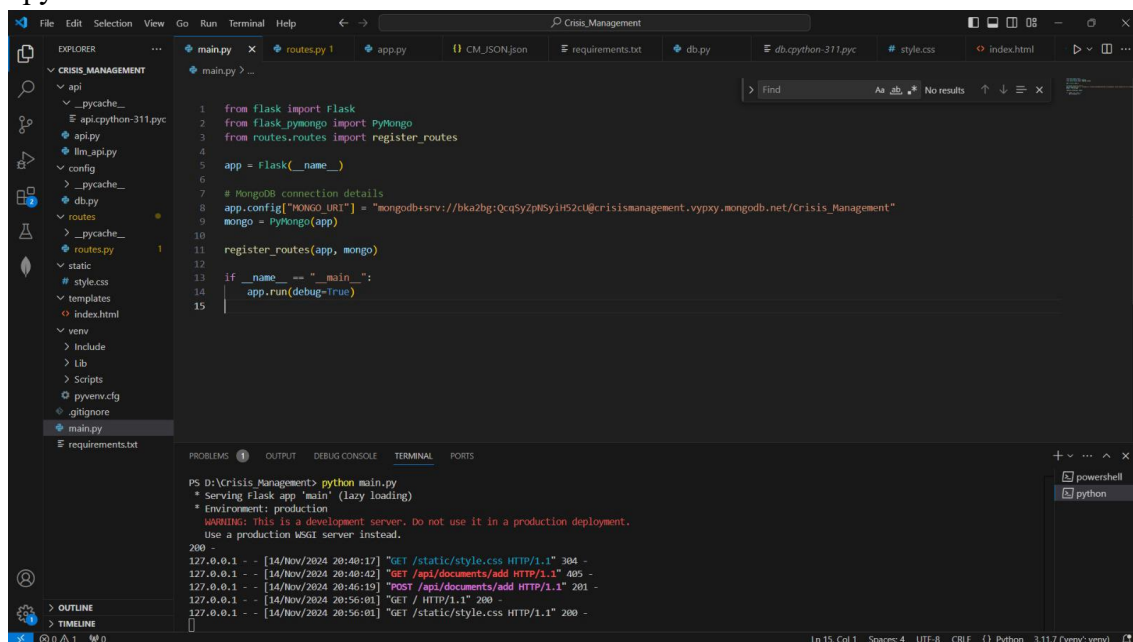
The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the code for `app.py`. The code imports Flask, jsonify, request, and PyMongo. It sets up a Flask app, configures MongoDB connection details, and defines two routes: `/api/documents` for GET (retrieving documents) and `/api/documents` for POST (creating a new document). The terminal at the bottom shows the command `python main.py` being executed, with output indicating the Flask app is running on `0.0.0.0` and the MongoDB connection is established.

```
1 from flask import Flask, jsonify, request
2 from flask_pymongo import PyMongo
3
4 app = Flask(__name__)
5
6 # MongoDB connection details
7 app.config["MONGO_URI"] = "mongodb+srv://bka2bg:QcQ5y2pt5y1H62ct@crisismanagement.vypxy.mongodb.net/Crisis_Management"
8 mongo = PyMongo(app)
9
10 @app.route("/api/documents", methods=["GET"])
11 def get_documents():
12     documents = list(mongo.db.crisismanagement.find())
13     for doc in documents:
14         doc['_id'] = str(doc['_id'])
15     return jsonify(documents)
16
17 @app.route("/api/documents", methods=["POST"])
18 def create_document():
19     data = request.get_json()
20     result = mongo.db.crisismanagement.insert_one(data)
21     return jsonify({"id": str(result.inserted_id)}), 201
22
```

Terminal Output:

```
PS D:\Crisis_Management> python main.py
* Serving Flask app "main" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
200 -
127.0.0.1 -- [14/Nov/2024 20:40:17] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 -- [14/Nov/2024 20:40:42] "GET /api/documents/add HTTP/1.1" 405 -
127.0.0.1 -- [14/Nov/2024 20:46:19] "POST /api/documents/add HTTP/1.1" 201 -
127.0.0.1 -- [14/Nov/2024 20:56:01] "GET / HTTP/1.1" 200 -
127.0.0.1 -- [14/Nov/2024 20:56:01] "GET /static/style.css HTTP/1.1" 200 -
```

main.py



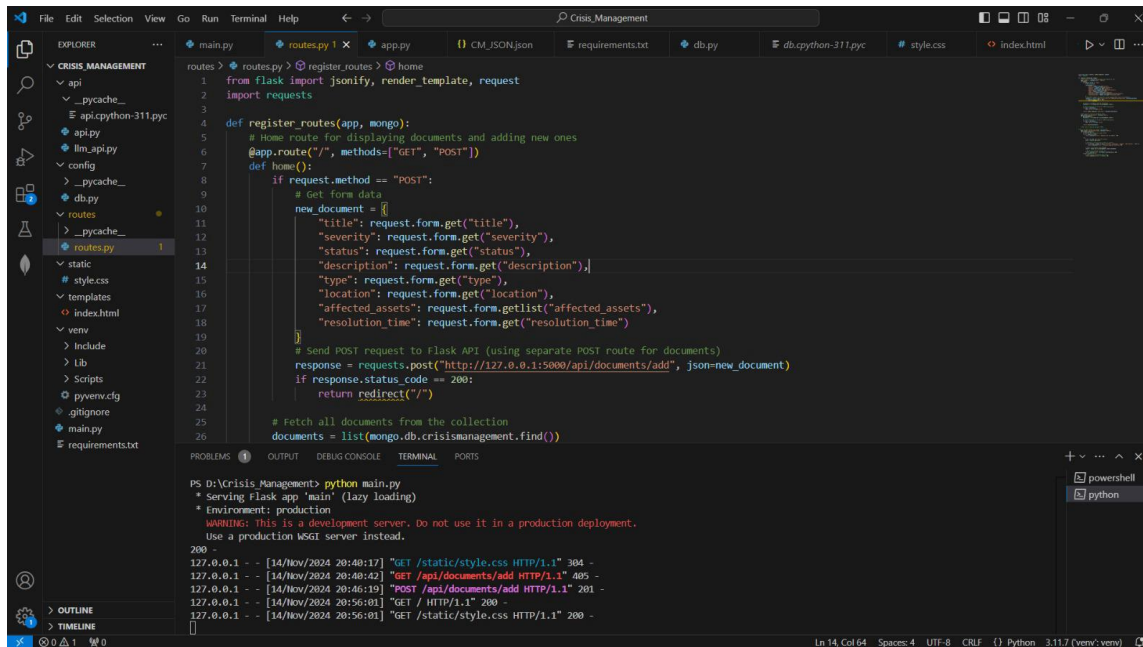
The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the code for `main.py`. The code imports Flask, jsonify, request, and PyMongo. It sets up a Flask app, configures MongoDB connection details, and defines two routes: `/api/documents` for GET (retrieving documents) and `/api/documents` for POST (creating a new document). The terminal at the bottom shows the command `python main.py` being executed, with output indicating the Flask app is running on `0.0.0.0` and the MongoDB connection is established.

```
1 from flask import Flask
2 from flask_pymongo import PyMongo
3 from routes.routes import register_routes
4
5 app = Flask(__name__)
6
7 # MongoDB connection details
8 app.config["MONGO_URI"] = "mongodb+srv://bka2bg:QcQ5y2pt5y1H62ct@crisismanagement.vypxy.mongodb.net/Crisis_Management"
9 mongo = PyMongo(app)
10
11 register_routes(app, mongo)
12
13 if __name__ == "__main__":
14     app.run(debug=True)
15
```

Terminal Output:

```
PS D:\Crisis_Management> python main.py
* Serving Flask app "main" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
200 -
127.0.0.1 -- [14/Nov/2024 20:40:17] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 -- [14/Nov/2024 20:40:42] "GET /api/documents/add HTTP/1.1" 405 -
127.0.0.1 -- [14/Nov/2024 20:46:19] "POST /api/documents/add HTTP/1.1" 201 -
127.0.0.1 -- [14/Nov/2024 20:56:01] "GET / HTTP/1.1" 200 -
127.0.0.1 -- [14/Nov/2024 20:56:01] "GET /static/style.css HTTP/1.1" 200 -
```

## routes.py



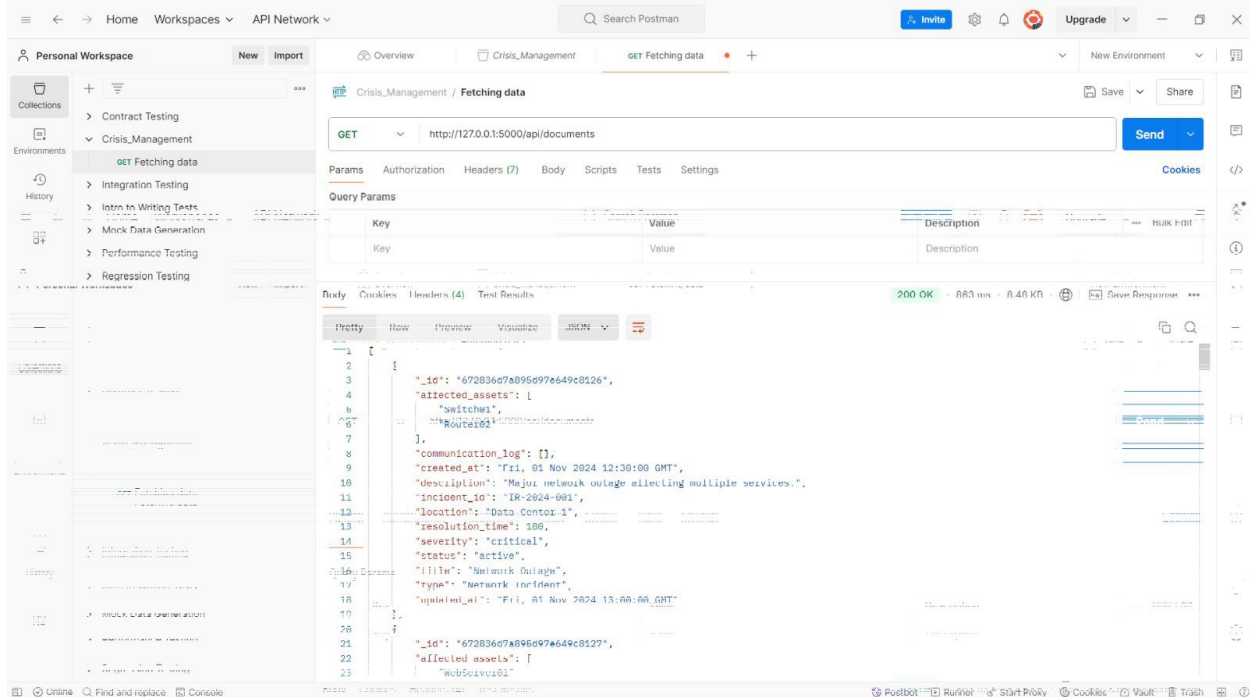
```
1 from flask import jsonify, render_template, request
2 import requests
3
4 def register_routes(app, mongo):
5     # Home route for displaying documents and adding new ones
6     @app.route("/", methods=["GET", "POST"])
7     def home():
8         if request.method == "POST":
9             # Get form data
10             new_document = {
11                 "title": request.form.get("title"),
12                 "severity": request.form.get("severity"),
13                 "status": request.form.get("status"),
14                 "description": request.form.get("description"),
15                 "type": request.form.get("type"),
16                 "location": request.form.get("location"),
17                 "affected_assets": request.form.getlist("affected_assets"),
18                 "resolution_time": request.form.get("resolution_time")
19             }
20             # Send POST request to Flask API (using separate POST route for documents)
21             response = requests.post("http://127.0.0.1:5000/api/documents/add", json=new_document)
22             if response.status_code == 200:
23                 return redirect("/")
24
25     # Fetch all documents from the collection
26     documents = list(mongo.db.crisismanagement.find())
```

PS D:\Crisis\_Management> python main.py  
\* Serving Flask app 'main' (lazy loading)  
\* Environment: production  
WARNING: This is a development server. Do not use it in a production deployment.  
Use a production WSGI server instead.

200 -  
127.0.0.1 - - [14/Nov/2024 20:40:17] "GET /static/style.css HTTP/1.1" 304 -  
127.0.0.1 - - [14/Nov/2024 20:40:42] "GET /api/documents/add HTTP/1.1" 405 -  
127.0.0.1 - - [14/Nov/2024 20:46:19] "POST /api/documents/add HTTP/1.1" 201 -  
127.0.0.1 - - [14/Nov/2024 20:56:01] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [14/Nov/2024 20:56:01] "GET /static/style.css HTTP/1.1" 200 -

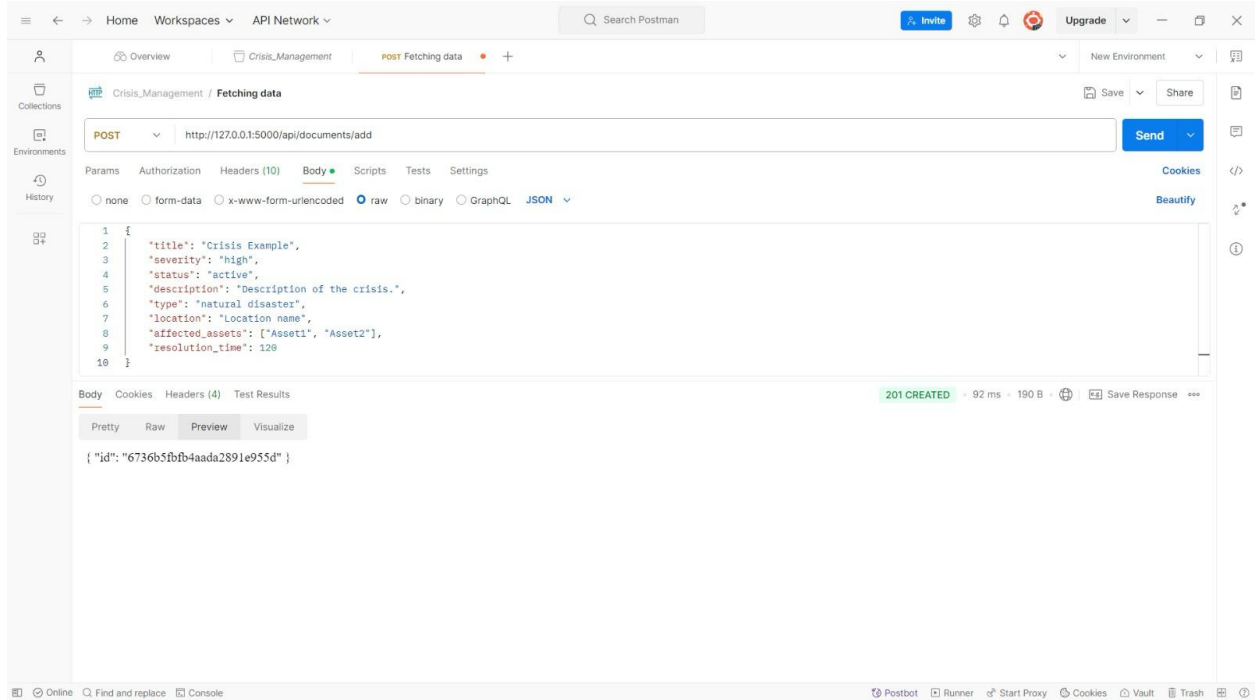
## Testing:

### 1. Testing the GET API endpoint using Postman

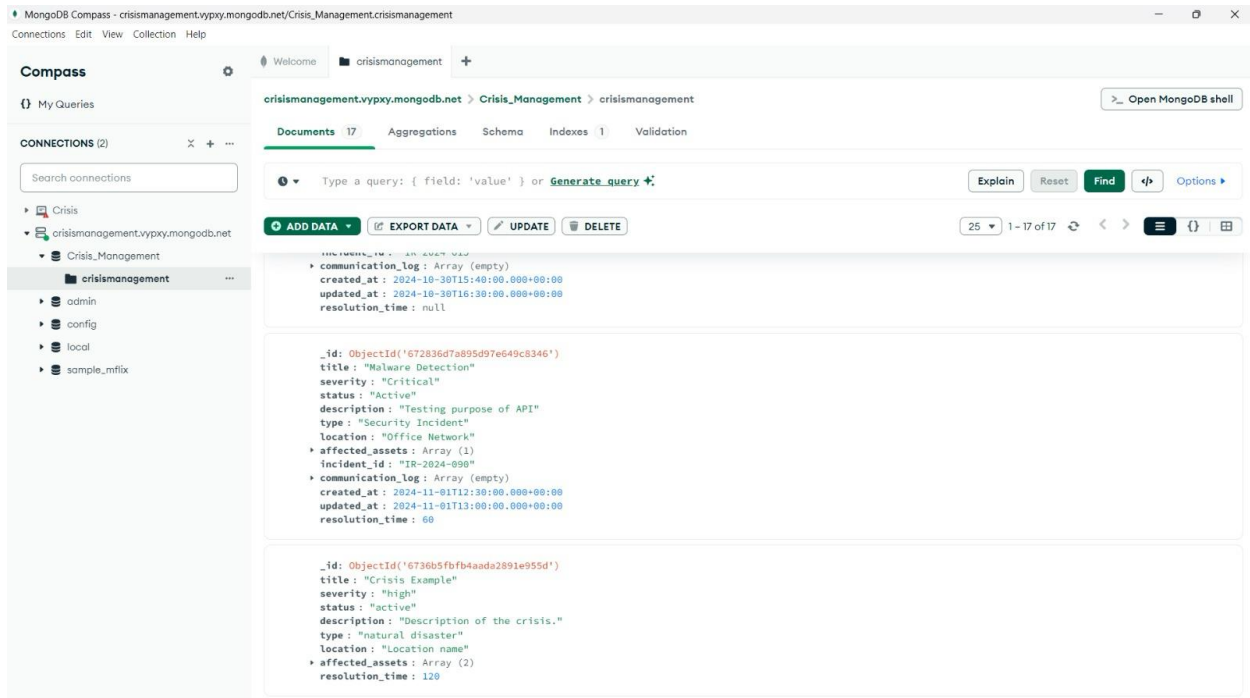




## 2. Testing the POST API using postman:



## 3. Mongodb-post working: Updated record in the database from POST API





#### 4. Reflection in front end:

127.0.0.1:5000

login Google USICS CodeAcademy Gmail DataCamp YouTube Amazon.in BITHUB books Excel Download Latest M... Bookmarks All Bookmarks

### Crisis Management

Crisis management is the process by which an organization deals with a disruptive and unexpected event that threatens to harm the organization or its stakeholders. Effective crisis management involves a series of steps to prevent the crisis, respond appropriately, and recover from its impact.

672836d7a895d97e649c8131	Backup Failure	high	active	Backup process failed for critical databases.	Backup Incident	Remote Backup Server	BackupServer02	2024-11-01 21:30:00	2024-11-01 22:00:00	N/A
672836d7a895d97e649c8132	User Account Compromise	high	resolved	User account compromise detected and contained.	Security Incident	Remote	UserAccount03	2024-10-29 11:15:00	2024-10-29 12:45:00	45
672836d7a895d97e649c8133	Latency Spike	medium	resolved	Latency spike in the customer portal.	Performance Incident	Customer Portal	WebServer03	2024-11-02 08:10:00	2024-11-02 09:30:00	25
672836d7a895d97e649c8134	Malware Detection	critical	active	Malware detected on several end-user devices.	Security Incident	Office Network	Laptop03, Desktop01	2024-10-30 15:40:00	2024-10-30 16:30:00	N/A
672836d7a895d97e649c8346	Malware Detection	Critical	Active	Testing purpose of API	Security Incident	Office Network	Testing API	2024-11-01 12:30:00	2024-11-01 13:00:00	60
6736b5fbf4aada2891e955d	Crisis Example	high	active	Description of the crisis.	natural disaster	Location name	Asset1, Asset2			120

#### Add New Crisis Document

Title:

Severity:

Low

## Core functionality:

### LLM – distilgpt2

File Edit Selection View Go Run Terminal Help

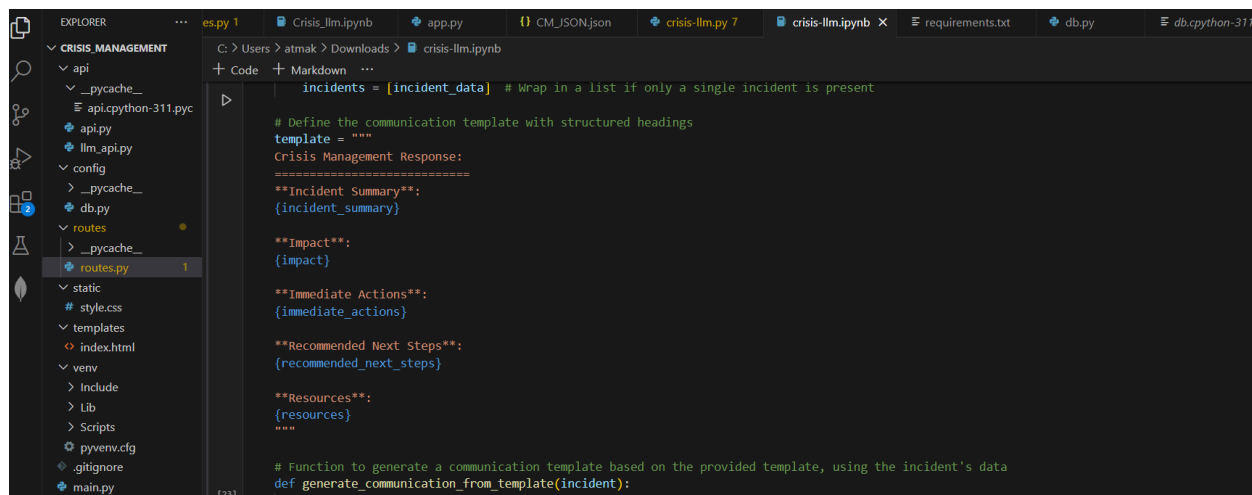
CRISIS MANAGEMENT

api \_pycache\_ api.py llm\_api.py config \_pycache\_ db.py routes \_pycache\_ routes.py static style.css templates index.html venv Include Lib Scripts pyvenv.cfg .gitignore main.py requirements.txt

import torch  
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM  
import json  
import os  
import pandas as pd  
  
# Check if GPU is available  
device = 0 if torch.cuda.is\_available() else -1  
  
# Optional: Set environment variables for memory allocation optimization  
os.environ["PYTORCH\_CUDA\_ALLOC\_CONF"] = 'max\_split\_size\_mb:128'  
  
# Load the model and tokenizer for DistilGPT2 (smaller LLM)  
model\_name = "distilgpt2" # A smaller LLM model, alternative to GPT-3.5  
pipe = pipeline("text-generation", model=model\_name, device=device)  
  
# Alternatively, you can load the tokenizer and model explicitly if needed:  
tokenizer = AutoTokenizer.from\_pretrained(model\_name)  
model = AutoModelForCausalLM.from\_pretrained(model\_name)  
  
# Set the padding token to eos\_token (commonly used as the padding token for models without a pad\_token)  
tokenizer.pad\_token = tokenizer.eos\_token

PS D:\Crisis\_Management> python main.py  
\* Serving Flask app "main" (lazy loading)  
\* Environment: production  
200 -  
127.0.0.1 - - [14/Nov/2024 20:40:17] "GET /static/style.css HTTP/1.1" 304 -  
127.0.0.1 - - [14/Nov/2024 20:40:42] "GET /api/documents/add HTTP/1.1" 405 -  
127.0.0.1 - - [14/Nov/2024 20:46:19] "POST /api/documents/add HTTP/1.1" 201 -  
127.0.0.1 - - [14/Nov/2024 20:56:01] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [14/Nov/2024 20:56:01] "GET /static/style.css HTTP/1.1" 200 -

## Communication template:



```
incidents = [incident_data] # Wrap in a list if only a single incident is present

# Define the communication template with structured headings
template = """
Crisis Management Response:
=====
**Incident Summary**:
{incident_summary}

**Impact**:
{impact}

**Immediate Actions**:
{immediate_actions}

**Recommended Next Steps**:
{recommended_next_steps}

**Resources**:
{resources}
"""

# Function to generate a communication template based on the provided template, using the incident's data
def generate_communication_from_template(incident):
    """
```

## Output tested at Kaggle website for GPU accessibility-

```
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Setting 'pad_token_id' to 'eos_token_id':None for open-end generation.
Generated Crisis Communication Template:

Crisis Management Response:
=====
**Incident Summary**:
Major network outage affecting multiple services.

**Impact**:
Severity: critical, Affected Assets: Switch01, Router02 Severity: critical, Affected Assets: Switch01, Router02 Severity: critical, Affected Assets: Switch01, Router02 Severity: critical, Affected Assets: Switch01, Router02

**Immediate Actions**:
Isolate affected network components, start immediate diagnostics, involve technical teams, and begin root cause analysis. Ensure continuous communication with stakeholders.

**Recommended Next Steps**:
Conduct thorough investigation, communicate with affected users, coordinate with cybersecurity and network teams to apply mitigation strategies. Ensure all impacted systems are restored.

**Resources**:
Cybersecurity team, network engineers, customer support for communication, incident management team, and external partners for advanced mitigation support.

**Resources**:
Maintain and mitigate the vulnerability in a network environment.

**Resources**:
Conduct detailed internal security risk assessment on a distributed network.
```

## Github Link -

[https://github.com/RTTIP/real\\_time\\_threat\\_intelligence/tree/crisis\\_management\\_dev](https://github.com/RTTIP/real_time_threat_intelligence/tree/crisis_management_dev)

### **Integration work with other modules -**

We have not yet received any updated information from Module 4, so we tested the data using our custom created data.

### **Testing Strategies -**

Right now, we have tested our APIs using Postman and it was successful. But integration of it with other modules must be done, and we plan to do that in week 4.

LLM API testing needs to be done and must be integrated with our own API framework and implementation of CRUD into the database.

Once the API, DB connection to and from the LLM and is taken care of, we will give the endpoints to module to integrate our endpoints and also will ask them to act as testers for our API endpoints.

We've been trying different LLM models to generate the templates but running out of memory issues.

We're still left with developing the post-incident response using LLMs which we will complete by week 4.