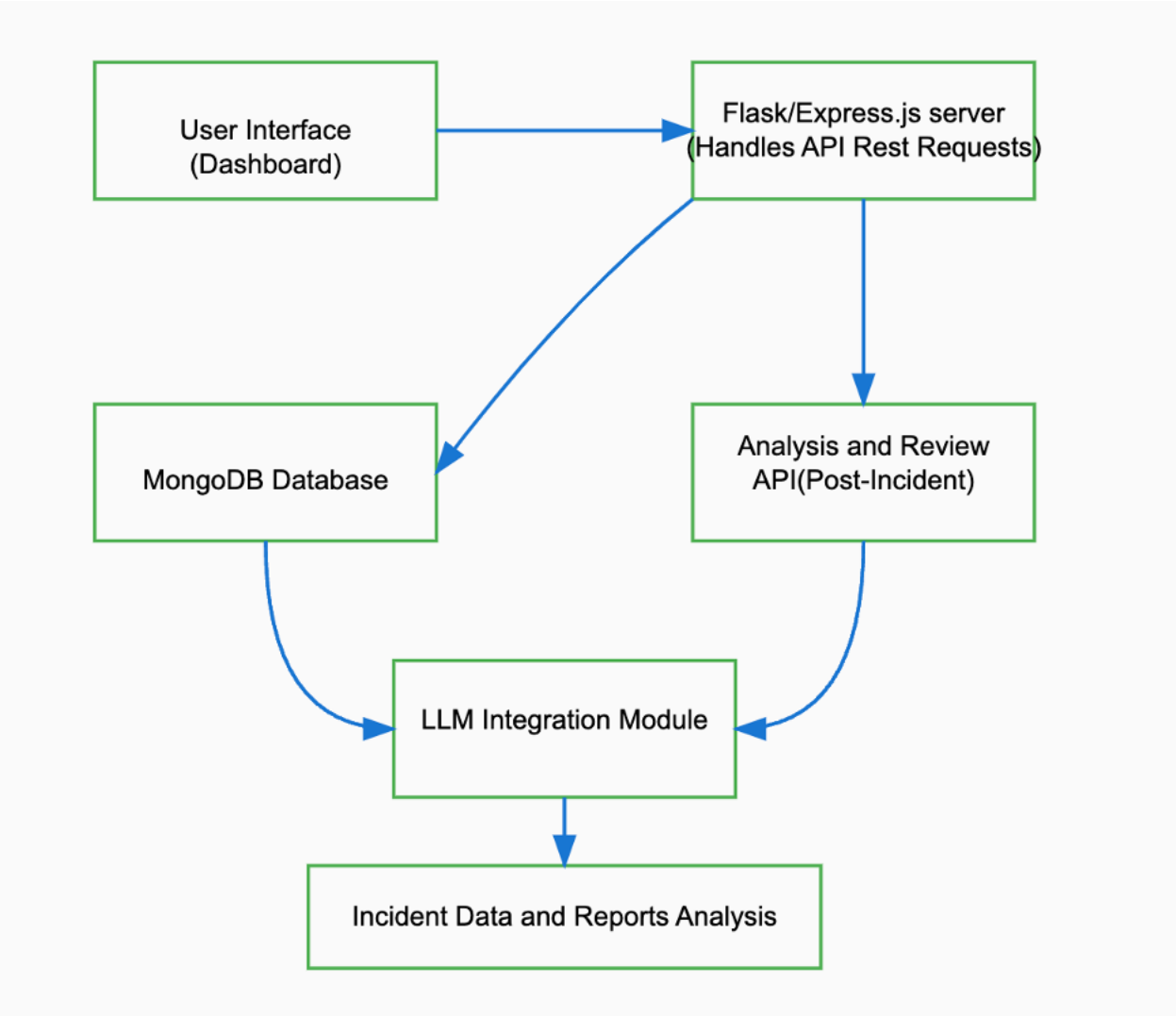
Module 5: Crisis Management

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:

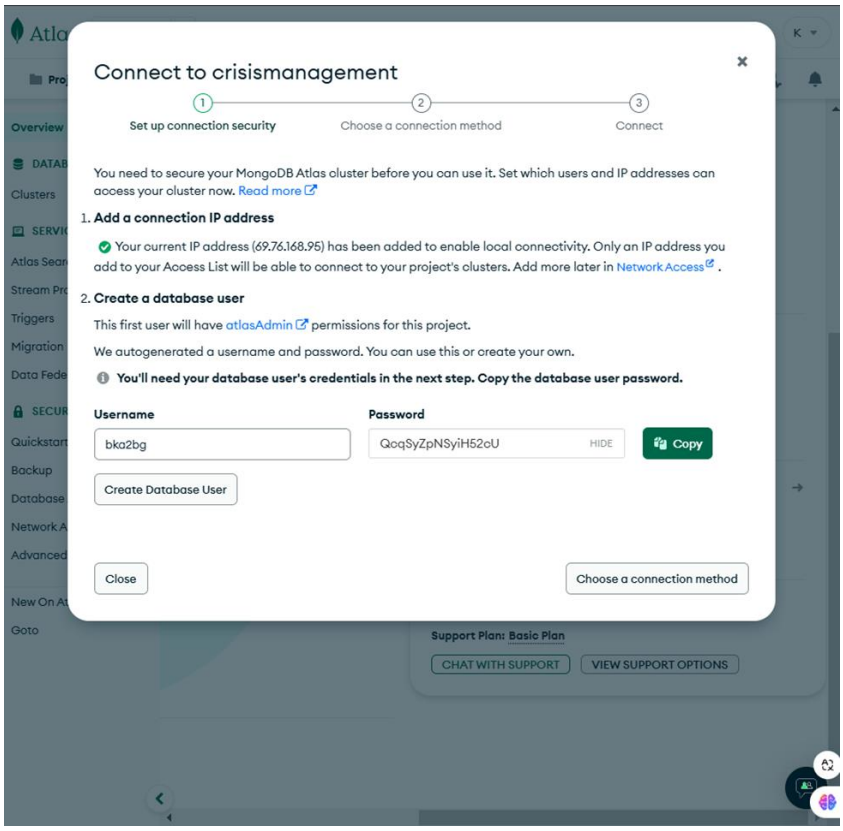
1. Get Crisis:

- GET /api/documents- get all crisis details
- GET /api/documents/<id> - get specific crisis details

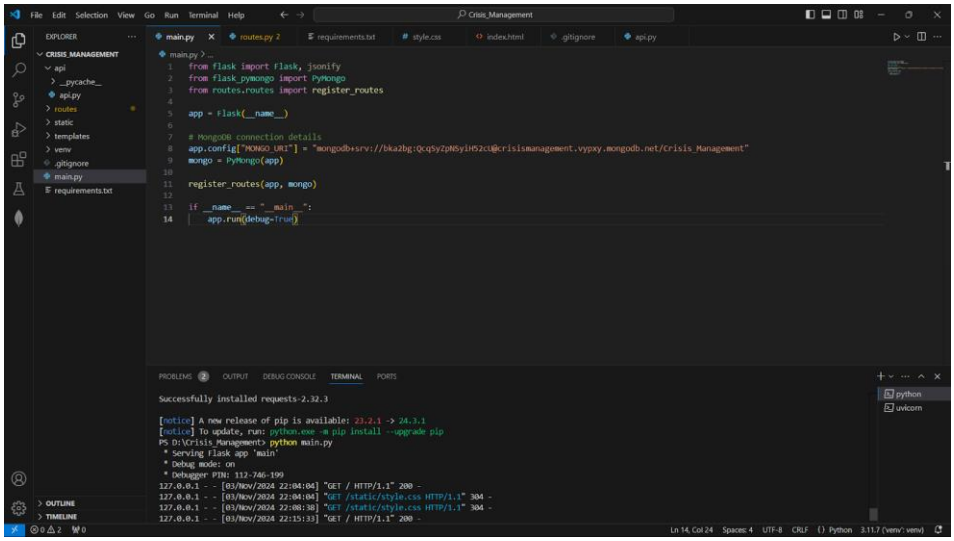
2. Post Crisis

- POST /api/documents/add- post crisis details

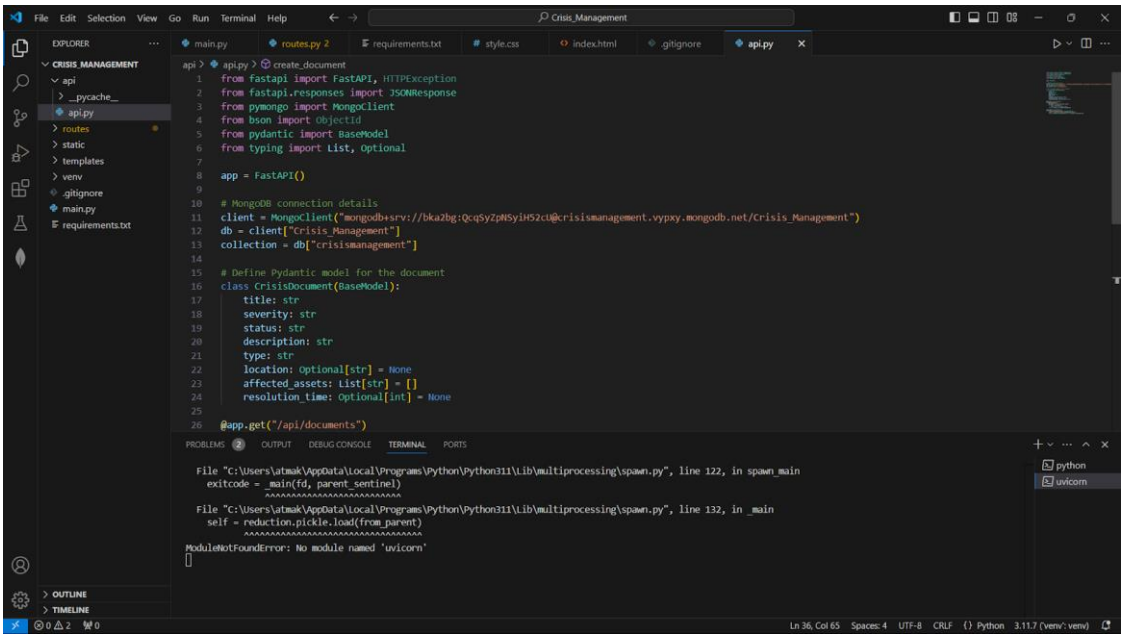
Installed MongoDB Compass app and connect using the private connection key.



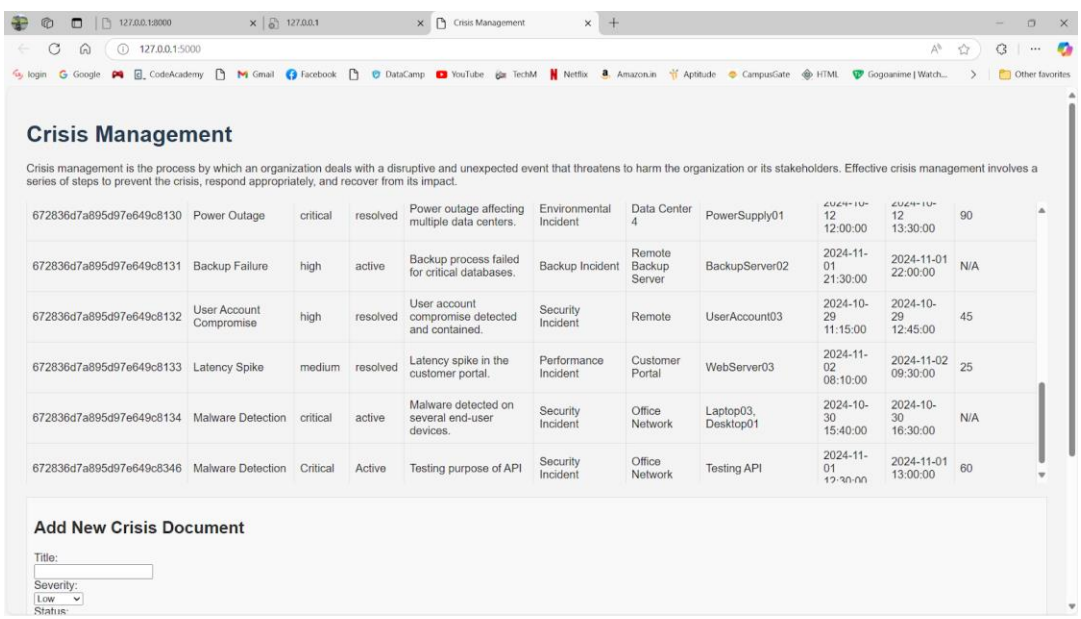
Front-end code setup in VS Code:



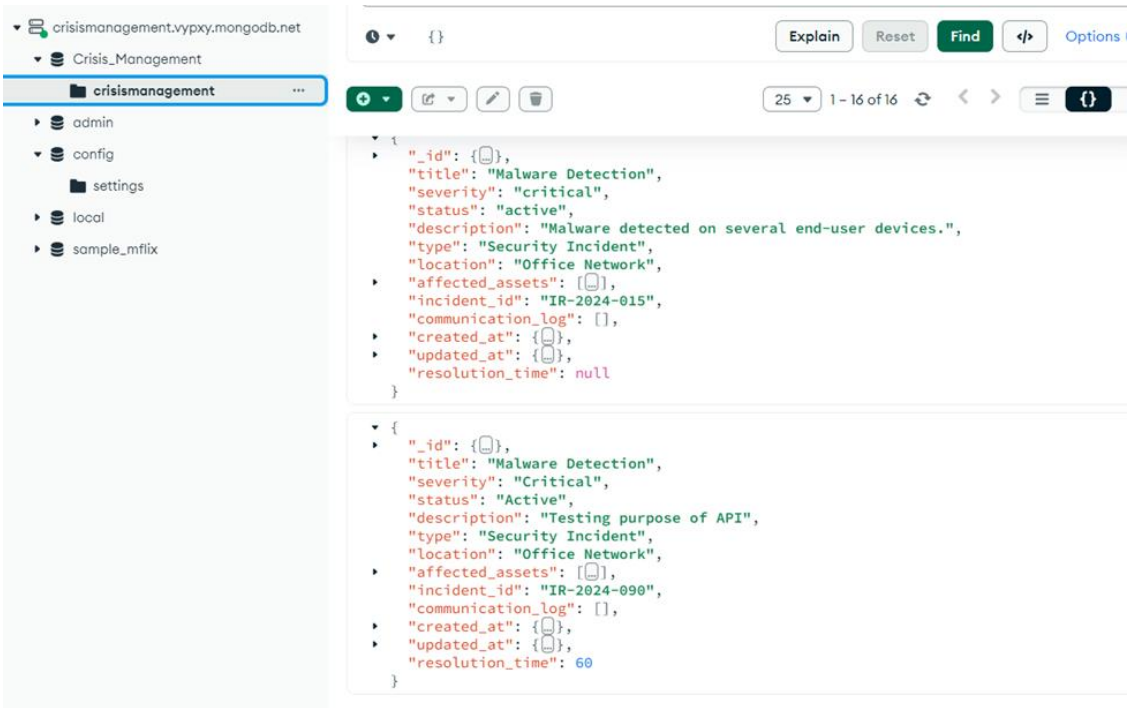
Backend code setup in VS Code:



UI access through localhost http://127.0.0.1:5000



MongoDB database data in Compass app:

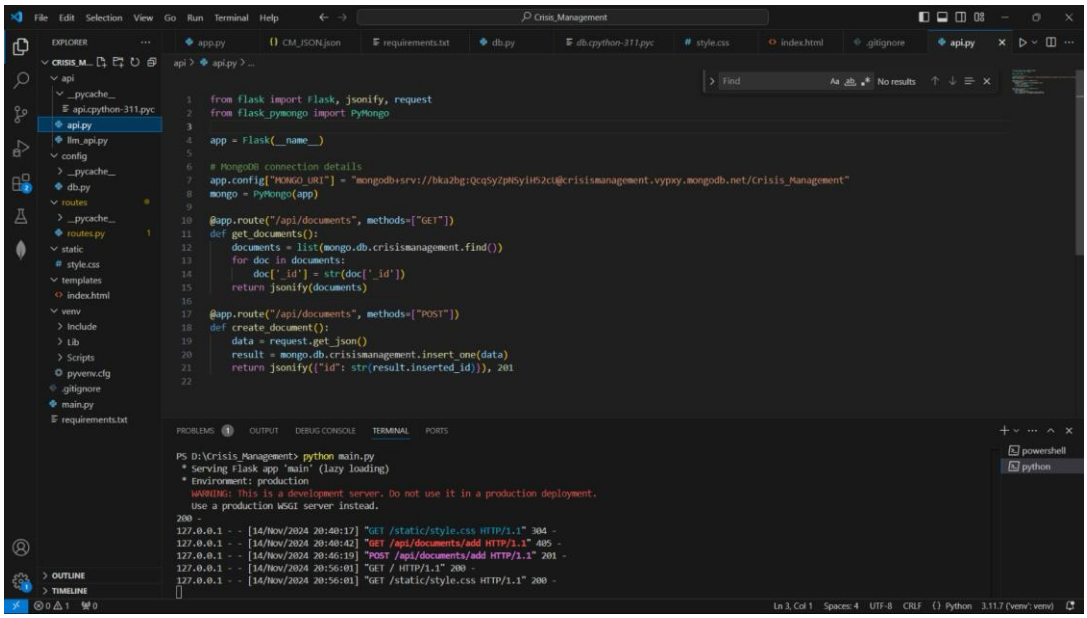


Week 3 Updates

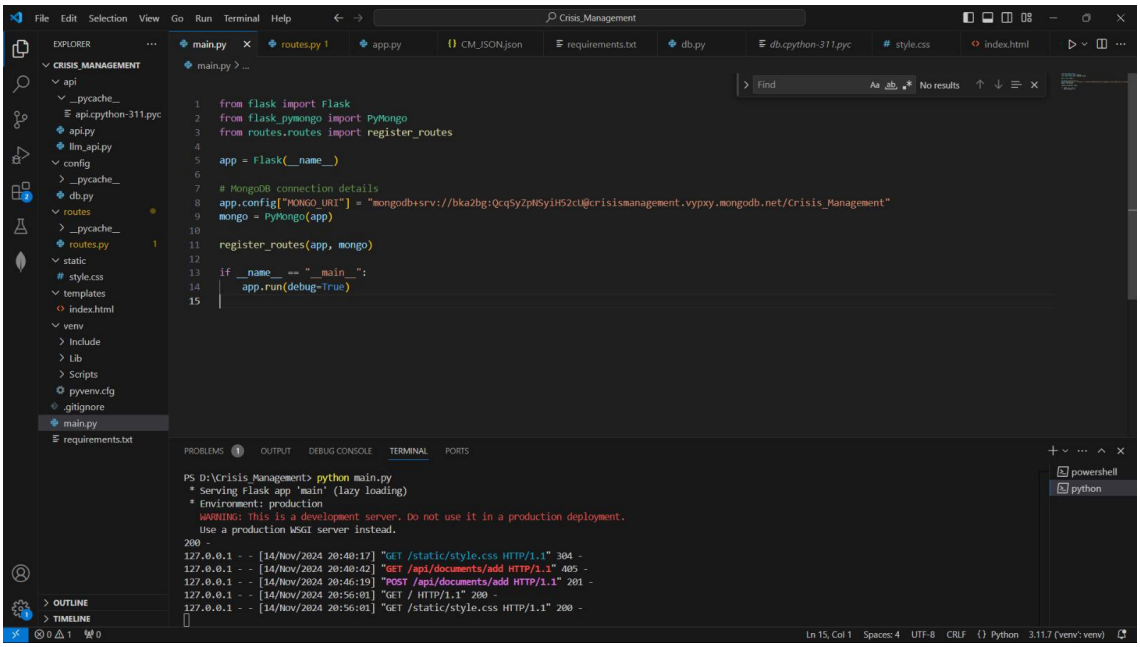
Create APIs for Incident Logging: We have created the API's for performing the CRUD operations on the database. We have tested the API end points using Postman.

API CRUD Operations

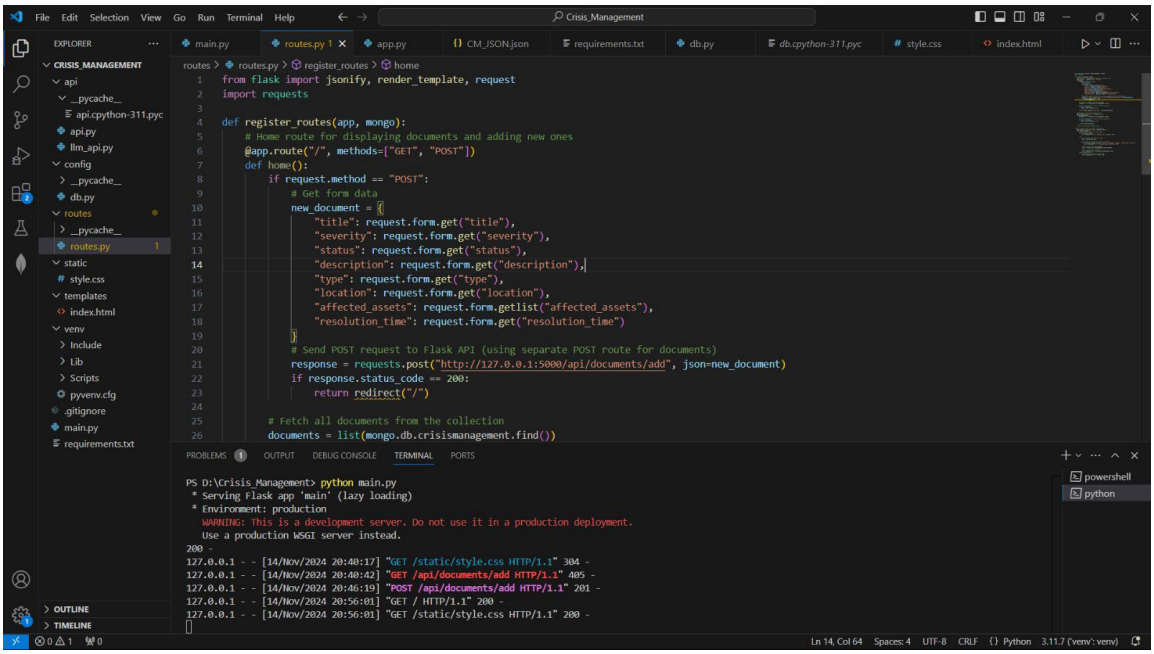
app.py



main.py

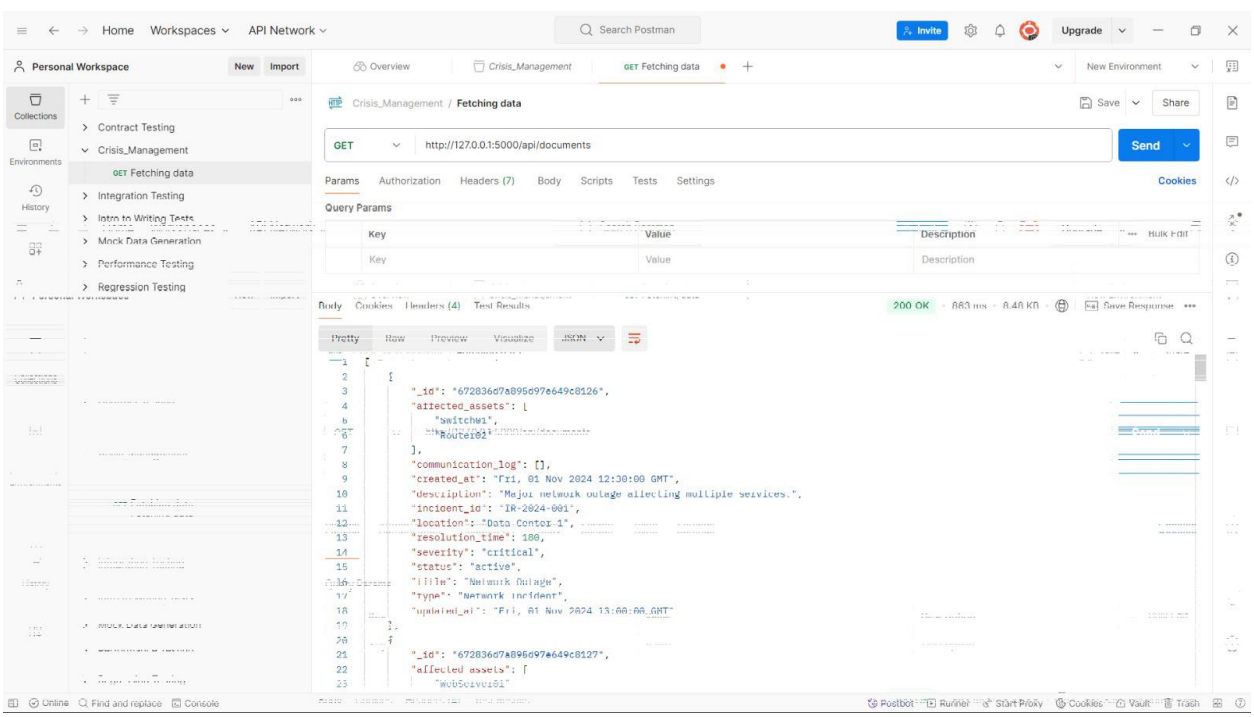


routes.py

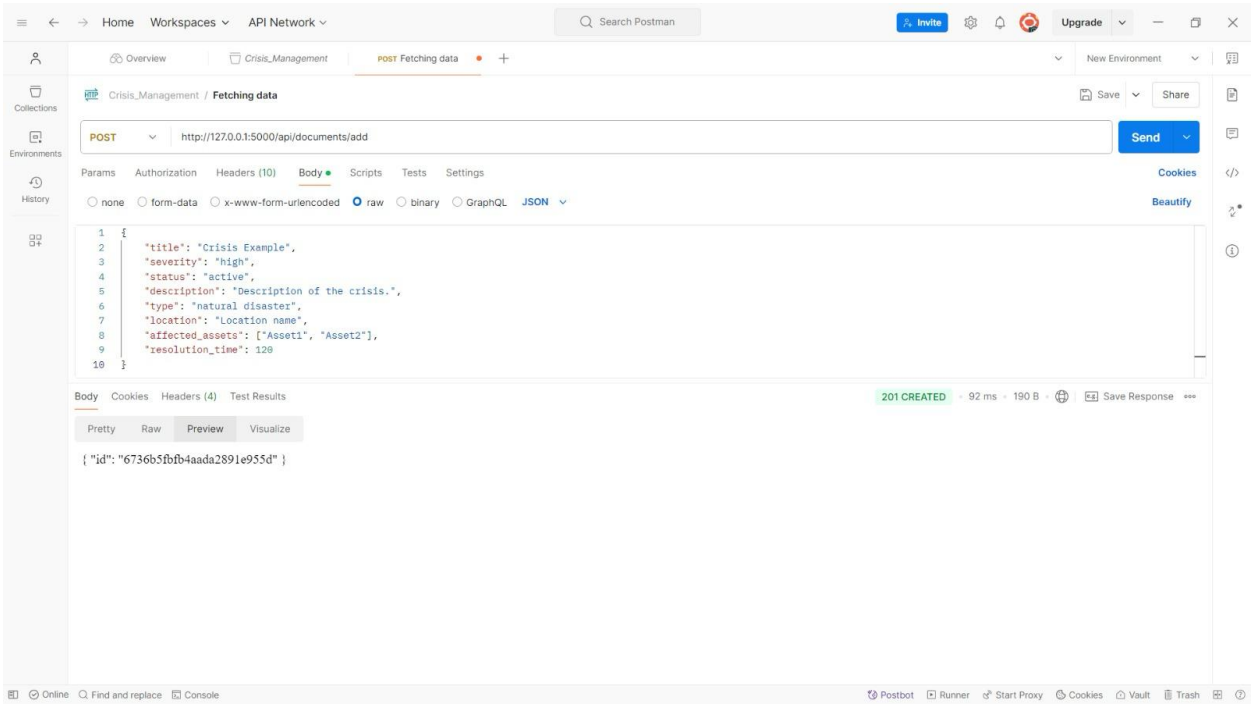


Testing:

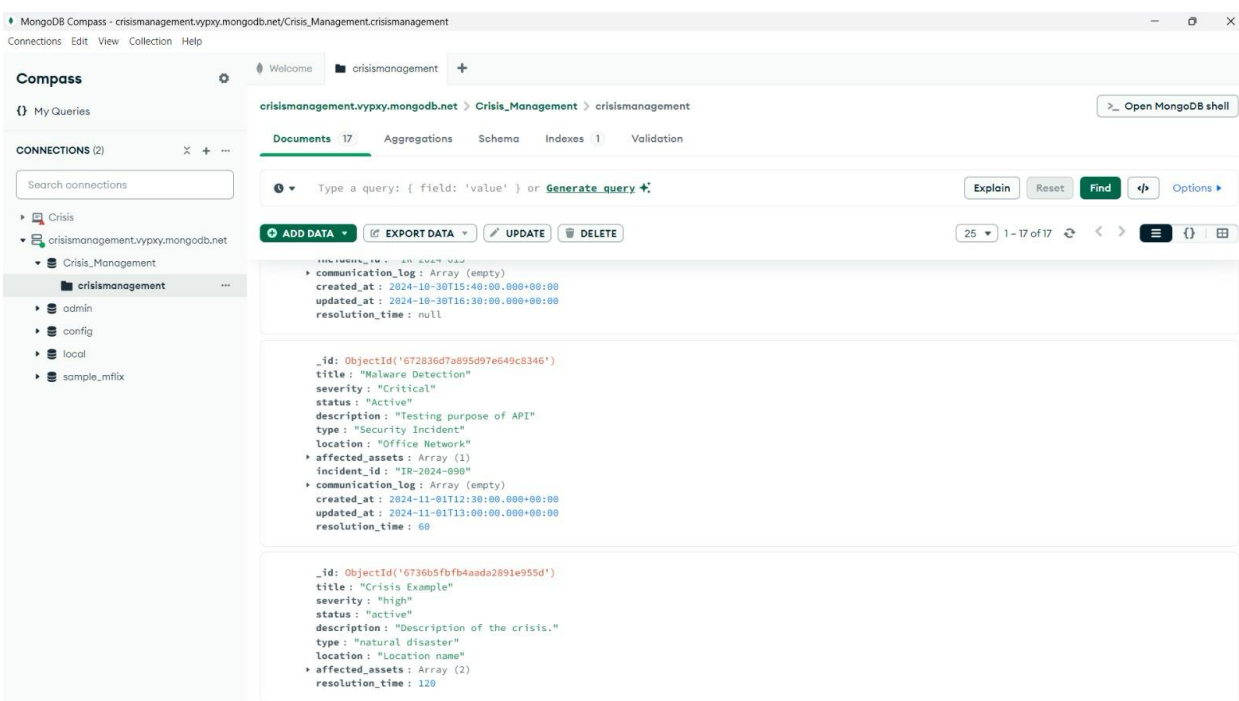
1. Testing the GET API endpoint using Postman



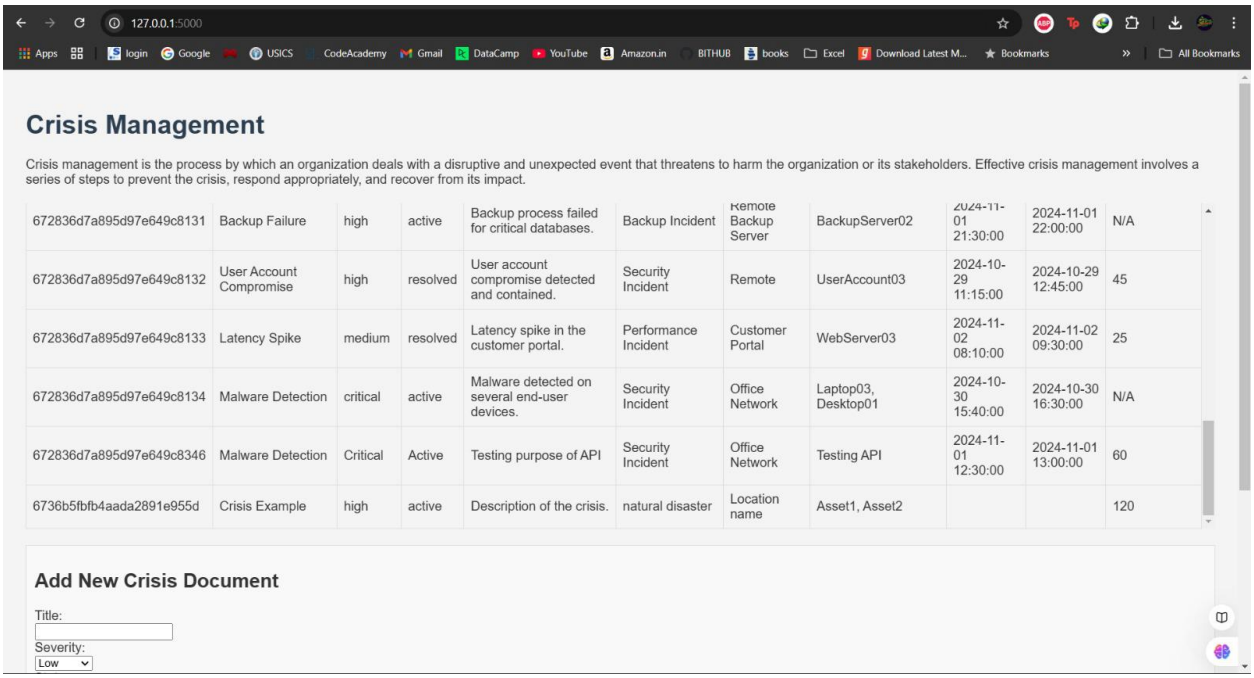
2. Testing the POST API using postman:



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

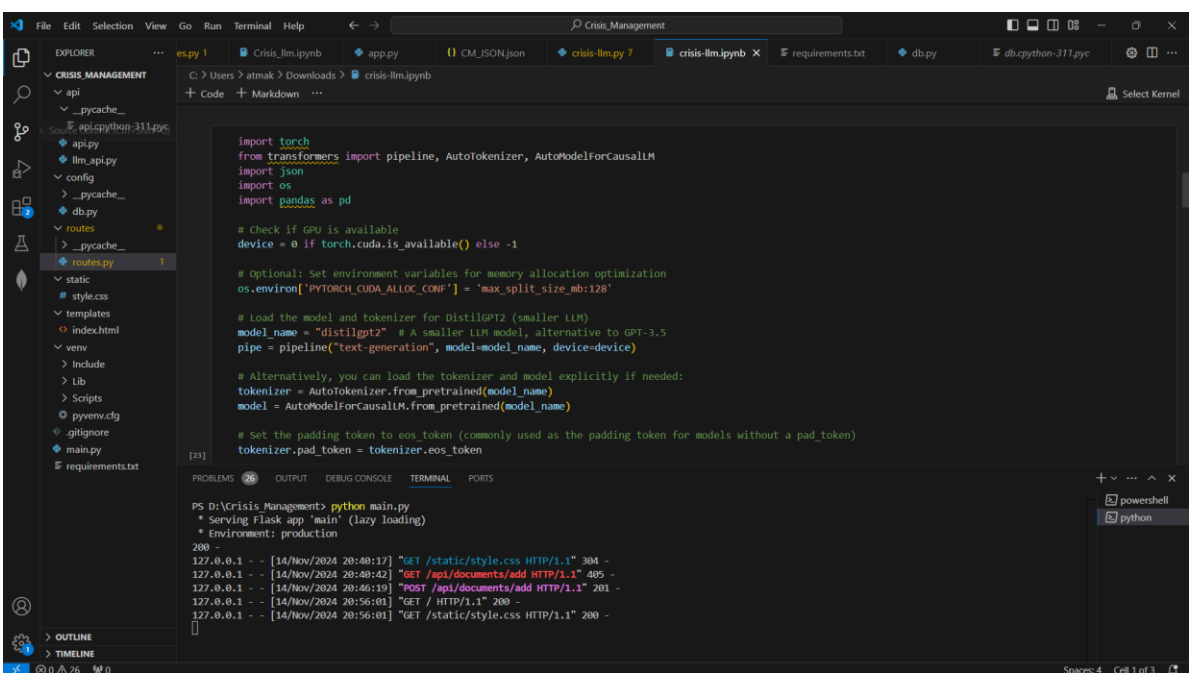


4. Reflection in front end:

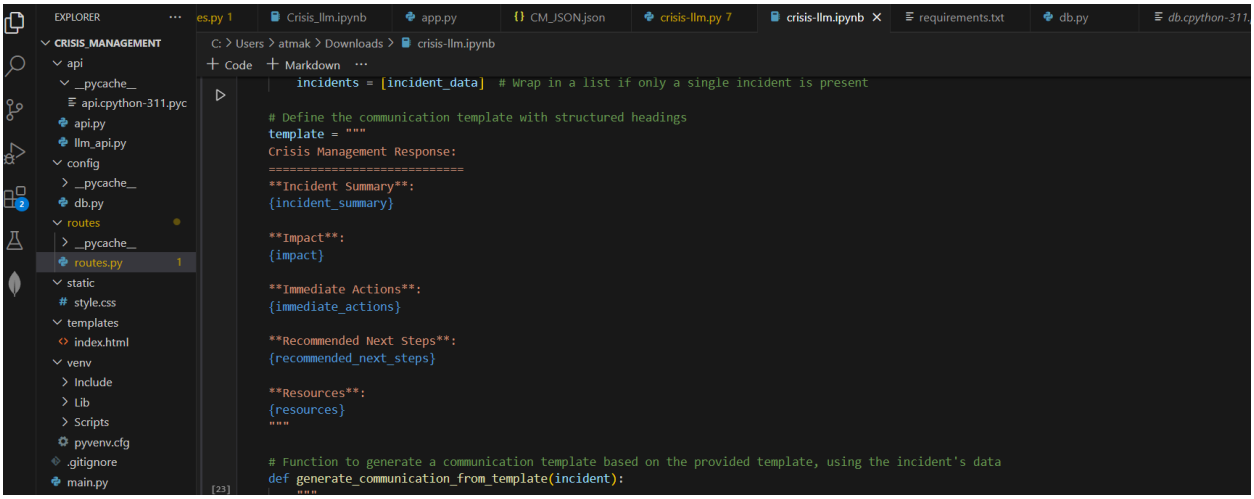


Core functionality:

LLM – distilgpt2

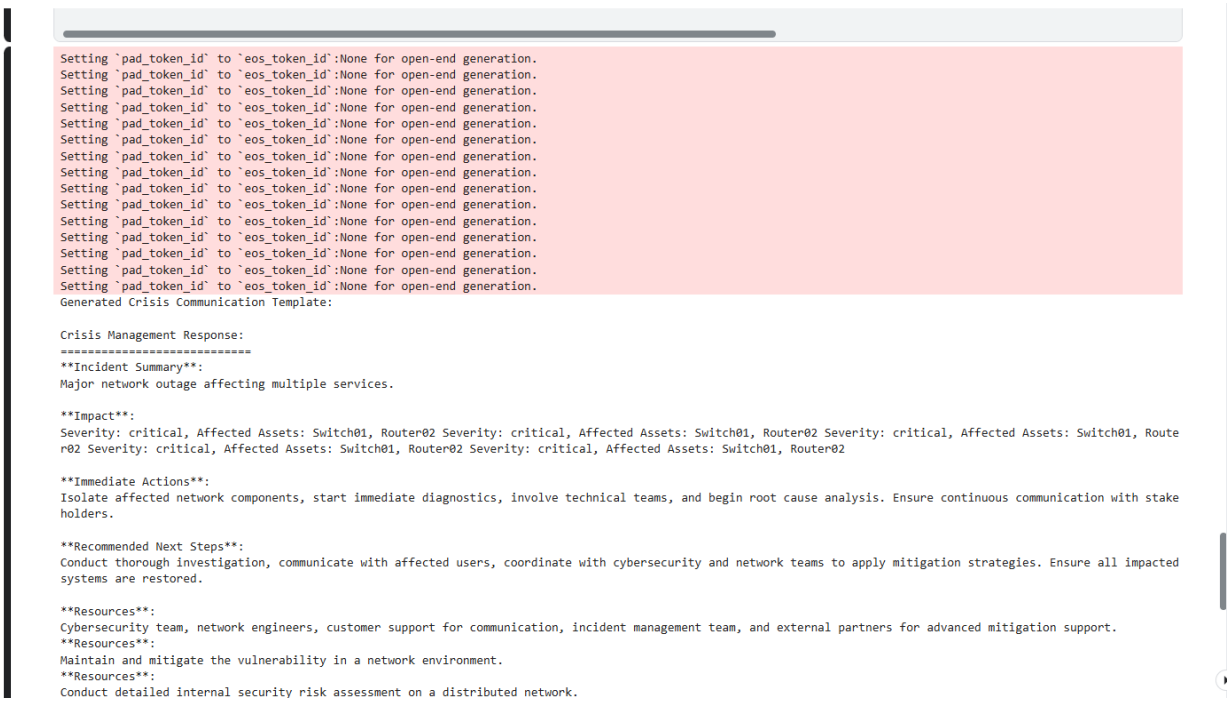


Communication template:



```
    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.
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, Route
r02 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 stake
holders.

**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

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 fro 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.