1. Project Overview

We are building a **Threat Monitoring & Alert Management Backend API** using:

- **Django**
- **Django REST Framework (DRF)**
- **JWT Authentication**

Core Features

- User Authentication (Admin / Analyst)
- Threat Event Ingestion
- Automatic Alert Generation
- Role-based Access Control
- Secure REST APIs

---

2. Tech Stack

- Python 3.11+
- Django 6.x
- Django REST Framework
- Simple JWT
- SQLite (can switch to PostgreSQL)

PYTHON MAKE IT PROJECT FOR STEP BY STEP CYETHACK SOLUTIONS

Step 1: Create Virtual Environment

python -m venv venv

venv\Scripts\activate

Step 2: Install Dependencies

pip install Django

pip install mysqlclient

pip install django djangorestframework djangorestframework-simplejwt

Step 3: Create Project & App

django-admin startproject cyethack_solutions

cd cyethack_solutions

python manage.py startapp core

4. Project Structure

```
cyethack_sloutions/
|— cyethack_sloutions/
|  ├— settings.py
|  ├— urls.py
|  └— ...
|— core/
|  ├— models.py
|  ├— serializers.py
|  ├— views.py
|  ├— permissions.py
|  ├— urls.py
|— manage.py
|--.env
|--dockerfile
```

```
# MySQL configuration in setting.py file
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'threat_monitor_db',
        'USER': 'threat_user',
        'PASSWORD': 'StrongPassword123',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        },
    }
}
```

Implemented Features

- Django project setup with proper app structure
    - Event model with severity levels (Low, Medium, High, Critical)
- Alert model linked to Event using OneToOne relationship
- Automatic alert creation for High and Critical severity events
- JWT authentication using SimpleJWT
- Role-based access control (Admin and Analyst
- Event ingestion API (Admin only)
    - Alert listing API with filtering by severity and status
    - Admin-only alert status update API
    - Pagination enabled using Django REST Framework
    - Optimized database queries using select_related
    - REST-compliant URL structure

Security & Best Practices Followed

• JWT-based stateless authentication

• Role-based permission enforcement

• Serializer-level input validation

• Restricted write operations for analysts

• Environment variable usage for sensitive configuration