Name: Akash Yadav

SAP ID: 590014126

Batch: 12 – CSF

Subject: Software Engineering

Topic: Software Requirement Specifications

Team: BluePrint Solutions

My Project Plan: Building an AI-Driven OSINT Knowledge Graph

1.The Problem:

As someone specializing in cybersecurity, I've realized that the biggest headache in OSINT (Open Source Intelligence) isn't finding data—it's making sense of it. You end up with a folder full of text files, WHOIS records, and social media screenshots, but you can't see the "big picture."

My goal for this project is to build a tool that acts like a digital brain. It will take that messy, unstructured text and automatically map out the hidden connections between people, their emails, and their digital infrastructure using a Knowledge Graph.

2. Working:

I want the interface to be straightforward so that an analyst can focus on the investigation, not the software:

- **The "Dump" Feature:** You should be able to just throw text, JSON, or CSV files into the system without worrying about formatting.
- **Search & Discovery:** Instead of scrolling, you'll just search for a "seed" (like a username) and see a web of everything connected to it.
- **Spotting the Threat:** The tool will visually highlight paths that look like security risks, making it easier to identify potential attack vectors.

3. Functional Requirements:

To make this actually useful, I've designed three core stages:

- **The AI "Reader":** I'm integrating the Gemini API to handle the heavy lifting. It will "read" through my raw data and pull out specific entities like IP addresses or handles. This is much more flexible than using old-school regex patterns.

- **The Graph Memory:** All these pieces of data will live in a Neo4j database. Unlike a standard table, Neo4j treats relationships as first-class citizens.
- **Smart Linking:** The system won't just list data; it will create links. If it finds a common IP between two different usernames, it will draw that line automatically.

4. Non-Functional Requirements:

- **Speed is Key:** I'm aiming for a "live" feel. Even with a large data dump, the AI should return results within about 10–15 seconds.
- **Hardened Security:** Since this is a security project, I'm being extra careful. I'm using `.env` files to make sure my Gemini API keys and database credentials stay off GitHub.
- **Growth:** The tool needs to stay fast even when the graph gets complex (aiming for smooth performance up to 1,000+ nodes).
- **Error Handling:** If the Gemini API hits a limit or the database disconnects, the app should tell me exactly what happened instead of just hanging.

5. Tech Stack Used:

I've chosen **Python** for the backend because it's the gold standard for both AI and security scripts. **Neo4j** is my choice for the database because it's built specifically for this kind of relationship mapping, and **Gemini** gives me the NLP power I need without needing a massive local GPU.