✅ Hands-on Lab: Prompt + Ontology Pattern with SPARQL + Internal API

This lab covers two parts:

✅ PART A: Implement a Prompt + Ontology Pattern Using SPARQL Queries

We will:

- Create an ontology in Protégé
- Store it in a triple store (e.g., Apache Jena Fuseki or Stardog)
- Run SPARQL queries to fetch data
- Dynamically generate a prompt using the query result

🔷 Step 1: Create an Ontology in Protégé

1. **Install Protégé**:
   Download and install from https://protege.stanford.edu
2. **Create a new ontology** (e.g., `HealthcareAssistant.owl`)
3. **Define Classes**:
   - `Disease`, `Drug`, `Treatment`, `Symptom`, `MechanismOfAction`
4. **Define Properties**:
   - `treats (Drug → Disease)`
   - `hasMechanism (Drug → MechanismOfAction)`
   - `hasSymptom (Disease → Symptom)`
5. **Add Individuals**:
   - Drug: `Ibuprofen`
   - Disease: `Arthritis`
   - Mechanism: `COX Inhibitor`
   - Symptom: `Joint Pain`
6. **Save as RDF/XML or TTL** format

🔷 Step 2: Load Ontology into a Triple Store

Use **Apache Jena Fuseki** (easy to run locally):

```
docker run -d -p 3030:3030 --name fuseki stain/jena-fuseki
```

1. Visit `http://localhost:3030`
2. Create a dataset (e.g., `healthcare`)
3. Upload your `.ttl` or `.owl` ontology

🔷 Step 3: Query Data Using SPARQL

Example SPARQL query:

```
PREFIX ex: <http://www.example.org/ontology#>
SELECT ?drug ?mechanism
WHERE {
  ?drug ex:treats ex:Arthritis .
  ?drug ex:hasMechanism ?mechanism .
}
```

Run this from the Fuseki UI or use Python (see next step).

🔷 Step 4: Generate Prompt Dynamically in Python

Install required packages:

```
pip install rdflib requests openai
```

Python code:

```
import requests
import openai

SPARQL_ENDPOINT = "http://localhost:3030/healthcare/sparql"

def query_sparql():
    sparql_query = """
    PREFIX ex: <http://www.example.org/ontology#>
    SELECT ?drug ?mechanism
    WHERE {
      ?drug ex:treats ex:Arthritis .
      ?drug ex:hasMechanism ?mechanism .
    }
    """
    response = requests.post(SPARQL_ENDPOINT, data={'query': sparql_query},
                             headers={'Accept': 'application/sparql-
                             results+json'})
    data = response.json()
    bindings = data['results']['bindings']
    return [(b['drug']['value'].split('#')[-1], b['mechanism']
    ['value'].split('#')[-1]) for b in bindings]

def generate_prompt():
    pairs = query_sparql()
    for drug, mechanism in pairs:
        prompt = f"Explain how {drug} treats arthritis by acting as a
        {mechanism}."
```

```
        return prompt

print(generate_prompt())
```

✅ PART B: Build an Internal API to Expose Graph-Based LLM Capabilities

We will now create a REST API using **FastAPI** to expose this functionality.

🔷 Step 5: Build a FastAPI Server

Install FastAPI:

```
pip install fastapi uvicorn
```

Create `main.py`:

```
from fastapi import FastAPI
from fastapi.responses import JSONResponse
import requests

app = FastAPI()

@app.get("/generate-prompt")
def get_prompt():
    sparql_query = """
    PREFIX ex: <http://www.example.org/ontology#>
    SELECT ?drug ?mechanism
    WHERE {
      ?drug ex:treats ex:Arthritis .
      ?drug ex:hasMechanism ?mechanism .
    }
    """
    response = requests.post("http://localhost:3030/healthcare/sparql",
                             data={"query": sparql_query},
                             headers={"Accept": "application/sparql-
                             results+json"})
    data = response.json()
    pairs = [(b['drug']['value'].split('#')[-1], b['mechanism']
    ['value'].split('#')[-1])
             for b in data['results']['bindings']]
    prompt = f"Explain how {pairs[0][0]} treats arthritis by acting as a
 {pairs[0][1]}."
    return JSONResponse(content={"prompt": prompt})
```

Run the API:

```
uvicorn main:app --reload
```

Test the endpoint:

```
curl http://localhost:8000/generate-prompt
```

✅✅ Bonus: Call the API in ChatGPT or LLM Workflow

Now this API can be used in:

- LangChain tools
- Agentic frameworks (CrewAI / AutoGen)
- Internal chatbots

🎯 What You Learned:

- How to link ontologies with SPARQL-driven prompt construction
- How to serve that logic via a REST API
- How to power GenAI apps with dynamic, explainable knowledge

---