Capstone Lab: Full Implementation of a Graph-Powered GenAI Application

🔷 Step 1: Define the Problem & Use Case

Objective: Choose a real-world problem where semantic understanding, retrieval, and reasoning matter.

Example Use Cases:

- Healthcare: Clinical decision support
- Retail: Personalized shopping assistant
- Legal: Contract clause summarization

Deliverables:

- Problem statement
- Business objective (what value GenAI brings)
- Key actors and workflows (input/output)

🔷 Step 2: Design Ontology & Build Knowledge Graph

Tools: Protégé, TopBraid Composer, RDF/OWL, Neo4j/Stardog

Actions:

1. Define domain-specific ontology (classes, relationships, properties)
2. Save in RDF/OWL format
3. Populate the graph with sample or real data
4. Import RDF into Neo4j/Stardog using a loader or converter

Deliverables:

- Ontology diagram (PDF or image)
- Graph snapshot (nodes, edges)
- Example SPARQL/Cypher queries

🔷 Step 3: Build RAG Pipeline with Graph + Vector Retrieval

Tools: FAISS or Pinecone (for vector search), RDF graph database (Neo4j), OpenAI API

Actions:

1. Embed your knowledge snippets (documents, FAQs) using OpenAI or HuggingFace embeddings
2. Store embeddings in FAISS or Pinecone
3. Build SPARQL/Cypher queries to retrieve entities from your graph
4. Merge both results into a unified context for the prompt

Deliverables:

- Hybrid retriever function (code)
- Example retrieval results (log file or screenshot)
- Combined context passed to LLM

🔷 Step 4: Implement Agent Workflow (LangGraph or CrewAI)

Agent Roles:

- **Planner Agent**: Breaks down user input into subtasks
- **Retriever Agent**: Queries graph and vector DBs
- **Summarizer Agent**: Synthesizes response from retrieved content

Actions:

1. Use LangGraph or CrewAI to create each agent node
2. Define memory (context passing), tools (SPARQL, FAISS client), and transitions
3. Chain agents into a LangGraph workflow

Deliverables:

- Agent definitions (JSON or Python)
- Workflow diagram (Mermaid or Draw.io)
- Demo output from a sample query

🔷 Step 5: Containerize and Deploy to the Cloud

Platforms: AWS Fargate, Azure Container Apps, or GCP Cloud Run

Actions:

1. Write Dockerfile for your app (LLM + graph + vector + agent orchestrator)
2. Push image to ECR/ACR/GCR
3. Deploy with proper secrets for APIs and DBs
4. Set autoscaling, logging (CloudWatch or Prometheus), and endpoint routing

Deliverables:

- Dockerfile + cloud deploy config (YAML/JSON)
- Live endpoint or screencast of deployed app
- Monitoring dashboard screenshot

🔷 Step 6: Document the Architecture

Tools: Mermaid, Draw.io, Markdown, Google Docs

Sections to Include:

- Business problem and user need
- Ontology structure and graph schema
- RAG pipeline with code snippets
- Agent flow with reasoning logic
- Deployment diagram and endpoints

Deliverables:

- Capstone documentation PDF or GitHub repo
- Explainer video or slides (optional)

🔷 Step 7: Stakeholder Presentation

Actions:

1. Prepare a 10-15 minute walkthrough
2. Cover what problem you solved, how it works, and what value it provides
3. Show live demo or screencast
4. Invite questions, feedback, and reflection

Deliverables:

- Slide deck (PPT or PDF)
- Recording or live presentation
- Reviewer feedback summary

🎓 Final Output

- Working, deployed GenAI application with hybrid retrieval and agents
- Knowledge graph and ontology implementation
- Complete documentation and demo materials

> Congratulations—you've built a production-grade, graph-powered Generative AI system that integrates symbolic reasoning, vector search, and cloud-native deployment.