AllyIn Compass – Full Enterprise AI Agent

(15-Day Plan for Undergraduate Developers)

Goal: Build a working prototype of AllyIn Compass, a smart enterprise assistant that can search through company documents, structured data like spreadsheets, and even graphs of relationships to answer complex questions. It uses advanced tools like Retrieval-Augmented Generation (RAG), agents that can choose which tool to use, and logging to track everything. This version is written for beginner and intermediate undergraduate-level developers with clear steps, daily tasks, goals, and code locations.

Day 0: Preparation

What to do:

- Install Docker Desktop
- Install Python 3.10+
- Create a GitHub account and fork the starter repo (or clone from your team leader)
- Tools to install: VS Code, Postman (for testing APIs)

Deliverables:

- `~/allyin-compass/` directory on your machine
- Docker working (`docker run hello-world` test passed)

Day 1: Project Setup + Structured Data

Tasks:

- Set up Python virtual environment: `python -m venv venv && source venv/bin/activate`
- Install dependencies: `pip install -r requirements.txt`
- Create a `data/structured/` folder and add sample `.csv` files:
  - `customers.csv`, `orders.csv`, `emissions.csv`
- Use `pandas` to load them into DuckDB:

```
Unset
import duckdb
import pandas as pd
customers = pd.read_csv('data/structured/customers.csv')
duckdb.sql("CREATE TABLE customers AS SELECT * FROM customers")
```

- Explore the table using SQL:

```
Unset
duckdb.sql("SELECT * FROM customers LIMIT 10").df()
```

Deliverables:

- `src/ingest/structured_loader.py`
- Verified tables in DuckDB

Day 2: Ingest Unstructured Data (PDF + Email)

Tasks:

- Create `data/unstructured/` folder and collect 3 PDFs + 3 .eml files
- Use PyMuPDF to extract text from PDFs:

```
Unset
import fitz
with fitz.open("file.pdf") as doc:
    text = "".join([page.get_text() for page in doc])
```

- Use the Python `email` package to extract subject and body from emails
- Save output as JSONL (one document per line)

Deliverables:

- `src/ingest/document_parser.py`

- `data/unstructured/parsed.jsonl`

Day 3: Embedding Text and Creating a Vector DB

Tasks:

- Use the sentence-transformers library to create embeddings:

```
Unset
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = model.encode(["example sentence 1", "another one"])
```

- Store them into Qdrant:

```
Unset
from qdrant_client import QdrantClient
client = QdrantClient("localhost", port=6333)
client.upload_collection(name="docs", vectors=embeddings,
payloads=[{"text": "..."}])
```

Deliverables:

- `src/ingest/embedder.py`
- 100+ embedded chunks in Qdrant

Day 4: Set Up SQL, Vector, and Graph Retrieval

Tasks:

- SQL: Build simple queries using `langchain.chains.SQLDatabaseChain`
- Vector: Use Qdrant to retrieve top 3 similar chunks
- Graph: Install Neo4j Desktop or Sandbox; create 10 entities and their connections
- Sample Cypher:

```
Unset
CREATE (:Facility {name: "Plant A"})-[:EXCEEDS]->(:Regulation
{type: "CO2 Limit"})
```

Deliverables:

- `src/retrievers/sql_retriever.py`, `vector_retriever.py`, `graph_retriever.py`

Day 5: Build Your First Agent

Tasks:

- Learn LangChain Tool usage
- Define tools for vector, SQL, and graph search
- Use `LangChain AgentExecutor` to chain these tools:

```
Unset
from langchain.agents import initialize_agent
agent = initialize_agent([vector_tool, sql_tool], llm,
verbose=True)
```

- Log every call with time and tool used

Deliverables:

- `agents/multi_tool_agent.py`
- Working tool-chaining example

Day 6: Retrieval-Augmented Generation (RAG)

Tasks:

- Take output from Qdrant (vector search)
- Build prompt:

```
Unset
context = "".join([doc['text'] for doc in top_3_docs])
prompt = f"Use this information to answer the
question:\n{context}\nQuestion: {user_query}"
```

- Call OpenAI or another LLM:

```
Unset
from openai import ChatCompletion
answer = ChatCompletion.create(prompt=prompt, model="gpt-4")
```

Deliverables:

- `tools/rag_tool.py`
- Answers with citations printed

Day 7: UI Development with Streamlit

Tasks:

- Install Streamlit: `pip install streamlit`
- Create `ui/app.py`
- Components:
  - Text input box
  - Dropdown for domain (finance, biotech, energy)
  - Output window for answer
  - Sidebar: filters by confidence, date, source

Deliverables:

- `ui/app.py` running: `streamlit run ui/app.py`

Day 8: Add Feedback Loop

Tasks:

- Create thumbs up/down buttons for feedback
- Record data:

```
Unset
log = {"query": user_query, "answer": output, "rating": 1}
with open("feedback_log.jsonl", "a") as f: json.dump(log, f)
```

Deliverables:

- `feedback/logger.py`
- `feedback_log.jsonl`

Day 9: Fine-tuning Simulation

Tasks:

- Collect 10+ positive feedback examples
- Format as prompt-completion pairs
- Use HuggingFace's PEFT + LoRA for tuning

Deliverables:

- `notebooks/simulate_finetuning.ipynb`
- `models/lora_adapter/`

Day 10: PII + Compliance Guardrails

Tasks:

- Use regex to detect email, SSN, phone:

```
Unset
import re
re.findall(r'\b\d{3}-\d{2}-\d{4}\b', text)  # SSN format
```

- Flag content containing terms like "restatement", "earnings risk"

Deliverables:

- `security/pii_filter.py`, `compliance_tagger.py`

Day 11: Observability Dashboard

Tasks:

- Use Streamlit charts to track:
  - Number of queries/day
  - Tool usage frequency
  - Avg response time

Deliverables:

- `dashboards/metrics.py`
- Live dashboard in sidebar of `ui/app.py`

Day 12: Test Use Cases

Tasks:

- Run queries like:
  - Finance: "Top flagged clients in last quarter"
  - Biotech: "Adverse outcomes involving molecule X"
  - Energy: "CO2 violations since Q1 near San Jose"
- Verify:
  - Output matches sources
  - Trace shows correct tool flow

Deliverables:

- `examples/use_case_tests.json`
- Screenshots for demo deck

Day 13: Finalize UI + Add Domain Filters

Tasks:

- Add domain dropdown: Finance, Biotech, Energy

- Highlight source text in answer window

Deliverables:

- Fully styled `ui/app.py`
- Final screenshots saved in `demo_assets/`

Day 14: Record Demo + Build Slide Deck

Tasks:

- Record 2–3 min demo using Loom or Zoom
- Slides:
    - Title + team
    - Architecture diagram
    - Sample queries
    - Screenshot of each UI tab

Deliverables:

- `demo_assets/demo.mp4`
- `demo_assets/slide_deck.pdf`

Day 15: Final Polish + GitHub Push

Tasks:

- Cleanup code and add docstrings
- Push to GitHub with README:
    - Overview
    - Setup instructions
    - How to run a query
- Add license and credits

Deliverables:

- GitHub repo ready to share
- Demo link sent to stakeholders