Project Plan: LLM Data Integration MVPs

This document outlines the project plan, including implementation details, a schedule, and member assignments for the three LLM Data Integration MVP projects. The project is scheduled for 8 weeks.

---

1. Proposed Implementation Details

Task 1: Implement Ontology System for Unstructured Data to Graph Conversion

Goal: Extract structured information from unstructured text and convert it into a knowledge graph.

Member Assignments:

- **Abhijith (Lead - Extraction Engine):** Core logic of entity and relationship extraction using LLMs and NLP libraries; ontology schema definition.
- **Anne (Lead - Data Pipeline):** Building the data ingestion and preprocessing pipeline to prepare text for extraction.

Implementation Strategy:

- **Data Ingestion & Preprocessing (Anne):**
  A Python-based pipeline will be developed to handle various document formats.
  - Use `PyMuPDF` for PDFs and `python-docx` for Word documents.
  - Clean text (remove special characters, standardize whitespace), segment into sentences using `spaCy` or `NLTK`.
- **Entity & Relationship Extraction (Abhijith):**
  - Use pre-trained models from `spaCy` or Hugging Face's `transformers` library for Named Entity Recognition (NER).
  - For complex relationships, engineer prompts for an LLM (e.g., via the `google-generativeai` library) to return structured JSON output, which is then parsed.
- **Ontology & Graph Formation (Abhijith):**
  - Map extracted entities and relationships to a predefined ontology.
  - Convert structured data into nodes and edges, inserting into a Neo4j graph database using the `neo4j-driver`.
- **Configuration (Abhijith & Anne):**
  - A YAML file defines the ontology (entity types, relationship types) and extraction rules for easy modification without changing core code.

Task 2: Implement Ontology System for Structured Data to Graph Conversion

Goal: Connect to structured data sources (databases, APIs) and map their schemas to a knowledge graph.

Member Assignments:

- **Sandra (Lead - Schema Mapping & Architecture):** Overall architecture and schema-to-ontology translation.
- **Devan (Lead - Data Connectors & Performance):** Building high-performance connectors to data sources.

Implementation Strategy:

- **Data Connectors (Devan):**
  - Build connectors in Python.
  - Use `SQLAlchemy` for SQL database compatibility and the `requests` library for REST APIs.
  - Implement schema introspection to discover table structures and API schemas.
- **Schema Mapping (Sandra):**
  - Design a mapping engine to read rules from a YAML file.
  - The YAML file specifies how tables, columns, and API fields map to nodes, properties, and relationships in the knowledge graph.
- **Data Transformation (Sandra & Devan):**
  - Transform data from the source into graph format based on mapping rules.
  - Optimize transformation—potentially using asynchronous API calls with `aiohttp`.
- **Graph Integration (Sandra):**
  - Load transformed data into the Graphiti knowledge graph, ensuring relationships (like foreign keys) are represented as graph edges.

Task 3: Implement Graphiti-Based Organizational Chart System

Goal: Manage an organizational chart within a knowledge graph, supporting data import from spreadsheets and Lucidchart.

Member Assignments:

- **Alwin (Lead - Graph Model & Core Operations):** Graph data model and high-performance data import logic.
- **Jonas (Lead - API Integration & Data Integrity):** Lucidchart API integration and data validation.

Implementation Strategy:

- **Graph Data Model (Alwin):**
  - Design the organizational chart schema in the graph database with "Employee" and "Department" nodes, and "REPORTS_TO" and "MEMBER_OF" relationships.
- **CRUD Operations (Alwin):**

  - Implement core functions (Create, Read, Update, Delete) for managing employees and departments directly in the graph using Python.

- **Spreadsheet Import (Alwin):**

  - Build spreadsheet import functionality in Python.
  - Use the `openpyxl` library for Excel files and the built-in `csv` module or `pandas` for CSVs.
  - Handle bulk updates to the organizational structure efficiently.

- **Lucidchart Integration (Jonas):**

  - Integrate with the Lucidchart API using Python's `requests` library.
  - Handle OAuth for authentication, fetch chart data, and parse it into the graph structure.

- **Data Integrity (Jonas):**

  - Develop a validation engine to check for duplicates and structural conflicts during data import, with logic to resolve these issues.

2. Schedule (8 Weeks)

Week(s)	Key Activities & Milestones
1-2	Foundation & Setup: Set up development environments, graph DBs, define YAML configs, initial scaffolding
3-4	Core Component Development: Data ingestion (TXT, PDF), basic NER, SQL connector, schema mapping, CRUD, CSV import
5-6	Feature Expansion & Integration: More file formats, refine relationship extraction, REST API connector, graph DB integration, Lucidchart API, validation engine
7-8	Testing, Refinement & Documentation: End-to-end testing, performance optimization, documentation, MVP demo prep

3. Member Assignments Summary

Task	Team Member	Role	Key Responsibilities
Unstructured Data to Graph	Abhijith	Lead - Extraction Engine	LLM prompt engineering, entity/relationship extraction, ontology
	Anne	Lead - Data Pipeline	Data ingestion, text preprocessing, pipeline integration
Structured Data to Graph	Sandra	Lead - Schema Mapping & Arch.	Architecture, schema/ontology translation engine
	Devan	Lead - Data Connectors & Perf.	High-performance connectors for DBs and APIs

Task	Team Member	Role	Key Responsibilities
Org Chart System	Alwin	Lead - Graph Model & Operations	Graph model design, CRUD, high-performance spreadsheet import
	Jonas	Lead - API Integration & Data Integrity	Lucidchart API integration, validation, conflict resolution