Project Architecture and Module Overview

Overview

This project is structured around two core backend modules, supported by a unified frontend. The goal is to:

1. **Organize and tag files in a Google Drive** using AI and maintain an up-to-date **knowledge graph**.
2. Use that structured information to power an **AI agent** that can answer user queries by retrieving relevant files intelligently.

The architecture is designed to be clean, modular, and collaborative — optimized for a 5-member development team.

Backend Folder Structure (Simplified)

```
backend/
├── main.py                 # Entry point for backend server
├── .env                   # Environment variables
├── requirements.txt       # Python dependencies
│
├── config/                # Configuration files (settings, credentials)
│
├── api/                   # API route handlers for organizing and querying
│
├── modules/               # Core logic modules for organization and AI agent
│   ├── organizer/         # Module 1: Google Drive organization + Knowledge
Graph
│   ├── ai_agent/          # Module 2: AI agent with RAG-based querying
│   ├── vector_store/      # Embedding and semantic search components
│   ├── knowledge_graph/   # Shared knowledge graph utilities and storage
│
├── shared/                # Common utilities, schemas, prompts
│
├── tests/                 # Unit and integration tests
```

Module Descriptions

Module 1: Drive Organizer & Knowledge Graph (Folder: `organizer/`)

This module manages file categorization and organization within Google Drive, and also constructs and updates the knowledge graph.

Responsibilities:

- Access Google Drive using APIs
- Categorize files based on content using LLMs
- Organize files into folders based on tags and metadata
- Maintain a knowledge graph that:
    - Represents topics, subtopics, and their relationships
    - Links to files and summaries
- Update the graph upon addition, deletion, or modification of files

Knowledge Graph Role:

- Stored as a JSON or in a graph database
- Used as the first-pass filter for narrowing down documents during query processing

---

Module 2: AI Agent & Query Answering (Folder: `ai_agent/`)

This module powers the natural language query functionality. It retrieves relevant information using the organized drive and knowledge graph.

Responsibilities:

- Accept user queries from frontend/API
- Use the knowledge graph to identify the relevant topics or file clusters
- Retrieve document chunks using semantic similarity (via vector store)
- Pass the chunks + user query to an LLM (Gemini, GPT, etc.)
- Return an answer with contextual relevance and optional source links

---

Vector Store (Folder: vector_store/)

Handles:

- Chunking document text
- Generating embeddings
- Indexing and retrieval using **ChromaDB via LangChain**

This project is configured to use **ChromaDB only**, as a scalable and free vector database compatible with cloud deployment.

Why ChromaDB?

- Open-source and fast

- Supports persistent storage for local and hosted environments
- Allows metadata filtering and document management
- Easy to integrate via LangChain's unified vector store API

The system assumes **ChromaDB is the exclusive backend** for all vector store operations. No fallback or switch logic is included.

Knowledge Graph (Folder: `knowledge_graph/`)

Shared logic for managing and traversing the knowledge graph. Used by both modules:

- Module 1 writes to it
- Module 2 reads from it

Can be expanded to support more advanced reasoning or visualization later.

Shared Utilities (Folder: `shared/`)

Contains:

- Reusable helper functions
- Schema definitions (e.g., for request/response models)
- Prompt templates for the AI agent

Testing (Folder: `tests/`)

All modules will have corresponding unit/integration tests, organized by function.

Member Responsibilities

Member	Responsibility Area	Primary Folders
Member 1 & 5	Google Drive integration & file organization	`organizer/`
Member 2	Embedding generation and vector search	`vector_store/`

Member	Responsibility Area	Primary Folders
Member 3	AI agent, query logic, LLM interaction	`ai_agent/`, `api/query`
Member 4	Frontend interface (outside backend scope)	`frontend/`
Final check	Integration, config, testing, glue code	`main.py`, `shared/`, `tests/`