🧠 Take-Home Task: Agentic Content Intelligence CLI

🎯 Goal

Design a CLI-based, agent-driven content intelligence system using **LangChain**, **Firecrawl.dev**, and **GenAI**. The system should accept natural language prompts from a user and dynamically route them to the appropriate agents that perform tasks like crawling, summarizing, classifying, and storing content in a vector database for future Q&A.

You **must use LangChain** as your agentic framework, and **TypeScript** (Node.js) as the programming language. You are free to use any LLM provider (OpenAI, Anthropic, etc.), and we also provide a **free OpenAI API key**.

💻 What You'll Build

A CLI tool where a user can write prompts like:

> "Summarize this article and give me the key takeaways: https://example.com/blog/my-post"

> "Crawl this domain, build a knowledge base, and prepare it for Q&A."

> "Summarise the content from this URL for me."

> "What does the Posthog article say about GDPR compliance?" (assumes article is already crawled)

Based on the user input, your system should:

- Parse intent from natural language
- Dispatch the task to the correct agent(s)

- Orchestrate multi-step flows where needed

📦 Features & Requirements

🧠 Agentic CLI Interface

- Accept prompts via command line (natural language).
- Parse user requests and invoke the appropriate agents:
  - **Scraping Agent:** Uses Firecrawl to crawl single or multiple URLs.
  - **Summary Agent:** Produces a concise 3–4 sentence summary of content.
  - **Key Takeaways Agent:** Extracts 3 core takeaways from each content item.
  - **Knowledge Base Agent:**
    - Stores content and metadata in a vector database (FAISS, Pinecone, or Chroma).
    - Accepts user questions and queries the stored vector database for answers.
- Implement orchestration logic where needed (e.g., chunk → embed).

🧰 Technologies

- **LangChain.js**: Required agent framework
- **Firecrawl.dev**: For content scraping
- **LLM**: Any provider (OpenAI key included below)
- **Vector DB**: FAISS, Chroma, or Pinecone
- **Language:** TypeScript (Node.js only)

🧪 Evaluation Methodology

In your `README.md`, describe how you would evaluate your solution. Consider including:

- Accuracy and relevance of agent responses
- CLI usability (input clarity, helpful feedback)
- Agent orchestration and task routing robustness
- Embedding and search quality for vector-based retrieval

✅ Evaluation Criteria

Category	Description
Prompt Engineering	Prompts are optimized for specific tasks and demonstrate clear design reasoning.
Agent Workflow & Orchestration	Uses LangChain agents effectively, with intelligent dispatching and modular orchestration logic.
CLI Usability	User-facing CLI supports intuitive, flexible inputs and provides meaningful feedback.
Content Understanding & Metadata Extraction	Summaries, takeaways, and classifications are relevant, accurate, and well-structured.
Knowledge Base Functionality	Embeddings are stored efficiently and used effectively to answer user queries.
Code Quality & Architecture	Modular, maintainable, and idiomatic TypeScript code with appropriate separation of concerns.
Documentation & Evaluation Strategy	Clear explanation of design choices, setup instructions, and evaluation methodology.
GenAI Usage Transparency	README shows how LLMs and tools like Cursor/Copilot were used during development.

📄 Deliverables

Submit a **ZIP file** containing:

- Full source code
- `README.md` with:
  - Setup and execution instructions
  - Agent orchestration and dispatching logic

- Example prompts and outputs
- Prompt engineering strategy
- Evaluation methodology
- Documentation on GenAI-assisted development

🔐 OpenAI API Key

You may use your own LLM provider, but for convenience, here is a pre-provided **OpenAI API key** for free usage during the test:

```
sk-proj-OlpMidoXcacWqVlj9bzoah-dUsk1UJbyUahgfV3rb9oks9v1bnHHtWsaOx3_32GV1axl1×5fmgT3BlbkFJytRa9ar3id4Xd9mPyJFs1-ipbNLrUUxyacuG_DApYi2VuZu3N7MzcdP7_8J1VWZViu90Z3yX8A
```