AI Exam Creator Agent

Github Repository

1. Introduction

This project implements an AI agent system designed to generate educational exams based on provided study materials. The system uses a multi-agent architecture to ensure that generated exams are factually accurate, correctly formatted in Markdown, and automatically saved to the local file system.

The core functionality relies on **Retrieval Augmented Generation (RAG)** to ground questions in a specific dataset of notes, preventing hallucinations and ensuring relevance to the curriculum.

2. Framework and Architecture

The system is built using the **Microsoft AutoGen** framework (`autogen_agentchat`). It employs a `RoundRobinGroupChat` architecture, allowing a team of agents to collaborate sequentially until a task is done and termination condition is met.

3. Agent Design and Roles

The system consists of five specialized agents working in a team:

1. Planner

- **Role**: Coordinator.
- **Responsibility**: Orchestrates the workflow. It directs the `ExamCreator` to generate content, the `MarkdownVerifier` to check syntax, and the `Reviewer` to validate accuracy. It ensures the process moves efficiently from creation to saving.

2. ExamCreator

- **Role**: Content Generator.
- **Responsibility**: Generates the exam content (questions and answers).
- **Capabilities**: Uses **RAG** (ChromaDB) to retrieve relevant context from the `docs/` folder. It is instructed to create multiple-choice and open-ended questions based *only* on the provided context.

3. MarkdownVerifier

- **Role**: Quality Assurance (Format).
- **Responsibility**: Ensures the generated output is valid Markdown.
- **Tools**: Uses the `verify_markdown_content` tool.

4. Reviewer

- **Role**: Quality Assurance (Content).
- **Responsibility**: Reviews the exam for correctness and quality. It provides feedback for improvements or issues the "Exam is approved" signal.

5. Saver

- **Role**: File I/O.
- **Responsibility**: Listens for the approval signal and saves the final exam to a file.
- **Tools**: Uses the `save_file` tool.

4. Tools and Data Sources

Data Source (RAG)

The agent solves the task of exam creation by retrieving information from a local knowledge base.

- **Source**: Markdown documents in the `docs/` directory (covering topics like History, Literature).
- **Indexing**: Documents are processed using **chonkie** for chunking and stored in **ChromaDB**.
- **Retrieval**: The `ExamCreator` agent has access to `chroma_memory` to query this database during generation.

Custom Tools

The agents are equipped with Python functions to perform actions:

1. `verify_markdown_content`: Scans the generated text string using `pymarkdown` to identify syntax errors.
2. `save_file`: Writes the approved content to the `exams/` directory with a timestamped filename (e.g., `exam_20251204_201944.md`).

5. Technical Implementation Details (RAG)

The RAG pattern encompasses two distinct phases:

1. Indexing: Loading documents, chunking them, and storing them in a vector database
2. Retrieval: Finding and using relevant chunks during conversation runtime

Indexing

Using **chonkie** to index markdown documents

```
docs: List[MarkdownDocument] = (
    Pipeline()
    .fetch_from("file", dir="./docs", ext=[".md"])
    .process_with("markdown")
    .chunk_with("semantic", threshold=0.8, chunk_size=1024, similarity_window=3)
    .run()
)
```

Adding documents to memory via **AutoGen**

```
async def add_docs_to_memory(memory: Memory, docs: List[MarkdownDocument]) ->
None:
```

```
    for doc in docs:
        for i, chunk in enumerate(doc.chunks):
            await memory.add(
                MemoryContent(
                    content=chunk.text,
                    mime_type=MemoryMimeType.MARKDOWN,
                    metadata={"chunkId": chunk.id,
                    "chunkIndex": i,
                    "tokenCount": chunk.token_count,
                    "sourceDocumentId": doc.id},
                )
            )
```

Retrieval

During the exam creation, the `ExamCreator` agent retrieves relevant context from the ChromaDB vector store using the `chroma_memory` instance.

```
exam_creator = ExamCreator(
    name="ExamCreator",
    tools=[chroma_memory],
    system_prompt=EXAM_CREATOR_SYSTEM_PROMPT,
)
```

ChromaDB Vector Memory configuration:

```
chroma_memory = ChromaDBVectorMemory(
    config=PersistentChromaDBVectorMemoryConfig(
        collection_name="documents",
        persistence_path="./chroma_db",
        k=10,
        score_threshold=0.6,
        )
)
```