**Students: Setting Up Your MCP-Powered GRC Retrieval System — *OpenAI Edition***

Imagine you’re building a GRC assistant that can:

* Keep track of every user conversation for context (“memory” server).
* Answer compliance and risk questions by pulling facts from company intake data, regulatory CSVs, and PDF files (“GRC retrieval” server).
* Use OpenAI’s API (e.g., GPT-4) to generate answers—**but only using the information it retrieves from your data sources, never hallucinating**.

**System Architecture & Data Sources**

| **Data Store** | **Backend Data File/Folder** | **Use Case Example** |
| --- | --- | --- |
| Memory Store | In-memory or DB (mcp\_memory\_server.py) | Conversation history for context |
| Company Info | trustlayer\_intake\_example\_cleaned\_final\_v3.csv | Questions about company risk controls |
| GRC Regulations | external\_regulations.csv | Legal/compliance lookups (NIST, GDPR) |
| PDF Knowledge Base | data/ (folder with PDFs) | Deep-dive, cited GRC documents |

**1. MCP Memory Server**

**File:** mcp\_memory\_server.py  
**Purpose:**

* Store and retrieve conversation history by session/user.
* Every time the user (or your notebook) sends/receives a message, the Q&A pair is logged.
* Returns: Conversation history as a list (for context injection).

**Benefit:**  
In GRC, knowing the history of questions/answers is essential for context, traceability, and reducing repetitive questions.

**2. MCP GRC Retrieval Server**

**File:** mcp\_grc\_server.py  
**Purpose:**

* Receives search requests for company info, regulatory text, and PDF content.
* Returns: Top-matching text chunks + metadata (file name, page, etc.) relevant to the GRC query.

**Benefit:**  
Provides **auditable, real-source data** for compliance and risk answers. Separation from the LLM keeps your sensitive data secure.

**3. The Orchestration Client (Jupyter Notebook Example)**

**File:** client\_example.ipynb  
**How it works:**

1. **You enter a compliance question.**
2. Notebook **routes** the question to:
   * The memory server (for context)
   * The GRC server (for facts)
3. Notebook **builds a prompt** for OpenAI (e.g., GPT-4) using retrieved context (not the LLM’s own knowledge!).
4. Notebook **calls OpenAI’s API** and displays the answer—with citations.

**Code Block Example & Explanation:**

python

CopyEdit

import openai

# 1. Get prior conversation from Memory Server

history = get\_conversation\_history(session\_id)

# Returns: List of Q&A pairs, e.g., [{"question": "...", "answer": "..."}]

# 2. Query GRC Server for relevant facts/chunks

context\_chunks = query\_grc\_server("search\_pdfs", {"query": user\_question})

# Returns: List of {"text": "...", "source": "GDPR.pdf", "page": 12, ...}

# 3. Build prompt for OpenAI

prompt = build\_prompt(history, context\_chunks, user\_question)

# Returns: A string with explicit citation instructions

# 4. Call OpenAI (GPT-4) API

response = openai.ChatCompletion.create(

model="gpt-4",

messages=[{"role": "system", "content": system\_prompt},

{"role": "user", "content": prompt}],

temperature=0.0

)

answer = response['choices'][0]['message']['content']

# Returns: Synthesized answer, citing only retrieved context, e.g.,

# "Yes, Trust Layer is subject to GDPR [GDPR.pdf, page 12]."

# 5. Display answer, optionally log to Memory Server

print(answer)

insert\_memory(session\_id, user\_question, answer)

**Step-by-Step What Happens:**

* **Step 1:** Fetches the conversation so far—so GPT-4 knows what’s already been discussed (prevents repeating and improves context).
* **Step 2:** Pulls only relevant regulatory/factual info—so GPT-4 is grounded in real data.
* **Step 3:** Builds a strict prompt, instructing GPT-4 to only use the provided sources and cite everything.
* **Step 4:** Calls OpenAI—GPT-4 processes only your retrieved context (not its own training data).
* **Step 5:** Displays and logs the answer for future reference.

**4. Why This Is Important for GRC Using MCP**

* **Auditability:** Every answer is backed by a real source (file/page)—essential for compliance evidence and regulatory audits.
* **Separation of Duties:** The LLM never sees the database/files directly—reduces data leakage and risk.
* **Extensibility:** Add new regulations or internal policies by updating your CSVs or PDFs—no model retraining required.
* **Transparency:** “Show your work” is enforced. If the answer isn’t in your data, GPT-4 is instructed to say “I do not know.”
* **Consistency:** Context-aware, session-specific memory allows longitudinal compliance analysis and helps with investigations.

**5. Assignment: Your Turn**

1. **Run both MCP servers** (memory and GRC) locally.
2. **Use the Jupyter notebook** to simulate client logic: Route questions, collect context, and call OpenAI’s API.
3. **Test various GRC questions:**
   * “Summarize our last three compliance conversations.”
   * “What does NIST say about incident response?”
   * “What’s in our TrustLayer intake about vendor risk management?”
   * “Find the GDPR section on breach notification in our PDFs.”
4. **Observe and document:**
   * How OpenAI’s answer is *improved and grounded* when given only high-quality, cited context.
   * How every response is transparent and auditable.

**Diagram**

If you want, request a visual architecture diagram—happy to include!



**This workflow empowers GRC teams to leverage world-class language models safely, transparently, and with audit-ready confidence.**