

**required steps to build the agentic ai solution for the catalouge request items which are requesting for db size user role service account role and access info of database using langraph/lanchain LLM model mcp design the flow**

To build an agentic AI solution for catalog request items—specifically to handle requests involving database size, user roles, service account roles, and access info of databases—using LangGraph/LangChain LLM models combined with Anthropic’s Model Context Protocol (MCP), you can follow a structured multi-step design and implementation flow. This approach leverages agentic AI principles, LangGraph’s flexible workflow orchestration, and MCP’s standardized external tool integration.

**Required Steps and Flow Design for the Agentic AI Solution**

**1. Define the Problem and Task Decomposition**

* The agent needs to handle complex queries about database metadata and access control, such as:
  + Database size
  + User roles
  + Service account roles
  + Access permissions
* These queries require:
  + Multi-step reasoning and sequential subtasks (e.g., fetch metadata, check roles, verify access)
  + Access to multiple external data sources and tools (database APIs, role management systems)
* Agentic AI is suitable because it can autonomously plan and execute these subtasks with memory and tool access[[1]](#fn1).

**2. Set Up the LangGraph Agentic Workflow**

* Use **LangGraph** to build a graph-based agent workflow where each node represents a step or tool interaction.
* Define a **State** type to hold conversation context and intermediate results (e.g., messages, query parameters)[[2]](#fn2).
* Create nodes for:
  + Intent understanding (LLM node to parse user request)
  + Data retrieval nodes (call database size API, user role API, service account role API)
  + Access info retrieval node
  + Result aggregation and response generation (LLM summarization node)
* Connect nodes with edges to define the flow, allowing branching or looping if more info is needed[[2]](#fn2)[[3]](#fn3).

Example snippet to create nodes and edges:

graph\_builder = StateGraph(State)  
graph\_builder.add\_node("parse\_intent", run\_llm\_parse\_intent)  
graph\_builder.add\_node("fetch\_db\_size", fetch\_db\_size\_tool)  
graph\_builder.add\_node("fetch\_user\_roles", fetch\_user\_roles\_tool)  
graph\_builder.add\_node("fetch\_service\_account\_roles", fetch\_service\_account\_roles\_tool)  
graph\_builder.add\_node("fetch\_access\_info", fetch\_access\_info\_tool)  
graph\_builder.add\_node("aggregate\_response", run\_llm\_aggregate)  
# Define edges to control flow  
graph\_builder.add\_edge(START, "parse\_intent")  
graph\_builder.add\_edge("parse\_intent", "fetch\_db\_size")  
graph\_builder.add\_edge("parse\_intent", "fetch\_user\_roles")  
graph\_builder.add\_edge("parse\_intent", "fetch\_service\_account\_roles")  
graph\_builder.add\_edge("parse\_intent", "fetch\_access\_info")  
graph\_builder.add\_edge("fetch\_db\_size", "aggregate\_response")  
graph\_builder.add\_edge("fetch\_user\_roles", "aggregate\_response")  
graph\_builder.add\_edge("fetch\_service\_account\_roles", "aggregate\_response")  
graph\_builder.add\_edge("fetch\_access\_info", "aggregate\_response")  
graph\_builder.add\_edge("aggregate\_response", END)  
graph = graph\_builder.compile()

**3. Integrate External Data Sources via MCP**

* Use **Anthropic’s Model Context Protocol (MCP)** to standardize access to external data sources and tools:
  + Define MCP **Tools** for each external API:
    - Database size retrieval
    - User role queries
    - Service account role queries
    - Access info queries
  + Each tool has metadata describing inputs, outputs, and error handling[[4]](#fn4)[[3]](#fn3)[[5]](#fn5).
* MCP acts as a universal bridge allowing the LangGraph agent to call these tools seamlessly without custom integration for each data source[[4]](#fn4).
* Implement or connect to an MCP server that exposes these tools.

Example MCP tool usage in LangGraph:

tools = await load\_mcp\_tools(session) # session connected to MCP server  
agent = create\_react\_agent(model, tools)  
result = await agent.ainvoke({"messages": user\_query})

**4. Implement Memory and State Persistence**

* Use LangGraph’s built-in state management to track intermediate outputs and conversation context[[2]](#fn2)[[3]](#fn3).
* Persist session data in a database to maintain context across multiple user interactions (important for complex or follow-up queries)[[3]](#fn3).
* This ensures the agent remembers previous steps and user-specific info, improving continuity.

**5. Control Agent Autonomy and Flow**

* Design opinionated, modular flows rather than a single free-form agent:
  + Separate sub-agents or nodes handle specific subtasks (e.g., role fetching vs. size retrieval)[[3]](#fn3).
* Use LangGraph’s graph structure to enforce controlled sequences and avoid unpredictable tool usage.
* Implement guardrails to restrict which tools can be called and when, ensuring reliability.

**6. Testing and Iteration**

* Test the multi-step workflow with sample catalog requests.
* Validate that each tool call returns correct data and that the LLM nodes correctly interpret and aggregate results.
* Stress test for performance and scalability, especially for state persistence and MCP tool calls[[3]](#fn3).

**Summary of the Agentic AI Solution Flow**

|  |  |  |
| --- | --- | --- |
| Step | Description | Technology/Component |
| 1. Intent Parsing | LLM node parses user query to identify requested info (db size, roles, access) | LangGraph LLM Node |
| 2. Tool Invocation | Call MCP tools to fetch database size, user roles, service account roles, access info | MCP Tools via LangGraph |
| 3. Data Aggregation | LLM node summarizes and formats the collected data into a user-friendly response | LangGraph LLM Node |
| 4. State Management | Track conversation state and intermediate outputs across steps | LangGraph built-in state + persistent DB |
| 5. Controlled Workflow | Use graph edges and nodes to enforce logical, safe sequence of operations | LangGraph graph structure |
| 6. Extensibility | Add new tools or data sources by registering them with MCP, integrated seamlessly | MCP standardized protocol |

**Conclusion**

By combining LangGraph’s flexible graph-based orchestration with Anthropic’s MCP standardized tool integration, you can build a robust agentic AI solution that autonomously handles complex catalog requests involving database size, user roles, service account roles, and access information. This design supports multi-step reasoning, reliable state management, and extensible tooling, enabling efficient and scalable AI-powered catalog item management[[1]](#fn1)[[2]](#fn2)[[4]](#fn4)[[3]](#fn3)[[5]](#fn5).

⁂

1. <https://www.k2view.com/what-is-agentic-ai/>

1. <https://www.getzep.com/ai-agents/langgraph-tutorial>

1. <https://www.qodo.ai/blog/building-agentic-flows-with-langgraph-model-context-protocol/>

1. <https://www.confluent.io/blog/ai-agents-using-anthropic-mcp/>

1. <https://www.getzep.com/ai-agents/developer-guide-to-mcp>