Agent Architecture - Movie/Series Assistant

Overview

Single-loop conversational agent that queries databases, APIs, and web sources to answer questions about movies and series. Each user message triggers one complete workflow cycle.

Core Principles

- **Tool Priority**: SQL → OMDB → Web Search (use simpler sources first)
- **Structured Outputs**: Pydantic models guarantee stable, type-safe responses
- **Stateful Memory**: LangGraph manages conversation history automatically
- **Conditional Execution**: Only run tools that are actually needed

Workflow Steps

1. **Planner Node**

Purpose: Analyze question + history → decide which tools to use

Input:

- Current user question
- Conversation history (messages)
- DB catalog (if loaded)

Output (structured):

- `resolved_query`: Question reformulated with context from history
- `planning_reasoning`: Why these tools are needed
- `needs_sql/omdb/web`: Boolean flags for each tool
- `sql_query/omdb_query/web_query`: Prepared queries for each tool

Why: One decision point prevents redundant LLM calls and ensures consistent tool selection

2. **Conditional Routing**

Purpose: Execute only the tools flagged as needed

Logic:

- If `needs_sql` → route to SQL node
- If `needs_omdb` → route to OMDB node
- If `needs_web` → route to Web node
- Skip tools with `False` flags

Why: Saves API calls, reduces latency, avoids unnecessary data

3. **Tool Nodes** (SQL / OMDB / Web)

Purpose: Execute tool and return raw results

Each node:

- Receives prepared query from planner
- Executes tool operation
- Stores raw result in state (`sql_result`, `omdb_result`, `web_result`)
- Tracks source for citations

Why: Clean separation - tools just fetch data, don't format responses

4. **Synthesizer Node**

Purpose: Compile all results into natural language response

Input:

- Original question
- All tool results (raw data)
- Planning reasoning (why tools were used)
- Sources used

Output:

- `final_response`: Natural, coherent answer citing sources
- Formatted for user display

Why: Single point for response generation ensures consistent tone and proper citation

State Management

Persistent across turns:

- `messages`: Full conversation history
- `db_catalog`: Database schema (loaded once)

Reset each turn:

- Tool flags (`needs_*`)
- Tool queries (`*_query`)
- Tool results (`*_result`)

Why: Conversation context persists, but tool decisions are fresh each turn

Key Design Decisions

Structured outputs: Pydantic models prevent JSON parsing errors and guarantee type safety

Single planner: One analysis step is cleaner than multi-stage decision making

Conditional edges: Skip unnecessary tools automatically based on flags

Raw results in state: Synthesizer has full data access for best response quality

Tool priority embedded in prompt: Guides planner to prefer SQL over external APIs

Scalability

To add new tools:

1. Add `needs_newtool: bool` to state
2. Add `newtool_query: str` and `newtool_result: str` to state
3. Create tool node
4. Add condition to routing edge
5. Update planner prompt with tool description

No architectural changes needed - pattern scales linearly