MCP Project Exercise - E-Commerce Data Analysis Agent

Project Overview

In this exercise, you'll build a **Model Context Protocol (MCP) application** that enables natural language querying of e-commerce data. Participants will create the necessary components to connect LangChain agents with DuckDB databases using the MCP framework.

📋 What You're Given (Inputs)

Pre-Built Components:

1. **E-Commerce Dataset** - Maven Analytics Toy Store data (CSV files)
    - orders, order_items, order_item_refunds, products, website_pageviews, website_sessions
2. **Base Project Structure** - Git repository with:
    - pyproject.toml (project configuration)
    - README.md (documentation)
    - .gitignore (version control)
    - uv.lock (dependency lock file)
3. **Database Utils** - Partial implementation of:
    - `db_utils_mcp_server.py` (MCP server skeleton)
    - Connection utilities and view creation logic

🎯 What You Need to Create

Task 1: Implement the MCP Server (`db_utils_mcp_server.py`)

Objective: Create MCP tools that expose database operations

Requirements:

- [ ] Implement `get_connection()` function to:
    - ○ Establish DuckDB connection
    - ○ Create temporary views from parquet files
    - ○ Return the connection object
- [ ] Implement `@mcp.tool` decorated function: `run_sql(sql_query: str)`
    - ○ Execute SQL queries on DuckDB

    - ○ Return results as JSON with metadata (row count, execution time)
    - ○ Handle errors gracefully

- [ ] Implement `@mcp.tool` decorated function: `get_table_schema(table_name: str)`
    - ○ Retrieve schema information for any table
    - ○ Return schema as JSON
    - ○ Support all 6 available tables

Acceptance Criteria:

- [ ] Server starts without errors: `uv run db_utils_mcp_server.py`
- [ ] Both tools have proper docstrings
- [ ] Error handling is implemented
- [ ] Results are returned in JSON format

Task 2: Build the LangChain Agent (`agent.py`)

Objective: Create an agent that uses MCP tools to answer database questions

Requirements:

- [ ] Initialize MCP client with proper configuration
- [ ] Connect to the MCP server via stdio transport
- [ ] Retrieve available tools from the server
- [ ] Create a LangChain agent with GPT-4o-mini model
- [ ] Implement `run_agent(system_message: str, query: str)` function
    - ○ Accept user queries
    - ○ Use the agent to answer questions
    - ○ Return formatted responses
- [ ] Create a system prompt that instructs the agent to:
    - ○ Use `get_table_schema` to understand table structures
    - ○ Use `run_sql` to query data
    - ○ Think step-by-step
    - ○ Handle date/timestamp casting for `created_at` columns

Acceptance Criteria:

- [ ] Agent can be run: `uv run agent.py`
- [ ] Test query returns data from the database
- [ ] Agent properly uses both tools
- [ ] Responses are clear and formatted

Task 3: Build the Streamlit Web Interface (`streamlit_app.py`)

Objective: Create a user-friendly web interface for database querying

Requirements:

- [ ] Configure Streamlit page with title and layout
- [ ] Create a sidebar with:
  - ○ Dataset information (name and description)
  - ○ List of available tables
  - ○ Example queries for users
  - ○ Query history dashboard with metrics
- [ ] Main interface with:
  - ○ Text input area for user queries
  - ○ Submit button to execute queries
  - ○ Results display area
  - ○ Error handling and user feedback
- [ ] Implement query logging:
  - ○ Auto-create `logs/` directory if it doesn't exist
  - ○ Save all queries and responses to `logs/query_logs.json`
  - ○ Log success/failure status and timestamps
  - ○ Provide query history dropdown in sidebar

Acceptance Criteria:

- [ ] App runs: `uv run streamlit run streamlit_app.py`
- [ ] Can input natural language queries
- [ ] Responses display correctly
- [ ] Query logs are saved to JSON
- [ ] Sidebar shows dataset info and history
- [ ] Error handling is user-friendly

Task 4: Configure Environment & Documentation

Objective: Set up configuration files and documentation

Requirements:

- [ ] Create `.env` file with:

```
OPENAI_API_KEY=your-api-key-here
```

- [ ] Ensure `README.md` includes:
  - ○ Project overview
  - ○ Prerequisites and setup instructions
  - ○ Project structure
  - ○ Usage examples for all three components
  - ○ Configuration details
  - ○ Available tables documentation

- [ ] Ensure data is available:
    - ○ Convert CSV files to Parquet format OR
    - ○ Ensure CSV files are in `data/` directory

Acceptance Criteria:

- [ ] .env file is created and .env is in .gitignore
- [ ] README is comprehensive and up-to-date
- [ ] Data files are accessible (parquet or CSV)

📊 Optional Challenges (Advanced)

Challenge 1: Enhanced Agent Capabilities

- Add tool for generating data visualizations
- Implement multi-step reasoning for complex queries
- Add data export functionality (CSV, Excel)

Challenge 2: Improved UI/UX

- Add query templates/presets
- Implement query result caching
- Add data refresh capabilities
- Create custom styling with CSS

Challenge 3: Advanced Logging & Analytics

- Analyze query patterns and performance
- Create a dashboard showing query statistics
- Implement query search/filter functionality
- Add query timing analytics

Challenge 4: Production Readiness

- Add comprehensive error handling
- Implement rate limiting
- Add authentication/authorization
- Create deployment configuration (Docker, etc.)

✅ Testing Checklist

Before submission, ensure:

- [ ] MCP Server starts and exposes tools
- [ ] Agent can answer: "What are the top 5 products by sales?"
- [ ] Streamlit app runs without errors
- [ ] Query logs are created in `logs/query_logs.json`
- [ ] Sidebar displays dataset info and example queries
- [ ] At least 5 successful queries are logged

- [ ] Error handling works (test with invalid query)
- [ ] README is clear and complete

📝 Submission Requirements

1. **Code Quality**
    - Clean, readable code with comments
    - Proper error handling throughout
    - No hardcoded paths (use relative paths)
2. **Documentation**
    - Docstrings for all functions
    - README covers all components
    - Comments for complex logic
3. **Functionality**
    - All three components working together
    - Query logging functional
    - At least 10 successful query logs
4. **Git Repository**
    - Clean commit history
    - Descriptive commit messages
    - .gitignore properly configured

🎓 Learning Outcomes

After completing this exercise, you'll understand:

✓ How Model Context Protocol (MCP) works
✓ Building MCP servers with FastMCP
✓ Integrating LLMs with tool use and function calling
✓ Using LangChain agents for autonomous tool selection
✓ Creating data applications with Streamlit
✓ Database querying with DuckDB
✓ Building end-to-end AI applications
✓ Logging and monitoring AI agent behavior

📚 Resources

- FastMCP Documentation
- LangChain Documentation
- Streamlit Documentation
- DuckDB Documentation
- Model Context Protocol Spec

? Questions?

- Refer to the README.md for setup issues
- Check individual file docstrings for implementation details
- Review error messages carefully - they often indicate what's missing
- Ask instructors for clarification on requirements

🚀 Expected Timeline

- **Task 1 (MCP Server):** 30-45 minutes
- **Task 2 (LangChain Agent):** 30-45 minutes
- **Task 3 (Streamlit UI):** 45-60 minutes
- **Task 4 (Configuration):** 15-20 minutes
- **Testing & Refinement:** 20-30 minutes

Total: 2.5 - 3.5 hours

Good luck! 🎉