Assignment: Multi-Agent Customer Service System with A2A and MCP

Overview

Build a multi-agent customer service system where specialized agents coordinate using Agent-to-Agent (A2A) communication and access customer data through the Model Context Protocol (MCP).

Learning Objectives

- Implement agent coordination using A2A protocols
- Integrate external tools via MCP
- Design multi-agent task allocation and negotiation
- Build a practical customer service automation system

Assignment Requirements

Part 1: System Architecture

Design a multi-agent system with **at least three specialized agents**:

1. **Router Agent** (Orchestrator)
    - Receives customer queries
    - Analyzes query intent
    - Routes to appropriate specialist agent
    - Coordinates responses from multiple agents
2. **Customer Data Agent** (Specialist)
    - Accesses customer database via MCP
    - Retrieves customer information
    - Updates customer records
    - Handles data validation
3. **Support Agent** (Specialist)
    - Handles general customer support queries
    - Can escalate complex issues
    - Requests customer context from Data Agent
    - Provides solutions and recommendations

Part 2: MCP Integration (25 points)

Implement an MCP server with the following tools:

See attached setup_database.py file

Required Tools:

1. `get_customer(customer_id)` - uses customers.id
2. `list_customers(status, limit)` - uses customers.status
3. `update_customer(customer_id, data)` - uses customers fields
4. `create_ticket(customer_id, issue, priority)` - uses tickets fields
5. `get_customer_history(customer_id)` - uses tickets.customer_id

Database Schema:

Your MCP server should maintain two main data structures:

Customers Table:

`id`	`INTEGER PRIMARY KEY`
`name`	`TEXT NOT NULL`
`email`	`TEXT`
`phone`	`TEXT`
`status`	`TEXT ('active' or 'disabled')`
`created_at`	`TIMESTAMP`
`updated_at`	`TIMESTAMP`

Tickets Table:

`id`	`INTEGER PRIMARY KEY`
`customer_id`	`INTEGER (FK to customers.id)`
`issue`	`TEXT NOT NULL`
`status`	`TEXT ('open', 'in_progress', 'resolved')`
`priority`	`TEXT ('low', 'medium', 'high')`
`created_at`	`DATETIME`

Part 3: A2A Coordination

Choose ONE of the following approaches:

Option A: Lab Notebook Approach (Recommended Starting Point)

- Use the A2A coordination pattern from your lab notebook: https://colab.research.google.com/drive/1YTVbosORUrKe_qOysA3XEhhBaCwdbMj5
- Extend it to support the three required scenarios (task allocation, negotiation, multi-step)
- Add explicit logging to show agent-to-agent communication
- Document how agents coordinate and transfer control
- You may use the same framework but must demonstrate more complex coordination patterns

Option B: LangGraph Message Passing

- Define a shared state structure that agents can read/write
- Create nodes for each agent
- Implement conditional edges for routing between agents
- Use message passing to share information between agents
- Handle state transitions explicitly

Scenario 1: Task Allocation

Query: "I need help with my account, customer ID 12345"

A2A Flow:

1. Router Agent receives query
2. Router Agent → Customer Data Agent: "Get customer info for ID 12345"
3. Customer Data Agent fetches via MCP
4. Customer Data Agent → Router Agent: Returns customer data
5. Router Agent analyzes customer tier/status
6. Router Agent → Support Agent: "Handle support for premium customer"
7. Support Agent generates response
8. Router Agent returns final response

Scenario 2: Negotiation/Escalation

Query: "I want to cancel my subscription but I'm having billing issues"

A2A Flow:

1. Router detects multiple intents (cancellation + billing)
2. Router → Support Agent: "Can you handle this?"
3. Support Agent → Router: "I need billing context"
4. Router → Customer Data Agent: "Get billing info"
5. Router negotiates between agents to formulate response

6. Coordinated response sent to customer

Scenario 3: Multi-Step Coordination

Query: "What's the status of all high-priority tickets for premium customers?"

A2A Flow:

1. Router decomposes into sub-tasks
2. Router → Customer Data Agent: "Get all premium customers"
3. Customer Data Agent → Router: Returns customer list
4. Router → Support Agent: "Get high-priority tickets for these IDs"
5. Support Agent queries tickets via MCP
6. Agents coordinate to format report
7. Router synthesizes final answer

Test Scenarios

Your system must successfully handle these queries:

1. **Simple Query**: "Get customer information for ID 5"
   - Single agent, straightforward MCP call
2. **Coordinated Query**: "I'm customer 12345 and need help upgrading my account"
   - Multiple agents coordinate: data fetch + support response
3. **Complex Query**: "Show me all active customers who have open tickets"
   - Requires negotiation between data and support agents
4. **Escalation**: "I've been charged twice, please refund immediately!"
   - Router must identify urgency and route appropriately
5. **Multi-Intent**: "Update my email to new@email.com and show my ticket history"
   - Parallel task execution and coordination
   -

Deliverables

1. Code Repository (GitHub)

- MCP server implementation
- Agent implementations
- Configuration and deployment scripts
- `README.md` with setup instructions

2. Colab Notebook or a python program that runs end to end

- End-to-end demonstration
- At least 3 test scenarios showing A2A coordination
- Output captured properly that shows the queries.

3. Conclusion

1-2 paragraphs of what you learned and challenges

Common Pitfalls to Avoid

MCP Integration Issues

- **Problem:** MCP server becomes unreachable during testing
  - **Solution:** Keep ngrok session active, implement reconnection logic
- **Problem:** Tools timeout or fail silently
  - **Solution:** Add explicit error handling and logging in each MCP tool
- **Problem:** Database state gets corrupted during testing
  - **Solution:** Implement database reset function, use transactions

A2A Coordination Issues

- **Problem:** Agents get stuck in infinite loops
  - **Solution:** Add maximum iteration limits, implement timeout logic
- **Problem:** Agent responses are inconsistent
  - **Solution:** Be explicit in system instructions, add examples
- **Problem:** Information gets lost between agent transfers
  - **Solution:** Use structured state, log all transfers

Implementation Challenges

- **Problem:** Response times exceed 3 seconds
  - **Solution:** Parallelize independent agent calls, cache frequent queries
- **Problem:** Agents don't coordinate properly
  - **Solution:** Test each agent independently first, then test pairs
- **Problem:** Difficult to debug multi-agent interactions
  - **Solution:** Add comprehensive logging at every coordination point