Overview of the Workflow

The agent workflow diagram shows how a central system manages requests from users, like checking inventory, getting price quotes, and placing orders.

The process starts when a user makes a request. A central orchestrator agent takes charge and assigns tasks to specialized agents. Each of these agents uses a specific tool to do its job, so responsibilities are clear and decisions are made in a controlled way.

Components and Agent Roles

1. **User Query**

Query represents any request initiated by the user, this request is always passed directly to the orchestrator for interpretation and routing.

2. **Orchestrator (Agent 1)**

The orchestrator is the central decision-making agent in the architecture. Its primary role is to analyze the user query and determine which specialized agent(s) should be invoked.

The orchestrator does not directly perform inventory checks, pricing calculations, or order placement; instead, it coordinates these actions by dispatching tasks and collecting responses. This centralized control simplifies flow management and ensures consistent handling of user requests.

3. **Inventory Agent (Agent 2)**

The agent is responsible for handling all inventory-related queries. When the orchestrator determines that a request requires stock availability information, it routes the task to this agent. The inventory agent then calls the `check_inventory_tool()` to retrieve up-to-date inventory data and returns the result to the orchestrator.

4. **Quoting Agent (Agent 3)**

The agent manages pricing and quotation requests - upon instruction from the orchestrator, it invokes the `generate_quote_tool()` to calculate pricing based on product name and quantity. The generated quote is then sent back to the orchestrator for further processing or presentation to the user.

5. **Ordering Agent (Agent 4)**

The agent handles transactional operations related to order placement. When the orchestrator identifies that the user intends to place an order, it delegates the task to this agent. The ordering agent calls the `place_order_tool()` to complete the transaction and reports the outcome back to the orchestrator.

6. **Tools Layer**

Each agent (excluding the orchestrator) is connected to a specific tool that performs the actual system-level operation (inventory lookup, quote generation, or order placement). This

separation allows agents to focus on decision logic while tools handle execution, improving maintainability and security.

Decision-Making Process and Architectural Rationale

The architecture was intentionally designed around a central orchestrator with specialized agents for the following reasons:

- Clear Control Flow: By routing all decisions through the orchestrator, the system ensures that user intent is interpreted consistently before any action is taken.
- Separation of Concerns: Each agent has a single, well-defined responsibility, reducing complexity and making the system easier to debug and extend.
- Scalability: New agents and tools (e.g., payment, shipping, or returns) can be added without modifying existing agents, only extending the orchestrator's routing logic.
- Reliability and Safety: Sensitive actions such as order placement are isolated within the ordering agent and its tool, reducing the risk of unintended side effects.

Overall, the workflow diagram demonstrates a deliberate design choice favoring modularity, centralized decision-making, and tool-based execution. This structure supports robust handling of user queries while remaining flexible for future expansion.

Evaluation

The system demonstrates several notable strengths, primarily through its modular, agent-based architecture and centralized orchestration. By separating responsibilities across specialized agents such as inventory checking, quoting, and ordering, the design promotes clarity, maintainability, and ease of extension. The use of an orchestrator ensures consistent interpretation of user intent and controlled decision-making, reducing the likelihood of unintended actions. Additionally, delegating execution to dedicated tools improves reliability and allows each agent to focus on reasoning rather than low-level operations. Overall, this structured workflow enhances transparency, supports scalability as new agents can be added with minimal disruption, and provides a solid foundation for building robust and adaptable transactional systems.

One notable weakness of the system arises during the inventory checking process, where user-specified item names do not always match the exact product names stored in the database. This mismatch can lead to failed inventory lookups or incorrect results, even when the requested item is available. The issue is primarily caused by variations in naming conventions, such as abbreviations, spelling differences, pluralization, or user-friendly descriptions that differ from the database's standardized product identifiers.

Potential improvements include implementing fuzzy matching, synonym mapping, or introducing a product-mapping layer that translates user inputs into canonical database product names before inventory checks are performed. Addressing this weakness would significantly enhance system reliability and user experience.

Example:

When a customer requests an item such as A4 white printer paper or A4 glossy paper (seen in test_results - row 9 and row 8 respectively) - it does not recognise these as actual products, as the LLM is unaware of what the correct naming is within our database. This is an issue, as these products should be ordered and are in stock - but will never be ordered (again showing us the limitation spoken above).

While your current workflow is functional, switching models could significantly sharpen its performance. Using a reasoning-focused model would help the agent plan complex tasks more accurately and avoid logical errors during multi-step processes. At the same time, integrating a lighter, faster model for simpler tasks would drastically reduce wait times, making the entire system feel more responsive and efficient. Essentially, the right model swap could provide the "brains" for better logic and the "speed" for a smoother experience.

Example:

In the prompt I have enforced ‘No recommendation’ should be offered by the LLM if an item is unavailable, however there are places in data where it has recommended me to look at other places to purchase such items (such as row 2) - this might have been avoided if I used a better LLM which had better reasoning abilities.