**Example of how a Query would work in the System**

“Rank our top IT hardware suppliers based on price, delivery performance, and risk over the past 12 months.”

**🧠 End-to-End System Flow (Agentic Architecture)**

**1. User Input**

* **Input:** Natural language query from user
* **Intent Detected:** RankSuppliers
* **Agent Activated:** SupplierRankingAgent

**2. Orchestrator**

* **Responsibility:** Routes the request
* **Steps:**
  + Recognizes this is a multi-agent task with rules
  + Sends task to:
    - **Prompt Engine** to construct LLM instructions
    - **Policy Engine** to load all enforcement rules
    - **Context Manager** to enrich the query with scope (e.g., category = "IT hardware")

**3. Prompt Engine**

* **Matches**: User query to structured prompt template
* **Output Prompt** (to LLM):

Rank suppliers in the IT hardware category based on price, delivery performance, and risk over the past 12 months. Weight them respectively by 0.4, 0.3, and 0.3. Return the result in table format.

* **Template Applied From:** SupplierRankingAgent\_PromptLibrary.json

**4. Policy Engine**

* **Fetches and Enforces Policies:**
  + WeightAllocationPolicy: 40% price, 30% delivery, 30% risk
  + SupplierTieringPolicy: Only “preferred” suppliers
  + ContractCompliancePolicy: Exclude suppliers without active contract
  + RiskRatingPolicy: Exclude suppliers with risk score > 5
  + DeliveryPerformancePolicy: Minimum 95% on-time delivery
* **Returns:** Score logic, filters, overrides to be applied by agent

**5. Context Manager**

* **Fills Gaps:**
  + If category or supplier\_scope was missing, fills it with context from previous user input or profile
  + Ensures LLM gets a complete prompt

**6. LLM Engine**

* **Function:** Generates reasoning chain if needed or rewrites the query to interact with the Knowledge Graph
* May output:

query {

suppliers(category: "IT hardware", is\_preferred: true, contract\_signed: true) {

name

unit\_price

on\_time\_delivery\_rate

risk\_score

}

}

**7. Query Engine**

* Translates the supplier ranking task into a **query plan**:
  + SQL/GraphQL or DSL query based on metadata
  + Applies all filters and scoring policies
* Retrieves:
  + Unit Price
  + Delivery Performance
  + Risk Score
  + Contract/Tiering/Compliance flags

**8. Knowledge Graph (Procurement Data Layer)**

* **Sources:**
  + Supplier Master Data (compliance, risk, tiering)
  + Historical Performance (delivery, invoice history)
  + Contract Metadata (signed status, expiry)
* Returns a linked supplier dataset to Query Engine

**9. SupplierRankingAgent**

* **Scoring Logic**: Based on Supplier\_Ranking\_DSL.json
  + Applies weights and normalizes:

score = (

normalize(unit\_price) \* 0.4 +

normalize(delivery\_rate) \* 0.3 +

normalize\_inverse(risk\_score) \* 0.3

)

* **Ranks Suppliers** by final score

**10. Response Rendering**

* Returns ranked table to UI:  
  | Supplier | Price | Delivery | Risk | Score |  
  |--------------|-------|----------|------|--------|  
  | Supplier X | £2.50 | 98% | 3 | 8.9 |  
  | Supplier Y | £2.40 | 96% | 5 | 8.6 |  
  | Supplier Z | £2.30 | 94% | 4 | filtered (low delivery) |

**✅ Summary of Components Involved**

| **Component** | **Function** |
| --- | --- |
| **Orchestrator** | Directs the flow and agent interactions |
| **Prompt Engine** | Generates structured prompts and templates |
| **Policy Engine** | Applies rules and weights for scoring, filtering |
| **Context Manager** | Enriches missing data or resolves ambiguity |
| **LLM** | Generates reasoning, natural language to structured query |
| **Query Engine** | Executes optimized queries into the data sources |
| **Knowledge Graph** | Stores supplier data, contract metadata, compliance attributes |
| **Agent** | Applies logic, scoring model, ranking logic |