**Dual Database AI Agent**

- Natural Language Interface for MongoDB and MYSQL.

**USE CASE:**

AI-Powered Natural Language Query Agent for Database Management. Non-technical users, like product managers and analysts, often need to access data but struggle with complex queries and risk damaging the database. This AI agent solves these problems by allowing users to ask questions in plain language, such as:

**- “Show all users who signed up last week.”**

**- “Get the total number of orders from Hyderabad.”**

The agent translates these questions into safe, optimized SQL or Mongoose queries, executes them, and provides the results automatically. This way, users can access the data they need without needing technical skills or risking the integrity of the database.

**PROBLEM FRAMING:**

**1.1. What problem does your AI Agent solve?**

Non-technical users often struggle to write complex SQL or MongoDB queries and rely heavily on developers for basic data access. This reliance slows down workflows and creates unnecessary bottlenecks within teams.

**1.2. Why is this agent useful?**

The agent enables users to query MongoDB and MySQL databases using natural language, automatically converting questions into safe, executable database queries. This functionality promotes faster decision-making, saves developers' time, and ensures secure, code-free access to data.

**1.3. Who is the target user?**

Business analysts who need quick data insights

Junior developers who are unfamiliar with MongoDB and SQL

Product managers or testers who are reviewing live data

Students who are learning about database systems.

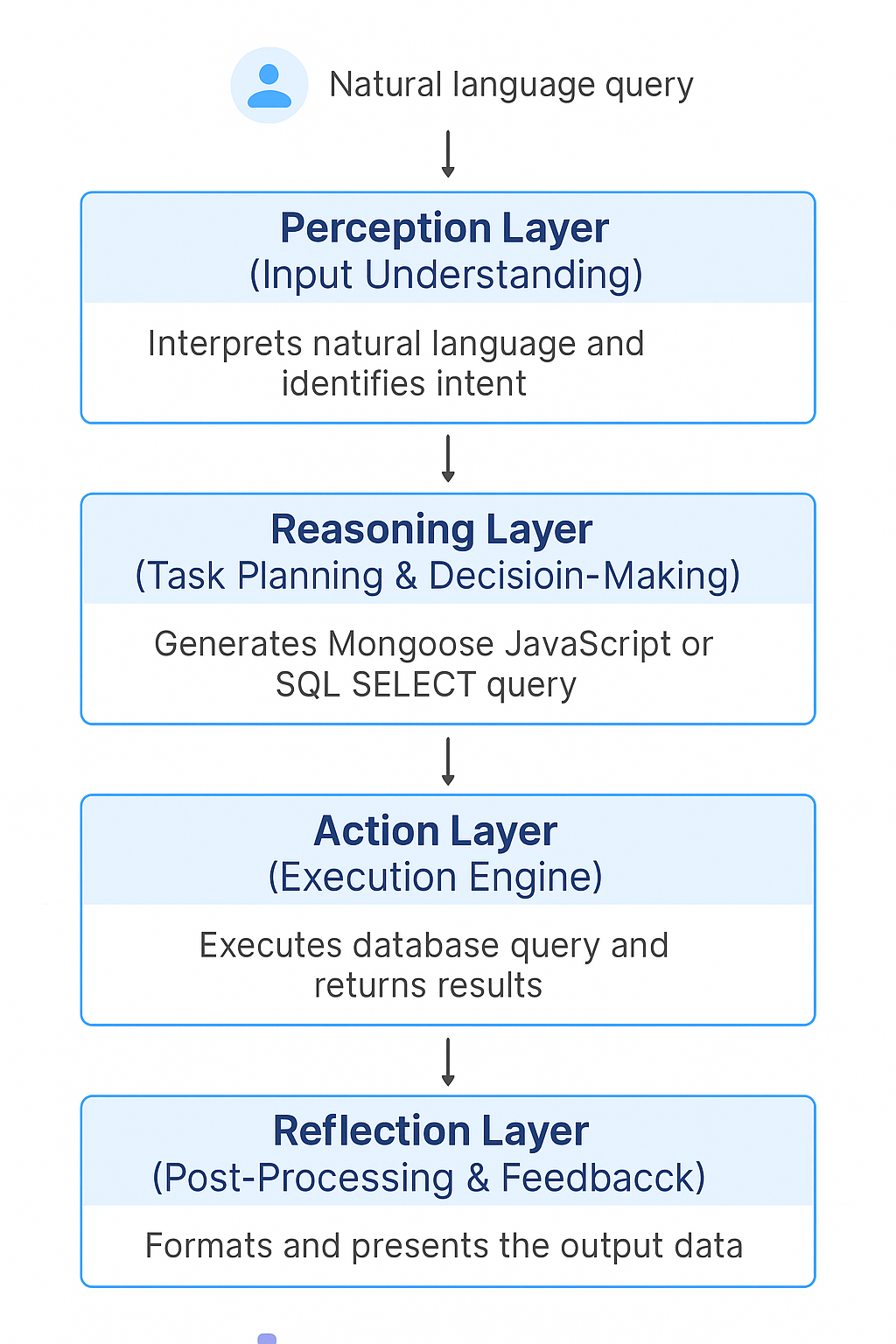
**1.4. What not to include?**

Write/delete/update functionality (to avoid accidental data loss)

Multi-table joins or aggregations for complex analytics (excluded for scope control)

Voice or chatbot interface (deferred for future enhancement)

**4-LAYER PROMPT DESIGN:**



**3.1 INPUT UNDERSTANDING**

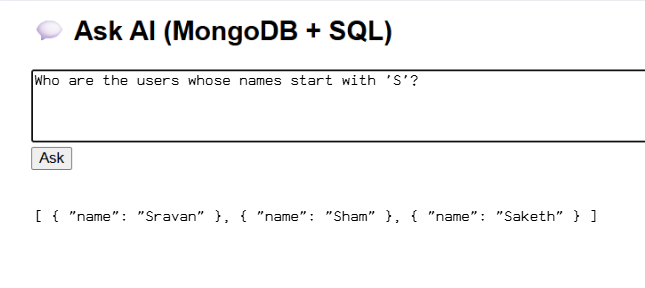
**Prompt:**

> “Can you guide me to build an AI agent that helps with MongoDB?”

> “Can you create an AI for both SQL and MongoDB? That would be outstanding.”

**What does this prompt indicate?**

These prompts define the user's \*\*intent\*\*—to create a dual-database AI agent that works with natural language inputs for both \*\*SQL\*\* and \*\*MongoDB\*\* databases. The system recognizes that the user desires functionality for converting plain English queries into database queries.

****

**3.2 STATE TRACKER**

**Prompt:**

> “I don't know anything about AI. You need to guide me every step of the way and also provide the code.”

**How does this help the agent retain context?**

These prompts suggest that the agent should function as a \*\*step-by-step guide\*\* and maintain context. We simulate memory using:

\* Internal project state tracking (System variables and controlled flow)

\* State-aware response chaining (e.g., not suggesting a chatbot until confirming the server is running)

**3.3 TASK PLANNER**

**Prompt:**

> “What problem does our agent solve?”

**What steps does your agent take internally to address the problem?**

Step 1: Detect whether the query is for MongoDB or SQL

Step 2: Generate the correct query syntax using OpenAI

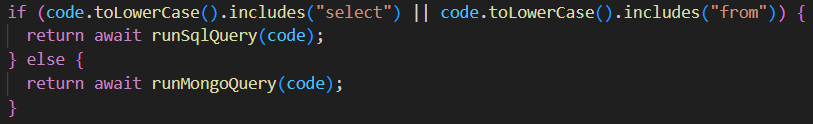
Step 3: Validate query safety (ensuring it does not include harmful commands like DROP or DELETE)

Step 4: Route to the correct database and execute the query

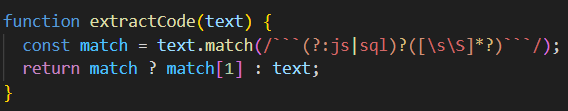
Step 5: Format and return the results

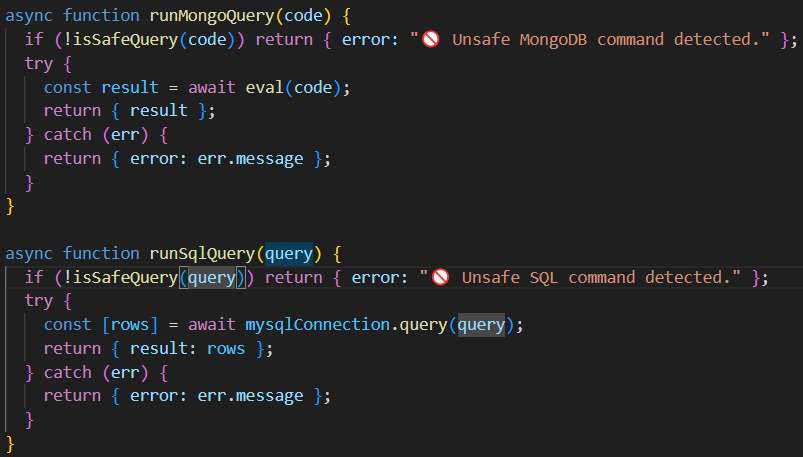
We employed **branching logic** to differentiate between SQL and Mongo queries, along with **modular chaining** using helper functions (e.g., `extractCode()`, `runMongoQuery()`, etc.).

**branching logic**

****

**modular chaining**

****

****

**3.4 OUTPUT GENERATOR**

**Prompt:**

> “Now, build a chatbot for it and also add a visualization feature.”

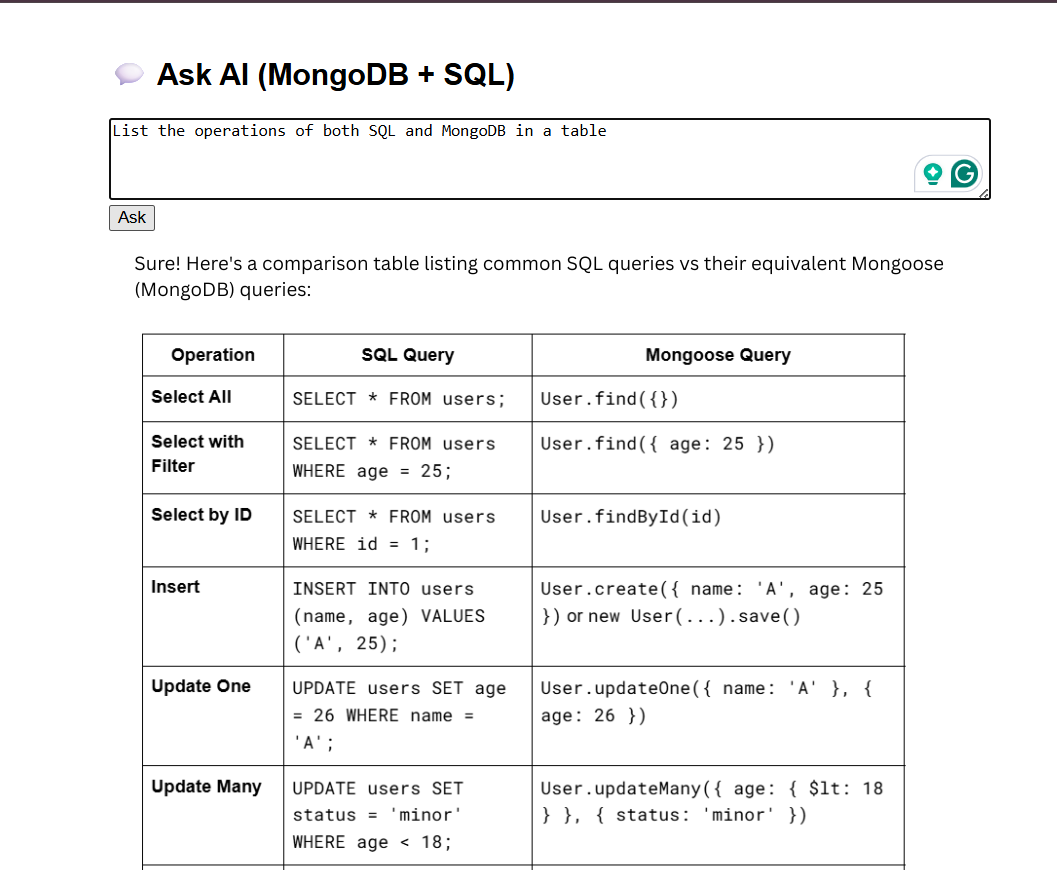
**What type of output formatting or phrasing was aimed for?**

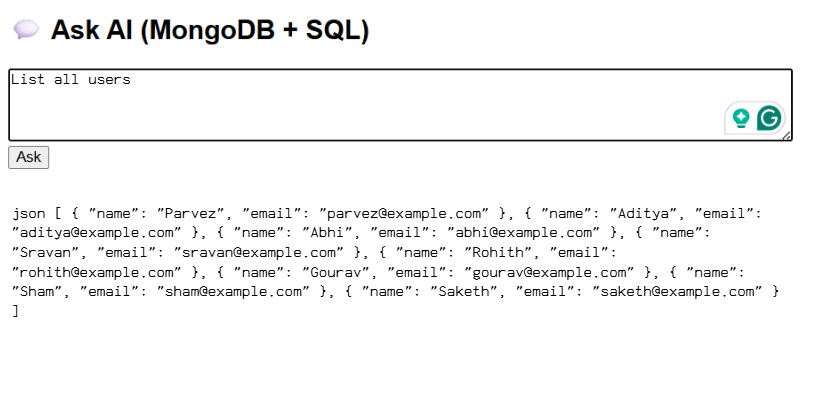
Visual block diagrams to enhance architectural clarity. And a consistent teaching tone suitable for beginners

**CHATGPT EXPLORATION LOG**

| **Attempt** | **Prompt Variant** | **What Happened** | **What You Changed** | **Why You Changed It** |
| --- | --- | --- | --- | --- |
| 1 | I want to make an AI agent | ChatGPT provided general guidance on AI agents and architecture types. | Can you guide me to build an AI agent that helps in MongooseDB? | Narrowed the scope from general AI to a specific, useful use case involving MongoDB. |
| 2 | What problem does our agent solve? | Detects query type, generates and validates syntax, executes, and returns results. | Now build a chatbot for it and also add the visualization feature to it | ChatGPT helped build a frontend chatbot interface and added a /public folder to serve HTML. |
| 3 | Explain the TypeError: Configuration is not a constructor" | You encountered an OpenAI SDK version issue due to outdated Configuration usage. | How to replace the correct OpenAI SDK instantiation: const openai = new OpenAI({ apiKey: ... }). | Fixed compatibility with OpenAI v5.x API (used correct syntax for newer SDK). |
| 5 | Explain the Error: 429 exceeded your current quota… | API quota limit hit. The AI agent stopped responding correctly. | How to handle the problem without any further problem | To understand quota issues and prepare graceful error handling in the agent. |

**OUTPUT TEST:**

****

****

**REFLECTION:**

**6.1. What was the hardest part of this assignment?**

The most challenging aspect was integrating both MongoDB and SQL into a single AI agent. Ensuring that the agent correctly identified the type of database and returned the appropriate query format required careful tuning of prompts and effective error handling. I also encountered API compatibility issues and had to debug several runtime errors.

**6.2. What part did you enjoy the most?**

I particularly enjoyed seeing my natural language queries transform into actual database queries and receiving live results. Building the chatbot interface and visualizing the data felt exciting; it made the project more interactive and intelligent. It was satisfying to witness the entire system function from start to finish.

**6.3. If given more time, what would you improve or add?**

If I had more time, I would implement user authentication, add support for updates and inserts, and possibly incorporate voice input. I would also enhance the error responses and implement logging for AI decision-making. Additionally, if feasible, I would train the agent on a custom dataset to align it more closely with my specific use case.

**6.4. What did you learn about ChatGPT or prompt design?**

I learned that the way a question is asked significantly influences the agent's response. Prompt design is not just about what you want to know; it's also about how clearly you frame the question, considering safety and constraints. Furthermore, layering prompts—such as differentiating between perception and task planning—made my agent more structured and reliable.

**6.5. Did you ever feel stuck? How did you handle it?**

Yes, I often felt stuck, especially when encountering OpenAI errors, outdated syntax, or database connection issues. I managed these challenges by carefully reviewing the error logs, asking specific follow-up questions to ChatGPT, and researching version differences online. Breaking down the problems into smaller parts helped me to keep moving forward without feeling overwhelmed.

**HACK VALUE:**

I exceeded the initial brief by creating a dual-mode agent that supports both MongoDB (using Mongoose) and MySQL. This development involved implementing conditional branching and safety checks. Additionally, I designed the agent to function as a guide for beginners by simulating memory through stateful prompts and layered logic. I also explored query chaining with different types of queries and integrated a chatbot interface for user interaction.