MVP Tech Architecture

Technical Architecture Overview: AlchemData.ai

The technical architecture for AlchemData.ai is designed as a modular, three-pillar system to create a trusted, no-code Analytics Agent Builder . The core objective is to empower business users with reliable data insights by enabling analysts to build powerful, context-aware agents in a one-time setup process .

The system comprises three main independent services that work in concert: a **UI Layer** (React) , a central **Orchestration Service** (the "Master Agent" housed within the Context Builder) , and a dedicated **Platform Service** (the "Query Engine") .

Here is a breakdown of the architecture through the lens of the three foundational pillars.

Pillar 1: The Context Builder

This is the most critical pillar for achieving the product's primary differentiator: delivering 100% trust in the data insights . Its core function is to construct a rich, multi-layered knowledge graph that captures the complete semantic and relational context of the customer's data. This graph serves as the single source of truth for the Master Agent, ensuring it can generate accurate, hallucination-free queries.

Core Component: The Hierarchical Knowledge Graph (using Neo4j)
The context is structured as a hierarchical knowledge graph, which will be implemented using **Neo4j**. This technology is well-suited for managing the complex relationships and hierarchical data required. The graph organizes data context from broad business domains down to granular data points.

The hierarchy is defined as follows:

1. **Agent Persona**: The highest-level node. This represents a logical business grouping or domain, such as "Orders," "Marketing," or "Product Catalog." An analyst defines these personas to encapsulate all the data required to answer questions about that specific domain .
2. **Table Nodes**: Each Agent Persona contains multiple Table nodes, representing the actual database tables relevant to that persona.

3. **Column Nodes**: Each Table node is composed of Column nodes, which hold metadata about the specific columns in that table.
4. **Value Nodes**: For low-cardinality columns (e.g., a "status" column with values like "shipped," "pending," "delivered"), the distinct values themselves become the final leaf nodes in the graph, providing the most granular level of context .

Multi-Layered Context Ingestion Flow:
The knowledge graph is built and enriched through a systematic, three-stage process:

1. **Automated System Layer:** This foundational layer programmatically extracts objective metadata directly from the database schema.
    - **Schema & DDL Analysis**: It ingests Data Definition Language (DDL) to map out Table and Column nodes, capturing properties like data types, `is_nullable`, and `is_unique` constraints .
    - **Statistical Analysis**: The system runs sampling queries to determine statistical properties of columns . A key property is **cardinality**, which dictates the creation of Value nodes .
        - **High-Cardinality Columns** (>5,000 unique values) are represented only by their Column node metadata .
        - **Low-Cardinality Columns** (<50 unique values) are expanded to include distinct Value nodes in the graph .
2. **AI-Assisted Layer:** The system uses an LLM as a co-pilot to accelerate the context-building process for the analyst.
    - **Auto-Description**: For columns and tables, the AI generates preliminary descriptions for the analyst to review and refine .
    - **Suggested Joins**: The AI suggests potential relationships (edges) between Table nodes that aren't explicitly defined by foreign keys, which the analyst can then validate .
3. **Human-in-the-Loop (Analyst) Layer**: This is the crucial final layer where an analyst (the "Agent Builder") provides the domain-specific business logic required for 100% accuracy . The questions/validations in this step are expected to be generated by system as well as LLM. LLM goes through a

sample of data (for now, 1000 rows - but we need to make this more elaborate). Protoype for this - https://alchemi-insight.lovable.app/builder

- **Persona Definition**: The analyst defines the top-level Agent Personas and assigns the relevant tables to each.
- **Annotation and Refinement**: The analyst reviews, edits, and approves all AI-generated descriptions and relationships within the "Column Context Builder" flow .
- **Ambiguity Resolution**: The analyst resolves ambiguities flagged by the system, such as defining the precise relationship between similarly named columns (`order_id` vs. `main_order_id`) to ensure correct query generation .

Pillar 2: The Platform Layer

This pillar reflects the core architectural decisions that make AlchemData.ai a secure, enterprise-ready solution. It acts as a dedicated, standalone query engine, completely decoupled from the main application's business logic .

Core Principles & Architecture:

1. **Hosted in Customer's Backend**: To eliminate data privacy and security hurdles, the entire platform will be deployed within the customer's cloud environment.
2. **Customer-Supplied Credentials**: The platform uses the customer's own LLM API keys and data warehouse credentials, ensuring they maintain full control.
3. **Asynchronous Query Execution**: The platform handles long-running queries without blocking the application, using a `job_id` based queue (e.g., RabbitMQ, SQS) for requests and a pub/sub system (e.g., Redis) for results .
4. **Standardized Data Source Interface**: It provides a single, generic interface for the Master Agent to interact with disparate data sources like Spark, SQL, and Snowflake, handling differences in SQL syntax internally .

Pillar 3: The Agentic Chat Layer

This pillar governs user interaction and AI-driven orchestration. It is coordinated by a central **Master Agent**.

Framework: `smol-agent`
The choice of `smol-agent` is strategic. It is a lightweight, code-centric framework ideal for our use case, which prioritizes generating SQL and Python code as its primary output, avoiding the overhead of more complex multi-agent systems .

End-to-End Information Flow:

1. **Query Input**: A user asks a natural language question in the UI (e.g., "What was the GMV for staples last month?") .
2. **Agent Persona Identification**: The Master Agent parses the query for keywords ("GMV," "staples") and maps them to the most relevant **Agent Persona(s)** from the knowledge graph (e.g., the "Orders" and "Catalog" personas) .
3. **Brute Force Context Stuffing**: Instead of a risky step-by-step traversal of the graph, the system retrieves the *entire context* of all identified Agent Personas. This includes all their associated tables, columns, value-level descriptions, and defined relationships, and "stuffs" it into the LLM's context window . This gives the LLM a complete picture to reason from, significantly improving the accuracy of complex query generation .
4. **Planning & Tool Use**: With this rich context, the Master Agent plans the execution, which may involve generating a single SQL query or a multi-step plan involving Python for data federation across different sources .
5. **Execution**:
    - **SQL Queries**: The Master Agent sends the SQL query as a job to the **Platform Layer** and awaits the result.
    - **Python Code**: Any Python code is sent to a **Secure Sandbox Environment** (e.g., a Docker container) for execution .
6. **Synthesis & Response**: The Master Agent receives the data, performs any final processing or visualization, and streams the complete, human-readable answer back to the user's UI .