Project Specification: Autonomous Financial Risk & Compliance System

1. Executive Summary

This project aims to build an **Autonomous Research & Reasoning Agent System** designed to revolutionize financial command centers in the Real Estate & Construction industry. By leveraging **GraphRAG (Graph Retrieval-Augmented Generation)**, the system will move beyond simple data retrieval to perform complex reasoning, self-correction, and predictive risk analysis on construction financial data.

2. Core Objectives

- **Automate Financial Oversight:** Ingest and verify high volumes of construction financial documents (invoices, change orders, payment applications).
- **Proactive Risk Detection:** Identify liquidity risks, budget overruns, and compliance violations before they become critical issues.
- **Context-Aware Reasoning:** Use a Financial Knowledge Graph to understand the relationships between entities (contractors, projects, cost codes) and detect anomalies based on historical benchmarks.

3. System Architecture

The architecture is designed as a **Multi-Agent System (MAS)** where specialized agents collaborate to process data and generate insights.

A. Key Components

1. **Orchestrator (The Brain):** Manages the workflow, delegates tasks to sub-agents, and synthesizes final reports.
2. **Financial Knowledge Graph (The Memory):** A graph database (e.g., Neo4j) mapping relationships between:
    - *Projects ↔ Budgets*
    - *Contractors ↔ Contracts*
    - *Invoices ↔ Cost Codes*
3. **Vector Database (The Context):** Stores unstructured data (contract text, emails, meeting minutes) for semantic retrieval.

B. Agent Roster

Agent Name	Role & Responsibility	Key Tools

Document Intelligence Agent	**Ingestion & Extraction:** Extracts structured data (dates, amounts, line items) from unstructured PDFs (invoices, pay apps).	OCR (Tesseract/Azure Form Rec), Regex, JSON parsers.
Contract Compliance Auditor	**Validation:** Verifies if an invoice matches the agreed-upon contract terms (e.g., retention rates, unit prices).	Vector Search (RAG), Contract Database access.
AI Benchmarking Agent	**Contextual Analysis:** Compares current costs against historical project data to flag outliers (e.g., "Concrete is 20% more expensive than similar projects").	Graph Queries (Cypher), Statistical Analysis tools.
Cost-to-Complete Forecaster	**Prediction:** Predicts final project costs based on current spend rate and identified risks.	Time-series forecasting models, Budget API.
Critic / Supervisor Agent	**Quality Control:** Reviews the output of other agents for hallucinations or logic errors before finalizing the report.	Logic validation rules, Self-Correction prompts.

4. Workflow: The "Loop" Architecture

Unlike a linear pipeline, this system uses a **reasoning loop** to ensure accuracy:

1. **Ingest:** The *Document Intelligence Agent* receives a new invoice.
2. **Extract & Verify:** It extracts data and passes it to the *Contract Compliance Auditor*.
3. **Reason:** The *Auditor* checks against the contract. If a discrepancy is found (e.g., "Billing exceeds agreed cap"), it triggers the *Benchmarking Agent*.
4. **Contextualize:** The *Benchmarking Agent* checks if this overage is standard for this phase or an anomaly.
5. **Critique:** The *Critic Agent* reviews the findings. If the reasoning is weak, it sends the task back for re-analysis.
6. **Update:** Validated data updates the *Financial Knowledge Graph*.
7. **Alert:** The *Orchestrator* generates a risk alert for the human project manager.

5. Technology Stack

- **LLM Framework:** LangChain or LangGraph (for stateful multi-agent orchestration).
- **Knowledge Graph:** Neo4j (for modeling complex project relationships).
- **Vector Store:** Pinecone or Weaviate (for contract and document search).
- **Backend:** Python (FastAPI).
- **Frontend:** React / TypeScript (integrated into the PROBIS dashboard).

6. Next Steps

1. **Define Schema:** Finalize the node and edge types for the Knowledge Graph (e.g., (Contractor)-[SUBMITS]->(Invoice)).
2. **Build Pilot Agent:** Start by building the *Document Intelligence Agent* to reliably extract data from sample invoices.
3. **GraphRAG Prototype:** Implement a basic GraphRAG pipeline to query "Show me all high-risk invoices from Contractor X."

Phase 1: The "Knowledge Foundation" (Schema & Data)

Goal: Define how the AI understands the world of construction finance.

1. **Define the Graph Schema (Ontology)**
    - Map out your Nodes: `Project`, `Contractor`, `Contract`, `Invoice`, `BudgetLine`, `RiskFactor`.
    - Map out your Edges:
        - `(Contractor)-[:HAS_CONTRACT]->(Contract)`
        - `(Invoice)-[:BILLED_AGAINST]->(Contract)`
        - `(Invoice)-[:CONTAINS_ITEM]->(LineItem)`
    - *Deliverable:* A `.json` or `.cypher` file defining these relationships.
2. **Set Up the Hybrid Database**
    - **Graph Store (Neo4j):** Spin up a local Neo4j instance (Docker or Desktop). This stores the *structure* (who paid whom).
    - **Vector Store (Pinecone/Chroma):** Set this up to store the *unstructured text* (contract clauses, email threads).
    - *Deliverable:* A Python script that connects to both DBs successfully.
3. **Data Collection (The "Gold" Set)**
    - Gather 5-10 sample PDF invoices, 1 sample contract, and 1 project budget (Excel/CSV).
    - *Deliverable:* A folder of raw "test data."

Phase 2: The "Ingestion Engine" (PDF → Graph)

Goal: Turn raw documents into structured knowledge without human help.

1. **Build the "Document Intelligence" Agent**
    - **Tooling:** Use a library like `LlamaParse` or `Azure Document Intelligence` to extract text from PDFs.
    - **Task:** Convert a PDF Invoice into a clean JSON object (Date, Vendor, Amount, Line Items).
    - *Deliverable:* A Python script: `parse_invoice(pdf_path) -> json_data`.
2. **Build the "Graph Builder" Function**
    - **Logic:** Take that JSON and write Cypher queries to insert it into Neo4j.
    - **Constraint:** Ensure it checks for duplicates (e.g., *Merge* nodes instead of *Create*).
    - *Deliverable:* Running the script populates your Neo4j graph with the 5 sample invoices.

Phase 3: The "Brain" (Orchestration & Reasoning)

Goal: The core "Agentic" loop that finds risks.

1. **Design the LangGraph Workflow**
    - Set up the state machine: `Ingest` -> `Verify` -> `Critique` -> `Finalize`.
    - Define the "State" object (what data gets passed between agents).
2. **Develop the "Auditor" Agent (The GraphRAG Component)**
    - **Task:** Give it a tool to query Neo4j.
    - **Prompt:** "Check if Invoice #102 exceeds the remaining budget for 'Concrete' in Project Alpha."
    - *Deliverable:* The agent can query the graph and return "Yes, it exceeds by $500."
3. **Develop the "Critic" Agent (The Safety Net)**
    - **Task:** Review the Auditor's output.
    - **Prompt:** "Review the reasoning. Did the Auditor check for *pending* change orders before flagging this as an overage?"
    - *Deliverable:* A feedback loop where the Critic rejects a wrong answer and forces the Auditor to try again.

Phase 4: The "Command Center" (UI & Visualization)

Goal: Visualize the risks for the user.

1. **Build the Dashboard (Streamlit or Next.js)**

    - **View 1:** The "Risk Feed" (Alerts generated by the agents).
    - **View 2:** The "Graph Explorer" (Visualizing the specific nodes involved in a risk).
    - *Deliverable:* A web page where you can upload a PDF and see the resulting analysis.