

CloudRedux Assignment

Statement of Work (SoW)

Intelligent Procurement Agent

Stack: Google Vertex AI SDK / Agent Development Kit (ADK)

Model: Gemini Pro / Gemini Flash

Timebox: 48 Hours

Role Context: APA Engineer - Agentic Workflow Architecture

1. Objective

The objective of this work is to design and implement a **stateful, backend-only agent** that assists a Construction Site Manager with procurement decisions.

The agent must:

- Persist site-specific rules beyond a single conversation
- Apply those rules deterministically during execution
- Enforce approval limits via a **human-in-the-loop pause**
- Demonstrate clean separation between **reasoning, memory, and tools**

This system is **not a chatbot demo**, but a controlled agentic workflow aligned with enterprise constraints.

2. Understanding the Context

2.1 Business Context

Construction procurement decisions are governed by:

- Site-specific policies (vendor bans, approval thresholds)
- Cost controls requiring managerial approval
- Repeat interactions across time

Relying on LLM context alone is insufficient because:

- Context windows are transient
- Policies must persist across sessions

- Decisions must be auditable and enforceable

2.2 Technical Context

The agent operates in **two distinct interaction modes**:

1. **Memory Ingestion Mode**

The agent extracts and stores rules defined by the user.

2. **Reasoning & Execution Mode**

The agent retrieves stored rules, applies them to vendor data, and decides whether execution is allowed or must pause.

3. Scope of Work

In Scope

- Agentic workflow using Google ADK
- Persistent memory outside the LLM
- Deterministic vendor filtering logic
- Explicit pause/resume state for approvals
- Clean, modular Python codebase
- Demonstration via Loom video

Out of Scope

- UI or frontend
- Cloud infrastructure provisioning
- Authentication or real vendor APIs
- Performance optimization beyond clarity

4. System Design Overview

4.1 High-Level Architecture

The system is composed of **four independent layers**:

1. **Agent Layer (Reasoning & Orchestration)**

Controls flow and decision-making using Gemini via Vertex AI.

2. Memory Layer (Long-Term State)

Stores site-specific rules in a persistent store (JSON or local DB).

3. Tool Layer (Deterministic Functions)

Handles vendor querying and memory read/write.

4. State Layer (Execution Control)

Manages pause/resume for human approval.

This separation ensures:

- Predictable behavior
- Testability
- Clear responsibility boundaries

5. Phase-wise Execution Plan

Phase 1: Context & Rule Ingestion (Turn 1)

Input Example:

"For the Pune site, the maximum approval limit is ₹40,000 and we strictly avoid 'BadRock Cements'."

Responsibilities

- Extract:
 - Site name
 - Approval limit
 - Banned vendors
- Normalize and persist rules
- Store rules outside the LLM context

Memory Schema (Example)

{

 "Pune": {

 "approval_limit": 40000,

 "banned_vendors": ["BadRock Cements"]

```
}
```

```
}
```

Design Notes

- Memory is **explicit**, not implicit
- File I/O or local store is intentionally visible
- This phase does not perform procurement logic

Phase 2: Reasoning & Execution (Turn 2)

Input Example:

“Order 100 bags of cement for the Pune site.”

Execution Flow

1. Identify site from request
2. Retrieve site rules from Memory Store
3. Query vendor data (mock_vendors.json)
4. Filter vendors based on stored rules
5. Select lowest-cost valid vendor
6. Compare cost against approval limit

Decision Outcomes

- **If vendor is banned:** Reject explicitly
- **If cost ≤ limit:** Approve execution
- **If cost > limit:** Pause and await approval

This logic is implemented **outside the prompt**, ensuring transparency and control.

Phase 3: Human-in-the-Loop Control

When approval is required, the agent **must not proceed**.

Pause State Example

```
{
```

```
  "status": "AWAITING_APPROVAL",
```

```
"site": "Pune",  
"selected_vendor": "GoodRock",  
"amount": 42000,  
"reason": "Cost exceeds site approval limit"  
}  
}
```

Design Rationale

- Pausing is a **state**, not a failure
- This models real enterprise workflows
- Resume can be implemented later without redesign

6. Implementation Pattern

6.1 Repository Structure

```
/agent/  
    agent.py      # ADK agent orchestration  
    memory.py    # Persistent rule storage  
    tools.py     # Vendor querying & utilities  
    state.py     # Pause / resume handling
```

```
/data/  
    memory.json  
    mock_vendors.json  
main.py
```

6.2 Design Principles

- Business logic > prompt complexity
- Explicit state > implicit assumptions
- Readable code > clever abstractions

7. Platform Usage (Vertex AI & ADK)

Why Vertex AI + Gemini

- Native integration with ADK
- Managed LLM inference
- Enterprise-aligned tooling

How ADK is Used

- Define agent and tools
- Route reasoning through Gemini
- Maintain clean boundaries between logic and model

The LLM is treated as a **reasoning component**, not a rule engine.

8. Model Considerations (Contextual Note)

While recent models such as **Claude Opus 4.6** emphasize long-context reasoning, this implementation intentionally avoids reliance on extended context windows.

The system is designed so that:

- Memory persistence is **architectural**, not model-dependent
- The same workflow can be ported across models or platforms
- Model upgrades do not change system correctness

This aligns with enterprise requirements where **workflow stability matters more than model novelty**.

9. Testing Strategy

Functional Tests

- Rule ingestion correctness
- Vendor filtering accuracy
- Approval threshold enforcement

Scenario Validation

- Banned vendor rejection
- Cost-based pause trigger
- Correct memory retrieval across turns

Testing focuses on **behavior**, not token output.

10. Deliverables

1. GitHub Repository

- Modular, readable Python code
- Clear separation of concerns

2. Loom Video (≤ 5 mins)

- Memory design explanation
- Pause/resume logic walkthrough
- Live demo rejecting banned vendor and pausing execution

11. Success Criteria

This work is successful if:

- The agent enforces rules across turns
- Memory persists outside the LLM
- Human approval is required when limits are exceeded
- Architecture is easy to reason about and extend

Closing Note

This implementation prioritizes **clarity, control, and correctness** over surface-level sophistication.

The goal is not to show what the model can do, but to demonstrate **how agentic systems should behave in real enterprise environments**.