Part 1 – Persona & Interaction Design (Logic Layer)

1.1 Persona Definition (AI Co-worker as NPC)

Chosen AI Co-worker: Gucci Group CHRO

The CHRO is designed as a **group-level strategic stakeholder**, not an executional HR operator.

Key characteristics:

- Operates strictly at Group level (never brand-level)
- Anchors all responses in the Group Competency Framework:
    - Vision
    - Entrepreneurship
    - Passion
    - Trust
- Actively highlights trade-offs between group coherence and brand autonomy

Hidden constraints:

- Cannot override brand authority
- Cannot make final decisions
- Refuses to engage in generic HR advice or off-scope topics

This constraint-driven persona ensures the AI behaves like a real senior stakeholder rather than a generic assistant.

1.2 Dialogue Quality – Good vs. Bad Interaction

Good interaction (on-scope):
• User asks how leadership development can increase inter-brand mobility
• CHRO provides structured framing, surfaces risks, and references the competency framework
Bad interaction (off-scope):
• User asks for operational hiring steps at a single brand
• CHRO politely redirects, clarifying Group vs Brand responsibility
This distinction prevents the AI from becoming overly helpful in unrealistic ways.

1.3 State Awareness Over Time

Conversation history is preserved and injected into the system prompt.
While the prototype does not yet implement sentiment scoring or persona drift, the design allows:
• Escalation of skepticism if the user repeatedly ignores scope
• Supervisor-triggered nudges if the conversation stalls or loops

Part 2 – System Architecture (Engine Layer)

2.1 High-Level Architecture

The AI Co-worker Engine is designed as a modular, agent-based system that can be reused across different job simulations.

At a high level, the architecture follows a game-inspired NPC engine pattern:

- The **User Front-End** (web chat UI) sends user messages to a **FastAPI Orchestration Layer**.
- The orchestration layer first invokes a **Supervisor (Director) Agent**, which monitors scope, pacing, and conversation quality.

- If the request is valid, it is routed to a **role-specific AI Co-worker (NPC) Agent** (e.g., CHRO, CEO).
- The NPC Agent generates a response using an **LLM API**, guided by its persona, constraints, and explicit conversation state.
- The response is returned to the user through the orchestration layer.

This separation of concerns (UI → orchestration → supervision → NPC → LLM) makes the system scalable, testable, and easy to replicate across multiple simulations with different personas and rules.

2.2 Tool Use

Yes, AI Co-workers can be designed to access tools, but **only through the orchestration layer**, not directly.

Tools (e.g., fake JIRA ticket lookup, KPI calculator, prompt library) are exposed as controlled functions or adapters managed by the system.
The NPC Agent can request a tool via structured intent (e.g., "lookup_ticket"), but the **Supervisor or Orchestration Layer decides whether the tool call is allowed**.

Example:
- An "Engineering Lead" NPC may request access to a fake JIRA database.
- The orchestration layer validates the request, executes the tool, and injects the result back into the NPC's context.
- The NPC then explains or reacts to the result, rather than acting as a raw data retriever.

This design prevents unrealistic NPC behavior, enforces role boundaries, and keeps tool usage auditable and safe.

2.3 Latency vs. Quality Trade-off

Real-time chat requires low latency, while complex pipelines (e.g., RAG) increase response time and cost. To balance this trade-off, the system follows a **progressive complexity strategy**:

- **Default path:** Fast, persona-driven responses without retrieval, using cached persona prompts and conversation state.
- **Conditional RAG:** Retrieval is triggered only when the Supervisor Agent detects a knowledge-heavy or grounding-required query.
- **Caching:** Frequently used persona data, competency frameworks, and simulation context are cached to reduce repeated retrieval.
- **Graceful degradation:** If retrieval is slow or unavailable, the NPC responds with high-level framing instead of blocking the conversation.

This approach prioritizes responsiveness for most interactions while still allowing higher-quality, grounded answers when necessary.

Part 3 – Supervisor / Director Layer

The Supervisor Agent acts as an **invisible director**, inspired by game AI systems.
Responsibilities:
• Monitor conversation scope
• Detect looping or stalled dialogue
• Signal AI Co-workers to redirect or nudge the user
Example intervention:
• If the user repeatedly asks tactical questions, the supervisor signals the CHRO to reframe at a strategic level rather than hard-rejecting.
This preserves realism while keeping the simulation on track.

Part 4 – Prototype & Implementation Strategy

4.1 Technology Stack

- Python
- FastAPI for orchestration and API layer
- LLM API (model-agnostic)

4.2 Prototype Logic (Pseudocode)

The following pseudocode illustrates the core execution flow of the AI Co-worker Engine. It demonstrates how user input is monitored, routed, and handled by role-specific NPC agents under supervisor control.

```
async def run_simulation(persona_id, user_message, state):
supervisor = Supervisor()
signal = supervisor.monitor(user_message, state)


if signal["status"] != "OK":
return {
        "assistant_message": "Request is outside my role scope.",
        "state_update": None,
        "safety_flags": signal
}


agent = NPCAgent(persona_id)
reply = await agent.respond(user_message, state)


return {
        "assistant_message": reply,
        "state_update": state,
        "safety_flags": signal
}
```

```
class NPCAgent:
    def __init__(self, persona_id):
        self.persona_id = persona_id
        self.system_prompt = load_persona_prompt(persona_id)


    async def respond(self, user_message, state):
    prompt = build_prompt(
        system_prompt=self.system_prompt,
        state=state,
        user_input=user_message
    )
    return call_llm(prompt)
```

```
class Supervisor:
    def monitor(self, user_message, state):
        if violates_scope(user_message, state):
            return {"status": "BLOCK", "hint": "Out of role
        authority"}
        return {"status": "OK", "hint": None}
```

Prototype Behavior

The current prototype demonstrates:
- Persona-driven responses
- Explicit conversation state
- Supervisor-agent signaling

Outputs are plain text for clarity. In a production system, responses could include:
- State updates
- Safety flags
- Structured metadata

Evaluation Alignment & Conclusion

This solution focuses on:

- **Role-playing fidelity** through constraints and persona discipline
- **Architecture soundness** via modular agent design
- **Problem anticipation** using a supervisor layer

While not production-complete, the system demonstrates a clear and scalable approach to building AI Co-workers as interactive, goal-driven NPCs for job simulations.