AI Engineer — Senior-Level AI-Assisted Build Test (90 Minutes)

Focus

Practical AI systems, not research

Purpose

This test is intentionally designed to allow and encourage the use of AI tools such as: - ChatGPT / Claude - LangChain / LangGraph docs - Vector DB docs - Any public LLM tooling references

We are **not** testing ML theory, model training, or academic knowledge.

We are evaluating: - Ability to design real AI systems used by non-technical teams - Judgment in choosing appropriate AI techniques - Ability to combine LLMs + tools + data - Engineering trade-offs under time pressure - Readiness to ship internal AI tools

Timebox

⏱️ **90 minutes (strict)** - You are not expected to finish everything - Trade-offs are expected - Over-engineering is discouraged - **Correct architecture > perfect code**

Scenario

You are an AI Engineer at a company building internal AI tools to support: - Operations - HR - Marketing - Customer Support

The company already has core systems (POS, CRM, databases). Your job is to build AI tools that sit on top of existing data.

Critical context

- Multi-tenant B2B SaaS environment
- AI systems must **not leak data across tenants**
- **Cost control** matters (LLM calls are not free)
- AI output must be **explainable** to business users

Core feature to build (choose ONE)

You must implement a working backend service (**code required**) that demonstrates real AI engineering practices.

Choose one use case:

Option A — AI Resume Screening Tool (HR)

Input: Job description + multiple resumes

Output: - Candidate score - Strengths / weaknesses summary - Reasoning for score

Option B — Marketing Content Insight AI

Input: Past campaign data (CSV / text)

Output: - Key insights - Suggested next actions - Campaign improvement ideas

Option C — Internal Knowledge Assistant

Input: Internal documents (markdown / text)

Output: - Answer employee questions - Cite sources - Refuse to answer if context is missing

---

Mandatory technology requirements

Your solution must use **all** of the following: - **PostgreSQL** — system-of-record (jobs, documents, requests, results, audit) - **Vector database** — embeddings & semantic search - (Milvus, Qdrant, pgvector, Weaviate, etc.) - **Redis** — caching, short-lived state, or idempotency

Infrastructure must be runnable via **Docker Compose**.

You may stub external LLM APIs, but **data flow must be real**.

Task set — You have 90 minutes

You must complete **ALL of Section A** and **ANY TWO of Section B / C / D**.

Section A — Core AI System Design (Mandatory)

A1. Problem framing

Explain: - Who is the user? - What decision are they trying to make? - Why a normal rule-based system is insufficient?

Max 1 page.

A2. System architecture (required)

Provide a high-level architecture including: - API layer (FastAPI / ASP.NET / etc.) - LLM usage - Prompt layer - PostgreSQL schema (high-level) - Vector DB usage - Redis usage

Diagram or bullet list accepted.

A3. Data model (required)

Define minimal schemas for: - Tenant - Source data (resumes / documents / campaigns) - AI request / result

Show how **TenantId** is enforced.

A4. Prompt design (required)

Provide at least one real prompt you would use: - System prompt - User prompt - Output format (**structured JSON**)

Explain why you structured it this way.

Section B — Data, RAG & Safety

Choose **ONE**:

B1. RAG design

Explain: - How documents are chunked - How embeddings are stored - How retrieval is filtered per tenant

OR

B2. Hallucination & safety

Explain: - How your system reduces hallucinations - How it responds when it does not know

---

Section C — Cost, Performance & Evaluation

Choose **ONE**:

C1. Cost control strategy

Explain: - How you limit token usage - When you cache AI responses - When AI should NOT be used

OR

C2. Output evaluation

Explain: - How you measure AI output quality - How humans can give feedback

---

Section D — Multi-Tenant & Scale Thinking

Choose **ONE**:

D1. Tenant isolation strategy

Explain: - How prompts avoid cross-tenant leakage - How vector search is scoped

OR

D2. v1 → v2 evolution

Explain: - One decision that enables future expansion - One thing you intentionally delay

---

Section E — Execution reality check (lightweight)

Answer briefly: 1. What would you ship in 2 weeks? 2. What would you explicitly not build yet? 3. What risks worry you the most?

Required deliverables

```
/src
  /backend
  /infra
README.md
AI_PROMPTS.md
```

README.md (required)

Explain: - Your approach - Assumptions - Trade-offs - What you would improve with more time

Include a **Runbook** section with: - Prerequisites (Docker + Docker Compose) - One-command startup (e.g., `docker compose up --build`) - Environment variables and sample `.env.example` - Health checks / how to verify the system works - Example API calls (`curl`) demonstrating the core flow end-to-end

AI_PROMPTS.md (mandatory)

Include: - Exact prompts used - Iterations - Accepted vs rejected AI outputs - Why human judgment was required

---

Submission requirements (important)

- Push your solution to **GitHub**.
- The **repository must be PUBLIC**.
- Email us the **GitHub repository link**.

Your submission should allow another engineer to clone, run `docker compose up --build`, and verify the system end-to-end.

Rules

- Backend code is required (API + service layer)
- Any backend language except **Node.js**
- No frontend required
- Docker is required: the project must run via Docker/Compose on another machine
- Use docker compose to start dependencies (PostgreSQL, Redis, Vector DB)
- Optional: include a minimal local seed script to load sample data
- Pseudocode acceptable only for non-core parts
- Focus on correctness and architecture, not UI

Final instruction

We are not hiring research scientists.

We are hiring AI Engineers who can: - Build useful tools - Control cost & risk - Explain AI behavior to non-technical teams - Ship something real under pressure