AI Engineer Role

Interview Activity

Disclaimer: This activity is solely for interview purposes and must not be claimed as a novel idea or used for any other purpose by either party.

Confidentiality Note: All materials and discussions related to this activity are confidential and should not be shared or disclosed outside the interview process.

BLACK BOX

Role: AI Engineer - Full-Stack Agent Development

Exercise Overview

Scenario

Your organization wants to empower Project Managers (PMs) with an AI assistant to accelerate project initiation. Currently, PMs must manually sift through various documents (company policies, past project reports) and check multiple systems (HR directory, task trackers) to get a new project off the ground.

You are tasked with building a full-stack "ProjectPal" web application. This AI assistant will use **Mastra** agents to provide a single, conversational interface for these tasks. The assistant must be able to reason, use tools, and query internal knowledge.

The Solution Must Achieve the Following:

- **Full-Stack Application:** A working web app built with **Astro** (for the frontend) and **TypeScript** (end-to-end).
- **Knowledge Retrieval (RAG):** Comprehend and retrieve information from a knowledge base (e.g., policy documents, past project summaries).
- **Tool Use (Agentic Action):** Integrate with mock "internal systems" (e.g., find an available employee, create a project ticket).
- **Conversational AI:** Use a **Mastra** agent to orchestrate RAG, tool use, and LLM reasoning to answer complex, multi-step user queries.

ProjectPal: The AI Project Assistant

User & Current Workflow

Project Managers (PMs) currently spend hours:

1. Reading SharePoint for project management guidelines (PM_Handbook.pdf).
2. Checking an HR portal or spreadsheet to find available staff.
3. Manually creating kick-off tickets in a system like Jira or Asana.

Proposed Workflow with "ProjectPal"

The PM uses a simple chat interface and asks a natural language question. The Mastra agent does the work.

Example Query: *"I'm starting a new 'Global Payments' project. What's the required budget approval level according to the handbook, can you find me one available 'Engineering Lead', and create a 'Project Kick-off' task for them?"*

Task: Design & Build the AI Solution

You are tasked with designing *and* building a proof-of-concept for the "ProjectPal" application.

Part 1: Data, Tools, and RAG

Your agent will need to interact with two types of resources. You will mock these for the exercise.

- **RAG Data (Knowledge Base):**
    1. Create a simple `docs/PM_Handbook.pdf` or `.txt` file. It should contain sample text, e.g., *"All projects over $50,000 require VP approval."*
- **Tools (Internal Systems):**
    1. Implement these as simple TypeScript functions that the Mastra agent can call.
    2. `getTeamDirectory()`: A function that returns a hard-coded JSON array of employees and their status, e.g.:

```
[
  { "name": "Alice Smith", "role": "Engineering Lead", "status": "Available" },
  { "name": "Bob Johnson", "role": "Designer", "status": "On-Project" }
]
```

    3. `createProjectTicket(assignee: string, title: string)`: A function that simulates API-based ticket creation. It should just `console.log` the ticket and return a mock ticket ID, e.g., `{ "success": true, "ticketId": "PROJ-123" }`.
    4. Consider implementing RAG as a tool

Part 2: Full-Stack Application (Astro + Mastra)

Build a working, end-to-end TypeScript application.

- **Frontend (Astro):**
    - Create a simple, single-page web app.
    - It must have a chat input box and a display area for the conversation.
    - The frontend must call your backend API and handle streaming responses (to show the agent's thinking process or final answer).
- **Backend (TypeScript + Mastra):**
    - Use **Astro's API routes** or a simple **Node.js/Express** server (must be TypeScript).
    - Create a single API endpoint (e.g., `/api/chat`) that the frontend calls.
    - This endpoint must use **Mastra** to orchestrate the entire response.
    - **Mastra Agent:** Configure a Mastra `Agent` (or `workflow`).
    - **RAG:** Use Mastra's RAG capabilities (`Mastra.rag`) to load and query the `PM_Handbook.pdf` file.
    - **Tools:** Integrate the `getTeamDirectory` and `createProjectTicket` functions as tools for the Mastra agent.

Part 3: Architecture & Deployment (Conceptual)

In your design document, you must also describe how you would deploy this application in a scalable, secure way using the **Microsoft Azure** ecosystem.

- **LLM Integration:** Which LLM would you use and why? (e.g., Azure OpenAI Service).
- **Hosting:** Where would you host the Astro frontend and the Mastra backend? (e.g., Azure Static Web Apps, Azure App Service, Azure Functions).
- **Scalable RAG:** How would you replace the simple file-based RAG for production? (e.g., Azure AI Search).

- **Security:** How would you secure the application and its API endpoints? (e.g., Azure AD for user auth, API Management).

Delivery

This is a two-part submission.

1. Initial Submission (Submit within 5 days)

Prepare and submit two items:

1. **Comprehensive Word Document:**
    - A high-level conceptual design of your solution (addressing "Part 3").
    - A high-level architecture diagram for the *production* Azure deployment.
    - Rationale for your Azure service choices, focusing on cost, scalability, and security.
    - Identification of potential risks (e.g., agent hallucination, tool failure) and your mitigation strategies.
2. **Working Code Repository:**
    - A link to a public GitHub repository containing your complete, working **Astro + TypeScript + Mastra** application.
    - The repository **must** include a `README.md` with clear, step-by-step instructions on how to install dependencies and run the application locally.

2. Second-Level Presentation

If selected for the second-level interview, prepare a **15-minute presentation** to:

1. **Walk through the conceptual solution** (your architecture diagram and Azure rationale).
2. **Provide a live demo** of your working "ProjectPal" application.
3. **Showcase the code**, focusing on the Mastra agent configuration, how you defined the tools, and how the frontend streams the agent's response.

Evaluation Criteria

- **Problem Understanding & Solution Alignment:** Does your solution effectively solve the PM's problem?
- **Technical Feasibility & Code Quality:** Is the Astro/TypeScript/Mastra code clean, well-structured, functional, and idiomatic?
- **AI Implementation:** Is the Mastra agent correctly implemented? Does it successfully use both RAG and tools to answer multi-step queries? Can you implement RAG as a tool ?
- **Full-Stack Integration:** Does the application work end-to-end, with the Astro frontend and TypeScript backend communicating correctly?
- **Azure Expertise:** Depth of knowledge shown in your conceptual design for production deployment.
- **Scalability, Security, and Compliance:** Consideration of enterprise standards in your design document.
- **Communication and Presentation Skills:** Clarity and conciseness in explaining your solution and demoing your application.
- **Awareness of Risks:** Identification of potential challenges (e.g., ambiguous queries, prompt injection) and logical mitigation strategies.