AI Enterprise Platform

Interview Preparation Guide

Position: AI Implementation Engineer

Reference: R-10367735

Key Topics Covered:

- Project Overview and Architecture
- Generative AI and RAG Systems
- AI Governance and Compliance
- Technical Stack and Skills
- Cloud Infrastructure and DevOps
- System Design Decisions
- Implementation Challenges
- Best Practices and Learnings

1. Project Overview

What is this project?

This is an enterprise AI platform that shows how to add generative AI services to a company's existing systems. It includes AI governance tools to manage AI projects safely and comply with regulations. The platform uses modern cloud technologies and follows best practices for security and scalability.

Main Goals:

- Make it easy for companies to use AI while staying compliant with rules
- Provide a smart AI assistant that can answer questions using company documents
- Track and manage all AI projects in one place
- Automatically check AI projects for risks and compliance issues
- Show how different technologies (Python, .NET, React) work together

Key Features:

The platform has four main parts: a RAG-based AI assistant that answers questions using company documents, an AI governance system that tracks and assesses all AI projects, automated risk assessment that scores projects based on privacy and compliance factors, and a modern web dashboard where users can interact with all these features.

2. System Architecture

How is the system designed?

The system uses a microservices design where each part does one job well. This makes it easier to update parts without affecting the whole system. All services talk to each other through APIs.

Four Main Services:

1. React Frontend

This is what users see and click on. It's built with React 18 and shows dashboards, project lists, and the AI chat interface. It's responsive, meaning it works well on phones and computers.

2. Node.js API Gateway

This is the front door for all requests. It handles user login (authentication), decides which service should handle each request (routing), and prevents spam by limiting how many requests one user can make (rate limiting).

3. Python AI Service

This service runs the AI features. It connects to Azure OpenAI to use GPT-4, implements the RAG system to search documents before answering questions, and manages the AI assistant's memory and tools. Built with FastAPI for fast async operations.

4. .NET Governance Service

This tracks all AI projects in the company. It stores project information, calculates risk scores, checks compliance with policies, and maintains audit logs. Built with ASP.NET Core and Entity Framework.

Data Storage:

- SQL Server: Stores structured governance data and project information
- Cosmos DB: Stores flexible AI metadata and conversation history
- Pinecone: Vector database for semantic search of documents
- Redis: Caches frequently accessed data for better performance

3. Generative AI and RAG System

What is RAG and how does it work?

RAG stands for Retrieval Augmented Generation. Instead of just sending a question to the AI, we first search our documents to find relevant information, then give that information to the AI along with the question. This helps the AI give accurate answers based on company knowledge rather than making things up.

RAG Process Step by Step:

Step 1: Prepare Documents

Company documents (policies, manuals, reports) are broken into small chunks. Each chunk is converted into numbers (called embeddings) that represent the meaning of the text. These embeddings are stored in a vector database.

Step 2: User Asks Question

When a user asks something like 'What are our AI data privacy policies?', their question is also converted into an embedding.

Step 3: Search for Relevant Info

The system compares the question's embedding with all document embeddings to find the most similar chunks. This is called semantic search because it finds meaning, not just keywords.

Step 4: Generate Answer

The relevant document chunks and the question are sent to GPT-4. The AI reads the documents and creates an answer based on that information, with citations showing where the info came from.

Agentic AI Framework:

This is an advanced feature where the AI can use tools. For example, if you ask 'How many high-risk AI projects do we have?', the AI can call a database query function to get the actual number rather than guessing. The AI decides which tools to use and in what order.

Safety Measures:

- Content filtering prevents harmful or inappropriate responses
- Input validation stops prompt injection attacks
- All AI interactions are logged for audit purposes
- Output is checked to ensure it meets compliance requirements

4. AI Governance and Compliance

Why is AI governance important?

Companies using AI need to make sure they follow laws and regulations, protect customer data, avoid bias in AI decisions, and maintain trust. AI governance provides tools to track all AI projects, assess risks, and ensure compliance with company policies and regulations.

Project Registration System:

Every AI project in the company must be registered. This creates a central database of all AI initiatives. Registration captures important details: what the AI does, what data it uses, who is responsible, what stage it's in (development, testing, production), and when it was approved. This prevents shadow AI projects that no one knows about.

Automated Risk Assessment:

Each registered project gets automatically scored on multiple risk factors. The system looks at:

- **Data Privacy and Security:** Does it handle sensitive data like personal information or financial records?
- **Bias and Fairness:** Could the AI make unfair decisions about people?
- **Transparency:** Can we explain how the AI makes decisions?
- **Regulatory Compliance:** Does it follow laws like GDPR, HIPAA, or financial regulations?
- **Operational Risk:** What happens if the AI fails or makes mistakes?

Each factor is scored from 1 to 10. Projects with high scores in critical areas get flagged for additional review. The system provides specific recommendations to reduce each risk.

Policy Enforcement:

Companies can define AI policies such as 'All AI systems handling customer data must be reviewed quarterly' or 'AI models must be tested for bias before deployment'. The system automatically checks if projects comply with these policies and alerts teams when they don't.

Audit Trail:

Everything is logged: who created projects, when they were approved, what changes were made, what AI interactions occurred. This creates a complete history that auditors can review.

5. Technical Stack and Skills

What technologies were used?

This project uses multiple programming languages and frameworks to show real-world enterprise development where different teams might use different tools. All pieces work together through APIs.

Python (AI Service):

Python is perfect for AI and data work. FastAPI was chosen because it's fast, has automatic API documentation, and supports async operations. LangChain helps build the RAG system and manage AI interactions. Pydantic validates all data to prevent errors.

- **Why FastAPI?** Modern, fast, automatic documentation, type safety
- **Why LangChain?** Makes it easy to build RAG and agentic AI systems
- **Testing:** pytest for unit tests and integration tests with 80%+ coverage

.NET (Governance Service):

.NET is common in enterprises for backend services. ASP.NET Core 8 provides a robust framework for APIs. Entity Framework Core handles database operations with LINQ for clean queries. Strong typing and excellent tooling make it reliable for business-critical operations.

- **Why .NET?** Enterprise-ready, excellent performance, strong typing
- **Why Entity Framework?** Safe database operations, migrations, LINQ queries
- **Testing:** xUnit with mocking for comprehensive test coverage

React (Frontend):

React 18 with modern hooks creates a responsive, fast user interface. Vite provides instant hot reload during development. TailwindCSS allows rapid UI development with utility classes. The app works smoothly on desktop and mobile devices.

- **Why React?** Component reusability, large ecosystem, excellent performance
- **Why Vite?** Extremely fast development builds and hot reload
- **State Management:** React Context and hooks for clean state handling

Node.js (API Gateway):

Node.js Express serves as the entry point for all client requests. It handles authentication with JWT tokens, routes requests to the right backend service, implements rate limiting to prevent abuse, and provides centralized logging.

- **Why Node.js?** Non-blocking I/O, perfect for API gateway, same language as frontend

• **Middleware:** Authentication, rate limiting, logging, error handling

6. Cloud Infrastructure and DevOps

How is the application deployed?

The application is designed to run on cloud platforms (Azure or AWS) using containers and Kubernetes. This makes it scalable, reliable, and easy to update without downtime.

Containerization with Docker:

Each service is packaged in a Docker container. This means the service runs the same way on a developer's laptop, in testing, and in production. No more 'it works on my machine' problems. Multi-stage builds keep images small and efficient.

- Each service has its own Dockerfile
- Docker Compose allows running all services locally with one command
- Health checks ensure containers are running properly
- Resource limits prevent any service from using too much memory or CPU

Kubernetes Orchestration:

Kubernetes manages all the containers in production. It automatically starts new containers if one crashes, balances traffic across multiple instances of each service, scales services up or down based on load, and handles rolling updates with zero downtime.

- **Deployments:** Define how many copies of each service to run
- **Services:** Provide stable addresses for communication between services
- **Auto-scaling:** Add more instances when traffic increases
- **ConfigMaps and Secrets:** Manage configuration and sensitive data securely

CI/CD Pipeline:

GitHub Actions automates the entire deployment process. When code is pushed to GitHub, it automatically runs tests, builds Docker images, and deploys to the cloud if all tests pass.

Pipeline Steps:

1. Code is pushed to GitHub
2. GitHub Actions runs all unit tests
3. If tests pass, Docker images are built
4. Images are pushed to a container registry
5. Kubernetes pulls new images and updates services
6. Health checks verify the update succeeded

Multi-Cloud Support:

The design works on both Azure and AWS. Azure OpenAI or AWS Bedrock for AI, SQL Server or Aurora for databases, Azure Container Apps or AWS ECS for hosting. This prevents vendor lock-in.

7. Key Design Decisions

Why these architectural choices?

Microservices vs Monolith:

Decision: Use microservices architecture where each service handles one responsibility.

Reasoning: In an enterprise, different teams often work on different parts. Microservices allow the Python team to update the AI service without touching the .NET governance service. Each service can be scaled independently - if the AI assistant gets more traffic, we only scale that service. Services can be deployed independently, reducing risk of breaking the entire system.

Trade-offs: More complex than a single application, requires good API design, needs service discovery. Worth it for enterprise scale and team independence.

Database Strategy:

Decision: Use different databases for different needs (polyglot persistence).

Reasoning: SQL Server is perfect for structured governance data that needs transactions and complex queries. Cosmos DB is better for flexible AI metadata that changes often and needs global distribution. Pinecone specializes in vector search which is essential for RAG. Each database is optimized for its use case.

API Gateway Pattern:

Decision: All client requests go through a single Node.js API gateway.

Reasoning: Centralized authentication means login logic is in one place instead of duplicated across services. Rate limiting prevents abuse. The gateway can route requests to different backend versions for A/B testing. It provides a single entry point that's easy to monitor and secure.

Async vs Sync Processing:

Decision: Use async processing in the Python AI service, sync in .NET governance.

Reasoning: AI operations (calling OpenAI, searching vectors) can take seconds. Async allows handling many requests at once without blocking. Governance operations are typically fast database queries where sync code is simpler and sufficient.

Security Approach:

Decision: JWT tokens for authentication, API keys in Azure Key Vault, encryption everywhere.

Reasoning: JWTs are stateless and work well in distributed systems. Never hardcode secrets in code - Key Vault provides secure storage with access logging. Encrypt data at rest and in transit to meet compliance requirements.

8. Implementation Challenges and Solutions

What problems came up and how were they solved?

Challenge 1: RAG Context Window Limits

Problem: GPT-4 has a maximum context window. If we retrieve too many document chunks, we exceed the limit and the API call fails.

Solution: Implement smart chunking that retrieves the top 5 most relevant chunks by default. For complex questions, use a re-ranking step to prioritize the best chunks. Track token usage and truncate if necessary. Consider using Claude's longer context window for complex cases.

Challenge 2: Cross-Service Communication

Problem: Services need to talk to each other reliably, but one service might be down or slow.

Solution: Implement retry logic with exponential backoff. Use circuit breakers to fail fast if a service is consistently down. Add timeout limits to prevent hanging requests. Log all service calls for debugging. Consider message queues for async communication.

Challenge 3: AI Response Quality

Problem: Sometimes the AI gives generic answers or hallucinates information not in documents.

Solution: Improve prompts to explicitly instruct the AI to only use provided documents. Add citation requirements so every claim must reference a source. Implement confidence scoring where the AI rates its certainty. Use content filtering to catch inappropriate responses. Regular testing with real questions helps refine prompts.

Challenge 4: Risk Assessment Accuracy

Problem: Automated risk scoring might miss nuances that humans would catch.

Solution: Use the automated score as a first pass, but require human review for high-risk projects. Train a machine learning model on historical risk assessments to improve accuracy. Allow experts to override scores with documented reasoning. Regular calibration with compliance teams.

Challenge 5: Performance at Scale

Problem: As document collections grow and user traffic increases, search gets slower.

Solution: Implement caching for common queries using Redis. Use Kubernetes auto-scaling to add instances during peak times. Optimize vector database indices. Pre-compute embeddings in batch instead of on-demand. Monitor performance metrics and set up alerts for slow queries.

9. Interview Tips and Key Talking Points

How to talk about this project in interviews:

Start with Business Value:

Don't jump straight to technical details. Explain that companies need to use AI to stay competitive, but they also need to manage risks and comply with regulations. This project solves both problems by providing AI capabilities and governance in one platform.

Use the STAR Method for Examples:

Situation: Describe the context (e.g., 'Companies need AI but worry about compliance')

Task: Your goal (e.g., 'Build a system that provides AI safely')

Action: What you did (e.g., 'Implemented RAG with governance controls')

Result: The outcome (e.g., 'System can answer questions accurately while tracking all AI usage')

Key Talking Points by Topic:

On Python Skills:

'I used FastAPI to build the AI service because of its async support and automatic API docs. I implemented RAG using LangChain, which handles vector embeddings and document retrieval. The service uses Pydantic models for type safety and clear API contracts.'

On .NET Skills:

'The governance service uses ASP.NET Core 8 with Entity Framework for database operations. I used LINQ for complex queries and implemented dependency injection for testability. The risk assessment logic is separated into services for clean architecture.'

On RAG Implementation:

'RAG improves AI accuracy by giving it relevant context. Documents are split into chunks, converted to embeddings, and stored in a vector database. When users ask questions, we find similar chunks using semantic search and pass them to GPT-4 with the question. This grounds AI responses in actual company knowledge.'

On AI Governance:

'AI governance is critical for enterprises. We track all AI projects in a registry, automatically assess risk based on data sensitivity and regulatory requirements, enforce compliance policies, and maintain complete audit trails. This helps companies use AI confidently.'

On Cloud and DevOps:

'The system is containerized with Docker and orchestrated with Kubernetes for scalability. CI/CD pipelines automatically test and deploy changes. The architecture works on Azure or AWS, using services like Azure OpenAI, Cosmos DB, and Container Apps. Infrastructure as Code makes

deployments repeatable.'

Questions You Might Ask:

- How does your company currently handle AI governance and risk assessment?
- What AI/ML models are you currently using in production?
- What are the biggest challenges in implementing generative AI at your company?
- How do your teams currently collaborate on AI projects?
- What tools and frameworks does your engineering team prefer?