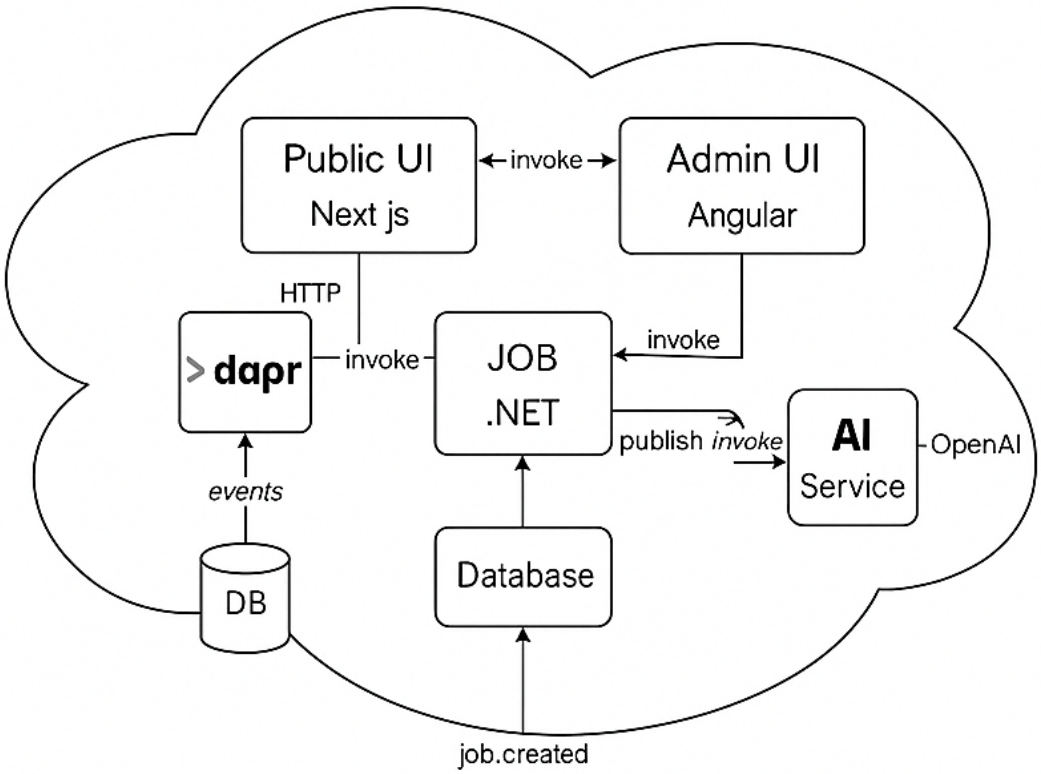
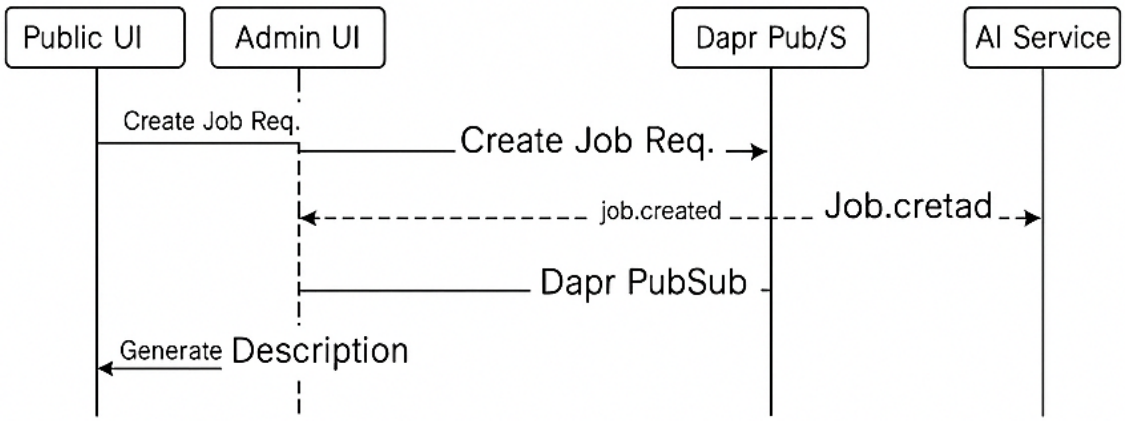
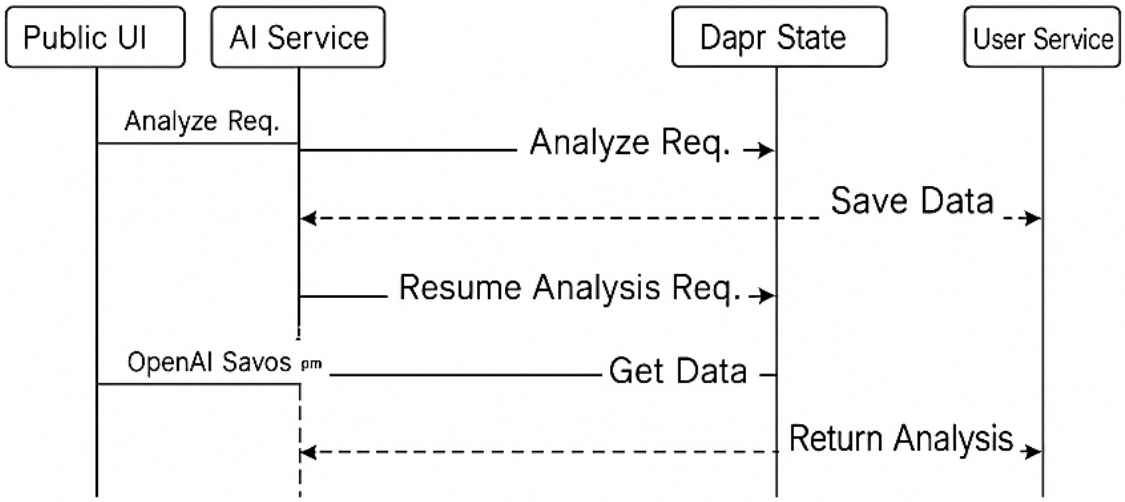
AI-Enhanced Job Board







Genentecy Job Description

Microservices Architecture: AI-Enhanced Job Board (Dapr + Azure + RabbitMQ)

Phase-by-Phase Implementation Plan

Phase 0: Environment Setup (Day 1-2)

- Install Docker Desktop, Dapr CLI, RabbitMQ, .NET SDK, Node.js, PostgreSQL/SQL Server.
- Set up a `docker-compose.yml` for all services.
- Initialize local Dapr components and secrets configuration.

Phase 1: Core Domain - Job Management (Week 1)

- Create `job-service` in .NET Core with EF Core + Dapr.
- Expose CRUD endpoints for jobs.
- Publish events like `job.created`, `job.applied` using Dapr pub/sub.
- Test locally using Postman or Thunder Client.

Phase 2: Public UI - Job Board (Week 1-2)

- Build Next.js public site with Tailwind.
- Fetch jobs from `job-service` via Dapr HTTP.
- Create job detail page and application form.
- Trigger `job.applied` event to backend.

Phase 3: AI Integration - Node Service (Week 2)

- Create `ai-service` in Express + OpenAI SDK.
- Create endpoints: `/generate-job-description`, `/analyze-resume`.

- Subscribe to `job.created` to auto-generate content.

- Return AI responses in structured format.

Phase 4: Admin UI - Angular Dashboard (Week 3)

- Build Angular app with Signals + Reactive Forms.

- Admins can create/edit jobs and trigger AI generation.

- Connect to `job-service` and `ai-service` via Dapr HTTP.

Phase 5: User Management - Auth and Profiles (Week 4)

- Build `user-service` with .NET Web API and EF Core.

- Handle registration, login, roles.

- Generate JWTs or integrate with Azure AD B2C.

- Protect endpoints in Next.js and Angular.

Phase 6: AI Resume Matching (Week 5)

- Upload resumes to `ai-service`.

- Analyze text via OpenAI and match to job listings.

- Return suggestions via public or admin UI.

Phase 7: Notifications & Events (Week 6)

- Use RabbitMQ to publish `resume.submitted`, `user.registered`.

- Build notification service for email/SMS alerts (optional).

- Store prompt and response logs in MongoDB or CosmosDB.

Phase 8: Cloud Readiness (Week 7+)

- Deploy microservices to Azure Container Apps or App Services.

- Move pub/sub to Azure Service Bus.

- Store secrets in Azure Key Vault.

- Deploy static frontends to Azure Static Web Apps.

Technology Breakdown by Service

Public UI (Next.js 14 + Tailwind) - Job seekers

- SEO-friendly job listings

- Apply with resume upload

- Resume analysis via AI service

Admin UI (Angular 17) - Employers & Admins

- Reactive Forms + Signals

- Manage job postings, review applicants

- Trigger AI-powered enhancements

Job Service (.NET + EF Core) - Jobs CRUD API

- Dapr-enabled

- Publishes job-created and application events

User Service (.NET + EF Core) - Auth, JWT

- Register/login

- Role-based access

- Integration with Angular and Next.js

AI Service (Node.js + OpenAI) - Resume/job analysis

- Handles GPT prompts for matching

- Subscribes to job creation events

Dapr + RabbitMQ - Event-driven integration

- Simplifies pub/sub between services

- Switch to Azure Service Bus in production

This document describes the overall flow and deployment strategy for a scalable, cloud-native job board using microservices architecture, AI, and modern frontends.