Architecture and Design Documentation

Purpose: Visualize system components and interactions.

Expanded Sections:

- **Component Diagram:**
  - User → Frontend → FastAPI → NLP Engine → Response
- **Sequence Diagram:**
  - JD/resume upload → text extraction → embedding → similarity → response
- **Deployment Diagram:**
  - Local dev → staging → cloud (Azure/GCP)
- **Design Principles:**
  - Modular backend
  - Stateless API
  - Scalable NLP pipeline

Purpose

The main goal is to visualize key system components and their interactions—helping both technical and business stakeholders understand how user actions translate into system processing and results. Visualization enables clarity for planning, troubleshooting, and scaling.

Component Diagram

The component diagram shows how main system blocks connect:

- User → Frontend: The user interacts with the platform through a web-based UI (built in HTML/CSS/JS). All requests and responses begin here.
- Frontend → FastAPI: The frontend securely sends user data (such as job description and resumes) to the backend. FastAPI orchestrates the core processing.
- FastAPI → NLP Engine: FastAPI passes extracted text to the NLP component—which uses Sentence Transformers for semantic analysis.

- NLP Engine → Response: The NLP engine produces match scores and explanations, which are returned to FastAPI and then relayed to the user through the frontend.

Each component is modular, allowing updates or maintenance without disrupting the entire flow. Can you picture how a user's action triggers a chain through these layers?

Sequence Diagram

A sequence diagram helps clarify the order of operations for a common user workflow:

1. JD/resume upload: The user submits files via the frontend.
2. Text extraction: FastAPI invokes file handlers (PyMuPDF, python-docx, Tesseract OCR) to extract raw text from the uploaded resumes and JD.
3. Embedding: The system encodes this extracted text into semantic vector embeddings using Sentence Transformers.
4. Similarity: Match scores are computed by measuring vector similarity between each resume and the job description.
5. Response: Results (ranked matches, optional chatbot output) are sent back to the frontend interface for user review.

This stepwise model ensures that every user request is processed predictably and transparently.

Deployment Diagram

Deployment diagrams illustrate how the system is set up and maintained across environments:

- Local Development: Developers build and debug features locally, running all components on their machines with mock data or isolated user accounts.
- Staging Environment: The code is deployed to a controlled environment replicating production, used for testing, bug fixes, and stakeholder demos.
- Cloud (Azure/GCP): Final production deployment happens on a scalable cloud platform. This ensures uptime, data security, and seamless updates—whether using Azure's App Service or Google Cloud Run, for example.

This multi-stage approach enables safe development and smooth production launches.

Design Principles

Every architectural decision is influenced by guiding design principles:

- Modular Backend: Core functions (file handling, NLP, scoring, chat) are separated into modules/services. This makes maintenance, upgrades, and testing far easier.
- Stateless API: FastAPI endpoints don't rely on in-memory session data. All requests contain the full required information (like files, IDs), supporting resilience and scalability for concurrent users.
- Scalable NLP Pipeline: NLP models are containerized and orchestrated (potentially with tools like Kubernetes). This allows parallel processing of multiple resumes, quick adaptation to heavier loads, and plug-and-play upgrades for newer AI models.