

Technical Design Document (TDD) & Software Development Plan (SDP)

TDD Expanded Sections:

- **Architecture Overview:**
 - Frontend: HTML/CSS/JS
 - Backend: FastAPI
 - NLP: Sentence Transformers
 - File Handling: PyMuPDF, python-docx, Tesseract OCR
- **Data Flow:**
 - Upload → Text Extraction → Embedding → Similarity → Ranking → UI Response
- **API Design:**
 - `/upload` → accepts JD and resumes
 - `/rank` → returns sorted match scores
 - `/chat` → handles UI updates

SDP Expanded Sections:

- **Development Phases:**
 - Phase 1: MVP (semantic matching + chatbot UI)
 - Phase 2: Skill gap analyzer
 - Phase 3: Q&A assistant and cover letter generator
- **Milestones:**
 - UI prototype Backend API integration
 - NLP embedding and scoring
 - Beta release
- **Tools & Tech:**
 - GitHub, VS Code, Postman, Docker

Technical Design Document (TDD)

The Technical Design Document defines the technical architecture, data flow, and integration mechanisms used to implement the product. It ensures all components—from user interface to NLP pipeline—work cohesively to deliver fast, accurate, and scalable performance.

Architecture Overview

The system is designed with a modular architecture allowing independent updates to the frontend, backend, and NLP components without disrupting overall functionality.

- Frontend (HTML/CSS/JS):
The web interface is built using HTML, CSS, and JavaScript for a clean, responsive, and intuitive user experience. It provides drag-and-drop resume uploads, interactive chatbot panels, and visual representation of match scores. Frameworks such as React or vanilla JS can be used to ensure dynamic rendering and smooth UX transitions.
- Backend (FastAPI):
The backend is powered by FastAPI, chosen for its asynchronous capabilities, lightweight design, and fast performance. It exposes RESTful endpoints for uploads, ranking, and chat interactions. FastAPI also handles request validation, background tasks for document processing, and secure API transactions.
- NLP Layer (Sentence Transformers):
The NLP core leverages Sentence Transformers to compute semantic embeddings of resumes and job descriptions. This model translates text into vector representations capturing contextual meaning—enabling accurate similarity searches and ranking. Fine-tuning can be performed using domain-specific HR datasets to improve contextual relevance.
- File Handling (PyMuPDF, python-docx, Tesseract OCR):
The system supports multiple file types including PDF, DOCX, and scanned documents.
 - PyMuPDF extracts structured text and metadata from PDFs.
 - python-docx handles Microsoft Word documents by reading text and maintaining formatting.
 - Tesseract OCR is used for image-based or scanned resumes to convert them into machine-readable text.

Data Flow

The core workflow is designed around a sequential yet efficient data pipeline:

1. Upload – Users upload a job description and candidate resumes via the UI.
2. Text Extraction – The backend processes files, extracting raw text using appropriate parsers or OCR.
3. Embedding – Both the job description and resumes are encoded into vector representations using Sentence Transformers.
4. Similarity Scoring – The system computes cosine similarity between JD and resume embeddings to assess match quality.
5. Ranking – Candidates are sorted by similarity scores, forming a ranked list that highlights the top fits.
6. UI Response – Results are displayed dynamically through the frontend interface with visual match indicators, chat prompts, and optional feedback mechanisms.

API Design

The backend exposes modular, REST-compliant APIs to ensure ease of integration and clarity in data exchange.

- `/upload` – Accepts job descriptions and resume files. Triggers text extraction and temporary file storage.
- `/rank` – Processes embeddings, computes similarity scores, and returns a JSON response containing the ranked list of resumes with match percentages.
- `/chat` – Powers the interactive chatbot interface. Handles user messages, guides them through the screening process, and fetches real-time results or explanations (e.g., “Why is this candidate ranked higher?”).

Software Development Plan (SDP)

The Software Development Plan outlines the strategy, phases, milestones, and tools required to build and ship the product efficiently and iteratively.

Development Phases

- **Phase 1: MVP (Semantic Matching + Chatbot UI)**
 - Core focus: implementing resume-JD matching and an interactive chatbot interface.
 - Deliverables: upload API, semantic similarity engine, real-time chat feedback, and basic UI visualization.
 - Goal: deliver a functional proof of concept enabling rapid stakeholder validation.
- **Phase 2: Skill Gap Analyzer**
 - Enhances the MVP by introducing analytics that identify missing or partially matched skills.
 - Visualizes insights such as “Top 5 missing skills” or “Key overlapping skills.”
 - Integrates a data-driven recommendation engine for improving candidate-job alignment.
- **Phase 3: Q&A Assistant and Cover Letter Generator**
 - Extends the chatbot’s capabilities to answer HR queries (e.g., “Show best candidates for Java roles”) and generate personalized cover letters using large language models.
 - Focuses on enhancing user engagement and automation depth.

Milestones

Key progress checkpoints are defined to ensure timely delivery and quality.

Milestone	Description	Expected Outcome
UI Prototype	Design interactive web UI mockups	Approved visual direction and layout
Backend API Integration	Connect frontend and backend workflow	Smooth data exchange between components
NLP Embedding and Scoring	Implement and validate the semantic model	Accurate candidate-job relevance calculation
Beta Release	Deliver functional version for limited stakeholder testing	Feedback on usability and accuracy

Tools & Technologies

The development ecosystem is centered around efficient collaboration, containerization, and reproducibility.

- GitHub – Version control, issue tracking, and pull request workflows for collaboration.
- VS Code – Primary IDE for coding, debugging, and live testing.
- Postman – API testing and validation during backend development.
- Docker – Containerization for reproducible environments and seamless deployment.

