Shashank – 4-Day Production Sprint

Theme: From Resume Parsing → Recruiter/Client Ready AI Platform

Total Time: ~4 Days

Day 1 – Semantic Resume Enrichment + Job Match Basics

Goal: Move beyond regex → embed intelligence into resume parsing.

Tasks:

- Integrate a sentence embedding model (SBERT / HuggingFace / OpenAI).
- Add semantic extraction for: skills, roles, industries, projects.
- Implement resume ↔ job description similarity scoring (cosine similarity).
- Store enriched candidate profiles in structured DB (Mongo/Postgres).
- Keep CSV export (for recruiters who like Excel).

Values Reflection (Mandatory Log):

- Humility: Write what regex couldn't capture & how embeddings solved it.
- Gratitude: Acknowledge the open-source model creators/tools.
- Honesty: Mention any biases/limitations noticed in results.

Deliverable:

- Candidate profiles with both regex + embedding-based fields.
- Fit score (%) when given a sample job description.

Day 2 – Recruiter Dashboard + API Layer

Goal: Build recruiter-facing tools to explore candidates.

Tasks:

- Create a recruiter web dashboard (FastAPI + simple React or Streamlit).
- Features:
    - Search/filter by skill, experience, location.

    - Sort by candidate-job match score.
    - Candidate profile cards (summary view).
- Expose APIs:
    - /upload_resume → parse + store candidate.
    - /get_candidates → query candidates with filters.
    - /match_job → return ranked candidates for JD.

Values Reflection:

- Humility: Note UI/UX issues faced (not everything can be polished).
- Gratitude: Recognize inspiration (e.g., competitor UIs, tutorials).
- Honesty: If shortcuts taken (hardcoding, sample data), write it down.

Deliverable:

- Recruiter dashboard running locally (with Docker).
- API endpoints functional with Postman tests.

Day 3 – Client Portal + Dynamic Scoring

Goal: Add client-facing side + improve scoring logic.

Tasks:

- Build client portal (separate login/view):
    - Client can post job description.
    - See ranked candidate list (from recruiter DB).
- Enhance scoring logic:
    - Blend rule-based weights (location, experience years, certifications).
        - Semantic similarity.
    - Output: transparent "Why matched" explanation.
- Add batch resume upload (drag-and-drop or folder scan).

Values Reflection:

- Humility: Share what was hardest in balancing recruiter vs client needs.
- Gratitude: Reflect on team inspiration (Talah’s RL work, your own foundation).
- Honesty: Log gaps (e.g., not secure yet, scoring rules subjective).

Deliverable:

- Recruiter + client portals functional.
- Candidate ranking system that explains scores.

Day 4 – Integration, Deployment & Final Reflection

Goal: Make platform deployable, production-like, and values-aligned.

Tasks:

- Dockerize entire system (APIs + dashboard + client portal).
- Deploy on AWS/GCP free tier (at least demo deployment).
- Connect monitoring/logging (basic stats on resumes processed, errors).
- Polish docs (README.md, PROJECT_STRUCTURE.md).
- Final Reflection.md update:
    - Humility: Acknowledge limits (not fully secure, scaling next step).
    - Gratitude: Thank the team, mentors, open-source tools.
    - Honesty: Share real state vs ideal vision.

Deliverable:

- Cloud-deployed demo link.
- Final Reflection.md (with Humility, Gratitude, Honesty entries).
- Working recruiter + client platform with semantic scoring.

Success Metrics

- Functional: Resume upload → recruiter dashboard → client portal → scoring works.
- Usability: Recruiter can search/filter, client can post JD and see candidates.

- Deployment: At least one demo deployment (AWS/GCP/Docker Hub).
- Values: Daily reflection updates, not skipped.