AI Resume Shortlister – Functional Requirements Document (FRD)

Version: 1.0
Owner: Recruitment Solutions Team
Date: 23 Aug 2025

---

1) Executive Summary

We are building an internal AI tool to help non-technical recruiters rapidly shortlist high-quality technical IT candidates in the USA. The system uses a Retrieval-Augmented Generation (RAG) pipeline on an actively maintained resume repository (S3 → embeddings → FAISS/Chroma) to match each client Job Description (JD) and return the top 20 resumes with transparent, explainable scores. The tool reduces screening time, improves matching quality at the skill-and-experience level, and flags potentially fake or manipulated resumes.

---

2) Business Goals & Pain Points

Goals - Reduce recruiter effort per requisition and time-to-shortlist. - Improve precision and recall of matching for technical skills, seniority, and domain. - Provide transparent reasons for match quality (matched vs. missing skills), and automated profile reports. - Detect and de-prioritize fraudulent or inflated resumes.

Pain Points Addressed - Difficulty finding best-fit resumes against complex, technical JDs. - “Fake” resumes or manufactured profiles that waste recruiter and client time. - Manual, time-consuming resume triage with low consistency.

---

3) In-Scope / Out-of-Scope

In Scope - Secure storage and ingestion of resumes in AWS S3. - Parsing and normalization of resumes (PDF/DOCX) into structured profiles. - Candidate skill/experience extraction and normalization (skills ontology). - Embedding and vector indexing with FAISS or Chroma, hosted on ECS/EC2. - JD intake (text or file), constraint capture (must-have, years, location, visa, rate). - Matching, re-ranking, explainable scoring, and anti-fraud signals. - Recruiter UI (React/Next.js) for search, filters, and profile report generation. - Download/export of resumes and profile reports; audit logging.

Out of Scope (v1) - Automated outreach to candidates. - Full ATS integration (provide APIs; native connectors planned later). - Live coding assessments; external background checks (future roadmap).

---

4) Users & Personas

- **Non-technical Recruiters (Primary):** Create searches, review top-20 results, download resumes, share reports.
- **Recruitment Leads:** Configure weights, review audit logs, manage taxonomies, approve fraud rules.
- **ML/Ops Engineers:** Operate ingestion, embeddings, and indexes; monitor and optimize.
- **Compliance/SecOps:** Review access controls, PII handling, data retention.

---

5) Functional Requirements

5.1 Resume Ingestion & Storage

1. System shall accept resume uploads via UI, bulk S3 upload, and email ingestion (optional v1.1).
2. New/updated files shall be stored in **S3 Landing** with server-side encryption (SSE-KMS).
3. S3 event triggers shall invoke a **Lambda Orchestrator** that enqueues parsing jobs.
4. Orchestrator shall write ingestion status and metadata to **DynamoDB** (or **RDS** if relational needed).

5.2 Parsing & Normalization

1. Parser service (container on ECS/EC2) shall support PDF and DOCX.
2. Extracted fields: name, contact redacted view, education, employers, titles, dates (start/end), skills, certifications, locations, visa status (if present), clearance, summary.
3. Normalize titles (e.g., Software Engineer → SWE), skills (map to ontology), and durations (compute years per skill from experience spans and evidence snippets).
4. Maintain original file and parsed JSON side-by-side in **S3 Curated**.

5.3 Deduplication & Fraud Signals

1. System shall compute content hashes and fuzzy signatures to detect duplicates/near-duplicates.
2. Signals (non-exhaustive): chronology inconsistencies, overlapping full-time roles, impossible seniority, copy-paste fragments across unrelated resumes, excessive LLM-like phrasing density, mismatched locations, unverifiable employers.
3. Assign a **Fraud Risk Score** per resume; use to re-rank or flag.

5.4 Embeddings & Vector Index

1. For each resume and for atomic sections (skills, experiences, summaries), compute text embeddings (Python service on ECS/EC2).
2. Persist vectors in **FAISS** or **Chroma** hosted on ECS/EC2 with EBS volumes; snapshot indexes nightly to **S3**.
3. Expose k-NN search (cosine/inner product) API.
4. Maintain versioned indexes per skill taxonomy version to avoid drift.

5.5 JD Intake & Constraint Capture

1. Recruiter can paste text or upload JD file; system extracts skills, seniority, domain, tools, certs, responsibilities, and constraints.

2. UI shall support toggles: **Must-have** skills, **Nice-to-have** skills, years of experience per skill, **US work auth**, **onsite/remote**, **location radius**, **rate/salary bands**, **clearance**.

5.6 Matching & Re-Ranking

1. Build a structured query from JD + constraints.
2. Retrieve top-K candidates via vector search (K=200 default) on combined fields.
3. Apply a **weighted re-ranker** to produce Top-20 with scores:
4. Skill match (semantic + exact),
5. Experience years per skill (from dated spans),
6. Recency of relevant work,
7. Domain/industry alignment,
8. Location fit & work authorization,
9. Fraud risk penalty,
10. Client-specific boosts (whitelist universities/certs if configured).
11. Provide **explanations**: matched evidence snippets with source lines; list **missing skills**.

5.7 Profile Report Generation

1. For each shortlisted candidate, generate an **HTML/PDF profile report** with:
2. Overall match score and rank,
3. Matched skills (with evidence & years),
4. Missing skills & gaps,
5. Roles timeline (Gantt-like),
6. Location/work auth/clearance,
7. Fraud signals (if any),
8. Download links to original resume (PII-safe view),
9. Recruiter notes.
10. Batch export: a single PDF/ZIP bundle for Top-20.

5.8 Recruiter UI (React/Next.js)

1. JD input pane with skill extraction preview and editable chips.
2. Filters drawer (must-have skills, experience sliders, location radius, visa/work auth, clearance, pay bands, availability).
3. Results grid (Top-20): score, provenance badges, quick evidence tooltip, fraud badge, download, and **Create Report** button.
4. Candidate detail page with evidence, skills timeline, and feedback (“Good fit”, “Maybe”, “Reject”, reasons).
5. **Compare** up to 5 candidates side-by-side.
6. Accessibility: WCAG 2.1 AA.

5.9 Feedback & Learning

1. Capture interactions: clicks, dwell, downloads, client interview outcomes (if integrated) to **Feedback Store**.
2. Use feedback to recalibrate weights (offline job); admins can A/B test strategies.

5.10 Security, Privacy & Compliance

1. SSO via **Amazon Cognito** (OIDC/JWT); role-based access.
2. PII handling: encryption at rest (S3 SSE-KMS, EBS), in transit (TLS 1.2+), redaction in UI for external sharing.
3. Audit logs for all access and downloads (immuutable in S3 with Object Lock, Governance mode).
4. Data retention & deletion policies configurable per client and candidate consent.

5.11 Observability & Operations

1. Structured logs (JSON) to **CloudWatch**; metrics and traces (OpenTelemetry) to a dashboard.
2. Health checks for all services; autoscaling rules for ECS services based on CPU/RAM/QPS.
3. Daily index snapshot verification; disaster recovery runbook (RTO $\leq$ 4h, RPO $\leq$ 24h).

---

6) Non-Functional Requirements (NFRs)

- **Latency:** < 3s P95 JD→Top-20 for indexes $\leq$ 5M resumes; < 6s for $\leq$ 20M.
- **Throughput:** 50 concurrent recruiters sustained; burst to 200.
- **Availability:** 99.9% monthly.
- **Scalability:** Horizontal scaling of embedding workers and FAISS/Chroma shards.
- **Cost Targets:** <$0.05 per resume ingested; <$0.01 per JD query at 5M scale.
- **Reliability:** Idempotent ingestion; exactly-once index updates.
- **Security:** Least-privilege IAM; private subnets; VPC endpoints to S3; WAF on public endpoints.

---

7) Data Model (v1)

Resume (JSON) - resume_id, s3_uri_original, s3_uri_parsed, candidate_hash, name (masked), contacts (masked), locations[], work_auth, education[], experiences[], skills[], certifications[], fraud_score, version, timestamps.

Experience: employer, title_norm, start_date, end_date, responsibilities_text, skills_evidence[] (skill, evidence_span, confidence), location.

Skill: name_norm, taxonomy_id, years_estimated, last_used_date, evidence_refs[]

JD Query: jd_id, text, skills_must[], skills_nice[], years_per_skill{}, location, radius, work_auth, clearance, pay_band, client_id.

MatchResult: jd_id, resume_id, score_overall, scores_breakdown{}, matched_skills[], missing_skills[], evidence_snippets[], rank.

---

8) API (FastAPI examples)

- `POST /jd/parse` → Extract skills/constraints from JD text or file.
- `POST /search` → body: JD Query → returns Top-K list with scores and explanations.
- `GET /candidate/{id}` → resume profile (masked PII).
- `POST /report` → generate profile report(s) for ids.
- `POST /ingest` → upload/bulk load webhook.
- `GET /health` → liveness/readiness.
- Admin: `POST /weights`, `GET /audit`, `POST /taxonomy`.

---

9) Matching & Scoring (Reference)

Overall Score (0–100)

```
score = 100 * [
  w_sem * sim_semantic(JD_text, resume_corpus)
+ w_exact * sim_exact(skills_must, skills_resume)
+ w_years * match_years(years_required, years_estimated)
+ w_recency * f_recency(last_used_dates)
+ w_domain * sim_domain(JD_domain, resume_domain)
+ w_geo * f_geo(JD_location, candidate_locations)
+ w_auth * f_auth(JD_auth, resume_auth)
- w_fraud * fraud_score_normalized
]
```

- Default weights: `w_sem=0.25, w_exact=0.20, w_years=0.15, w_recency=0.10, w_domain=0.10, w_geo=0.10, w_auth=0.05, w_fraud=0.05` (admin-tunable).
- Explanations include top contributing terms and evidence spans with provenance.

---

10) RAG Pipeline Details

1. **Retriever(s):** hybrid vector + keyword. Separate sub-indexes: skills, experiences, summaries.
2. **Augmenter:** assemble top chunks (evidence-ranked) and structured fields.
3. **Generator (optional):** create readable summaries and the profile report; enforce grounded citations back to resume.
4. **Guardrails:** JD privacy; redaction; profanity and PII checks on generated text; report templates.

---

11) AWS Architecture (Reference)

- **S3** (Landing, Curated, Snapshots) with KMS keys.
- **Lambda**: Orchestration on S3 events, lightweight transforms.

- **ECS on EC2**: Parser, Embedder, RAG Service, Vector DB (FAISS/Chroma), Report Generator.
- **EC2 Auto Scaling Groups**: for compute-intensive workloads.
- **API Access**: ALB → FastAPI; optional API Gateway.
- **DynamoDB** (or RDS PostgreSQL) for metadata, statuses, and feedback.
- **CloudWatch** for logs/metrics; **OpenTelemetry** traces.
- **Cognito** for AuthN/Z; **WAF** for edge protection; VPC private subnets & endpoints.

High-Level Architecture Diagram
See downloadable PNG: *diagram_high_level_architecture.png*.

Ingestion & Indexing Flow
See downloadable PNG: *diagram_ingestion_indexing_flow.png*.

Matching & RAG Flow
See downloadable PNG: *diagram_matching_rag_flow.png*.

---

12) Deployment & Environments

- **Environments:** Dev → Staging → Prod (separate AWS accounts).
- **CI/CD:** GitHub Actions → ECR → ECS blue/green; infra as code (Terraform/CDK).
- **Config:** SSM Parameter Store for secrets; KMS for encryption.
- **Backups:** S3 + EBS snapshots; daily FAISS/Chroma snapshot to S3.

---

13) Testing & QA

- **Unit tests:** parsing, normalization, scoring math, API handlers.
- **Integration tests:** end-to-end ingestion → search → report.
- **IR (Information Retrieval) eval:** Precision@K, Recall@K, nDCG on labeled JD↔resume pairs.
- **Adversarial tests:** fake resume detection cases, prompt-injection in content.
- **Load tests:** P95 latency at target scale; autoscaling verification.
- **Security tests:** IAM least privilege, S3 policies, OWASP ASVS, dependency scans.

---

14) Risks & Mitigations

- **Embedding drift / taxonomy drift:** versioned indexes; backfill jobs.
- **False positives in fraud detection:** keep as re-ranking penalty; require human confirmation.
- **PII exposure via reports:** default redaction; separate secure download links; short-lived pre-signed URLs.
- **Cost overruns:** autoscaling with budgets/alerts; batch embedding windows.

---

15) Milestones (Indicative)

1. Week 0–2: Infra scaffolding, S3/Lambda/ECS baseline, FastAPI skeleton, UI wireframes.
2. Week 3–5: Parsing & normalization MVP; embeddings/FAISS shard; JD parser and search API.
3. Week 6–8: Re-ranker, explanations, Top-20 UI; reports v1; observability.
4. Week 9–10: Fraud signals v1; security hardening; IR evaluation and tuning.
5. Week 11–12: UAT, docs, handover, go-live.

---

16) Glossary

- **RAG:** Retrieval-Augmented Generation.
- **FAISS/Chroma:** Vector index libraries/databases for similarity search.
- **nDCG/Precision@K:** Ranking quality metrics.

---

17) Appendices

- **A. Diagram Downloads:** See links in the delivery message for PNGs.
- **B. Resume Skills Ontology:** seed list + normalization rules (maintained by admins).
- **C. Report Templates:** Admin-editable handlebars/Jinja templates.

---

18) Related/Supporting Documents to Produce

1. **Product Requirements Document (PRD).**
2. **System Requirements Specification (SRS).**
3. **High-Level Design (HLD)** and **Low-Level Design (LLD)** with sequence diagrams.
4. **Data Governance & PII Handling Plan** (classification, retention, DLP, redaction).
5. **Threat Model & Security Plan** (STRIDE, IAM matrix, KMS key policy, WAF rules).
6. **IR Evaluation Plan** (labeling guidelines, gold set creation, metrics definitions).
7. **Taxonomy & Normalization Guide** (skills dictionary, title mapping, synonyms).
8. **API Contract** (OpenAPI/Swagger for FastAPI; versioning policy).
9. **UI/UX Wireframes & Style Guide** (Figma; accessibility checklist).
10. **Runbook & On-call Playbooks** (SLOs, alerts, incident response).
11. **CI/CD & IaC Guide** (Terraform/CDK modules, environments, rollback).
12. **Cost Model & FinOps Plan** (unit economics, scaling scenarios, budgets/alerts).
13. **Testing Strategy & Test Cases** (unit/integration/load/security/adversarial).
14. **Data Quality SLAs** (parsing accuracy targets, dedupe thresholds, monitoring).
15. **Compliance Addendum** (candidate consent, US privacy regs, record of processing).

Notes

- The FRD focuses on AWS components: **S3, FAISS, Chroma, RAG pipeline, ECS, EC2, Python, Lambda, FastAPI, React/Next.js** as requested.
- Bedrock/Comprehend/OpenSearch are intentionally optional; can be added in future design iterations if needed.