Functional Requirements

Scope of System

Core system comprises the following subsystems:

- `Resume Processing Service` (AI-powered parsing)
- `Job Template Service` (AI-assisted template creation)
- `Intelligent Matching Engine` (semantic analysis and scoring)

- `Human Review Interface` (score overrides, intangible notes)
- `Bias Detection Service` (monitoring for bias in job descriptions)
- `Analytics Engine` (HR insights dashboard)
- AI Integration Points (Chatbot/AI prompts, model calls, explainability)
- AI Usage Strategy (AI as development tool, prompt engineering, etc.)

System Context and Reference Architecture

Core Tech Stack:

Backend: `Java` with `Spring Boot` + `Spring AI`

AI Platform: `AWS Bedrock` (Claude Sonnet 4 + Haiku)

File Processing: `Apache Tika` for multi-format resume parsing

Frontend: `Angular` (mobile-responsive)

Database: `PostgreSQL or local file storage (if no time for integrating DM)`

Core System Components and responsibilities:

`Resume Processing Service:` AI-powered parsing to extract structured candidate data.

`Job Template Service:` AI-assisted creation and versioning of job templates.

`Intelligent Matching Engine`: Semantic analysis and scoring against job templates.

`Human Review Interface`: Enables manager overrides and capture of intangibles.

`Bias Detection Service`: Monitors language in job descriptions and candidate materials.

`Analytics Engine`: HR dashboards and data exports.

blocked URL

End to End Call Flow Overview

Functional Requirements (by subsystem)

Resume Processing Service (`Resume Processing Service`)

F1.1 Resume Ingestion

- The system must accept resumes in multiple formats (PDF, DOC, DOCX, TXT) via batch upload, API, or front-end drag-and-drop.

F1.2 Structured Data Extraction

- Given a resume, the service must extract structured data fields including: candidate name, contact information, education, professional experience (titles, companies, dates, locations), skills (explicit and inferred), certifications, and publications if present.

F1.3 Parsing Accuracy and Confidence

- The parser must return a confidence score per field to indicate extraction reliability; the system should surface low-confidence fields to human reviewers for validation.

F1.4 Multi-Resume Batch Processing

- The service must support batch ingestion of up to a defined batch size (e.g., 50 resumes for the demo) with progress tracking and re-try logic for failed parses.

F1.5 Data Normalization

- Normalize job-relevant terms (e.g., skill naming, acronyms, date formats) to a canonical representation for downstream processing.

F1.6 Auditability

- All parsing actions and outcomes must be logged with a user/system timestamp, resume identifier, and deterministic hash for traceability.

F1.7 Error Handling and Recovery

- Invalid file formats or severely corrupted documents must be reported to the caller with actionable error messages and not crash the pipeline.

F1.8 Output Interface

- Output a standardized candidate profile object to the next stage (Intelligent Matching Engine) including fields and confidence metrics.

Job Template Service (`Job Template Service`)

F2.1 AI-Assisted Creation

- The service must support natural-language prompts to generate job templates (e.g., “Create template for senior Java developer”).

F2.2 Template Schema

- Templates must include: job title, seniority level, required skills, experience bands, education requirements, location/remote options, languages, certifications, and any domain-specific constraints.

F2.3 Versioning and Audit

- Every creation/modification must be versioned; historical templates must be retrievable and auditable.

F2.4 Template Reuse and Inheritance

- Support base templates that can be extended for specific teams or roles while preserving core requirements.

F2.5 Validation

- Ensure templates contain at least a title and a non-empty skills list; enforce data-type constraints and field-level validations.

F2.6 API for Template Retrieval

- Provide endpoints to fetch templates by ID, list all templates, and search by keywords or skills.

Intelligent Matching Engine (`Intelligent Matching Engine`)

F3.1 Semantic Matching

- The engine must compare parsed resume data against job templates to produce a semantic compatibility score.

F3.2 Scoring Weighting

- Implement a two-part scoring model: AI-based assessment (75%) and human-intangible assessment (25%) as the default hybrid model.

F3.3 Explainable Scoring

- For each candidate, provide an explanation detailing key factors driving the AI score (e.g., “Strong React experience, but limited leadership roles”).

F3.4 Handling Partial Matches

- System should handle partial matches and present a graduated score with confidence indicators.

F3.5 Natural Language Query Support

- Support natural-language queries such as “Show me candidates with strong React skills but junior experience” and return filtered results.

F3.6 Known Limitations Handling

- Return warnings when the AI model cannot determine a reliable match and route to human review if confidence is too low.

F3.7 Performance and Throughput

- For the demo scope, demonstrate processing batches up to 50 resumes within a defined SLA (e.g., under 10 minutes for the batch, under 2 seconds per resume in the pipeline).

F3.8 Explainability and Traceability

- Each score should be accompanied by traceable prompts and model selections used for the decision.

Human Review Interface (`Human Review Interface`)

F4.1 Candidate List View

- Display a list of candidates with AI scores, confidence, and a summary profile.

F4.2 Score Overrides

- Allow reviewers to override the AI score and to add intangible notes (e.g., culture fit, leadership potential).

F4.3 Intangible Scoring

- Provide a dedicated field or UI surface to capture cultural fit, leadership potential, communication style, growth mindset, etc.

F4.4 Override Audit Trail

- Record who modified what score and when, including the reason for override.

F4.5 Collaboration and Commenting

- Support threaded comments or notes per candidate for team discussions.

F4.6 Data Privacy in UI

- Ensure display of PII complies with privacy rules; implement client-side masking where appropriate.

Bias Detection Service (`Bias Detection Service`)

F5.1 Job Description Bias Monitoring

- Analyze job descriptions to detect biased language and fairness issues (e.g., gendered phrasing, age-related indicators).

F5.2 Candidate Language Bias Monitoring

- Flag potential bias in resumes (e.g., experience gaps that could reflect bias rather than capability).

F5.3 Suggestions and Remediation

- Provide concrete remediation suggestions to improve fairness in job descriptions and evaluation criteria.

F5.4 Reporting

- Surface bias flags in the Analytics Engine and the UI.

Analytics Engine (`Analytics Engine`)

F6.1 HR Insights Dashboard

- Provide dashboards for metrics such as processing time per batch, consistency of scoring across reviewers, bottlenecks by stage, and reviewer workload.

F6.2 Data Export

- Allow export of aggregated data for reporting (CSV/JSON, with PII redaction as needed).

F6.3 Real-time/Near Real-time Metrics

- Display metrics with a short refresh interval to reflect ongoing batch processing.

AI Integration Points and Prompts (`AI Integration Points`)

F7.1 AI Call Orchestration

- All Claude Bedrock interactions must be orchestrated through a centralized prompt management layer (e.g., `Prompt Engine`) with versioned prompts.

F7.2 Prompt Templates

- Store prompt templates in a versioned repository; support A/B testing of prompts.

F7.3 Response Validation

- Validate AI outputs against schema (e.g., required fields in resume parsing, scoring structure) and implement fallbacks if outputs are missing or inconsistent.

F7.4 Logging and Observability

- Log all AI calls (prompt, model version, inputs, outputs) for audit and debugging.

AI Choices and Usage Strategy (`AI Choices and Usage Strategy`)

F8.1 AI as Development Tool

- Claude should be leveraged for boilerplate generation, code scaffolding, and test case generation where appropriate.

F8.2 Mixed-Grade AI Prompts

- Use Claude for primary AI tasks with human-in-the-loop for edge cases or high-stakes decisions.

F8.3 Data Generation and Testing

- Use AI-generated realistic resumes as test data for validation, ensuring synthetic data is distinguishable from real data for privacy.

Data Management and Test Data (`Data Management and Test Data`)

F9.1 Test Data Generation

- The system must be capable of generating 20+ realistic synthetic resumes for testing and demo purposes.

F9.2 Data Retention

- For the demo, retain data only for the duration of the sprint; define a cleanup policy.

F9.3 PII Handling

- Ensure PII is masked or de-identified in non-production environments; implement data minimization in the UI.

Security, Privacy, and Compliance

F9.4 Access Control

- Implement role-based access control (RBAC) for the core features; at minimum a single hardcoded “HR Manager” user is used in the mock /demo.

F9.5 Audit Logging

- Maintain audit logs for critical actions: resume ingestion, parsing results, AI scoring results, human overrides, and template changes.

F9.6 Data Encryption

- Encrypt data at rest and in transit (TLS for API traffic; encryption for database at rest).

F9.7 Data Residency and Compliance

- Ensure compliance with applicable data protection regulations; document data flow for privacy review.

F9.8 Security Testing

- Include basic security testing in the sprint (threat modeling, basic penetration testing for the demo environment).

Data Objects and Classes

Core Domain Objects

Object	Summary
CandidateProfile	Identity, parsed resume data, and associations to Education, Experience, Skills, Certifications, Publications, ResumeDocument, and ScoreRecords.
ResumeDocument	Raw resume metadata and parsing status; linked to CandidateProfile.
EducationRecord	Educational background for a candidate.

ExperienceRecord	Work experience entries.
Skill	Skills (with proficiency, source).
Certification	Certifications earned.
Publication	Publications listed on the resume.
JobTemplate	AI-assisted or manual job template with required skills, experience bands, locations, etc.
ScoreRecord	AI score, human score, final score, confidence, rationale, and audit trail references.
OverrideNote	Human override details and rationale, linked to a ScoreRecord.
BiasFlag	Flags raised during bias analysis (job description, resume, or template).
User	System user (for demo RBAC and audit).
AuditLog	Centralized action trail for compliance and debugging.

Domain Model Details (fields and relationships)

Object	Fields and Relationships

CandidateProfile	`candidateId: UUID (PK)`
	`name: String`
	`email: String (PII; encrypted at rest in prod)`
	`phone: String (PII)`
	`parsedAt: LocalDateTime`
	`parsingConfidence: Map<String, Double> or dedicated ParsingConfidence`
	`resumeDocument: ResumeDocument (1:1)`
	`education: List<EducationRecord> (1:N)`
	`experiences: List<ExperienceRecord> (1:N)`
	`skills: List<Skill> (1:N)`
	`certifications: List<Certification> (1:N)`
	`publications: List<Publication> (1:N)`
	`scores: List<ScoreRecord> (1:N)`

ResumeDocument	`resumeId: UUID (PK)`
	`candidate: CandidateProfile (FK, nullable)`
	`fileName: String`
	`fileType: String`
	`size: long`
	`ingestionTimestamp: LocalDateTime`
	`parsingStatus: ParsingStatus (enum: PENDING, COMPLETE, FAILED)`
	`extractSummary: String (JSON or structured object)`
	`fileStorageLocation: String (e.g., path or S3 key)`
	`createdAt: LocalDateTime`

EducationRecord	`educationId: UUID (PK)`
	`candidate: CandidateProfile (FK)`
	`school: String`
	`degree: String`
	`fieldOfStudy: String`
	`startDate: LocalDate`
	`endDate: LocalDate`
	`gpa: String`
	`location: String`

ExperienceRecord	`experienceId: UUID (PK)`
	`candidate: CandidateProfile (FK)`
	`title: String`
	`company: String`
	`startDate: LocalDate`
	`endDate: LocalDate`
	`location: String`
	`description: Text`

Skill	skillId: UUID (PK) candidate: CandidateProfile (FK) name: String proficiencyLevel: ProficiencyLevel (enum: JUNIOR, MID, SENIOR, EXPERT) source: String (parsed, user-added, inferred)
Certification	certificationId: UUID (PK) candidate: CandidateProfile (FK) name: String authority: String startDate: LocalDate endDate: LocalDate status: String
Publication	publicationId: UUID (PK) candidate: CandidateProfile (FK) title: String publisher: String date: LocalDate url: String
JobTemplate	templateId: UUID (PK) version: int title: String seniorityLevel: String requiredSkills: List<String> (or a separate TemplateSkill join table) experienceBand: String educationRequirements: List<String> locationPreferences: List<String> languages: List<String> certifications: List<String> domainConstraints: DomainConstraints (value object) createdAt: LocalDateTime updatedAt: LocalDateTime createdBy: User (FK)
ScoreRecord	scoreId: UUID (PK) candidate: CandidateProfile (FK) template: JobTemplate (FK) aiScore: BigDecimal aiConfidence: Map<String, Double> or dedicated AIConfidence humanScore: BigDecimal intangiblesScore: BigDecimal finalScore: BigDecimal (derived: 0.75ai + 0.25human) scoringTimestamp: LocalDateTime rationale: String
OverrideNote	overrideId: UUID (PK) score: ScoreRecord (FK) reviewer: User (FK) overrideValue: BigDecimal reason: String timestamp: LocalDateTime
BiasFlag	biasId: UUID (PK) itemType: ItemType (enum: JOB_DESCRIPTION, RESUME, TEMPLATE) itemId: UUID (FK to the corresponding row; may be candidate_id, template_id, etc.) flagDetails: String (JSON) or structured object severity: Severity (enum: LOW, MEDIUM, HIGH) flaggedAt: LocalDateTime
User	userId: UUID (PK) username: String email: String role: Role (enum: HR_MANAGER, REVIEWER, ADMIN) createdAt: LocalDateTime

AuditLog	`logId: UUID (PK)`
	`user: User (FK)`
	`action: String`
	`targetType: String`
	`targetId: UUID`
	`timestamp: LocalDateTime`
	`details: String (JSON)`

Interfaces (API and UI)

Rest API endpoints:

Resumes / Ingestion

POST /api/v1/resumes/upload

Purpose: Ingest one or more resumes and start parsing (async).

Request (multipart):

- file: resume file (PDF/DOC/DOCX/TXT)
- candidate_meta (optional JSON): { "custom_id": "...", "source": "batch_upload" }

Request (JSON base64):

```
    {
        "candidates": [
                {
                        "candidate_id": "uuid",
                        "file_name": "resume.pdf",
                        "file_type": "application/pdf",
                        "size": 123456,
                        "content_base64": "JVBERi0x..."
                }
        ]
}
```

Responses:

202 Accepted

```
    {
        "batch_id": "batch-uuid",
        "ingested_count": 2,
        "status": "in_progress",
        "submitted_at": "2025-09-02T12:00:00Z"
}
```

400 Bad Request with structured error payload.

Behavior:

- Triggers asynchronous parsing via `Resume Processing Service`.

GET /api/v1/resumes/batch/{batch_id}/status

Purpose: Retrieve batch ingestion/parsing status.

Response:

```
    {
        "batch_id": "batch-uuid",
        "total": 2,
        "completed": 2,
        "failed": 0,
        "progress_percent": 100,
        "candidates": [
                {
                        "candidate_id": "uuid",
                        "parsing_status": "complete",
                        "parsing_confidence": {
                                "name": 0.98,
                                "skills": 0.89
                        }
                }
        ]
}
```

Candidate Profiles / Resume Data

GET /api/v1/candidates

Purpose: List candidates with filters and pagination.

Query params: `skill`, `experience_min_years`, `template_id`, `status`, `page`, `pageSize`

Response:

```
    {
        "total": 123,
        "page": 1,
        "pageSize": 20,
        "items": [
                {
                        "candidate_id": "uuid",
                        "name": "Jane Doe",
                        "skills": ["Java","Spring Boot"],
                        "ai_score": 88.5,
                        "parsing_status": "complete",
                        "parsing_confidence": { "name": 0.98, "skills": 0.89 },
                        "parsed_at": "2025-09-02T12:01:00Z"
                },
                ...
        ]
}
```

GET /api/v1/candidates/{candidate_id}

Purpose: Retrieve full candidate profile (PII masked in UI unless authorized).

Response:

```
    {
        "candidate_id": "...",
        "name": "Jane Doe",
        "email": "encrypted",
        "phone": "masked",
        "parsed_at": "...",
        "parsing_confidence": { "name": 0.98, "skills": 0.89 },
        "resume_document_id": "...",
        "education": [ ... ],
        "experience": [ ... ],
        "skills": [ ... ],
        "certifications": [ ... ],
        "publications": [ ... ],
        "scores": [ { "score_id": "...", "template_id": "...", "ai_score": 88.5, "final_score": 85.2, "rationale":
"..." } ]
}
```

POST /api/v1/candidates/{candidate_id}/override

Purpose: Create an override for a candidate's AI score.

Body:

```
    {
        "reviewer_id": "user-uuid",
        "override_value": 87.5,
        "reason": "Strong leadership potential observed in interview notes",
        "timestamp": "2025-09-02T12:10:00Z"
}
```

Response: 200 OK with updated score linkage.

Job Templates (AI-assisted)

POST /api/v1/templates

Purpose: Create a new job template (AI-assisted or manual).

Body:

```
    {
        "version": 1,
        "title": "Senior Java Developer",
        "seniority_level": "Senior",
        "required_skills": ["Java","Spring Boot","AWS","SQL"],
        "experience_band": "5-7 years",
        "education_requirements": ["BS in CS or related field"],
        "location_preferences": ["Remote","On-site in NYC"],
        "languages": ["English"],
        "certifications": [],
        "domain_constraints": { "industry": "Tech" }
}
```

Response:
201 Created

```
    {
        "template_id": "uuid",
        "version": 1,
        "created_at": "...",
        "created_by": "user-uuid"
}
```

GET /api/v1/templates

Purpose: List templates (summary).

Response:

```
    {
        "total": 5,
        "items": [
                { "template_id": "uuid", "title": "Senior Java Developer", "version": 1, "created_at": "...",
"created_by": "user-uuid" },
                ...
        ]
}
```

GET /api/v1/templates/{template_id}

- **Purpose**: Retrieve full template details.

PUT /api/v1/templates/{template_id}

- **Purpose**: Update template (new version or updated document).
- **Response**: 200 OK with updated template.

Scoring and Overrides

POST /api/v1/scores/compute

Purpose: Trigger AI scoring for a candidate against a template (single/batch).

Body:

```
    {
        "candidate_id": "...",
        "template_id": "...",
        "sources": { "resume_profile": "...", "prompt_version": "...", "model_version": "Claude Sonnet 4" }
}
```

Response:

```
    {
        "score_id": "...",
        "status": "queued" // or "processing", "complete"
}
```

GET /api/v1/scores/{score_id}

Purpose: Retrieve ScoreRecord with AI/human scores and rationale.

Response:

```
    {
        "score_id": "...",
        "candidate_id": "...",
        "template_id": "...",
        "ai_score": 88.5,
        "ai_confidence": { "overall": 0.92, "skills": 0.89 },
        "human_score": 85.0,
        "intangibles_score": 9.0,
        "final_score": 85.25,
        "scoring_timestamp": "...",
        "rationale": "Strong backend experience..."
}
```

POST /api/v1/scores/{score_id}/override

Purpose: Submit a human override post AI scoring.

Body:

```
    {
"reviewer_id": "...",
"override_value": 87.5,
"reason": "Adjusted for leadership potential observed in interview"
}
```

Response: 200 OK with updated ScoreRecord (and audit trail).

Bias Detection

POST /api/v1/bias/check

Purpose: Trigger bias analysis on provided text (job description or resume).

Body:

```
    {
        "item_type": "job_description",
        "item_id": "template-uuid",
        "text": "We are looking for a rockstar coder..."
}
```

Response:

```
    {
        "flags": [
                { "phrase": "rockstar", "severity": "low", "context": "..." }
        ],
        "remediation_suggestions": ["Use inclusive language", "..."]
}
```

GET /api/v1/bias/flags

Purpose: List bias flags.

Response:

```
    {
        "total": 4,
        "flags": [
                { "flag_id": "flag-uuid", "item_type": "job_description", "item_id": "template-uuid", "severity":
"high", "flag_details": { ... } }
        ]
}
```

GET /api/v1/bias/flags/{flag_id}

Purpose: Retrieve flag details.

Analytics

GET /api/v1/analytics/dashboard

Purpose: Retrieve metrics for HR dashboards.

Response:

```
    {
        "processing_time_avg": 9.2,
        "score_variance_across_reviewers": 3.1,
        "top_bottlenecks": ["batch_ingestion", "parsing"],
        "reviewer_workload": { "reviewerA": 12, "reviewerB": 8 }
}
```

GET /api/v1/analytics/export

Optional: Exports aggregated data (CSV/JSON) with redaction as configured.

AI Prompts (Internal / Auditing)

GET /api/v1/prompts/version

GET /api/v1/prompts/{prompt_id}

POST /api/v1/prompts/validate

```
    {
        "prompt_id": "...",
        "outputs_schema": { "required_fields": ["name","skills","experience"], ... }
}
```

UI Interfaces: Components and Screens

Candidate Review Dashboard

Pages/Components:

- CandidateListComponent
- CandidateDetailComponent
- AIExplainabilityPanel
- BatchUploadPanel

Data interactions:

- GET `/api/v1/candidates` for list
- GET `/api/v1/candidates/{candidate_id}` for detail
- GET `/api/v1/scores/{score_id}` for score context
- POST `/api/v1/scores/{score_id}/override` for overrides

UX:

- List shows AI score, confidence, actions (override, annotate)
- Detail includes parsed resume data, AI rationale, per-field confidence

Batch Upload

Component:

- BatchUploadPanel: Drag-and-drop, per-file progress, batch status, and link to batch status endpoint

Job Template Editor

Components:

- TemplateListComponent
- TemplateEditorComponent (AI-assisted generation via prompts)

UX:

- Create templates via AI prompt (e.g., “Create template for senior Java developer”)
- Validate fields, versioning, history

Bias Visualization

Components:

- BiasFlagPanel
- RemediationSuggestions

Data:

- Bias flags from `/bias/check` results
- Redaction controls

Analytics Dashboard

Components:

- ProcessingTimeChart
- ScoreConsistencyPanel
- BottlenecksHeatmap
- ReviewerWorkloadBarChart

Data:

- Metrics from `/analytics/dashboard`

Settings and Help

Components:

- HelpCenter
- DataPrivacyMaskingToggle
- Seed/demo data controls

End to End Call Flow Overview

The application follows a linear pipeline from resume ingestion to hiring insights, with AI integration at core stages (parsing, matching, bias). Flows are async for performance, with real-time UI updates. Here's a step-by-step breakdown for a typical session (e.g., HR Manager processing 50 resumes):

1. **Resume Ingestion and Batch Upload** (User initiates via UI):
    - Hiring manager logs in (mock: hardcoded user) and accesses the Candidate Review Dashboard in Angular.
    - Uses BatchUploadPanel to drag-and-drop up to 50 resumes (PDF/DOCX).
    - UI calls `POST /api/v1/resumes/upload` (multipart). Files are stored locally (mock: filesystem instead of S3 for sprint).
    - Backend (Spring Boot) initiates async processing via ResumeProcessingService. Apache Tika extracts raw text; Claude (Bedrock) parses into structured data (CandidateProfile with fields like name, skills, confidence scores).
    - Progress is tracked via `GET /api/v1/resumes/batch/{batch_id}/status`; UI shows progress bars (e.g., "20/50 completed").
2. **Data Parsing and Structuring** (AI + Backend processing):

    - Claude call: Prompt like "Extract candidate name, contact, education, experience, skills, certifications, publications. Output JSON with confidence per field."
    - Outputs: Populate CandidateProfile, EducationRecord, ExperienceRecord, Skill, etc. Handling errors: Skip corrupted files, log failures.
    - Normalization: AI cleans terms (e.g., "React development" "React.js"); audit via AuditLog.
3. **Scoring and Matching** (Intelligent Matching Engine):
    - For each candidate, UI/Backend calls `POST /api/v1/scores/compute` (async).
    - Claude matches resume against JobTemplate (e.g., "Senior Java Dev"): Semantic analysis on skills, experience, qualifications.
    - Hybrid scoring: AI score (75%) + Human intangibles (25%). Rationale includes explainability (e.g., "Strong backend, but junior leadership").
    - Outputs: ScoreRecord with aiScore, confidence (per-field), rationale. Queued for review.
4. **Human Review and Overrides** (User interaction):
    - UI fetches list via `GET /api/v1/candidates` (paginated); filters by skill/experience.
    - CandidateListComponent shows scores/confidence; CandidateDetailComponent displays parsed data, AI rationale via AIExplainabilityPanel.
    - Manager overrides: Use score override interface (slider for intangibles, e.g., culture fit 8/10). Calls `POST /api/v1/scores/{score_id}/override`.
    - Updates finalScore, logs OverrideNote and AuditLog for compliance.
5. **Bias Detection and Monitoring** (Parallel/AI):
    - On job templates/resumes, `POST /api/v1/bias/check` triggers Claude: Analyze for phrases like "rockstar" (prompt: "Flag biased language in job desc/resume").
    - Outputs: BiasFlag with severity, remediation (e.g., "Use gender-neutral terms").
    - Surfaced in UI (BiasVisualization); no stopping flow, but flagged for HR.
6. **Analytics and HR Insights** (Post-processing):
    - HR accesses Analytics Dashboard via `GET /api/v1/analytics/dashboard`.
    - Metrics: Avg time/batch, scoring variance, bottlenecks (e.g., parsing delays). Exports CSV/JSON via `GET /api/v1/analytics/export`.
    - Insights: Improve job templates based on patterns (e.g., high drop-off on skills).

Performance Targets: Full batch (50) processes in <10 mins (parsing: 2-3 mins, scoring: 4-5 mins via AI). Errors route to human review. Data retained only for sprint; cleanup post-demo.

Integration Points:

- **AI (Claude/Bedrock)**: Central for parsing/scoring/bias; versioned prompts logged.
- **Database**: PostgreSQL stores entities (e.g., CandidateProfile linked to ScoreRecord).
- **Frontend/Backend**: Angular calls APIs; Spring AI handles integrations.
- **Security**: Mock auth; PII masked in UI/responses.