PDF & EML Resume Extractor — BRD

1. Project Summary

Goal: Build a web app with a Node.js backend and Vue.js frontend that ingests resumes (PDF, EML), finds and extracts structured data (name, email, phone, education, experience, skills, attachments), supports a manual extraction view and an AI-assisted extraction view, allows flagging records for additional extraction, and exports results to Excel and summary files.

Primary users: Recruiters, HR staff, hiring managers.

Two main flows (UI pages): 1. **Manual Extractor** — Upload files, preview parsed text, manually map fields, save/export. 2. **AI Agent Extractor** — Upload files, call AI module (API key provided) to auto-extract fields, show confidence scores, allow human review and flagging for extra extraction.

---

2. Scope & Key Features

Supported Inputs

- PDF resumes (single/multi-page, images + text)
- EML files (email format); extract embedded attachments (PDFs, DOCX, images) and process them
- ZIP uploads (optional): unpack and process contained files

Output / Exports

- Excel (.xlsx) with one row per candidate + columns for all extracted fields
- CSV export
- Human-readable summary (.txt or .md) per resume
- JSON export (for integrations)

Core Features

- Bulk upload and queued processing
- Text extraction & OCR for scanned PDFs
- Email parsing for .eml files and extraction of attachments (recursive)
- AI-based entity extraction using provided API key (name, contact, education, experience, company, dates, skills, certifications)
- Manual field-mapping UI and inline editing
- Flagging mechanism: user flags a candidate/field to trigger *additional* extraction (deeper AI prompt or extra OCR passes)
- Confidence score display and thresholding (auto-accept vs require manual review)
- Audit trail: source file, timestamp, extraction engine version, user who verified
- REST API endpoints for programmatic upload and retrieval
- Authentication (JWT) and basic RBAC (uploader, reviewer, admin)

3. User Stories

- As a recruiter I can upload a PDF or EML so the system extracts candidate details automatically.
- As a reviewer I can open AI-extracted results, edit fields, and save corrected data.
- As a user I can flag a record to run additional extraction steps for missing or low-confidence fields.
- As an admin I can download all candidate data as Excel for downstream systems.

---

4. Data Model (high-level)

- Candidate: id, fullName, emails[], phones[], summary, education[], experience[], skills[], certifications[], attachments[], sourceFile, extractedAt, confidence
- Attachment: id, filename, mimeType, size, extractedText, processed
- ExtractionJob: jobId, files[], status, engineVersion, startedAt, finishedAt

---

5. System Architecture (high level)

Backend: Node.js (Express) with these suggested libraries: - File upload: Multer / busboy - EML parsing: mailparser (simpleParser) - PDF text/OCR: pdf-parse for native text; Tesseract or an OCR microservice for scanned PDFs - Document parsing: pdfjs-dist / pdf2json for structure - AI calls: HTTP client (axios) to your AI provider using provided API key - Email attachments: save to temp storage, enqueue for parsing - Queue: BullMQ / Redis for background jobs (bulk processing) - DB: PostgreSQL / MongoDB for extracted data

Frontend: Vue.js 3 + Vue Router + Pinia (or Vuex) - Two main pages/components: `ManualExtractor.vue` and `AIAgentExtractor.vue` - Candidate detail modal with inline edit, flag button, and history

Storage: local disk or cloud (S3) for file blobs; DB for metadata

---

6. API Endpoints (suggested)

- `POST /api/upload` — multipart upload (files[] + mode=manual|ai)
- `GET /api/jobs/:jobId/status` — job progress
- `GET /api/candidates` — paginated list
- `GET /api/candidates/:id` — full extracted record
- `POST /api/candidates/:id/flag` — flag for additional extraction
- `POST /api/extract/ai` — send file(s) to AI extractor (returns extraction JSON + confidence)
- `GET /api/export?format=xlsx|csv|json` — export results

---

7. AI Agent Integration Design

Preprocessing: extract raw text and basic metadata, convert attachments to text. Build a canonical input package:

```
{
  text: "<full extracted text up to X chars>",
  metadata: { filename, filetype, emailSubject?, from?, receivedDate? },
  attachmentsText: ["<text from attachment1>", ...]
}
```

Prompting Strategy: use a structured prompt asking the AI to return JSON with fixed keys (name, emails, phones, education[], experience[], skills[], summary, confidenceScores). Example: `Extract fields as JSON. If unsure, leave field empty and provide explanation.`

Flagging / Additional extraction: when user flags a record, send an expanded prompt: include attachments, ask targeted questions (e.g. "Find certification names and issuing dates"). Allow a 'deep extraction' mode that increases token budget or runs a multi-pass pipeline (OCR -> structure -> AI).

Safety & Validation: validate returned emails/phones with regex; normalize dates; reject outputs that are not valid JSON.

---

8. UI Wireframes (brief)

Page 1 — Manual Extractor - Upload area (drag & drop) - Files list + extracted plain text preview - Mapping panel: left raw text / right form fields (editable) - Save button, Export options

Page 2 — AI Agent Extractor - Upload area + toggle: `Auto-extract (AI)` ON/OFF - Results table: name | email | phone | confidence | flagged - Click candidate → detail with inline edits, `Flag for Deep Extraction` button - Settings: confidence threshold, model selection, API usage stats

---

9. Acceptance Criteria / Tests

- Upload a PDF resume; system extracts name, email, phone, top 3 skills and returns JSON with confidence > 0.6 for at least 80% of files in test set.
- Upload an EML containing a resume as attachment; system extracts attachments and parses them.
- Flagging a candidate triggers additional extraction and updates fields.
- Exported Excel contains all mapped columns and matches saved entries.
- Manual edits persist and are reflected in exported output.

---

10. Security & Privacy

- Secure file uploads (size limits, virus scan optionally)
- Store API keys in environment variables (never checked into repo)
- GDPR considerations: data retention policy, delete candidate data on request
- Transport: HTTPS

---

11. Minimum Viable Implementation Plan (4 sprints)

Sprint 1: Basic upload, PDF text extraction, store raw text, manual mapping UI. **Sprint 2:** EML parsing + attachment extraction, OCR for scanned PDFs. **Sprint 3:** AI integration (simple prompt → JSON), display results in AI page, edit/save. **Sprint 4:** Flagging workflow, deep extraction, exports, auth, polishing.

---

12. Deliverables

- Working Node.js backend with endpoints and worker queue
- Vue.js frontend with Manual & AI pages
- Basic tests for parsing and API
- README with deployment steps and how to set AI API key

---

13. Next actions for you

1. Decide DB (Postgres / Mongo).
2. Provide the AI API details or confirm which provider to target for exact prompt tuning.
3. Confirm whether you need on-prem OCR or can use cloud OCR.

Prepared by: Project Spec Generator *Date:* (today)