Resume Parsing System — TechCombo

Language: English + Hinglish notes

---

1. Goal

Ek scalable system banayenge jisme: - 1,00,000+ resumes (PDF/DOCX/TXT) ko upload karenge - Har resume se structured data extract karenge: `name, email, phone, skills, experience, education, locations` - Data PostgreSQL me store karenge (normalized tables) - React frontend se advanced filters (skills, experience, education) use karke search karenge - Parsing accuracy badhane ke liye Apache Tika + Python NLP parser (FastAPI) use karenge

---

2. High-level architecture

```
React (Frontend)
   ↕ HTTPS
Spring Boot (REST API)
   ↕ Queue (DB / Redis)
Resume Storage (S3 or VPS filesystem)
   ↕
Spring Batch / Worker -> sends file to Python FastAPI Parser
   ↕
Python FastAPI (PyResparser / spaCy / custom rules) -> returns JSON
   ↕
PostgreSQL (normalized tables)
   ↕
Optional: Elasticsearch for fast full-text & faceted search
```

Notes: - Use S3 in production for scalability; local filesystem ok for MVP. - Use Spring Batch to process large number of files in parallel and resume on failure.

---

3. Database Schema (PostgreSQL)

Table: candidates

Column	Type	Notes
id	BIGSERIAL PRIMARY KEY	
full_name	VARCHAR(255)	nullable true if not parsed

Column	Type	Notes
email	VARCHAR(255)	unique? maybe null
phone	VARCHAR(50)	normalized digits only
total_experience_months	INT	store months for range queries
current_location	VARCHAR(255)	parsed location
highest_education	VARCHAR(255)	e.g., B.Tech, MBA
parsed_status	VARCHAR(20)	`PENDING`, `IN_PROGRESS`, `SUCCESS`, `FAILED`
source	VARCHAR(100)	e.g., `naukri`, `manual_upload`
created_at	TIMESTAMP WITH TIME ZONE DEFAULT now()	
updated_at	TIMESTAMP WITH TIME ZONE DEFAULT now()	

Table: resumes

Column	Type	Notes
id	BIGSERIAL PRIMARY KEY	
candidate_id	BIGINT REFERENCES candidates(id) ON DELETE CASCADE	
file_name	VARCHAR(500)	
file_path	TEXT	S3 URL or filesystem path
file_type	VARCHAR(20)	`pdf`, `docx`
parsed_at	TIMESTAMP	
parsed_by	VARCHAR(50)	`python-parser-v1`
parsed_quality_score	NUMERIC(3,2)	0.00 - 1.00 confidence score

Table: skills

Column	Type	Notes
id	BIGSERIAL PRIMARY KEY	
candidate_id	BIGINT REFERENCES candidates(id)	

Column	Type	Notes
skill_name	VARCHAR(200)	
extracted_position	INT	order found in resume (optional)

Table: experiences

Column	Type	Notes
id	BIGSERIAL PRIMARY KEY	
candidate_id	BIGINT REFERENCES candidates(id)	
title	VARCHAR(255)	
company	VARCHAR(255)	
start_date	DATE	
end_date	DATE	
duration_months	INT	
description	TEXT	

Table: education

Column	Type	Notes
id	BIGSERIAL PRIMARY KEY	
candidate_id	BIGINT REFERENCES candidates(id)	
degree	VARCHAR(255)	
college	VARCHAR(255)	
start_year	INT	
end_year	INT	
percentage	VARCHAR(50)	

Indexes and optimizations

- Index on `candidates(email)` (if many unique emails)
- Index on `skills(skill_name)` for faster JOIN filter
- GIN index on `to_tsvector` of combined searchable fields if using Postgres full-text search

---

4. Register page and candidate registration fields

Register page (for candidate manual sign-up): - fullName (required) - email (required) — validated - password (required) — bcrypt hashed in backend - phone (optional) - currentLocation (optional) - resumeUpload (optional) — accept pdf/docx - termsAccepted (boolean)

Backend considerations: - Use `spring-security` for auth (JWT or session) - Password stored with BCryptPasswordEncoder - When a user uploads resume during register, create `resumes` entry + push to parse queue

---

5. API Design (Spring Boot)

Auth & User

- `POST /api/auth/register` — body: {fullName, email, password, phone}
- `POST /api/auth/login` — returns JWT

Resume Upload

- `POST /api/resumes/upload` — multipart file -> returns resumeId (creates candidate if needed or link to candidate)
- `GET /api/resumes/{id}` — download

Parsing & Status

- `GET /api/resumes/{id}/status` — returns parsed_status and parsed_quality_score
- `POST /api/resumes/{id}/reparse` — requeue for parsing

Search & Filters

- `POST /api/search` — body: {skills:[], minExpMonths, maxExpMonths, education, location, page, size}
- Use pagination & sorting

---

6. Spring Boot — Example snippets

> NOTE: Ye snippets minimal hai; production me exception handling, DTOs, validation add karna.

Entity (Candidates)

```
@Entity
@Table(name = "candidates")
public class Candidate {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String fullName;
    private String email;
    private String phone;
    private Integer totalExperienceMonths;
    private String highestEducation;
    private String currentLocation;
    private String parsedStatus;

    @Column(nullable = false)
    private OffsetDateTime createdAt = OffsetDateTime.now();

    // getters/setters
}
```

Resume Upload Controller (simplified)

```
@RestController
@RequestMapping("/api/resumes")
public class ResumeController {

    @PostMapping("/upload")
    public ResponseEntity<?> uploadResume(@RequestParam("file") MultipartFile
file,
                                          @RequestParam(value = "candidateId",
required = false) Long candidateId) {
        // 1. Save file to disk or S3
        // 2. Create resume db row parsed_status = PENDING
        // 3. Push job to parsing queue (DB / Redis)
        return ResponseEntity.ok(Map.of("resumeId", resumeId));
    }
}
```

Background Worker trigger (Spring Batch / Scheduler)

- Use Spring Batch job to pick `resumes` with `parsed_status = PENDING` in batches of N (e.g., 100)
- For each resume, call Python FastAPI parser (HTTP POST with file path)
- Update DB with parsed JSON

---

7. Python FastAPI Parser — Example

Objective: Receive file path or file bytes, extract plain text (Apache Tika or pdfminer), run PyResparser / spaCy NER, return JSON with fields and confidence.

```python
from fastapi import FastAPI, UploadFile, File
import requests
import json

app = FastAPI()

@app.post('/parse')
async def parse(file: UploadFile = File(...)):
    contents = await file.read()
    # 1. Use tika or pdfminer to extract text
    # 2. Run NLP: regex fallback + spaCy NER for names, orgs, dates
    # 3. Skill extraction: use skill list dictionary + fuzzy match

    return {
        "name": "Rahul Sharma",
        "email": "rahul@example.com",
        "phone": "9876543210",
        "skills": ["Java","Spring Boot","React"],
        "experience_months": 60,
        "education": [{"degree":"B.Tech","college":"ABC Institute","year":2018}]
    }
```

Parsing tips: - Use a **skills master list** (10k common skills) and fuzzy-match tokens from text. - For phone/email use regex (safer) but for `name`, use spaCy NER models and heuristics (like name appears before email, top of doc). - Compute a `parsed_quality_score` based on how many mandatory fields were found and confidence of NER/fuzzy-match.

---

8. Batch processing & scaling

- Use **Spring Batch** with chunk-oriented processing.
- Use `taskExecutor` to parallelize workers.
- Use `retry` / `skip` policies for bad files.
- Monitor with metrics (Prometheus + Grafana) for throughput.

---

9. Search: Postgres vs Elasticsearch

- For simple filters & small scale, Postgres `JOINs` + indexes okay.

- For 1L resumes and advanced search (fuzzy, multi-field, relevance), use **Elasticsearch** or OpenSearch.
- Sync flow: After parsing, update Postgres and index a document in Elasticsearch:

```json
{
  "candidateId": 123,
  "name": "Rahul Sharma",
  "skills": ["Java","Spring Boot"],
  "experience_months": 60
}
```

---

10. Sample SQL Create Statements

```sql
CREATE TABLE candidates (
  id BIGSERIAL PRIMARY KEY,
  full_name VARCHAR(255),
  email VARCHAR(255),
  phone VARCHAR(50),
  total_experience_months INT,
  highest_education VARCHAR(255),
  current_location VARCHAR(255),
  parsed_status VARCHAR(20) DEFAULT 'PENDING',
  source VARCHAR(100),
  created_at timestamptz DEFAULT now(),
  updated_at timestamptz DEFAULT now()
);

CREATE TABLE resumes (
  id BIGSERIAL PRIMARY KEY,
  candidate_id BIGINT REFERENCES candidates(id),
  file_name VARCHAR(500),
  file_path TEXT,
  file_type VARCHAR(20),
  parsed_at timestamptz,
  parsed_by VARCHAR(50),
  parsed_quality_score NUMERIC(3,2)
);

CREATE TABLE skills (
  id BIGSERIAL PRIMARY KEY,
  candidate_id BIGINT REFERENCES candidates(id),
  skill_name VARCHAR(200)
);
```

11. Deployment (Docker Compose example)

- Services: `spring-api`, `python-parser`, `postgres`, `redis` (queue), `elasticsearch` (optional)

```
version: '3.8'
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: techcombo
  spring-api:
    build: ./spring-api
    depends_on: [db]
  parser:
    build: ./python-parser
    depends_on: [db]
```

12. Implementation Roadmap (phases)

1. MVP (2-3 weeks): File upload, Tika text extraction, basic regex for email/phone, store in Postgres
2. Parser (2-4 weeks): Python FastAPI + spaCy + skills dictionary, integrate with Spring via HTTP
3. Batch & Scale (2-3 weeks): Spring Batch, Redis queue, parallel workers
4. Search & UI polish (2-3 weeks): Elasticsearch, React advanced filters, pagination

13. Extras & Tips

- Keep a `failed_files` directory to inspect parsing failures manually.
- Keep a manual correction UI for HR to fix parsed data (human-in-loop improves ML model).
- Use logs & sample store: save original text for each resume to re-run improved parsers later.

14. Appendix: Useful libraries & references

- Java: Spring Boot, Spring Batch, Apache Tika, Spring Data JPA
- Python: FastAPI, spaCy, PyResparser, pdfminer.six, tika-python
- Search: Elasticsearch / OpenSearch
- Cloud: AWS S3, ECS or EKS or simple VPS

Document prepared for TechCombo — ready to convert into PDF or Word. Agar chahiye to mai isse PDF me export kar ke de du (downloadable) with code files attached.