**JobFit – Development Diary**

**Day 3: Technology Keyword Matching & Result Display**  
**Date:** 23 July 2025

**1. Objective**

Implement the core functionality to compare extracted resume and job description texts using a predefined list of technology keywords. Calculate match percentage and identify missing skills. Display results on a new results page.

**2. Technology Keywords Setup**

Defined a dictionary with keywords and their common variants/synonyms for better matching accuracy:

python

CopyEdit

TECH\_KEYWORDS = {

'python': ['python'],

'java': ['java'],

'c++': ['c++', 'cpp'],

'flask': ['flask'],

'django': ['django'],

'react': ['react', 'reactjs'],

'nodejs': ['nodejs', 'node.js', 'node'],

'sql': ['sql', 'mysql', 'postgresql'],

'mongodb': ['mongodb', 'mongo'],

'apis': ['api', 'apis', 'rest api', 'restful'],

'machine learning': ['machine learning', 'ml'],

'deep learning': ['deep learning', 'dl'],

'html': ['html'],

'css': ['css'],

'javascript': ['javascript', 'js'],

'aws': ['aws', 'amazon web services'],

'docker': ['docker']

}

**3. Matching Logic Implementation**

* Convert texts to lowercase for case-insensitive matching.
* Use regex with word boundaries to prevent false positives (e.g., "java" inside "javascript").
* Identify matched and missing technologies.
* Calculate match percentage based on the number of matched vs total job-required technologies.

python

CopyEdit

import re

def match\_technologies(resume\_text, job\_text):

resume\_text = resume\_text.lower()

job\_text = job\_text.lower()

resume\_tech = set()

job\_tech = set()

for tech, variants in TECH\_KEYWORDS.items():

for variant in variants:

pattern = r'\b' + re.escape(variant) + r'\b'

if re.search(pattern, resume\_text):

resume\_tech.add(tech)

if re.search(pattern, job\_text):

job\_tech.add(tech)

matched\_tech = resume\_tech & job\_tech

missing\_tech = job\_tech - resume\_tech

match\_percent = (len(matched\_tech) / len(job\_tech) \* 100) if job\_tech else 0

return round(match\_percent, 2), matched\_tech, missing\_tech

**4. Integration in Flask Route**

* After text extraction, apply matching function.
* Pass match results to result.html for display.

python

CopyEdit

@app.route("/", methods=["GET", "POST"])

def home():

if request.method == "POST":

resume = request.files.get("resume")

job = request.files.get("job")

match\_percent = 0

matched\_tech = set()

missing\_tech = set()

resume\_text = ""

job\_text = ""

if resume:

resume\_path = os.path.join(UPLOAD\_FOLDER, resume.filename)

resume.save(resume\_path)

resume\_text = extract\_text(resume\_path)

if job:

job\_path = os.path.join(UPLOAD\_FOLDER, job.filename)

job.save(job\_path)

job\_text = extract\_text(job\_path)

if resume\_text and job\_text:

match\_percent, matched\_tech, missing\_tech = match\_technologies(resume\_text, job\_text)

return render\_template(

"result.html",

match\_percent=match\_percent,

matched\_tech=matched\_tech,

missing\_tech=missing\_tech

)

return render\_template("index.html")

**5. Result Page (result.html)**

Created a new template to display:

* Match percentage
* Lists of matched and missing technologies

This provides clear feedback to users about their fit for the job.

**6. Testing**

* Tested with multiple resumes and job descriptions.
* Verified accuracy of matched and missing skills.
* Checked proper display on the results page.

**7. Project Structure**

cpp

CopyEdit

jobfit/

│

├── app.py

├── templates/

│ ├── index.html

│ └── result.html

├── static/

│ ├── css/

│ └── js/

└── upload/

**8. Status Summary**

* ✅ Technology keyword list defined with variants
* ✅ Matching logic implemented and integrated
* ✅ Results displayed on dedicated results page