AI-RESUME-ANALYZER — Working Code Pack

FastAPI backend + Streamlit frontend + skills taxonomy + basic test.

Generated: 2026-02-05

1) Repository Structure

```
AI-RESUME-ANALYZER/
├── backend/
│   ├── app/
│   │   ├── main.py
│   │   ├── api/
│   │   │   └── routes_analyze.py
│   │   ├── services/
│   │   │   ├── resume_parser.py
│   │   │   ├── skill_extractor.py
│   │   │   ├── similarity.py
│   │   │   └── suggestions.py
│   │   └── utils/
│   │       └── text_clean.py
│   ├── tests/
│   │   └── test_health.py
│   └── requirements.txt
├── frontend/
│   ├── streamlit_app.py
│   └── requirements.txt
├── data/
│   └── skills_taxonomy.json
└── README.md
```

2) Backend Files

backend/requirements.txt

```
fastapi==0.115.0
uvicorn[standard]==0.30.6
python-multipart==0.0.9
pydantic==2.8.2

pdfplumber==0.11.4
python-docx==1.1.2

scikit-learn==1.5.1
numpy==2.0.1
```

backend/app/main.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.api.routes_analyze import router as analyze_router

app = FastAPI(
    title="AI Resume Analyzer",
    version="1.0.0",
    description="Analyze resumes vs job descriptions: parsing, skills extraction, similarity scoring, ATS s
)

# Allow Streamlit (localhost) and general local testing
```

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # for demo; tighten later for production
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(analyze_router, prefix="")

@app.get("/health")
def health():
    return {"status": "ok"}
```

backend/app/api/routes_analyze.py

```
from fastapi import APIRouter, UploadFile, File, Form, HTTPException
from pydantic import BaseModel

from app.services.resume_parser import extract_text_from_resume
from app.services.skill_extractor import load_skills_taxonomy, extract_skills
from app.services.similarity import compute_similarity_score
from app.services.suggestions import build_suggestions

router = APIRouter()

class AnalyzeResponse(BaseModel):
    match_score: int
    matched_skills: list[str]
    missing_skills: list[str]
    suggestions: list[str]
    resume_preview: str
    jd_preview: str

@router.post("/analyze", response_model=AnalyzeResponse)
async def analyze_resume(
    resume: UploadFile = File(...),
    jd_text: str = Form(...),
):
    if not jd_text or len(jd_text.strip()) < 30:
        raise HTTPException(status_code=400, detail="Job description text is too short. Paste a longer JD."

    try:
        resume_text = await extract_text_from_resume(resume)
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

    if len(resume_text.strip()) < 30:
        raise HTTPException(status_code=400, detail="Could not extract enough text from the resume file.")

    taxonomy = load_skills_taxonomy()
    resume_skills = extract_skills(resume_text, taxonomy)
    jd_skills = extract_skills(jd_text, taxonomy)

    matched = sorted(list(resume_skills.intersection(jd_skills)))
    missing = sorted(list(jd_skills.difference(resume_skills)))

    score = compute_similarity_score(resume_text, jd_text)
    suggestions = build_suggestions(score, missing, resume_text)

    return AnalyzeResponse(
        match_score=score,
        matched_skills=matched[:50],
        missing_skills=missing[:50],
```

```
        suggestions=suggestions,
        resume_preview=resume_text[:500].strip(),
        jd_preview=jd_text[:500].strip(),
    )
```

backend/app/services/resume_parser.py

```
import io
from typing import Optional

import pdfplumber
from docx import Document
from fastapi import UploadFile

from app.utils.text_clean import clean_text

ALLOWED_EXTENSIONS = {".pdf", ".docx"}

def _get_extension(filename: Optional[str]) -> str:
    if not filename or "." not in filename:
        return ""
    return "." + filename.split(".")[-1].lower()

async def extract_text_from_resume(file: UploadFile) -> str:
    ext = _get_extension(file.filename)
    if ext not in ALLOWED_EXTENSIONS:
        raise ValueError("Unsupported file type. Please upload PDF or DOCX.")

    content = await file.read()

    if ext == ".pdf":
        text = _extract_pdf_text(content)
    else:
        text = _extract_docx_text(content)

    return clean_text(text)

def _extract_pdf_text(content: bytes) -> str:
    text_parts = []
    with pdfplumber.open(io.BytesIO(content)) as pdf:
        for page in pdf.pages:
            t = page.extract_text() or ""
            if t.strip():
                text_parts.append(t)
    return "\n".join(text_parts)

def _extract_docx_text(content: bytes) -> str:
    doc = Document(io.BytesIO(content))
    lines = []
    for p in doc.paragraphs:
        if p.text and p.text.strip():
            lines.append(p.text)
    return "\n".join(lines)
```

backend/app/services/skill_extractor.py

```
import json
import os
import re

REPO_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), "..", "..", ".."))
SKILLS_PATH = os.path.join(REPO_ROOT, "data", "skills_taxonomy.json")

def load_skills_taxonomy() -> dict:
```

```
    with open(SKILLS_PATH, "r", encoding="utf-8") as f:
        return json.load(f)

def _normalize(text: str) -> str:
    text = text.lower()
    text = re.sub(r"[^a-z0-9\+\#\.\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

def extract_skills(text: str, taxonomy: dict) -> set[str]:
    norm = _normalize(text)
    found = set()

    for item in taxonomy.get("skills", []):
        name = item.get("name", "").strip()
        aliases = item.get("aliases", [])
        if not name:
            continue

        patterns = [name] + aliases
        for p in patterns:
            p_norm = _normalize(p)
            if re.search(rf"(^| )({re.escape(p_norm)})( |$)", norm):
                found.add(name)
                break

    return found
```

backend/app/services/similarity.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def compute_similarity_score(resume_text: str, jd_text: str) -> int:
    docs = [resume_text, jd_text]
    vec = TfidfVectorizer(stop_words="english", max_features=5000, ngram_range=(1, 2))
    X = vec.fit_transform(docs)
    sim = cosine_similarity(X[0], X[1])[0][0]
    sim = max(0.0, min(1.0, float(sim)))
    return int(round(sim * 100))
```

backend/app/services/suggestions.py

```
import re

def build_suggestions(score: int, missing_skills: list[str], resume_text: str) -> list[str]:
    suggestions = []

    if score < 50:
        suggestions.append("Your resume has low alignment with this JD. Tailor the summary + experience bul
    elif score < 75:
        suggestions.append("Good alignment, but you can improve by adding more role-specific keywords in ex
    else:
        suggestions.append("Strong match! Minor improvements: add measurable impact and ensure ATS-friendly

    if missing_skills:
        suggestions.append(f"Add missing skills (only if you truly have them): {', '.join(missing_skills[:1

    if "table" in resume_text.lower() or re.search(r"\|\s*\w+\s*\|", resume_text):
        suggestions.append("Avoid heavy tables/columns in resumes. ATS tools may parse them incorrectly.")

    if len(resume_text) < 800:
        suggestions.append("Resume content looks short. Add more project/experience details with action + i
```

```
    suggestions.append("Use strong verbs + impact: Built, Automated, Improved, Reduced, Delivered + add num
    return suggestions
```

backend/app/utils/text_clean.py

```
import re

def clean_text(text: str) -> str:
    if not text:
        return ""
    text = text.replace("\x00", " ")
    text = re.sub(r"[ \t]+", " ", text)
    text = re.sub(r"\n{3,}", "\n\n", text)
    return text.strip()
```

backend/tests/test_health.py

```
from fastapi.testclient import TestClient
from app.main import app

client = TestClient(app)

def test_health():
    r = client.get("/health")
    assert r.status_code == 200
    assert r.json()["status"] == "ok"
```

3) Frontend Files

frontend/requirements.txt

```
streamlit==1.37.1
requests==2.32.3
```

frontend/streamlit_app.py

```
import streamlit as st
import requests

st.set_page_config(page_title="AI Resume Analyzer", layout="centered")

st.title("AI Resume Analyzer")
st.write("Upload a resume (PDF/DOCX), paste a Job Description, and get match score + skills + suggestions."

backend_url = st.text_input("Backend URL", value="http://localhost:8000")

resume_file = st.file_uploader("Upload Resume (PDF/DOCX)", type=["pdf", "docx"])
jd_text = st.text_area("Paste Job Description", height=220)

if st.button("Analyze"):
    if not resume_file:
        st.error("Please upload a resume file.")
        st.stop()
    if not jd_text.strip():
        st.error("Please paste the job description.")
        st.stop()

    with st.spinner("Analyzing..."):
        files = {"resume": (resume_file.name, resume_file.getvalue(), resume_file.type)}
        data = {"jd_text": jd_text}
        try:
            r = requests.post(f"{backend_url}/analyze", files=files, data=data, timeout=120)
        except Exception as e:
            st.error(f"Could not reach backend: {e}")
            st.stop()

    if r.status_code != 200:
        st.error(f"Error: {r.status_code} - {r.text}")
        st.stop()

    result = r.json()

    st.subheader(f"Match Score: {result['match_score']}%")

    col1, col2 = st.columns(2)
    with col1:
        st.markdown("### Matched Skills")
        st.write(result["matched_skills"] if result["matched_skills"] else ["None found"])
    with col2:
        st.markdown("### Missing Skills")
        st.write(result["missing_skills"] if result["missing_skills"] else ["None"])

    st.markdown("### Suggestions")
    for s in result["suggestions"]:
        st.write(f"- {s}")

    with st.expander("Resume preview"):
        st.code(result["resume_preview"])

    with st.expander("JD preview"):
```

```
        st.code(result["jd_preview"])
```

4) Data File

data/skills_taxonomy.json

```
{
  "skills": [
    { "name": "Java", "aliases": ["core java", "java 8", "java11", "java 17"] },
    { "name": "Spring Boot", "aliases": ["springboot", "spring boot", "spring"] },
    { "name": "REST API", "aliases": ["rest", "restful", "api development"] },
    { "name": "Microservices", "aliases": ["microservice", "distributed services"] },

    { "name": "Selenium", "aliases": ["selenium webdriver"] },
    { "name": "TestNG", "aliases": ["test ng"] },
    { "name": "JUnit", "aliases": ["junit5", "junit 5"] },
    { "name": "API Testing", "aliases": ["postman", "rest assured", "api automation"] },
    { "name": "Automation Testing", "aliases": ["test automation", "automation framework"] },

    { "name": "Python", "aliases": ["python3", "py"] },
    { "name": "FastAPI", "aliases": ["fast api"] },

    { "name": "Docker", "aliases": ["containerization"] },
    { "name": "Kubernetes", "aliases": ["k8s"] },
    { "name": "AWS", "aliases": ["amazon web services"] },

    { "name": "SQL", "aliases": ["postgresql", "mysql", "oracle sql"] },
    { "name": "Git", "aliases": ["github", "gitlab"] },
    { "name": "CI/CD", "aliases": ["jenkins", "github actions", "gitlab ci"] }
  ]
}
```

5) README.md

```
# AI-RESUME-ANALYZER

AI-Resume-Analyzer is an NLP-based application that analyzes resumes, extracts technical skills, matches th

## Features
- Upload Resume (PDF/DOCX) and extract text
- Skill extraction using curated taxonomy (skills JSON)
- JD vs Resume similarity scoring (TF-IDF cosine similarity)
- ATS-style improvement suggestions
- FastAPI backend + Streamlit UI

## Tech Stack
- Backend: FastAPI, Python
- NLP/ML: scikit-learn (TF-IDF), keyword taxonomy
- Parsing: pdfplumber, python-docx
- Frontend: Streamlit

## Run Locally

### 1) Start Backend
cd backend
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000

### 2) Start Frontend
cd frontend
pip install -r requirements.txt
streamlit run streamlit_app.py
```

```
Open Streamlit and keep Backend URL as: http://localhost:8000

## API
- GET /health
- POST /analyze (multipart form: resume file + jd_text)
```

6) Run Commands and Git Push

```
# Backend (Terminal 1)
cd backend
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000

# Frontend (Terminal 2)
cd frontend
pip install -r requirements.txt
streamlit run streamlit_app.py

# Push to GitHub
git add .
git commit -m "Add working Resume Analyzer (FastAPI + Streamlit)"
git push
```