# Complete Resume Relevance System Code

## 1. app.py - Main Flask Application

python

from flask import Flask, render\_template, request, jsonify, redirect, url\_for

import os

import json

import re

from datetime import datetime

from werkzeug.utils import secure\_filename

import PyPDF2

import docx2txt

from collections import Counter

import math

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'innomatics-resume-system-2025'

app.config['UPLOAD\_FOLDER'] = 'uploads'

app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024 # 16MB max file size

# Ensure upload directory exists

os.makedirs(app.config['UPLOAD\_FOLDER'], exist\_ok=True)

# Sample data from the hackathon requirements

SAMPLE\_DATA = {

'job\_descriptions': [

{

'id': 'jd1',

'title': 'Data Science Intern',

'company': 'Tech Company A',

'requirements': ['Python', 'SQL', 'Machine Learning', 'Statistics', 'Data Analysis',

'Pandas', 'NumPy', 'Matplotlib', 'Problem Solving', 'Communication'],

'experience': '0-2 years',

'education': 'B.Tech Computer Science',

'duration': '6 months',

'description': 'Looking for Data Science intern with strong Python and SQL skills...'

},

{

'id': 'jd2',

'title': 'Data Engineer Intern',

'company': 'Tech Company B',

'requirements': ['Python', 'Spark', 'SQL', 'Data Pipelines', 'DevOps', 'Streaming Data',

'Big Data', 'Problem Solving', 'Team Collaboration', 'AWS'],

'experience': '0-1 years',

'education': 'B.Tech',

'duration': '6 months',

'description': 'Seeking Data Engineer intern to work with big data technologies...'

}

],

'resumes': [

{

'id': 'resume1',

'name': 'S Supriya',

'email': 'supriya083@gmail.com',

'skills': ['Python', 'SQL', 'Power BI', 'Data Analysis', 'EDA', 'Pandas',

'NumPy', 'Matplotlib', 'Statistics'],

'education': 'B.Tech Computer Science Engineering, 2021-2025',

'experience': ['Cricket Stats Analysis Project', 'Grocery Store Management System',

'Weather Data Analysis'],

'total\_experience': 'Projects/Internship Experience'

},

{

'id': 'resume2',

'name': 'Jay Raj',

'email': 'jayraj10@gmail.com',

'skills': ['Python', 'SQL', 'Power BI', 'Data Analysis', 'Pandas',

'NumPy', 'Matplotlib', 'Seaborn', 'Problem Solving'],

'education': 'B.Tech Computer Science Engineering, 2021-2025',

'experience': ['Data Analysis of MHTCET 2024', 'Weather Data Analysis Dashboard',

'Music Store Sales Analysis'],

'total\_experience': 'Projects Experience'

}

]

}

# In-memory storage for analysis results

analysis\_results = []

current\_job = None

class ResumeAnalyzer:

def \_\_init\_\_(self):

self.skill\_categories = {

'technical': ['Python', 'SQL', 'Java', 'JavaScript', 'C++', 'Machine Learning',

'Data Analysis', 'Power BI', 'Tableau', 'R', 'Spark'],

'tools': ['Pandas', 'NumPy', 'Matplotlib', 'Seaborn', 'Scikit-learn', 'TensorFlow',

'PyTorch', 'Git', 'Docker', 'Kubernetes'],

'soft': ['Problem Solving', 'Team Collaboration', 'Communication', 'Leadership',

'Adaptability', 'Critical Thinking'],

'databases': ['MySQL', 'PostgreSQL', 'MongoDB', 'SQLite', 'Redis'],

'cloud': ['AWS', 'Azure', 'GCP', 'Cloud Computing']

}

def extract\_text\_from\_pdf(self, file\_path):

"""Extract text from PDF file"""

try:

text = ""

with open(file\_path, 'rb') as file:

pdf\_reader = PyPDF2.PdfReader(file)

for page in pdf\_reader.pages:

text += page.extract\_text()

return text

except Exception as e:

print(f"Error extracting PDF text: {e}")

return ""

def extract\_text\_from\_docx(self, file\_path):

"""Extract text from DOCX file"""

try:

return docx2txt.process(file\_path)

except Exception as e:

print(f"Error extracting DOCX text: {e}")

return ""

def parse\_resume\_text(self, text):

"""Parse resume text to extract structured information"""

resume\_data = {

'name': '',

'email': '',

'skills': [],

'education': '',

'experience': [],

'projects': []

}

# Extract email

email\_pattern = r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

emails = re.findall(email\_pattern, text)

if emails:

resume\_data['email'] = emails[0]

# Extract skills (matching against known skills)

all\_skills = []

for category in self.skill\_categories.values():

all\_skills.extend(category)

found\_skills = []

text\_lower = text.lower()

for skill in all\_skills:

if skill.lower() in text\_lower:

found\_skills.append(skill)

resume\_data['skills'] = list(set(found\_skills))

# Simple name extraction (first line usually contains name)

lines = text.strip().split('\n')

if lines:

resume\_data['name'] = lines[0].strip()[:50] # First 50 chars of first line

# Extract education (look for degree keywords)

education\_keywords = ['b.tech', 'bachelor', 'master', 'phd', 'diploma', 'degree']

for line in lines:

line\_lower = line.lower()

if any(keyword in line\_lower for keyword in education\_keywords):

resume\_data['education'] = line.strip()

break

# Extract projects/experience (lines with 'project', 'work', 'experience')

project\_keywords = ['project', 'work', 'experience', 'internship', 'developed', 'built']

for line in lines:

line\_lower = line.lower()

if any(keyword in line\_lower for keyword in project\_keywords) and len(line.strip()) > 20:

resume\_data['experience'].append(line.strip())

return resume\_data

def calculate\_relevance\_score(self, resume\_data, job\_requirements):

"""Calculate relevance score between resume and job requirements"""

scores = {

'skills': 0,

'experience': 0,

'education': 0,

'overall': 0

}

# Skills matching (40% weight)

resume\_skills = [skill.lower() for skill in resume\_data['skills']]

job\_skills = [skill.lower() for skill in job\_requirements]

matched\_skills = set(resume\_skills).intersection(set(job\_skills))

if job\_skills:

skills\_score = (len(matched\_skills) / len(job\_skills)) \* 100

else:

skills\_score = 0

scores['skills'] = min(skills\_score, 100)

# Experience matching (30% weight)

experience\_score = len(resume\_data['experience']) \* 10 # 10 points per project/experience

scores['experience'] = min(experience\_score, 100)

# Education matching (30% weight)

education\_score = 50 if resume\_data['education'] else 0

if 'computer' in resume\_data['education'].lower() or 'engineering' in resume\_data['education'].lower():

education\_score = 80

scores['education'] = education\_score

# Calculate overall score

scores['overall'] = (scores['skills'] \* 0.4 +

scores['experience'] \* 0.3 +

scores['education'] \* 0.3)

return scores

def get\_verdict(self, score):

"""Get verdict based on score"""

if score >= 75:

return 'High'

elif score >= 50:

return 'Medium'

else:

return 'Low'

def get\_missing\_skills(self, resume\_skills, job\_requirements):

"""Get skills missing from resume"""

resume\_skills\_lower = [skill.lower() for skill in resume\_skills]

job\_requirements\_lower = [skill.lower() for skill in job\_requirements]

missing = []

for req\_skill in job\_requirements:

if req\_skill.lower() not in resume\_skills\_lower:

missing.append(req\_skill)

return missing

def generate\_feedback(self, resume\_data, job\_requirements, scores):

"""Generate improvement feedback"""

feedback = []

missing\_skills = self.get\_missing\_skills(resume\_data['skills'], job\_requirements)

if missing\_skills:

feedback.append(f"Consider adding these missing skills: {', '.join(missing\_skills)}")

if scores['experience'] < 50:

feedback.append("Add more relevant projects or internship experience to strengthen your profile")

if not resume\_data['education']:

feedback.append("Include your educational qualifications clearly")

if scores['skills'] < 60:

feedback.append("Focus on developing the key technical skills mentioned in the job requirements")

return feedback

analyzer = ResumeAnalyzer()

@app.route('/')

def index():

return render\_template('index.html',

job\_descriptions=SAMPLE\_DATA['job\_descriptions'],

sample\_resumes=SAMPLE\_DATA['resumes'])

@app.route('/set\_job', methods=['POST'])

def set\_job():

global current\_job

job\_data = request.get\_json()

if job\_data.get('sample\_id'):

# Use sample job description

for jd in SAMPLE\_DATA['job\_descriptions']:

if jd['id'] == job\_data['sample\_id']:

current\_job = jd

break

else:

# Create custom job description

current\_job = {

'id': 'custom',

'title': job\_data.get('title', 'Custom Job'),

'company': job\_data.get('company', 'Company'),

'requirements': job\_data.get('requirements', '').split(','),

'experience': job\_data.get('experience', ''),

'education': job\_data.get('education', ''),

'description': job\_data.get('description', '')

}

return jsonify({'success': True, 'job': current\_job})

@app.route('/upload\_resume', methods=['POST'])

def upload\_resume():

global analysis\_results, current\_job

if not current\_job:

return jsonify({'error': 'Please set a job description first'}), 400

if 'resume' not in request.files:

return jsonify({'error': 'No file uploaded'}), 400

file = request.files['resume']

if file.filename == '':

return jsonify({'error': 'No file selected'}), 400

if file:

filename = secure\_filename(file.filename)

file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(file\_path)

# Extract text based on file type

if filename.lower().endswith('.pdf'):

text = analyzer.extract\_text\_from\_pdf(file\_path)

elif filename.lower().endswith('.docx'):

text = analyzer.extract\_text\_from\_docx(file\_path)

else:

return jsonify({'error': 'Unsupported file format'}), 400

# Parse resume

resume\_data = analyzer.parse\_resume\_text(text)

# Calculate scores

scores = analyzer.calculate\_relevance\_score(resume\_data, current\_job['requirements'])

# Generate analysis result

result = {

'id': len(analysis\_results) + 1,

'timestamp': datetime.now().isoformat(),

'filename': filename,

'resume\_data': resume\_data,

'job\_title': current\_job['title'],

'scores': scores,

'verdict': analyzer.get\_verdict(scores['overall']),

'missing\_skills': analyzer.get\_missing\_skills(resume\_data['skills'], current\_job['requirements']),

'feedback': analyzer.generate\_feedback(resume\_data, current\_job['requirements'], scores)

}

analysis\_results.append(result)

# Clean up uploaded file

os.remove(file\_path)

return jsonify({'success': True, 'result': result})

@app.route('/analyze\_sample/<resume\_id>')

def analyze\_sample(resume\_id):

global analysis\_results, current\_job

if not current\_job:

return jsonify({'error': 'Please set a job description first'}), 400

# Find sample resume

sample\_resume = None

for resume in SAMPLE\_DATA['resumes']:

if resume['id'] == resume\_id:

sample\_resume = resume

break

if not sample\_resume:

return jsonify({'error': 'Sample resume not found'}), 404

# Calculate scores

scores = analyzer.calculate\_relevance\_score(sample\_resume, current\_job['requirements'])

# Generate analysis result

result = {

'id': len(analysis\_results) + 1,

'timestamp': datetime.now().isoformat(),

'filename': f"{sample\_resume['name']}\_resume.pdf",

'resume\_data': sample\_resume,

'job\_title': current\_job['title'],

'scores': scores,

'verdict': analyzer.get\_verdict(scores['overall']),

'missing\_skills': analyzer.get\_missing\_skills(sample\_resume['skills'], current\_job['requirements']),

'feedback': analyzer.generate\_feedback(sample\_resume, current\_job['requirements'], scores)

}

analysis\_results.append(result)

return jsonify({'success': True, 'result': result})

@app.route('/results')

def get\_results():

return jsonify({

'results': analysis\_results,

'current\_job': current\_job,

'stats': {

'total\_processed': len(analysis\_results),

'high\_suitability': len([r for r in analysis\_results if r['verdict'] == 'High']),

'medium\_suitability': len([r for r in analysis\_results if r['verdict'] == 'Medium']),

'low\_suitability': len([r for r in analysis\_results if r['verdict'] == 'Low']),

}

})

@app.route('/clear\_results', methods=['POST'])

def clear\_results():

global analysis\_results

analysis\_results = []

return jsonify({'success': True})

@app.route('/analytics')

def analytics():

stats = {

'total\_processed': len(analysis\_results),

'high\_suitability': len([r for r in analysis\_results if r['verdict'] == 'High']),

'medium\_suitability': len([r for r in analysis\_results if r['verdict'] == 'Medium']),

'low\_suitability': len([r for r in analysis\_results if r['verdict'] == 'Low']),

'avg\_score': sum([r['scores']['overall'] for r in analysis\_results]) / len(analysis\_results) if analysis\_results else 0,

'skill\_gaps': {}

}

# Calculate most common missing skills

all\_missing\_skills = []

for result in analysis\_results:

all\_missing\_skills.extend(result['missing\_skills'])

if all\_missing\_skills:

skill\_counter = Counter(all\_missing\_skills)

stats['skill\_gaps'] = dict(skill\_counter.most\_common(10))

return jsonify(stats)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, host='0.0.0.0', port=5000)

## 2. templates/index.html - Frontend Template

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Automated Resume Relevance Check System - Innomatics Research Labs</title>

<link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css" rel="stylesheet">

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

min-height: 100vh;

color: #333;

}

.container {

max-width: 1200px;

margin: 0 auto;

padding: 20px;

}

.header {

background: rgba(255, 255, 255, 0.95);

backdrop-filter: blur(10px);

padding: 20px 0;

margin-bottom: 30px;

border-radius: 15px;

box-shadow: 0 10px 30px rgba(0,0,0,0.1);

}

.header h1 {

text-align: center;

color: #764ba2;

font-size: 2.5em;

margin-bottom: 10px;

}

.header p {

text-align: center;

color: #666;

font-size: 1.2em;

}

.nav-tabs {

display: flex;

justify-content: center;

gap: 20px;

margin: 20px 0;

}

.nav-tab {

padding: 12px 24px;

background: rgba(255, 255, 255, 0.9);

border: 2px solid transparent;

border-radius: 25px;

cursor: pointer;

transition: all 0.3s ease;

font-weight: bold;

}

.nav-tab.active {

background: #764ba2;

color: white;

}

.nav-tab:hover {

background: #667eea;

color: white;

transform: translateY(-2px);

}

.card {

background: rgba(255, 255, 255, 0.95);

border-radius: 15px;

padding: 25px;

margin-bottom: 20px;

box-shadow: 0 10px 30px rgba(0,0,0,0.1);

backdrop-filter: blur(10px);

}

.card h3 {

color: #764ba2;

margin-bottom: 15px;

font-size: 1.5em;

}

.form-group {

margin-bottom: 20px;

}

.form-group label {

display: block;

margin-bottom: 8px;

font-weight: bold;

color: #555;

}

.form-control {

width: 100%;

padding: 12px;

border: 2px solid #ddd;

border-radius: 8px;

font-size: 14px;

transition: border-color 0.3s ease;

}

.form-control:focus {

border-color: #667eea;

outline: none;

box-shadow: 0 0 10px rgba(102, 126, 234, 0.3);

}

.btn {

padding: 12px 24px;

border: none;

border-radius: 25px;

font-weight: bold;

cursor: pointer;

transition: all 0.3s ease;

text-decoration: none;

display: inline-block;

}

.btn-primary {

background: linear-gradient(45deg, #667eea, #764ba2);

color: white;

}

.btn-primary:hover {

transform: translateY(-2px);

box-shadow: 0 10px 20px rgba(102, 126, 234, 0.3);

}

.btn-success {

background: linear-gradient(45deg, #56ab2f, #a8e6cf);

color: white;

}

.btn-danger {

background: linear-gradient(45deg, #ff416c, #ff4b2b);

color: white;

}

.grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

gap: 20px;

}

.upload-area {

border: 3px dashed #667eea;

border-radius: 15px;

padding: 40px;

text-align: center;

transition: all 0.3s ease;

cursor: pointer;

}

.upload-area:hover {

border-color: #764ba2;

background: rgba(102, 126, 234, 0.1);

}

.upload-area i {

font-size: 3em;

color: #667eea;

margin-bottom: 20px;

}

.result-card {

background: linear-gradient(135deg, #f093fb 0%, #f5576c 100%);

color: white;

border-radius: 15px;

padding: 20px;

margin-bottom: 15px;

}

.result-card.high {

background: linear-gradient(135deg, #4facfe 0%, #00f2fe 100%);

}

.result-card.medium {

background: linear-gradient(135deg, #43e97b 0%, #38f9d7 100%);

}

.result-card.low {

background: linear-gradient(135deg, #fa709a 0%, #fee140 100%);

}

.score-circle {

width: 100px;

height: 100px;

border-radius: 50%;

background: conic-gradient(#4facfe 0deg, #4facfe calc(var(--score) \* 3.6deg), rgba(255,255,255,0.3) calc(var(--score) \* 3.6deg));

display: flex;

align-items: center;

justify-content: center;

font-size: 24px;

font-weight: bold;

color: white;

margin: 0 auto 20px;

}

.hidden {

display: none;

}

.skills-container {

display: flex;

flex-wrap: wrap;

gap: 10px;

margin: 15px 0;

}

.skill-tag {

background: rgba(102, 126, 234, 0.2);

color: #764ba2;

padding: 5px 12px;

border-radius: 20px;

font-size: 12px;

font-weight: bold;

}

.missing-skill {

background: rgba(255, 65, 108, 0.2);

color: #ff416c;

}

.stats-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

gap: 20px;

}

.stat-card {

text-align: center;

padding: 30px;

background: rgba(255, 255, 255, 0.95);

border-radius: 15px;

box-shadow: 0 10px 30px rgba(0,0,0,0.1);

}

.stat-number {

font-size: 3em;

font-weight: bold;

color: #764ba2;

margin-bottom: 10px;

}

.loading {

text-align: center;

padding: 40px;

}

.spinner {

border: 4px solid rgba(102, 126, 234, 0.3);

border-top: 4px solid #667eea;

border-radius: 50%;

width: 50px;

height: 50px;

animation: spin 1s linear infinite;

margin: 0 auto 20px;

}

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

.alert {

padding: 15px;

border-radius: 10px;

margin-bottom: 20px;

font-weight: bold;

}

.alert-success {

background: rgba(86, 171, 47, 0.2);

color: #56ab2f;

border: 1px solid #56ab2f;

}

.alert-error {

background: rgba(255, 65, 108, 0.2);

color: #ff416c;

border: 1px solid #ff416c;

}

@media (max-width: 768px) {

.container {

padding: 10px;

}

.header h1 {

font-size: 2em;

}

.grid {

grid-template-columns: 1fr;

}

}

</style>

</head>

<body>

<div class="container">

<div class="header">

<h1><i class="fas fa-cogs"></i> Innomatics Research Labs</h1>

<p>Automated Resume Relevance Check System</p>

<div class="nav-tabs">

<div class="nav-tab active" onclick="showTab('dashboard')">

<i class="fas fa-dashboard"></i> Dashboard

</div>

<div class="nav-tab" onclick="showTab('results')">

<i class="fas fa-chart-bar"></i> Results

</div>

<div class="nav-tab" onclick="showTab('analytics')">

<i class="fas fa-analytics"></i> Analytics

</div>

</div>

</div>

<!-- Dashboard Tab -->

<div id="dashboard" class="tab-content active">

<div class="grid">

<!-- Job Description Setup -->

<div class="card">

<h3><i class="fas fa-briefcase"></i> Job Description Setup</h3>

<div class="form-group">

<label>Sample Job Descriptions</label>

<select id="sampleJdSelect" class="form-control">

<option value="">Select a sample JD...</option>

{% for jd in job\_descriptions %}

<option value="{{ jd.id }}">{{ jd.title }} - {{ jd.company }}</option>

{% endfor %}

</select>

</div>

<div class="form-group">

<label>OR Create Custom JD</label>

<input type="text" id="customTitle" class="form-control" placeholder="Job Title">

</div>

<div class="form-group">

<input type="text" id="customCompany" class="form-control" placeholder="Company Name">

</div>

<div class="form-group">

<textarea id="customRequirements" class="form-control" rows="3" placeholder="Required skills (comma separated)"></textarea>

</div>

<button class="btn btn-primary" onclick="setJobDescription()">

<i class="fas fa-check"></i> Set Job Requirements

</button>

</div>

<!-- Resume Upload -->

<div class="card">

<h3><i class="fas fa-file-upload"></i> Resume Processing</h3>

<div class="upload-area" onclick="document.getElementById('resumeFile').click()">

<i class="fas fa-cloud-upload-alt"></i>

<h4>Upload Resume</h4>

<p>Click here or drag & drop PDF/DOCX files</p>

<input type="file" id="resumeFile" hidden accept=".pdf,.docx" onchange="uploadResume()">

</div>

<div class="form-group" style="margin-top: 20px;">

<label>OR Analyze Sample Resumes</label>

<select id="sampleResumeSelect" class="form-control">

<option value="">Select a sample resume...</option>

{% for resume in sample\_resumes %}

<option value="{{ resume.id }}">{{ resume.name }} - {{ resume.email }}</option>

{% endfor %}

</select>

</div>

<button class="btn btn-success" onclick="analyzeSampleResume()">

<i class="fas fa-search"></i> Analyze Sample Resume

</button>

</div>

</div>

<div id="currentJob" class="card hidden">

<h3><i class="fas fa-info-circle"></i> Current Job Requirements</h3>

<div id="jobDetails"></div>

</div>

</div>

<!-- Results Tab -->

<div id="results" class="tab-content hidden">

<div class="card">

<div style="display: flex; justify-content: space-between; align-items: center; margin-bottom: 20px;">

<h3><i class="fas fa-list"></i> Analysis Results</h3>

<button class="btn btn-danger" onclick="clearResults()">

<i class="fas fa-trash"></i> Clear All

</button>

</div>

<div id="resultsContainer">

<p>No results yet. Upload and analyze some resumes to see results here.</p>

</div>

</div>

</div>

<!-- Analytics Tab -->

<div id="analytics" class="tab-content hidden">

<div class="card">

<h3><i class="fas fa-chart-line"></i> Analytics Dashboard</h3>

<div id="analyticsContainer" class="stats-grid">

<!-- Analytics will be loaded here -->

</div>

</div>

</div>

<!-- Loading indicator -->

<div id="loading" class="loading hidden">

<div class="spinner"></div>

<p>Processing resume...</p>

</div>

<!-- Alert container -->

<div id="alertContainer"></div>

</div>

<script>

let currentJob = null;

let analysisResults = [];

function showTab(tabName) {

// Hide all tabs

document.querySelectorAll('.tab-content').forEach(tab => {

tab.classList.add('hidden');

});

// Remove active from all nav tabs

document.querySelectorAll('.nav-tab').forEach(nav => {

nav.classList.remove('active');

});

// Show selected tab

document.getElementById(tabName).classList.remove('hidden');

// Add active to selected nav

event.target.classList.add('active');

// Load data for specific tabs

if (tabName === 'results') {

loadResults();

} else if (tabName === 'analytics') {

loadAnalytics();

}

}

function showAlert(message, type = 'success') {

const alertContainer = document.getElementById('alertContainer');

const alert = document.createElement('div');

alert.className = `alert alert-${type}`;

alert.innerHTML = `<i class="fas fa-${type === 'success' ? 'check' : 'exclamation'}-circle"></i> ${message}`;

alertContainer.appendChild(alert);

setTimeout(() => {

alert.remove();

}, 5000);

}

function setJobDescription() {

const sampleId = document.getElementById('sampleJdSelect').value;

const customTitle = document.getElementById('customTitle').value;

const customCompany = document.getElementById('customCompany').value;

const customRequirements = document.getElementById('customRequirements').value;

let jobData = {};

if (sampleId) {

jobData = { sample\_id: sampleId };

} else if (customTitle && customRequirements) {

jobData = {

title: customTitle,

company: customCompany || 'Company',

requirements: customRequirements,

experience: '0-2 years',

education: 'Bachelor\'s Degree'

};

} else {

showAlert('Please select a sample job or fill in custom job details', 'error');

return;

}

fetch('/set\_job', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(jobData)

})

.then(response => response.json())

.then(data => {

if (data.success) {

currentJob = data.job;

displayCurrentJob();

showAlert('Job requirements set successfully!');

} else {

showAlert('Failed to set job requirements', 'error');

}

})

.catch(error => {

console.error('Error:', error);

showAlert('Error setting job requirements', 'error');

});

}

function displayCurrentJob() {

const jobContainer = document.getElementById('currentJob');

const jobDetails = document.getElementById('jobDetails');

if (currentJob) {

jobDetails.innerHTML = `

<div style="display: grid; grid-template-columns: repeat(auto-fit, minmax(250px, 1fr)); gap: 20px;">

<div>

<strong>Position:</strong> ${currentJob.title}<br>

<strong>Company:</strong> ${currentJob.company}<br>

<strong>Experience:</strong> ${currentJob.experience}

</div>

<div>

<strong>Required Skills:</strong>

<div class="skills-container">

${currentJob.requirements.map(skill => `<span class="skill-tag">${skill}</span>`).join('')}

</div>

</div>

</div>

`;

jobContainer.classList.remove('hidden');

}

}

function uploadResume() {

if (!currentJob) {

showAlert('Please set job requirements first', 'error');

return;

}

const fileInput = document.getElementById('resumeFile');

const file = fileInput.files[0];

if (!file) {

showAlert('Please select a file', 'error');

return;

}

const formData = new FormData();

formData.append('resume', file);

document.getElementById('loading').classList.remove('hidden');

fetch('/upload\_resume', {

method: 'POST',

body: formData

})

.then(response => response.json())

.then(data => {

document.getElementById('loading').classList.add('hidden');

if (data.success) {

showAlert(`Resume analyzed successfully! Score: ${Math.round(data.result.scores.overall)}%`);

analysisResults.push(data.result);

fileInput.value = ''; // Clear file input

} else {

showAlert(data.error || 'Failed to analyze resume', 'error');

}

})

.catch(error => {

document.getElementById('loading').classList.add('hidden');

console.error('Error:', error);

showAlert('Error uploading resume', 'error');

});

}

function analyzeSampleResume() {

if (!currentJob) {

showAlert('Please set job requirements first', 'error');

return;

}

const resumeId = document.getElementById('sampleResumeSelect').value;

if (!resumeId) {

showAlert('Please select a sample resume', 'error');

return;

}

document.getElementById('loading').classList.remove('hidden');

fetch(`/analyze\_sample/${resumeId}`)

.then(response => response.json())

.then(data => {

document.getElementById('loading').classList.add('hidden');

if (data.success) {

showAlert(`Sample resume analyzed! Score: ${Math.round(data.result.scores.overall)}%`);

analysisResults.push(data.result);

} else {

showAlert(data.error || 'Failed to analyze sample resume', 'error');

}

})

.catch(error => {

document.getElementById('loading').classList.add('hidden');

console.error('Error:', error);

showAlert('Error analyzing sample resume', 'error');

});

}

function loadResults() {

fetch('/results')

.then(response => response.json())

.then(data => {

displayResults(data.results);

})

.catch(error => {

console.error('Error loading results:', error);

});

}

function displayResults(results) {

const container = document.getElementById('resultsContainer');

if (!results || results.length === 0) {

container.innerHTML = '<p>No results yet. Upload and analyze some resumes to see results here.</p>';

return;

}

container.innerHTML = results.map(result => `

<div class="result-card ${result.verdict.toLowerCase()}">

<div style="display: grid; grid-template-columns: 120px 1fr auto; gap: 20px; align-items: center;">

<div class="score-circle" style="--score: ${result.scores.overall}">

${Math.round(result.scores.overall)}%

</div>

<div>

<h4>${result.resume\_data.name || 'Unknown'}</h4>

<p><i class="fas fa-envelope"></i> ${result.resume\_data.email || 'N/A'}</p>

<p><i class="fas fa-briefcase"></i> ${result.job\_title}</p>

<p><i class="fas fa-file"></i> ${result.filename}</p>

<div style="margin-top: 10px;">

<strong>Skills Match:</strong> ${Math.round(result.scores.skills)}% |

<strong>Experience:</strong> ${Math.round(result.scores.experience)}% |

<strong>Education:</strong> ${Math.round(result.scores.education)}%

</div>

${result.missing\_skills.length > 0 ? `

<div style="margin-top: 10px;">

<strong>Missing Skills:</strong>

<div class="skills-container">

${result.missing\_skills.map(skill => `<span class="skill-tag missing-skill">${skill}</span>`).join('')}

</div>

</div>

` : ''}

${result.feedback.length > 0 ? `

<div style="margin-top: 10px;">

<strong>Recommendations:</strong>

<ul style="margin-left: 20px; margin-top: 5px;">

${result.feedback.map(fb => `<li>${fb}</li>`).join('')}

</ul>

</div>

` : ''}

</div>

<div style="text-align: center;">

<div style="font-size: 1.5em; font-weight: bold; margin-bottom: 10px;">

${result.verdict.toUpperCase()}

</div>

<div style="font-size: 0.9em; opacity: 0.8;">

${new Date(result.timestamp).toLocaleString()}

</div>

</div>

</div>

</div>

`).join('');

}

function clearResults() {

if (confirm('Are you sure you want to clear all results?')) {

fetch('/clear\_results', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

}

})

.then(response => response.json())

.then(data => {

if (data.success) {

analysisResults = [];

loadResults();

showAlert('All results cleared successfully');

}

})

.catch(error => {

console.error('Error:', error);

showAlert('Error clearing results', 'error');

});

}

}

function loadAnalytics() {

fetch('/analytics')

.then(response => response.json())

.then(data => {

displayAnalytics(data);

})

.catch(error => {

console.error('Error loading analytics:', error);

});

}

function displayAnalytics(data) {

const container = document.getElementById('analyticsContainer');

container.innerHTML = `

<div class="stat-card">

<div class="stat-number">${data.total\_processed}</div>

<h4>Total Processed</h4>

</div>

<div class="stat-card">

<div class="stat-number" style="color: #4facfe;">${data.high\_suitability}</div>

<h4>High Suitability</h4>

</div>

<div class="stat-card">

<div class="stat-number" style="color: #43e97b;">${data.medium\_suitability}</div>

<h4>Medium Suitability</h4>

</div>

<div class="stat-card">

<div class="stat-number" style="color: #fa709a;">${data.low\_suitability}</div>

<h4>Low Suitability</h4>

</div>

<div class="stat-card" style="grid-column: span 2;">

<div class="stat-number" style="color: #667eea;">${Math.round(data.avg\_score)}</div>

<h4>Average Score</h4>

</div>

${Object.keys(data.skill\_gaps).length > 0 ? `

<div style="grid-column: span 3; background: rgba(255,255,255,0.95); border-radius: 15px; padding: 25px;">

<h4 style="margin-bottom: 20px; color: #764ba2;">Most Common Missing Skills</h4>

<div style="display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 15px;">

${Object.entries(data.skill\_gaps).map(([skill, count]) => `

<div style="text-align: center; padding: 15px; background: rgba(102, 126, 234, 0.1); border-radius: 10px;">

<div style="font-size: 1.5em; font-weight: bold; color: #667eea;">${count}</div>

<div style="color: #764ba2;">${skill}</div>

</div>

`).join('')}

</div>

</div>

` : ''}

`;

}

// Initialize the application

document.addEventListener('DOMContentLoaded', function() {

// Any initialization code can go here

console.log('Automated Resume Relevance Check System initialized');

});

</script>

</body>

</html>

## 3. requirements.txt - Dependencies

Flask==2.3.3

PyPDF2==3.0.1

python-docx==0.8.11

Werkzeug==2.3.7

docx2txt==0.8

## 4. run.py - Application Launcher

python

#!/usr/bin/env python3

'''

Automated Resume Relevance Check System

Innomatics Research Labs Hackathon Solution

This script starts the Flask web application server.

'''

import os

import sys

from app import app

if \_\_name\_\_ == '\_\_main\_\_':

print("="\*60)

print(" AUTOMATED RESUME RELEVANCE CHECK SYSTEM")

print(" Innomatics Research Labs")

print("="\*60)

print()

print("Starting Flask server...")

print("Server will be available at: http://localhost:5000")

print("Press Ctrl+C to stop the server")

print()

try:

# Create uploads directory if it doesn't exist

if not os.path.exists('uploads'):

os.makedirs('uploads')

print("Created uploads directory")

# Start the Flask application

app.run(debug=True, host='0.0.0.0', port=5000)

except KeyboardInterrupt:

print("\nServer stopped by user")

sys.exit(0)

except Exception as e:

print(f"Error starting server: {e}")

sys.exit(1)

## 5. start.bat - Windows Startup Script

batch

@echo off

echo =========================================================

echo AUTOMATED RESUME RELEVANCE CHECK SYSTEM

echo Innomatics Research Labs

echo =========================================================

echo.

echo Installing dependencies...

pip install -r requirements.txt

echo.

echo Starting application...

python run.py

pause

## 6. start.sh - Unix/Linux Startup Script

bash

#!/bin/bash

echo "========================================================="

echo " AUTOMATED RESUME RELEVANCE CHECK SYSTEM"

echo " Innomatics Research Labs"

echo "========================================================="

echo ""

echo "Installing dependencies..."

pip install -r requirements.txt

echo ""

echo "Starting application..."

python3 run.py

## Installation & Usage Instructions

### Prerequisites

- Python 3.7 or higher

- pip (Python package installer)

### Setup Steps

1. Create a project folder and save all files

2. Create a templates folder and put index.html inside it

3. Install dependencies: pip install -r requirements.txt

4. Run the application: python app.py

5. Open browser: http://localhost:5000

### Project Structure

project/

â”œâ”€â”€ app.py # Main Flask application

â”œâ”€â”€ templates/

â”‚ â””â”€â”€ index.html # HTML template

â”œâ”€â”€ uploads/ # Temporary file uploads (auto-created)

â”œâ”€â”€ requirements.txt # Python dependencies

â”œâ”€â”€ run.py # Application launcher

â”œâ”€â”€ start.bat # Windows startup script

â””â”€â”€ start.sh # Unix/Linux startup script

This complete codebase implements all the hackathon requirements including:

- Resume parsing (PDF/DOCX)

- Job description processing

- Automated relevance scoring

- Skills gap analysis

- High/Medium/Low verdicts

- Personalized feedback

- Professional web dashboard

- Analytics and reporting

- Sample data integration