Project: AI-Powered Resume Analyzer

This project will allow users to upload resumes, and AI will analyze them based on job descriptions, extracting skills, experience, and other relevant details to provide feedback.

---

1. Tech Stack Breakdown

Frontend (User Interface)

• Languages: HTML, CSS, JavaScript

• Frameworks/Libraries:

• React.js (for dynamic UI) or Vanilla JS if you want a simpler version

• Bootstrap/Tailwind CSS (for styling)

• Features:

• File upload functionality (for PDF/DOCX resumes)

• Job description input field

• Display analysis results

Backend (Server & API Handling)

• Languages: JavaScript (Node.js) or Python (Flask/Django)

• Frameworks:

• Express.js (if using Node.js)

• Flask/Django (if using Python)

• Features:

• Handle file uploads and process resume text

• Call AI APIs for analysis

• Store results (if needed)

AI & NLP (Natural Language Processing)

• APIs:

• OpenAI API (for extracting skills and comparing with job descriptions)

• IBM Watson Natural Language Understanding (for keyword extraction)

• Resume Parser API (e.g., Affinda, Sovren)

• Libraries (if processing manually without APIs):

• spaCy (Python NLP library for extracting named entities)

• NLTK (Natural Language Toolkit for text processing)

• PyPDF2 (for extracting text from PDFs)

Database (If Storing Results)

• Options:

• MongoDB (NoSQL, great for storing structured data like parsed resume JSON)

• PostgreSQL/MySQL (Relational database for structured resume storage)

• Firebase (If using a serverless approach)

Cloud Storage (For Storing Uploaded Resumes)

• Options:

• AWS S3

• Firebase Storage

• Cloudinary

---

2. How the System Works

Step 1: Upload Resume

• User uploads a resume (PDF/DOCX).

• Frontend sends the file to the backend.

Step 2: Extract Resume Text

• Backend extracts text from the file using PyPDF2 or Tika (for DOCX).

Step 3: AI/NLP Processing

• AI extracts key sections (education, experience, skills, etc.).

• Compares extracted skills with job description keywords.

• Provides a score or feedback.

Step 4: Display Results

• The analyzed data and recommendations are displayed in a user-friendly format.

---

3. Tools & APIs for Resume Analysis

Resume Parsing APIs (Alternative to Manual Processing)

1. Affinda Resume Parser

2. Sovren Resume Parser

3. Hireability Resume Parser

NLP APIs for AI Processing

1. OpenAI GPT API

2. IBM Watson NLU

Text Extraction from PDFs

1. PyPDF2 (Python)

2. Apache Tika (Java/Python)

---

4. Deployment Options

• Frontend: Vercel, Netlify, Firebase Hosting

• Backend: Render, Heroku, AWS Lambda (if using serverless functions)

• Database: MongoDB Atlas, Firebase, Supabase

1. Accepts a PDF or DOCX resume.

2. Extracts text from the resume.

3. Uses OpenAI API to analyze and compare it with a job description.

4. Displays relevant skills and a match score.

Step 1: Setup

Install Dependencies

If using Node.js, install required packages:

```
npm init -y
npm install express multer axios dotenv pdf-parse openai
```

Step 2: Backend (Node.js)

```
require('dotenv').config();
const express = require('express');
const multer = require('multer');
const fs = require('fs');
const pdfParse = require('pdf-parse');
const axios = require('axios');

const app = express();
const upload = multer({ dest: 'uploads/' });

app.use(express.json());

// OpenAI API Call
async function analyzeResume(text, jobDescription) {
  const response = await axios.post(
    'https://api.openai.com/v1/chat/completions',
    {
      model: "gpt-3.5-turbo",
      messages: [
        { role: "system", content: "You are an expert in resume analysis." },
        { role: "user", content: `Analyze this resume:\n${text}\n\nMatch it with this job description:\n${jobDescription}\n\nGive a match percentage and suggested improvements.` }
      ]
    },
    { headers: { 'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`, 'Content-Type': 'application/json' } }
  );
  return response.data.choices[0].message.content;
```

```
}

// Resume Upload & Processing
app.post('/upload', upload.single('resume'), async (req, res) => {
  if (!req.file) return res.status(400).send("No file uploaded.");

  const fileBuffer = fs.readFileSync(req.file.path);
  const pdfText = await pdfParse(fileBuffer);

  const jobDescription = req.body.jobDescription || "Software Engineer with React and Node.js experience.";

  const analysis = await analyzeResume(pdfText.text, jobDescription);

  res.json({ analysis });
});

app.listen(5000, () => console.log("Server running on port 5000"));
```

Step 3: Frontend (HTML, CSS, JavaScript)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Resume Analyzer</title>
  <style>
```

```
        body { font-family: Arial, sans-serif; text-align: center; margin: 20px; }
        #result { margin-top: 20px; white-space: pre-wrap; }
    </style>
</head>
<body>
    <h2>Upload Resume for AI Analysis</h2>
    <input type="file" id="resume" accept=".pdf,.docx">
    <textarea id="jobDescription" placeholder="Enter job description..."></textarea>
    <button onclick="uploadResume()">Analyze Resume</button>

    <h3>Analysis Result</h3>
    <div id="result"></div>

    <script>
        async function uploadResume() {
            const file = document.getElementById('resume').files[0];
            const jobDescription = document.getElementById('jobDescription').value;
            if (!file) return alert("Please select a resume file.");

            let formData = new FormData();
            formData.append("resume", file);
            formData.append("jobDescription", jobDescription);

            const response = await fetch("http://localhost:5000/upload", {
                method: "POST",
                body: formData
            });
```

```
        const data = await response.json();
        document.getElementById("result").textContent = data.analysis;
    }
  </script>
</body>
</html>
```

Step 4: Run the Project

1. Start the backend:

```
node server.js
```

Enhancements (Next Steps)

✅ Add support for DOCX resumes (using mammoth in Node.js or python-docx in Python).

✅ Store parsed resume data in a database (MongoDB or Firebase).

✅ Show match percentage with visual charts.