

Resume AI analyzer

Overview

The Resume AI Analyzer is a web application that evaluates resumes using AI-powered text analysis. Users upload a PDF resume, and the system automatically extracts the content, sends it to an LLM, and returns structured feedback such as key skills, strengths, weaknesses, and overall job-match insights to support resume improvement and screening.

Architecture

Frontend (Flask Rendered UI)

- Framework: Flask with Jinja2 rendered HTML templates
- Styling: CSS for layout, theming, and responsive design
- Pages: Main upload page, loading/progress page, and results page that displays the AI analysis
- Behavior: JavaScript for handling UI updates, progress animations, and interaction with backend routes
- Features: Resume upload form, animated loading screen, and results page displaying AI-generated insights

Backend (Flask)

- Framework: Flask application with REST style routes
- Document Processing: Uses PDF parsing libraries to extract raw text from uploaded resumes
- AI Integration: OpenAI GPT-based LLM accessed via API
- API: Endpoints for file upload, resume parsing, AI analysis, and returning structured results to the frontend

AI Model

- Service: OpenAI API (cloud-hosted LLM)
- Default Model: Configurable GPT5mini model selected via environment/configuration
- Capabilities: Summarizes resume content, extracts and groups skills, highlights strengths and weaknesses, and generates tailored recommendations

Features

- AI-driven analysis of uploaded resumes (PDF/DOCX)
- Automatic extraction of skills and key resume information
- Job match style scoring and concise resume summary
- Clear breakdown of strengths, weaknesses, and improvement suggestions

Prerequisites

Before running this application, ensure you have:

1. Python (v3.8 or higher)
2. Open AI API key
3. Document Parsing Dependencies

Configuration

Backend Configuration

The backend is built using **Flask**, which handles:

- File uploads (PDF, DOCX)
- Integration with AI/ML models (LLMs, NLP models, keyword extractors, etc.)
- Parsing and extracting resume content
- Skill/job-match scoring
- Routing between frontend pages

Frontend Configuration

The frontend uses:

- **HTML templates** rendered by Flask (Jinja2)
- **CSS** for layout and styling

- **JavaScript** for UI updates, progress animations, and interacting with backend routes

Templates

- `main.html` — Homepage where users upload their resume
- `loading.html` — Intermediate page showing analysis progress
- `results.html` — Final AI-generated resume insights

Static Files

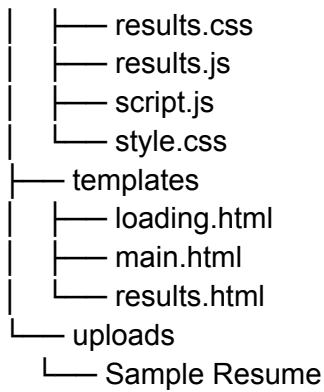
- `style.css` — Global styling
- `script.js` — Main frontend logic
- `loading.css` / `loading.js` — Animated loading screen
- `results.css` / `results.js` — Rendering and interactivity for the results page

API Documentation

The backend exposes a small set of internal API routes that handle resume processing. When a user uploads a resume, the `/upload` endpoint extracts the text using pdfplumber. This extracted text is then sent to the `/analyze` endpoint, where the backend prompts the OpenAI API to generate insights such as skill extraction, summary, strengths, weaknesses, and recommendations. The frontend retrieves the final analysis through the results endpoint and displays it to the user.

File Structure

```
.
├── app.py
├── static
│   ├── background image.png
│   ├── image 1.png
│   ├── loading.css
│   └── loading.js
```



Usage Guide

Follow these steps to use the AI Resume Analyzer:

1. **Go to the Website**

<https://ai-resume-screening.alwaysdata.net/>

2. **Upload Your Resume**

- Upload a PDF format of your resume
- There is no fixed limit; however, we recommend keeping it short for quicker results.

3. **Wait for Analysis**

You will see a loading screen while the AI:

- Extracts content
- Analyzes skills
- Score your resume
- Generates recommendations

4. **View Your Results**

The results page will display:

- Extracted skills

- Job-match score
- Resume summary
- Strengths & weaknesses
- Improvement suggestions

Performance Optimization

Backend Optimizations

1. PDF parsing: Used known PDF Python libraries to improve the accuracy of the raw text extracted from the resume
2. Performed prompt engineering to improve and sanitize the prompt done on the API
3. Designed the Flask server with deployment in mind to facilitate the deployment and maintenance of the app.

Frontend Optimizations

1. Developed a loading screen to emphasize the analysis done on the resume
2. Analyzed and designed options to select the most accessible design.
3. Used common practices to make the interaction of the website predictable.

Deployment

- **Project Upload / Repository Connection**

The project files were uploaded or linked to a GitHub repository, which the hosting platform uses to build and deploy the application

- **Environment Configuration**

Environment variables such as API keys and secret keys were securely added in the hosting dashboard.

- **Build Setup**

The hosting service automatically installed dependencies.

- **Web Server Configuration**

The platform launched the app using:

- A production-ready WSGI server (Always data)
- The Flask `app.py` entry point
- **Static & Template Handling**
Static assets (`/static`) and HTML templates (`/templates`) were automatically served by the Flask application.
- **Deployment Success**
After a successful build, the hosting platform generated a **live public URL**, making the service accessible to all users.

Support

For technical assistance, inquiries, or feedback regarding the AI Resume Analyzer, please reach out to our support team. We are committed to ensuring a smooth and reliable user experience and will respond as soon as possible.