🐙🔗 [GitHub Repository](https://github.com/YashCodes16/Resume_Ranker)

**#High-Level Solution Overview: Resume Ranker Application**

### **Objective**

**Resume Ranker** is a web-based application that analyzes resumes against job descriptions using Natural Language Processing (NLP) to generate insights, match scores, recommendations, and ATS compatibility feedback.

### **Microservice Architecture –** (You can use them separately as well)

The system is built on a **3-layer architecture**:

#### **1. Frontend Layer (User Interface)**

* **Technology**: HTML, CSS, JavaScript (Vanilla JS)
* **Role**:  
  + Accepts resume (doc/pdf) and job description input from the user and can **analyze 20 resumes** at one time.
  + Sends data to the backend via API calls (usually fetch()).
  + Displays results: match score, missing skills, keywords, recommendations, etc.
* **Hosted as**: Static files.

#### **2. Backend Layer (API Server)**

* **Technology**: Node.js + Express.js + libraries
* **Role**:  
  + Handles incoming HTTP requests from the frontend.
  + Acts as an orchestrator by forwarding requests to the NLP microservice.
  + Optionally handles user logging, Application level error handling , validations etc.
  + Provides REST APIs like:

* POST /api/resumes/analyze
* POST /api/resumes/generate-pdf
* GET /
* **Additional features**:  
  + Winston logging.
  + Multer for file uploads (if PDFs are supported).
  + .env for configuration.

#### **3. NLP Microservice Layer (Resume Analysis Engine)**

* **Technology**: Python + Flask + spaCy
* **Role**:  
  + Performs actual resume vs. job description analysis.
  + Computes:  
    - Match score using Jaccard & spaCy similarity
    - Extracted keywords & named entities
    - Format issues
    - Recommendations
  + Returns structured JSON response to the backend.
* **Model**:  
  + Uses en\_core\_web\_md spaCy model for vector similarity, NER, lemmatization.

### **🔄 Flow Summary**

1. **User** submits resume and job description via frontend.
2. **Frontend** sends the request (via fetch) to the **Node.js backend API**.
3. **Backend** validates the data and forwards it to the **Python microservice** via an internal API call (e.g., using axios).
4. **NLP Microservice** processes the request, analyzes the resume, and returns detailed metrics.
5. **Backend** sends the final result back to the **frontend** for display.

**#Project Structure**   
  
HackStudioMay2025/

├── frontend/ # HTML/CSS/JS static files

├── backend/ # Node.js server (Express)

│ ├── routes/

│ ├── controllers/

│ ├── middleware/

│ └── .env

├── resumeRankerNLPMicroservice/ # Flask microservice (Python)

│ ├── app/

│ ├── run.py

│ └── requirements.txt

└── README.md

## **🛠 Tools / Libraries Used**

### **📦 Backend – Node.js (Express)**

The Node.js backend serves as the middleware layer between the frontend and the Python microservice. The following libraries are used:

| **Library** | **Description** |
| --- | --- |
| express | Web framework for building REST APIs |
| dotenv | Loads environment variables from .env file |
| axios | Used to make HTTP requests to the Python microservice |
| cors | Enables Cross-Origin Resource Sharing |
| express-validator | Middleware for request validation and sanitization |
| mammoth | Converts .docx files to plain text |
| multer | Handles file uploads |
| pdf-parse | Extracts text content from PDF files |
| pdfkit | Generates PDF reports dynamically |
| puppeteer | Headless Chromium for HTML-to-PDF conversion |
| winston | Logging utility |
| path | Node.js module to handle file paths |
| nodemon | Automatically restarts the server on file changes (dev use) |

### **🐍 Microservice – Python (Flask + NLP)**

The Python service handles resume analysis using Natural Language Processing.

| **Library** | **Description** |
| --- | --- |
| Flask | Web framework to expose REST APIs |
| spaCy | Industrial-strength NLP library |
| en\_core\_web\_md | Pretrained spaCy model used for embeddings & entity extraction |
| re and collections | Standard libraries for text processing and counting |

🚀 **How to Run the Project (Frontend / Backend / Microservice)**

This project follows a 3-layer architecture. To run it successfully, each layer must be started independently in separate terminal windows or tabs.

### **🔹 1st Layer – Frontend (HTML / CSS / JS)**

A static web interface that collects input and displays analysis results.

Terminal

cd frontend

npx serve .

* 🌐 Hosted at: http://localhost:3000/

### **🔹 2nd Layer – Backend (Node.js / Express.js)**

The Node.js server that connects frontend with the Python microservice.

Terminal

cd backend

npm install

node server.js

* 🌐 Hosted at: http://localhost:8000/

### **🔹 3rd Layer – Microservice (Python / Flask / spaCy)**

The NLP engine that performs resume analysis using spaCy.

### **✅ Recommended Setup:**

Terminal

cd resumeRankerNLPMicroservice

python -m venv venv

.\venv\Scripts\Activate.ps1

pip install -r requirements.txt

python -m spacy download en\_core\_web\_md

python run.py

### **🔁 Alternate (Quick Setup):**

Terminal

cd resumeRankerNLPMicroservice

pip install -r requirements.txt

python -m spacy download en\_core\_web\_md

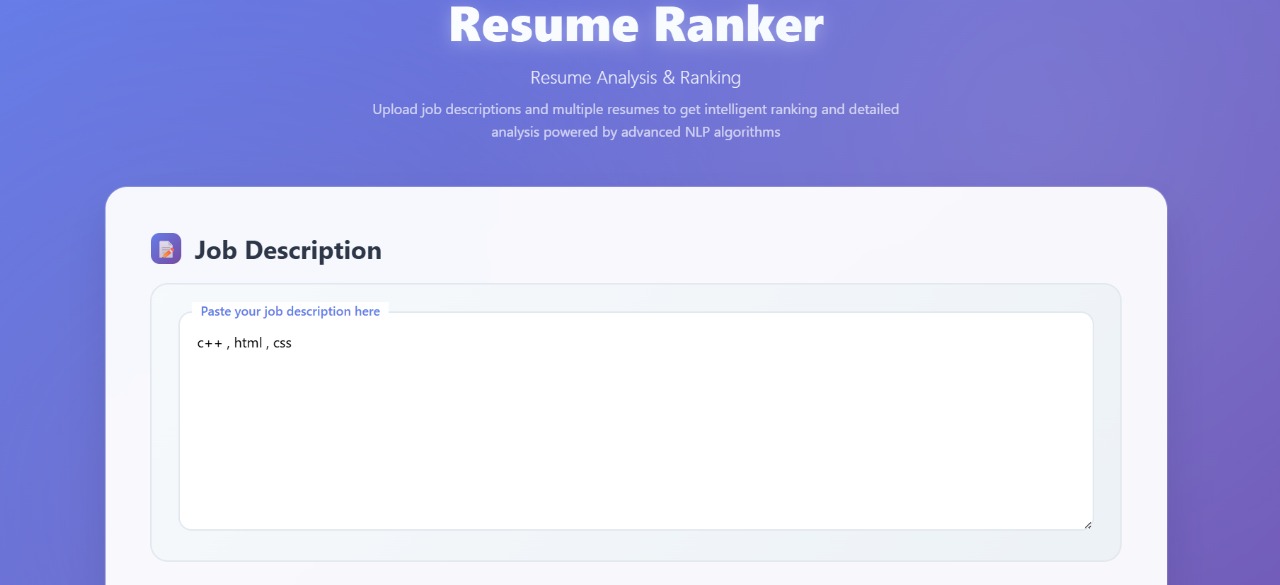
python run.py

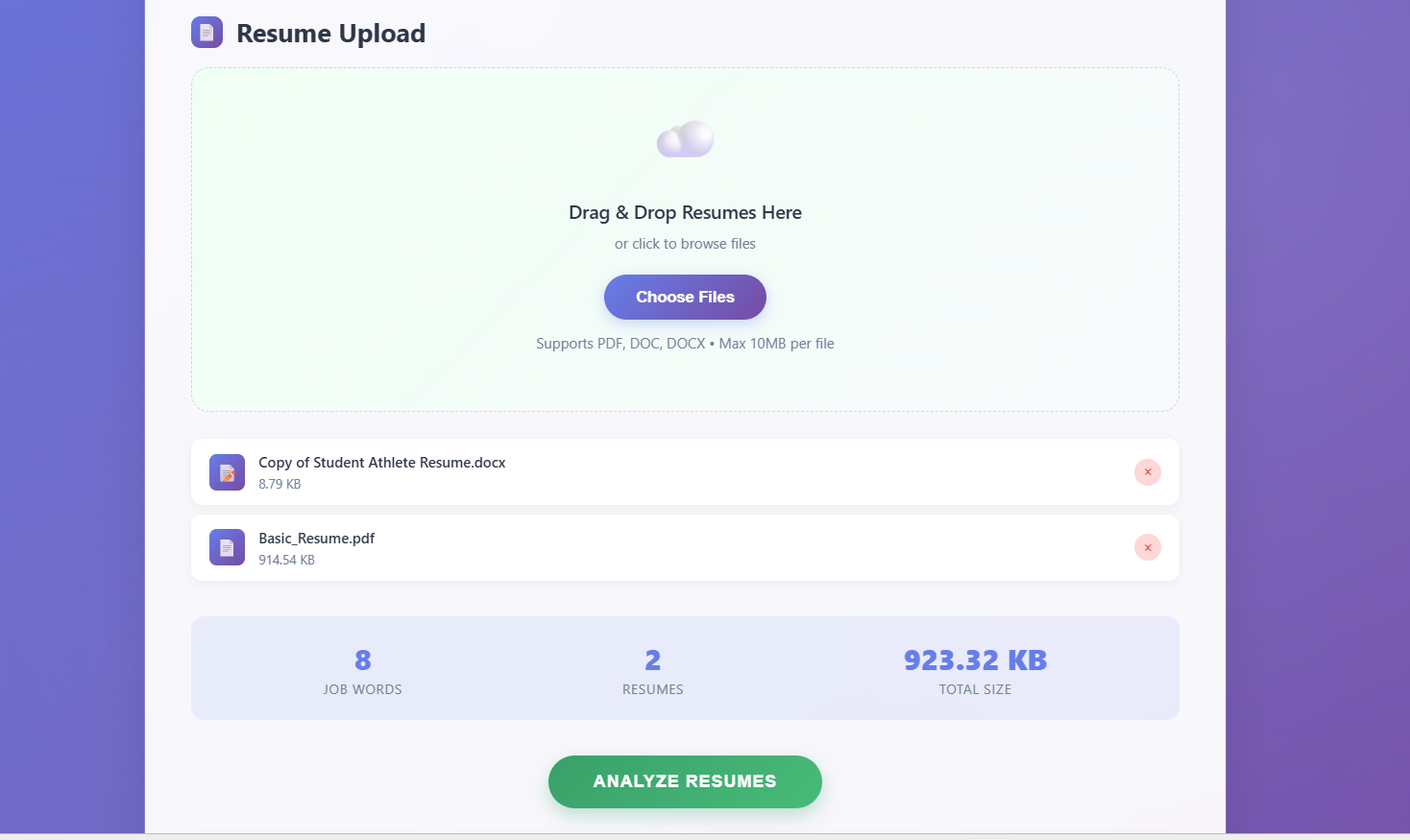
* 🌐 Hosted at: http://localhost:5000/

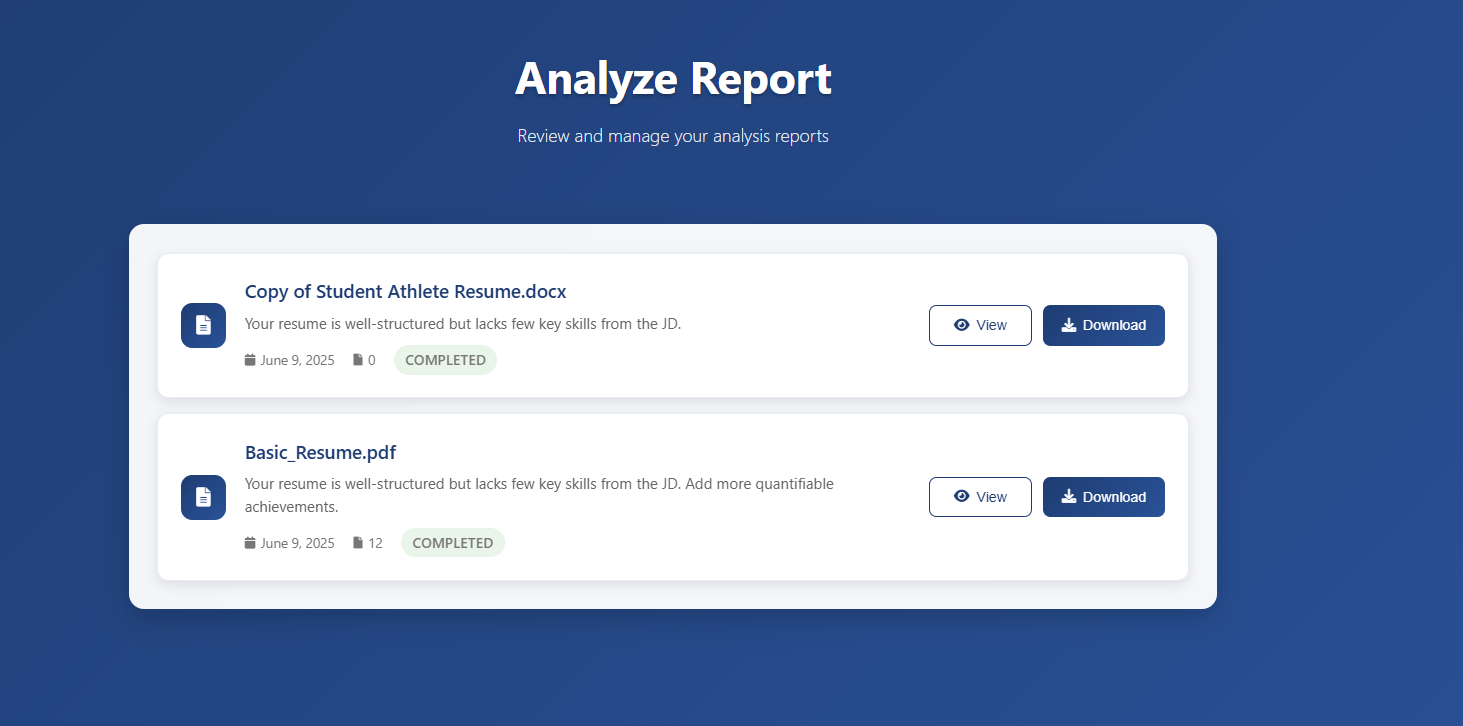
### **📝 Note:**

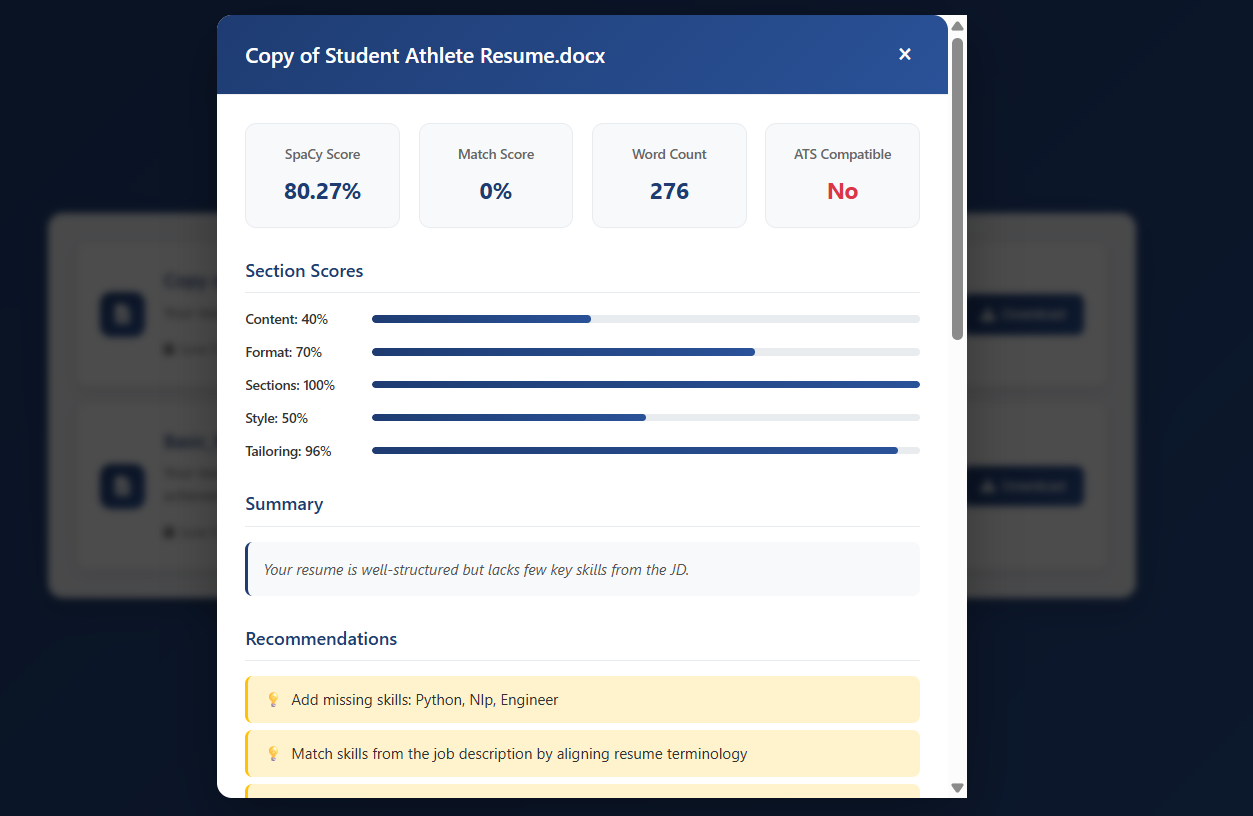
* A .env file and config.py are included in the ZIP archive.
* These files define **port configurations** and are **essential for correct inter-service communication**.
* Do **not change the port numbers** unless you also update them in the frontend and backend code accordingly.

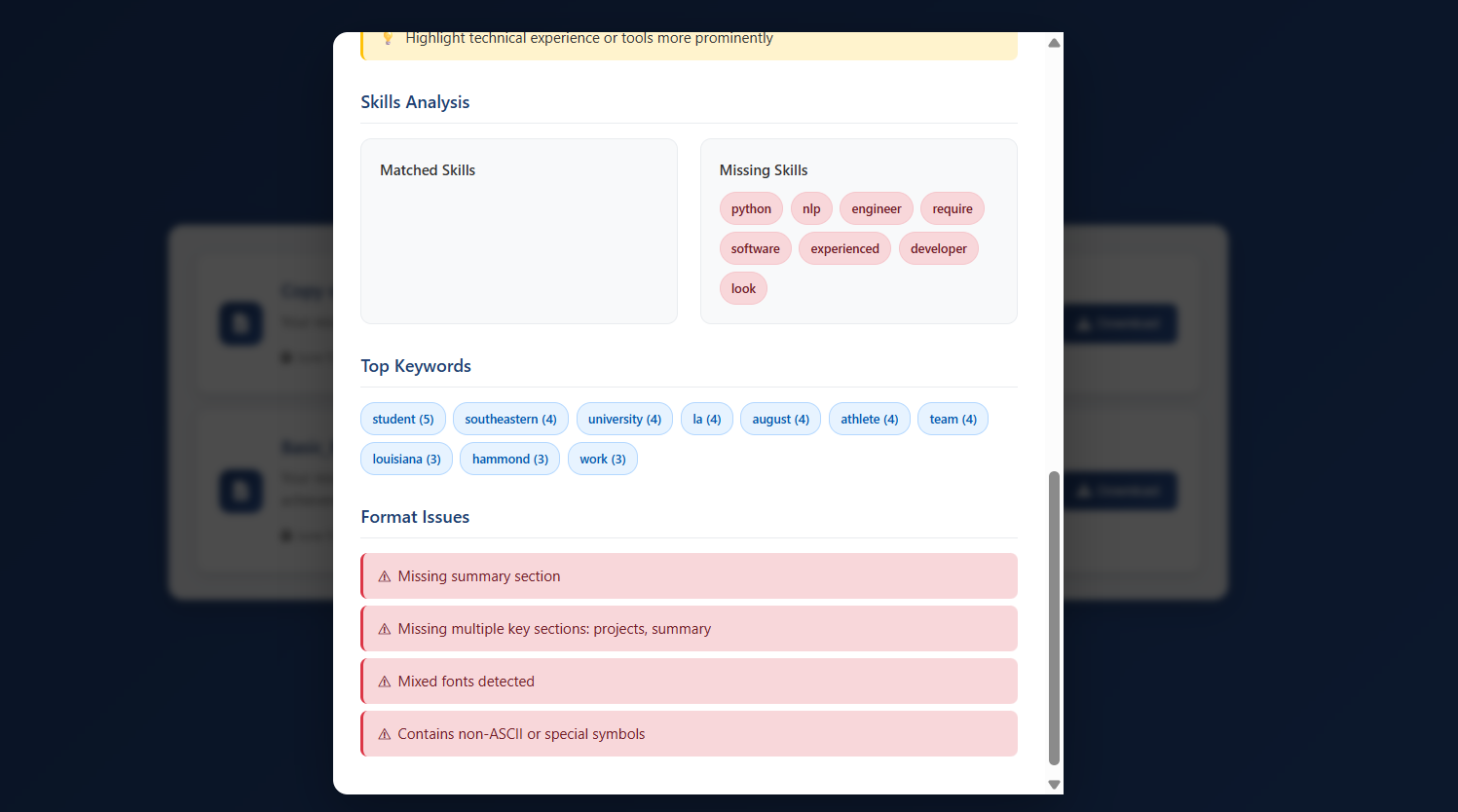
# Screenshots of UI











**#Downloaded Sample Report**

