

Create Epic Stories

EPIC Name: Automated Resume Upload Compliance Tracker



EPIC Description: Prototype a rule-based system to simulate identification of employees who haven't uploaded their resumes in the past 6 months. Use a mock Excel dataset containing upload date, name, employee ID, and email. The system will compare each upload date against the current date to find non-compliant users and generate simulated reminder emails. No APIs are used; this is a mock-driven simulation to validate logic.

Cross-functional Requirement:

- Mock Excel reader with date comparator
- Simulated email trigger logic
- Logging for reminder tracking
- Time-based check without backend API



Step 1 - Feature Breakdown

Key Feature: Resume Upload Verification

- **Example scenario:** The system checks if an employee has uploaded their resume within the last 6 months by comparing the upload date against the current date.
- **Key user stories**
 - As a user, I want the system to read the upload dates from the mock Excel dataset.
 - As a user, I want the system to compare each upload date with the current date to identify non-compliant users.

Key Feature: Email Reminder Generation

- **Example scenario:** The system generates reminder emails for employees who have not uploaded their resumes in the last 6 months.
- **Key user stories**
 - As a user, I want the system to generate simulated reminder emails for non-compliant users.
 - As a user, I want the system to log the actions taken for reminder tracking.

Key Feature: Mock Data Simulation

- **Example scenario:** The system uses mock data from an Excel file for verification and reminders.
- **Key user stories**
 - As a user, I want the system to use mock data to simulate the logic without needing APIs.
 - As a user, I want the system to perform time-based checks on the user compliance data.

Cross-functional Requirement Handling

- **Example scenario:** Implementing mock Excel reader, date comparator, simulated email trigger logic, and logging without backend API involvement.
- **Key user stories**
 - As a user, I want the system to be capable of reading data from an Excel file using a mock reader.
 - As a user, I want the system to include a date comparator to evaluate compliance.
 - As a user, I want the system to simulate email trigger logic based on non-compliance findings.
 - As a user, I want the system to log reminders and tracking actions for transparency.

[Back to bottom](#)

Have I understood your task correctly? Would you like to add anything to this step?



EPIC Name: Skill Extraction & Recency Analyzer from Resume



EPIC Description: Build a prototype that parses mock resume templates to extract skills, infer their recency of use, frequency of occurrence, and approximate experience (in years). The parser will use rule-based logic and section-aware parsing to simulate how recently a skill has been applied. This module does not rely on any LLM/API and will be purely based on offline processing.

Cross-functional Requirement:

- Mock resume corpus for testing
- Sectional rule-based text extraction
- Frequency + recency logic
- Display parsed skill metadata (Streamlit or console)



Step 1 - Feature Breakdown

Key Feature: Resume Parsing for Skill Extraction

- **Example scenario:** The system scans a resume to identify and extract listed skills.
- **Key user stories**
 - As a user, I want the system to parse resumes and identify skills listed in the document.
 - As a user, I want the system to recognize different sections of the resume to enhance accuracy in skill extraction.

Key Feature: Skill Recency Inference

- **Example scenario:** The system determines how recently each skill has been used based on the resume content.
- **Key user stories**
 - As a user, I want the system to infer the recency of each skill use by analyzing context and sections.
 - As a user, I want the system to use rule-based logic to approximate the last usage of each skill.

Key Feature: Skill Frequency Analysis

- **Example scenario:** The system calculates the frequency of skill occurrences within the resume.
- **Key user stories**
 - As a user, I want the system to count how often each skill appears in the resume.
 - As a user, I want the system to provide an analysis of skill frequency across multiple resumes if needed.

Key Feature: Experience Approximation

- **Example scenario:** The system estimates the total years of experience for each skill.
- **Key user stories**
 - As a user, I want the system to approximate the experience in years for each skill mentioned in the resume.
 - As a user, I want the system to calculate experience based on contextual cues and role descriptions.

Cross-functional Requirement Handling

- **Example scenario:** Implementing mock resume corpus testing, sectional rule-based skill extraction, and skill frequency + recency logic without LLM/API.
- **Key user stories**
 - As a user, I want the system to use a mock resume corpus for testing the skill extraction logic.
 - As a user, I want the system to apply sectional rule-based text extraction to parse skills accurately.
 - As a user, I want the system to identify the frequency and recency of skill use based on predefined rules.
 - As a user, I want the parsed skill metadata to be displayed either via Streamlit or the [Back to bottom](#)

Have I understood your task correctly? Would you like to add anything to this step?

EPIC Name: Candidate Fitment Ranking Engine



EPIC Description: Design a prototype that evaluates candidates against a mock job description by ranking them based on skill match, recency, frequency, and experience. The job description and candidate profiles are mock inputs. The ranking algorithm will use a weighted scoring system to simulate realistic fitment scores and produce a shortlist.

Cross-functional Requirement:

- Hardcoded job description input
- Scoring mechanism based on extracted skill metadata
- Ranked output generation
- Optional Streamlit UI for role input and ranking view



Step 1 - Feature Breakdown

Key Feature: Candidate Skill Matching

- **Example scenario:** The system evaluates candidates' skills against the listed requirements in a job description.
- **Key user stories**
 - As a user, I want the system to compare candidate skills with the skills required in the job description.
 - As a user, I want the system to ensure accurate alignment of candidate and job skills by using predefined criteria.

Key Feature: Fitment Scoring Mechanism

- **Example scenario:** The system assigns scores to candidates based on skill match, recency, frequency, and experience using a weighted scoring system.
- **Key user stories**
 - As a user, I want the system to score candidates based on how closely their skills match the job description.
 - As a user, I want the system to factor in recency of skill use, frequency of occurrence, and total experience while scoring.
 - As a user, I want the scoring system to use weights to prioritize different aspects of skill fitment appropriately.

Key Feature: Ranked Output Generation

- **Example scenario:** The system generates a ranked list of candidates based on their fitment scores.
- **Key user stories**
 - As a user, I want the system to produce a shortlist of candidates ranked by their fitment score.
 - As a user, I want the ranked output to be sorted in descending order of fitment scores for easy review.

Key Feature: Interactive UI for Role Input and Ranking View

- **Example scenario:** The system allows users to input job descriptions and view ranked candidates through an optional Streamlit UI.
- **Key user stories**
 - As a user, I want the system to provide an interface for entering job descriptions conveniently.
 - As a user, I want the system to display the ranked candidate list through the Streamlit UI for better visualization.
 - As a user, I want the option to view rankings in the console if the UI is not used.

Cross-functional Requirement Handling

- **Example scenario:** Implementing hardcoded job description input, scoring mechanisms, and ranked output generation without backend API support.
- **Key user stories**
 - As a user, I want the system to accept a hardcoded job description for testing fitment ranking logic.
 - As a user, I want the system to utilize the extracted skill metadata to calculate fitment scores.
 - As a user, I want the system to generate ranked outputs based on the calculated fitment scores for candidates.
 - As a user, I want the option to view ranking results either in a Streamlit UI or console based on requirements.

Staffing Assistant - Resume Parsing

1. Generate gitignore file:

The screenshot shows the VS Code interface with a dark theme. On the left, a code editor displays a .gitignore file for a Python project. The file contains numerous ignore patterns for various temporary and build artifacts. On the right, a "Python Project .gitignore Configuration" panel is open, providing instructions on how to generate a .gitignore file for a Python project. It lists several rules, such as ignoring the .venv directory and environment variable files (.env, .env.*). Below the instructions is a note about ensuring the rules are clean and don't exclude versioned files. At the bottom of the configuration panel, there are buttons for "Accept" and "Reject". The status bar at the bottom of the screen shows "Ln 1, Col 1" and other standard editor settings.

```
#!/.gitignore
# Python virtual environment
.venv/
# Environment variable files
.env
.env.*
# Output and log files
*.log
*.json
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class
# Jupyter Notebook checkpoints
.ipynb_checkpoints/
# macOS system files
.DS_Store
# Test coverage reports
htmlcov/
.coverage
.coverage.-
.cache
nosetests.xml
coverage.xml
*.cover
# pytest cache
.pytest_cache/
# mypy cache
.mypy_cache/
# Pyre type checker
.pyre/
# Build directories
build/
dist/
*.egg-info/
.eggs/
# VSCode settings
.vscode/
# Do not ignore source files in src/
!src/
```

Python Project .gitignore Configuration

Generate a '.gitignore' file for a Python project. The project has the following:

- A '.venv' virtual environment directory
- Uses '.env' files for API keys
- May produce '.log' or '.json' outputs
- Includes build artifacts like '.pyc', '__pycache__', '.DS_Store', and '.ipynb_checkpoints'
- Should ignore test coverage and cache folders
- Source code is inside 'src/' but the '.gitignore' should be placed at the project root

Make sure the rules are clean and don't exclude versioned files

Restore checkpoint

.gitignore (new) +51 -1

```
.venv/
# Environment variable files
.env
.env.*

# Output and log files
*.log
*.json
```

A clean and comprehensive .gitignore file has been generated at your project root. It covers your virtual environment, environment files, log and JSON outputs, build artifacts, test coverage, cache folders, and common system/editor files. It also ensures that your versioned source files in src/ are not excluded.

If you need any customizations or want to add more rules, let me know!

Review changes

> Edited 1 file +51 -1 Reject ⌘N Accept ⌘Y

@ Add context

Plan, search, build anything

Agent gpt-4.1

Ln 1, Col 1 Spaces: 4 UTF-8 LF Ignore Cursor Tab Continue

2. Creating the Project structure for resume_parser:

The screenshot shows a VS Code interface with several tabs open. The main editor tab contains the following Python code:

```

1 def main():
2     print("Resume Parser Main Module")
3
4 if __name__ == "__main__":
5     main()

```

A code completion tooltip is displayed over the line `if __name__ == "__main__":` with the following suggestions:

- src/backend/resume_parser
- __init__.py
- main.py
- reader.py
- extractor.py
- writer.py
- prompt/ folder with:
 - __init__.py
 - reader.py
 - extractor.py
 - writer.py
 - resume_prompt.txt
 - data/ folder to place the resume.docx

The status bar at the bottom indicates the file is `~/de/ATS-ASAP-Code-Custodians` and the cursor is at `main.py | at 02:48:13`.

3. Creating Mock Resume values:



staffing_profile_template_2021_do...
Document

Generate 5 distinct and realistic resume profiles based on the structure and tone of the provided template `staffing_profile_template_2021_doc.docx`.

Each profile should be written in a professional format and structured in plain text so that it can later be converted into a `.docx` resume file using a script.

Requirements for each profile:

1. Each candidate should have a **unique full name**.
2. Vary the **years of experience** (range from 1 to 15 years).
3. Cover different **domains**: Finance, Healthcare, Retail, E-commerce, AI, DevOps, and Data Engineering.
4. Vary **job roles** such as:
 - Application Developer
 - Data Analyst
 - DevOps Engineer
 - Backend Engineer
 - ML Engineer
 - Cloud Architect
5. Include **different projects** (2–4 per resume) with:
 - Project title
 - Role played
 - Duration
 - Tech stack
 - Description
6. Add a clearly marked Skills section grouped into:
 - Programming Languages
 - Frameworks/Tools
 - Databases
 - Cloud Platforms
 - Methodologies
7. Optional but preferred: Include **Certifications**, **Education**, and **Tools**.

📌 Output:
Create 10 separate docx files.

🎯 Objective:
These will be programmatically converted into `.docx` files using `python-docx`, so:

- Do NOT use markdown or HTML.
- Use clear headings and line breaks like the template.
- Maintain a clean, consistent structure across resumes.

📅 Updated saved memory

Here are 5 distinct and realistic resume profiles, structured in plain text following the format and tone of your provided template. These are ready to be converted into `.docx` files using `python-docx`.

Resume 1: Sarah Thomas

Role: Data Analyst

Years of Experience: 3 Years

Domain: Retail

Summary:

Sarah serves Thoughtworks as a Data Analyst with three years of experience in designing dashboards, data pipelines, and statistical models to improve business decision-making. She specializes in SQL, Python, and BI tools, and has worked across diverse datasets in e-commerce and retail environments.

Thoughtworks Experience

Retail Chain Inventory Optimization

April 2023 - Present

Sarah was assigned as a Data Analyst for 12 months, working on improving inventory forecasting using historical sales and seasonality data. She developed regression-based models and Power BI dashboards to assist in restocking decisions.



Analyzed ✓

All five .docx resumes have been successfully generated. You can download them below:

1. Sarah_Thomas.docx
 2. Rahul_Nair.docx
 3. Ayesha_Singh.docx
 4. Vikram_Mehta.docx
 5. Meena_Roy.docx

Would you like to generate more resumes or zip and download all at once? [\[>\]](#)



4. Prompt to Generate the Copy Command:

The screenshot shows a terminal window with several code snippets related to file copying. The snippets are as follows:

```
cp ~/Downloads/Balaji_staffing_profile.docx src/backend/data/
Note: This will overwrite any existing file with the same name in
src/backend/data/. If you want to avoid overwriting, use cp -i
instead:
cp -i ~/Downloads/Balaji_staffing_profile.docx src/backend/data/
```

At the bottom of the terminal window, there is a status bar with the following information:

Review next file /

Problems Output Debug Console Terminal Ports

> cp ~/Downloads/Balaji_staffing_profile.docx src/backend/data/

~/Desktop/ATS-ASAP-Code-Custodians | on main 72 |

ok | myenv py | at 02:56:43

5. Prompt to Generate main.py and reader.py with Good Practices

main.py — ATS-ASAP-Code-Custodians

```

main.py U reader.py 2,U
src > backend > resume_parser > main.py > ...
1 import os
2 from parser.reader import extract_text_from_docx
3
4 def main():
5     print("Resume Parser Main Module")
6
7     Main function to extract and display text from resume.docx in the data folder.
8     Prints the extracted text between clear markers.
9     Raises FileNotFoundError if the file does not exist.
10
11     resume_path = os.path.join(os.path.dirname(__file__), 'data', 'resume.docx')
12     if not os.path.isfile(resume_path):
13         raise FileNotFoundError(f"The file 'resume.docx' was not found in the 'data/' directory: {resume_path}")
14
15     try:
16         resume_text = extract_text_from_docx(resume_path)
17         print("----- START OF RESUME TEXT -----")
18         print(resume_text)
19         print("----- END OF RESUME TEXT -----")
20     except ValueError as e:
21         print(f"Error: {e}")
22
23 if __name__ == "__main__":
24     main()

```

Reject XN Accept XW

Rescue Parser Module Structure Setup

```

reader.py +27-1 *
src > backend > resume_parser > reader.py > ...
1 Join non-empty paragraphs and handles invalid or corrupted files gracefully
2
3
4 main.py +18-1 *
Import os
From parser.reader import extract_text_from_docx
5
6 def main():
7     print("Resume Parser Main Module")
8
9     Main function to extract and display text from resume.docx in the data folder.
10    Prints the extracted text between clear markers.
11    Raises FileNotFoundError if the file does not exist.
12
13    resume_path = os.path.join(os.path.dirname(__file__), 'data', 'resume.docx')
14    if not os.path.isfile(resume_path):

```

Your requirements have been implemented with clean, well-documented, and Pythonic code:

src/backend/resume_parser/parser/reader.py

- Added extract_text_from_docx(filepath: str) -> str:
 - Uses python-docx to extract and clean text from a .docx file.
 - Joins non-empty paragraphs.
 - Handles invalid/corrupted files with clear error messages.
 - Includes a docstring and follows best practices.

src/backend/resume_parser/main.py

- Imports extract_text_from_docx from parser.reader.
- Tries to read resume.docx from the data/ folder.
- Prints the extracted text between ----- START OF RESUME TEXT ----- and ----- END OF RESUME TEXT -----.
- Raises a FileNotFoundError with a meaningful message if the file does not exist.
- Includes a docstring and follows best practices.

You can now run main.py to extract and display the resume text. If you need further enhancements or integration, let me know!

zsh + zsh ... ^ x

Problems 2 Output Debug Console Terminal Ports

cp ~/Downloads/Balaji staffing profile.docx src/backend/data/

~/De/ATS-ASAP-Code-Custodians | on main 72 | ok | myenv.py | at 02:56:43

Review changes

Edited 8 files +65-5 Reject all XG Accept all XW

@ Add context Plan, search, build anything

Agent x gpt-4.1

Ln 22, Col 12 Spaces: 4 UTF-8 LF ↴ Python 3.13.3 64-bit Cursor Tab ⌂ Continue ⌂

reader.py — ATS-ASAP-Code-Custodians

```

main.py U reader.py 2,U
src > backend > resume_parser > parser > reader.py > ...
1 from docx import Document
2 from docx.opc.exceptions import PackageNotFoundError
3
4 class ResumeReader:
5     def read(self, file_path):
6         print("Reading resume from {file_path}")
7
8     def extract_text_from_docx(filepath: str) -> str:
9
10        Extracts and returns the text content from a .docx file as a single string.
11        Joins non-empty paragraphs and handles invalid or corrupted files gracefully.
12
13        Args:
14            filepath (str): Path to the .docx file.
15
16        Returns:
17            str: Cleaned text content of the document.
18
19        Raises:
20            ValueError: If the file is not a valid .docx or is corrupted.
21
22        try:
23            document = Document(filepath)
24            paragraphs = [para.text.strip() for para in document.paragraphs if para.text.strip()]
25            return '\n'.join(paragraphs)
26        except PackageNotFoundError:
27            raise ValueError(f"The file at '{filepath}' is not a valid .docx file or is corrupted.")
28        except Exception as e:
29            raise ValueError(f"An error occurred while reading the .docx file: {e}")

```

Reject XN Accept XW

Rescue Parser Module Structure Setup

```

reader.py +27-1 *
src > backend > resume_parser > reader.py > ...
1 Join non-empty paragraphs and handles invalid or corrupted files gracefully
2
3
4 main.py +18-1 *
Import os
From parser.reader import extract_text_from_docx
5
6 def main():
7     print("Resume Parser Main Module")
8
9     Main function to extract and display text from resume.docx in the data folder.
10    Prints the extracted text between clear markers.
11    Raises FileNotFoundError if the file does not exist.
12
13    resume_path = os.path.join(os.path.dirname(__file__), 'data', 'resume.docx')
14    if not os.path.isfile(resume_path):

```

Your requirements have been implemented with clean, well-documented, and Pythonic code:

src/backend/resume_parser/parser/reader.py

- Added extract_text_from_docx(filepath: str) -> str:
 - Uses python-docx to extract and clean text from a .docx file.
 - Joins non-empty paragraphs.
 - Handles invalid/corrupted files with clear error messages.
 - Includes a docstring and follows best practices.

src/backend/resume_parser/main.py

- Imports extract_text_from_docx from parser.reader.
- Tries to read resume.docx from the data/ folder.
- Prints the extracted text between ----- START OF RESUME TEXT ----- and ----- END OF RESUME TEXT -----.
- Raises a FileNotFoundError with a meaningful message if the file does not exist.
- Includes a docstring and follows best practices.

You can now run main.py to extract and display the resume text. If you need further enhancements or integration, let me know!

zsh + zsh ... ^ x

Problems 2 Output Debug Console Terminal Ports

cp ~/Downloads/Balaji staffing profile.docx src/backend/data/

~/De/ATS-ASAP-Code-Custodians | on main 72 | ok | myenv.py | at 02:56:43

Review changes

Edited 8 files +65-5 Reject all XG Accept all XW

@ Add context Plan, search, build anything

Agent x gpt-4.1

Ln 1, Col 1 Spaces: 4 UTF-8 LF ↴ Python 3.13.3 64-bit Cursor Tab ⌂ Continue ⌂

6. Prompt to create requirement.txt, installing pyenv, setting up Python, and installing from requirements.txt

The screenshot shows a code editor interface with several windows open. On the left, a terminal window displays shell commands for setting up Python 3.11.8 using pyenv. The commands include:

```
# 1. Install pyenv on macOS or Linux
curl https://pyenv.run | bash

# Follow the installation instructions for your system.
# For macOS, you might need to add this line to your shell configuration file:
echo "export PATH=\"$HOME/.pyenv/bin:$PATH\"" >> ~/.bash_profile
source ~/.bash_profile

# If on Linux, it's usually sufficient to run the following command instead:
source /etc/profile.d/pyenv.sh

# 2. Use pyenv to install Python 3.11.8
pyenv install 3.11.8

# 3. Create and activate a virtual environment in src/backend/resume_parser/.venv
cd src/backend/resume_parser
python -m venv .venv
source .venv/bin/activate

# If you are on Windows, replace the above command with:
.\.venv\Scripts\activate

# 4. Install dependencies listed in src/backend/resume_parser/requirements.txt using pip
pip install -r requirements.txt
```

Below the terminal, a note says: "After running these commands, your environment should be set up for Python 3.11.8 with".

In the center, there is a "requirements.txt" file containing:

```
python-docx==0.8.11
openai==1.14.3
pandas==2.2.2
argparse==1.4.0
```

To the right, a "Resume Parser Module Structure Setup" window provides instructions for generating a requirements.txt file. It lists dependencies and their purposes:

- * python-docx* for reading ".docx" files
- * openai* for LLM API usage
- * pandas* for CSV writing
- * argparse* for CLI handling (optional but standard)
- * Minimal, clean, pinned versions (no extra dev tools)

It also suggests saving the content to a root requirements.txt file.

At the bottom, a status bar shows: Edited 8 files +65-5, Local Assistant, Add context, Plan, search, build anything, Agent x1, gpt-4.1, Ln 4, Col 16, Spaces: 4, UTF-8, LF, pip requirements, Cursor Tab, Continue.

7. Refactoring code:

```

1 import os
2 from parser.reader import extract_text_from_docx
3
4 def main():
5     """
6         Main function to extract and display text from resume.docx in the data folder.
7         Prints the extracted text between clear markers.
8         Raises FileNotFoundError if the file does not exist.
9     """
10    Main function to extract and display text from all .docx resumes in the data folder.
11    Prints each resume's content between clear markers with the filename.
12    Skips non-.docx files and handles corrupted files gracefully.
13
14    resume_path = os.path.join(os.path.dirname(__file__), 'data', 'resume.docx')
15    if not os.path.isfile(resume_path):
16        raise FileNotFoundError(f"The file 'resume.docx' was not found in the 'data/' directory: {e}")
17
18    resume_text = extract_text_from_docx(resume_path)
19    print("----- START OF RESUME TEXT -----")
20    print(resume_text)
21    print("----- END OF RESUME TEXT -----")
22
23    except ValueError as e:
24        print(f"[ERROR] Skipping '{filename}': {e}")
25        continue
26
27    resume_texts = []
28
29    for filename in os.listdir(data_dir):
30        if not filename.lower().endswith('.docx'):
31            continue # Skip non-.docx files
32        file_path = os.path.join(data_dir, filename)
33
34        try:
35            resume_text = extract_text_from_docx(file_path)
36            resume_texts.append((filename, resume_text))
37        except ValueError as e:
38            print(f"[ERROR] Skipping '{filename}': {e}")
39            continue
40
41    resume_path = os.path.join(os.path.dirname(__file__), 'data', 'resume.docx')
42    if not os.path.isfile(resume_path):
43        raise FileNotFoundError(f"The 'data/' directory was not found: {data_dir}")
44
45    resume_texts = []
46
47    for filename, resume_text in resume_texts:
48
49        resume_text = extract_text_from_docx(resume_path)
50        print("----- START OF RESUME TEXT -----")
51        print(resume_text)
52        print("----- END OF RESUME TEXT -----")
53
54        except ValueError as e:
55            print(f"[ERROR] Skipping '{filename}': {e}")
56            continue
57
58    resume_texts = []
59
60    for filename, resume_text in resume_texts:
61
62        resume_text = extract_text_from_docx(resume_path)
63        print("----- START OF RESUME TEXT -----")
64        print(resume_text)
65        print("----- END OF RESUME TEXT -----")
66
67        except ValueError as e:
68            print(f"[ERROR] Skipping '{filename}': {e}")
69            continue
70
71    resume_texts = []
72
73    for filename, resume_text in resume_texts:
74
75        resume_text = extract_text_from_docx(resume_path)
76        print("----- START OF RESUME TEXT -----")
77        print(resume_text)
78        print("----- END OF RESUME TEXT -----")
79
80        except ValueError as e:
81            print(f"[ERROR] Skipping '{filename}': {e}")
82            continue
83
84    resume_texts = []
85
86    for filename, resume_text in resume_texts:
87
88        resume_text = extract_text_from_docx(resume_path)
89        print("----- START OF RESUME TEXT -----")
90        print(resume_text)
91        print("----- END OF RESUME TEXT -----")
92
93        except ValueError as e:
94            print(f"[ERROR] Skipping '{filename}': {e}")
95            continue
96
97    resume_texts = []
98
99    for filename, resume_text in resume_texts:
100
101        resume_text = extract_text_from_docx(resume_path)
102        print("----- START OF RESUME TEXT -----")
103        print(resume_text)
104        print("----- END OF RESUME TEXT -----")
105
106        except ValueError as e:
107            print(f"[ERROR] Skipping '{filename}': {e}")
108            continue
109
110    resume_texts = []
111
112    for filename, resume_text in resume_texts:
113
114        resume_text = extract_text_from_docx(resume_path)
115        print("----- START OF RESUME TEXT -----")
116        print(resume_text)
117        print("----- END OF RESUME TEXT -----")
118
119        except ValueError as e:
120            print(f"[ERROR] Skipping '{filename}': {e}")
121            continue
122
123    resume_texts = []
124
125    for filename, resume_text in resume_texts:
126
127        resume_text = extract_text_from_docx(resume_path)
128        print("----- START OF RESUME TEXT -----")
129        print(resume_text)
130        print("----- END OF RESUME TEXT -----")
131
132        except ValueError as e:
133            print(f"[ERROR] Skipping '{filename}': {e}")
134            continue
135
136    resume_texts = []
137
138    for filename, resume_text in resume_texts:
139
140        resume_text = extract_text_from_docx(resume_path)
141        print("----- START OF RESUME TEXT -----")
142        print(resume_text)
143        print("----- END OF RESUME TEXT -----")
144
145        except ValueError as e:
146            print(f"[ERROR] Skipping '{filename}': {e}")
147            continue
148
149    resume_texts = []
150
151    for filename, resume_text in resume_texts:
152
153        resume_text = extract_text_from_docx(resume_path)
154        print("----- START OF RESUME TEXT -----")
155        print(resume_text)
156        print("----- END OF RESUME TEXT -----")
157
158        except ValueError as e:
159            print(f"[ERROR] Skipping '{filename}': {e}")
160            continue
161
162    resume_texts = []
163
164    for filename, resume_text in resume_texts:
165
166        resume_text = extract_text_from_docx(resume_path)
167        print("----- START OF RESUME TEXT -----")
168        print(resume_text)
169        print("----- END OF RESUME TEXT -----")
170
171        except ValueError as e:
172            print(f"[ERROR] Skipping '{filename}': {e}")
173            continue
174
175    resume_texts = []
176
177    for filename, resume_text in resume_texts:
178
179        resume_text = extract_text_from_docx(resume_path)
180        print("----- START OF RESUME TEXT -----")
181        print(resume_text)
182        print("----- END OF RESUME TEXT -----")
183
184        except ValueError as e:
185            print(f"[ERROR] Skipping '{filename}': {e}")
186            continue
187
188    resume_texts = []
189
190    for filename, resume_text in resume_texts:
191
192        resume_text = extract_text_from_docx(resume_path)
193        print("----- START OF RESUME TEXT -----")
194        print(resume_text)
195        print("----- END OF RESUME TEXT -----")
196
197        except ValueError as e:
198            print(f"[ERROR] Skipping '{filename}': {e}")
199            continue
200
201    resume_texts = []
202
203    for filename, resume_text in resume_texts:
204
205        resume_text = extract_text_from_docx(resume_path)
206        print("----- START OF RESUME TEXT -----")
207        print(resume_text)
208        print("----- END OF RESUME TEXT -----")
209
210        except ValueError as e:
211            print(f"[ERROR] Skipping '{filename}': {e}")
212            continue
213
214    resume_texts = []
215
216    for filename, resume_text in resume_texts:
217
218        resume_text = extract_text_from_docx(resume_path)
219        print("----- START OF RESUME TEXT -----")
220        print(resume_text)
221        print("----- END OF RESUME TEXT -----")
222
223        except ValueError as e:
224            print(f"[ERROR] Skipping '{filename}': {e}")
225            continue
226
227    resume_texts = []
228
229    for filename, resume_text in resume_texts:
230
231        resume_text = extract_text_from_docx(resume_path)
232        print("----- START OF RESUME TEXT -----")
233        print(resume_text)
234        print("----- END OF RESUME TEXT -----")
235
236        except ValueError as e:
237            print(f"[ERROR] Skipping '{filename}': {e}")
238            continue
239
240    resume_texts = []
241
242    for filename, resume_text in resume_texts:
243
244        resume_text = extract_text_from_docx(resume_path)
245        print("----- START OF RESUME TEXT -----")
246        print(resume_text)
247        print("----- END OF RESUME TEXT -----")
248
249        except ValueError as e:
250            print(f"[ERROR] Skipping '{filename}': {e}")
251            continue
252
253    resume_texts = []
254
255    for filename, resume_text in resume_texts:
256
257        resume_text = extract_text_from_docx(resume_path)
258        print("----- START OF RESUME TEXT -----")
259        print(resume_text)
260        print("----- END OF RESUME TEXT -----")
261
262        except ValueError as e:
263            print(f"[ERROR] Skipping '{filename}': {e}")
264            continue
265
266    resume_texts = []
267
268    for filename, resume_text in resume_texts:
269
270        resume_text = extract_text_from_docx(resume_path)
271        print("----- START OF RESUME TEXT -----")
272        print(resume_text)
273        print("----- END OF RESUME TEXT -----")
274
275        except ValueError as e:
276            print(f"[ERROR] Skipping '{filename}': {e}")
277            continue
278
279    resume_texts = []
280
281    for filename, resume_text in resume_texts:
282
283        resume_text = extract_text_from_docx(resume_path)
284        print("----- START OF RESUME TEXT -----")
285        print(resume_text)
286        print("----- END OF RESUME TEXT -----")
287
288        except ValueError as e:
289            print(f"[ERROR] Skipping '{filename}': {e}")
290            continue
291
292    resume_texts = []
293
294    for filename, resume_text in resume_texts:
295
296        resume_text = extract_text_from_docx(resume_path)
297        print("----- START OF RESUME TEXT -----")
298        print(resume_text)
299        print("----- END OF RESUME TEXT -----")
300
301        except ValueError as e:
302            print(f"[ERROR] Skipping '{filename}': {e}")
303            continue
304
305    resume_texts = []
306
307    for filename, resume_text in resume_texts:
308
309        resume_text = extract_text_from_docx(resume_path)
310        print("----- START OF RESUME TEXT -----")
311        print(resume_text)
312        print("----- END OF RESUME TEXT -----")
313
314        except ValueError as e:
315            print(f"[ERROR] Skipping '{filename}': {e}")
316            continue
317
318    resume_texts = []
319
320    for filename, resume_text in resume_texts:
321
322        resume_text = extract_text_from_docx(resume_path)
323        print("----- START OF RESUME TEXT -----")
324        print(resume_text)
325        print("----- END OF RESUME TEXT -----")
326
327        except ValueError as e:
328            print(f"[ERROR] Skipping '{filename}': {e}")
329            continue
330
331    resume_texts = []
332
333    for filename, resume_text in resume_texts:
334
335        resume_text = extract_text_from_docx(resume_path)
336        print("----- START OF RESUME TEXT -----")
337        print(resume_text)
338        print("----- END OF RESUME TEXT -----")
339
340        except ValueError as e:
341            print(f"[ERROR] Skipping '{filename}': {e}")
342            continue
343
344    resume_texts = []
345
346    for filename, resume_text in resume_texts:
347
348        resume_text = extract_text_from_docx(resume_path)
349        print("----- START OF RESUME TEXT -----")
350        print(resume_text)
351        print("----- END OF RESUME TEXT -----")
352
353        except ValueError as e:
354            print(f"[ERROR] Skipping '{filename}': {e}")
355            continue
356
357    resume_texts = []
358
359    for filename, resume_text in resume_texts:
360
361        resume_text = extract_text_from_docx(resume_path)
362        print("----- START OF RESUME TEXT -----")
363        print(resume_text)
364        print("----- END OF RESUME TEXT -----")
365
366        except ValueError as e:
367            print(f"[ERROR] Skipping '{filename}': {e}")
368            continue
369
370    resume_texts = []
371
372    for filename, resume_text in resume_texts:
373
374        resume_text = extract_text_from_docx(resume_path)
375        print("----- START OF RESUME TEXT -----")
376        print(resume_text)
377        print("----- END OF RESUME TEXT -----")
378
379        except ValueError as e:
380            print(f"[ERROR] Skipping '{filename}': {e}")
381            continue
382
383    resume_texts = []
384
385    for filename, resume_text in resume_texts:
386
387        resume_text = extract_text_from_docx(resume_path)
388        print("----- START OF RESUME TEXT -----")
389        print(resume_text)
390        print("----- END OF RESUME TEXT -----")
391
392        except ValueError as e:
393            print(f"[ERROR] Skipping '{filename}': {e}")
394            continue
395
396    resume_texts = []
397
398    for filename, resume_text in resume_texts:
399
400        resume_text = extract_text_from_docx(resume_path)
401        print("----- START OF RESUME TEXT -----")
402        print(resume_text)
403        print("----- END OF RESUME TEXT -----")
404
405        except ValueError as e:
406            print(f"[ERROR] Skipping '{filename}': {e}")
407            continue
408
409    resume_texts = []
410
411    for filename, resume_text in resume_texts:
412
413        resume_text = extract_text_from_docx(resume_path)
414        print("----- START OF RESUME TEXT -----")
415        print(resume_text)
416        print("----- END OF RESUME TEXT -----")
417
418        except ValueError as e:
419            print(f"[ERROR] Skipping '{filename}': {e}")
420            continue
421
422    resume_texts = []
423
424    for filename, resume_text in resume_texts:
425
426        resume_text = extract_text_from_docx(resume_path)
427        print("----- START OF RESUME TEXT -----")
428        print(resume_text)
429        print("----- END OF RESUME TEXT -----")
430
431        except ValueError as e:
432            print(f"[ERROR] Skipping '{filename}': {e}")
433            continue
434
435    resume_texts = []
436
437    for filename, resume_text in resume_texts:
438
439        resume_text = extract_text_from_docx(resume_path)
440        print("----- START OF RESUME TEXT -----")
441        print(resume_text)
442        print("----- END OF RESUME TEXT -----")
443
444        except ValueError as e:
445            print(f"[ERROR] Skipping '{filename}': {e}")
446            continue
447
448    resume_texts = []
449
450    for filename, resume_text in resume_texts:
451
452        resume_text = extract_text_from_docx(resume_path)
453        print("----- START OF RESUME TEXT -----")
454        print(resume_text)
455        print("----- END OF RESUME TEXT -----")
456
457        except ValueError as e:
458            print(f"[ERROR] Skipping '{filename}': {e}")
459            continue
460
461    resume_texts = []
462
463    for filename, resume_text in resume_texts:
464
465        resume_text = extract_text_from_docx(resume_path)
466        print("----- START OF RESUME TEXT -----")
467        print(resume_text)
468        print("----- END OF RESUME TEXT -----")
469
470        except ValueError as e:
471            print(f"[ERROR] Skipping '{filename}': {e}")
472            continue
473
474    resume_texts = []
475
476    for filename, resume_text in resume_texts:
477
478        resume_text = extract_text_from_docx(resume_path)
479        print("----- START OF RESUME TEXT -----")
480        print(resume_text)
481        print("----- END OF RESUME TEXT -----")
482
483        except ValueError as e:
484            print(f"[ERROR] Skipping '{filename}': {e}")
485            continue
486
487    resume_texts = []
488
489    for filename, resume_text in resume_texts:
490
491        resume_text = extract_text_from_docx(resume_path)
492        print("----- START OF RESUME TEXT -----")
493        print(resume_text)
494        print("----- END OF RESUME TEXT -----")
495
496        except ValueError as e:
497            print(f"[ERROR] Skipping '{filename}': {e}")
498            continue
499
500    resume_texts = []
501
502    for filename, resume_text in resume_texts:
503
504        resume_text = extract_text_from_docx(resume_path)
505        print("----- START OF RESUME TEXT -----")
506        print(resume_text)
507        print("----- END OF RESUME TEXT -----")
508
509        except ValueError as e:
510            print(f"[ERROR] Skipping '{filename}': {e}")
511            continue
512
513    resume_texts = []
514
515    for filename, resume_text in resume_texts:
516
517        resume_text = extract_text_from_docx(resume_path)
518        print("----- START OF RESUME TEXT -----")
519        print(resume_text)
520        print("----- END OF RESUME TEXT -----")
521
522        except ValueError as e:
523            print(f"[ERROR] Skipping '{filename}': {e}")
524            continue
525
526    resume_texts = []
527
528    for filename, resume_text in resume_texts:
529
530        resume_text = extract_text_from_docx(resume_path)
531        print("----- START OF RESUME TEXT -----")
532        print(resume_text)
533        print("----- END OF RESUME TEXT -----")
534
535        except ValueError as e:
536            print(f"[ERROR] Skipping '{filename}': {e}")
537            continue
538
539    resume_texts = []
540
541    for filename, resume_text in resume_texts:
542
543        resume_text = extract_text_from_docx(resume_path)
544        print("----- START OF RESUME TEXT -----")
545        print(resume_text)
546        print("----- END OF RESUME TEXT -----")
547
548        except ValueError as e:
549            print(f"[ERROR] Skipping '{filename}': {e}")
550            continue
551
552    resume_texts = []
553
554    for filename, resume_text in resume_texts:
555
556        resume_text = extract_text_from_docx(resume_path)
557        print("----- START OF RESUME TEXT -----")
558        print(resume_text)
559        print("----- END OF RESUME TEXT -----")
560
561        except ValueError as e:
562            print(f"[ERROR] Skipping '{filename}': {e}")
563            continue
564
565    resume_texts = []
566
567    for filename, resume_text in resume_texts:
568
569        resume_text = extract_text_from_docx(resume_path)
570        print("----- START OF RESUME TEXT -----")
571        print(resume_text)
572        print("----- END OF RESUME TEXT -----")
573
574        except ValueError as e:
575            print(f"[ERROR] Skipping '{filename}': {e}")
576            continue
577
578    resume_texts = []
579
580    for filename, resume_text in resume_texts:
581
582        resume_text = extract_text_from_docx(resume_path)
583        print("----- START OF RESUME TEXT -----")
584        print(resume_text)
585        print("----- END OF RESUME TEXT -----")
586
587        except ValueError as e:
588            print(f"[ERROR] Skipping '{filename}': {e}")
589            continue
590
591    resume_texts = []
592
593    for filename, resume_text in resume_texts:
594
595        resume_text = extract_text_from_docx(resume_path)
596        print("----- START OF RESUME TEXT -----")
597        print(resume_text)
598        print("----- END OF RESUME TEXT -----")
599
599

```

Your main.py script is now updated to:

- Read all .docx files from the data/ folder.
- Extract text from each file using extract_text_from_docx.
- Store results as (filename, resume_text) in resume_texts.
- Print each resume's content between markers with the filename.
- Skip non-.docx files and handle corrupted files with error messages.
- Maintain a clean, Pythonic structure with comments and robust error handling.

Let me know if you want to add CSV export, OpenAI integration, or further enhancements!

Review changes

Edited 8 files +76 -5

Reject all ⌘⌫ Accept all ⌘⏎

Add context

Plan, search, build anything

Agent #1 gpt-4.1

Ln 14, Col 52 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

7. Prompt to Ask Cursor to Generate the LLM Prompt:

Chat with Haiven

AI model: GPT-4o on Azure | AI-generated content can be inaccurate—validate all important information. Do not include client confidential information or personal data in your inputs. Review our guidelines [here](#).

Generate a prompt for an LLM (like ChatGPT or Groq) that takes plain resume text as input and returns structured data in a specific JSON format.

The prompt should:

- Ask the LLM to parse key sections from a resume such as personal info, summary, skills, experience, projects, etc.
- Specifically extract and format the skills section like this: { "SkillName": { "count": number of distinct projects where the skill was used, "age": number of months since the skill was last used (0 if currently used), "experience_years": total years the skill was used } }
- Include instructions for:
 - work experience: job title, company, dates, responsibilities, technologies used
 - education, certifications, projects, languages, awards, interests, references
- Return output in clean JSON following the example provided
- Keep the formatting LLM-friendly and include placeholders where needed

End the prompt with:

```
"----- Resume Text Starts -----  
{resume_text}  
----- Resume Text Ends -----"
```



Please parse the provided resume text and extract key sections such as personal info, summary, skills, experience, education, and projects.

Skills Section:

Extract and format each skill with the following properties:

```
```json  
{
 "SkillName": {
 "count": number of distinct projects where the skill was used,
 "age": number of months since the skill was last used (0 if currently used),
 "experience_years": total years the skill was used
 }
}
```

[Back to bottom](#)

Type a message...

SEND



```
⌚ main.py U ⌚ resume_prompt.txt U X | ⌚ ...
src > backend > resume_parser > prompt > resume_prompt.txt
1 Please parse the provided resume text and extract the following sections. Return the result as a **single well-formatted JSON object**.
2 ---
3
4
5 ## 1. Personal Information
6 - full_name
7 - contact: { phone, email, location, linkedin/github/portfolio }
8 - availability or work authorization (if mentioned)
9
10 ## 2. Professional Summary
11 - A brief paragraph summarizing experience, skills, and domain focus
12
13 ## 3. Skills
14 Return as an object where each skill is a key, and value is an object with:
15 {
16 "SkillName": {
17 "count": number of distinct projects the skill was used in,
18 "age": number of months since the skill was last used (0 if currently active),
19 "experience_years": total estimated years of using the skill
20 }
21 }
22
23 ## 4. Work Experience
24 A list of jobs/projects with:
25 - job_title
26 - company
27 - location (if available)
28 - start_date, end_date (Month YYYY format)
29 - responsibilities (as bullet points)
30 - technologies_used (list)
31
32 ## 5. Education
33 A list with:
34 - degree
35 - institution
36 - start_date, end_date
37 - field_of_study
38
39 ## 6. Certifications
40 A list with:
41 - name
42 - issuing_organization
43 - issue_date, expiry_date (if available)
44
45 ## 7. Projects
46 A list with:
47 - title
48 - description
49 - technologies_used
50 - role
51 - start_date, end_date
52 - link (if available)
53
54 ## 8. Languages
```

1 / 1 ▾ Reject file ↵ Accept file ↵ < 3 / 3 files >

Ln 35, Col 14 Spaces: 4 UT

## 8. Prompt to Generate Config File + Dataclass + Main Update:

The screenshot shows a code editor interface with a sidebar for refactoring. The main area displays code changes for 'main.py' and the creation of 'config.yaml'. A sidebar on the right provides instructions for refactoring.

**Refactor the Python script 'main.py' in the folder 'src/backend/resume\_parser/' to use a configuration file and a dataclass for cleaner config management.**

**Make the following changes:**

1. Create a file named 'config.yaml' inside 'src/backend/resume\_parser/' with the following contents:

```
yaml
data_folder: "data"
resume_extension: ".docx"
```

**! config.yaml (new) \***

```
data_folder: "data"
resume_extension: ".docx"
```

**! main.py +26 -6 \***

```
import os
import yaml
from dataclasses import dataclass
from parser.reader import extract_text_from_docx

@dataclass
class ResumeParserConfig:
 data_folder: str
 resume_extension: str

 def load_config(config_path: str) -> ResumeParserConfig:
 ...
```

Your script is now refactored to use a YAML configuration file and a dataclass for configuration management:

- A new `config.yaml` is created in `src/backend/resume_parser/` with:  
! yaml  
data\_folder: "data"  
resume\_extension: ".docx"
- `main.py` now:
  - Loads config from `config.yaml` using PyYAML.
  - Uses a `ResumeParserConfig` dataclass for type-safe config access.
  - Uses the config values for folder and extension handling.

Edited 9 files +98 -6

Reject all ⌂ Accept all ⌂

Add context

Plan, search, build anything

Agent gpt-4.1

Ln 2, Col 27 Spaces: 2 UTF-8 LF YAML Cursor Tab Continue

## 9. Prompt to Generate extractor.py

```

writer.py U extractor.py 1, U ...
src > backend > resume_parser > parser > extractor.py > ...
1 import os
2 import json
3 from typing import Dict
4 import openai
5
6 def extract_resume_json(resume_text: str, prompt_path: str) -> Dict:
7 """
8 Extract structured resume information as a JSON dictionary using an LLM.
9
10 Args:
11 resume_text (str): The raw text of the resume to be parsed.
12 prompt_path (str): Path to the prompt template file containing a '{{resume_text}}' placeholder.
13
14 Returns:
15 dict: Parsed resume information as a Python dictionary.
16
17 Raises:
18 RuntimeError: If the API call fails or the response cannot be parsed as JSON.
19 """
20
21 # Load API key from environment
22 api_key = os.getenv("OPENAI_API_KEY")
23 if not api_key:
24 raise RuntimeError("OPENAI_API_KEY environment variable not set.")
25 openai.api_key = api_key
26
27 # Load and prepare the prompt
28 try:
29 with open(prompt_path, "r", encoding="utf-8") as f:
30 prompt_template = f.read()
31 except Exception as e:
32 raise RuntimeError(f"Failed to read prompt template: {e}")
33
34 prompt = prompt_template.replace("{{resume_text}}", resume_text)
35
36 # Call OpenAI API
37 try:
38 response = openai.ChatCompletion.create(
39 model="gpt-3.5-turbo",
40 messages=[{"role": "user", "content": prompt}],
41 temperature=0.0,
42 max_tokens=2048
43)
44 content = response['choices'][0]['message']['content']
45 except Exception as e:
46 raise RuntimeError(f"OpenAI API call failed: {e}")
47
48 # Parse JSON output
49 try:
50 result = json.loads(content)
51 except Exception as e:
52 raise RuntimeError(f"Output was: {content}")
53
54 return result

```

**New chat**

- Define a function `extract\_resume\_json(resume\_text: str, prompt\_path: str) -> dict`  
- It should:
  - Load the prompt template from the file at `prompt\_path`
  - Replace the placeholder `{{resume\_text}}` with the actual `resume\_text`
  - Send the final prompt to an LLM using OpenAI (or GPT) API
  - Parse the JSON output from the LLM into a Python dictionary and return it
- Handle basic API error handling and response parsing (assume OpenAI for now).

Restored checkpoint

Listed 4 items in src/backend/resume\_parser/parser

```

extractor.py +55 *
import os
import json
from typing import Dict
import openai

def extract_resume_json(resume_text: str, prompt_path: str) -> Dict:
 """
 Extract structured resume information as a JSON dictionary using an LLM.

 Args:
 resume_text (str): The raw text of the resume to be parsed.
 """

The extractor.py module has been created with the extract_resume_json function as specified. It includes:
 • Loading the prompt template and replacing the {{resume_text}} placeholder.
 • Sending the prompt to the OpenAI API (using the key from OPENAI_API_KEY).
 • Parsing the LLM's JSON output into a Python dictionary.
 • Error handling for file I/O, API errors, and JSON parsing.
 • Clean code, type hints, and docstrings.

Let me know if you need tests, usage examples, or further adjustments!

```

Review changes

Edited 1 file +55

Add context

Plan, search, build anything

Agent gpt-4.1

Ln 58, Col 58 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

## 10. Integrate extractor.py into [main.py](#)

```

extractor.py 1, U ● main.py 1, U ●
src > backend > resume_parser > main.py > main
24
def main():
25
26 config_path = os.path.join(os.path.dirname(__file__), 'config.yaml')
27 config = load_config(config_path)
28 data_dir = os.path.join(os.path.dirname(__file__), config.data_folder)
29 if not os.path.isdir(data_dir):
30 raise FileNotFoundError(f"The '{config.data_folder}' directory was not found: {data_dir}")
31
32 resume_texts = [] # List of tuples: (filename, resume_text)
33
34 for filename in os.listdir(data_dir):
35 if not filename.lower().endswith(config.resume_extension.lower()):
36 continue # Skip files that do not match the configured extension
37 file_path = os.path.join(data_dir, filename)
38 try:
39 resume_text = extract_text_from_docx(file_path)
40 resume_texts.append((filename, resume_text))
41 except ValueError as e:
42 print(f"[ERROR] Skipping '{filename}': {e}")
43
44 for filename, resume_text in resume_texts:
45 print("----- START OF {filename} -----")
46 print(resume_text)
47 print("----- END OF {filename} -----\\n")
48
49 # Extract structured JSON using LLM
50 prompt_path = os.path.join(os.path.dirname(__file__), 'prompt', 'resume_prompt.txt')
51 try:
52 parsed_json = extract_resume_json(resume_text, prompt_path)
53 print(f"Extracted JSON for {filename}:")
54 print(parsed_json)
55 except Exception as e:
56 print(f"[ERROR] Failed to extract JSON for '{filename}': {e}")
57 continue
58
59 # Optionally save to output/parsed_{filename}.json
60 output_dir = os.path.join(os.path.dirname(__file__), 'output')
61 os.makedirs(output_dir, exist_ok=True)
62 base_name, _ = os.path.splitext(filename)
63 output_path = os.path.join(output_dir, f"parsed_{base_name}.json")
64 try:
65 with open(output_path, 'w', encoding='utf-8') as f:
66 json.dump(parsed_json, f, ensure_ascii=False, indent=2)
67 except Exception as e:
68 print(f"[ERROR] Could not save JSON for '{filename}': {e}")
69
70
71 if __name__ == "__main__":
72 main()

```

Reject 3/3 | Accept 3/3

Python Module for Resume Data Extraction

5. Keep logging and error handling clean and minimal.

Make sure the code:

- Uses imports only if needed

Searched files "main.py"

Read main.py

Listed 6 items in src/backend/resume\_parser/

Listed 1 items in src/backend/resume\_parser/prompt

main.py +23 \*

```

import os
import yaml
import json
from dataclasses import dataclass
from parser.reader import extract_text_from_docx
from parser.extractor import extract_resume_json

```

@dataclass

```

class ResumeParserConfig:
 resume_text
 print(resume_text)
 print("----- END OF {filename} -----\\n")

```

main.py is now updated to:

- Call `extract_resume_json` for each resume after extracting its text.
- Use the correct `prompt_path` for the prompt template.
- Print the extracted JSON for each resume.
- Save each JSON result to `output/parsed_{filename}.json`, creating the `output/` folder if needed.
- Keep error handling and logging minimal and clear.
- Only import what is necessary and maintain a readable loop.

Let me know if you want to test this or need further changes!

Review changes

Edited 2 files +78

Reject all | Accept all

Add context

Plan, search, build anything

Agent gpt-4.1

Ln 71, Col 71 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

## 11. Refactoring [extractor.py](#) to use Groq API mistral:

Python Module for Groq API Integration

config.yaml extractor.py main.py

Create a minimal and production-grade Python module to call the Groq API using their official endpoint ('<https://api.groq.com/openai/v1/chat/completions>').

Groq\_API\_KEY = "gsk\_h3cVmFl8KXi2GFMcfALWGdyb3FYkdHJ588eNDZJPx8Tepk7W1Fz"

Requirements:

1. Use 'requests' (not 'openai' SDK).
2. Read API key from the 'GROQ\_API\_KEY' environment variable.
3. Function should be named 'extract\_resume\_json\_with\_groq(prompt\_text: str) -> dict'.
4. Model should default to '"gemma-7b-it"', with optional override.
5. It should accept a string 'prompt\_text' and return the parsed JSON response from Groq API.
6. Raise helpful exceptions on:
  - Missing API key
  - Bad status codes

[Restore checkpoint](#)

```

❸ groq_extractor.py (new) +77 -1 •
import os
import requests
import json
from typing import Optional, Dict

GROQ_API_URL = "https://api.groq.com/openai/v1/chat/completions"
DEFAULT_MODEL = "gemma-7b-it"

class GroqAPIError(Exception):
 pass

def extract_resume_json_with_groq(prompt_text: str, model: Optional[str] = None) -> Dict:
 """
 Calls the Groq API with the given prompt_text and returns the parsed JSON response.

 Args:
 prompt_text (str): The prompt to send to the Groq API.
 model (str, optional): The model to use. Defaults to 'gemma-7b-it'.

 Returns:
 dict: The parsed JSON response from the Groq API.

 Raises:
 RuntimeError: If the API key is missing, the API call fails, or the response cannot be parsed as JSON.
 """
 api_key = os.getenv("GROQ_API_KEY")
 if not api_key:
 raise RuntimeError("Groq API key is missing")

 headers = {
 "Content-Type": "application/json",
 "Authorization": f"Bearer {api_key}"
 }

 data = {
 "model": model,
 "prompt": prompt_text
 }

 response = requests.post(GROQ_API_URL, headers=headers, json=data)
 response.raise_for_status()

 return response.json()

```

Edited 1 file +77 -1

Reject ⌘% Accept ⌘=

config.yaml extractor.py main.py

Plan, search, build anything

Agent gpt-4.1

File Cell Kernel Help Home

## 12. Extending requirements.txt using AI

The screenshot shows a code editor interface with several tabs open: `extractor.py`, `requirements.txt`, `main.py`, and `config.yaml`. A modal window titled "JSON Extraction Error Troubleshooting" is displayed, containing the following text:

cleaner and easier to maintain. You can now run your script and it should work as expected. Let me know if you need further changes or if you encounter any other issues!

Below this, another modal window titled "requirements.txt" shows the contents of the file:

```
python-docx==0.8.11
python-dotenv==1.0.1
openai==1.14.3
pandas==2.2.2
pyyaml==6.0.1
requests==2.31.0
```

A message at the bottom of this window says: "The `requests` module has been added to your `requirements.txt` file. You can now install it with your other dependencies using `pip install -r requirements.txt`. Let me know if you need anything else!"

At the bottom of the editor, there are "Reject" and "Accept" buttons.

Below the editor is a terminal window showing the following command and output:

```
> python src/backend/resume_parser/main.py
```

Traceback (most recent call last):  
File "/Users/arghyabenerjee/Desktop/ATS-ASAP-Code-Custodians/src/backend/resume\_parser/main.py", line 6, in <module>  
> from parser.extract import extract\_resume\_json  
File "/Users/arghyabenerjee/Desktop/ATS-ASAP-Code-Custodians/src/backend/resume\_parser/parser/extractor.py", line 3  
, in <module>  
 import requests  
ModuleNotFoundError: No module named 'requests'

~/De/ATS-ASAP-Code-Custodians | on main !26 73 | 1 err | ATS-ASAP-Code-Custodians.py | at 10:49:58

The terminal also shows a status bar with "Edited 3 files +8 -16", "Agent gpt-4.1", and "Ln 7, Col 1 Spaces: 4 UTF-8 LF pip requirements Cursor Tab Continue".

### 13. Fixing bugs:

extractor.py 1, U requirements.txt U main.py 1, U config.yaml U JSON Extraction Error Troubleshooting

```

src > backend > resume_parser > main.py
 def load_config(config_path: str) -> ResumeParserConfig:
 llm_config = config_dict['llm']
 for key in ['provider', 'model_name', 'endpoint', 'api_key']:
 if key not in llm_config:
 raise ValueError(f"Missing required LLM config key: '{key}'")
 return ResumeParserConfig(
 data_folder=config_dict['data_folder'],
 resume_extension=config_dict['resume_extension'],
 llm=LLMConfig(
 provider=llm_config['provider'],
 model_name=llm_config['model_name'],
 endpoint=llm_config['endpoint'],
 api_key=llm_config['api_key'] or os.getenv("GROQ_API_KEY"),
 temperature=llm_config.get('temperature', 0.0)
)
)
 def main():
 config_path = os.path.join(os.path.dirname(__file__), 'config.yaml')
 config = load_config(config_path)
 data_dir = os.path.join(os.path.dirname(__file__), config.data_folder)

 if not os.path.isdir(data_dir):
 raise FileNotFoundError(f"Directory '{config.data_folder}' not found at path: {data_dir}")
 resume_texts = []
 for filename in os.listdir(data_dir):
 if filename.lower().endswith(config.resume_extension.lower()):
 try:
 file_path = os.path.join(data_dir, filename)
 resume_text = open(file_path, 'r').read()
 resume_texts.append((filename, resume_text))
 except Exception as e:
 print(f"Error reading file {file_path}: {e}")
 print(resume_texts)

```

Would you like me to update the script to print the loaded config and handle missing/empty YAML more gracefully? This will help pinpoint the issue if it happens again.

Fix

```

def load_config(config_path: str) -> ResumeParserConfig:
 with open(config_path, 'r') as file:
 config_dict = yaml.safe_load(file)
 print(f"[DEBUG] Loaded config_dict: {config_dict} (type: {type(config_dict)})")
 if not config_dict:
 raise ValueError("Config file is empty or could not be parsed. Please check the file path and file content.")
 if 'data_folder' not in config_dict or 'resume_extension' not in config_dict:
 raise ValueError("Missing required config keys: 'data_folder', 'resume_extension' or 'llm'. Please add them to your config.yaml file.")

I've added debug output to print the loaded YAML config and its type, and improved error handling if the config file is empty or cannot be parsed.
Please run your script again. If the error persists, the debug print will show exactly what is being loaded from config.yaml, which will help us pinpoint the issue. Share the debug output if you need further help!

```

Review changes

python src/backend/resume\_parser/main.py

Traceback (most recent call last):
 File "/Users/arghyabenerjee/Desktop/ATS-ASAP-Code-Custodians/src/backend/resume\_parser/main.py", line 96, in <module>
 main()
 File "/Users/arghyabenerjee/Desktop/ATS-ASAP-Code-Custodians/src/backend/resume\_parser/main.py", line 52, in main
 config = load\_config(config\_path)
 File "/Users/arghyabenerjee/Desktop/ATS-ASAP-Code-Custodians/src/backend/resume\_parser/main.py", line 30, in load\_config
 raise ValueError("Missing required config keys: 'data\_folder', 'resume\_extension', or 'llm'")
ValueError: Missing required config keys: 'data\_folder', 'resume\_extension', or 'llm'

Edited 3 files +12-19 Reject ✖ Accept ✅

extractor.py main.py config.yaml Plan, search, build anything

Agent gpt-4.1

Ln 54, Col 1 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

extractor.py — ATS-ASAP-Code-Custodians

```

! config.yaml main.py 1, U extractor.py 1, U extract_resume_json
src > backend > resume_parser > parser > extractor.py > extract_resume_json
 10 def extract_resume_json():
 11 payload = {
 12 "model": llm_config.model_name,
 13 "messages": [{"role": "user", "content": prompt}],
 14 "temperature": llm_config.temperature,
 15 "max_tokens": 4096
 16 }
 17
 18 try:
 19 response = requests.post(llm_config.endpoint, headers=headers, json=payload, timeout=10)
 20 response.raise_for_status()
 21 content = response.json()["choices"][0]["message"]["content"].strip()
 22 return json.loads(content)
 23 except json.JSONDecodeError as e:
 24 raise GroqAPIError(f"Failed to parse JSON from Groq content: {content}\nError: {e}")
 25 except Exception as e:
 26 raise GroqAPIError(f"Failed to get or parse response from Groq: {e}")
 27
 28 resp_json = response.json()
 29
 30 # Check for expected structure
 31 if (
 32 "choices" not in resp_json or
 33 not resp_json["choices"] or
 34 "message" not in resp_json["choices"][0] or
 35 "content" not in resp_json["choices"][0]["message"]
 36):
 37 raise GroqAPIError(f"Unexpected response structure from Groq: {resp_json}")
 38 content = resp_json["choices"][0]["message"]["content"].strip()
 39 if not content:
 40 raise GroqAPIError(f"Empty content in Groq response: {resp_json}")
 41 try:
 42 return json.loads(content)
 43 except json.JSONDecodeError as parse_err:
 44 raise GroqAPIError(f"Failed to parse JSON from Groq content: {content}\nError: {parse_err}")
 45
 46 except Exception as e:
 47 raise GroqAPIError(f"Failed to get or parse response from Groq: {e}")
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70

```

Fixing JSON Parsing Error in Code

Before calling `json.loads(content)`, ensure `content` is not empty and is valid JSON.

- 2. Improve Error Reporting:**

Log the actual response content if parsing fails, to help with debugging.

- 3. Handle API Errors Gracefully:**

Sometimes the API may return an error message in a different format. Check for this and raise a more informative error.

**Improved Code**

I will update the `extract_resume_json` function to:

- Check if the response contains the expected fields.
- Log the raw response content if parsing fails.
- Raise a more informative error if the response is empty or not as expected.

Let me make these changes.

extractor.py +16 -2

```

res_json = response.json()
Check for expected structure
if (
 "choices" not in res_json or
 not res_json["choices"] or
 "message" not in res_json["choices"][0] or
 "content" not in res_json["choices"][0]["message"]
):
 raise GroqAPIError(f"Unexpected response structure from Groq: {res_json}")
content = res_json["choices"][0]["message"]["content"].strip()
if not content:
 raise GroqAPIError(f"Empty content in Groq response: {res_json}")
try:
 return json.loads(content)
except json.JSONDecodeError as parse_err:
 raise GroqAPIError(f"Failed to parse JSON from Groq content: {content}\nError: {parse_err}")
except Exception as e:
 raise GroqAPIError(f"Failed to get or parse response from Groq: {e}")

```

I've improved the error handling in your `extract_resume_json` function. Now, if the Groq API response is empty, malformed, or not valid JSON, you'll get a much clearer error message—

Edited 1 file +16 -2

Reject all ✖ Accept all ✅

Problems 2 Output Debug Console Terminal Ports

NIT Calicut, 2015  
---- END OF Rahul\_Nair.docx ----  
[ERROR] Failed to extract JSON for Rahul\_Nair.docx: Failed to get or parse response from Groq: Expecting value: line 1 column 1 (char 0)

~De/ATS-ASAP-Code-Custodians | on main !26 73

Ln 52, Col 36 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

## 14. Successful to fix the bug

The screenshot shows a code editor interface with two main panes. The left pane displays a Python file named `extractor.py` containing code for extracting JSON from LLM responses. The right pane shows a GPT-4 generated fix for a JSON parsing error, with a summary of the fix and the corrected code.

```

src > backend > resume_parser > parser > extractor.py > extract_resume_json
10 def extract_resume_json():
11 payload = {
12 "model": llm_config.model_name,
13 "messages": [{"role": "user", "content": prompt}],
14 "temperature": llm_config.temperature,
15 "max_tokens": 4096
16 }
17
18 try:
19 response = requests.post(llm_config.endpoint, headers=headers, json=payload, timeout=5)
20 response.raise_for_status()
21 resp_json = response.json()
22 # Check for expected structure
23 if (
24 "choices" not in resp_json or
25 not resp_json["choices"] or
26 "message" not in resp_json["choices"][0] or
27 "text" not in resp_json["choices"][0]
28):
29 raise GroqAPIError(f"Empty content in Groq API response")
30
31 return resp_json["choices"][0]["text"]
32 except requests.exceptions.RequestException as e:
33 print(f"An error occurred: {e}")
34
35 return None

```

**Fixing JSON Parsing Error in Code**

\*\*Summary of the fix:\*\*  
- If the content starts with triple backticks, extract the text after them.

Let me implement this for you.

```

raise GroqAPIError(f"Empty content in Groq API response")
Strip Markdown code block if present
if content.startswith('```'):
 lines = content.split('\n')
 # Remove the first line ('``' or ```json)
 if lines[0].startswith('```'):
 lines = lines[1:]
 if lines and lines[-1].strip() == '```':
 lines = lines[:-1]
 content = '\n'.join(lines).strip()
return json.loads(content)

```

The function now strips Markdown code block markers (triple backticks and optional `json`) from the LLM response before parsing. This will allow `json.loads` to work even if the response is wrapped in a code block, fixing the error you encountered. You can now re-run your code—the JSON extraction should succeed even if the LLM returns Markdown-wrapped JSON. If you encounter any further issues or edge cases, let me know!

Edited 1 file +25 -2

extractor.py

Review changes

Ln 52, Col 36 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

## 15. Create `writer.py` that convert json to csv files

src > backend > resume\_parser > parser > writer.py > ...

```
1 import pandas as pd
2 from typing import List, Dict, Any
3 from datetime import datetime
4
5 def resumes_to_flat_csv(resumes: List[Dict], output_csv_path: str):
6 rows = []
7 for resume in resumes:
8 if not resume.get('full_name'):
9 continue
10 # Contact
11 contact = resume.get('contact', {}) or {}
12 email = contact.get('email') or ''
13 # Professional summary
14 professional_summary = resume.get('professional_summary') or ''
15 # Skills
16 skills_dict = resume.get('skills') or {}
17 skills = '|'.join(skills_dict.keys()) if isinstance(skills_dict, dict) else ''
18 # Work experience
19 work_experience = resume.get('work_experience') or []
20 recent_job_title = ''
21 recent_company = ''
22 num_jobs = 0
23 if isinstance(work_experience, list) and work_experience:
24 # Sort by end_date (latest first), fallback to first if all end_dates are missing
25 def parse_end_date(job):
26 end = job.get('end_date')
27 if end:
28 try:
29 # Try to parse as year or month-year
30 return datetime.strptime(end, '%Y')
31 except Exception:
32 try:
33 return datetime.strptime(end, '%b %Y')
34 except Exception:
35 return None
36
37 sorted_jobs = sorted(work_experience, key=parse_end_date, reverse=True)
```

The function `resumes_to_flat_csv` has been created in `src/backend/resume_parser/parser/writer.py` as specified. It:

- Flattens each resume JSON into a single CSV row with the required columns.
- Handles missing/null fields safely.
- Skips resumes without a `full_name`.
- Uses pandas to write the CSV with a header.

Let me know if you want an example usage or integration with your main script!

Review changes

Edited 1 file +78 -1 Reject 0% Accept 0%

src > backend > resume\_parser > parser > writer.py > ...

"email": null,

Restore checkpoint

writer.py +78 -1 •

```
import pandas as pd
from typing import List, Dict, Any
from datetime import datetime

def resumes_to_flat_csv(resumes: List[Dict], output_csv_path: str):
 rows = []
 for resume in resumes:
 if not resume.get('full_name'):
 continue
 # Contact
 contact = resume.get('contact', {}) or {}
 email = contact.get('email') or ''
```

Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.13.3 64-bit Cursor Tab Continue

```

main.py 1, M x writer.py 1, M
src > backend > resume_parser > main.py > main
 def main():
 for filename, resume_text in resume_texts:
 print(f"[ERROR] Skipping {filename}: {e}")

 parsed_resumes = []
 for filename, resume_text in resume_texts:
 print(f"\n----- START OF {filename} ----- \n----- END OF {filename} -----")
 prompt_path = os.path.join(os.path.dirname(__file__), 'prompt', 'resume_prompt.txt')
 try:
 parsed_json = extract_resume_json(
 resume_text=resume_text,
 prompt_path=prompt_path,
 llm_config=config.llm
)
 print(f"[SUCCESS] Extracted JSON for {filename}")
 parsed_resumes.append(parsed_json)
 except Exception as e:
 print(f"[ERROR] Failed to extract JSON for {filename}: {e}")
 continue

 output_dir = os.path.join(os.path.dirname(__file__), 'output')
 os.makedirs(output_dir, exist_ok=True)
 base_name, _ = os.path.splitext(filename)
 output_file = os.path.join(output_dir, f"parsed_{base_name}.json")

 output_dir = os.path.join(os.path.dirname(__file__), 'output')
 os.makedirs(output_dir, exist_ok=True)
 output_csv = os.path.join(output_dir, 'parsed_resumes.csv')
 if parsed_resumes:
 try:
 Chat *+L Edit *+I
 with open(output_file, 'w', encoding='utf-8') as f:
 json.dump(parsed_json, f, indent=2, ensure_ascii=False)
 resumes_to_flat_csv(parsed_resumes, output_csv)
 print(f"[SUCCESS] Wrote CSV to {output_csv}")
 except Exception as e:
 print(f"[ERROR] {e}")
 print(f"[ERROR] Could not save CSV: {e}")
 except Exception as e:
 Reject file *+E Accept file *+A
 print(f"[ERROR] {e}")
 print(f"[ERROR] Could not save CSV: {e}")

 Problems 2 Output Debug Console Terminal Ports Comments + ... ^ x
B.Tech in Electronics and Communication
NIT Calicut, 2015
----- END OF Rahul_Nair.docx -----
[SUCCESS] Extracted JSON for Rahul_Nair.docx
~/De/ATS-ASAP-Code-Custodians | on main []

```

Ln 98, Col 54 (207 selected) Spaces: 4 UTF-8 LF (Python 3.13.3 64-bit Cursor Tab ✓ Continue

Convert Resumes to Flat CSV in Python  
 (src/backend/resume\_parser/main.py)  
 Grepped codebase for "import " (\*.py)

```

from dataclasses import dataclass
from parser.reader import extract_text_from_docx
from parser.extractor import extract_resume_json
from parser.writer import resumes_to_flat_csv

@dataclass
59 hidden lines ...
except Exception as e:
 print(f"[ERROR] Skipping {filename}: {e}")

parsed_resumes = []
```

Your script now writes all parsed resumes to a single flat CSV file using the new `resumes_to_flat_csv` function, instead of saving individual JSON files. The changes:

- Import `resumes_to_flat_csv` from `parser.writer`.
- Collect all parsed JSONs into a list (`parsed_resumes`).
- After processing all resumes, write them to `output/parsed_resumes.csv` using the CSV writer.
- Removed the per-resume JSON writing.

You can now run your script and get a single, clean CSV output for all resumes! Let me know if you want to further customize the output or need help with anything else.

Review changes

Edited 2 files +88-9 Reject ⌂ Accept ⌂

@ writer.py Plan, search, build anything

Agent \*+I gpt-4.1

## 16. Generate commit msg using Copilot

The screenshot shows a code editor interface with a source control panel on the left and a code editor on the right. The source control panel displays a commit message: "feat: implement resume parser with Groq LLM integration and configuration". Below it is a list of changes, all marked as 'U' (Untracked). The code editor shows a Python file named "main.py" with the following content:

```
src > backend > resume_parser > main.py
54 def main():
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...

 prompt_path = os.path.join(resume_text_dir, f'{base_name}.json')
 parsed_json = json.load(open(prompt_path))
 prompt_path = os.path.join(resume_text_dir, f'{base_name}_llm_config.json')
 llm_config = json.load(open(prompt_path))

 try:
 response = requests.post(
 url="https://api.groq.com/v1/completions",
 headers=headers,
 json=prompt
)
 response.raise_for_status()
 print(f"[SUCCESS] Response: {response.json()}")
 except Exception as e:
 print(f"[ERROR] {e}")
 continue

 output_dir = os.path.join(resume_text_dir, base_name)
 os.makedirs(output_dir, exist_ok=True)
 output_file = os.path.join(output_dir, f'{base_name}.json')

 try:
 with open(output_file, 'w') as f:
 json.dump(parsed_json, f)
 except Exception as e:
 print(f"[ERROR] {e}")

 print(f"Generated resume parser output for {base_name} at {output_file}")
```

## 17. Generate Unit Tests for Resume Parser Backend

ATS-ASAP-CODE-CUSTODIANS

```

src > backend > resume_parser > test_resume_parser > parser > test_extractor.py > DummyConfig
1 import unittest
2 from unittest.mock import patch, mock_open, MagicMock
3 from parser.extractor import extract_resume_json, extract_skill_entries, GrogAPIError, SkillEntry
4
5 class DummyConfig:
6 api_key = 'key'
7 model_name = 'model'
8 endpoint = 'http://fake-endpoint'
9 temperature = 0.0
10 max_tokens = 10
11
12 class TestExtractResumeJson(unittest.TestCase):
13 @patch('parser.extractor.open', new_callable=mock_open, read_data='prompt {{resume_text}}')
14 @patch('parser.extractor.requests.post')
15 def test_extract_resume_json_success(self, mock_post, mock_file):
16 mock_response = MagicMock()
17 mock_response.raise_for_status.return_value = None
18 mock_response.json().return_value = {
19 'choices': [
20 {'message': {'content': ('"foo": "bar")')}
21]
22 }
23 mock_post.return_value = mock_response
24 result = extract_resume_json('resume', 'prompt.txt', DummyConfig())
25 self.assertEqual(result, {'foo': 'bar'})
26
27 @patch('parser.extractor.open', side_effect=Exception('fail'))
28 def test_extract_resume_json_file_error(self, mock_file):
29 with self.assertRaises(GrogAPIError) as cm:
30 extract_resume_json('resume', 'prompt.txt', DummyConfig())
31 self.assertIn('Failed to read prompt', str(cm.exception))
32
33 @patch('parser.extractor.open', new_callable=mock_open, read_data='prompt {{resume_text}}')
34 @patch('parser.extractor.requests.post', side_effect=Exception('fail'))
35 def test_extract_resume_json_network_error(self, mock_post, mock_file):
36 with self.assertRaises(GrogAPIError) as cm:
37 extract_resume_json('resume', 'prompt.txt', DummyConfig())
38 self.assertIn('Failed to parse response from Grog', str(cm.exception))
39
40 @patch('parser.extractor.open', new_callable=mock_open, read_data='prompt {{resume_text}}')
41 @patch('parser.extractor.requests.post')
42 def test_ext(self, mock_res):
43 mock_res.seek(1) # Reject file
44 mock_res.read(1) # Accept file
45 mock_response.raise_for_status.return_value = None

```

Ln 10, Col 20 Spaces: 4 UTF-8 LF ¶ Python 3.12.9 ('venv': venv) Cursor Tab Continue

ATS-ASAP-CODE-CUSTODIANS

```

src > backend > resume_parser > test_resume_parser > parser > test_writer.py > ...
1 import unittest
2 from unittest.mock import patch, mock_open, MagicMock
3 from parser.writer import write_skill_entries_to_csv
4 from parser.extractor import SkillEntry
5
6 class TestWriteSkillEntriesToCsv(unittest.TestCase):
7 @patch('parser.writer.open', new_callable=mock_open)
8 @patch('parser.writer.csv.DictWriter')
9 def test_write_skill_entries_to_csv_success(self, mock_dict_writer, mock_file):
10 entry = SkillEntry(
11 candidate_name='John', email='john@example.com', summary='summary', skill='Python',
12 skill_count=1, skill_age=2, skill_experience_years=3.0, highest_degree='MSC',
13 graduation_year=2022, certifications=[], projects=[], languages={},
14 awards=[], interests=[], references=[]
15)
16 mock_writer = MagicMock()
17 mock_dict_writer.return_value = mock_writer
18 write_skill_entries_to_csv(entry, 'fake.csv')
19 mock_dict_writer.assert_called_once()
20 mock_writer.writerow.assert_called_once()
21 mock_writer.writerow.assert_called()
22
23 @patch('parser.writer.open', new_callable=mock_open)
24 @patch('parser.writer.csv.DictWriter')
25 def test_write_skill_entries_to_csv_empty(self, mock_dict_writer, mock_file):
26 with patch('builtins.print') as mock_print:
27 write_skill_entries_to_csv([], 'fake.csv')
28 mock_print.assert_called_with('[WARN] No skill entries to write.')
29 mock_dict_writer.assert_not_called()
30
31 if __name__ == '__main__':
32 unittest.main()

```

Ln 1, Col 1 Spaces: 4 UTF-8 LF ¶ Python 3.12.9 ('venv': venv) Cursor Tab Continue

Problems Output Debug Console Terminal Ports Comments

```
> python -m pytest src/backend/test_resume_parser/parser -v
=====
platform darwin -- Python 3.12.9, pytest-7.4.3, pluggy-1.5.0 -- /Users/arghyabanerjee/Desktop/ATS-ASAP-Code-Custodians/.venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/arghyabanerjee/Desktop/ATS-ASAP-Code-Custodians
plugins: anyio-4.9.0
collected 7 items

src/backend/test_resume_parser/parser/test_extractor.py::test_extract_resume_json_success PASSED [14%]
src/backend/test_resume_parser/parser/test_extractor.py::test_extract_resume_json_prompt_not_found PASSED [28%]
src/backend/test_resume_parser/parser/test_extractor.py::test_extract_skill_entries_minimal PASSED [42%]
src/backend/test_resume_parser/parser/test_reader.py::test_extract_text_success PASSED [57%]
src/backend/test_resume_parser/parser/test_reader.py::test_extract_text_invalid PASSED [71%]
src/backend/test_resume_parser/parser/test_writer.py::test_write_skill_entries PASSED [85%]
src/backend/test_resume_parser/parser/test_writer.py::test_write_empty_list PASSED [100%]

=====
7 passed in 0.13s =====
```

~/De/ATS-ASAP-Code-Custodians | on main >2 \*3 14 ?1 | ok | ATS-ASAP-Code-Custodians py | at 19:23:50

## 18. Self completion

The screenshot shows a code editor interface with a project navigation sidebar on the left and a main code editor area on the right. The code editor has tabs for api.py, main.py, config\_loader.py (which is currently active), and config.yaml. The config\_loader.py file contains Python code for loading configuration files. A cursor is visible in the code, and a tooltip or completion dropdown is shown above it, indicating the current word being completed. The completion dropdown lists several options starting with 'raise'. The project sidebar shows various files and directories, including test\_resume\_parser, components, frontend, utils, and testcases. The bottom status bar displays file paths, line numbers, and other system information.

```
def load_config(config_path: str = None) -> ResumeParserConfig: new =
 raw_yaml = read_file(config_path)
 expanded_yaml = os.path.expandvars(raw_yaml)
 config_dict = yaml.safe_load(expanded_yaml)

 if "data_folder" not in config_dict:
 raise ValueError("Missing 'data_folder' in the configuration file.")
 if "resume_extension" not in config_dict:
 raise ValueError("Missing 'resume_extension' in the configuration file.")
 if "llm" not in config_dict:
 raise ValueError("Missing 'llm' in the configuration file.")

 llm_cfg = config_dict['llm']

 if "provider" not in llm_cfg:
 raise ValueError("Missing 'provider' in the LLM configuration.")
 if "model_name" not in llm_cfg:

 return ResumeParserConfig(
 data_folder=config_dict['data_folder'],
 resume_extension=config_dict['resume_extension'],
 llm=LLMConfig(
 provider=llm_cfg['provider'],
 model_name=llm_cfg['model_name'],
 endpoint=llm_cfg['endpoint'],
 api_key=llm_cfg['api_key'],
 temperature=llm_cfg.get('temperature', 0.8),
 max_tokens=llm_cfg.get('max_tokens', 4096)
)
)
```

AP-Code-Custodians > src > backend > resume\_parser > config\_loader.py    30:5 ⚡ LF UTF-8 🗑 4 spaces Python 3.12 (ATS-ASAP-Code-Custodians) ⌂ ⓘ

# Staffing Assistant - Resume Ranking

## 1. Created a mock JD

You're an experienced Thoughtworks staffing consultant.

You are given a pool of candidates to staff for XYZ project with the following collective qualifications, skills, and experiences:

**Common Skills**: Python, Java, SQL, GCP, AWS, Azure, Terraform, Jenkins, Kubernetes, Machine Learning, Data Analysis, REST APIs, Cloud Architecture  
**Certifications**: Google Professional Cloud Architect, Oracle Certified Java Programmer, TensorFlow Developer Certificate  
**Education Levels**: Bachelor's and Master's in Computer Science, Information Systems, IT, Electronics  
**Experience Range**: 2 to 14 years  
**Job Roles**: Backend Engineer, Cloud Architect, DevOps Engineer, Data Analyst, ML Engineer, Application Developer  
**Tools & Frameworks**: Spring Boot, Hibernate, Airflow, MLflow, Power BI, Docker, Spark  
**Locations**: India-based talent pool (e.g., Bangalore, Mumbai, Hyderabad)  
**Work Style**: Most candidates have worked in agile teams and contributed to scalable enterprise projects

Based on this, generate a **realistic and professional job description** for a role that:

- Is suitable for a candidate with 4–8 years of experience
- Is remote-friendly or hybrid within India
- Requires strong backend or cloud engineering capabilities with optional ML/data skills
- Can be used to evaluate resumes using a weighted ranking system

Include:

- Company intro (1 paragraph)
- Role overview
- Responsibilities (5–6 bullets)
- Required skills (technical and soft)
- Nice-to-have skills or certifications
- Education requirements
- Experience range
- Employment type and location

Return the output in plain text. Avoid any JSON formatting or markdown.



### Company Introduction:

At Thoughtworks, we are a community of passionate technologists dedicated to driving innovation and solving complex challenges through human-centered software design and agile delivery. Our people-first culture, commitment to learning, and focus on social impact help us build sustainable and scalable solutions that create real business value. As a technology partner to leading enterprises globally, we believe in empowering our teams to shape the future of technology.

Role: Backend / Cloud Engineer (Mid-Level) 

## 2. Generate the Project structure

The screenshot shows a code editor interface with a sidebar and a main workspace.

**Project Structure:**

```
AT5-ASAP-CODE-CUSTODIANS
├── .venv
├── bin
├── compiler
├── configs
├── docker
├── documentation
├── envs
├── kubernetes
├── licenses
├── logs
├── models
├── notebooks
└── src
 ├── backend
 │ └── resume_parser
 │ ├── data
 │ ├── output
 │ ├── parser
 │ └── prompt
 │ ├── jd_prompt.txt
 │ ├── resume_prompt.txt
 │ └── ranking
 │ ├── __init__.py
 │ ├── jd_analyzer.py
 │ ├── ranker.py
 │ └── scorer.py
 └── main.py
```

**File Changes:**

- scorer.py (U)
- main.py (M)
- jd\_prompt.txt (U)
- \_\_init\_\_.py (U)
- ranker.py (U)
- scorer.py (U)
- \_\_init\_\_.py (U)
- config.yaml (M)
- main.py (M)
- test\_resume\_parser.py (U)
- \_\_init\_\_.py (U)
- .coverage (U)
- settings.py (U)
- web.py (U)

**File Details:**

scorer.py — ATS-ASAP-Code-Custodians

src > backend > resume\_parser > ranking > scorer.py

1 Reject 31N Accept 36Y

I have an existing Python project located at 'src/backend/resume\_parser' that currently:

- Parses resumes using an LLM
- Extracts structured JSON
- Writes enriched skill entries to CSV

Now I want to:

1. Add functionality to \*\*assign attribute weights\*\* based on a given Job Description using prompting.
2. Perform \*\*weighted ranking of parsed candidates\*\* based on those weights.
3. Export the \*\*final ranked list to a CSV\*\*.

I'll help you design and implement the extended folder structure for the resume parser with the new functionality. First, let me check the current structure.

Listed 8 items in src/backend/resume\_parser

Let me check the main parser module to understand the current implementation better.

Read main.py

Based on the current structure and your requirements, I'll propose an extended folder structure that maintains modularity while adding the new functionality. Here's the proposed structure:

```
src/backend/resume_parser/
├── __init__.py
├── config.yaml
└── main.py
```

Edited 6 files +183 -9 Reject all 31N Accept all 36Y

Add context

Plan, search, build anything

Agent Auto Continue

## 3. Prompt to Generate the JD Weighting Prompt

**CONTINUE**

You are a prompt engineering assistant.

Generate a reusable LLM prompt for a technical recruiter agent that does the following:

1. Accepts a \*\*Job Description\*\* as '{JOB\_DESCRIPTION}' and a list of \*\*keywords or attributes\*\* as '{KEYWORDS}'.
2. Instructs the LLM to:
  - Assign a \*\*weight between 0 and 1\*\* to each keyword based on its relevance to the job description.
  - Ensure the \*\*sum of all weights equals exactly 1.0\*\*.
3. Return the weights in \*\*valid JSON\*\* format (no extra text or explanation).
4. Provide an example JSON structure to guide the LLM.
5. Use clear section formatting like:

Here is a reusable LLM prompt for a technical recruiter agent:

### **Job Description and Keyword Weighting Prompt**

We will work with the following job description and keywords:

{JOB\_DESCRIPTION} - A detailed job description of the position we are recruiting for.

{KEYWORDS} - A list of relevant keywords or attributes that should be considered in the recruitment process.

**Task:** Assign a weight between 0 and 1 to each keyword based on its relevance to the job description. Ensure that the sum of all weights equals exactly 1.0. Return the weights in valid JSON format, without any extra text or explanation.

### **Example JSON Structure:**

```
Json [{ "keyword": "Machine Learning", "weight": 0.2 }
```

Ask a follow-up

Chat Llama 3.1 8B @ ⌂ @codebase Enter

#### 4. Generate the code

jd\_prompt.txt U jd\_analyzer.py U

```
src > backend > resume_parser > ranking > jd_analyzer.py > _clean_json_response
1 import json
2 import requests
3 from typing import List, Dict
4 from dataclasses import dataclass
5 import re
6
7 class JDAnalyzerError(Exception):
8 """Custom exception for JD analyzer errors."""
9 pass
10
11 @dataclass
12 class LLMConfig:
13 provider: str
14 model_name: str
15 endpoint: str
16 api_key: str
17 temperature: float = 0.0
18 max_tokens: int = 4096
19
20 def _clean_json_response(response_text: str) -> str:
21 """Clean the LLM response to extract valid JSON."""
22 # Remove markdown code blocks if present
23 json_match = re.search(r'```(?:(?<json>)?)\s*(?:(?<js>.*?)(?<js>))?\s*```', response_text)
24 if json_match:
25 response_text = json_match.group(1)
26
27 # Remove any leading/trailing whitespace
28 response_text = response_text.strip()
29
30 return response_text
31
32 def _parse_weights_response(response_text: str) -> Dict[str, float]:
33 """Parse the LLM response into a dictionary of weights."""
34 try:
35 # Clean the response
36 cleaned_text = _clean_json_response(response_text)
37
38 # Parse JSON
39 weights_list = json.loads(cleaned_text)
40
41 # Convert list of dicts to single dict
42 weights_dict = {i: v for i, v in enumerate(weights_list)}
43
44 # Validate weights
45 except Exception as e:
46 raise JDAnalyzerError(f"Error parsing weights: {e}")
47
48 return weights_dict
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
267
268
269
269
270
271
272
273
274
275
276
277
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
316
317
317
318
319
319
320
321
322
323
324
325
326
327
327
328
329
329
330
331
332
333
334
335
336
337
337
338
339
339
340
341
342
343
344
345
345
346
347
347
348
349
349
350
351
352
353
354
355
356
357
357
358
359
359
360
361
362
363
364
365
366
367
367
368
369
369
370
371
372
373
374
375
376
377
377
378
379
379
380
381
382
383
384
385
386
387
387
388
389
389
390
391
392
393
394
395
396
397
397
398
399
399
400
401
402
403
404
405
406
407
407
408
409
409
410
411
412
413
414
415
416
416
417
418
418
419
419
420
421
422
423
424
425
426
426
427
428
428
429
429
430
431
432
433
434
435
436
437
437
438
439
439
440
441
442
443
444
445
446
447
447
448
449
449
450
451
452
453
454
455
456
457
457
458
459
459
460
461
462
463
464
465
466
467
467
468
469
469
470
471
472
473
474
475
476
477
477
478
479
479
480
481
482
483
484
485
486
487
487
488
489
489
490
491
492
493
494
495
496
497
497
498
499
499
500
501
502
503
504
505
506
507
507
508
509
509
510
511
512
513
514
515
516
517
517
518
519
519
520
521
522
523
524
525
526
527
527
528
529
529
530
531
532
533
534
535
536
537
537
538
539
539
540
541
542
543
544
545
546
547
547
548
549
549
550
551
552
553
554
555
556
557
557
558
559
559
560
561
562
563
564
565
566
567
567
568
569
569
570
571
572
573
574
575
576
577
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
607
608
609
609
610
611
612
613
614
615
616
616
617
618
618
619
619
620
621
622
623
624
625
626
626
627
628
628
629
629
630
631
632
633
634
635
636
637
637
638
639
639
640
641
642
643
644
645
646
646
647
648
648
649
649
650
651
652
653
654
655
656
657
657
658
659
659
660
661
662
663
664
665
666
666
667
668
668
669
669
670
671
672
673
674
675
676
676
677
678
678
679
679
680
681
682
683
684
685
686
686
687
688
688
689
689
690
691
692
693
694
695
696
696
697
698
698
699
699
700
701
702
703
704
705
706
706
707
708
708
709
709
710
711
712
713
714
715
715
716
717
717
718
718
719
719
720
721
722
723
724
725
726
726
727
728
728
729
729
730
731
732
733
734
735
736
736
737
738
738
739
739
740
741
742
743
744
745
745
746
747
747
748
748
749
749
750
751
752
753
754
755
756
756
757
758
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
776
777
778
778
779
779
780
781
782
783
784
785
786
786
787
788
788
789
789
790
791
792
793
794
795
796
796
797
798
798
799
799
800
801
802
803
804
805
806
806
807
808
808
809
809
810
811
812
813
814
815
815
816
817
817
818
818
819
819
820
821
822
823
824
825
825
826
827
827
828
828
829
829
830
831
832
833
834
835
835
836
837
837
838
838
839
839
840
841
842
843
844
845
845
846
847
847
848
848
849
849
850
851
852
853
854
855
855
856
857
857
858
858
859
859
860
861
862
863
864
865
865
866
867
867
868
868
869
869
870
871
872
873
874
875
875
876
877
877
878
878
879
879
880
881
882
883
884
885
885
886
887
887
888
888
889
889
890
891
892
893
894
894
895
896
896
897
897
898
898
899
899
900
901
902
903
904
904
905
906
906
907
907
908
908
909
909
910
911
912
913
914
914
915
916
916
917
917
918
918
919
919
920
921
922
923
924
925
925
926
927
927
928
928
929
929
930
931
932
933
934
935
935
936
937
937
938
938
939
939
940
941
942
943
944
944
945
946
946
947
947
948
948
949
949
950
951
952
953
954
954
955
956
956
957
957
958
958
959
959
960
961
962
963
964
964
965
966
966
967
967
968
968
969
969
970
971
972
973
974
974
975
976
976
977
977
978
978
979
979
980
981
982
983
984
984
985
986
986
987
987
988
988
989
989
990
991
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1011
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1021
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1031
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1041
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1751
1752
1752
1753
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1761
1762
1762
1763
1763
1764
1764
1765
176
```

The screenshot shows a code editor interface with two panes. The left pane displays the `main.py` file, which contains Python code for resume parsing. The right pane is a chat window titled "Enhancing Resume Parser Functionality" where a developer is interacting with an AI assistant.

**Code Editor (Left):**

```

src > backend > resume_parser > main.py > ResumeParserConfig
29 def load_config(config_path: str) -> ResumeParserConfig:
30 max_tokens=llm_config.get('max_tokens', 4096)
31
32)
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94

```

**Chat Window (Right):**

Developer: Enhancing Resume Parser Functionality

AI Assistant:

- Ask for context: @ Add context
- Suggest code: 

```
def generate_weights_from_id(jd_id: str, keywords: List[str], prompt_path: str, llm_config: LLMConfig) -> Dict[str, float]
```

In my main.py, after parsing all resumes and writing parsed\_resume\_skills.csv, I want to:
- Provide instructions: Load a job\_description.txt from the data/ folder
- Offer help: Define a keyword list by reading the columns from parsed\_resume\_skills.csv (except name/email)
- Provide instructions: Call generate\_weights\_from\_id(...) with the JD, keywords, and prompt (prompt/jd\_prompt.txt)
- Offer help: I'll help you integrate the JD analysis into main.py. First, let me check the structure of the parsed skills CSV to understand the columns.
- Provide instructions: @ Read writer.py
- Offer help: Now I'll update the main.py to include the JD analysis step:

AI Assistant's code suggestion:

```

main.py +60 -4
import os
import yaml
import json
import csv
from dataclasses import dataclass
from parser.reader import extract_text_from_docx
from parser.extractor import extract_resume_json, extract_skills
from parser.writer import write_skill_entries_to_csv
> Edited 6 files +206 -8

```

Accept %

Plan, search, build anything

Agent AI Auto

scorer.py M

```

src > backend > resume_parser > ranking > scorer.py > build_candidate_profiles
1 import json
2 import pandas as pd
3 from typing import List, Dict
4 import logging
5
6 logging.basicConfig(level=logging.INFO)
7 logger = logging.getLogger(__name__)
8
9 def build_candidate_profiles(csv_path: str, jd_weights_path: str) -> List[Dict]:
10 """
11 Build candidate profiles from parsed resume data and JD weights.
12
13 Args:
14 csv_path: Path to the parsed resume skills CSV file
15 jd_weights_path: Path to the JD weights JSON file
16
17 Returns:
18 List of candidate profile dictionaries with skill scores
19 """
20
21 # Load JD weights
22 logger.info("Loading JD weights...")
23 with open(jd_weights_path, 'r') as f:
24 jd_weights = json.load(f)
25 logger.info(f"Loaded JD weights with {len(jd_weights)} skills")
26
27 # Load and group CSV data by candidate
28 logger.info("Loading and processing resume data...")
29 df = pd.read_csv(csv_path)
30 candidate_groups = df.groupby('candidate_name')
31
32 candidate_profiles = []
33
34 # Process each candidate
35 for candidate_name, group in candidate_groups:
36 logger.info(f"Processing candidate: {candidate_name}")
37
38 # Initialize candidate profile
39 profile = {
40 "name": candidate_name,
41 "summary": group['summary'].iloc[0],
42 "languages": group['languages'].iloc[0],
43 "certifications": group['certifications'].iloc[0]
44 }
45
46 # Calculate scores for each JD skill
47 for skill, weight in jd_weights.items():
48 # Check if skill exists in candidate's data
49 skill_data = group[group['skill'].str.lower() == skill.lower()]
50
51 if not skill_data.empty:
52 # Get skill metrics
53 experience_years = float(skill_data['experience'].iloc[0])
54 count = float(skill_data['count'].iloc[0])
55 age = float(skill_data['skill_age'].iloc[0])
56
57 # Calculate skill score
58 skill_score = (experience_years * weight) / count
59
60 # Add skill score to profile
61 profile[skill] = skill_score
62
63 # Return candidate profiles
64 return candidate_profiles

```

Python Script for Candidate Scoring

I have the following files in my project:

- CSV File: 'src/backend/resume\_parser/output/parsed\_resume\_skills.csv'
- JD Weights JSON: 'src/backend/resume\_parser/output/jd\_weights.json'

Each row in the CSV contains:

- candidate\_name
- email
- summary
- skill
- skill\_count
- skill\_age

Restore checkpoint

I'll help you create the `scorer.py` file with the specified functionality. First, let me check if the directory exists and create the file.

Listed 5 items in `src/backend/resume_parser/ranking`

I see the directory exists and there's already a `scorer.py` file. Let me create the implementation with the required functionality.

scorer.py +77 -1

```

import json
import pandas as pd
from typing import List, Dict
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def build_candidate_profiles(csv_path: str, jd_weights_path: str) -> List[Dict]:
 """
 Build candidate profiles from parsed resume data.
 """

```

I've created the `scorer.py` file with the `build_candidate_profiles` function that implements all the requested functionality. Here's what the implementation does:

- Imports necessary libraries: `json`, `pandas`, `typing`, and `logging`
- Sets up logging to show INFO level messages
- Implements the `build_candidate_profiles` function that:

Edited 1 file +77 -1 Reject Auto Accept Add context Plan, search, build anything Agent Auto Continue

```

scorer.py M main.py M X
src > backend > resume_parser > main.py > ...
121 parsed_json_dir = os.path.join(base_dir, 'output', 'parsed_resumes')
122 csv_output_path = os.path.join(base_dir, 'output', 'parsed_resume_skills.csv')
123
124 _setup_output_directories(base_dir, parsed_json_dir, csv_output_path)
125
126 prompt_path = os.path.join(base_dir, 'prompt', 'resume_prompt.txt')
127
128 # Process all resumes and collect skill entries
129 all_skill_entries = _process_resumes(resumes_dir, parsed_json_dir, prompt_path, config.llm, config.r)
130
131 # Write CSV of skills
132 write_skill_entries_to_csv(all_skill_entries, csv_output_path)
133 print(f"[SUCCESS] Wrote enriched CSV to {csv_output_path}")
134
135 # Analyze job description and generate weights
136 _analyze_job_description(base_dir, config, csv_output_path)
137
138 # Build candidate profiles
139 from_path = os.path.join(base_dir, 'output', 'parsed_resume_skills.csv')
140 weights_path = os.path.join(base_dir, 'output', 'jd_weights.json')
141 candidate_profiles = build_candidate_profiles(from_path, weights_path)
142 print(f"[SUCCESS] Built {len(candidate_profiles)} candidate profiles for ranking")
143
144 # Save candidate profiles to JSON for verification
145 profiles_path = os.path.join(base_dir, 'output', 'candidate_profiles.json')
146 with open(profiles_path, 'w', encoding='utf-8') as f:
147 json.dump(candidate_profiles, f, indent=2, ensure_ascii=False)
148 print(f"[SUCCESS] Saved candidate profiles to {profiles_path}")
149
150
151 if __name__ == '__main__':
152 main()

```

Reject ⌘N Accept ⌘Y

Python Script for Candidate Scoring

main.py | scorer.py

Update main.py to call the build\_candidate\_profiles function from scorer.py after writing the enriched resume CSV. The steps should be:

Import the function build\_candidate\_profiles from ranking.scorer.

After the [SUCCESS] Wrote enriched CSV to parsed\_resume\_skills.csv log, invoke:

python

Copy

Edit

```
from_path = os.path.join(base_dir, 'output', 'parsed_resume_skills.csv')
weights_path = os.path.join(base_dir, 'output', 'jd_weights.json')
candidate_profiles = build_candidate_profiles(from_path, weights_path)
```

Restore checkpoint

I'll help you update main.py to include the candidate profile building functionality. Let me make the changes:

```

from src.backend.resume_parser.parser.reader import extract_text_from_docx
from src.backend.resume_parser.parser.writer import write_skill_entries_to_csv
from src.backend.resume_parser.ranking.jd_analyzer import generate_weights_from
from src.backend.resume_parser.ranking.scorer import build_candidate_profiles

def load_config(config_path: str) -> ResumeParserConfig:
 # Analyze job description and generate weights
 _analyze_job_description(base_dir, config, csv_output_path)

 # Build candidate profiles
 from_path = os.path.join(base_dir, 'output', 'parsed_resume_skills.csv')

```

I've updated main.py with the following changes:

- Added the import statement for `build_candidate_profiles` from the scorer module
- Added the candidate profile building step after the job description analysis
- Added code to save the candidate profiles to a JSON file for verification

The changes include:

- Using the correct paths for the CSV and weights files
- Building the candidate profiles using the imported function
- Ensuring the number of profile built

Edited 2 files +90 -1

Plan, search, build anything

Agent Auto

Ln 163, Col 1 Spaces: 4 UTF-8 LF ↓ Python 3.12.9 ('.venv': venv) Cursor Tab ✎ Continue

## 5. Fixing Error:



File "/Users/arghyabenerjee/Desktop/ATS-ASAP-Code-Custodians/src/backend/resume\_parser/ranking/jd\_analyzer.py", line 129, in generate\_weights\_from\_jd  
 jd\_analyzer.py", line 59, in \_parse\_weights\_response  
 raise JDAnalyzerError("Missing 'keyword' field in item {i}: keys={list(obj.keys())}")

I see the issue now. The LLM is returning a dictionary directly instead of a list of objects with 'keyword' and 'weight' fields as expected by our parser. Let me modify the `_parse_weights_response` function to handle both formats:

```
Handle both dictionary and list-of-objects formats
if isinstance(data, dict):
 # Direct dictionary format: {"python": 0.25, "aws": 0.20, ...}
 weights = {}
 for key, value in data.items():
 if isinstance(value, (int, float)):
 weights[str(key).strip()] = float(value)
 elif isinstance(value, str):
 try:
 weights[str(key).strip()] = float(value.strip())
 except ValueError:
 raise JDAnalyzerError(f"Unexpected JSON structure: {type(data).__name__}")
 else:
 raise JDAnalyzerError(f"Item {key} is not an object: {repr(value)}")
else:
 # support both list-of-objects and single-object formats
 items = data if isinstance(data, list) else [data] if isinstance(data, dict) else None
 if items is None:
 raise JDAnalyzerError(f"Unexpected JSON structure: {type(data).__name__}")

 weights: Dict[str, float] = {}
 for i, obj in enumerate(items):
 if not isinstance(obj, dict):
 raise JDAnalyzerError(f"Item {i} is not an object: {repr(obj)}")

 # Handle both dictionary and list-of-objects formats
 if isinstance(data, dict):
 # Direct dictionary format: {"python": 0.25, "aws": 0.20, ...}
 weights = {}
 for key, value in data.items():
 if isinstance(value, (int, float)):
 weights[str(key).strip()] = float(value)
 elif isinstance(value, str):
 try:
 weights[str(key).strip()] = float(value.strip())
 except ValueError:
 raise JDAnalyzerError(f"Unexpected JSON structure: {type(data).__name__}")
 else:
 raise JDAnalyzerError(f"Item {key} is not an object: {repr(value)}")
 else:
 # support both list-of-objects and single-object formats
 items = data if isinstance(data, list) else [data] if isinstance(data, dict) else None
 if items is None:
 raise JDAnalyzerError(f"Unexpected JSON structure: {type(data).__name__}")

 weights: Dict[str, float] = {}
 for i, obj in enumerate(items):
 if not isinstance(obj, dict):
 raise JDAnalyzerError(f"Item {i} is not an object: {repr(obj)}")

 # Support both direct dictionary and list-of-objects formats
 if isinstance(data, dict):
 # Direct dictionary format: {"python": 0.25, "aws": 0.20, ...}
 weights = {}
 for key, value in data.items():
 if isinstance(value, (int, float)):
 weights[str(key).strip()] = float(value)
 elif isinstance(value, str):
 try:
 weights[str(key).strip()] = float(value.strip())
 except ValueError:
 raise JDAnalyzerError(f"Unexpected JSON structure: {type(data).__name__}")
 else:
 raise JDAnalyzerError(f"Item {key} is not an object: {repr(value)}")
 else:
 # support both list-of-objects and single-object formats
 items = data if isinstance(data, list) else [data] if isinstance(data, dict) else None
 if items is None:
 raise JDAnalyzerError(f"Unexpected JSON structure: {type(data).__name__}")

 weights: Dict[str, float] = {}
 for i, obj in enumerate(items):
 if not isinstance(obj, dict):
 raise JDAnalyzerError(f"Item {i} is not an object: {repr(obj)}")
```

I've updated the `_parse_weights_response` function to handle both formats that the LLM might return:

1. Direct dictionary format: {"python": 0.25, "aws": 0.20, ...}
2. List of objects format: [{"keyword": "python", "weight": 0.25}, ...]

The function now:

1. First checks if the response is a dictionary and handles it directly
2. If it's a list, it checks each item to see if it's a dictionary and handles it directly
3. If it's neither, it raises an error

The screenshot shows a GitHub pull request interface. On the left, the code for `ranker.py` is displayed, which reads candidate profiles and JD weights from JSON files and outputs a ranked CSV. The right side shows the commit history and a summary of changes.

**Python Candidate Ranking Script**

Listed 11 items in `src/backend/resume_parser`

Great, I see the directory structure exists. Now I'll create the ranker script with the specified functionality.

```
ranker.py +31 -1
import json
import csv
from pathlib import Path

def rank_candidates():
 # Load candidate profiles
 with open('src/backend/resume_parser/output/candidate_profiles.json', 'r') as f:
 profiles = json.load(f)

 # Load JD weights
 with open('src/backend/resume_parser/output/jd_weights.json', 'r') as f:
 jd_weights = json.load(f)

 # Calculate scores for each candidate
 ranked_candidates = []
 for name, profile in profiles.items():
 score = sum(profile[skill] * weight for skill, weight in jd_weights.items())
 ranked_candidates.append((name, score))

 # Sort by descending score
 ranked_candidates.sort(key=lambda x: x[1], reverse=True)

 # Write to CSV
 output_path = Path('src/backend/resume_parser/output/final_ranking.csv')
 with open(output_path, 'w', newline='') as f:
 writer = csv.writer(f)
 writer.writerow(['candidate_name', 'score'])
 writer.writerows(ranked_candidates)

if __name__ == '__main__':
 rank_candidates()
```

I've created the ranker script that implements all the requested functionality:

1. It loads the candidate profiles and JD weights from their respective JSON files
2. For each candidate, it calculates their score by multiplying their skill values with the corresponding JD weights and summing them up
3. The candidates are sorted by their scores in descending order
4. The results are written to a CSV file with the specified format

The script uses only the standard library (`json`, `csv`, `pathlib`) and can be run directly to generate the rankings. The output CSV will be created at the specified path with the exact format requested.

Review changes

Edited 1 file +31 -1

Add context

Plan, search, build anything

Agent Auto

Ln 32, Col 1 Spaces: 4 UTF-8 LF Python 3.12.9 ('.venv': venv) Cursor Tab Continue

## 6. Refactor:

src > backend > resume\_parser > main.py > load\_config

```
def load_config(config_path: str) -> ResumeParserConfig:
 data_folder=config_dict['data_folder'],
 resume_extension=config_dict['resume_extension'],
 ltm=LMConfig(
 provider=llm_cfg['provider'],
 model_name=llm_cfg['model_name'],
 endpoint=llm_cfg['endpoint'],
 api_key=llm_cfg['api_key'] or os.getenv("GROQ_API_KEY"),
 temperature=llm_cfg.get('temperature', 0.0),
 max_tokens=llm_cfg.get('max_tokens', 4096)
)
 return config

def main():
 base_dir = os.path.dirname(__file__)
 config = load_config(os.path.join(base_dir, 'config.yaml'))

 # Resumes directory
 resumes_dir = os.path.join(base_dir, config.data_folder, 'resumes')
 if not os.path.isdir(resumes_dir):
 raise FileNotFoundError(f"Resumes folder not found: {resumes_dir}")

 # Output paths
 parsed_json_dir = os.path.join(base_dir, 'output', 'parsed_resumes')
 csv_output_path = os.path.join(base_dir, 'output', 'parsed_resume_skills.csv')

 def _setup_output_directories(base_dir: str, parsed_json_dir: str, csv_output_path: str):
 """Create necessary output directories if they don't exist."""
 os.makedirs(parsed_json_dir, exist_ok=True)
 os.makedirs(os.path.dirname(csv_output_path), exist_ok=True)

 prompt_path = os.path.join(base_dir, 'prompt', 'resume_prompt.txt')
 all_skill_entries = []

 # Parse each resume
 for filename in os.listdir(resumes_dir):
 if not filename.lower().endswith(config.resume_extension.lower()):
 continue

 file_path = os.path.join(resumes_dir, filename)
 try:
 resume_text = open(file_path).read()
 print(f'[INFO] {filename} - {len(resume_text)} characters - {file_path}')
 print(f'[INFO] {1 / len(resume_text)} Reject Accept me')
 except Exception as e:
```

New chat

Refactor the `main()` function by splitting it into smaller, well-named helper functions that each handle a single responsibility. Follow clean code principles like:

- One function = one responsibility
- Group steps like resume parsing, JSON saving, CSV generation, and JD weight analysis
- Ensure clarity, reuse, and testability
- Preserve logging (`print(...)`) for progress tracking

I'll help you refactor the `main()` function by breaking it down into smaller, focused helper functions while maintaining the same functionality and logging. First, let me analyze the current code and then make the changes.

The main function currently has these major responsibilities:

1. Setup and configuration
2. Resume parsing and JSON extraction
3. CSV generation from skill entries
4. JD analysis and weight generation

I'll create helper functions for each of these responsibilities. Let me make these changes:

```
import csv
import json
import os
```

Plan, search, build anything

## 7. Generating tests:

ATS-ASAP-CODE-CUSTODIANS

```

1 import pytest
2 import json
3 import pandas as pd
4 from src.backend.resume_parser.ranking.scorer import build_candidate_profiles
5
6 def test_build_candidate_profiles(tmp_path):
7 # Create test CSV file
8 csv_data = pd.DataFrame([
9 {
10 'candidate_name': 'Alice',
11 'skill': 'python',
12 'skill_count': 2,
13 'skill_age': 0,
14 'skill_experience_years': 1.5,
15 'summary': '',
16 'languages': 'en',
17 'certifications': ''
18 },
19 {
20 'candidate_name': 'Alice',
21 'skill': 'aws',
22 'skill_count': 1,
23 'skill_age': 12,
24 'skill_experience_years': 0.5,
25 'summary': '',
26 'languages': '',
27 'certifications': ''
28 },
29 {
30 'candidate_name': 'Bob',
31 'skill': 'python',
32 'skill_count': 1,
33 'skill_age': 24,
34 'skill_experience_years': 1.0,
35 'summary': '',
36 'languages': '',
37 'certifications': ''
38 },
39 {
40 'candidate_name': 'Bob',
41 'skill': 'docker',
42 'skill_count':
43 'skill_age': 6 ^ 1/1 ^ Reject Accept
44 'skill_experience_years': 0.2,

```

New chat

Let me create each test file one by one. I'll start with `test_score.py`

```

1 import pytest
2 import json
3 import pandas as pd
4 from src.backend.resume_parser.ranking.scorer import build_candidate_profiles
5
6 def test_build_candidate_profiles(tmp_path):
7 # Create test CSV file
8 csv_data = pd.DataFrame([
9 {
10 'candidate_name': 'Alice',
11 'skill': 'python',
12 'skill_count': 2,
13 'skill_age': 0,
14 'skill_experience_years': 1.5,
15 'summary': '',
16 'languages': 'en',
17 'certifications': ''
18 },
19 {
20 'candidate_name': 'Alice',
21 'skill': 'aws',
22 'skill_count': 1,
23 'skill_age': 12,
24 'skill_experience_years': 0.5,
25 'summary': '',
26 'languages': '',
27 'certifications': ''
28 },
29 {
30 'candidate_name': 'Bob',
31 'skill': 'python',
32 'skill_count': 1,
33 'skill_age': 24,
34 'skill_experience_years': 1.0,
35 'summary': '',
36 'languages': '',
37 'certifications': ''
38 },
39 {
40 'candidate_name': 'Bob',
41 'skill': 'docker',
42 'skill_count':
43 'skill_age': 6 ^ 1/1 ^ Reject Accept
44 'skill_experience_years': 0.2,

```

Now let me create `test_ranker.py`:

```

1 test_ranker.py
2 profiles_path = tmp_path / "candidate_profiles.json"
3 with open(profiles_path, 'w') as f:
4 json.dump(profiles, f)
5
6 # Create JD weights
7 weights = {"python": 0.6, "aws": 0.4}
8 weights_path = tmp_path / "jd_weights.json"
9 with open(weights_path, 'w') as f:
10 json.dump(weights, f)
11
12 # Create output directory
13 out

```

Generating Stop Accept

Add context Plan, search, build anything

Agent Auto

ATS-ASAP-CODE-CUSTODIANS

```

1 import pytest
2 import json
3 import csv
4 import os
5 from src.backend.resume_parser.ranking.ranker import rank_candidates
6
7 def test_rank_candidates(tmp_path, monkeypatch):
8 # Create test candidate profiles
9 profiles = [
10 {"name": "A", "python": 1.0, "aws": 0.0},
11 {"name": "B", "python": 0.5, "aws": 0.5}
12]
13
14 profiles_path = tmp_path / "candidate_profiles.json"
15 with open(profiles_path, 'w') as f:
16 json.dump(profiles, f)
17
18 # Create JD weights
19 weights = {"python": 0.6, "aws": 0.4}
20 weights_path = tmp_path / "jd_weights.json"
21 with open(weights_path, 'w') as f:
22 json.dump(weights, f)
23
24 # Create output directory
25 output_dir = (function) mkdir: Any
26 output_dir.mkdir()
27
28 # Monkeypatch the file paths
29 def mock_path_join(*args):
30 if args[-1] == 'candidate_profiles.json':
31 return str(profiles_path)
32 elif args[-1] == 'jd_weights.json':
33 return str(weights_path)
34 elif args[-1] == 'final_ranking.csv':
35 return str(output_dir / 'final_ranking.csv')
36 return os.path.join(*args)
37
38 monkeypatch.setattr(os.path, 'join', mock_path_join)
39 monkeypatch setattr(os.path, 'dirname', lambda x: str(tmp_path))
40
41 # Run ranking
42 rank_candidates()
43
44 # Read and verify
45 ranking_path = output_dir / 'final_ranking.csv'

```

New chat

Now let me create `test_ranker.py`:

```

1 test_ranker.py (new) +61 -1
import pytest
import json
import csv
import os
from src.backend.resume_parser.ranking.ranker import rank_candidates
def test_rank_candidates(tmp_path, monkeypatch):
 # Create test candidate profiles
 profiles = [
 {"name": "A", "python": 1.0, "aws": 0.0},
 {"name": "B", "python": 0.5, "aws": 0.5}
]

```

Finally, let me create `test_id_analyzer.py`:

```

1 test_id_analyzer.py
2 generate_weights_from_jd,
3 JDAnalyzerError
4
5 def test_clean_json_response():
6 # Test with code fence
7 input_text = '''`json\n"key": "value"\n```'''
8 assert _clean_json_response(input_text) == '{"key": "value"}'
9
10 # Test with quotes
11 input_text = """{"key": "value"}"""
12 assert _clean_json_response(input_text) == '{"key": "value"}'

```

Generating Stop Accept all

Add context Plan, search, build anything

Agent Auto

The screenshot shows a code editor interface with several tabs open. On the left, a sidebar displays the project structure of 'ATS-ASAP-CODE-CUSTODIANS' with various sub-directories like 'src', 'backend', and 'test\_resume\_parser'. The main area contains code snippets from several test files:

- `test_id_analyzer.py`: Contains tests for `_clean_json_response`, `_parse_weights_response_list_format`, and `_parse_weights_response_dict_format`.
- `test_ranker.py`: Contains a test for `test_rank_candidates`.
- `test_scorer.py`: Contains a test for `test_clean_json_response`.
- `test_ranker.py`: Contains a test for `test_clean_json_response`.
- `test_id_analyzer.py`: Contains a test for `test_parse_weight`.

A terminal window at the bottom shows the output of running the tests with `python -m pytest src/backend/test_resume_parser/ranking`. The output indicates 10 items collected and 10 passed tests in 0.34 seconds.

```

Problems Output Debug Console Terminal Ports Comments
> python -m pytest src/backend/test_resume_parser/ranking
=====
platform darwin -- Python 3.12.9, pytest-7.4.3, pluggy-1.5.0 -- /Users/arghyabanerjee/Desktop/ATS-ASAP-Code-Custodians/.venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/arghyabanerjee/Desktop/ATS-ASAP-Code-Custodians
plugins: anyio-4.9.0
collected 10 items

src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_clean_json_response PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_parse_weights_response_list_format PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_parse_weights_response_dict_format PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_parse_weights_response_invalid_json PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_parse_weights_response_invalid_structure PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_parse_weights_response_invalid_weight PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_generate_weights_from_id PASSED
src/backend/test_resume_parser/ranking/test_id_analyzer.py::test_generate_weights_from_id_api_error PASSED
src/backend/test_resume_parser/ranking/test_ranker.py::test_rank_candidates_writes_csv PASSED
src/backend/test_resume_parser/ranking/test_scorer.py::test_build_candidate_profiles PASSED
=====
10 passed in 0.34s

```

The terminal also shows the status bar indicating the file is `ok | ATS-ASAP-Code-Custodians.py` at 19:23:20.

# Resume Update Notifier

## 1. Create a service to send email

```
1 import os
2 import smtplib
3 from email.mime.text import MIMEText
4 from email.mime.multipart import MIMEMultipart
5 from typing import Optional
6
7 def send_email(
8 receiver_email: str,
9 subject: str,
10 body: str
11) -> bool:
12 try:
13 sender_email = os.getenv('SENDER_EMAIL')
14 sender_password = os.getenv('SENDER_PASSWORD')
15 smtp_server = os.getenv('SMTP_SERVER')
16 smtp_port = os.getenv('SMTP_PORT')
17
18 # Create a multipart message
19 message = MIMEMultipart()
20 message['From'] = sender_email
21 message['To'] = receiver_email
22 message['Subject'] = subject
23
24 # Add body to email
25 message.attach(MIMEText(body, "plain"))
26
27 # Create SMTP session
28 with smtplib.SMTP(smtp_server, smtp_port) as server:
29 # Start TLS for security
30 server.starttls()
31
32 # Login to the server
33 if sender_email and sender_password:
34 server.login(sender_email, sender_password)
35
36 # Send email
37 server.send_message(message)
38
39 return True
40
41 except Exception as e:
42 print(f"An error occurred while sending the email: {str(e)}")
43
44 return False
```

## 2. Create a fastAPI endpoint to take file as an input

### 3. Write the mail body

create an email body to ask the employee to update/upload their latest resume in a specified portal like jigsaw as they have not updated it since the past 6 months

the body will be stored in an .env file in a python application with the provision to accept a date field

Here's a sample email body template written in a format suitable for storing inside a `.env` file as a multi-line string (you can escape newlines or use triple quotes based on how you load it in your Python app). It includes a placeholder for the dynamic date field:

#### 4. Fixing bugs

Fixing EMAIL\_TEMPLATE Rendering Issue

email\_controller.py

```
16 async def process_emails(file: UploadFile = File(...)):
17 # Read the Excel file
18 df = pd.read_excel(file.file)
19
20 # Validate required columns
21 required_columns = ['email', 'date']
22 if not all([col in df.columns for col in required_columns]):
23 raise HTTPException(
24 status_code=400,
25 detail="Excel file must contain columns: " + ", ".join(required_columns))
26
27 # Convert date column to datetime
28 df['date'] = pd.to_datetime(df['date'])
29
30 # Calculate date 6 months ago
31 six_months_ago = datetime.now() - timedelta(days=180)
32
33 # Filter rows where date is 6 months or more prior
34 old_records = df[df['date'] <= six_months_ago]
35
36 # Get email template from environment variable
37 email_template = os.getenv('EMAIL_TEMPLATE', '')
38 if not email_template:
39 raise HTTPException(
40 status_code=500,
41 detail="Email template not found in environment variables")
42
43 # Process each record and send emails
44 results = []
45 for _, row in old_records.iterrows():
46 # Format the email body with the date
47 email_body = email_template.format(lastUploadDate=row['date'],
48 email=row['email'])
49
50 # Send the email
51 results.append(await send_email(email_body))
52
53 return {"status": "Success", "details": "Emails sent successfully."}
```

My EMAIL\_TEMPLATE is not getting correctly rendered from the .env file that I have created, so when I hit the endpoint it is giving me detail: "500: Email template not found in environment variables". Can you fix it?

I'll help you fix the issue with the EMAIL\_TEMPLATE not being loaded from the .env file. Let me first check if the .env file exists and its contents.

Searched files ".env"

Let me check the root directory of your project to see if the .env file exists there.

Listed 2 items in current directory

Let me check inside the ATS-ASAP-Code-Custodians directory:

Listed 22 items in ATS-ASAP-Code-Custodians

I see that there's an .envs directory. Let me check its contents:

Listed 3 items in ATS-ASAP-Code-Custodians/.envs

I see the issue now. The .env file is not present in your project, which is why the EMAIL\_TEMPLATE environment variable is not being loaded. Let's create a .env file in the root of your project with the necessary configuration.

I'll create a .env file with a sample email template:

Cannot edit files blocked by globalignore

I apologize for the error. It seems there might be some restrictions on creating files directly. Let me suggest what you need to do to fix this issue:

- Create a new file named .env in the root directory of your project (ATS-ASAP-Code-Custodians/.env) with the following content:

# Email Template Configuration  
EMAIL\_TEMPLATE='Dear User,\n\nThis is a reminder that your record from {date} is over 6 months old. Please take necessary action.\n\nBest regards,  
The System'
- Make sure the .env file is in the correct location. Based on your code structure, it should be in the same directory as your src folder.

## 5. Write tests for service and controller

**Creating Tests for Email Service**

```

email_service.py test_email_service.py
from unittest.mock import patch, MagicMock
import os
from src.backend.email_sender.service.email_service import send_email

@ pytest.fixture
def mock_env_vars():
 return {
 'SENDER_EMAIL': 'test@example.com',
 'SENDER_PASSWORD': 'test_password',
 'SMTP_SERVER': 'smtp.test.com',
 'SMTP_PORT': '587'
 }

def test_send_email_success(mock_env_vars):
 with patch.dict(os.environ, mock_env_vars):
 with patch('smtplib.SMTP') as mock_smtp:
 # Setup mock SMTP server
 mock_server = MagicMock()
 mock_smtp.return_value.__enter__.return_value = mock_server

 # Test the function
 result = send_email(
 receiver_email='recipient@example.com',
 subject='Test Subject',
 body='Test Body'
)

 # Assertions
 assert result is True
 mock_smtp.assert_called_once_with('smtp.test.com', '587')
 mock_server.starttls.assert_called_once()
 mock_server.login.assert_called_once_with('test@example.com', 'test_password')
 mock_server.send_message.assert_called_once()

```

Implicitly in order to avoid unexpected behavior in the future. Valid fixture loop scopes are: "function", "class", "module", "package", "session"

warnings.warn(PytestDeprecationWarning(\_DEFAULT\_FIXTURE\_LOOP\_SCOPE\_UNSET))

platform darwin -- Python 3.12.2, pytest-6.3.5, pluggy-1.5.0 -- /Users/adrijachakraborty/Desktop/code-custodians/venv/bin/python

cachedir: .pytest\_cache

rootdir: /Users/adrijachakraborty/Desktop/code-custodians/ATS-ASAP-Code-Custodians/src

plugins: anyio-4.9.0, aiofiles-0.4.0, aiohttp-3.8.1, asynctest-0.20.0, asynchio-0.6.0, asyncio-0.6.0, asynchio\_default\_fixture\_loop\_scope=None, asyncio\_default\_test\_loop\_scope=function collected 3 items

backend/test\_email\_sender/service/test\_email\_service.py::test\_send\_email\_success PASSED [ 33%]
backend/test\_email\_sender/service/test\_email\_service.py::test\_send\_email\_failure PASSED [ 66%]
backend/test\_email\_sender/service/test\_email\_service.py::test\_send\_email\_missing\_credentials PASSED [100%]

(.venv) adrijachakraborty@Adrijas-MacBook-Pro ~Desktop/code-custodians/ATS-ASAP-Code-Custodians/src main %

To generate a comment:

**Email Controller Unit Testing Best Practice**

```

email_controller.py test_extractor.py
can you write the unit tests for this email controller following best practices?
refer @test_extractor.py to write the test in similar way

```

I'll help you write unit tests for the email controller following similar patterns and best practices as seen in `test_extractor.py`. I'll create comprehensive tests that cover the main functionality and error cases.

Let me create a new test file for the email controller.

```

test_email_controller.py +119 -1
import pytest
from fastapi import HTTPException
from unittest.mock import patch, MagicMock
import pandas as pd
from datetime import datetime, timedelta
import io

from src.backend.email_sender.api.email_controller import process_emails

class DummyUploadFile:
 def __init__(self, filename, content: bytes):
 self.filename = filename
 self.file = io.BytesIO(content)

 async def read(self):
 return self.file.getvalue()

 async def __aenter__(self):
 return self

 async def __exit__(self, exc_type, exc_val, exc_tb):
 self.file.close()

def create_test_excel(df: pd.DataFrame) -> bytes:
 output = io.BytesIO()
 df.to_excel(output, index=False, engine='openpyxl')
 output.seek(0)
 return output.read()

```

I've created a comprehensive test file for the email controller that follows similar patterns to `test_extractor.py`. Here's what I've included:

**Helper Classes and Functions:**

- `DummyUploadFile`: A mock class to simulate FastAPI's `UploadFile`
- `create_test_excel`: Helper function to create test Excel files

**Test Cases:**

- `test_process_emails_success`: Tests the happy path with both old and recent records
- `test_process_emails_invalid_file_type`: Tests rejection of non-xlsx files
- `test_process_emails_missing_required_columns`: Tests validation of required columns
- `test_process_emails_missing_email_template`: Tests handling of missing environment variables
- `test_process_emails_email_send_failure`: Tests handling of email send failures

For the code present, we get this error:

```

...
Import 'fastapi' could not be resolved
...
How can I resolve this? If you propose a fix, please make it concise.

```

## 6. Create ReadMe.md

# ATS-ASAP Code Custodians

## Project Structure

```

src/
 └── Frontend/ # React frontend application
 ├── src/ # Source files
 ├── public/ # Static assets
 └── package.json # Frontend dependencies

 └── backend/ # Python backend application
 ├── api.py # Main API endpoints
 ├── resume_parser/ # Resume parsing module
 ├── email_sender/ # Email notification system
 └── settings.py # Configuration settings

 └── utils/ # Shared utilities

```

## Contributing

1. Fork the repository
2. Create a feature branch
3. Commit your changes
4. Push to the branch
5. Create a Pull Request

## License

This project is proprietary and confidential.

## Support

For support and questions, please contact the development team.

I'll help you create a detailed README.md file for your application. First, let me gather some information about the project structure and components.

Listed 13 items in ATS-ASAP-Code-Custodians/src/frontend

Listed 9 items in ATS-ASAP-Code-Custodians/src/backend

Now I'll create a comprehensive README.md file for your application.

**ATS-ASAP (Automated Talent Sourcing and Applicant Processing)** is an integrated solution for talent matching and applicant processing.

### Key Features

- Intelligent skill-to-role matching based on multiple criteria:
- Technology stack
- Years of experience
- Certifications
- Other relevant qualifications

I've created a comprehensive README.md file for your application that includes:

1. A detailed project overview explaining the purpose and key features
2. The complete tech stack used in both frontend and backend
3. Prerequisites and installation instructions for both frontend and backend
4. Step-by-step instructions for running the application

Edited 1 file +142 -1

now you give me the technical solution for

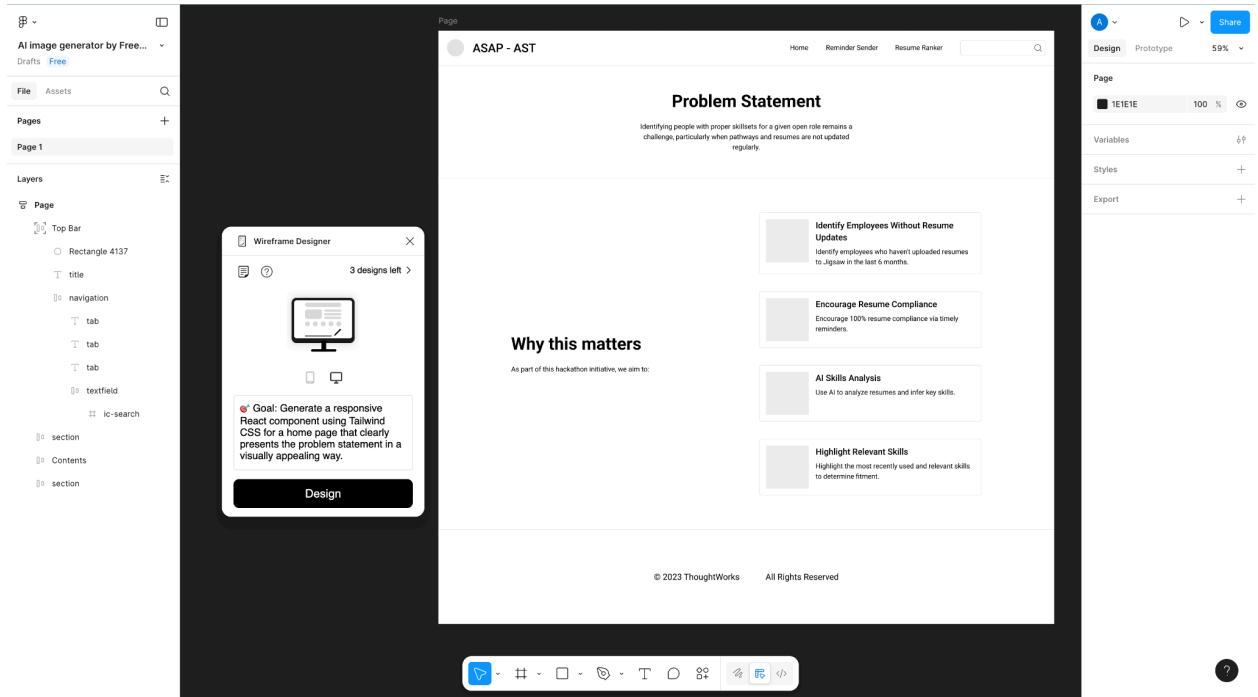
# Prototype

## 1. UI Design

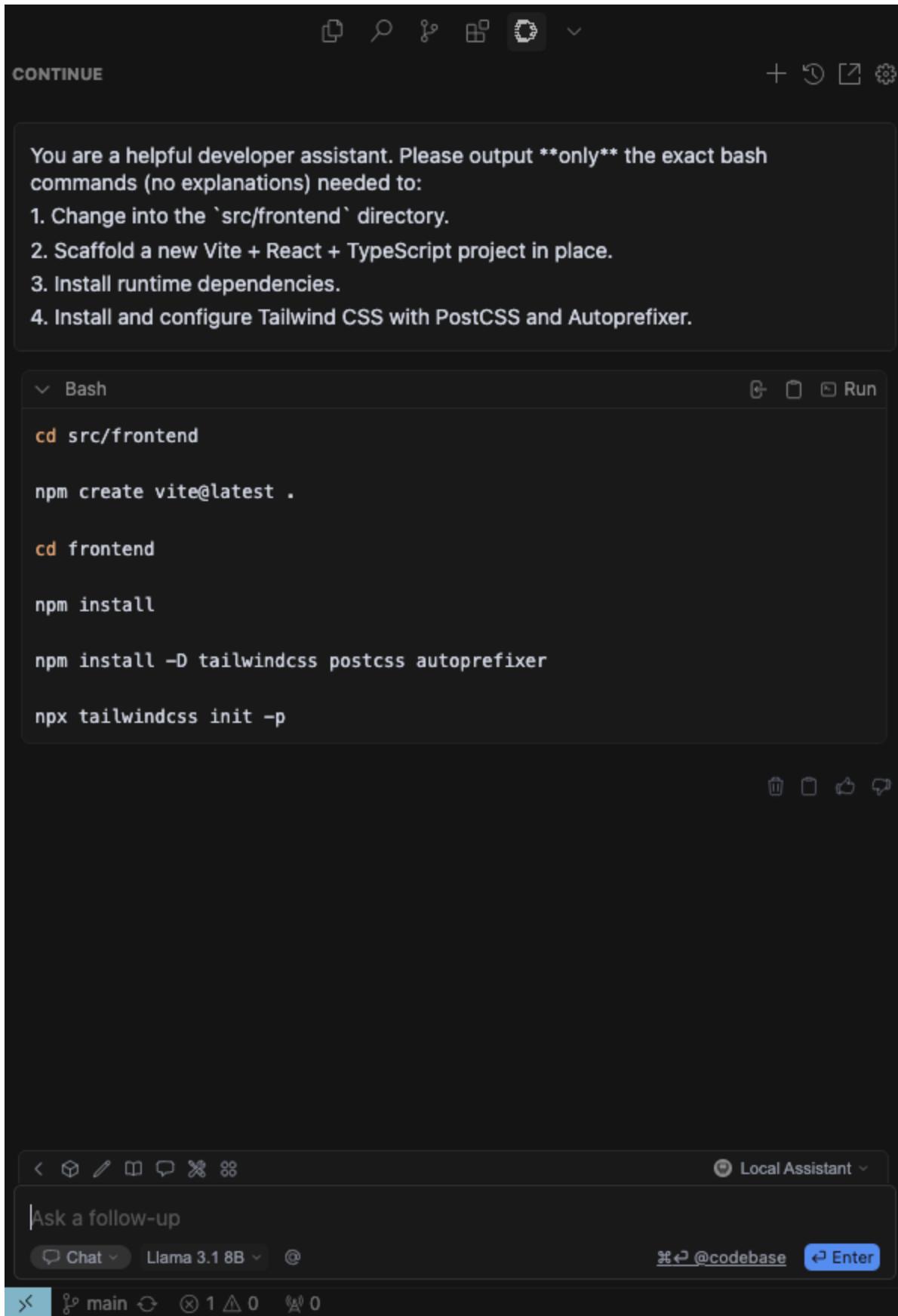
The image shows a Figma workspace with two open designs:

**Top Design (ASAP - AST):** This is a resume ranking web app. The wireframe shows a navigation bar with 'Home', 'Features', 'Pricing', and 'Contact'. Below it is a search bar with placeholder 'Paste your resumes below' and a 'Search' button. The main area is titled 'Results' and displays four candidate cards: Candidate A (Software Developer, Score 90, Top Performer), Candidate B (Software Developer, Score 75, Interim), Candidate C (Data Engineer, Score 80, High Recommended), and Candidate D (Project Manager, Score 75). The Figma sidebar on the left shows various components like Top Bar, title, navigation, tab, text, section, and Form.

**Bottom Design (ASAP - AST):** This is a reminder email sender. The wireframe shows a navigation bar with 'Home', 'Features', 'Pricing', and 'Contact'. Below it is a file upload section titled 'Upload Reminder Sheet' with a placeholder 'Choose your .xlsx file' and a 'Send Reminder Emails' button. The main area is titled 'Required Environment Variables' with a note 'Make sure you have the required variables set up.' It lists three variables: SENDER\_EMAIL (Your Gmail or SMTP-enabled sender address), SENDER\_PASSWORD (App-specific password or email token), and EMAIL\_TEMPLATE (Path to the email template file (HTML or TXT)). The Figma sidebar on the left shows a wireframe designer component with a note 'Create a single-page UI under the same styling as the current ASAP-AST application.' and a 'Page Title' section for 'Reminder Email Sender'.



## 2. Setting up the npm

CONTINUE + ⌂ ⌓ ⌚ ⌚

You are a helpful developer assistant. Please output \*\*only\*\* the exact bash commands (no explanations) needed to:

1. Change into the `src/frontend` directory.
2. Scaffold a new Vite + React + TypeScript project in place.
3. Install runtime dependencies.
4. Install and configure Tailwind CSS with PostCSS and Autoprefixer.

Bash

```
cd src/frontend
npm create vite@latest .

cd frontend

npm install

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p
```

Local Assistant

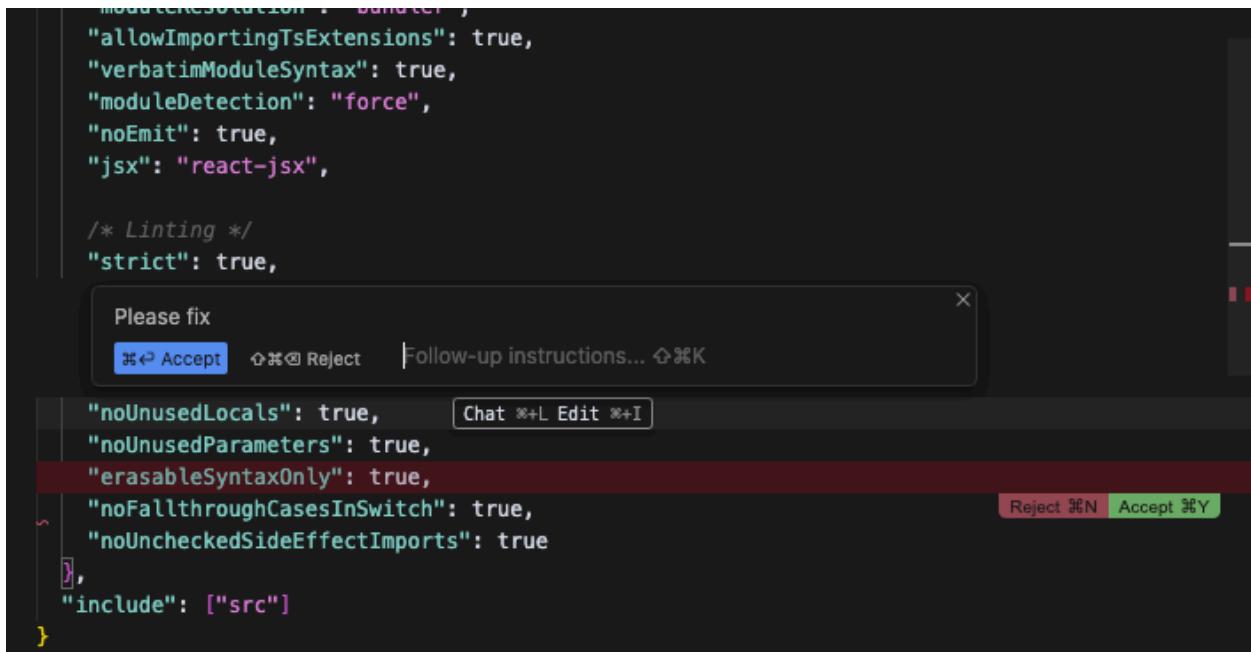
Ask a follow-up

Chat Llama 3.1 8B @

⌘ ↵ @codebase ⏪ Enter

✖ main ⌂ ⌓ ⌚ ⌚ 1 △ 0 ⌚ 0

3. Fix error:



```
moduleResolution: "bundler",
"allowImportingTsExtensions": true,
"verbatimModuleSyntax": true,
"moduleDetection": "force",
"noEmit": true,
"jsx": "react-jsx",

/* Linting */
"strict": true,
```

Please fix

Accept Reject Follow-up instructions... ⌘K

```
"noUnusedLocals": true, Chat ⌘L Edit ⌘I
"noUnusedParameters": true,
"erasableSyntaxOnly": true,
"noFallthroughCasesInSwitch": true,
"noUncheckedSideEffectImports": true
],
"include": ["src"]
```

Reject ⌘N Accept ⌘Y

The screenshot shows a code editor interface with three tabs at the top: '# index.css 5, U', 'App.tsx 9+, U', and '# App.css U'. The '# App.css U' tab is active, displaying a portion of the file 'App.css'.

```
index.css 5, U App.tsx 9+, U ● # App.css U
src > frontend > src > App.tsx
1 @tailwind base;
2 @tailwind components;
3 @tailwind utilities;
4
5 /* global resets & Inter font */
6 @layer base {
7 body {
8 background-color: #F5F7FA;
9 color: #333333;
10 font-family: 'Inter', sans-serif;
11 margin: 0;
12 padding: 0;
13 }
14 }

15 /* two-column layout: 1fr on mobile, 30%/70% on md+ */
16 @layer components {
17 .app-container {
18 @apply mx-auto min-h-screen p-8 grid grid-cols-1 md:[grid-template-columns:30%_70%] gap-8;
19 max-width: 1440px;
20 }
21 .app-container {
22 margin: 0 auto;
23 min-height: 100vh;
24 padding: 2rem;
25 display: grid;
26 grid-template-columns: 1fr;
27 gap: 2rem;
28 max-width: 1440px;
29 }
30 }
31
32
```

A modal window titled 'Please fix' is overlaid on the code editor. It contains three buttons: 'Accept' (highlighted), 'Reject', and 'Follow-up instructions...'. At the bottom right of the modal is a small 'X' icon.

At the bottom right of the code editor, there is a status bar with the text 'Ln 25, Col 1' and 'Spaces: 2'.

4. Create project structure:

The screenshot shows a code editor interface with a sidebar on the left displaying a file tree. The file tree includes files like `index.css`, `App.tsx`, `ResumeUploader.tsx`, and `CandidateCard.tsx`. The main editor area shows the `ResumeUploader.tsx` file content:

```
index.css 4, U App.tsx U ResumeUploader.tsx U CandidateCard.tsx U
src > frontend > src > components > ResumeUploader.tsx ...
1 import React, { useState } from 'react';
2
3 interface ResumeUploaderProps {
4 onSubmit: (text: string) => void;
5 }
6
7 export const ResumeUploader: React.FC<ResumeUploaderProps> = ({ onSubmit }) => {
8 const [resumeText, setResumeText] = useState('');
9
10 const handleSubmit = () => {
11 onSubmit(resumeText);
12 };
13
14 return (
15 <div className="w-full max-w-2xl mx-auto p-4">
16 <textarea
17 className="w-full h-64 p-4 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-300 placeholder='Paste resumes here...'"
18 value={resumeText}
19 onChange={(e) => setResumeText(e.target.value)}
20 />
21 <button
22 className="mt-4 px-6 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition"
23 onClick={handleSubmit}
24 >
25 Submit
26 </button>
27 </div>
28);
29 }
30
```

The right side of the interface features a "New chat" window titled "App.tsx". It contains a message from the AI asking to create four new React-TypeScript files under `src/frontend/src/components/` with specific requirements. Below this, there are sections for `ResumeUploader.tsx` and `CandidateCard.tsx`, each with a brief description and a "Generating..." status indicator.

```
index.css 4, U App.tsx U ResumeUploader.tsx U CandidateList.tsx U ...
src > frontend > src > components > CandidateList.tsx ...
1 import React from 'react';
2 import { CandidateCard } from './CandidateCard';
3
4 interface Candidate {
5 name: string;
6 title: string;
7 score: number;
8 badge?: string;
9 }
10
11 interface CandidateListProps {
12 candidates: Candidate[];
13 }
14
15 export const CandidateList: React.FC<CandidateListProps> = ({ candidates }) => {
16 return (
17 <div className="w-full max-w-3xl mx-auto p-4">
18 <div className="mb-6">
19 <h2 className="text-2xl font-bold text-gray-800">Results</h2>
20 <p className="text-gray-600 mt-1">
21 {candidates.length} candidates found
22 </p>
23 </div>
24 <div className="space-y-4">
25 {candidates.map((candidate, index) => (
26 <CandidateCard key={index} {...candidate} />
27))}
28 </div>
29 </div>
30);
31 }
```

New chat

App.tsx

Create four new React-TypeScript files under src/frontend/src/components/ with the following names and basic boilerplate:

ResumeUploader.tsx

A component that renders a multi-line <textarea> labeled "Paste resumes here..." and a "Submit" <button>.

Accepts an onSubmit(text: string) prop, stores the textarea value in state, and calls onSubmit when the button is clicked.

CandidateCard.tsx

Generating...

I'll help you create these four React TypeScript components with the specified requirements. Let me create each file one by one.

First, let me create the `ResumeUploader.tsx` component:

```
ResumeUploader.tsx +30 -1 ✓
import React, { useState } from 'react';

interface ResumeUploaderProps {
 onSubmit: (text: string) => void;
}

export const ResumeUploader: React.FC<ResumeUploaderProps> = () => {
 const [resumeText, setResumeText] = useState('');

 const handleSubmit = () => {
 onSubmit(resumeText);
 };
}
```

```
CandidateList.tsx U Header.tsx U CandidateCard.tsx U # App.css U ...
src > frontend > src > components > Header.tsx ...
1 import React from 'react';
2
3 export const Header: React.FC = () => {
4 return (
5 <header className="w-full bg-white shadow-sm py-6">
6 <div className="max-w-4xl mx-auto px-4">
7 <h1 className="text-3xl font-bold text-gray-900 mb-2">
8 What would you like to do today?
9 </h1>
10 <p className="text-lg text-gray-600">
11 ASAP AI AST is your intelligent AI assistant that helps you rank resumes efficiently.
12 </p>
13 </div>
14 </header>
15);
16 }
```

New chat

App.tsx

Create four new React-TypeScript files under src/frontend/src/components/ with the following names and basic boilerplate:

ResumeUploader.tsx

A component that renders a multi-line <textarea> labeled "Paste resumes here..." and a "Submit" <button>.

Accepts an onSubmit(text: string) prop, stores the textarea value in state, and calls onSubmit when the button is clicked.

CandidateCard.tsx

Restore checkpoint

I'll help you create these four React TypeScript components with the specified requirements. Let me create each file one by one.

First, let me create the `ResumeUploader.tsx` component:

```
ResumeUploader.tsx +30 -1 ✓
import React, { useState } from 'react';

interface ResumeUploaderProps {
 onSubmit: (text: string) => void;
}

export const ResumeUploader: React.FC<ResumeUploaderProps> = () => {
 const [resumeText, setResumeText] = useState('');

 const handleSubmit = () => {
 onSubmit(resumeText);
 };
}
```

Now, let me create the `CandidateCard.tsx` component:

```
Edited 4 files +113 -4
Reject all ✖ | Accept all ✅
App.tsx
Plan, search, build anything
Agent AI Auto
```

The screenshot shows a development environment with an open project structure on the left and a code editor on the right.

**File Tree (Left):**

```

AT-SAP-CODE-CUSTODIANS
 documentation
 envs
 kubernetes
 licenses
 logs
 models
 notebooks
 src
 __pycache__
 backend
 components
 frontend
 node_modules
 public
 src
 assets
 components
 CandidateCard.tsx
 CandidateList.tsx
 Header.tsx
 ResumeUploader.tsx
 # App.css
 # App.tsx
 index.css
 main.tsx
 vite-env.d.ts
 .gitignore
 eslint.config.js
 index.html
 package-lock.json
 package.json
 postcss.config.cjs
 README.md
 tailwind.config.cjs
 OUTLINE
 TIMELINE

```

**Code Editor (Right):**

**React-TypeScript Component Creation**

**App.tsx**

```

Create four new React-TypeScript files under
src/frontend/src/components/ with the following names and basic
boilerplate:

ResumeUploader.tsx

A component that renders a multi-line <textarea> labeled "Paste
resumes here..." and a "Submit" <button>.

Accepts an onSubmit(text: string) prop, stores the textarea value in
state, and calls onSubmit when the button is clicked.

CandidateCard.tsx

```

**Code Snippet (ResumeUploader.tsx):**

```

import React, { useState } from 'react';

interface ResumeUploaderProps {
 onSubmit: (text: string) => void;
}

export const ResumeUploader: React.FC<ResumeUploaderProps> = ({ onSubmit }) => {
 const [resumeText, setResumeText] = useState('');

 const handleSubmit = () => {
 onSubmit(resumeText);
 };

 return (
 <div className="flex items-center p-4 bg-white rounded-lg shadow-md mb-4">
 <div className="w-12 h-12 bg-gray-200 rounded-full mr-4 flex-shrink-0" />
 <div className="flex-grow">
 <h3 className="text-lg font-semibold">{name}</h3>
 <title && <p className="text-gray-600 text-sm">{title}</p>
 <p className="text-sm text-gray-500">Score: {score}</p>
 <badge && (

 {badge}

)
 </div>
 <div className="ml-4">

 {score}

 </div>
 </div>
);
}

```

**Instructions:**

I'll help you create these four React TypeScript components with the specified requirements. Let me create each file one by one.

First, let me create the `ResumeUploader.tsx` component:

**ResumeUploader.tsx +30 -1**

```

import React, { useState } from 'react';

interface ResumeUploaderProps {
 onSubmit: (text: string) => void;
}

export const ResumeUploader: React.FC<ResumeUploaderProps> = ({ onSubmit }) => {
 const [resumeText, setResumeText] = useState('');

 const handleSubmit = () => {
 onSubmit(resumeText);
 };

 return (
 <div className="flex items-center p-4 bg-white rounded-lg shadow-md mb-4">
 <div className="w-12 h-12 bg-gray-200 rounded-full mr-4 flex-shrink-0" />
 <div className="flex-grow">
 <h3 className="text-lg font-semibold">{name}</h3>
 <title && <p className="text-gray-600 text-sm">{title}</p>
 <p className="text-sm text-gray-500">Score: {score}</p>
 <badge && (

 {badge}

)
 </div>
 <div className="ml-4">

 {score}

 </div>
 </div>
);
}

```

**Next Step:**

Now, let me create the `CandidateCard.tsx` component:

**Edited 4 files +113 -4**

**Accept all ✅ | Reject all ❌**

**App.tsx**

Plan, search, build anything

Agent AI Auto

## 5. Generate code

App.tsx U

ResumeUploader.tsx U

Header.tsx U

CandidateList.tsx U

React-TypeScript Component Creation

```

1 import React from 'react';
2
3 export const Header: React.FC = () => {
4 return [
5 <header className="w-full bg-white shadow-sm py-6">
6 <div className="max-w-4xl mx-auto px-4">
7 <h1 className="text-3xl font-bold text-gray-900 mb-2">
8 What would you like to do today?
9 </h1>
10 <header className="w-full bg-white border-b border-gray-200">
11 /* Top Navigation Bar */
12 <div className="flex items-center justify-between max-w-7xl mx-auto px-6 py-3">
13 <div className="flex items-center space-x-2">
14 <div className="w-6 h-6 bg-gray-300 rounded-full" />
15 ASAP - AST
16 </div>
17 <nav className="flex items-center space-x-8">
18 Home
19 Features
20 Pricing
21 Contact
22 </nav>
23 <div className="relative">
24 <input
25 type="text"
26 placeholder="Search in site"
27 className="pl-3 pr-8 py-1 border border-gray-300 rounded focus:outline-none focus:ring-gray-300" />
28
29 <svg width="16" height="16" fill="none" viewBox="0 0 24 24" stroke="currentColor"></svg>
30
31 </div>
32 </div>
33 <div className="max-w-3xl mx-auto px-4 py-12 text-center">
34 <h1 className="text-4xl font-bold mb-4">What would you like to do today?</h1>
35 <p className="text-lg text-gray-600">
36 ASAP - AST is your intelligent AI assistant that helps you rank resumes efficiently.
37 </p>
38 </div>
39];
40 };
41
```

1 / 1 ▾ Reject file ⌂ Accept file ⌂ 2 / 3 files ▾

The `ResumeUploader` now matches the image: it has a section title, a single-line input, a horizontally aligned submit button, and a clean section background.

Next, I will:

- Update `CandidateList` to use a grid layout and update the heading/subtext.
- Update `CandidateCard` for a lighter card style, left-aligned avatar, and badge style.
- Add a footer component.

Let's update `CandidateList` and `CandidateCard` next.

CandidateList.tsx +15 -13

```

<section className="w-full bg-white py-12 border-b border-gray-200">
 <div className="max-w-5xl mx-auto px-4">
 <div className="mb-8 text-center">
 <h2 className="text-3xl font-bold mb-2">Results</h2>
 <p className="text-gray-600">
 Here are the candidates ranked based on their relevance.
 </p>
 </div>
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
 {candidates.map((candidate, index) => (
 <CandidateCard key={index} {...candidate} />
))}
 </div>
 </div>
</section>

```

Generating Stop ⌂ Accept ⌂

@ App.tsx

Plan, search, build anything

Agent ⌂ Auto

React-TypeScript Component Creation

- Update `CandidateCard` for a lighter card style, left-aligned avatar, and badge style.
- Add a footer component.

Let's update `CandidateList` and `CandidateCard` next.

`CandidateList.tsx +15 -13`

```
<section className="w-full bg-white py-12 border-t border-b border-gray-100">
 <div className="max-w-5xl mx-auto px-4">
 <div className="mb-8 text-center">
 <h2 className="text-3xl font-bold mb-2">ResumeUploader</h2>
 <p className="text-gray-600">Here are the candidates ranked based on their scores</p>
 </div>
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-4">
 {candidates.map((candidate, index) => (
 <CandidateCard key={index} {...candidate} />
))}
 </div>
 </div>
</section>
```

`CandidateCard.tsx +6 -11`

```
<div className="flex items-center p-4 bg-white rounded-lg">
 <div className="w-12 h-12 bg-gray-200 rounded-full">
 {candidate.avatar}
 </div>
 <div className="flex items-start p-6 bg-gray-200 rounded-md gap-4">
 <div className="flex-grow">
 <h3 className="text-lg font-semibold">{candidate.name}</h3>
 <div>
 {candidate.title} && {candidate.experience}
 </div>
 <div>
 Score: {candidate.score}
 </div>
 </div>
 </div>
</div>
```

Generating... Stop ⌘+Q Accept all ⌘+P

App.tsx Plan, search, build anything

Agent Auto

4, U      ⚡ App.tsx U    ●    ⚡ CandidateList.tsx U    ✘    ⚡ CandidateCard.tsx U    # App.css U    🌐 ...

```

src > frontend > src > components > ⚡ CandidateList.tsx > ...
1 import React from 'react';
2 import { CandidateCard } from './CandidateCard';
3
4 interface Candidate {
5 name: string;
6 title?: string;
7 score: number;
8 badge?: string;
9 }
10
11 interface CandidateListProps {
12 candidates: Candidate[];
13 }
14
15 export const CandidateList: React.FC<CandidateListProps> = ({ candidates }) => {
16 return (
17 <div className="w-full max-w-3xl mx-auto p-4">
18 <div className="mb-6">
19 <h2 className="text-2xl font-bold text-gray-800">Results</h2>
20 <p className="text-gray-600 mt-1">
21 {candidates.length} candidates found
22 </p>
23 </div>
24 <div className="space-y-4">
25 {candidates.map((candidate, index) => (
26 <CandidateCard key={index} {...candidate} />
27))}
28 </div>
29 </div>
30 </div>
31 </section>

```

The candidate list and cards now use a grid layout and lighter card style, closely matching the image.

Finally, I will add a `Footer.tsx` component for the copyright and links at the bottom, as shown in your screenshot.

Generating... Stop ⌂ Accept all ⌂

@ ⚡ App.tsx Plan, search, build anything

Agent Auto

# index.css 4, U    App.tsx U    Footer.tsx X    CandidateCard.tsx    # App.cs 70    ...

```

src > frontend > src > components > Footer.tsx > ...
1 import React from 'react';
2
3 export const Footer: React.FC = () => (
4 <footer className="w-full bg-white border-t border-gray-200 py-8 mt-12">
5 <div className="max-w-4xl mx-auto px-4 flex flex-col md:flex-row items-center justify-between">
6 <div> 2023 ASAP - AST</div>
7 <div className="flex space-x-6">
8 Privacy Policy
9 Terms of Service
10 </div>
11 </div>
12 </footer>
13);

```

Reject ⌘N    Accept ⌘Y

**React-TypeScript Component Creation**

```

CandidateCard.tsx +6 -11 *
return (
 <div className="flex items-center p-4 bg-white rounded-md">
 <div className="w-12 h-12 bg-gray-200 rounded-full"></div>
 <div className="flex items-start p-6 bg-gray-50 rounded-md">
 <div className="w-14 h-14 bg-gray-200 rounded-md"></div>
 <div className="flex-grow">
 <h3 className="text-lg font-semibold">{name}</h3>
 {title && <p className="text-gray-600 text-sm">{title}</p>}
 <p className="text-sm text-gray-500">Score: {score}</p>
 <h3 className="text-lg font-semibold mb-0.5">{name}</h3>
 {title && <p className="text-gray-600 text-sm">{title}</p>}
 </div>
 </div>
 </div>

```

The candidate list and cards now use a grid layout and lighter card style, closely matching the image.

Finally, I will add a `Footer.tsx` component for the copyright and links at the bottom, as shown in your screenshot.

**Footer.tsx (new) +13 -1 \***

```

import React from 'react';

export const Footer: React.FC = () => (
 <footer className="w-full bg-white border-t border-gray-200 py-8 mt-12">
 <div> 2023 ASAP - AST</div>
 <div className="flex space-x-6">
 Privacy Policy
 Terms of Service
 </div>
 </footer>
)

```

Your UI components now closely match the provided

Generating    Stop ⌘S    Accept all ⌘P

@ App.tsx

Plan, search, build anything

Agent Auto

1 / 1    Reject file ⌘D    Accept file ⌘E    2 / 2 files >





The screenshot shows a code editor interface with two main panes. The left pane displays the `App.tsx` file content:

```
index.css 4, U ⚡ App.tsx 1, U ×
src > frontend > src > ⚡ App.tsx > ...
1 import React, { useState } from 'react';
2 import { Header } from './components/Header';
3 import { ResumeUploader } from './components/ResumeUploader';
4 import { CandidateList } from './components/CandidateList';
5 import { Footer } from './components/Footer';
6
7 const initialCandidates = [
8 { name: 'Candidate A', title: 'Software Engineer', score: 85, badge: 'Top Performer' },
9 { name: 'Candidate B', title: 'Data Analyst', score: 78, badge: 'Strong Skills' },
10 { name: 'Candidate C', title: 'Web Developer', score: 90, badge: 'Highly Recommended' },
11 { name: 'Candidate D', title: 'Project Manager', score: 75 },
12];
13
14 export const App: React.FC = () => {
15 const [candidates, setCandidates] = useState(initialCandidates);
16
17 const handleResumeSubmit = (text: string) => {
18 // Placeholder: In a real app, process resumes and update candidates
19 // For now, just log the text
20 console.log('Submitted resumes:', text);
21 };
22
23 return (
24 <div className="min-h-screen flex flex-col bg-gray-50">
25 <Header />
26 <main className="flex-grow">
27 <ResumeUploader onSubmit={handleResumeSubmit} />
28 <CandidateList candidates={candidates} />
29 </main>
30 <Footer />
31 </div>
32);
33 }
34
35
36 export default App;
```

The right pane shows a "React-TypeScript Component Creation" tool window with the following text:

components for a complete page layout?

call the components from App.tsx as shown in the image

Generating...

Stop ⌘Q

App.tsx +33-1 \*

```
import React, { useState } from 'react';
import { Header } from './components/Header';
import { ResumeUploader } from './components/ResumeUploader';
import { CandidateList } from './components/CandidateList';
import { Footer } from './components/Footer';

const initialCandidates = [
 { name: 'Candidate A', title: 'Software Engineer', score: 85, badge: 'Top Performer' },
 { name: 'Candidate B', title: 'Data Analyst', score: 78, badge: 'Strong Skills' },
 { name: 'Candidate C', title: 'Web Developer', score: 90, badge: 'Highly Recommended' },
 { name: 'Candidate D', title: 'Project Manager', score: 75 },
];
```

Reject ⌘N | Accept ⌘Y

Generating...

Stop ⌘Q | Accept ⌘P

App.tsx

Plan, search, build anything

Agent ⌘I Auto

## 6. Create fast api

```

src > backend > api.py ...
1 from fastapi import FastAPI
2 from fastapi.middleware.cors import CORSMiddleware
3 import csv
4 from pathlib import Path
5
6 app = FastAPI()
7
8 # Enable CORS for all origins
9 app.add_middleware(
10 CORSMiddleware,
11 allow_origins=["*"],
12 allow_credentials=True,
13 allow_methods=["*"],
14 allow_headers=["*"]
15)
16
17 @app.get("/rankings")
18 async def get_rankings():
19 csv_path = Path("resume_parser/output/final_ranking.csv")
20 rankings = []
21
22 with open(csv_path, mode='r') as file:
23 reader = csv.DictReader(file)
24 for row in reader:
25 rankings.append({
26 "candidate_name": row["candidate_name"],
27 "score": float(row["score"])
28 })
29
30 return rankings
31
32 if __name__ == "__main__":
33 import uvicorn
34 uvicorn.run(app, host="0.0.0.0", port=8000)

```

FastAPI Server File Generation

Requirements:

- Load and serve 'final\_ranking.csv' located at: 'src/backend/resume\_parser/output/final\_ranking.csv'

Let me create the FastAPI server file with the specified requirements.

api.py (new) +34 -1

```

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
import csv
from pathlib import Path

app = FastAPI()

Enable CORS for all origins
app.add_middleware(
 CORSMiddleware,
 allow_origins=["*"],
 allow_credentials=True
)

```

To run the server:

- Navigate to the `src/backend` directory
- Run: `uvicorn api:app --reload`

The server will start on `http://localhost:8000` and you can access the rankings at `http://localhost:8000/rankings`.

Review changes

Edited 1 file +34 -1    Reject ✖ Accept ✅

Add context

Plan, search, build anything

Agent Auto

Ln 1, Col 1 | Spaces: 4 | UTF-8 | LF | Python 3.12.9 ('venv': venv) | Cursor Tab | Continue

## 7. Connecting backend and frontend

```

src > backend > api.py ...
1 from fastapi import FastAPI, HTTPException
2 from fastapi.middleware.cors import CORSMiddleware
3 import csv
4 from pathlib import Path
5 import uvicorn
6
7 import json
8 from pydantic import BaseModel
9 from src.backend.resume_parser.ranking.jd_analyzer import generate_weights_from_jd
10 from src.backend.resume_parser.ranking.scorer import build_candidate_profiles
11 from src.backend.resume_parser.ranking.ranker import rank_candidates
12 from src.backend.resume_parser.config_models import LLMConfig
13 import os
14
15 app = FastAPI()
16
17 # Enable CORS for all origins
18 app.add_middleware(
19 CORSMiddleware,
20 allow_origins=["*"],
21 allow_credentials=True,
22 allow_methods=["*"],
23 allow_headers=["*"],
24)
25
26 class JobDescription(BaseModel):
27 jd: str
28
29 @app.get("/rankings")
30 async def get_rankings():
31 csv_path = Path("resume_parser/output/final_ranking.csv")
32 rankings = []
33
34 with open(csv_path, mode='r') as file:
35 reader = csv.DictReader(file)
36 for row in reader:
37 rankings.append({
38 "candidate_name": row["candidate_name"],
39 "score": float(row["score"])
40 })
41
42 return rankings
43
44 try:
45
46 if (jd.trim() == ""):
47 raise HTTPException(status_code=400, detail="Job description cannot be empty")
48
49 jd_analyzer = generate_weights_from_jd(jd)
50 candidate_profiles = build_candidate_profiles(jd_analyzer)
51 ranked_candidates = rank_candidates(candidate_profiles)
52
53 response = {
54 "rankings": ranked_candidates
55 }
56
57 return response
58
59 except Exception as e:
60 return {"error": str(e)}

```

New chat

Now I'll implement the required changes. First, let's update the backend API:

api.py +71 -14

```

from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import csv
from pathlib import Path
import uvicorn
import json
from pydantic import BaseModel
from src.backend.resume_parser.ranking.jd_analyzer import generate_weights_from_jd
from src.backend.resume_parser.ranking.scorer import build_candidate_profiles
from src.backend.resume_parser.ranking.ranker import rank_candidates
from src.backend.resume_parser.config_models import LLMConfig

```

Now let's update the frontend App.tsx:

App.tsx

```

if (!jd.trim()) return;
setIsProcessing(true);
setError(null);

try {
 const res = await fetch(`http://localhost:8000/rankings`, {
 method: 'POST',
 headers: {
 'Content-Type': 'application/json',
 },
 body: JSON.stringify({
 jd
 })
 });
}

```

Generating...    Stop ✖ Accept ✅

Add context

Plan, search, build anything

Agent Auto

Ln 92, Col 49 | Spaces: 4 | UTF-8 | LF | Python 3.12.9 ('venv': venv) | Cursor Tab | Continue

src > backend > api.py > analyze\_jd

process Aa ⌂ 6 of 6 ↑ ↓ = x

```

1 from fastapi import FastAPI, HTTPException
2 from fastapi.middleware.cors import CORSMiddleware
3 import csv
4 from pathlib import Path
5 import uvicorn
6 import os
7 import sys
8 import json
9 import yaml
10 from pydantic import BaseModel
11 from typing import Dict, List
12
13 # Add the project root to Python path
14 sys.path.append(os.path.dirname(os.path.abspath(__file__)))
15
16 from src.backend.resume_parser.config_models import ResumeParserConfig, LLMConfig
17 from src.backend.resume_parser.main import main as process_resumes
18 from src.backend.resume_parser.ranking.jd_utils import analyze_jd_text
19 from src.backend.resume_parser.scoring import build_candidate_profiles
20 from src.backend.resume_parser.ranking.ranker import rank_candidates
21
22 def load_config() -> ResumeParserConfig:
23 config_path = Path(__file__).parent / "resume_parser" / "config.yaml"
24 try:
25 with open(config_path, 'r') as f:
26 config_data = yaml.safe_load(f)
27
28 llm_config = LLMConfig(**config_data['llm'])
29 return ResumeParserConfig(
30 data_folder=config_data['data_folder'],
31 resume_extension=config_data['resume_extension'],
32 llm=llm_config
33)
34 except Exception as e:
35 print(f"Error loading config: {e}")
36 raise HTTPException(
37 status_code=500,
38 detail="Failed to load configuration"
39)
40
41 app = FastAPI()
42
43 # Enable CORS for all origins
44 app.add_middleware(
45 CORSMiddleware,
46 allow_origins=["*"],
47 allow_credentials=True,
48 allow_methods=["*"],
49 allow_headers=["*"]
50)
51
52 class JobDescription(BaseModel):

```

Reject ⌘N Accept ⌘Y

api.py +21 -12

import os  
import sys  
import json  
import yaml  
from pydantic import BaseModel  
from typing import Dict, List

6 hidden lines  
from src.backend.resume\_parser.ranking.scorer import build\_candidate\_profiles  
from src.backend.resume\_parser.ranking.ranker import rank\_candidates

def load\_config() -> ResumeParserConfig

I've made the following improvements to the code:

- Added a new `load_config()` function that:
  - Reads the YAML configuration file
  - Parses it into the appropriate `ResumeParserConfig` and `LLMConfig` objects
  - Includes proper error handling
  - Returns the configuration object
- Replaced the hardcoded configuration in the `analyze_jd` endpoint with a call to `load_config()`
- Added the `yaml` import to handle YAML file parsing

The code now uses the configuration from `config.yaml` instead of hardcoding values. This makes the code more maintainable and configurable, as you can change settings by modifying the YAML file without touching the code.

A few benefits of this refactoring:

- Configuration is centralized in one place
- Easier to modify settings without changing code
- Better separation of concerns
- More maintainable and cleaner code
- Consistent with best practices of externalizing configuration

Would you like me to make any additional improvements to the refactoring?

Review changes

Edited 1 file +21 -12

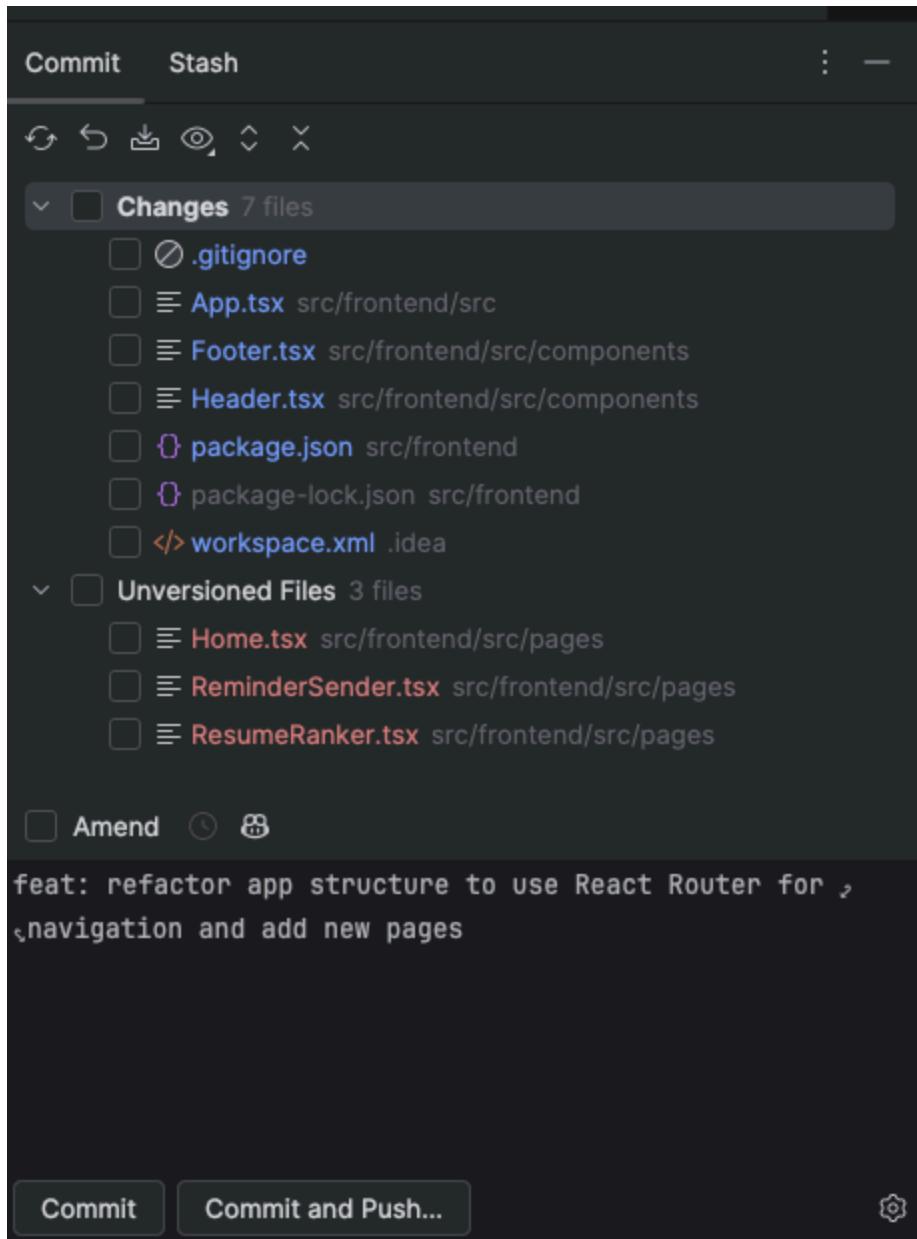
api.py config\_models.py

Plan, search, build anything

Agent Auto

Ln 139 Col 29 Spaces: 4 UTF-8 LF Python 3.12.9 ('venv': venv) Cursor Tab Continue

## 7. Commit using AI



## 8. Final output:

The screenshot shows the ThoughtWorks ASAP homepage. At the top, there's a dark header with the ThoughtWorks logo and the text "ASAP". On the right side of the header are links for "Home", "Staffing Assistant", and "Resume Update Notifier". Below the header is a light gray section titled "Welcome to ASAP" with a sub-section "Problem Statement". The "Problem Statement" text discusses how ThoughtWorks relies on Jigsaw or Pathways data for staffing decisions, but outdated resumes and missing skill insights can lead to misalignment and underutilization of talent. ASAP solves this by intelligently identifying resume gaps and prompting employees to update their profiles. A horizontal line separates this from the main content area. The main content area is divided into two columns: "Key Capabilities" on the left and "Contributors" on the right. The "Key Capabilities" section contains four boxes: "Identify Employees Without Resume Updates", "Encourage Resume Compliance", "AI-Driven Skills Analysis", and "Highlight Relevant Skills for Fitment". The "Contributors" section lists four team members with their names, email addresses, and roles: Arghya Banerjee (Application Developer / Consultant), Sayanta Kundu (Application Developer / Consultant), Adrita Chakraborty (Application Developer / Consultant), and Praveen Suryawanshi (Data Scientist / Senior Consultant). At the bottom of the page is a footer with the copyright notice "© 2025 ASAP", a "Privacy Policy" link, and a "Terms of Service" link.

The screenshot shows the ThoughtWorks ASAP search results page. At the top, there's a dark header with the ThoughtWorks logo and the text "ASAP". On the right side of the header are links for "Home", "Staffing Assistant", and "Resume Update Notifier". Below the header is a light gray section titled "What would you like to do today?". The text "ASAP is your intelligent AI assistant that helps you manage skill fitment." follows. The main content area is a search form with a text input field containing "#3 Position Title Backend / Cloud Engineer (4-8 years)" and a "Search Candidates" button. Below the search form is a section titled "Results" with the sub-instruction "Here are the candidates ranked based on the resumes you submitted.". Six candidate profiles are listed in a grid: #1 Meena Roy (Cloud Architect, Score: 94, Highly Recommended), #2 Balaji Vijayan (Application Developer, Score: 87, Well-Suited for the Role), #3 Rahul Nair (DevOps Engineer, Score: 66, Good Potential), #4 Vikram Mehta (Backend Engineer, Score: 48, May Require Further Evaluation), #5 Ayesha Singh (Machine Learning Engineer), and #6 Sarah Thomas (Data Analyst). At the bottom of the page is a footer with a circular logo containing a green "E" icon.

**⚠ Required Environment Variables**

Make sure you have the required variables set up.

SENDER\_EMAIL

Your Gmail or SMTP-enabled sender address.

SENDER\_PASSWORD

App-specific password or email token.

**Upload Reminder Sheet**

Upload an Excel file to send reminder emails in bulk.

 dummy-test-excel-email-read.xlsx

Selected file: dummy-test-excel-email-read.xlsx

**Submission Output**

Check the results of your submission.

Total Records

4

Processed Records

3

Emails Sent

3

**Email**

sayantakundu007@gmail.com

**Last Upload**

2024-05-11

**Status**

Sent

adrija.chakraborty@thoughtworks.com

2024-08-15

Sent

arghya.banerjee@thoughtworks.com

2024-01-30

Sent



# Deployment

## 1. Creating Docker files

```
You are a DevOps assistant. Generate Dockerfiles and a 'docker-compose.yml' for a full-stack project with the following requirements:
Frontend:
- Uses a JavaScript/TypeScript framework like React or Vite.
- Runs in development mode with hot reload.
- Uses Node.js 18 or above.
- Mounts local code for live development.
- Opens port 5173.
- Includes 'node_modules' mounting exception to avoid overwriting container dependencies.
- Uses environment variables like 'NODE_ENV' and 'WATCHPACK_POLLING'.
Backend:
- Uses Python with Poetry as the package manager.
- Based on Python 3.11 slim image.
- Has two stages: 'base' and 'prod' for multi-stage build.
- Installs dependencies using 'poetry install --no-root'.
- Starts the app on port 8000 using 'uvicorn'.
- Optionally exposes a debug port (5678).
- Mounts local code to '/app'.
NGINX:
- Uses official 'nginx:latest' image.
- Loads config from a custom 'nginx.conf' file.
- Mounts certificates from a 'configs/nginx/ssl' folder.
- Forwards HTTPS traffic to frontend/backend.
- Exposes port 5443.
- Depends on frontend and backend services.
docker-compose:
- Uses version '3.4'.
- Includes 3 services: 'ats-frontend', 'ats-backend', and 'ats-nginx'.
- Each service supports restart policies and replicas via '${ENV_VAR:-default}'.
- Volumes are correctly mounted.
- All services are in the same default network.
Return:
1. 'frontend/Dockerfile'
2. 'backend/Dockerfile' with multi-stage support
3. 'docker-compose.yml'
```

Here are the Dockerfiles and a docker-compose file that meets your requirements:

```
frontend/Dockerfile
└── Bash
language: bash frontend/Dockerfile

FROM node:18 as build

WORKDIR /app
COPY package*.json ./
RUN npm install --production

COPY . .
Local Assistant -
```

```

There are two Dockerfiles and a docker-compose file that meets your requirements.

frontend/Dockerfile
└── Bash
language: bash frontend/Dockerfile

FROM node:18 as build

WORKDIR /app
COPY package.json .
RUN npm install --production

COPY . .

RUN npm run build

FROM node:18 as development

WORKDIR /app
COPY package.json .
RUN npm install --production

COPY --from=build /app/dist /app/dist
COPY ./src ./src
COPY ./public ./public

ENV NODE_ENV=development
ENV WATCHPACK_POLLING=true

EXPOSE 5173

CMD ["npm", "run", "start"]

backend/Dockerfile (with multi-stage build)
└── Bash
language: bash backend/Dockerfile

Base stage for building the image with dependencies
FROM python:3.11-slim as base

WORKDIR /app
COPY pyproject.toml requirements.toml .
RUN poetry install --no-root

Build stage for creating a minimal image with only the required dependencies
FROM python:3.11-slim as build

Copy built dependencies from the base stage
COPY --from=base /app/.venv /app/.venv

WORKDIR /app
COPY app/__init__.py app/main.py .
RUN poetry install --no-root

Prod stage for creating a production-ready image with all dependencies and the main script
FROM python:3.11-slim as prod

```

( Local Assistant )

```

└── Bash
language: bash backend/Dockerfile

Base stage for building the image with dependencies
FROM python:3.11-slim as base

WORKDIR /app
COPY pyproject.toml requirements.toml .
RUN poetry install --no-root

Build stage for creating a minimal image with only the required dependencies
FROM python:3.11-slim as build

Copy built dependencies from the base stage
COPY --from=base /app/.venv /app/.venv

WORKDIR /app
COPY app/__init__.py app/main.py .
RUN poetry install --no-root

Prod stage for creating a production-ready image with all dependencies and the main script
FROM python:3.11-slim as prod

WORKDIR /app
COPY --from=build /app/.venv /app/.venv
COPY --from=base /app/app/__init__.py app/main.py .
RUN poetry install --no-root

Set the working directory to /app
WORKDIR /app

Expose port 8000 for the main application
EXPOSE 8000

Optionally expose a debug port (5678)
EXPOSE 5678

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

docker-compose.yml
└── Yml
version: '3.4'

services:
 ats-frontend:
 build: ./frontend
 environment:
 - NODE_ENV=development
 - WATCHPACK_POLLING=true
 ports:
 - "5173:5173"
 volumes:
 - ./frontend:/app
 - ./frontend/node_modules:/app/node_modules (ro)
 depends_on:
 - ats-backend

```

( Local Assistant )

## 2. Deploying instruction in GCP

ChatGPT 4o ▾

we have these docker file (/docker) and .env file (in the root)  
how to deploy this in GCP?

Share  

Great — you have a solid Docker-based setup with:

- A `docker-compose.yml` for multi-service deployment (frontend + backend + Nginx)
- Separate Dockerfiles for frontend (`frontend.Dockerfile`) and backend (`backend.Dockerfile`)
- An `.env` file in the root (for secrets/config)

### Goal:

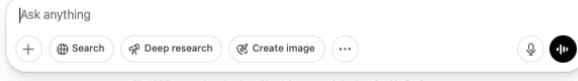
Deploy your full-stack app to Google Cloud Platform (GCP) using Docker Compose.

### Recommended Approach: Use GCP Compute Engine (VM + Docker)

### Step-by-step Plan:

#### Step 1: Prepare GCP VM Instance

1. Go to [Google Cloud Console](#).
2. Enable the [Compute Engine API](#) (if not already enabled).
3. Create a new VM instance:
  - Choose a region (e.g., `us-central1`).
  - Select an OS:  Ubuntu 22.04 LTS is recommended.
  - Allow HTTP and HTTPS traffic under [Firewall settings](#).



ChatGPT 4o ▾

### Step 1: Prepare GCP VM Instance

1. Go to [Google Cloud Console](#).
2. Enable the [Compute Engine API](#) (if not already enabled).
3. Create a new VM instance:
  - Choose a region (e.g., us-central1).
  - Select an OS:  Ubuntu 22.04 LTS is recommended.
  - Allow HTTP and HTTPS traffic under [Firewall settings](#).



### Step 2: SSH into your VM

In the Cloud Console, click "SSH" on your VM to open a terminal.

### Step 3: Install Docker & Docker Compose on the VM

```
bash ⌂ Copy ⌂ Edit

Install Docker
sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER

Install Docker Compose
sudo apt install -y docker-compose
```

Then run:

```
bash ⌂ Copy ⌂ Edit

docker --version
Ask anything
+ ⌂ Search ⌂ Deep research ⌂ Create image ...
```

ChatGPT can make mistakes. Check important info. See [Cookie Preferences](#).

ChatGPT 4o ▾

### Step 4: Upload Your Code

From your local machine:

```
bash ⌂ Copy ⌂ Edit

Zip everything except node_modules/.venv/__pycache__
zip -r ats-app.zip . -x "*/node_modules/*" "*/__pycache__/*" "*.*env"

Upload to VM (replace YOUR_VM_IP)
gcloud compute scp ats-app.zip your-vm-user@YOUR_VM_IP:~ --zone=your-vm-zone
```

Then SSH into the VM:

```
bash ⌂ Copy ⌂ Edit

unzip ats-app.zip
cd ats-app/
```

### Step 5: Set Up .env on the VM

Copy or manually create the `.env` file in your VM root directory. Don't commit secrets to GitHub.

### Step 6: Run Your App

From the root of your unzipped project:

```
bash ⌂ Copy ⌂ Edit

cd docker/
docker-compose up --build -d
```

This will:

```
Ask anything
+ ⌂ Search ⌂ Deep research ⌂ Create image ...
```

ChatGPT can make mistakes. Check important info. See [Cookie Preferences](#).

## 🚀 Step 1: Create a VM in GCP

### A. Open your terminal and login:

```
bash
gcloud auth login
```

Copy
Edit

### B. Set your GCP project:

```
bash
gcloud config set project YOUR_PROJECT_ID
```

Copy
Edit

### C. Create a Compute Engine VM:

```
bash
gcloud compute instances create ats-vm \
--zone=us-central1-a \
--machine-type=e2-medium \
--image-family=debian-11 \
--image-project=debian-cloud \
--boot-disk-size=30GB \
--tags=http-server,https-server \
--metadata startup-script="#! /bin/bash
apt-get update
apt-get install -y docker.io docker-compose
usermod -aG docker $USER
"
```

Copy
Edit

This:

- Creates a VM named `ats-vm`
- Installs Docker and Docker Compose



Ask anything

+
Search
Deep research
Create image
...


## 💻 Step 2: Install Google Cloud CLI (once)

If you haven't installed it:

### On macOS:

```
bash
brew install --cask google-cloud-sdk
```

Copy
Edit

Then run:

```
bash
gcloud init
```

Copy
Edit

It will:

- Open your browser
- Ask for login
- Let you select your project
- Set the default region

## ✓ Step 3: Create a VM with Docker

In terminal, run this (once `gcloud init` is done):

```
bash
gcloud compute instances create ats-vm \
--zone=us-central1-a \
--machine-type=e2-medium \
--image-family=debian-11 \
--image-project=debian-cloud \
--boot-disk-size=30GB \

```



Ask anything

+
Search
Deep research
Create image
...


ChatGPT can make mistakes. Check important info. See [Cookie Preferences](#).

ChatGPT 4o ▾

#### ✓ Recommended Region for You:

Since you are located in India 🇮🇳, the best option is:

```
asia-south1
```

Copy Edit

It's closer, likely has lower latency, and generally better performance for deployments from India.

#### 🚀 Let's proceed to:

### ✓ Step 5: Create the Artifact Registry Docker Repository

Run this command in your terminal:

```
bash
gcloud artifacts repositories create asap-repo \
--repository-format=docker \
--location=asia-south1 \
--description="Docker repo for ASAP project"
```

Copy Edit

Let me know once this completes — and we'll move to **Docker image build and push**.

🔗 🔍 🔍 🔍 🔍 🔍 🔍

ChatGPT 4o ▾

### ✓ Step 6.1: Authenticate Docker with GCP

Run this once to configure Docker to use your GCP credentials:

```
bash
gcloud auth configure-docker asia-south1-docker.pkg.dev
```

Copy Edit

### ✓ Step 6.2: Build the Backend Docker Image

Navigate to your project root and run:

```
bash
docker build -f docker/backend.Dockerfile -t backend-image .
```

Copy Edit

This uses the `backend.Dockerfile` and builds the image locally as `backend-image`.

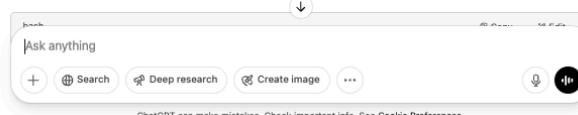
### ✓ Step 6.3: Tag the Docker Image for Artifact Registry

Replace `PROJECT_ID` below with `ats-asap-code-custodians-bbca`:

```
bash
docker tag backend-image asia-south1-docker.pkg.dev/ats-asap-code-custodians-bbca/asap-re
```

Copy Edit

### ✓ Step 6.4: Push the Docker Image to GCP



ChatGPT can make mistakes. Check important info. See [Cookie Preferences](#).

## ✓ Step 6: Push Docker Images to Artifact Registry

Let's push both the images you just built to your GCP Artifact Registry.

### ◆ Run these two commands:

bash

Copy Edit

```
Push backend image
docker push asia-south1-docker.pkg.dev/ats-asap-code-custodians-bbca/asap-repo/backend:latest

Push frontend image
docker push asia-south1-docker.pkg.dev/ats-asap-code-custodians-bbca/asap-repo/frontend:latest
```

These will upload the images to your Artifact Registry so they can be deployed to Cloud Run.

Let me know once done, then we'll proceed to the deployment steps.



Ask anything



Search

Deep research

Create image

...



ChatGPT can make mistakes. Check important info. See [Cookie Preferences](#).

## ✓ Step 8: Deploy the Backend on Cloud Run

Let's deploy the backend container first using this command:

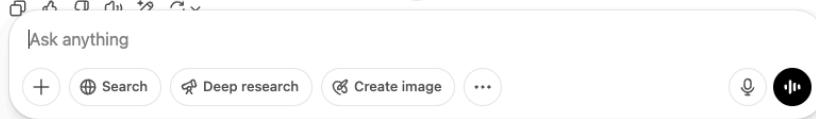
```
bash ⌂ Copy ⌂ Edit
gcloud run deploy ats-backend \
 --image=asia-south1-docker.pkg.dev/ats-asap-code-custodians-bbca/asap-repo/backend:late
 --platform=managed \
 --region=asia-south1 \
 --allow-unauthenticated \
 --port=8000
```

📌 This will:

- Deploy the `backend` service as `ats-backend`
- Use the container from Artifact Registry
- Expose it publicly via HTTPS
- Listen on port `8000`

Once it's deployed, GCP will give you a public URL — note it down so we can use it for the frontend.

Let me know once it's done. Ready? 



ChatGPT can make mistakes. Check important info. See [Cookie Preferences](#).

# READMEs

You are an expert technical writer. I want you to generate a professional README.md file for a full-stack project with the following characteristics:

## ## Overview:

- The project name is: ATS-ASAP Code Custodians
- One-liner: An internal system to match resumes to open roles and keep resume data updated.
- Features:
  - Skill-to-role matching (tech stack, experience)
  - Automated resume update reminders (6-month monitoring)
  - Resume parsing and skill extraction
  - Normalized matching algorithm

## ## Tech Stack:

### ### Frontend:

- React with TypeScript
- Tailwind CSS
- Vite

### ### Backend:

- Python
- Email notification system
- Resume parser module
- Uvicorn + FastAPI
- Testing with pytest

## ## Prerequisites:

- Node.js (v16+)
- Python 3.8+
- npm or yarn
- pip

## ## Setup Instructions:

Write a step-by-step \*\*Project Setup Guide\*\* that includes:



```
Setup Instructions:
Write a step-by-step **Project Setup Guide** that includes:

1. Creating and activating a Python virtual environment (.venv)
2. Copying and editing .env file from env-template at the root
3. Installing Python dependencies from requirements.txt
4. Starting the backend with uvicorn from src/backend
5. Navigating to the frontend in src/frontend
6. Installing node dependencies and starting the app with npm run dev:all

Testing:
Include command to run:
- backend tests with pytest

Project Structure:
Display the structure in code block format showing key backend and frontend folders, file types, and what each contains.

Contribution Guidelines:
Steps to fork, create branch, commit, push, and open pull request.

License & Support:
Mark as **proprietary and confidential** and include support contact info.

Use clear markdown formatting, avoid emojis, and maintain clean spacing and structure.
```



Sure, here's a professional `README.md` for your **ATS-ASAP Code Custodians** project:

```
markdown Copy Edit
ATS-ASAP Code Custodians
```

markdown

 Copy  Edit

## # ATS-ASAP Code Custodians

**\*\*An internal system to match resumes to open roles and keep resume data updated.\*\***

---

### ## 🌟 Overview

ATS-ASAP Code Custodians is an intelligent full-stack application designed for internal t

#### ### 🔎 Key Features

- **\*\*Skill-to-Role Matching\*\*:** Matches resumes based on tech stack and years of experience
- **\*\*Automated Resume Maintenance\*\*:** Sends reminders for updating resumes every 6 months.
- **\*\*Resume Parsing & Skill Extraction\*\*:** Extracts and normalizes skills from uploaded res
- **\*\*Intelligent Matching Algorithm\*\*:** Normalizes candidate attributes for accurate matchi

---

### ## 🚀 Tech Stack

#### ### Frontend

- React with TypeScript
- Tailwind CSS
- Vite

#### ### Backend

- Python
- FastAPI + Uvicorn
- Email Notification System
- Resume Parsing Module
- Testing with `pytest`

---



## **2. Create and Activate Python Virtual Environment**

```
bash Copy Edit

python3 -m venv .venv
source .venv/bin/activate # macOS/Linux
or
.venv\Scripts\activate # Windows
```

## **3. Copy and Configure Environment Variables**

```
bash Copy Edit

cp env-template .env
Edit the '.env' file to include actual environment values.
```

## **4. Install Python Dependencies**

```
bash Copy Edit

pip install -r requirements.txt
```

## **5. Start the Backend Server**

```
bash Copy Edit

cd src/backend
uvicorn main:app --reload
```

## **6. Start the Frontend App**

```
bash Copy Edit

cd ../../src/frontend
npm install
npm run dev:all
```



```

```

[Copy](#) [Edit](#)

##  Prerequisites

Ensure the following are installed on your system:

- [Node.js](<https://nodejs.org/>) (v16 or higher)
- [Python](<https://www.python.org/>) (3.8 or higher)
- [npm](<https://www.npmjs.com/>) or [yarn](<https://yarnpkg.com/>)
- [pip](<https://pip.pypa.io/>)

```

```

##  Project Setup Guide

### 1. Clone the Repository

```
```bash
git clone https://github.com/your-org/ats-asap-code-custodians.git
cd ats-asap-code-custodians
```

2. Create and Activate Python Virtual Environment

```
bash
```

[Copy](#) [Edit](#)

```
python3 -m venv .venv
source .venv/bin/activate    # macOS/Linux
# or
.venv\Scripts\activate       # Windows
```

3. Copy and Configure Environment Variables

```
bash
```

[Copy](#) [Edit](#)

```
cp env-template .env
# Edit the `.env` file to include actual environment values.
```

4. Install Python Dependencies

```
bash
```

[Copy](#) [Edit](#)

📁 Project Structure

```
bash
ats-asap-code-custodians/
|
└── src/
    ├── backend/
    │   ├── main.py          # FastAPI entry point
    │   ├── resume_parser/   # Resume parsing logic
    │   ├── notifications/  # Email system for reminders
    │   ├── matching/        # Skill normalization and match algorithm
    │   ├── config/          # Env config and settings
    │   └── tests/           # Pytest-based test suite
    |
    └── frontend/
        ├── src/
        │   ├── components/    # Reusable React components
        │   ├── pages/          # App views and routes
        │   ├── styles/         # Tailwind CSS styling
        │   ├── App.tsx         # Main app entry
        │   └── main.tsx        # Vite entry point
        ├── public/            # Static assets
        └── vite.config.ts     # Vite configuration
    |
    ├── .env-template        # Sample environment variables
    ├── requirements.txt      # Python dependencies
    ├── README.md             # Project documentation
    └── package.json          # Frontend scripts and dependencies
```

🤝 Contribution Guidelines

We welcome internal contributors. Please follow these steps: