**Resume Screener**

Case Study - [Resume Screener - Case Study.pptx](https://cognine1-my.sharepoint.com/personal/vinayaka_katilu_cognine_com/_layouts/15/Doc.aspx?sourcedoc=%7BFE0ABA7A-07B9-4B94-A925-EFCB20195EEA%7D&file=Resume%20Screener%20-%20Case%20Study.pptx&action=edit&mobileredirect=true)

**Problem Statement**

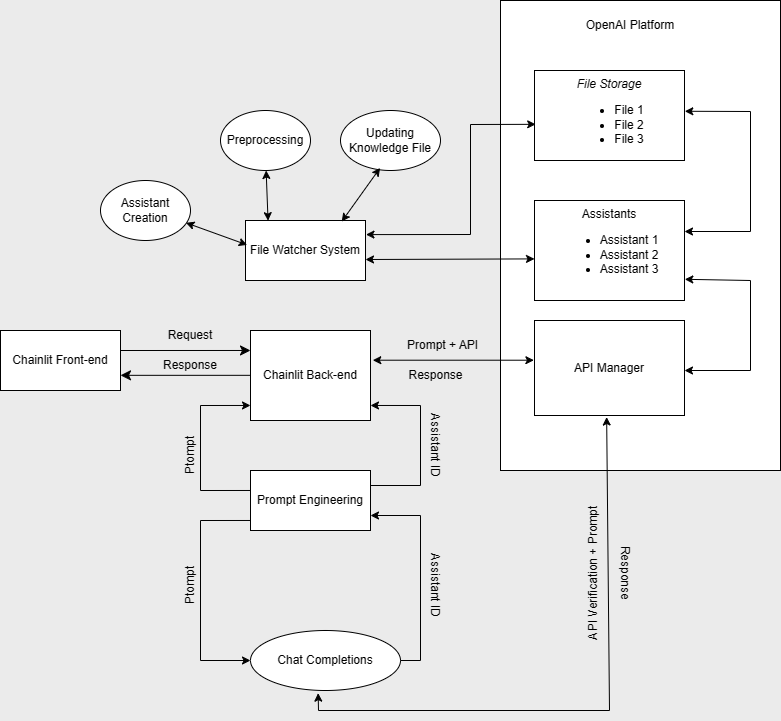
Many organizations face challenges in efficiently and effectively screening a large volume of resumes to identify qualified candidates for job positions. Manual resume screening processes are time-consuming, prone to errors, and often lead to inconsistencies in evaluation criteria. Additionally, managing resumes from various sources and formats can be challenging, requiring significant human intervention.

To address these challenges, there is a need for an AI-powered resume screening application with a file watcher system that can automate the screening process, improve accuracy, and enhance the overall efficiency of recruitment processes. This application should be able to intelligently parse resumes from different file formats, such as PDFs, Word documents, and text files, extract relevant information, and match candidates against predefined criteria.

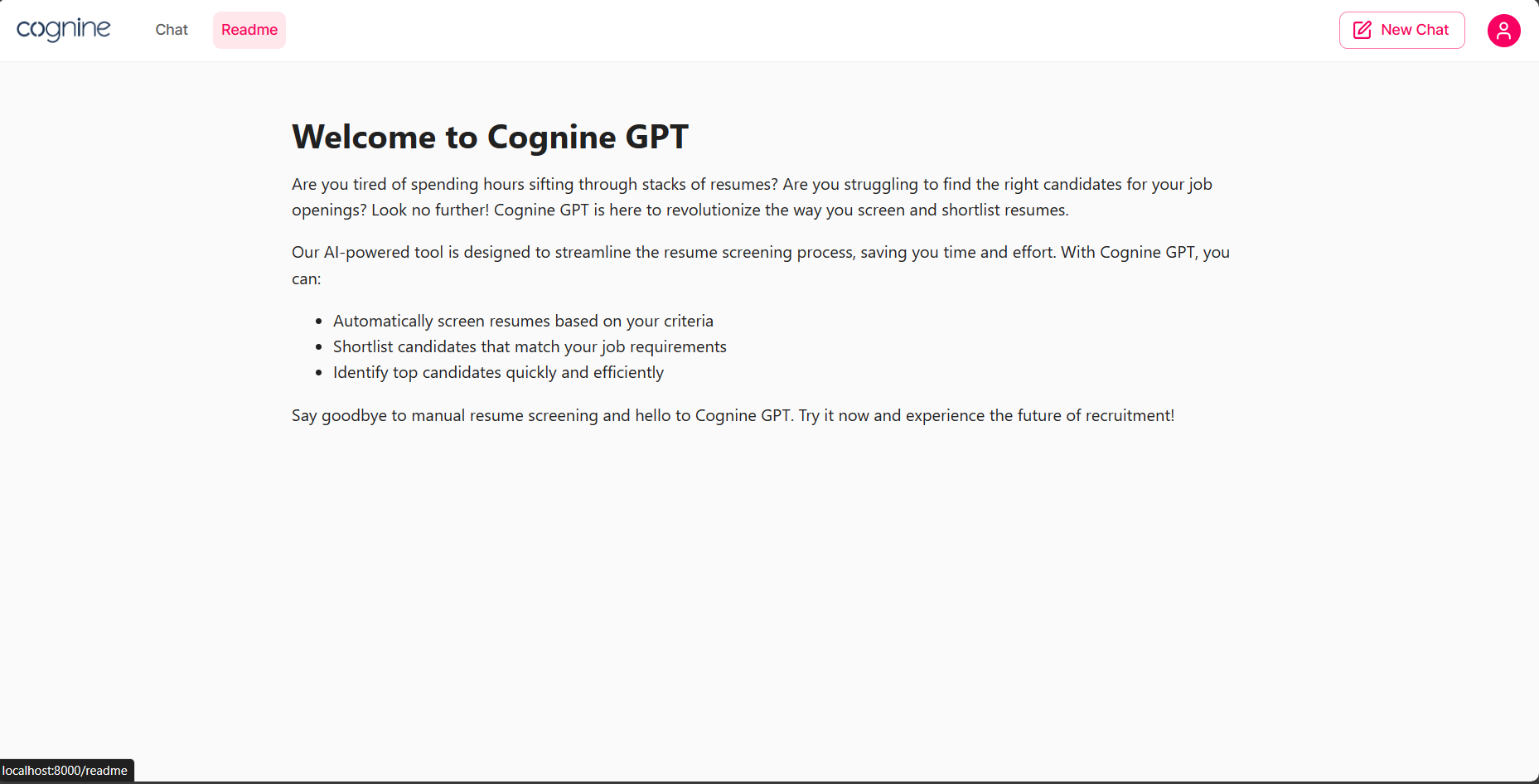
The AI resume screening application should also include a file watcher system that can monitor designated folders or email accounts for new resume submissions, automatically process incoming resumes, and update the screening results in real-time. This system should be customizable to support different job roles, industries, and company-specific requirements, ensuring that only the most qualified candidates are selected for further evaluation.

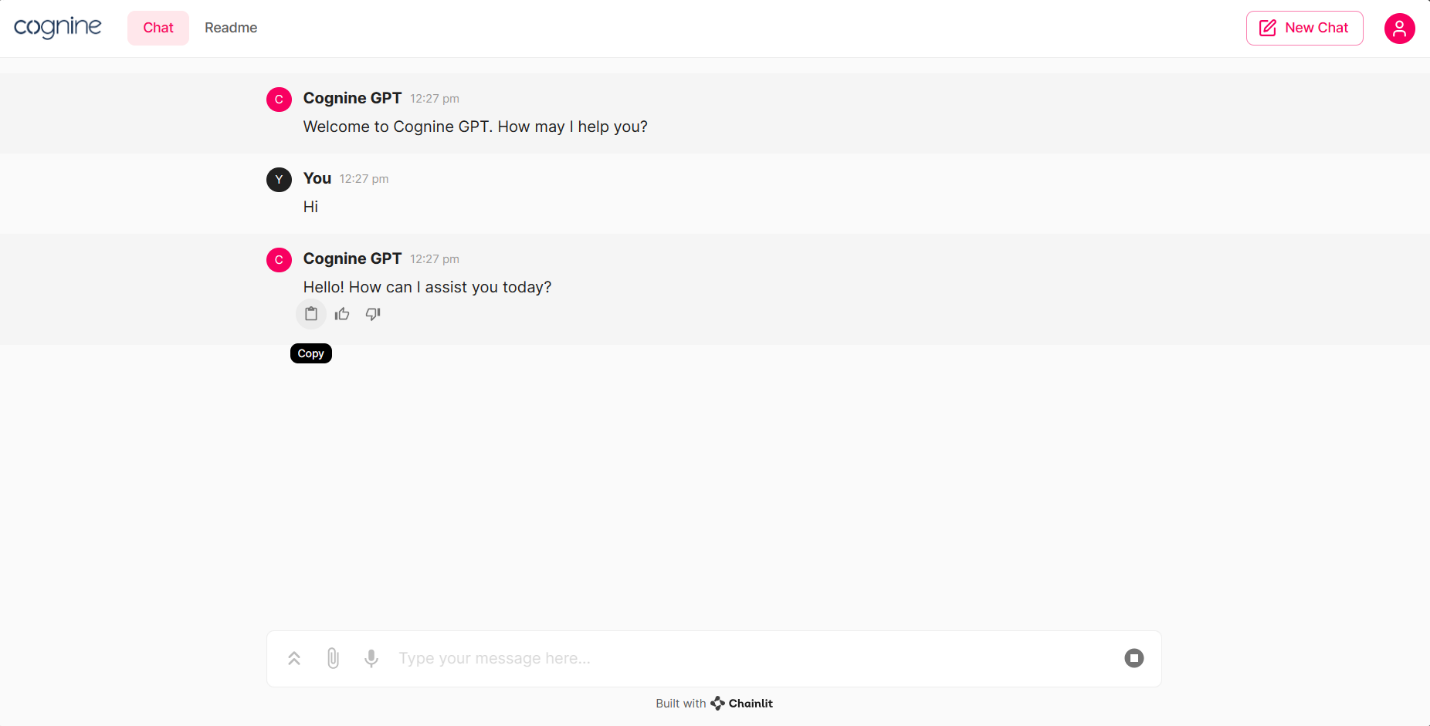
Overall, the AI resume screening application with a file watcher system aims to streamline the recruitment process, reduce manual effort, minimize bias, and improve the quality of hires for organizations.

**Proposed Architecture**



**Developed Front-end and User Interface**





**Application Workflow**

**File Watcher System**

1. **When no JSON file exists:** 
   1. When no JSON files are there in JSON folder, parseResume script will be triggered to create one JSON file per folder of the resume. Within this JSON file, resumes names and their contents are stored as key value pairs.
   2. Then createAssistant script will be triggered to create one Assistant for each of the JSON files with predefined instructions.
   3. After creating the assistants, a metadata file will be created with Assistant Names as keys and Assistant’s ID as values.
2. **When new resumes are added/ deleted:**
   1. When new resumes are added, a list with the files updated is created. Then the list will be iterated, and their respective JSON files will be updated. In case of new resumes, its JSON file will be added with a new key value pair. In case of resume being deleted, its JSON file will be accessed and the key value pair for those files will be removed.
   2. Once the JSON files are updated, updateAssistant script will be triggered to reupload the relevant Assistants with their new knowledge files.

**Chainlit Front-end**

1. Chainlit chatting UI is a flask-based application that will be hosted on a server.
2. It takes user’s prompts and displays the response as a formatted markup inside the chatting window.

**Chainlit Back-end**

1. When a user gives any prompt, the prompt will be sent to Chat Completions model that will return a predefined category. This category will be one of the names of assistants.
2. Assistant metadata file will be accessed, and Assistant’s ID will be queried based on the Assistant’s Name that Chat Completion returns.
3. Then the user’s prompt will be sent to that specific Assistant by accessing it through the Assistant ID.
4. This will be done using the Threads, Messages and Run objects. The prompt will be added to message object and the message object will be added to the Thread and Run object will be triggered based on this Thread. The Assistant will add the response back to the Thread which will be accessed through indexing. Then the actual message response will be sent back to the Chainlit Front-end to be formatted and displayed.

**Challenges**

1. **Message Context:** To make the AI remember the context of previous messages, all the system and user messages from previous prompts had to be passed along with current prompt which increased the number of input tokens.
2. **Token Limits:** OpenAI allows only 60k tokens to be used per minute and if the prompt’s input or output tokens are more than that, the application would behave abnormally.
3. **File Limit:** OpenAI only allows 20 knowledge files per assistant and 10 files in direct uploads.
4. **Retrieval File Format:** Retrieval only supports specific file formats so we cannot use tabular structures to store bulk resumes.
5. **Model Limitations:** Retrieval is only available for 3 specific models. Out of which 2 are GPT-4 Turbo. The cost of using GPT-4 Turbo is exponentially higher than that of its GPT-3.5 Turbo counterpart whose results are less accurate than them.
6. **Prompt Engineering:** User’s prompts need to be structured and formatted properly before sending them to the Assistant to get accurate results.
7. **Assistant Instructions:** Even with the costliest model, the accuracy of the results depends on the system instructions of the Assistant. Deciding on the best possible set of instructions for the model was a bit of a challenge.

**Usage**

1. Clone the repository
2. Install the requirements from the requirements.txt file

***pip install –r requirements.txt***

1. Change the necessary details inside the configuration file like API Key, Resume path, JSON path, Metadata path etc.
2. Navigate to the code directory in CLI and run the app.py application with chainlit using the below command:

***chainlit run app.py -w***

1. Open the browser and navigate to the application URL.

**Future Enhancements**

1. Use of Vector Databases instead of JSON to store knowledge files.
2. Use of Langchain for RAG and orchestration.
3. Use different models like Llama-2, Mistral, Gemini etc.
4. Use REACT.JS to build a highly customizable front-end instead of Chainlit.
5. Add User Logins.
6. Provide more User settings.