# Project Overview

**Purpose:**  
This project is a web application designed to process uploaded documents, chunk them into meaningful sections, and generate various types of content based on those sections. The application leverages OpenAI's GPT-4 model to create visual mind maps, summaries, and scenarios from the uploaded text. The generated content assists users in better understanding and analyzing their documents.

**Features:**

* **Document Upload:** Supports text (.txt) and Word (.docx) file formats.
* **Text Chunking:** Automatically splits documents into relevant sections.
* **Content Generation:** Generates visual mind maps, concise summaries, and scenario analyses for each section.
* **Interactive UI:** Built with Streamlit, providing an intuitive and responsive user interface.
* **Mermaid.js Integration:** Visualizes mind maps directly within the application.

**Technologies Used:**

* **Frontend:** Streamlit, Mermaid.js
* **Backend:** Python, LangChain
* **Language Models:** OpenAI GPT-4
* **Libraries:** LangChain, Streamlit, Pydantic

# Installation Instructions

**Prerequisites**

* **Python:** Version 3.8 or later
* **pip:** Python package installer
* **OpenAI API Key:** Obtain from [OpenAI](https://openai.com/)

**Setup Steps**

1. **Clone the Repository**
2. **Navigate to the Project Directory**
3. **Install Dependencies:**

pip install -r requirements.txt

1. **Set Up Environment Variables:**
   * Rename .env.example to .env.
   * Add your OpenAI API key in the .env file:

OPENAI\_API\_KEY=your\_openai\_api\_key

1. **Run the Application:**

streamlit run app.py

1. **Access the Application:**  
   Open your browser and navigate to http://localhost:8501.

# Usage Guide

1. **Upload a Document:**
   * Click on the "Upload" button and select a .txt or .docx file.
2. **View Chunked Sections:**
   * Once uploaded, the document is processed and divided into sections.
   * Select a section from the dropdown menu to explore further.
3. **Generate Content:**
   * Choose the type of content you want to generate for the selected section:
     + **Visual:** Generates a mind map visualization.
     + **Summary:** Provides a concise summary of the section.
     + **Scenario:** Creates scenarios based on the content.
4. **Interact with Generated Content:**
   * View the visualization directly within the app.
   * Read the summary in the provided text area.
   * Explore different scenarios and their details.
5. **Regenerate Content:**
   * If unsatisfied with the generated content, click the "Regenerate" button to create new outputs.

# Module Descriptions

## chunker.py

**Purpose:**  
The chunker.py module is responsible for processing the uploaded document by splitting it into numbered paragraphs and further chunking these paragraphs into meaningful sections based on their relevance. It utilizes OpenAI's GPT-4 model via LangChain to determine the indices and titles of relevant sections.

**Key Components:**

* **Imports:**
  + LangChain's ChatPromptTemplate, JsonOutputKeyToolsParser, and ChatOpenAI for prompt handling and interactions with the language model.
  + Custom classes Relevance and Indexes from output\_parser.py.
  + Prompts from prompts.py.
  + Python's os module for environment variable access.
* **Class: Chunker**
  + **Attributes:**
    - llm: An instance of OpenAI's GPT-4 model.
    - sections: A list to store sections (currently unused).
  + **Methods:**
    - number\_paragraphs(text): Splits the input text into paragraphs, numbers each paragraph, and rejoins them.
    - split(doc):
      * Numbers the paragraphs.
      * Binds the Indexes tool to the language model.
      * Uses a prompt to instruct the model to identify relevant sections.
      * Parses the model's output to extract section indices and titles.
      * Returns a dictionary mapping section titles to their corresponding text.

A black and white photo

Description automatically generated

## content\_generator.py

**Purpose:**  
The content\_generator.py module handles the generation of various types of content (visualizations, summaries, scenarios) based on the chunked sections of the document. It interacts with the GPT-4 model to produce the desired outputs.

**Key Components:**

* **Imports:**
  + Similar to chunker.py, utilizing LangChain components and custom classes MermaidCode, Summary, and Scenarios.
  + Prompts from prompts.py.
  + Python's os module.
* **Class: ContentGenerator**
  + **Attributes:**
    - llm: An instance of OpenAI's GPT-4 model.
    - sections: A list to store sections (currently unused).
  + **Methods:**
    - create\_visual(doc): Generates Mermaid.js code to visualize the provided text as a mind map. Retries up to 5 times if generation fails.
    - create\_summary(doc): Produces a concise summary of the provided text. Retries up to 5 times if generation fails.
    - create\_scenario(doc, prompt): Creates scenarios based on the provided text and custom prompt. Retries up to 5 times if generation fails.

A computer screen with white text

Description automatically generated

## output\_parser.py

**Purpose:**  
The output\_parser.py module defines data models using Pydantic to structure and validate the outputs from the language model. These models ensure that the generated content adheres to the expected format.

**Key Components:**

* **Imports:**
  + Pydantic's BaseModel and Field for data modeling.
  + Python's typing module for type hints.
* **Data Models:**
  + **Section:** Represents a single section with a summary, title, and text.
  + **Sections:** A list of Section instances.
  + **Relevance:** Indicates whether a piece of text is relevant.
  + **Index:** Defines the start and end indices of a section along with its title.
  + **Indexes:** A list of Index instances.
  + **MermaidCode:** Holds Mermaid.js code for visualizations.
  + **Summary:** Contains the summary text.
  + **Scenario:** Represents a single scenario with a name and details.
  + **Scenarios:** A list of Scenario instances.

## prompts.py

**Purpose:**  
The prompts.py module contains predefined prompt templates used to instruct the language model on the specific tasks it needs to perform, such as chunking text, generating visuals, summaries, and scenarios.

**Key Components:**

* **Prompts:**
  + **CHUNKER\_PROMPT:** Instructs the model to chunk text into sections with titles and indices.
  + **VISUAL\_PROMPT:** Guides the model to create Mermaid.js mind map code from the text.
  + **SUMMARY\_PROMPT:** Directs the model to generate a concise summary of the provided context.
  + **SCENARIO\_PROMPT:** Asks the model to generate distinct scenarios related to the content.

**Usage:**  
Each prompt is used within the respective methods in chunker.py and content\_generator.py to structure the input to the language model.

## app.py

**Purpose:**  
The app.py file is the main entry point of the web application, built using Streamlit. It orchestrates the user interface, handles user interactions, and integrates the functionalities provided by other modules to deliver a seamless experience.

**Key Components:**

* **Imports:**
  + Streamlit for UI components.
  + Custom classes Chunker and ContentGenerator from their respective modules.
  + Streamlit's components.v1 for embedding HTML content.
  + Prompts from prompts.py.
* **UI Configuration:**
  + Sets up the page layout to be wide.
  + Injects custom CSS to style the upload button and center elements.
* **Mermaid.js Integration:**
  + Defines a function create\_mermaid\_code(code) that wraps Mermaid.js code within HTML to render visualizations.
* **Main Workflow:**
  + **File Upload:**
    - Users upload a .txt or .docx file.
  + **Processing Uploaded File:**
    - Reads the file content.
    - Utilizes the Chunker class to split the document into sections.
    - Stores the sections in Streamlit's session state for persistence.
  + **Section Selection and Content Generation:**
    - Users select a section from a dropdown.
    - Choose the type of content to generate: Visual, Summary, or Scenario.
    - Generates and displays the selected content.
    - Provides options to regenerate content as needed.
* **Session State Management:**
  + Utilizes Streamlit's st.session\_state to store processed sections and generated content, ensuring data persists across interactions.

# Evaluation

## eval\_generator.py

**File Path:** evaluation/eval\_generator.py

**Purpose:**  
The eval\_generator.py module is responsible for generating keywords from the document sections and evaluating the generated mind maps based on these keywords. It uses OpenAI's GPT-4 model via LangChain to extract keywords and perform evaluations, ensuring that the generated content aligns with the key themes of the document.

### Detailed Breakdown

**Class: EvalGenerator**

* Initializes the GPT-4 model (ensure the model name is corrected to "gpt-4").
* Initializes an empty list for sections (currently unused). A computer screen with text

  Description automatically generated

**Method: generate\_keywords(self, doc)**

A computer screen shot of a program

Description automatically generated

* Binds the Keywords tool to the language model.
* Uses the KEYWORDS\_PROMPT to instruct the model.
* Parses the output to extract keywords.
* Retries up to 5 times in case of failures.

**Method: evaluation\_mindmap(self, mindmap, keywords)**

A computer screen shot of a program code

Description automatically generated

* Binds the Evaluation tool to the language model.
* Uses the EVALUATE\_MINDMAP prompt to assess the mind map.
* Parses the output to extract evaluation scores and comments.
* Retries up to 5 times in case of failures.

A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

## eval\_script.py

**File Path:** evaluation/eval\_script.py

**Purpose:**  
The eval\_script.py is an executable script designed to automate the generation of keywords, mind maps, summaries, and evaluate the generated content for a set of documents. It reads input data from a CSV file, processes each document, and outputs the results alongside evaluation scores and comments into a new CSV file.

**Function: evaluate\_summary(text, summary)**

A screen shot of a computer

Description automatically generated

* Evaluates the quality of summaries using deepeval.

**Function: import\_and\_generate(input\_file, output\_file)**

* Reads an input CSV file containing a text column.
* Generates keywords, mind maps, and summaries for each document.
* Saves the results to a new CSV file.

**Function: evaluate(input\_file, output\_file)**

* Reads the processed CSV file (test\_new.csv).
* Evaluates the generated mind maps and summaries.
* Appends evaluation scores and comments to the DataFrame.
* Saves the final evaluation results to evaluation.csv.

**Main Execution:**

**A computer screen with text on it

Description automatically generated**

* Checks if the processed CSV (test\_new.csv) exists. If not, it generates the required fields by reading test.csv.
* Performs evaluations and saves the results to evaluation.csv.

**Execution Command:**

A black rectangular object with green lines

Description automatically generated