Gatekeeper

Table of Contents

[Introduction 2](#_Toc193897534)

[Core Functionality 2](#_Toc193897535)

[Key Components 2](#_Toc193897536)

[Workflow 4](#_Toc193897537)

[Key Components 4](#_Toc193897538)

[FastAPI Endpoint 5](#_Toc193897539)

[How to Run 5](#_Toc193897540)

[Conclusion 6](#_Toc193897541)

### Introduction

This Gatekeeper application processes and classifies files such as emails, PDFs, and images. It uses machine learning models and external APIs to classify the content of these files into predefined request types and sub-request types. The application also checks for duplicate requests using a Faiss index.

It leverages a combination of techniques, including:

* Keyword and Cosine Similarity: A baseline approach using keyword matching and vector similarity.
* Large Language Models (LLMs): Integration with both Gemini and OpenAI (GPT) for more advanced semantic analysis.
* Faiss Indexing: For efficient duplicate detection.

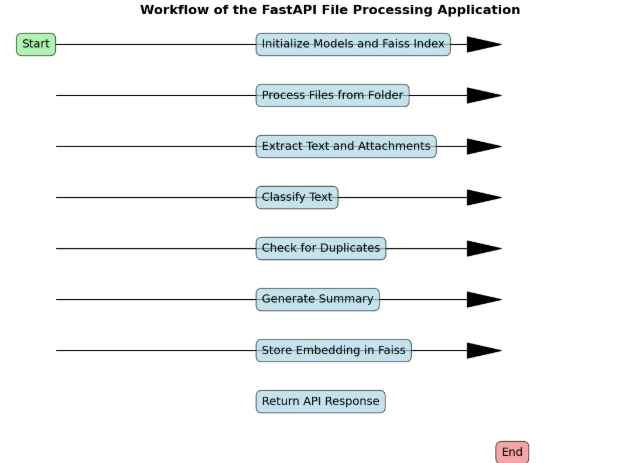
### Core Functionality

1. File Processing: The program can process different file types:
   * Emails (.eml): Extracts text and attachments. Handles text/plain, text/html, and image attachments, including OCR for images.
   * PDFs (.pdf): Extracts text content.
   * Images (.jpg, .png, etc.): Extracts text using OCR (Tesseract).
2. Text Classification: The extracted text is classified into request types and sub-request types. It uses a hybrid approach:
   * Baseline keyword and cosine similarity.
   * Classification using Gemini.
   * Classification using OpenAI.
   * Results from these methods are combined, and the most confident classification is selected.
3. Duplicate Detection: The program uses Faiss to maintain an index of processed text embeddings. This allows it to identify duplicate requests with high accuracy.
4. API Endpoint: The program exposes a FastAPI endpoint (/process\_file/) to upload files and receive classification results.

### Key Components

1. Libraries:
   * os: For file system operations.
   * email: For parsing email files.
   * quopri: For decoding quoted-printable email content.
   * PyPDF2: For extracting text from PDF files.
   * transformers: For loading pre-trained language models (Sentence Transformers).
   * pydantic: For data validation (e.g., ClassificationResult model).
   * BeautifulSoup: For parsing HTML content in emails.
   * torch: For tensor operations with language models.
   * PIL (Pillow): For image processing.
   * pytesseract: For OCR (text extraction from images).
   * sklearn.metrics.pairwise.cosine\_similarity: For calculating text similarity.
   * FastAPI: For creating the web API.
   * faiss: For similarity search and duplicate detection.
   * logging: For logging events.
   * requests: For making API calls to Gemini and OpenAI.
2. Models:
   * ClassificationResult (Pydantic model): Defines the structure of the output data, including file information, classification results, status, and messages.
3. Functions:
   * extract\_text\_from\_email(email\_path): Extracts text and attachments from an email file.
   * extract\_text\_from\_pdf(pdf\_path): Extracts text from a PDF file.
   * extract\_text\_from\_image(image\_path): Extracts text from an image using OCR.
   * get\_embedding(text): Computes the text embedding using the Sentence Transformer model.
   * is\_duplicate\_request(text, threshold): Checks if a text is a duplicate of previously processed requests.
   * classify\_text(text): Classifies the text into request types and sub-request types.
   * classify\_with\_gemini(text): Classifies text using the Gemini API.
   * classify\_with\_openai(text): Classifies text using the OpenAI API.
   * process\_file(file, request\_type, sub\_request\_type, summary): Handles file uploads, extracts text, classifies it, checks for duplicates, and returns the result.
   * initialize\_faiss(): Initializes the Faiss index for duplicate detection.

### Workflow



### Key Components

1. Initialize Models and Faiss Index:
   * Load pre-trained models for embedding generation and image classification.
   * Initialize or load the Faiss index for storing text embeddings and checking duplicates.
2. Process Files from Folder:
   * The application processes files from a specified folder.
   * It supports **.eml**, **.pdf**, **.jpeg**, and **.jpg** file types.
3. Extract Text and Attachments:
   * Extract text and attachments from email files.
   * Extract text from PDF files.
   * Extract text from images using OCR.
4. Classify Text:
   * Classify the extracted text using a combination of keyword matching, cosine similarity, and external APIs (Gemini and OpenAI).
   * Combine the results from different classification methods to determine the final request type and sub-request type.
5. Check for Duplicates:
   * Use the Faiss index to check if the text is a duplicate of any previously processed text.
   * If a duplicate is found, mark the request as duplicate and skip further processing.
6. Generate Summary:
   * Generate a summary of the text using a summarization pipeline.
7. Store Embedding in Faiss:
   * Store the text embedding in the Faiss index for future duplicate checks.
8. Return API Response:
   * Return the classification results as a JSON response through the FastAPI endpoint.

### FastAPI Endpoint

1. /process\_file/:
   * Method: POST
   * Parameters:
     1. file (File): The file to process.
     2. request\_type (Form): The request type (can be passed from the user, or determined by the program).
     3. sub\_request\_type (Form): The sub-request type (can be passed from the user, or determined by the program).
     4. summary (Form): A summary of the file content.
   * Returns: A JSON response containing the ClassificationResult.

### How to Run

1. Install Dependencies: Install the required Python packages using pip install -r requirements.txt. You'll need:
2. pip install fastapi uvicorn python-multipart requests PyPDF2 transformers torch Pillow pytesseract faiss-cpu scikit-learn

You will also need to install Tesseract OCR separately. See: <https://tesseract-ocr.github.io/tessdata/>

Also, set the environment variables GEMINI\_API\_KEY and OPENAI\_API\_KEY.

1. Run the Application: Run the FastAPI application using:
2. uvicorn main:app --reload
3. Use the API: You can use a tool like curl or a REST client (e.g., Postman) to send a POST request to the /process\_file/ endpoint with a file.

### Conclusion

This project demonstrates the integration of various technologies and frameworks to build a robust file processing and classification system. It leverages machine learning models and external APIs to provide accurate and efficient classification of text data. The use of FastAPI ensures high performance and ease of use, making it suitable for deployment in production environments. For more details or customization, refer to the codebase and associated documentation.