Extractor.py (Backend)

Major Libraries Used:

- PIL (Python Imaging Library): For image processing.
- easyOCR: For text detection and extraction from images.
- fitz (PyMuPDF): For extracting text from PDFs.
- io: For handling input and output streams, especially for images and PDFs.

Overview:

The `Extractor.py` file is designed to handle text extraction from PDFs and images of input files. It uses different methods depending on the content type of the file.

Key Functions:

1.extract_text_from_pdf(pdf_path):

- This function extracts text from PDFs. It determines whether the PDF contains selectable text or is a scanned image.
- If the PDF contains selectable text, fitz is used to extract the text.
- If the PDF contains scanned images or non-selectable text, easyOCR is used to extract the text. This decision is based on whether the text extracted from the PDF is empty or consists only of whitespace characters.
- In case of unsupported file types, an appropriate error message is returned.

2. extract_text_from_image(image_data):

- This function uses easyOCR to detect and extract text from images. It processes the provided image data and returns the extracted text. easyOCR supports multiple languages, but we have limited the tool to English due to issues with recognition accuracy in languages such as Hindi (unrecognized or misencoded text).

3. extract_text_auto(file_path):

- This function is a wrapper that selects the appropriate text extraction method based on the file extension. It calls either extract_text_from_pdf or extract_text_from_image` depending on the type of file provided.

Reusability:

- The extract_text_from_image function is reused when handling PDFs that contain scanned images or non-selectable text, avoiding the need to write redundant code.

app.py

Major Libraries Used:

- Flask: Used to create the backend API.
- Flask-CORS: Used to manage cross-origin requests from the frontend.

Overview:

The app.py file acts as the backend server, facilitating communication between the frontend (built with HTML, CSS, and JS) and the backend for text extraction tasks. Flask routes are defined to handle requests from the frontend and invoke the corresponding functions from `Extractor.py`.

Frontend Part:

- The frontend is built using HTML, CSS, and JavaScript, which handle the structure, styling, and logic of the application.
- It also checks for supported file types and sends the file to the backend for text extraction.

Testing:

The model was initially tested on multiple images and PDFs with the intention to support both Hindi and English. However, challenges arose with Hindi text (e.g., text being unrecognized or misencoded), so support was limited to English for now.

Possible Enhancements:

- Support for More Languages: Future versions can add support for more languages, such as Hindi and Spanish, as recognition accuracy improves.
- Text Correction: Implement additional processing to correct misrecognized text (e.g., spell-checking or grammar correction).
- Improved Error Handling**: Enhance error handling for unsupported or corrupted files, providing more informative messages to users.