PDF Outliner – Adobe Challenge Round 1A

This document outlines the solution developed for Adobe's Round 1A: Connecting the Dots Challenge. The core objective is to process PDF documents, extract a structured outline comprising the document's title, headings (H1, H2, H3), and their corresponding page numbers, and then save this information in a well-formatted JSON file. The solution is designed to be easily deployable and executable using Docker.

Directory Structure

The project's directory structure is organized as follows:

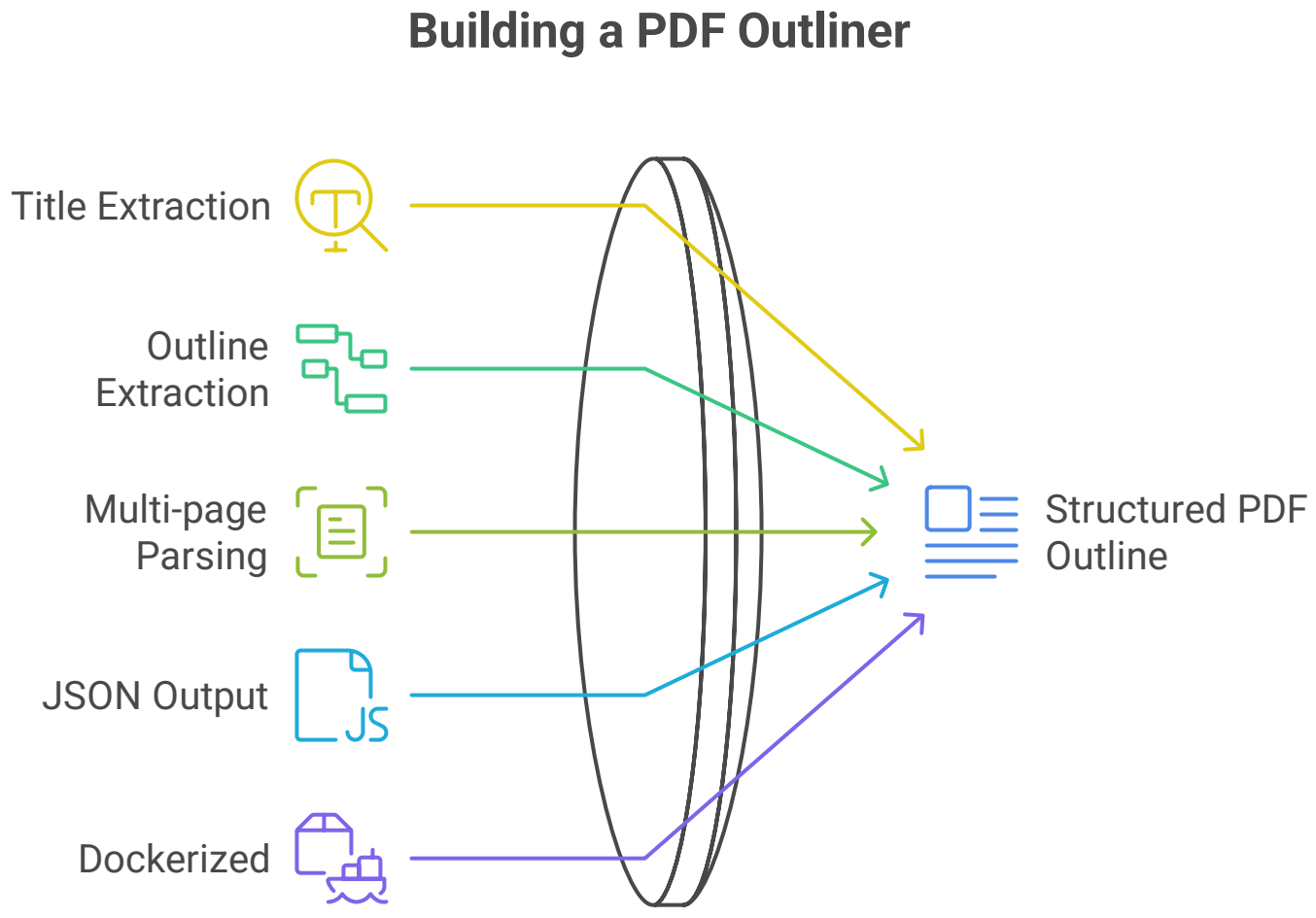
```
adobe_1a/
│
├── Dockerfile                  # Docker configuration for the project
├── final_pdf_processor.py     # Core logic to extract titles and outlines from
PDF
├── requirements.txt           # Python dependencies
├── run.py                     # Main script to process input PDFs
├── input_pdfs/                # Folder containing input PDF files
└── output_jsons/              # Output folder with extracted outline JSON
files
```

Each component plays a specific role in the overall functionality of the application.

⚙ Features

The PDF Outliner boasts the following key features:

- **Title Extraction**: Employs font size analysis and visual positioning to identify the most prominent title within the PDF document. The algorithm prioritizes larger font sizes and the text block located at the top of the first page.
- **Outline Extraction**: Detects headings (H1, H2, H3) by analyzing font sizes across all pages of the PDF. This allows for a comprehensive identification of the document's hierarchical structure.
- **Multi-page Parsing**: Scans every page of the PDF to ensure complete and accurate outline extraction, accommodating documents of varying lengths and complexities.
- **JSON Output**: Stores the extracted title and outline in a clean, structured JSON format, facilitating easy integration with other applications and systems.
- **Dockerized**: The entire application is containerized using Docker, ensuring platform-independent execution and simplified deployment.



Docker Setup (Recommended)

Using Docker is the recommended approach for running the PDF Outliner, as it ensures consistency and simplifies deployment across different environments.

Build the Docker Image

To build the Docker image, execute the following command in the project's root directory:

```
docker build -t pdf-outliner:v1 .
```

This command creates a Docker image named `pdf-outliner` with the tag `v1`, using the `Dockerfile` in the current directory.

▶ Run the Container

To run the Docker container and process the PDF files, use the following command:

```
docker run --rm \
  -v "$(pwd)/input_pdfs:/app/input_pdfs" \
  -v "$(pwd)/output_jsons:/app/output_jsons" \
  pdf-outliner:v1
```

This command performs the following actions:

- `--rm` : Automatically removes the container after it exits.
- `-v "$(pwd)/input_pdfs:/app/input_pdfs"` : Mounts the `input_pdfs` directory on your host machine to the `/app/input_pdfs` directory inside the container. This allows the container to access the PDF files you want to process.
- `-v "$(pwd)/output_jsons:/app/output_jsons"` : Mounts the `output_jsons` directory on your host machine to the `/app/output_jsons` directory inside the container. This allows the container to save the extracted outline JSON files to your host machine.
- `pdf-outliner:v1` : Specifies the Docker image to use for running the container.

This setup will process all PDF files located within the `input_pdfs/` directory and save the corresponding output `.json` files into the `output_jsons/` directory.

How It Works

The core logic of the PDF Outliner resides within the `final_pdf_processor.py` file.

`final_pdf_processor.py`

This script performs the following steps:

- **PDF Loading**: Opens each PDF document using the `PyMuPDF` (`fitz`) library.
- **Font Size Analysis**: Collects all font sizes present in the PDF to establish a basis for identifying heading hierarchies.
- **Title Detection**:
    - Identifies the largest font size used in the document.
    - Selects the text block with the largest font size that is located at the top of the first page as the document's title.
- **Heading Classification**:
    - Classifies text blocks into H1, H2, or H3 headings based on their font size relative to other text in the document. The frequency of each font size is also considered.
    - Records the page number and exact text content of each identified heading.
- **JSON Output**: Constructs a JSON object containing the extracted title and outline, formatted as follows:

```
{
  "title": "Document Title",
  "outline": [
    {
      "level": "H1",
      "text": "Section 1",
      "page": 1
    },
    {
      "level": "H2",
      "text": "Subsection 1.1",
      "page": 2
    }
  ]
}
```

`run.py`

The `run.py` script orchestrates the PDF processing workflow.

- **Directory Scanning**: Scans the `input_pdfs` directory for PDF files.
- **PDF Processing**: For each PDF file found:
    - Invokes the `process_pdf()` function from `final_pdf_processor.py` to extract the title and outline.
    - Saves the resulting JSON output to the `output_jsons` directory, using the same base name as the input PDF file.

Requirements

The project's dependencies are listed in the `requirements.txt` file:

```
PyMuPDF
PyPDF2
pdfminer.six
```

To install these dependencies manually (if not using Docker), execute the following command:

```
pip install -r requirements.txt
```

Tips

- For optimal results, use high-quality PDFs with searchable text (i.e., not scanned images).
- If headings are not being detected correctly, consider adjusting the font size threshold logic within the `final_pdf_processor.py` script.

Sample Input/Output

- Sample PDF files are provided in the `input_pdfs/` directory.
- The extracted JSON output for each PDF will be saved in the `output_jsons/` directory, with a `.json` extension.

Future Improvements

- **Multilingual Support**: Enhance the application to support PDFs in multiple languages.
- **Semantic Analysis**: Train a machine learning model to identify sections semantically, improving the accuracy of heading detection.
- **GUI/REST API**: Develop a graphical user interface (GUI) or a REST API to facilitate PDF uploads and processing.