Task Sheet: PDF to Structured Markdown Convertor

Document Information

Company:	Textro AI
Version:	0.2
Date:	May 1, 2025

Project Overview

The goal of this project is to develop a system that converts PDF documents into structured Markdown (`.md`) format. The system will extract relevant text from PDF files, eliminate unnecessary content, and produce a clean Markdown document, maintaining logical structure such as headings, subheadings, and bullet points.

Phased Approach

- **Phase 1:** Development of the standalone PDF to Markdown conversion system.
- **Phase 2:** Integration of the conversion logic into a Flask-based REST API.

Phase 1: Conversion System Design

Pipeline Design

The system must be designed as a modular pipeline with the following stages:

1. Input handling
2. Text extraction
3. Preprocessing and cleaning
4. Structuring and formatting
5. Output writing

Project Requirements

1. **PDF Input:**
    a. The system must accept standard PDF files either through file upload or by specifying a file path.
2. **Text Extraction:**
    a. Extract readable and clean text from the PDF document.
    b. Ensure the hierarchy and context of the document is preserved (e.g., titles, subheadings, bullet points).
3. **Information Filtering:**
    a. The system must remove the following elements from the document:
        i. Page numbers

   ii. Headers and footers
   iii. Irrelevant metadata
   iv. Repeated template sections

4. **Structured Formatting:**
   a. The output must be formatted in `Markdown` (`.md`) format.
   b. The system should use appropriate Markdown syntax, including:
      i. `#`, `#`, `#` for headings and subheadings
      ii. `-` or `*` for bullet points
      iii. `bold` for key terms (optional)

5. **Extensibility:**
   a. The system should be designed to be extensible, allowing for handling of different PDF formats and layouts in future iterations.

Deliverables (Phase 1)

1. A fully functional Python codebase for PDF to Markdown conversion.
2. A sample PDF input document and the corresponding Markdown output.
3. A system architecture diagram illustrating the conversion pipeline.

Phase 2: API Integration using Flask

In the second phase, the core PDF-to-Markdown conversion logic developed in Phase 1 will be exposed via a Flask-based REST API. A simple HTML/CSS frontend will be built to interact with the API, and Postman will be used for API testing and validation.

API Requirements

1. **Framework:**
   - **a.** Use the Flask web framework (Python) to develop a RESTful API.
2. **Endpoints:**
   - **a.** `POST /convert`
     Accepts a PDF file (multipart/form-data), runs the conversion pipeline, and returns the Markdown text.
   - **b.** `GET /health`
     A simple health check endpoint that returns service status.
3. **Error Handling:**
   - **a.** Return appropriate HTTP status codes (e.g., `400`, `500`) for invalid inputs or server errors.
   - **b.** JSON responses must include a `success` flag and a descriptive `message`.
   - **c.** Example error response:

```
{
    "success": false,
    "message": "Invalid file format. Please upload a PDF."
}
```

4. **Frontend:**
   - **a.** A minimal frontend built using HTML and CSS for uploading PDF files and displaying Markdown output.
   - **b.** Use JavaScript (optional) for asynchronous form submission and displaying results.
5. **API Testing:**
   - **a.** Use Postman to test API endpoints.
   - **b.** Verify correct behavior with valid and invalid inputs.

c. Save test cases in a shared collection for team use.

6. **Security:**

   a. Validate uploaded files (e.g., check MIME type and extension).

   b. Limit file size to prevent abuse (e.g., 5MB max).

Deliverables (Phase 2)

1. A REST API implemented using Flask for PDF-to-Markdown conversion.
2. HTML/CSS frontend for user interaction with the API.
3. Postman test collection with example API requests and test results.
4. API documentation (e.g., Swagger or Markdown file).