**Elaborazione di un Libro Tecnico in Corso di Autoistruzione**

**Descrizione del Problema**

Si richiede di sviluppare un sistema automatizzato che elabori un libro tecnico in formato PDF per trasformarlo in un corso di autoistruzione. Il sistema deve:

1. estrarre il contenuto,
2. estrarre le immagini rilevanti,
3. estrarre gli eventuali listati,
4. se il testo non fosse in italiano, tradurlo in italiano,
5. produrre una sintesi del testo,
6. produrre l’output.

Un libro tecnico ha tipicamente un contenuto di centinaia di pagine. Se si vuole trasformare efficacemente in un corso di autoistruzione, conviene pensare di suddividere il testo; ad esempio trasformare ogni capitolo del libro in un modulo del corso finale. Inoltre riassumere un contenuto fornisce risultati migliori se il volume dei dati da riassumere è piccolo.

Per il punto 6 si considera la produzione di un formato powerpoint (che, successivamente, sarà trasformato con strumenti già disponibili, in formato SCORM).

**Specifiche Funzionali**

1. **Input**:
   * Un file PDF contenente il libro tecnico
2. **Output**:
   * Uno o più file di testo (markdown)
   * Una directory Figures contenente le immagini tecniche rilevanti (se presenti)
   * Una directory Listing (se presenti)

Struttura dell’output:

Titolo/

├── capitolo\_1

├───── parte\_1

| ├──────── parte\_1.md # Riassunto del modulo

├───── parte\_2

├──────── parte\_2.md # Riassunto del modulo

├── capitolo\_2

├───── capitolo\_2.md # Riassunto del modulo

├── …

├── report.txt # Report dettagliato con statistiche

└── figures/ # Figure estratte

├── figura\_1.1.png # dove 1.1 è il numero della figura letto dalla caption nel testo

├── figura\_2.1.png

└── ...

1. **Requisiti per le sezioni didattiche**:
   * Ogni capitolo deve essere suddiviso in n sezioni cui corrisponderà un file di output
   * Ogni sezione produrrà in output una slide
   * Ogni sezione dovrà contenere tra 100 e 300 parole
2. **Requisiti per le immagini**:
   * Le immagini contenute nel testo devono essere estratte in formato png

**Implementazione**

**1. Struttura del Progetto**

src

├── main.py # Script principale

├── requirements.txt # Dipendenze

├── config.py # Configurazioni

├── utils/

│ ├── \_\_init\_\_.py

│ ├── pdf\_extractor.py # Estrazione di testo e immagini dal PDF

│ ├── llm\_client.py # Traduzione del testo

│ ├── output\_generator.py # Elaborazione e suddivisione del contenuto

└── output/ # Directory per i file di output

**2. Dipendenze (requirements.txt)**

# Estrazione PDF

pdfplumber>=0.9.0

# Estrazione immagini

PyMuPDF>=1.23.0

# Comunicazione HTTP con Ollama

requests>=2.31.0

# Opzionali per funzionalità avanzate

# pdf2image>=3.1.0 # Fallback per estrazione immagini

# Pillow>=10.0.0 # Manipolazione immagini

**3. Implementazione dei Moduli**

**3.1. Script Principale (main.py)**

#!/usr/bin/env python3

import os

import argparse

import logging

from config import Config

from utils.pdf\_extractor import PDFExtractor

from utils.llm\_client import LLMClient

from utils.output\_generator import OutputGenerator

class PDFProcessor:

"""

coordina la logica del processo

invoca PDFExtractor, LLMClient e OutputGenerator

"""

def main():

"""Funzione principale del programma."""

*# Setup*

logger = setup\_logging()

args = parse\_arguments()

config = Config()

processor = PDFProcessor()

results = processor.process\_pdf(args.pdf\_path, args.output)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**3.2. Script pdf\_extractor.py.py**

#!/usr/bin/env python3

"""

Elabora un PDF

1. Estrae il testo utilizzando pdfplumber
2. Estrae le figure con PyMuPDF fitz
3. Identifica i capitoli e i moduli

"""

class PDFExtractor:

**3.3. Script llm\_client.py.py**

#!/usr/bin/env python3

"""

Elabora la traduzione e la sintesi del testo

1. Si collega all’LLM locale tramite Ollama
2. Genera il riassunto del testo secondo l’indicazione del prompt inviato

"""

class LLMClient:

**3.2. Configurazione (config.py)**

class Config:

"""Classe per la configurazione del sistema."""

**4. Esecuzione del Programma**

Per eseguire il programma, utilizzare il seguente comando:

python main.py /percorso/al/libro.pdf --output-dir output

**5. Note Aggiuntive**

1. **Traduzione e sintesi**: la traduzione e la sintesi utilizzano un LLM locale acceduto tramite Ollama.
2. **Estrazione delle immagini**: L'estrazione delle immagini dal PDF è un processo complesso. Partiamo con un approccio semplice basato su fitz.
3. **Suddivisione del testo**: L'algoritmo di suddivisione del testo può essere implementato a più livelli di complessità. Il semplice conteggio delle parole potrebbe non produrre sezioni semanticamente coerenti. Potrebbe essere necessario un approccio più sofisticato basato sull'analisi semantica.
4. **Gestione degli errori**: Il codice deve prevedere una gestione degli errori di base.

**6. Documentazione Librerie**

- [pdfplumber](https://github.com/jsvine/pdfplumber) - Estrazione testo

- [PyMuPDF](https://pymupdf.readthedocs.io/) - Manipolazione PDF

- [Ollama](https://ollama.ai/docs) - LLM locale

- [requests](https://docs.python-requests.org/) - HTTP client

**File markdown**

Un **file Markdown** (estensione .md) è un file di testo che utilizza una **sintassi semplice** per formattare il contenuto in modo leggibile sia in forma "grezza" (testo puro) sia quando viene convertito in HTML o altri formati.

È usato per scrivere documenti strutturati (come README, documentazione, note) in modo **rapido e leggibile** senza bisogno di un editor complesso.

**Caratteristiche principali:**

* **Estensione**: .md
* **Sintassi leggera**: facilmente leggibile anche senza essere convertito

**Esempi di sintassi Markdown:**

| **Elemento** | **Sintassi Markdown** | **Risultato** |
| --- | --- | --- |
| Titolo | # Titolo | Titolo |
| Grassetto | \*\*testo\*\* | **testo** |
| Corsivo | \*testo\* | *testo* |
| Elenco puntato | - voce | • voce |
| Elenco numerato | 1. voce | 1. voce |
| Link | [nome](url) | [nome](https://esempio.it/) |
| Immagine | ![alt](url\_immagine) |  |
| Blocco di codice | ```linguaggio ... ``` | codice |
| Citazione | > testo | > testo |

**User Story per il Progetto**

Di seguito sono riportate le user story, categorizzate per epic, per guidare lo sviluppo del sistema.

**Epic: Estrazione Contenuto PDF**

* **User Story:** Come utente, voglio che il sistema estragga tutto il contenuto testuale da un libro tecnico in formato PDF, in modo che possa essere elaborato per la traduzione e la sintesi.
  + **Criteri di Accettazione:**
    - Dato un file PDF, il sistema estrae con successo tutto il testo leggibile.
    - Il testo estratto è reso disponibile per i passaggi di elaborazione successivi.
* **User Story:** Come utente, voglio che il sistema estragga le immagini rilevanti dal PDF, in modo che possano essere incluse nel corso di autoistruzione.
  + **Criteri di Accettazione:**
    - Dato un file PDF, il sistema identifica ed estrae le immagini.
    - Le immagini vengono salvate in formato PNG in una directory dedicata figures/.
    - Le immagini sono nominate in base al numero della figura letto dalla didascalia (es. figura\_1.1.png).
* **User Story:** Come utente, voglio che il sistema estragga gli eventuali listati di codice dal PDF, in modo che possano essere presentati come parte del corso.
  + **Criteri di Accettazione:**
    - Dato un file PDF, il sistema identifica ed estrae i listati di codice.
    - I listati estratti vengono salvati in una directory dedicata Listing/.

**Epic: Elaborazione e Trasformazione del Contenuto**

* **User Story:** Come utente, voglio che il sistema traduca il testo estratto in italiano se non è già in italiano, in modo che il corso sia largamente accessibile in Italia
  + **Criteri di Accettazione:**
    - Dato un testo in una lingua non italiana, il sistema lo traduce in italiano utilizzando la LLM locale tramite Ollama.
    - Il testo tradotto mantiene il significato e il contesto originali.
* **User Story:** Come utente, voglio che il sistema produca una sintesi del testo, in modo da poter creare moduli del corso concisi e mirati.
  + **Criteri di Accettazione:**
    - Data una sezione di testo, il sistema produce una sintesi.
    - Le sintesi sono generate in base alle indicazioni del prompt inviato alla LLM locale15.
    - Le sintesi per ogni sezione contengono tra 100 e 300 parole.
* **User Story:** Come utente, voglio che il sistema suddivida il contenuto del libro in sezioni gestibili, in modo che ogni sezione possa diventare un modulo o una slide distinta nel corso di autoistruzione.
  + **Criteri di Accettazione:**
    - Dato un libro tecnico, ogni capitolo è suddiviso in 'n' sezioni.
    - Ogni sezione corrisponde a un file di output.
    - Ogni sezione produrrà in output una slide.
    - Ogni capitolo del libro viene trasformato in un modulo del corso (tipicamente con un numero di slide tra 10 e 30).
* **User Story:** Come utente, voglio che il sistema produca il contenuto elaborato in formato Markdown, in modo che possa essere facilmente convertito in una presentazione PowerPoint (e successivamente in SCORM).
  + **Criteri di Accettazione:**
    - Il sistema genera uno o più file di testo Markdown.
    - I file Markdown di output aderiscono alla struttura specificata (es. capitolo\_1/parte\_1/parte\_1.md).
    - Ogni file .md contiene il riassunto del suo modulo corrispondente.
    - Viene utilizzata la sintassi Markdown per la formattazione (es. titoli, grassetto, elenchi).

**Epic: Report e Struttura del Sistema**

* **User Story:** Come utente, voglio che il sistema generi un report dettagliato con statistiche, in modo da poter monitorare l'elaborazione e identificare eventuali problemi.
  + **Criteri di Accettazione:**
    - Un file report.txt viene generato nella directory di output.
    - Il report contiene statistiche rilevanti sull'elaborazione.
* **User Story:** Come sviluppatore, voglio che il progetto abbia una struttura di file chiara e organizzata, in modo che sia facile da comprendere, mantenere ed estendere.
  + **Criteri di Accettazione:**
    - Il progetto include main.py, requirements.txt, config.py e la directory utils/.
    - La directory utils/ contiene pdf\_extractor.py, llm\_client.py e output\_generator.py.
    - È presente una directory output/ per i file generati.

**Epic: Gestione degli Errori e Robustezza**

* **User Story:** Come utente, voglio che il sistema gestisca gli errori di base in modo elegante, in modo che problemi inaspettati non portino a un arresto completo del programma.
  + **Criteri di Accettazione:**
    - Il sistema prevede meccanismi di gestione degli errori di base.
    - Vengono forniti messaggi informativi quando si verificano errori.