Archaic: Archival Web Scraper & PDF Converter

Date: September 3, 2025
Version: 2.0

1. Project Objective

The objective of this project is to develop an automated utility with a graphical user interface (GUI) that systematically discovers and downloads all historical web pages from a specific URL path (e.g., example.com/articles/*) as captured by the Internet Archive's Wayback Machine. The script will then process the retrieved HTML for each page, clean it of archival interface elements, and convert it into high-fidelity PDF and clean HTML documents for local, offline record-keeping.

2. Key Goals

- **User-Friendly Operation:** Provide a simple GUI for users to input a target URL, start the process, and monitor progress.
- **Automated Discovery:** Programmatically query the Internet Archive to get a complete list of all unique pages ever archived under a specific path.
- **Content Retrieval:** Systematically download the HTML content for each discovered page from its historical snapshot.
- **Content Cleaning:** Remove the injected Wayback Machine UI (header, toolbars) to produce a clean version of the original page content.
- **Dual-Format Archiving:** Save a clean, standalone HTML file in addition to a generated PDF for each page.
- **Organized Output:** Save the resulting PDF and HTML files with logical, corresponding filenames for easy reference.

3. Methodology

The project will be executed in a four-phase automated process initiated from the GUI:

1. **Phase 1: URL Index Discovery via CDX API**
   - The script will make a single API call to the Internet Archive's **CDX Server API**.
   - A wildcard search (example.com/articles/*) will be used to request a complete index of all captured URLs matching the path.
   - The script will parse this index to create a definitive list of unique pages, targeting the most recent capture of each. The GUI progress bar will update after this phase.
2. **Phase 2: HTML Content Retrieval**
   - For each URL in the list, the script will construct the full Wayback Machine URL (including the capture timestamp).

    - An HTTP request will be sent to fetch the raw HTML content, with the progress bar updating incrementally for each downloaded page.
3. **Phase 3: HTML Parsing and Saving**
    - The raw HTML will be parsed to remove the Wayback Machine's injected header, scripts, and styles.
    - The cleaned HTML will be saved directly to a local file.
4. **Phase 4: PDF Generation**
    - The cleaned HTML string will be passed to a rendering engine (**wkhtmltopdf**) that interprets the HTML and CSS.
    - The rendered page will be saved as a PDF file, preserving the layout, text, and images.

4. User Interface (UI)

The script will feature a simple Graphical User Interface (GUI) to provide user-friendly operation.

- **URL Input:** A text field for the user to enter the target URL path.
- **Controls:** Start and Stop buttons to initiate and interrupt the scraping process.
- **Progress Indicator:** A progress bar and status label to provide real-time feedback on the download and conversion process, showing the current file being processed and the overall completion percentage.

5. Technology Stack

- **Programming Language:** Python 3
- **Core Libraries:**
    - requests: For making HTTP calls to the Internet Archive API and servers.
    - BeautifulSoup4: For parsing and cleaning the retrieved HTML content.
    - pdfkit: A Python wrapper for the wkhtmltopdf utility.
    - Tkinter: For building the cross-platform graphical user interface.
- **Key Utility:** wkhtmltopdf: The underlying command-line tool that handles the HTML-to-PDF conversion.

6. Key Deliverables

1. **A collection of both HTML and PDF files**, with each file representing a single archived page from the target path.
2. **A corresponding collection of cleaned, standalone HTML files.**
3. **The final, commented Python script** capable of running the entire process via the GUI.
4. **A log file** documenting the URLs that were successfully processed and any that failed.

7. Risks & Mitigations

- **Risk:** Overwhelming the Internet Archive's servers, leading to IP blocks.
    - **Mitigation:** Implement "respectful" scraping practices, including a 1-2 second

delay between all requests.

- **Risk:** Incomplete archives (missing pages, images, or stylesheets).
    - **Mitigation:** This is an inherent limitation. The script will log any failed requests for later review.
- **Risk:** Pages with complex JavaScript or dynamic content may not render perfectly in PDF.
    - **Mitigation:** The project scope is limited to static HTML content. We will accept minor rendering imperfections as a known constraint.