Bern University
of Applied Sciences

URL-Archiver

Project report

Course of study	Project 1
Author	Nicolin Dora, Abidin Vejseli & Kilian Wampfler
Advisor	Dr. Simon Kramer
Co-advisor	Frank Helbling

Version 0.1 of October 9, 2023

- Technik und Informatik
- Informatik

Abstract

One-paragraph summary of the entire study – typically no more than 250 words in length (and in many cases it is well shorter than that), the Abstract provides an overview of the study.

Contents

List of Tables

List of Figures

Listings

1. Introduction

1.1. Initial Situation

In the age of rapidly evolving digital content, there's an increasing need to archive and preserve web resources. URLs, which provide direct access to these resources, are often embedded within various file types, including but not limited to Unicode-text files like .BIB, .TEX, and .HTML, as well as .PDF files. However, with the internet's transient nature, these URLs can become obsolete or the content they point to can change, leading to the loss of valuable information. It is, therefore, crucial to have a mechanism in place to archive these URLs, ensuring their content remains preserved and accessible over time.

1.2. Project Goal

The primary aim of the "URL-Archiver" project is to develop a Free/Libre and Open Source Software (FLOSS) licensed, platform-independent Java program that can effectively archive URLs. The envisioned software should:

1. Accept as input a directory or any Unicode-text file (like .BIB, .TEX, .HTML), or a .PDF file.
2. Scan the provided input for any embedded URLs.
3. Extract all identified URLs.
4. Offer an option to spring-load these URLs in a Web-browser, providing an immediate view of the referenced content.
5. Submit these URLs to the online archiving service, https://archive.ph, ensuring their content is stored and safeguarded against potential future changes or deletions.
6. Retrieve the archived URLs from the service.
7. Output a CSV-file detailing the original URLs and their corresponding archived versions.
8. Optionally allow the integration of the archived URLs into a .BIB file, facilitating the ease of reference in academic or professional contexts.

Furthermore, in the spirit of best programming practices, the program's code should be minimalistic, modular, and self-explanatory. This not only ensures maintainability but also aids any future development or modifications.

1.3. Priorities

1. **Functionality**: The primary priority is the accurate extraction and archiving of URLs. The software should reliably identify URLs in varied file types and ensure their successful archiving on https://archive.ph.
2. **Usability**: Given the diverse potential user base, the program should be platform-independent and possess a user-friendly interface. While the underlying mechanisms may be complex, the user experience should be seamless and intuitive.
3. **Code Quality**: Emphasis should be placed on writing clean, minimal, and modular code. This not only aids in potential future enhancements but also in debugging and troubleshooting.
4. **Documentation**: As with any software project, proper documentation is paramount. The project report should be concise, adhering to the principle of being "maximally informative, minimally long," ensuring clarity of information without overwhelming the reader.
5. **Integration with Existing File Types**: The ability to seamlessly insert archived URLs into .BIB files is a priority, given the potential academic applications of the software.

2. Specification

2.1. System Delimitation

2.1.1. System Environment (statics)

2.1.2. Process Environment (dynamics)

2.2. Requirements

2.2.1. Epics and User Stories

In this section, we outline the main features (Epics) of the project and break them down into detailed user tasks (User Stories). This helps provide a clear understanding of the desired functions and behaviors of our software.

Epic 1: File Input and Processing

Goal: Allow the user to input various file types via the command line and prepare these files for further processing.

1. **Prompt for File Path Input**
   - **Description**: As a user, when I start the tool, I want to be prompted to input the path to my file, so the tool knows which file to process.
   - **Acceptance Criteria**:
     - Upon starting the tool, it prompts the user to enter a file path.
     - On inputting an invalid path or if there are permissions issues, the tool provides a relevant error message.
2. **Automatic File Type Detection**
   - **Description**: As a user, I want the tool to automatically detect the file type (based on file extension) and treat it accordingly so that I don't need to specify the file type separately.
   - **Acceptance Criteria**:
     - The tool automatically identifies if the file is a .BIB, .TEX, .HTML, or .PDF.

- For unrecognized file types, the tool provides an appropriate error message.

3. **Processing of Directories**
   - **Description**: As a user, I want to input a whole directory, so the tool processes all supported files contained within.
   - **Acceptance Criteria**:
     - The tool can accept directory paths after the prompt.
     - It processes all supported file types within the directory.
     - The tool gives a message if files within the directory are skipped due to their type.
4. **Processing Feedback**
   - **Description**: As a user, I want to receive feedback when the tool starts processing the file and when it finishes, to know the status.
   - **Acceptance Criteria**:
     - A message is displayed when the processing of a file starts.
     - Upon completion, a confirmation message is shown, which also includes any potential errors or warnings.

Epic 2: URL Detection and Extraction

Goal: Accurately detect and extract URLs from input files for further processing.

1. **Scan Files for URLs**
   - **Description**: As a user, I want the system to scan my input files and identify any embedded URLs so that they can be extracted for archiving.
   - **Acceptance Criteria**:
     - System can detect URLs in a variety of file formats including .BIB, .TEX, .HTML, and .PDF.
     - Detected URLs are listed without any duplication.
2. **Use Regular Expressions for Extraction**
   - **Description**: As a user, I want the system to use regular expressions or other reliable techniques to extract URLs so that all valid URLs are captured without error.
   - **Acceptance Criteria**:

  - System uses a robust regular expression pattern that matches most URL formats.
  - Extracted URLs are validated to ensure they are in the correct format.

3. **Store URL Line Number or Context**
   - **Description**: As a user, when a URL is detected and extracted, I want the system to also store its line number or contextual information from the original file, enabling precise placement of its archived counterpart later on.
   - **Acceptance Criteria**:
     - Upon URL detection, the system captures and stores the line number or relevant context of the URL from the source file.
     - This information is utilized later if archived URLs need to be placed back into the original files.
4. **Compile a List of URLs**
   - **Description**: After extraction, I want all URLs to be compiled into a single list, eliminating any duplicates, so that I have a clean list for archiving.
   - **Acceptance Criteria**:
     - The list contains all the unique URLs found in the input files.
     - Invalid or broken URLs are flagged or removed from the list.

Epic 3: Web Browser Integration

Goal: Seamlessly open detected URLs, one at a time, in a web browser for user verification, and immediately initiate the archiving process upon user decision.

1. **Sequential URL Preview**
   - **Description**: As a user, I want to preview each detected URL in my default browser sequentially to verify its content.
   - **Acceptance Criteria**:
     - System opens one URL at a time in the default browser.
     - Immediately after the URL is displayed, the system presents the user with the option to archive.
2. **Immediate Archiving Upon Decision**

- **Description**: After reviewing a URL in the browser, I want to decide if it should be archived. If I decide to archive, the system should immediately initiate the archiving process.
- **Acceptance Criteria**:
  - System provides a prompt to accept or decline the archiving of the displayed URL.
  - If the user chooses to archive, the system directly begins the archiving process, and the user may need to manually solve captchas.

3. **Track Archiving Progress**
   - **Description**: As a user, I want a clear indicator of how many URLs have been displayed, archived, and how many are left to process.
   - **Acceptance Criteria**:
     - The system displays a counter indicating the number of URLs already shown to the user.
     - Another counter indicates how many URLs have been chosen for archiving.
     - Yet another counter shows how many URLs remain to be processed/displayed.
4. **Store User Decisions for Reporting**
   - **Description**: As a user, after making a decision about archiving each URL, I want the system to store my choices so that they can be referred to or reported on later.
   - **Acceptance Criteria**:
     - The system maintains a record of each URL and the user's decision (archived or not archived).
     - The stored decisions are available for any subsequent reporting needs.

Epic 4: Interaction with archive.ph

Goal: Automate the process of archiving URLs via archive.ph while ensuring user interaction is seamless and all necessary data is captured for later use.

1. **Automated URL Submission**
   - **Description**: As a user, I want the system to automatically fill in the URL into the archive.ph input field and submit it for archiving.
   - **Acceptance Criteria**:

- Upon initiation, system opens the archive.ph website in a browser.
- System auto-fills the given URL into the appropriate input field.
- System automatically triggers the submission process for archiving.

2. **User Interaction for Captchas**
   - **Description**: If required, I want to manually solve captchas to ensure the URL gets archived.
   - **Acceptance Criteria**:
     - If archive.ph presents a captcha, the system allows the user to solve it manually.
     - The archiving process proceeds once the captcha is successfully solved.
3. **Automatic Retrieval of Archived URL**
   - **Description**: Once a URL is archived, I want the system to automatically retrieve and display the archived URL to me.
   - **Acceptance Criteria**:
     - System captures the new archived URL from archive.ph after the process completes.
     - The archived URL is displayed to the user immediately.
     - The archived URL is stored for later processing and reporting.

Epic 5: Output and Reporting

Goal: Provide the user with an organized CSV file detailing URLs and their archived counterparts. Also, allow for integration of archived URLs back into supported input files.

1. **Generate CSV File**
   - **Description**: As a user, I want the system to produce a CSV file containing all original URLs and their corresponding archived URLs.
   - **Acceptance Criteria**:
     - A CSV file is generated upon completion of the archiving process.
     - Each row in the CSV contains the original URL and its archived counterpart.
2. **Integrate Archived URLs into Supported Files**

- **Description**: If desired, I want the system to insert the archived URL back into the original file, following its corresponding original URL.
- **Acceptance Criteria**:
  - The system recognizes supported file types for this integration process.
  - Upon user approval, the archived URL is inserted in the appropriate location (e.g., following its original URL) within the file.

2.2.2. Functional requirements (added value)

2.2.3. Boundary and Pre-Conditions

2.3. Usability

2.3.1. Personas

2.3.2. Storyboard

2.3.3. UX-Prototyping

3. Implementation

3.1. Architecture (e.g., back-/frontend)

3.2. Processes

4. Deployment/Integration

4.1. Installation (Sysadmin) Manual & Script

4.2. User Manual

5. Conclusion

5.1. Discussion

5.1.1. Example from BFH Template - Delete

What is the significance of your results? – the final major section of text in the paper. The Discussion commonly features a summary of the results that were obtained in the study, describes how those results address the topic under investigation and/or the issues that the research was designed to address, and may expand upon the implications of those findings. Limitations and directions for future research are also commonly addressed.

5.2. Bottom Line

5.3. Future Work

Bibliography

A. Original Project Description

```
# URL-Archiver

## Description

The goal of this project is to deliver a FLOSS-licensed,
platform-independent Java-program (called "URL-Archiver") that
(1) takes as input (the path of) a directory or any Unicode-text-
    (e.g.: .BIB, .TEX; .HTML; etc.) or .PDF-file
     (https://www.baeldung.com/java-curl);
(2) scans it for any URLs
 (https://stackoverflow.com/questions/4026614/extract-text-from-pdf-files ,
    https://librepdf.github.io/OpenPDF , https://pdfbox.apache.org ; see
     also
    https://en.wikipedia.org/wiki/List_of_PDF_software);
(3) extracts all URLs (regular expression ;-) from the text;
(4) optionally spring-loads all URLs in a Web-browser;
(5) posts all URLs to https://archive.ph ;
(6) gets the resulting archived URLs;
(7) outputs a CSV-file of the resulting key-value (URL, archived URL)
 pairs; and
(8) optionally inserts the archived URLs into a .BIB-file.

The program code should be minimal, modular, and self-explaining.

The project report should be concise (maximally informative, minimally
 long).
It must contain this project description as a quotation.

## Technologies

Java, LaTeX

## Advisor

Dr. Simon Kramer
```

Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those acknowledged.

All statements taken from other writings, either literally or in essence, have been marked as such.

I hereby agree that the present work may be reviewed in electronic form using appropriate software.

October 9, 2023

N. Dora

A. Vejseli

K. Wampfler