



Bern University
of Applied Sciences



URL-Archiver

Project report

Course of study

Author

Advisor

Co-advisor

Project 1

Nicolin Dora, Abidin Vejseli & Kilian Wampfler

Dr. Simon Kramer

Frank Helbling

Version 0.1 of December 29, 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

Abstract	ii
List of Tables	v
List of Figures	vi
Listings	vii
1. Introduction	1
1.1. Initial Situation	1
1.2. Product Goal	2
1.3. Priorities	2
2. Specification	4
2.1. System Delimitation	4
2.1.1. System Environment (statics)	4
2.1.2. Process Environment (dynamics)	5
2.2. Requirements	5
2.2.1. Epics and User Stories	5
2.2.2. Functional requirements (added value)	10
2.2.3. Boundary and Pre-Conditions	10
2.3. Usability	10
2.3.1. Personas	10
2.3.2. Storyboard	10
2.3.3. UX-Prototyping	10
3. Implementation	11
3.1. Architecture (e.g., back-/frontend)	11
3.2. Processes	11
4. Deployment/Integration	12
4.1. Licensing and Compliance	12
4.1.1. Licensing Overview	12
4.1.2. Compliance with Open Source Licenses	12
4.1.3. Purpose of Compliance	13
4.2. Installation (Sysadmin) Manual & Script	13
4.3. User Manual	13

5. Conclusion	14
5.1. Discussion	14
5.1.1. Example from BFH Template - Delete	14
5.2. Bottom Line	14
5.3. Future Work	14
Bibliography	15
A. Original Project Description	16
A.1. List of Used Libraries and Their Licenses	17

List of Tables

A.1. List of Used Libraries in the Project	17
--	----

List of Figures

Listings

1. Introduction

1.1. Initial Situation

The Internet is constantly evolving, which means that there is no guarantee that a website as it exists today will still exist in a few years' time, let alone contain the same information. While this might not be a concern that the average Internet user has to grapple with, it poses a challenge to the academic demographic, where it becomes crucial to reference sources and potentially integrate links to additional data. If links become inactive, verifying the sources becomes challenging, if not impracticable. Archiving the existing status of a website is achievable, but it currently necessitates a manual and hence time-intensive operation, which not many people take the time to do. The objective of this project is to devise an automated solution to this predicament that is independent of platforms. The stakeholders for this solution include:

- ▶ IT users possessing basic computer skills who can download and operate the program from Github
- ▶ Dr. Simon Kramer, the technical project supervisor and intellectual owner of the project idea

1.2. Product Goal

The product goal is a platform independent Java application called “URL-Archiver“. The application must be Free/Libre and Open Source Software (FLOSS) licensed and fulfil the following functionalities:

1. The software should be CLI¹-based and offer a clear command line.
2. The software should allow the user to input a path, which can be a folder or any Unicode text file.
3. The software examines the contents of a file or folder to extract any web URLs using a standard regular expression or similar method.
4. If desired, URLs can be automatically opened in a web browser.
5. The extracted URLs are archived on archive.today and/or web.archive.org (known as The Wayback Machine) as per the user’s preference.
6. The software outputs the resulting archive URLs to the user.
7. The software generates a CSV file containing the original URL and the archived Version of the URL.
8. Optionally, the archived Versions are written back into the provided .bib file.

The product goal is achieved if the software covers all the functionality listed above. Furthermore, the code should be minimalistic, modular, and self-explaining. In addition to the code, it is essential that the following documents are provided:

- ▶ User manual
- ▶ Installation instructions (including installation script)
- ▶ Software documentation

1.3. Priorities

The following priorities are listed in order of importance:

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.

¹Command Line Interface

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)

System Overview

The primary purpose of the URL-Archiver is to extract URLs from Unicode text files and PDFs, and archive them on supported platforms: Archive.today and the Wayback Machine. The system provides the archived URL versions to the user via a CSV file. Additionally, when a .bib file is provided by the user, the original bib file is updated with a note field containing these archived URLs for each entry.

Hardware Specifications

The URL-Archiver does not impose any special hardware requirements. However, an internet connection is essential for the archiving process to function.

Software Components

The URL-Archiver is platform-independent, operating on major systems such as Windows (tested on Windows 10 and 11), macOS, and Linux (tested on Ubuntu). The system has varying browser dependencies based on the operating system: Chrome is required for macOS, Edge for Windows, and Firefox for Ubuntu/Linux. Users can modify these settings in a configuration file. Other dependencies are installed with the URL-Archiver and do not require separate installation.

System Architecture

The URL-Archiver employs the Model-View-Controller (MVC) pattern to facilitate future enhancements like adding a GUI interface. The Factory pattern is applied where appropriate to simplify the extension of functionalities. For example, adding additional archiving services can be easily accomplished.

Data Management

Upon completion of its execution, the URL-Archiver stores all URLs in a CSV file. Optionally, it can also write back URLs into a .bib file.

User Interface

Currently, the system uses a command-line interface. The MVC pattern lays the groundwork for potential future implementation of a GUI interface.

Security and Compliance

The URL-Archiver does not have specific security requirements to meet.

Integration with Other Systems

The system integrates with the Wayback Machine via API, with certain limitations detailed in their API documentation (<https://archive.org/details/spn-2-public-api-page-docs/mode/2up>). For archiving on Archive.today, which lacks an API, Selenium is used to automate the process as much as possible. However, users must manually complete captchas.

Scalability and Performance

There are no specific scalability requirements or performance benchmarks that the URL-Archiver is designed to meet.

Maintenance and Support

Currently, there are no specified maintenance requirements or a support framework for the URL-Archiver.

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. Licensing and Compliance

4.1.1. Licensing Overview

The project, along with its original source code, is licensed under the permissive and open MIT License. This license aligns with the project's goal of accessibility and ease of use, allowing for free use, modification, distribution, and private use of the software.

4.1.2. Compliance with Open Source Licenses

Although the project itself is licensed under the MIT License, it uses several open-source libraries that are subject to their respective licenses. Please refer to Appendix A.1 for more information. It is worth noting that this project employs libraries licensed under the Eclipse Public License v2.0 (EPL-2.0), including JUnit Jupiter API.

Using EPL-2.0 licensed libraries in a MIT-licensed project is compliant with open source licensing standards, as long as certain conditions are met.

1. **Attribution and Notices:** The project includes a 'Licenses and Attributions' section in the README file. This section acknowledges the use of open-source libraries, specifying their licenses and providing due credits. Additionally, a 'NOTICES.txt' file is included in the project's resources, detailing the open-source components used, their licenses, and where to find the full license texts. This approach is a crucial step to respect and recognize the work of open-source contributors and to maintain transparency about the software's composition.
2. **Separation of Licenses:** It is important to clarify that the MIT License applies to the original code developed for this project. In contrast, the libraries used retain their original licenses (EPL-2.0 in the case of JUnit).
3. **No Modification of EPL-2.0 Libraries:** This project uses the EPL-2.0 licensed libraries in their unmodified form. Any modification to such libraries would require adherence to the specific terms and conditions of the EPL-2.0.

4.1.3. Purpose of Compliance

Ensuring compliance with the licensing terms of used libraries is not only a legal requirement but also a commitment to the open-source community's ethical standards. It ensures that the project respects the rights and efforts of other developers and contributes to the sustainable and responsible use of open-source software.

4.2. Installation (Sysadmin) Manual & Script

4.3. 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

A.1. List of Used Libraries and Their Licenses

Below is the list of libraries used in the project, along with a short description and their versions.

Library	Version	Short Description	Used License
JUnit Jupiter API	5.9.2	Unit testing framework for Java applications.	Eclipse Public License v2.0
JUnit Jupiter Engine	5.9.2	The test engine for running JUnit tests.	Eclipse Public License v2.0
Selenium Java	4.15.0	Automation framework for web applications testing.	Apache-2.0
Selenium Logger	2.3.0	A wrapper for enhanced Selenium log management.	MIT
Mockito Core	5.4.0	Mocking framework for unit tests in Java.	MIT
Mockito JUnit Jupiter	5.4.0	Integration of Mockito with JUnit Jupiter.	MIT
System Lambda	1.2.1	Utilities for testing Java code that uses system properties and environment variables.	MIT
Apache PDFBox	3.0.0	Library for creating and manipulating PDF documents.	Apache-2.0
Jackson Core	2.16.0	Core part of Jackson that defines common low-level features.	Apache-2.0
Jackson Dataformat XML	2.15.2	Support for reading and writing XML encoded data via Jackson abstractions.	Apache-2.0

Table A.1.: List of Used Libraries in the Project

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.

December 29, 2023



N. Dora



A. Vejseli



K. Wampfler