



Bern University
of Applied Sciences



URL-Archiver

Project report

Course of study

Author

Advisor

Co-advisor

Version 1.0 of January 15, 2024

Project 1

Nicolin Dora, Abidin Vejseli & Kilian Wampfler

Dr. Simon Kramer

Frank Helbling

- School of Engineering and Computer Science
- Computer Science

Abstract

The URL-Archiver is a platform-independent Java application which specializes in extracting and archiving URLs from Unicode text or PDF files. It supports archiving on Archive Today and the Wayback Machine, with options to export archived URLs into CSV or the original BibTeX files. The application was developed with a focus on simplicity, modularity and self-explanatory code.

This report details the development process of the URL-Archiver, explaining the implementation strategies, SCRUM methodology and challenges overcome, and concludes with a outlook on potential future enhancements.

Contents

Abstract	ii
List of Tables	v
List of Figures	vi
Listings	viii
1. Introduction	1
1.1. Initial Situation	1
1.2. Product Goal	2
1.3. Priorities	3
2. Specification	4
2.1. System Delimitation	4
2.1.1. System Environment (statics)	4
2.1.2. Process Environment (dynamics)	6
2.2. Requirements	8
2.2.1. Epics and User Stories	8
2.2.2. Functional requirements	14
2.2.3. Boundary	14
2.2.4. Pre-conditions	15
2.3. Usability	16
2.3.1. Personas	16
2.3.2. Persona 1 - Alex Frei (Investigative Journalist)	16
2.3.3. Persona 2 - Dr. Emma Katrin Winter (Academic Researcher)	17
2.3.4. Persona 3 - Michael Schwarz (Content Marketing Specialist)	18
2.3.5. Storyboard	18
2.3.6. UX-Prototyping	24
3. Implementation	28
3.1. Architecture	28
3.1.1. Frontend	28
3.1.2. Backend	30
3.1.3. URL Extraction Process	36
3.2. Processes	36

4. Deployment/Integration	37
4.1. Licensing and Compliance	37
4.1.1. Licensing Overview	37
4.1.2. Compliance with Open Source Licenses	37
4.1.3. Purpose of Compliance	38
4.2. Installation (Sysadmin) Manual & Script	38
4.2.1. Requirements	38
4.2.2. Clone the repository	38
4.2.3. Build and run scripts	38
4.3. User Manual	40
4.3.1. Getting Started	40
4.3.2. Operating Instructions	40
5. Conclusion	42
5.1. Discussion	42
5.2. Bottom Line	43
5.2.1. Conclusion	43
5.2.2. Productivity Increase Calculation	43
5.3. Future Work	45
Glossary	47
Index	49
Bibliography	50
A. Appendix	51
A.1. Original Project Description	52
A.2. List of Used Libraries and Their Licenses	53
A.3. Class Diagram	54

List of Tables

A.1. List of Used Libraries in the Project	53
--	----

List of Figures

2.1. High-level MVC-Pattern from URL-Archiver	5
2.2. Extension with new archiving service XY (using the Factory Pattern)	5
2.3. High-level operational process	7
2.4. The console greets Bob with a straightforward interface, presenting the archiving options. Bob opts to begin the archiving process.	19
2.5. Bob is prompted to enter the file or directory path. He types in the path to the PDF document that contains the URLs he wishes to archive.	20
2.6. Upon processing the input, the URL-Archiver quickly finds the first URL within the document. The console inquires if Alex wants to open it in a web browser or proceed with archiving. Alex decides not to open the URL in the browser and instead chooses to archive it directly.	20
2.7. The Archiver initiates the archiving process. A loading screen appears, indicating that the Archive Today platform is working on capturing the webpage.	21
2.8. The first URL is archived, and the Archiver outputs the archived link.	21
2.9. The application moves to the next URL. Without hesitation, Bob decides to archive this one as well.	22
2.10. The last URL is processed, and the Archiver asks Bob if he wants to save the list of archived URLs as a CSV file. Bob types "Y" to confirm.	23
2.11. Bob feels accomplished. The Archiver saved time, and the important URLs are now securely archived.	23
2.12. Starting point of the URL-Archiver where the user is prompted to begin archiving or exit the application.	24
2.13. Upon choosing to open a URL in the browser, the user is presented with the live web page.	25
2.14. The user opts to archive the second URL, initiating the archiving process through the interface.	25
2.15. A security step requiring captcha verification to proceed with the archiving process.	26
2.16. Archive Today informs that the website has been archived before. The application proceeds to re-archive the site.	26
2.17. The archiving process is actively underway on the Archive Today platform.	27
2.18. After successful archiving, the system provides the user with a confirmation message and the archived URL.	27
3.1. Screenshot of the Console View	29
3.2. Highlevel Diagram of implemented MVC Pattern	31

3.3. Diagram of the FileReader Factory	33
3.4. Diagram of the Exporter Factory	33
3.5. Class Diagram	35

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:

- ▶ Legal professionals and researchers who need to preserve web content as evidence or for case study references.
- ▶ Journalists and media agencies that require archiving web pages for future reporting or fact-checking.
- ▶ Librarians and archivists tasked with the digital preservation of online materials for historical records.
- ▶ Content creators and marketers who wish to maintain records of web content for portfolio or audit purposes.
- ▶ Educators and students who need to collect and cite online resources for academic projects and research.
- ▶ Organizations and businesses that need to archive their web presence for compliance and record-keeping.

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 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 BibTeX 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

¹Command Line Interface

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 Archive Today² or Wayback Machine³.
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 BibTeX files is a priority, given the potential academic applications of the software.

²<https://archive.today>

³<https://web.archive.org/save/>

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 BibTeX file is provided by the user, the original BibTeX 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, version 22H2 and Windows 11, version 23H2), macOS (tested on macOS Sonoma), and Linux (tested on Ubuntu 20.04.3 LTS). The system has varying browser dependencies based on the operating system: Chrome is required for macOS, Edge for Windows, and Firefox for Ubuntu/Linux (Latest stable versions of the browsers are recommended). Users can change the default Browser in the configuration of the application. Other dependencies are installed with the URL-Archiver and do not require separate installation.

System Architecture

The URL-Archiver uses the Model-View-Controller (MVC) pattern, as illustrated in figure 2.1, to enable future enhancements, such as adding a GUI interface. The Factory Pattern is applied where appropriate to simplify the extension of functionalities. For instance, adding extra archiving services can be easily accomplished by introducing a new archiving service, as shown in figure 2.2.

2. Specification

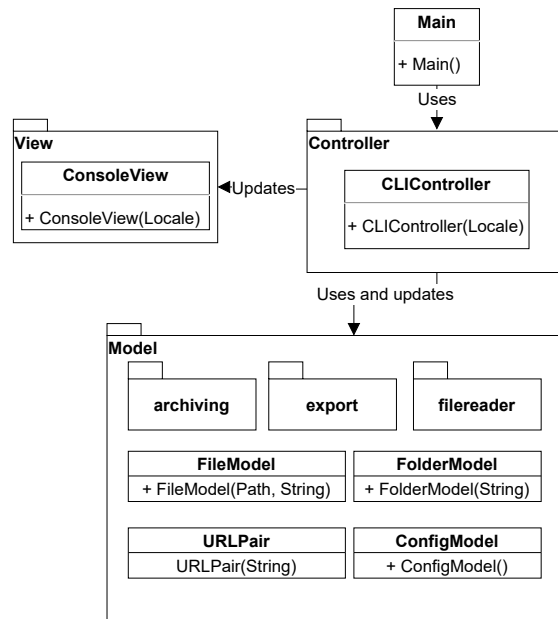


Figure 2.1.: High-level MVC-Pattern from URL-Archiver

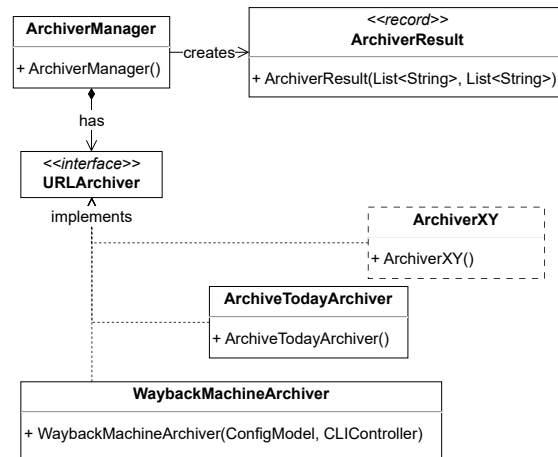


Figure 2.2.: Extension with new archiving service XY (using the Factory Pattern)

Data Management

Upon completion of its execution, the URL-Archiver generates a CSV file where each line contains an extracted URL and its archived versions, separated by semicolons. For example, a line like `https://xy.com;https://web.archive.org/xy;https://archive.ph/xy` shows the original URL and its archives. This simple format makes it easy to track and manage archived URLs.

Optionally, if the user provided a BibTeX file, URLs are integrated into the note field of each entry. If there's no existing note field, a new one is created with the format `note = Archived Versions: url1, url2`. If a note field already exists, the archived URLs are appended to it in the format `note = <current note>, Archived Versions: url1, url2`. This approach ensures that the archived URLs are neatly added to the BibTeX entries, maintaining the integrity of the original data.

User Interface

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

Integration with Other Systems

The system integrates with the Wayback Machine via API, with certain limitations detailed in their API documentation¹. 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.

Maintenance and Support

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

2.1.2. Process Environment (dynamics)

Operational Processes

The URL-Archiver is initiated by the user, who provides a path to Unicode text or PDF files or a directory that contains such files. The application extracts URLs from these files and presents them sequentially to the user. The user then has options to open or archive that URL. He has also the following other options:

- ▶ **s:** Show a list of previously archived URLs.

¹<https://archive.org/details/spn-2-public-api-page-docs/mode/2up>

- ▶ **u**: Update and view pending archive jobs.
- ▶ **n**: Navigate to the next URL.
- ▶ **q**: Quit the application.
- ▶ **c**: Change application settings.
- ▶ **h**: Access the Help Menu for assistance.

Upon completion, the user is prompted to save URL pairs to a CSV file and, if a Bibtex file is provided, to write the archived URLs back into it.

Event Handling

In the URL Archiver, user actions are efficiently facilitated through the main menu. When archiving a URL, users can select either the Wayback Machine or Archive.today. In addition, the 'c' option in the menu allows users to configure settings, including setting up API keys for the Wayback Machine and selecting a default browser. The application cleverly handles unsupported input and incorrect paths by prompting the user for the correct information or action. This ensures smooth operation and user guidance throughout the process.

Life Cycle

The URL-Archiver's life cycle begins with launch and path input, proceeding to URL extraction and user interactions via the menu options, and ends with prompts for data saving upon completion. See the high-level process in figure 2.3.

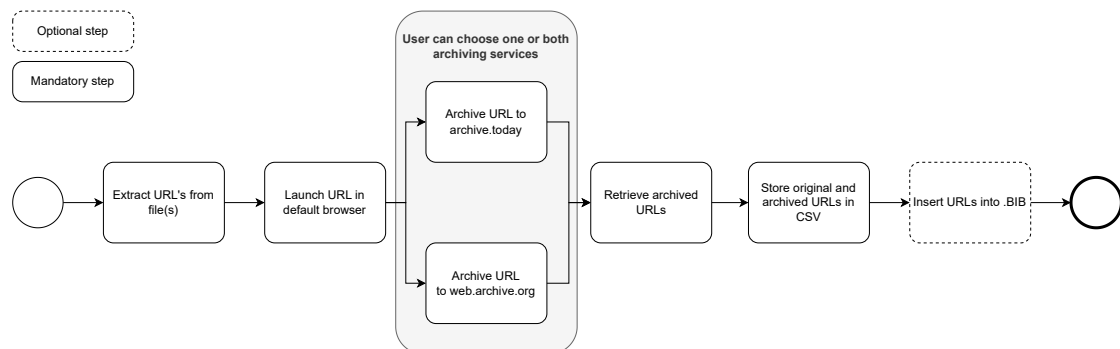


Figure 2.3.: High-level operational process

Error Management

Errors within the URL-Archiver are caught and handled, typically prompting the user with a customized error message asking to retry the action. No system stack traces are shown to the user.

Backup and Recovery

The URL Archiver immediately serialises queued jobs for the Wayback Machine into a JSON file at the start of the archiving sequence. This protocol is designed to preserve the integrity of jobs awaiting execution, thereby increasing the stability of the system in the midst of possible disturbances, including unexpected application terminations. Consequently, it facilitates the recovery of the queue when the application is subsequently activated. It should be noted, however, that URLs that have been successfully archived are not preserved automatically in the current version of the URL Archiver. This feature is a task for future enhancement of the application.

Update and Upgrade Policies

Software updates require manual download and recompilation from the Git repository. The system does not provide automatic updates or an in-built feature for update checks.

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

The URL-Archiver project, as described in the original project description, which can be found in the Appendix (A.1), is designed to be a platform-independent Java program that performs several key functions:

- ▶ **Input Processing:** It accepts directories or files (including any Unicode-text- and PDF-Files) as input and scans them for URLs.
- ▶ **URL Extraction:** The program extracts all URLs using regular expressions.
- ▶ **Optional Browser Interaction:** It has an optional feature to open URLs in a web browser.
- ▶ **URL Archiving:** The program posts URLs to an archiving service (Archive Today & The Wayback Machine) and retrieves the archived URLs.
- ▶ **Output Generation:** It outputs a CSV file containing the original and archived URL pairs.
- ▶ **Optional File Integration:** The program can optionally insert archived URLs into a .BIB file.

Additionally, the project emphasizes minimal, modular, and self-explanatory code. The report for the project is expected to be concise and include the original project description.

2.2.3. Boundary

The following limits apply:

- ▶ **Input:** The application processes only Unicode-text-files (e.g.: .BIB, .TEX; .HTML; etc.) and PDF-files
- ▶ **Output:** The application exports the extracted and archived URLs in either CSV format or directly in a BIB file.
- ▶ **Archiving:** At present, the application should only support Archive Today and The Wayback Machine. Please refer to the documentation of the Wayback Machine² or Archive Today³ for any restrictions on their APIs or services.

²API Documentation Wayback Machine

³FAQ of Archive Today

2.2.4. Pre-conditions

For the application to run smoothly, the following pre-conditions must be met:

- ▶ **Java Environment:** The application necessitates the installation of Java Runtime Environment (JRE) version 21 on the system where it is intended to be executed.
- ▶ **File Formats:** The software is designed to operate with specific file formats, primarily Unicode text files and PDF files. It is necessary to provide these file types for successful operation.
- ▶ **Archiving Service Access:** The application requires internet connectivity and access to web archiving services, such as Archive Today, to enable its URL archiving functionality.
- ▶ **Login + S3 Credentials (Wayback Machine only):** To archive with the Wayback Machine, a login and S3 credentials are required.

2.3. Usability

2.3.1. Personas

This section introduces personas for the URL-Archiver, detailing diverse user profiles and their interactions with the application, to provide a understanding of its usability and functionality.

2.3.2. Persona 1 - Alex Frei (Investigative Journalist)

Persona Overview

Name: Alex Frei

Age: 32

Occupation: Investigative Journalist

Industry: Media and News Reporting

Background

Alex is an experienced journalist specializing in political reporting. Known for in-depth investigative pieces, Alex often covers high-stakes political events, requiring rapid access to historical web data for fact-checking.

Goals

- ▶ Quickly archive web pages for timely reporting on political events.
- ▶ Maintain an archive of web pages for in-depth analysis and future reference.
- ▶ Retrieve archived URLs efficiently for referencing in articles and reports.

Challenges

- ▶ Finding a tool that archives web pages instantly and reliably.
- ▶ Ensuring easy access and shareability of archived content.
- ▶ Managing a collection of archived URLs efficiently and effectively.

Interactions with URL-Archiver

- ▶ Utilizes URL-Archiver's CLI for quick and efficient archiving.
- ▶ Values the open-source nature for trustworthiness in journalistic integrity.

- ▶ Relies on CSV output for organized referencing of archived content.

2.3.3. Persona 2 - Dr. Emma Katrin Winter (Academic Researcher)

Persona Overview

Name: Dr. Emma Katrin Winter

Age: 40

Occupation: Academic Researcher

Industry: Higher Education

Background

Dr. Winter is a tenured professor focused on social sciences, often citing digital journals and data repositories. She requires reliable digital archiving for scholarly citations and utilizes BibTeX files for managing bibliographic data.

Goals

- ▶ Ensure accurate citation of web content in academic papers.
- ▶ Maintain and manage a digital archive of resources for teaching and research.
- ▶ Share archived content with academic peers for collaboration.
- ▶ Integrate archived URLs into BibTeX files for streamlined academic referencing.

Challenges

- ▶ Needs a reliable method for archiving web pages.
- ▶ Seeks efficient integration of archived URLs into BibTeX files.
- ▶ Manages extensive digital archives for large-scale research projects.

Interactions with URL-Archiver

- ▶ Uses URL-Archiver for capturing and archiving academic papers.
- ▶ Values the integration of archived URLs into BibTeX files for academic referencing.
- ▶ Relies on CSV file generation for organizing digital resources.

2.3.4. Persona 3 - Michael Schwarz (Content Marketing Specialist)

Persona Overview

Name: Michael Schwarz

Age: 35

Occupation: Content Marketing Specialist

Industry: Digital Marketing

Background

Michael is a skilled content marketer, responsible for digital campaigns and web presence analytics in a dynamic marketing agency. His role involves archiving web content for brand reputation management and competitive analysis.

Goals

- ▶ Regularly archive web content for marketing audits and compliance.
- ▶ Manage a digital archive for campaign retrospectives.
- ▶ Ensure easy access to archived content for reviews and strategic planning.

Challenges

- ▶ Needs a tool capable of archiving web content efficiently.
- ▶ Requires a streamlined process for accessing archived material.
- ▶ Seeks an archive system that is user-friendly and reliable.

Interactions with URL-Archiver

- ▶ Regularly uses URL-Archiver for archiving online marketing content.
- ▶ Appreciates the CLI interface for ease and speed of use.
- ▶ Relies on the tool's capability to create organized audit trails.

2.3.5. Storyboard

In this section, we present the storyboard for the URL-Archiver, illustrating the user's interaction with the application. Through the sketches 2.4 to 2.11, we show the user's journey

from the initial greeting in the console to the final archiving of URLs. This story concentrate on the user's experience, the steps they take, and the decisions they make, rather than the precise UI design details.



Figure 2.4.: The console greets Bob with a straightforward interface, presenting the archiving options. Bob opts to begin the archiving process.

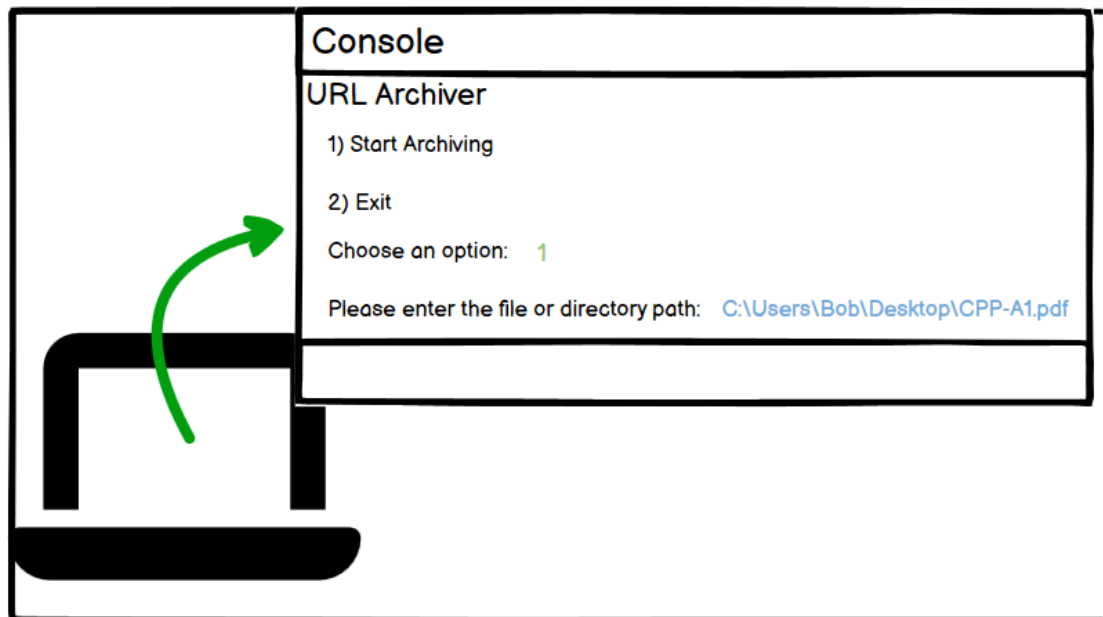


Figure 2.5.: Bob is prompted to enter the file or directory path. He types in the path to the PDF document that contains the URLs he wishes to archive.

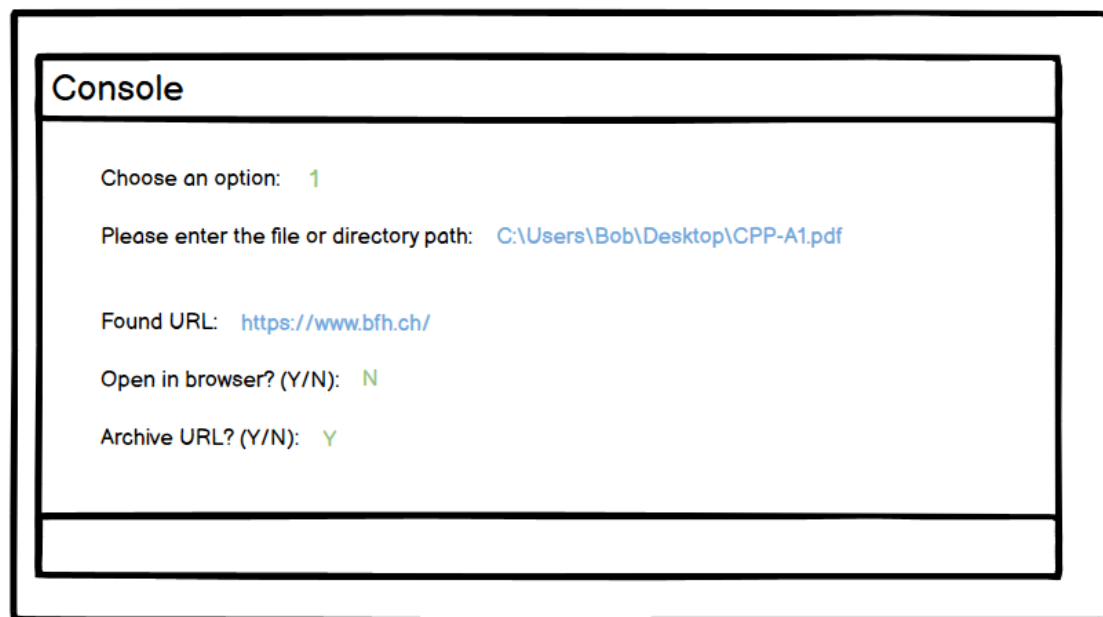


Figure 2.6.: Upon processing the input, the URL-Archiver quickly finds the first URL within the document. The console inquires if Alex wants to open it in a web browser or proceed with archiving. Alex decides not to open the URL in the browser and instead chooses to archive it directly.





Figure 2.9.: The application moves to the next URL. Without hesitation, Bob decides to archive this one as well.



Figure 2.10.: The last URL is processed, and the Archiver asks Bob if he wants to save the list of archived URLs as a CSV file. Bob types "Y" to confirm.

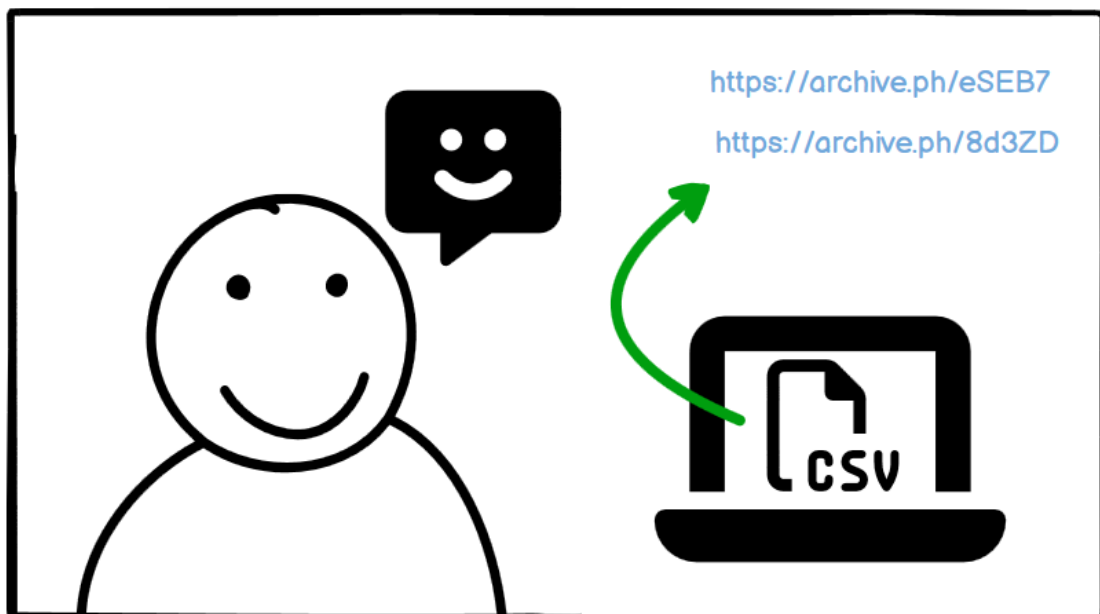


Figure 2.11.: Bob feels accomplished. The Archiver saved time, and the important URLs are now securely archived.

2.3.6. UX-Prototyping

This section presents the initial version of our UX prototype for the URL-Archiver, created using Balsamiq⁴. The series of UI sketches, from Figure 2.12 through Figure 2.18, represent an early conceptualization of the application's user experience. This prototype does not reflect the current state of the application, but serves as an important step in our design process. Each figure illustrates the key stages of user interaction, from starting the application to completing the archiving process.



Figure 2.12.: Starting point of the URL-Archiver where the user is prompted to begin archiving or exit the application.

⁴Balsamiq is a rapid wireframing tool that helps you work faster and smarter. It reproduces the experience of sketching on a whiteboard, but using a computer.



Figure 2.13.: Upon choosing to open a URL in the browser, the user is presented with the live web page.

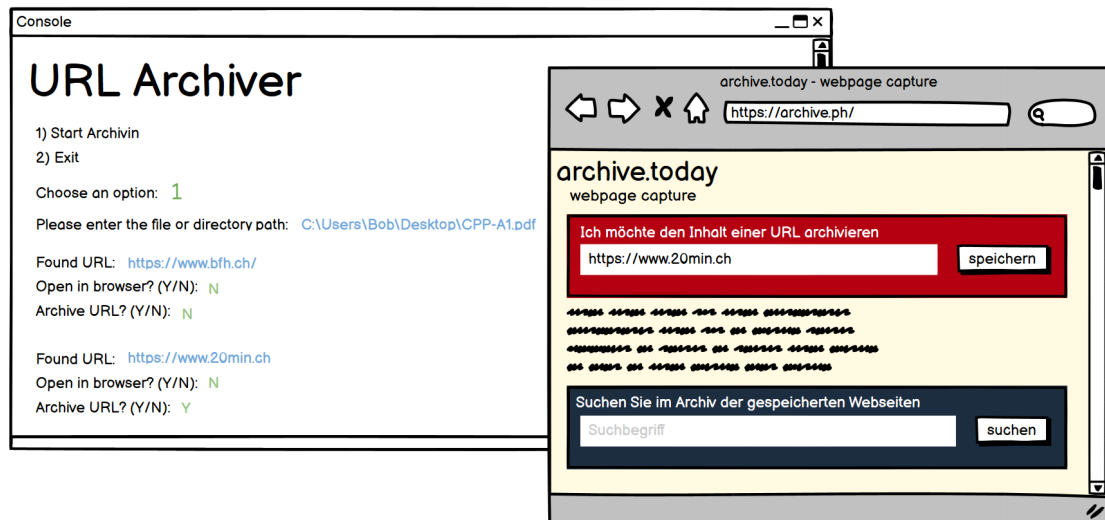


Figure 2.14.: The user opts to archive the second URL, initiating the archiving process through the interface.

2. Specification



Figure 2.15.: A security step requiring captcha verification to proceed with the archiving process.

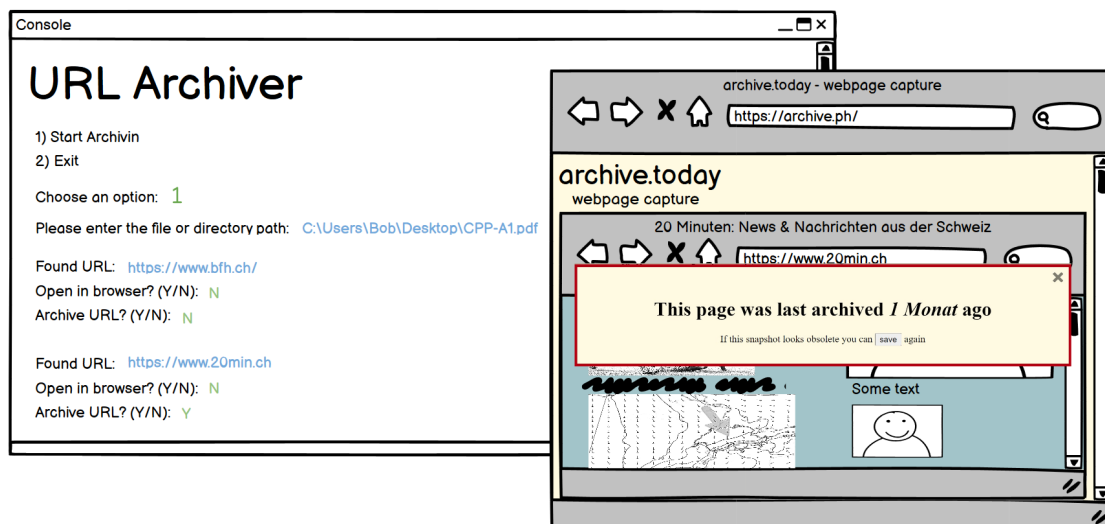


Figure 2.16.: Archive Today informs that the website has been archived before. The application proceeds to re-archive the site.

2. Specification

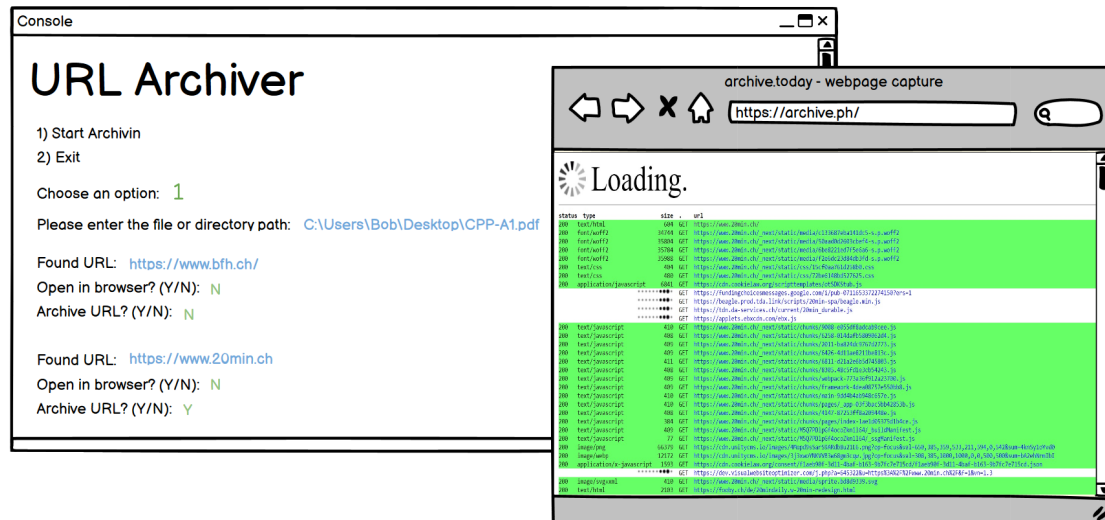


Figure 2.17.: The archiving process is actively underway on the Archive Today platform.



Figure 2.18.: After successful archiving, the system provides the user with a confirmation message and the archived URL.

3. Implementation

3.1. Architecture

This chapter describes the architecture of the URL-Archiver.

3.1.1. Frontend

The frontend architecture of the URL-Archiver is centered on a console-based interface, as evidenced by the `ConsoleView` and `CLIController` classes featured in Figure 3.2.

The `ConsoleView` class is essential for presenting information and managing user input in a console setting. It is specifically designed to display data and messages clearly and in a user-friendly way. In parallel, the `CLIController`, a key component of the Controller segment in the MVC (Model-View-Controller) framework, serves as an intermediary between the `ConsoleView` and the application's backend. This class efficiently processes user inputs from the `ConsoleView`, liaises with the model to retrieve or alter data, and then updates the console interface with these changes. This architecture is crafted to enable streamlined and effective user interactions within a command-line environment. The Figure 3.1 illustrates the welcome message that the URL archiver displays to users.



```
URL-ARCHIVER
Archive Your Web References
If you have already specified a correct path as command line argument, you can start
archiving directly. Otherwise, you will be prompted to enter a path.

For each URL you have the following options:
[o]   Open URL
[a]   Archive URL
[s]   Show archived URL
[u]   Show pending Jobs
[n]   Next URL
[q]   Quit
[c]   Modify Configuration
[h]   Display Help

Enter the file/directory path:
```

Figure 3.1.: Screenshot of the Console View

3.1.2. Backend

Software Architectural Design

The URL-Archiver is developed following the principles of the MVC (Model-View-Controller) software architectural pattern. The MVC framework, as detailed in Figure 3.2, is a structured approach to building user interfaces in software applications. It organizes an application into three interrelated components:

- ▶ **Model:** This layer embodies the URL-Archiver's data structures, business logic, and operational rules. Components such as `FileModel`, `FolderModel`, `URLPair`, and `ConfigModel`, as shown in Figure 3.2, represent the model.
- ▶ **View:** Tasked with rendering data from the model to the user and forwarding user commands to the controller. The `ConsoleView` in Figure 3.2 serves as a demonstration of this component.
- ▶ **Controller:** It is responsible for handling user inputs, engaging with the model to process data, and selecting the appropriate view for presentation. The `CLIController`, depicted in Figure 3.2, is indicative of this segment.

This modular structure allows for an efficient separation of code, which aids in easier maintenance and streamlined development. The `Main` class operates as the launching point for the URL-Archiver, effectively coordinating the MVC pattern. Interaction between the Controller and the Model results in updates to the Model, with the View being refreshed in tandem to maintain a clear demarcation of responsibilities.

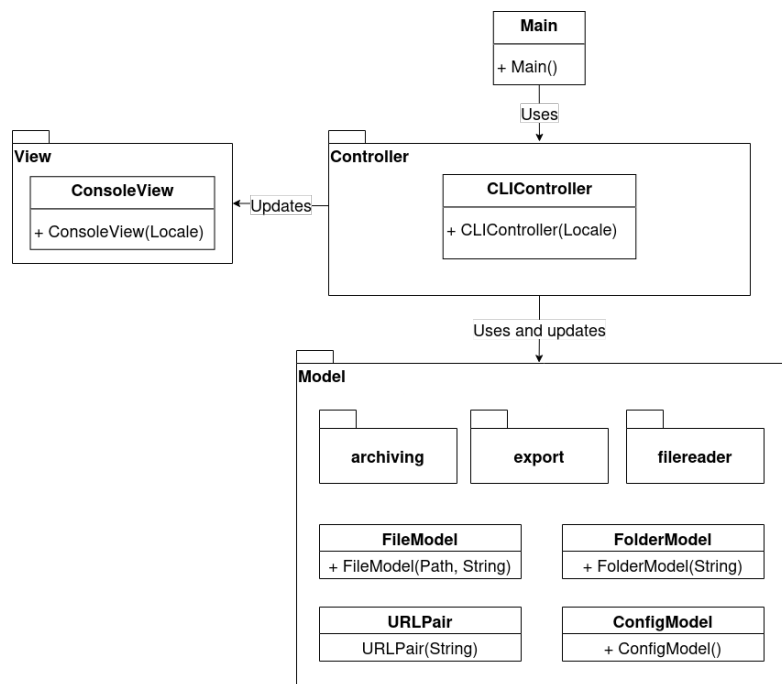


Figure 3.2.: Highlevel Diagram of implemented MVC Pattern

Factory Pattern

The URL-Archiver makes extensive use of the Factory Pattern in its architecture, a key strategy in software engineering recognised for its role in the creational design paradigm. This pattern is central to the encapsulation of the object instantiation process. Rather than creating objects directly using the "new" operator, the Factory Pattern specifies a factory for this purpose. This abstraction improves the flexibility and scalability of the code.

The factory pattern is particularly useful when dealing with a variety of object types to be created, or when specific logic is required in their creation. Its implementation in the URL Archive provides a clear separation of concerns and improves code reusability, in line with the core principles of object-oriented design and programming.

In the URL-Archiver, the Factory Pattern is implemented in several key areas:

- ▶ **File Readers:** Streamlines the addition of more input file types, thanks to its capability to produce objects conforming to the `FileReaderInterface`. This factory is pivotal in handling diverse file types.
- ▶ **Archiving Services:** Adding an additional service is straightforward, enhancing the application's ability to integrate different web archiving services.
- ▶ **Selenium Web Drivers:** Supports the integration of additional browsers with ease, showcasing the pattern's adaptability in browser interactions.
- ▶ **Export Functionality:** The pattern's application here simplifies the inclusion of support for various file types in the export functionality.

These factories contribute significantly to the modularity and extensibility of the URL Archive by encapsulating the creation logic of various components, thus promoting loose coupling and scalability. The alignment of these approaches with the principles of the Factory Design Pattern underscores the maintainability of the application and its adaptability to new file types and data export formats.

The application of the factory pattern in the URL Archive is illustrated in figures 3.3 and 3.4, which show the `FileReaderFactory` and `ExporterFactory` respectively. Similar design principles are applied to other factories within the application, ensuring a consistent and modular approach across all components.

Configuration File Management

The URL-Archiver uses a JSON configuration file to effectively manage various settings and parameters. Key classes such as `ConfigModel` and `ConfigFileHelper` are integral to this process. The `ConfigModel` is used to define the structure of the configuration data, ensuring that all essential settings are systematically represented and easily accessible within the application. Conversely, the `ConfigFileHelper` is responsible for reading and writing the JSON configuration file, decoupling these tasks from the rest of the application code.

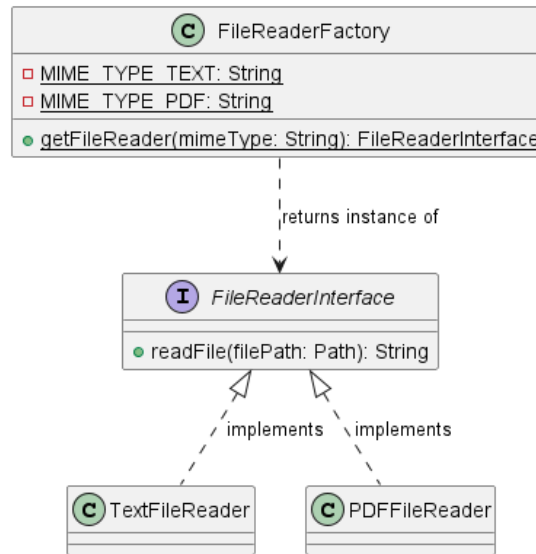


Figure 3.3.: Diagram of the FileReader Factory

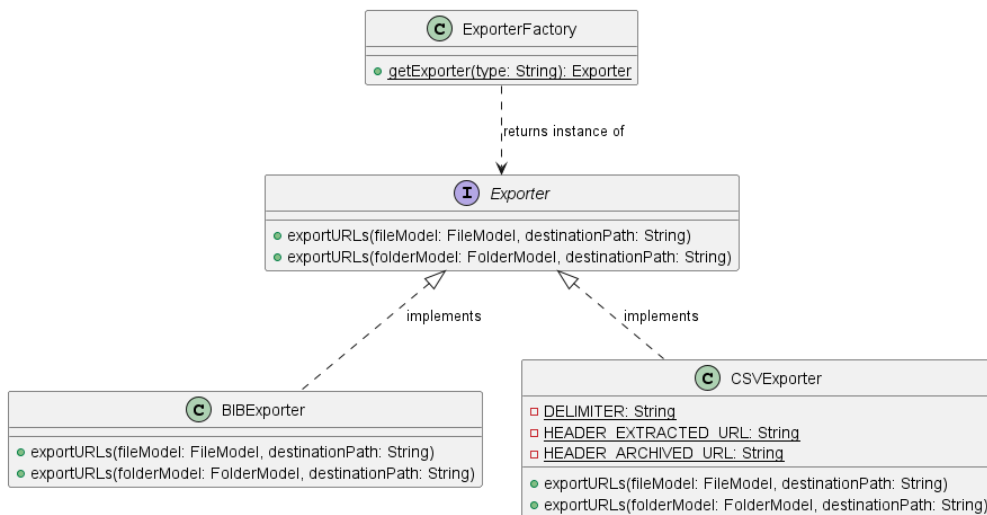


Figure 3.4.: Diagram of the Exporter Factory

This methodology not only streamlines the management of configuration settings, but also increases the adaptability of the application, as configuration changes do not require code-base changes.

An example of the configuration file structure is as follows:

```

{
  "accessKey": "[Access Key]",

```

```
"secretKey":["Secretkey"],  
"browser":"FIREFOX"  
}
```

Users have the flexibility to modify the configuration file at runtime, or edit the file directly. Currently, the application uses this file to store credentials for the Wayback Machine and to select the correct browser for the Archive Today service. This feature underlines the URL-Archiver's commitment to user customisation and operational efficiency.

Adherence to SOLID Principles

The URL-Archiver has been carefully designed with a strong emphasis on SOLID principles, ensuring a robust, maintainable and scalable architecture.

1. **Single Responsibility Principle:** Each class, such as `FileReaderFactory`, `ConfigModel` or `CLIController`, is dedicated to a single responsibility. This principle ensures that each module or class focuses on a single aspect of the application's functionality.
2. **Open/Closed Principle:** The application uses interfaces such as `FileReaderInterface` and factory classes to facilitate extensibility while keeping changes to existing code to a minimum. This approach allows new functionality to be added seamlessly.
3. **Liskov Substitution Principle:** The architectural design allows subclasses (such as specific file readers) to replace their parent class or interface without compromising the integrity of the application, thereby maintaining a robust design.
4. **Interface separation principle:** The design strategy of the URL Archive favours lean interfaces, ensuring that classes implement only the methods essential to their functionality. This is evident in the focused methods of interfaces such as `URLArchiver` and `Exporter`.
5. **Dependency Inversion Principle:** High-level modules such as `CLIController` rely on abstractions rather than concrete implementations. This is emphasised by the use of interfaces and factory classes for object creation, which reduces dependency on specific implementations.

The URL-Archiver's strict adherence to these SOLID principles is a testament to its well-structured, easily maintainable and scalable design. This approach underlines the application's commitment to high quality software engineering practices.

Figure 3.5 shows the complete class diagram of our application. A higher resolution version can be found in the appendix, see A.3) (it can be downloaded via double click).



3.1.3. URL Extraction Process

The extraction of URLs is a crucial aspect of the architecture, as it determines the effectiveness and accuracy of the archiving system. This description outlines the current approach to URL extraction and identifies areas for improvement.

URL Extraction Mechanism

The URL extraction process in the URL-Archiver involves scanning the text content of files and identifying strings that conform to the standard structure of URLs using regular expressions (regex) . This technique is essential for processing and archiving the URLs contained within the documents.

Challenges and Limitations

The current implementation of the URL extractor, which uses regex, struggles to accurately identify certain types of URLs. One particular challenge is URLs that end with text enclosed in parentheses "()". This issue exposes a limitation in the regex pattern's ability to handle complex URL structures. Detecting URLs through regex is a well-known challenge in the field. This is evidenced by the collection found at Mathias Bynens' "In search of the perfect URL validation regex" [1].

The project's time constraints prevented the development of a more optimized regex pattern to address these specific cases. However, this limitation has been acknowledged as an area for future improvement. Enhancing the regex would significantly improve the URL-Archiver's functionality, making it more robust in handling diverse URL formats.

Furthermore, since it is nearly impossible to achieve perfect URL detection through automated means alone, it is recommended to include a feature that allows users to manually modify the extracted URLs within the URL-Archiver. This functionality would add an extra layer of accuracy, enabling users to correct any errors in URL extraction and ensuring the integrity of the archived data.

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.2 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

The URL-Archiver enables the extraction of URLs from any Unicode text or PDF file and allows for interactive archiving on one of the supported archiving services .



The application was designed to be platform-independent . However, it has only been tested on the following systems, so it cannot be guaranteed to work without restrictions on other platforms.

- ▶ Windows 11 (Version 23H2)
- ▶ Windows 10 (Version 22H2)
- ▶ macOS (Ventura)
- ▶ Ubuntu (20.04.3 LTS)

4.2.1. Requirements

To build and start the application, ensure that the following dependencies are installed on your system:

- ▶ Git: Latest stable version recommended.
- ▶ Maven: Version 3.8 or higher.
- ▶ Java: Version 21.

4.2.2. Clone the repository

To clone the repository, run the following command in a terminal:

```
git clone https://github.com/devobern/URL-Archiver.git
```

4.2.3. Build and run scripts

The build and run scripts are provided for Windows (`build.ps1`, `run.ps1`, `build_and_run.ps1`), Linux, and MacOS (`build.sh`, `run.sh`, `build_and_run.sh`). The scripts are located in the root directory of the project.



The scripts need to be executable. To make them executable, run the following command in a terminal:

- ▶ Linux / MacOS: `chmod +x build.sh run.sh build_and_run.sh`
- ▶ Windows:
 - Open PowerShell as an Administrator.
 - Check the current execution policy by running: `Get-ExecutionPolicy`.
 - If the policy is Restricted, change it to RemoteSigned to allow local scripts to run. Execute: `Set-ExecutionPolicy RemoteSigned`.
 - Confirm the change when prompted.
 - This change allows you to run PowerShell scripts that are written on your local machine. **Be sure to only run scripts from trusted sources.**

Windows

Build the application

To build the application, open a command prompt and run the following script:

```
./build.ps1
```

Run the application

To run the application, open a command prompt and run the following script:

```
./run.ps1
```

Build and run the application

To build and run the application, open a command prompt and run the following script:

```
./build_and_run.ps1
```

Linux and macOS

Build the application

To build the application, open a command prompt and run the following script:

```
./build.sh
```

Run the application

To run the application, open a command prompt and run the following script:

```
./run.sh
```

Build and run the application

To build and run the application, open a command prompt and run the following script:

```
./build_and_run.sh
```

4.3. User Manual



To follow the instructions in this section, the application must be built. See 4.2.

The URL-Archiver is a user-friendly application designed for extracting and archiving URLs from text and PDF files. Its intuitive interface requires minimal user input and ensures efficient management of URLs.

4.3.1. Getting Started

Windows

Open Command Prompt, navigate to the application's directory, and execute:

```
./run.ps1
```

Linux / MacOS

Open Terminal, navigate to the application's directory, and run:

```
./run.sh
```

4.3.2. Operating Instructions

Upon launch, provide a path to a text or PDF file, or a directory containing such files. The application will process and display URLs sequentially.

Navigation

Use the following keys to navigate through the application:

- ▶ **o**: Open the current URL in the default web browser.
- ▶ **a**: Access the Archive Menu to archive the URL.
- ▶ **s**: Show a list of previously archived URLs.
- ▶ **u**: Update and view pending archive jobs.
- ▶ **n**: Navigate to the next URL.

- ▶ **q**: Quit the application.
- ▶ **c**: Change application settings.
- ▶ **h**: Access the Help Menu for assistance.

Archiving URLs

Choose between archiving to Wayback Machine, Archive.today, both services, or canceling.

When opting to use Archive.today for archiving, an automated browser session will initiate, requiring you to complete a captcha. Once resolved, the URL is archived, and the corresponding archived version is then collected and stored within the application.

Configuration

Customize Access/Secret Keys and the default browser. Current settings are shown with default values in brackets.

Exiting

To exit, press **q**. If a BibTex file was provided, you'll be prompted to save the archived URLs in the BibTex file. Otherwise, or after saving the URLs in the BibTex file, you'll be prompted to save the archived URLs in a CSV file.

For BibTex entries:

- ▶ Without an existing note field, URLs are added as: `note = {Archived Versions: \url{url1}, \url{url2}}`
- ▶ With a note field, they're appended as: `note = {<current note>, Archived Versions: \url{url1}, \url{url2}}`

5. Conclusion

5.1. Discussion

In the course of our academic endeavor, the URL-Archiver project, we have successfully designed and implemented a Java-based application capable of extracting and archiving URLs from Unicode text and PDF documents. This development demonstrates the practical application of our academic learning in real-world scenarios. The application is intended to assist professionals in research and journalism by providing a streamlined approach to archive URLs and organize them.

Our project's outcome highlights the application's proficiency in fulfilling all requirements and reaching the goal to deliver a FLOSS-licensed, platform-independent Java-program called URL-Archiver. As young professionals in the IT field, our efforts not only gave us valuable hands-on experience, but also contributed to the larger dialogue regarding digital data management.

However, like any academic project, our application is not without its limitations. Currently, the tool has a limited compatibility with certain file formats, which may limit its applicability. Moreover, the necessity for manual captcha solving, while ensuring security, does pose an inconvenience and limits the tool's automation capabilities.

Looking ahead, we have a number of enhancements in mind for the URL-Archiver. Enhancing the application's compatibility with a broader range of file formats would increase its utility. Furthermore, integrating support for additional archiving services, particularly those with accessible APIs, stands as a significant upgrade. This would not only automate interactions with a wider range of archiving solutions but also improve both the utility and user experience of the application.

In conclusion, the URL-Archiver effectively serves its intended purpose. However, there is still room for optimisation. The findings and challenges identified during this project lay the groundwork for future improvements. We believe that with continued research and development, the URL-Archiver can become an even more versatile and valuable tool.

5.2. Bottom Line

5.2.1. Conclusion

The journey of developing the URL-Archiver has been a mix of unexpected challenges and rewarding experiences. Initially assumed to be straightforward, the project revealed its complexity as we delved deeper. This experience highlighted the importance of effective team communication, a critical factor in overcoming the unforeseen challenges of the project. Reflecting on our approach, establishing a more robust project structure from the beginning would have been beneficial. Despite these challenges, the project was rewarding and a great learning experience.

One of the key takeaways was the value of practical experience with Git. This tool not only enhanced our collaborative efforts but also contributed to our personal skill development. Remarkably, the URL-Archiver stands as our first project built entirely from scratch, marking a significant milestone in our journey as developers.

We firmly believe that the URL-Archiver has the potential to be a valuable asset to its users. By significantly reducing the time and effort required for extracting and archiving URLs, the application promises an increase in productivity for its users. This improvement is not just theoretical; it's a practical solution addressing a real need in the digital world.

In conclusion, while the project journey had its ups and downs, the collective learning and the potential impact of the URL-Archiver make it a fulfilling experience. We are optimistic about its utility and look forward to seeing its adoption and evolution in the professional world.

5.2.2. Productivity Increase Calculation

This section calculates the estimated productivity increase for users of our URL-Archiver software. The calculation factors include task duration without and with the software, the resulting time economy, cost per time unit, and overall cost economy.

Assumptions

The following assumptions were made for our calculations:

- ▶ The file contains 60 URLs.
- ▶ Without URL-Archiver, manually finding and copying each URL takes approximately 12 seconds.
- ▶ The URL-Archiver takes an average of 2 seconds to find all URLs in a file.
- ▶ Archiving one URL using Wayback Machine manually takes 60 seconds.
- ▶ Archiving one URL using Archive Today manually takes 90 seconds.

- ▶ With URL-Archiver, initiating the archiving process for one URL in Wayback Machine takes 10 seconds. The actual archiving in the background takes in average 30 seconds.
- ▶ Archiving one URL using Archive Today remains at 90 seconds even with the URL-Archiver.
- ▶ The average hourly wage for a professional in Switzerland is CHF 40.
- ▶ Parallel archiving with Wayback Machine allows for multiple URLs to be archived concurrently, reducing the total time significantly.

Wayback Machine Analysis

Without URL-Archiver:

$$\begin{aligned}
 \text{Task duration without SW} &= \text{URL extraction} + \text{URL archiving} \\
 &= (60 \times 12 \text{ seconds}) + (60 \times 60 \text{ seconds}) \\
 &= 720 \text{ seconds} + 3600 \text{ seconds} \\
 &= 4320 \text{ seconds (72 minutes)}
 \end{aligned}$$

With URL-Archiver:

$$\begin{aligned}
 \text{Task duration with SW} &= \text{URL extraction} + \text{URL archiving} \\
 &= 2 \text{ seconds} + (60 \times 40 \text{ seconds}) \\
 &= 2 \text{ seconds} + 2400 \text{ seconds} \\
 &= 2402 \text{ seconds (40 minutes)}
 \end{aligned}$$

Productivity Increase:

$$\text{Task duration economy} = 72 - 40 = 32 \text{ minutes}$$

$$\text{Cost per time unit} = \frac{40 \text{ CHF}}{60 \text{ minutes}}$$

$$\text{Cost economy} = 32 \times \left(\frac{40 \text{ CHF}}{60} \right) = 21.33 \text{ CHF per document}$$

Archive Today Analysis

Without URL-Archiver:

$$\begin{aligned}
 \text{Task duration without SW} &= \text{URL extraction} + \text{URL archiving} \\
 &= (60 \times 12 \text{ seconds}) + (60 \times 90 \text{ seconds}) \\
 &= 720 \text{ seconds} + 5400 \text{ seconds} \\
 &= 6120 \text{ seconds (102 minutes)}
 \end{aligned}$$

With URL-Archiver:

$$\begin{aligned}\text{Task duration with SW} &= \text{URL extraction} + \text{URL archiving} \\ &= 2 \text{ seconds} + (60 \times 90 \text{ seconds}) \\ &= 2 \text{ seconds} + 5400 \text{ seconds} \\ &= 5402 \text{ seconds (90 minutes)}\end{aligned}$$

Productivity Increase:

$$\text{Task duration economy} = 102 - 90 = 12 \text{ minutes}$$

$$\text{Cost per time unit} = \frac{40 \text{ CHF}}{60 \text{ minutes}}$$

$$\text{Cost economy} = 12 \times \left(\frac{40 \text{ CHF}}{60} \right) = 8.00 \text{ CHF per document}$$

5.3. Future Work

The future vision for the URL-Archiver project includes several enhancements to extend its capabilities and improve the user experience. These potential additions, formulated within our project's limited timeframe, include:

- ▶ Enhancing the URL extraction process to accurately identify complex URL structures and allow manual adjustments by users.
- ▶ Adding more archiving services for a broader reach.
- ▶ Expanding support for various input file types.
- ▶ Implementing a user-friendly graphical interface.
- ▶ Enabling multilingual support for global accessibility.
- ▶ Automatically archiving all URLs in a file for efficiency.
- ▶ Providing more detailed setting options for user customization.
- ▶ Publishing the application in package repositories to simplify installation.
- ▶ Improving the code layout, like breaking up the controller for better clarity.

Although the current URL extraction process is a foundational element of the URL-Archiver's architecture, it can be refined and enhanced. Critical steps towards improving the overall efficacy of the system include addressing the challenges in regex-based URL detection and introducing user-driven URL modification capabilities. Integrating services such as Memento Time Travel would expand our archiving capabilities and provide a wider historical perspective. Adapting the application to handle various file formats like .docx could

increase its relevance in various other settings. The introduction of a graphical user interface is another key area, which would dramatically improve user interaction, making the tool more accessible and engaging, particularly for those less versed in CLI environments. Adding multilingual support is also crucial, as it would break down language barriers, enhancing the tool's usability. The proposed enhancements, such as automatic URL archiving, detailed settings, application publishing in repositories, and code layout improvements like refactoring the controller, collectively aim to not only improve the tool's functionality and user experience but also to ensure better maintainability and ease of future extensions.

Glossary

Archive Today **A web archiving service that stores snapshots of web pages** for preservation and retrieval. ii

BibTeX **Program for the creation of bibliographical references** and directories in T_EX or L_AT_EX documents ii

CLI A **Command Line Interface (CLI)** is a type of user interface navigated entirely using text commands. It allows users to interact with software or operating systems by typing commands into a console or terminal. 2

CSV **CSV (Comma-Separated Values)** is a file format used to store tabular data, such as a spreadsheet or database. Data fields in CSV files are separated using commas. ii

Epic **A large body of work in agile development**, broken down into smaller tasks or user stories. 8

Factory Pattern A **design pattern in software development** used to create objects, allowing interfaces to define object creation but letting subclasses alter the type of objects that will be created. 4

FLOSS **Free/Libre and Open Source Software (FLOSS)** refers to software that is both free in the sense of freedom (libre) and open source. It allows users the freedom to run, study, modify, and distribute the software for any purpose. 2

GUI GUI, or **Graphical User Interface**, refers to a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators, as opposed to text-based interfaces, typed command labels, or text navigation. It simplifies the user experience by employing graphical representations of the commands and actions available. 4

Model-View-Controller (MVC) pattern Model-View-Controller, a **software design pattern** for implementing user interfaces, data, and controlling logic. 4

regex **Regex (Regular Expression)** is a sequence of characters forming a search pattern, primarily used for string matching and manipulation. Regex is used in various programming languages and tools for complex text processing tasks. 36

SCRUM A **framework for agile software development** focusing on iterative progress through sprints and collaborative team efforts. ii

Unicode Text File A **Unicode Text File is a file that uses the Unicode standard** for encoding its content. Unicode enables the representation of a vast array of characters from various scripts and symbol sets, making it suitable for internationalization and localization. 2

URL A **URL (Uniform Resource Locator)** is an internet address that directs to a specific resource, like a webpage. ii

URL-Archiver A **software tool designed to extract, archive, and manage URLs found within digital documents**. It supports various file formats and integrates with services like the Wayback Machine for archiving web pages. ii

Wayback Machine A **digital archive of the World Wide Web**, allowing users to see older versions of web pages. ii

Index

Archiving Services, 4, 7, 15, 32, 38, 42, 45

Configuration File, 32–34

Eclipse Public License v2.0 (EPL-2.0), 37,
53

Export Functionality, 32

Factory Pattern, 4, 5, 32

internet connection, 4

JUnit, 37, 53

Model-View-Controller (MVC) pattern, 4–6,
28, 30, 31

platform-independent, 4, 38

priorities, 3

product goal, 2

regex, 36

Selenium, 6, 32, 53

SOLID Principles, 34

Bibliography

- [1] Mathias Bynens. In search of the perfect url validation regex. <https://mathiasbynens.be/demo/url-regex>. Accessed: 2024-01-15.

A. Appendix

A.1. 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.2. 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

A.3. Class Diagram

[Class Diagram PDF \(Double click to download\)](#)

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.


January 15, 2024



N. Dora



A. Vejseli



K. Wampfler