UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Project Final Paper 2022

Title: Simple Archive Management Tool

Author: Laylaa Varachia

Project Abbreviation: ARCHMAN

Supervisor(s): Hussein Suleman, Abayomi Agbeyangi

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	15
Theoretical Analysis	*0*	*25*	
Experiment Design and Execution	*0*	*20*	
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	15
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

ARCHMAN: Simple Archive Management Tool

Laylaa Varachia
vrclay001@myuct.ac.za
Department of Computer Science
University Of Cape Town
Cape Town, South Africa

ABSTRACT

The Simple DL toolkit provides a means for archivists to create digital libraries that do not require much computational power or maintenance. This enables them to be implemented successfully in low-bandwidth environments. As it stands, the administrative functionality for Simple DL is limited and does not allow the in-place editing of files that are part of an archive. Currently, administrators need to download a file, make the necessary changes, and then re-upload the file, which requires unnecessary time and effort. This report discusses the development of a tool that allows administrators to view and edit comma-separated value (CSV) files containing an archive's metadata directly from their browser. The new tool was developed successfully and subjected to various usability and system tests.

KEYWORDS

Digital Archive, In-place Editing, Data Management, Administrative System

1 INTRODUCTION

Digital archives are collections of online-accessible digital objects such as numerical data, pictures, maps, videos, and audio recordings [11]. In most cases, digital archives are easily accessible and free, making them ideal for preserving history and culture.

In South Africa, initiatives such as the Bleek and Lloyd Collection [26] and Emandulo [1] give the public the opportunity to browse through artifacts that document the language and culture of indigenous communities in Southern Africa. Emandulo is a project belonging to the Archives and Public Culture Center (APC). AtoM [12] was initially used to implement the Emandulo archive but was later replaced by Simple DL [27].

Simple DL is a system that essentially has no software being executed at run-time, making it a 'software-less' solution to implement digital libraries [27]. The digitization of collections is a technically challenging, time-consuming, and potentially expensive process [24]. Archivists had to embrace this if they wanted their collections to be discoverable and used. However, the Simple DL toolkit lives up to its name by making the implementation process quick and easy, requiring very few resources [27].

As it stands, Simple DL has limited functionality for its administrative system [27]. Administrators are unable to change configurations or edit files online. Additionally, the current user interface for the administrative system is unintuitive and difficult to navigate. Figure 1 shows the current administrative interface of Simple DL.

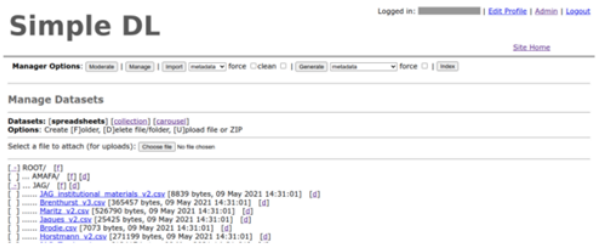

Figure 1: Simple DL administrative interface

1.1 Project Context and Aims

The current administrative system of Simple DL lacks the functionality to edit files through the user's browser. As it stands, the file that requires editing would need to be downloaded, edited, and then re-uploaded. Additionally, external software, such as Microsoft Excel, would be required to edit the file.

There are many established digital archiving tools available; through research, we found Greenstone, DSpace [28], and EPrints [14] to be the most widely used [29]. Unlike Simple DL, these tools have sophisticated ways to edit the metadata of collections without the need for third-party applications. The added functionality of being able to edit the metadata online and through a browser would bring Simple DL closer in functionality to the above-mentioned tools.

The goal of this project is to develop a file editing tool for comma-separated value (CSV) files containing, but not limited to, metadata belonging to digital collections. In a broad sense, the following functionality should be added to the administrative system of Simple DL:

- The ability to open and view files that are already uploaded.
- The feature to edit existing CSV files online through the browser.

This enhancement to the Simple DL toolkit would significantly contribute to making it a more desirable and comprehensive digital archiving tool.

1.2 Solution Outline

Administrators need to manage digital archives effectively, and the improvements made to the Simple DL administrative system

would make this task more pleasant. The improved administrative system developed throughout this project is intended for archive administrators. Potential users could include researchers, historians, students, or anyone interested in curating their own digital archive. These users may have little to no knowledge or experience with digital archiving tools but should be able to successfully navigate their way through the administrative system.

The system consists of three main components:

- *File editor:* Allowing the user to view and edit metadata files in CSV format.
- *File management:* Allowing the user to view, upload and delete files and folders in the archive.
- *Configuration management:* Allowing users to easily edit the system configurations as well as the look and feel of the site.

This paper focuses on the tool developed by the author, namely the file editor.

1.3 Report Structure

Section 2 provides an overview of the research done on digital archives and the in-place [21] editing of files. Section 3 details the requirements gathering process and the design of the system. The implementation of the project based on the requirements is then discussed in Section 4. Section 5 discusses the testing process, which included software, usability, and acceptance testing. Section 6 draws conclusions about the project, followed by future work and acknowledgements in sections 7 and 8, respectively.

2 BACKGROUND AND RELATED WORK

The project commenced with conducting research on existing digital archiving toolkits and becoming familiar with Simple SL. Since this project required building on the Simple DL software and improving it, possessing a certain degree of knowledge regarding digital archiving and having a high-level understanding of Simple DL and how it compares to other digital archiving toolkits was essential. Research was conducted on in-place editing and metadata editing, as well as the tools employed for these purposes.

2.1 Background Information

Simple DL's main operation is the creation of a website from an ingested metadata collection. This process is done in three steps, as detailed below.

Step 1: Importing
XML files are created for metadata that is read in from source XML files and spreadsheets [27]. Based on the original structure of the source directories, a directory structure is created using nesting rules specified in metadata entries by AtoM. Additionally, user profiles are generated automatically in XML format.

Step 2: Indexing
The information retrieval engine is capable of faceted search [16], which allows users to filter search results based on different facets of the data [26]. User files and XML metadata are indexed by the information retrieval engine. The full text is extracted from PDF files and cached when necessary [27].

Step 3: Generating
The files containing metadata, users, and website pages are converted from XML to HTML files using the XSLT stylesheet [27]. The template website is also duplicated. Thumbnails are created for individual items and subcollections.

This sequence of events occurs each time a new metadata subcollection is added to the system. If no changes to a spreadsheet have occurred, it will not be imported again, and an HTML file will not be regenerated from its original XML source if it is up to date.

2.2 Digital Archiving

DSpace [28] and EPrints [14] are well-known open-source toolkits used for building digital repositories. They offer web-based interfaces and databases [27]. DSpace employs a relational database for storing digital artifacts, while EPrints uses the UNIX file system and a MySQL database for data such as user details and metadata [23]. In contrast, Simple DL takes a different approach by using minimal web applications, lacking a formal database management system. It relies on spreadsheets, XML files for structured data, and flat files for unstructured data storage [27].

The Greenstone Digital Library Software from the New Zealand Digital Library Project [32] offers a unique way of organizing and accessing digital artifacts. It can be used through a web browser with internet access, and collections are also available offline on CD-ROMs with browser capabilities. This offline access requires a software installation. Notably, both Simple DL and Greenstone share the common feature of catering to low-bandwidth communities.

The digital archiving toolkits mentioned above lack at least one of the following attributes: offline functionality, simplicity, and low maintenance needs. This gap prompted the development of Simple DL. The Digital Bleek and Lloyd Collection [26], created in 2006, served as a customized solution [14], adhering to principles such as network uncertainty, avoidance of mediation by software systems, and the creation of static representations through data preprocessing. Simple DL was constructed based on these principles, addressing the absence of digital libraries in developing nations and highlighting the necessity for a digital archiving toolkit with these specific characteristics.

2.3 Editing Files

In-place editing, also known as live site editing, in-context editing, or in-situ editing, is a technique commonly used for editing web pages, enabling users to modify content directly; early examples include Sparrow, DirectEdit, and ISAWiki, which, unlike Wikis, allowed content changes without requiring HTML knowledge [21].

The Sparrow [13] in-place editing system focuses on collaborative web pages where users can edit content directly within the browser, enabling independent contributions without relying on the original author. This concept is relevant to Simple DL, but with some

functional differences.

Just as Sparrow users can edit web pages using forms, Simple DL administrators should be able to modify archive files without direct exposure to code. However, the key difference is that Sparrow caters to community-shared web pages, while Simple DL empowers administrators to edit files in CSV format within archives, aligning the in-place editing concept with the needs of digital archiving.

Back-end editing, as opposed to in-place editing [21], involves users interacting with a separate interface to modify web content, often associated with systems like Wikis. A Wiki is a linked set of Web pages that were created collaboratively and can be improved iteratively [30]. Simple DL allows browser-based file editing like a Wiki, though exclusively by administrators, avoiding the need for HTML or coding expertise. Unlike early Wikis, where users required HTML knowledge, the file editing tool prioritizes intuitive editing. Early Wikis directed users to a new interface for editing. In contrast, the approach taken by Simple DL's file editing tool allows a user to view and edit a file from the same interface.

3 DESIGN

3.1 Requirements Gathering

Requirements gathering was conducted with three people from Emandulo. Emandulo was developed by the Five Hundred Year Archive (FHYA) [2] and serves as a platform to bring the past five hundred years of Southern African history together by digitizing scattered material, contextualizing it, and making it available online [27]. The three people from Emandulo who participated in our requirements gathering phase make frequent use of Simple DL to manage archived collections. In particular, they interact with Simple DL's administrative system, making them expert users that could give valuable input.

We reached out to the expert users via email and set up individual meetings with each of them over Google Meet. This approach aimed to ensure that their input into the system's design was unbiased and reflective of their unique perspectives, rather than leading to unanimous agreement among them.

We began the meeting by asking the users what functionality and features they would like to see included in the system. Due to a delay in receiving ethical clearance, a basic version including only some core functionality was already developed at the time of the requirements gathering meeting. The fundamental features recommended by the users aligned with the initial prototype. We agreed to develop it further to include more advanced features that are tailored to the tool's intended use, editing metadata of digital archives.

The prototype could read a CSV file from a specified file path and display it in table format. Users could edit cell content by clicking on individual cells and entering the desired information. Clicking 'save' converted the table's content back into CSV format and stored it in its original directory.

The users expressed the need to insert and delete rows and columns within the table. They also emphasized the need to mark certain fields as mandatory and/or repeatable. Furthermore, some columns should exclusively accept data in predefined formats, which the user should have the option to define.

The expert users agreed that viewing CSV data in a table format was the most intuitive approach. They recommended aligning the design closely with that of Microsoft Excel to ensure familiarity among users when using our editing tool. Navigating a system that is similar to Excel would be easy, even for new users. Regarding the tool's aesthetics, the users advised maintaining a minimalistic appearance with as little clutter as possible, allowing focused and efficient navigation of the data presented.

The feedback was used to refine the minimum feature set as well as identify additional desirable features to potentially include in the future if time permits.

3.2 Non-Functional Requirements

Non-functional requirements were determined based on the needs of the users expressed during the requirements gathering meetings. These included:

3.2.1 Usability. The user interface should be intuitive, and new users should not require more than 15 minutes of learning to become proficient in basic file editing tasks.

3.2.2 Performance. The tool should be real-time responsive when loading, editing, and saving files, with a maximum delay of 2 seconds for any user action [18].

3.2.3 Compatibility. The tool should be compatible with all popular Web browsers.

3.3 System Architecture

To mitigate the risk of a team member failing to complete their tasks, a vertical split of the project ensured that each team member could work independently of one another. Figure 2 demonstrates this modular approach to the system.

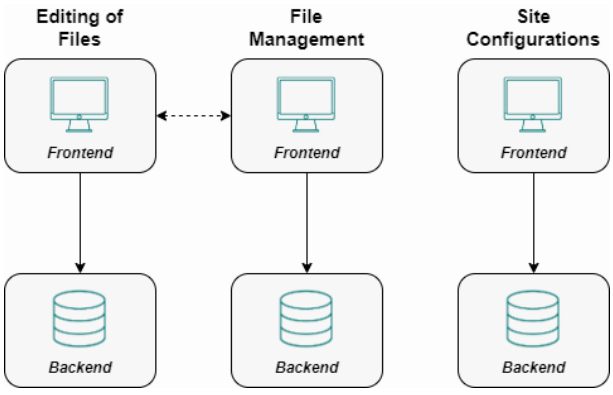

Figure 2: Vertical split of the project

The file editing tool discussed in this paper is a Web application and follows a typical Web application architecture. It is made up of a browser layer, an application layer, a data access layer, and a data layer. Figure 3 shows a high-level view of the system architecture.

3.3.1 Presentation layer. The presentation layer of the file editing tool provides the user interface accessible through standard Web browsers, allowing users to interact with the application to edit CSV files seamlessly. Since all modern browsers adhere to the HTML5 standard [15], these are the browsers that should be supported.

3.3.2 Application layer. This layer is concerned with the logic for processing user actions and managing file edits.

3.3.3 Data access layer. The data access layer enables interaction with the stored data.

3.3.4 Data layer. This layer includes the CSV metadata files being edited as well as any files stored on the server, such as the digital artifacts that make up a collection.

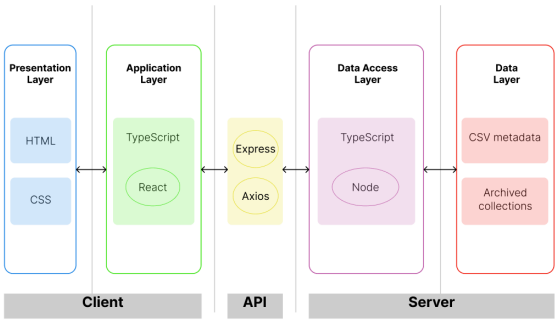

Figure 3: High-level view of system architecture

3.4 Design Artifacts

Based on the requirements gathering phase, Unified Modeling Language (UML) diagrams were drawn. The use case diagram in Figure 4 highlights the most important interactions between the user and the file editing tool. Furthermore, the activity diagram in Figure 5 demonstrates the sequence of actions involved in editing a file. Refer to Figures 13-16 in the Appendix for additional UML diagrams.

4 IMPLEMENTATION

4.1 Software Development Methodology

4.1.1 Agile methodology. An iterative approach was used for the development of this project, which recognizes that software development can be volatile and allows the team to respond to changes that emerge throughout the project. The Agile methodology [10][31] entails dividing the project into phases and emphasizes ongoing collaboration and improvements [8]. The four values of the Agile methodology were adhered to:

- *Individuals and interactions over processes and tools:* The requirements gathered from the expert users were prioritized to ensure that their needs would be met. The users were

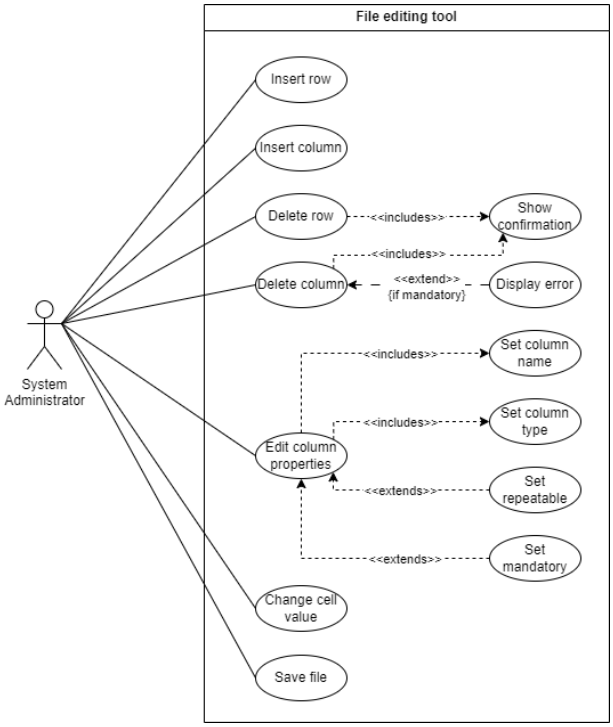

Figure 4: Use case diagram

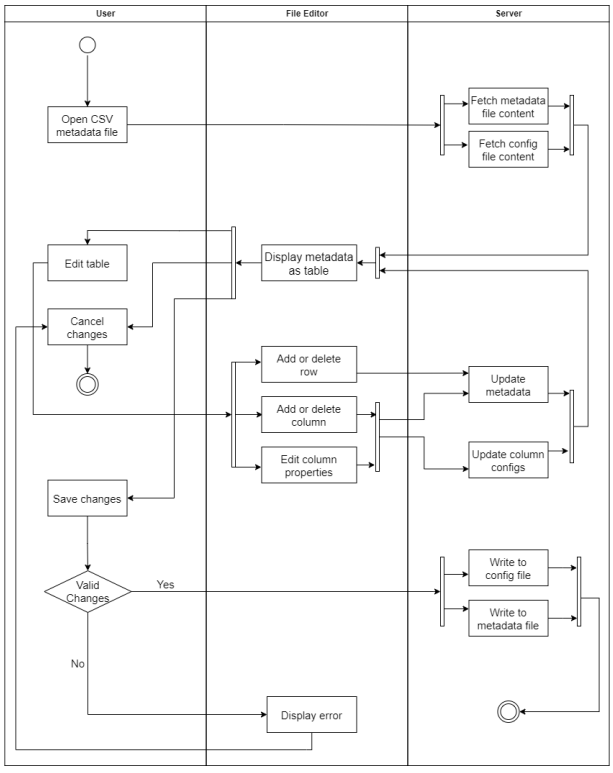

Figure 5: Activity diagram

also involved in the testing process. Additionally, weekly meetings with our supervisor and team members were held to monitor progress being made. These direct interactions allowed for clear understanding and helpful feedback.

- *Working software over comprehensive documentation:* Writing extensive documentation was avoided by creating a prototype, having demos, and ensuring the code had comments. Creating a prototype reduced the need for comprehensive design documents. Demos for the users as well as our supervisor were held instead of writing progress reports. Lastly, prioritizing self-explanatory code with meaningful comments reduced the reliance on extensive documentation explaining the code's functionality. Software testing was conducted to guarantee the correct functionality of the code.
- *Customer collaboration over contract negotiation:* The expert users were contacted frequently during development, maintaining open lines of communication to account for evolving needs. User acceptance testing allowed them to test the functionality of the developed software against their requirements, which ensured that the end product aligned with their needs.
- *Responding to change:* Throughout the development process, requirements and functionality were revised and adjusted based on user feedback. New insights and emergent ideas led to re-prioritizing the backlog. Updating the timeline was also necessary to deal with unexpected circumstances, such as the delay in obtaining ethical clearance.

Implementation of the Agile methodology involved utilizing the Kanban [19] approach. A Kanban board was used to provide a visual representation of the workflow. The board was used to track the progress of work items that moved sequentially from one workflow stage to the next. These workflow stages were *Backlog, Prioritized, In Progress, In Testing, Complete* and *Passed UAT*.

4.1.2 Feature-driven development. A feature list was created for the file editing tool from the inception of the project. The core features were then identified from this list. Development occurred in iterations, beginning with the implementation of these core features.

The initial stage of development involved producing a prototype with minimal styling and functionality. Only the ability to view a file, edit basic cell content, and save the file back to its original directory was possible.

The second stage introduced the functionality allowing the insertion and deletion of rows and columns, as well as the ability to configure the type of input a user would like a column to accept. Some effort was dedicated to refining the user interface, making it visually appealing yet simplistic.

The third and final stage largely focused on small bug fixes as well as incorporating extensive error checking and validation. Other small changes were made in accordance with the feedback received from user testing.

4.2 Technology, Programming Languages, and Environment

The CSV file editing tool was developed in Typescript, using Node.js for the backend and the React library for the frontend. Cascading Style Sheets (CSS) were used for the styling of the site. The Integrated Development Environment (IDE) of our choice was Visual Studio Code.

Node.js is known for its asynchronous and non-blocking nature, producing scalable and efficient server-side applications [6]. On the other hand, React facilitates the quick creation of dynamic and interactive user interfaces [7]. When paired with Typescript, which is a strongly typed language built on JavaScript [3], errors can be caught early, enhancing the reliability and maintainability of the code.

Express [17] was the framework of choice to use with Node.js for the backend due to its simplicity in creating application programming interfaces (APIs). Express provides tools to define routes as well as handle HTTP requests and responses. Axios, the library chosen to use with React for the frontend, serves as a promise-based HTTP client that facilitates requests to a given endpoint [25]. Axios was chosen since it is straightforward, promise-based, and supports common HTTP methods.

Since both Node.js and React can be used with TypeScript, integration is well documented and seamless. This makes it easy to create well-structured websites that result in user-friendly experiences.

Git was used locally for version control, ensuring that changes to the codebase could be tracked, managed, and reverted effectively during the development process.

4.3 Feature Details

Features and components that were implemented as part of the CSV file editing tool are detailed below.

4.3.1 View CSV files. The metadata of archived collections is stored in individual CSV files in specific directories set out in the configurations of Simple DL. To view a CSV file, a GET request is made to an API endpoint, which reads in the specified CSV file and parses its contents using a CSV parsing library. The CSV parser assembles the parsed rows and columns into an array, which gets sent as a JSON response. If an error occurs at this stage, the error message will be logged with an error code of 500, indicating an internal server error. On the frontend, the JSON data is fetched asynchronously from the API endpoint using Axios and stored as an array. Each element of the array is a row of data from the CSV file. Each row is an object that has values stored for every column heading. HTML is then used to map the data to a table and display it. The cells in each column displayed the data using text areas, and the width of the columns was calculated as a function of the average number of characters in that column. Figure 6 shows two columns of a table presenting CSV metadata.

4.3.2 Inserting and deleting rows and columns. Right-clicking on any cell in the body of the table brings up a context menu. From

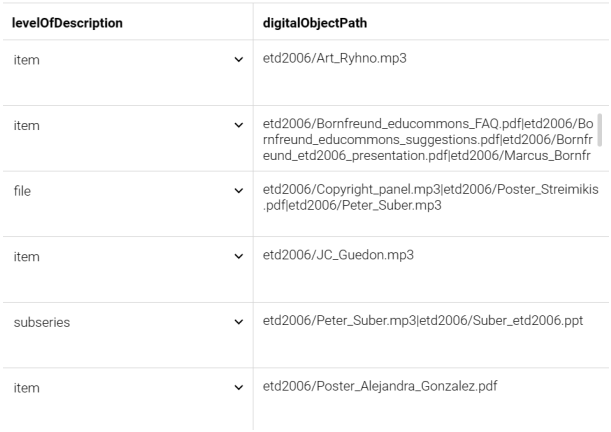

Figure 6: Screenshot of interface

the context menu, a user can choose to insert a row above or below the row that was clicked on. The user can also choose to insert a column to the left or right of the column that was clicked on. Figure 7 shows the context menu that appears when right-clicking in a cell of the table.

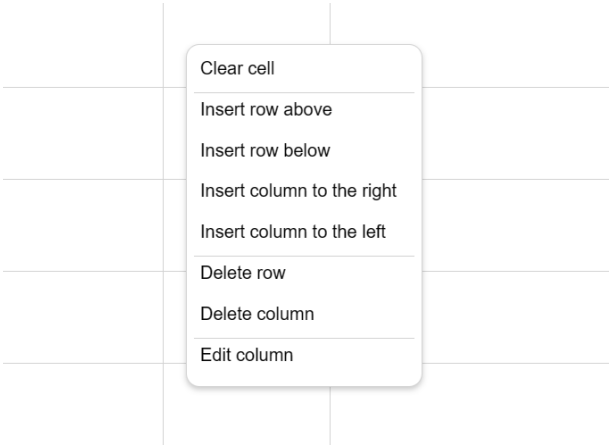

Figure 7: Screenshot of the context menu

Inserting a row is straightforward and is achieved by creating a new entry at the relevant index of the array storing the table data. The new row will contain empty strings for each column heading. Deleting a row is done inversely by deleting the entry at the relevant index of the array.

Inserting a column, as detailed in Section 4.3.3, is slightly more complex since the file editing tools give the user the ability to specify certain properties regarding the input in each column. However, once a user specifies the new column's properties, the column will be created by iterating through the array storing the CSV data and adding the new field to each of the rows. Deleting a column is done inversely by iterating through the array and removing the relevant field and its value from each of the array entries.

4.3.3 Customizing column input. The columns of the table are configurable based on the type of input a user wants to store in that field. The properties of a column are set upon its insertion. The properties of an existing column can be edited by right-clicking and choosing to edit the column. In both cases, a pop-up will appear on the screen, allowing the user to set the desired properties. There are three aspects of a column that can be customized:

- *Mandatory field:* A column can be set to mandatory using a checkbox. A mandatory column cannot be deleted or left blank.
- *Repeatable field:* A column can be set to allow multiple input values using a checkbox. Repeatable values are separated by a configurable character, often the '|' symbol, in the CSV file.
- *Input type:* The type of input values allowed for the cells of a column can be set to one of the four options specified below using radio buttons.
  - *Text:* Allows a user to click into a cell and type.
  - *Drop-down:* Ensures that the value is selected from a drop-down menu of options. These options are specified when inserting or editing a column, before the user populates the column.
  - *File path:* Allows the user to specify a directory when inserting or editing a column. When a user clicks on a cell in that column, a popup containing files in the specified directory lets the user select their desired input for the cell. Figure 8 shows the popup that appears to select files if the input type is set to 'file path'.

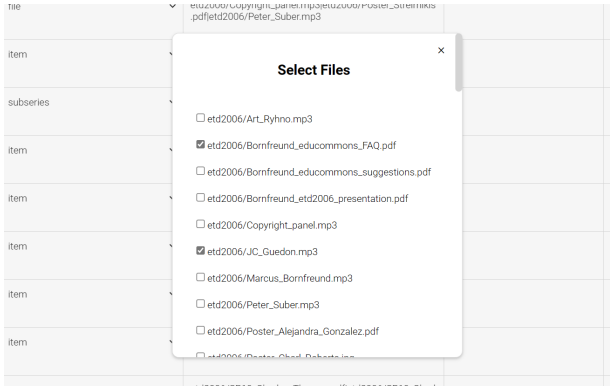

Figure 8: Screenshot of popup to select files

  - *Date:* If a column is set to allow dates as input, the user will be able to select the desired date when clicking on the cell they wish to edit. The date format can be set when inserting or editing a column.

The configurations for the columns of each metadata file are stored in their own CSV file, with each column being specified using its own row. These configurations were read in similar to the method used to read in the metadata file in Section 4.3.1 and stored in an array. Creating a new column inserts a new row in the configuration file. Consequently, deleting a column removes the corresponding column from the configuration file. Editing a column's properties updates the corresponding row in the configuration file. Figures 9

and 10 show the popup menus used to set the properties of a column when inserting a new column and editing an existing column, respectively.

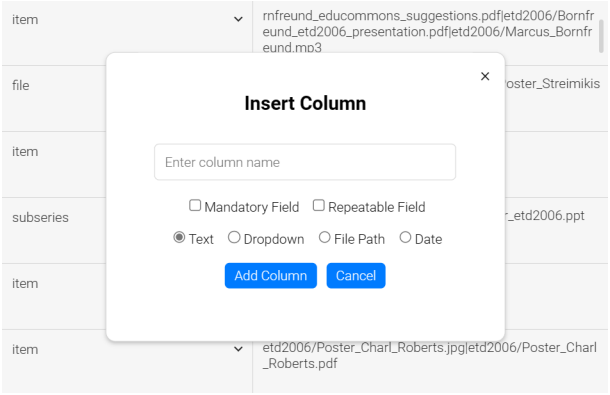

Figure 9: Screenshot of the 'Insert Column' popup

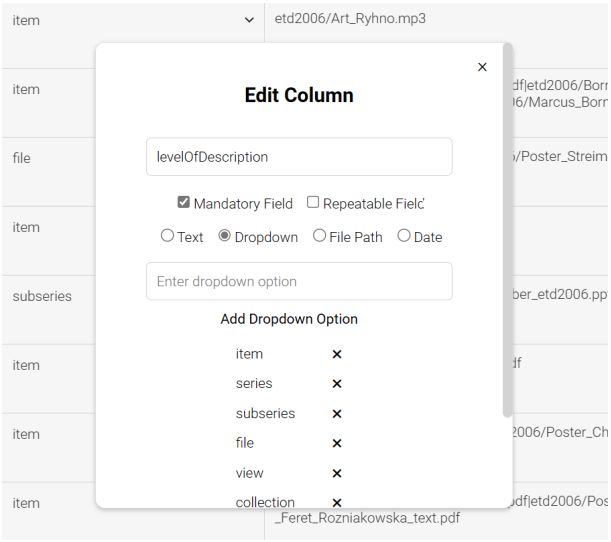

Figure 10: Screenshot of the 'Edit Column' popup

4.3.4 Saving. Once a user makes all the desired changes to the metadata in the table, the 'save' button can be used to store the data back in its original directory. The data in each cell is first cleaned by removing any leading or trailing spaces. This is followed by error checking and validation. If there are no errors, the data is sent as a POST request using Axios to the respective Express API endpoint. The data is received as JSON data, and a CSV string is constructed. The header of the CSV string is based on the keys of the objects in the array. Finally, the CSV data is written back to its original directory. An alert indicates to the user whether the data was saved successfully or if an error occurred while saving.

5 TESTING

After stage 2 of development, we began user acceptance testing and the first round of usability testing. The second round of usability testing occurred after stage 3 of development. Meetings with users were done virtually using Google Meet.

5.1 System Testing

5.1.1 Compatibility testing: All browsers adhere to the HTML5 standard [15], but there are slight differences in the way an application can look between different browsers. This is usually due to differences in the interpretation of CSS, but it could also be due to the rendering engine being used, JavaScript interpretation, or operating system differences. The free online tool LamdaTest [5] was used to ensure that all aspects of the application functioned as intended across various browsers and operating systems.

We tested the application for the latest three versions of each browser. While there were slight differences in appearance, no visual or functional inconsistencies that affected the user experience were identified from this process.

5.1.2 Performance testing: To measure the performance of the file editing tool, ChromeDevTools [9] and Lighthouse [4] were used. All significant user actions were measured to be under 2 seconds, which satisfied our non-functional requirement regarding performance. Figure 11 provides a performance summary for the initial loading of the Web page, in which all significant processes took a total of 320 ms.

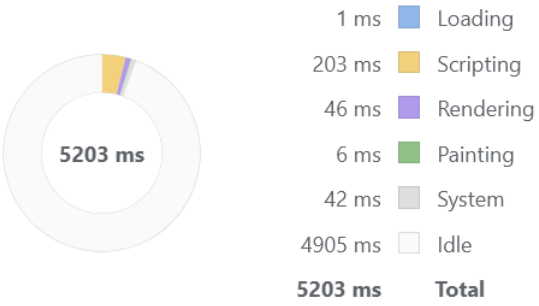

Figure 11: ChromeDevTools performance summary

The opportunities to minify JavaScript and reduce the amount of unused code were identified using the performance metric on Lighthouse. Performance is measured as a score between 0 and 100 by taking into account five factors [22]:

- *First contentful paint:* The time taken before a user sees the first image or text.
- *Largest contentful paint:* The time taken to load the largest component of the website.
- *Total blocking time:* The amount of time a page is blocked from user input.
- *Cumulative Layout Shift:* Measures the changes in page layout brought on by the user.

- *Speed Index:* How quickly a page's content gets loaded.

The file editing tool achieved a performance score of 86 after implementing recommendations from Lighthouse to improve its performance. Figure 12 shows the resulting performance scores from Lighthouse.

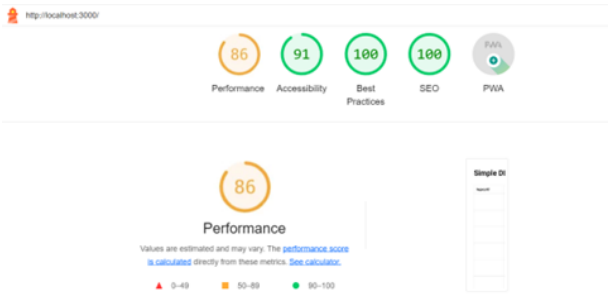

Figure 12: Performance results from Lighthouse

5.1.3 User acceptance testing: Acceptance testing was used to ensure that all the functional requirements were met. A cognitive walk-through of each requirement was done to ensure that all the necessary functionality was implemented and worked as expected. Table 1 summarizes the results of the acceptance testing.

Table 1: User acceptance testing results

Functionality	Result
View a CSV file	PASS
Insert a row	PASS
Delete a row	PASS
Insert a column	PASS
Delete a column	PASS
Edit properties of column	PASS
Clear cell	PASS
Edit cell content	PASS
Change column name	PASS
Save CSV file	PASS

5.2 Usability Testing – Iteration 1

5.2.1 User selection. We contacted the same expert users from Emandulo who participated in our requirements gathering phase. Out of the 3 users contacted, we received 2 responses. Both expert users that participated are archivists and content managers from the Five Hundred Year Archive (FHYA) who have experience managing archives built using Simple DL.

5.2.2 Procedure. The evaluation was carried out using Nielsen's 10 Usability Heuristics [20] to help uncover usability issues and enhance user satisfaction. We went through the heuristics one by one and asked the users to provide a severity rating for each of them according to Table 2.

In cases where a heuristic was severely violated, users were asked to detail the problem and propose potential solutions. Users were also offered the opportunity to provide comments on each heuristic during this evaluation process.

Table 2: Severity rating scale

Problem type	Severity	Description
No problem	0	This is not a usability problem at all.
Cosmetic	1	Need not be fixed unless extra time is available.
Minor	2	Fixing this should be given low priority.
Major	3	Fixing and should be prioritized.
Catastrophe	4	Imperative to fix this problem.

5.2.3 Results. Feedback received from the evaluators is listed below for each heuristic.

- *Visibility of system status:* The users all rated this a 0 and felt as if they did not have to wait for feedback due to the responsiveness of the system.
- *Match between the system and the real world:* No usability problems were identified for this heuristic. One of the users specifically mentioned that they liked the familiarity the system had with Excel and felt as if the data was presented logically.
- *User control and freedom:* Both users rated this a 2. The users felt that in addition to 'cancel' buttons on some of the popup menus, there should be an 'x' in the top right corner to close the popup faster since some popups required the user to scroll down first to get to the cancel button.
- *Consistency and standards:* One user gave a low score of 1, while the second user gave a severe score of 4 for this heuristic. The reason for the 4 was that if a column's input type is set to 'file path' the user gets to pick input values for a cell from a popup by selecting the relevant files using checkboxes. If more than one file is selected in the popup, the cell's value is set automatically to the selected files, separated by the '|' symbol. However, for columns that are set to handle 'text' input, the user would need to manually type the '|' symbol if they wanted to give a cell multiple values. The user felt that this was a major inconsistency.
- *Error prevention:* One user gave this a 1 and the other gave it a 2. The 2 was due to the lack of a confirmation option given before saving a file. The user's concern was that it would be easy to overwrite data by mistake if you saved a file unintentionally.
- *Recognition rather than recall:* No usability issues were found for this heuristic.
- *Flexibility and efficiency of use:* No usability issues were found for this heuristic. Users pointed out that making use of more complex features like customizing columns could be used by more experienced users, while novice users could leave all columns set to the default properties and still edit files effectively.
- *Aesthetic and minimalist design:* Both users gave this a 2 and thought the design could be cleaned up to make the table look neater.

- *Help users recognize, diagnose, and recover from errors:* Both users gave this heuristic a 4. Users mentioned that they would like to see a lot more extensive error checking and validation.
- *Help and documentation:* One user gave this heuristic a 0 while the other user gave it a 2 and suggested that short descriptions, especially for the column editing and column creation popups, would be helpful for first-time users who do not understand what properties they are setting.

5.2.4 Discussion: This was the first round of usability testing conducted with the file editing tool. It proved to be beneficial in identifying usability issues and helping to prioritize these issues for further development. The users analyzed the system in depth and provided valuable insights into the improvements that could be made. The majority of their suggestions were implemented during Stage 3 of development. Figures 17-21 in the Appendix show how the interface looked for this iteration of testing.

5.3 Usability Testing – Iteration 2

5.3.1 User selection. Once again, the three expert users from Emandulo were contacted. This time, all three users participated in the evaluation. The additional expert user for this round of testing worked in the Digital Libraries Laboratory at the University of Cape Town (UCT) and has experience in creating digital archives using Simple DL.

5.3.2 Procedure. The same procedure as round 1, using Nielson's 10 Usability Heuristics [20], was followed.

5.3.3 Results. Feedback received from the evaluators is listed below for each heuristic.

- *Visibility of system status:* No usability problems were identified for this heuristic.
- *Match between the system and the real world:* All three users rated this heuristic a 0, and no usability issues were found.
- *User control and freedom:* Two users rated this a 0. The third user gave it a 1. While all users felt as if they could easily back out of processes, the third user suggested that when a popup is open, clicking on the screen behind it should close the popup, which makes it easier to back out of processes.
- *Consistency and standards:* All users gave this heuristic a 0. The user who had previously given it a 4 felt as if the consistency had been improved throughout the system.
- *Error prevention:* All three users gave this heuristic a score of 0. Two users pointed out and showed appreciation for the confirmation messages before deleting rows and columns.
- *Recognition rather than recall:* The three users each gave this heuristic a 0. One user mentioned that all menus and popups were easily retrievable when needed.
- *Flexibility and efficiency of use:* No usability issues were found for this heuristic.
- *Aesthetic and minimalist design:* All users gave this heuristic a 0. The two users that participated in the first round of testing said they liked the new user interface because it looked cleaner.
- *Help users recognize, diagnose, and recover from errors:* Two users acknowledged that error messages were presented and gave this heuristic a 0. However, one user mentioned that some error messages could be more specific and decided to score this a 2.
- *Help and documentation:* Two users found no usability issues. The third user suggested including tooltips to specify the type of input a cell can handle and gave this heuristic a 1.

5.3.4 Discussion. This was the second round of usability testing. The severity ratings for each of the heuristics decreased, and no major usability issues were found. For heuristics that scored severity ratings greater than 0, small adjustments or additions to the code can be made to address the identified issues and improve the overall quality and functionality of the system. Figures 6-10 in Section 4.3 show how the interface looked for this iteration of testing.

6 CONCLUSIONS

This paper addressed a gap in the functionality of the Simple DL toolkit, focusing on enhancing the administrative system's capabilities for digital archivists. The existing limitation in the system, being the inability to edit a collection's metadata, was recognized and addressed through the development of a new file editing tool. The tool provides administrators with the capability to seamlessly view and edit metadata files stored in CSV format online and through the browser.

By enabling in-browser editing of CSV files, the enhanced administrative system minimizes the need for downloading, editing, and reuploading files, saving valuable time and effort for archivists. This development brings Simple DL closer to established digital archiving tools by providing similar editing functionalities.

Acceptance testing ensured that the newly developed file editor for the Simple DL toolkit met its intended functional requirements and performed as expected. Usability testing involving three expert users provided valuable feedback that contributed to the tool's success. By giving us valuable insights and assisting in the refinement of the tool, usability testing ensured that the tool provided a pleasant user experience.

Our results suggest that both functional and non-functional requirements were met. The successful development and implementation of the CSV file editor not only address existing limitations but also pave the way for Simple DL to become a more comprehensive and user-friendly digital archiving toolkit.

7 ACKNOWLEDGEMENTS

Thank you to our supervisor, Prof. H. Suleman, and co-supervisor, Dr. A. Agbeyangi, for their guidance and feedback throughout the project. The time and contributions of the archivists from Emandulo who assisted in our requirements gathering and testing phases are also greatly appreciated.

REFERENCES

[1] 2022. EMANDULO — emandulo.apc.uct.ac.za. http://emandulo.apc.uct.ac.za/. [Accessed 12-09-2023].
[2] 2023. About - Five Hundred Year Archive — fhya.org. https://fhya.org/about. [Accessed 12-09-2023].
[3] 2023. JavaScript With Syntax For Types. — typescriptlang.org. https://www.typescriptlang.org/. [Accessed 12-09-2023].

[4] 2023. Lighthouse overview - Chrome Developers — developer.chrome.com. https://developer.chrome.com/docs/lighthouse/overview/. [Accessed 13-09-2023].

[5] 2023. Next-Generation Mobile Apps and Cross Browser Testing Cloud | LambdaTest — https. https://www.lambdatest.com/. [Accessed 12-09-2023].

[6] 2023. NodeJS Tutorial | Learn NodeJS - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/nodejs/. [Accessed 12-09-2023].

[7] 2023. What is React — w3schools.com. https://www.w3schools.com/whatis/whatis_react.asp. [Accessed 12-09-2023].

[8] Atlassian. 2023. What is Agile? | Atlassian — atlassian.com. https://www.atlassian.com/agile. [Accessed 12-09-2023].

[9] Kayce Basques. 2017. Chrome DevTools - Chrome Developers. https://developer.chrome.com/authors/kaycebasques/. [Accessed 12-09-2023].

[10] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. 2001. Manifesto for agile software development. (2001).

[11] Cheryl Mason Bolick. 2006. Digital archives: Democratizing the doing of history. *International Journal of Social Education* 21, 1 (2006), 122–134.

[12] Jessica Bushey. 2012. ICA-AtoM: Open-source software for archival description. *Archivi & Computer* 22, 1 (2012), 10–25.

[13] Bay-Wei Chang. 1998. In-place editing of Web pages: Sparrow community-shared documents. *Computer Networks and ISDN Systems* 30, 1-7 (1998), 489–498.

[14] Stevan Harnad. 2001. The self-archiving initiative. *Nature* 410, 6832 (2001), 1024–1025.

[15] Ian Hickson. 2011. HTML5 — w3.org. https://www.w3.org/TR/2011/WD-html5-20110405/. [Accessed 12-09-2023].

[16] Jonathan Koren, Yi Zhang, and Xue Liu. 2008. Personalized interactive faceted search. In *Proceedings of the 17th international conference on World Wide Web*. 477–486.

[17] Azat Mardan. 2014. *Express. js Guide: The Comprehensive Book on Express. js.* Azat Mardan.

[18] Fiona Fui-Hoon Nah. 2004. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology* 23, 3 (2004), 153–163.

[19] Gillian M Nicholls, Neal A Lewis, and Ted Eschenbach. 2015. Determining when simplified agile project management is right for small teams. *Engineering Management Journal* 27, 1 (2015), 3–10.

[20] Jakob Nielsen. 1994. Usability Heuristics for User Interface Design.

[21] Brook Jesse Novak. 2010. *Seamlessly Editing the Web.* Ph. D. Dissertation. The University of Waikato.

[22] Tushar Pol. 2023. Google Lighthouse: What It Is How to Use It — semrush.com. https://www.semrush.com/blog/google-lighthouse/. [Accessed 13-09-2023].

[23] Mayukh Sarkar and Sruti Biswas. 2020. Exploring Archives Space: An Open Source Solution for Digital Archiving. *Desidoc journal of library & information technology* 40, 5 (2020), 272–276.

[24] Stephanie A Schlitz and Garrick S Bodine. 2012. The Martha Berry Digital Archive Project: A Case Study in Experimental pEDagogy. *Code4Lib Journal* 17 (2012).

[25] Ujjawal Shrestha. 2020. Implementing API in ReactJS. (2020).

[26] Hussein Suleman. 2007. Digital libraries without databases: The bleek and lloyd collection. In *International Conference on Theory and Practice of Digital Libraries*. Springer, 392–403.

[27] Hussein Suleman. 2021. Simple dl: A toolkit to create simple digital libraries. In *Towards Open and Trustworthy Digital Societies: 23rd International Conference on Asia-Pacific Digital Libraries, ICADL 2021, Virtual Event, December 1–3, 2021, Proceedings 23*. Springer, 325–333.

[28] Robert Tansley, Mick Bass, David Stuve, Margret Branschofsky, Daniel Chudnov, Greg McClellan, and MacKenzie Smith. 2003. The DSpace institutional digital repository system: current functionality. In *2003 Joint Conference on Digital Libraries, 2003. Proceedings.* IEEE, 87–97.

[29] Shahkar Tramboo, SM Shafi, Sumeer Gul, et al. 2012. A study on the open source digital library software's: special reference to DSpace, EPrints and Greenstone. *arXiv preprint arXiv:1212.4935* (2012).

[30] Christian Wagner. 2004. Wiki: A technology for conversational knowledge management and group collaboration. *Communications of the association for information systems* 13, 1 (2004), 19.

[31] Aaron Watters, James C Ahlstrom, and Guido Van Rossum. 1996. *Internet programming with Python.* Henry Holt and Co., Inc.

[32] Ian H Witten, Stefan J Boddie, David Bainbridge, and Rodger J McNab. 2000. Greenstone: a comprehensive open-source digital library software system. In *Proceedings of the fifth ACM conference on Digital libraries*. 113–121.

APPENDIX

A UML Diagrams

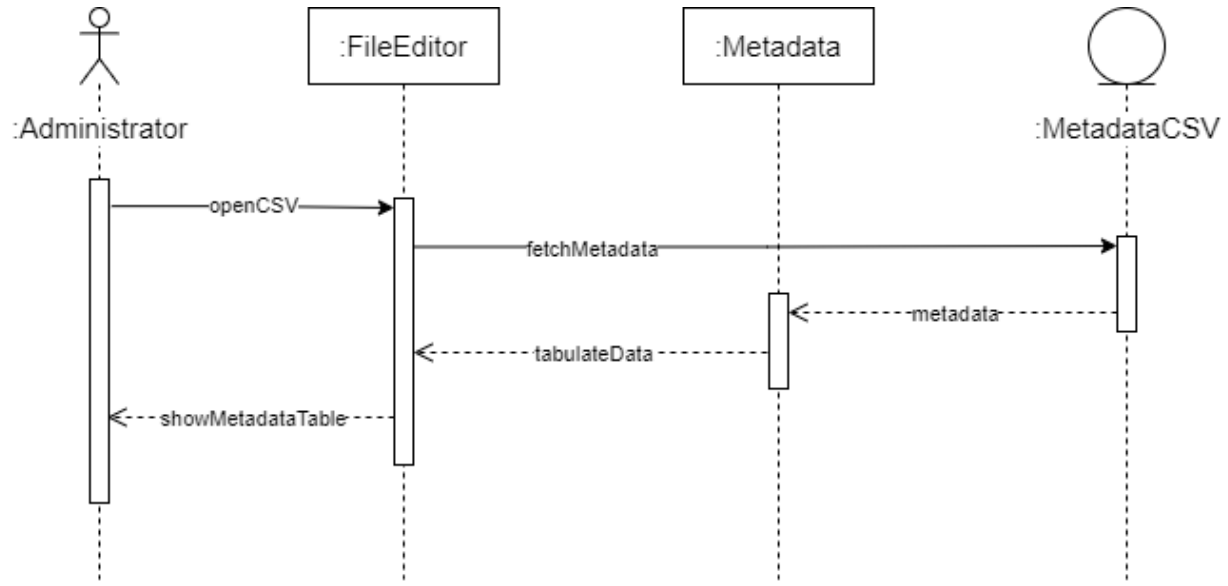

Figure 13: Sequence diagram showing the interactions that take place to display the metadata

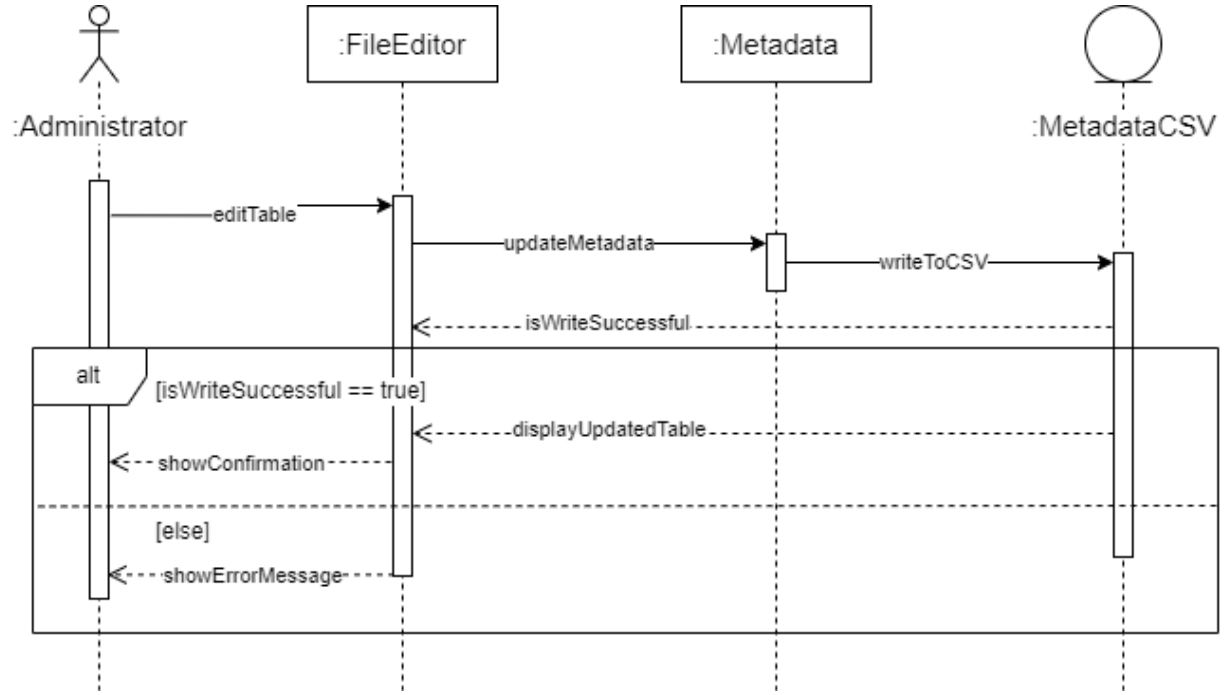

Figure 14: Sequence diagram showing the interactions that take place to edit the content of the metadata table

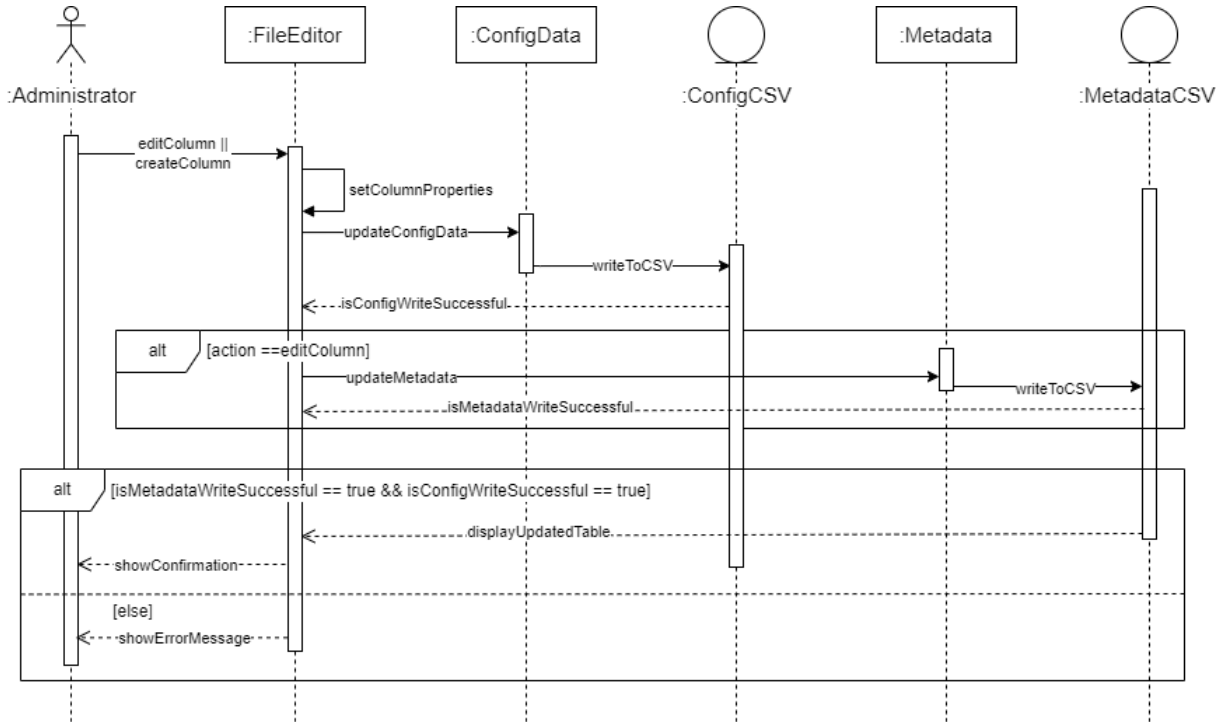

Figure 15: Sequence diagram showing the interactions that take place to create or edit a column of the metadata table

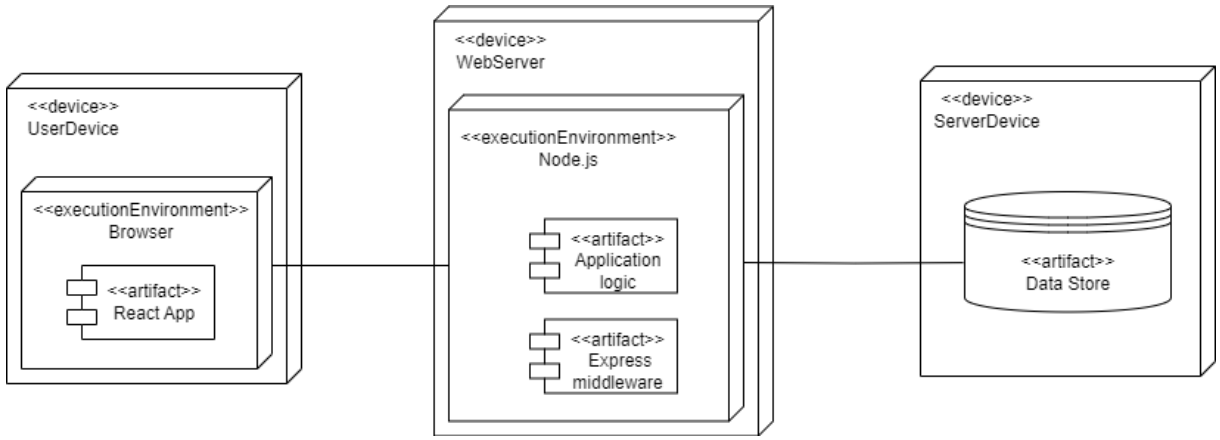

Figure 16: Deployment diagram of the file editing tool

B Initial Interface

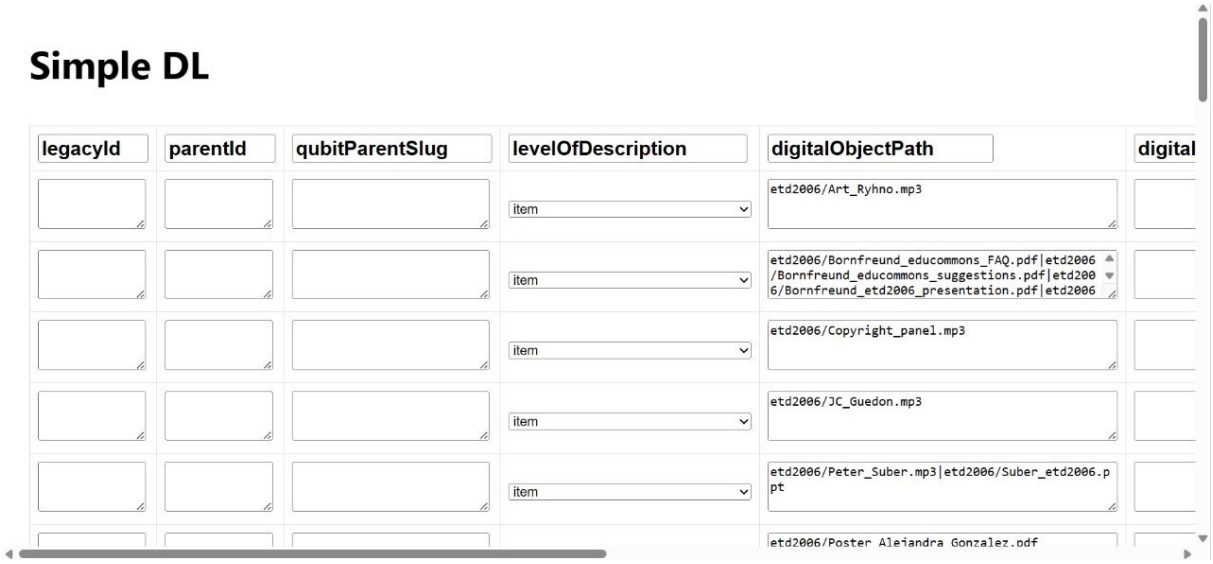

Figure 17: Screenshot of initial interface

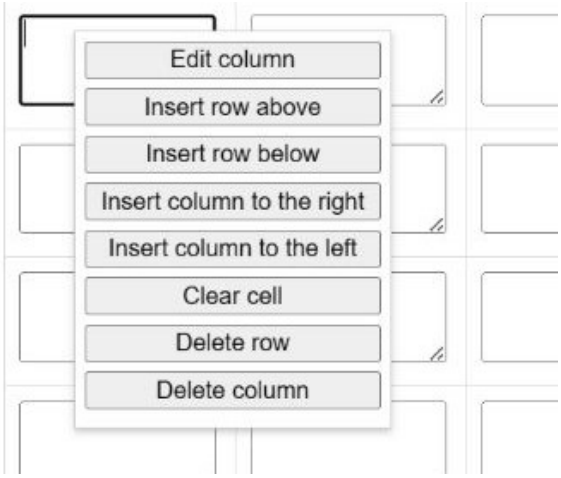

Figure 18: Screenshot of initial context menu

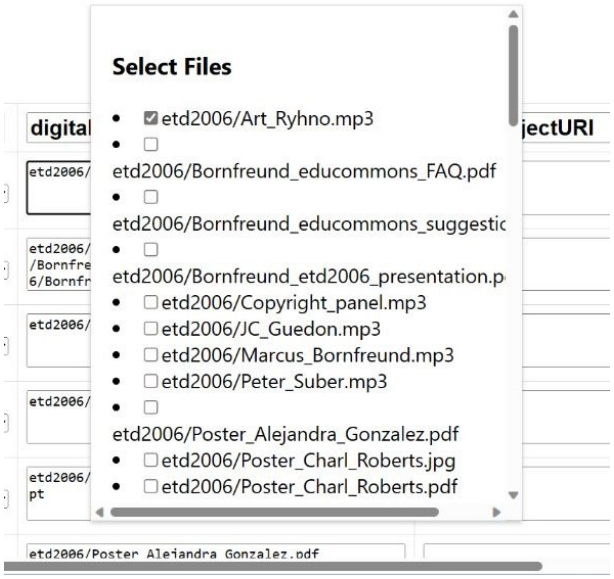

Figure 19: Screenshot of initial popup menu to select files

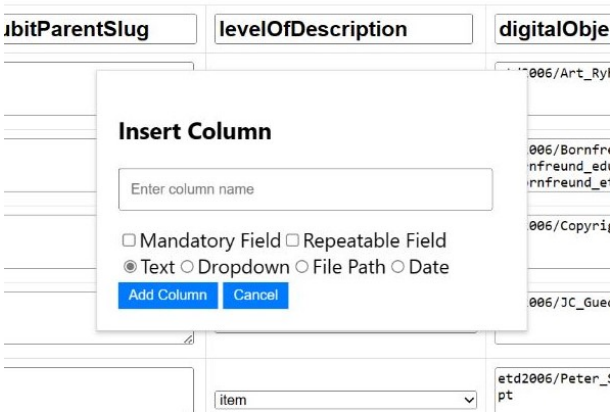

Figure 20: Screenshot of the initial 'Insert Column' popup

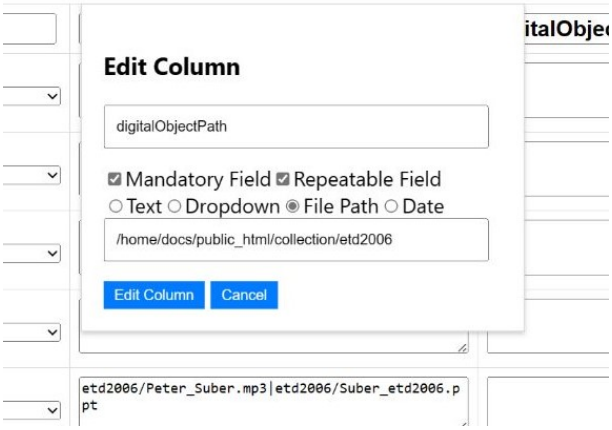

Figure 21: Screenshot of the initial 'Edit Column' popup