UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Project Final Paper 2022

Title: ARCHMAN

Author: Rushil Vallabh

Project Abbreviation: ARCHMAN

Supervisor(s): *Professor Hussein Suleman (supervisor) and Dr. Abayomi Agbeyangi (co-supervisor):*

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	15
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
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

Configurations

Rushil Vallabh
vllrus002@myuct.ac.za
University Of Cape Town
Cape Town, South Africa

1 ABSTRACT

With a vast number of users using a digital repository system and those who have administrative privileges over the configurations of a system, it is important to create a uniform user-friendly interface for configurations that all users can understand and use. SimpleDL currently has no interface for configurations. This paper discusses the development of a web application interface that presents the configurations at a high level and provide functions to perform tasks such as adding new formats of collections to upload and changing directories of main files without editing the configuration file directly. The interface as well provides options to the user to change the look of a digital repository being the SimpleDL site. Functions were created for each configuration on the web application interface to reduce the amount of human error and for users with without a coding background to use. The development of the system involved multiple iterations and was evaluated at each stage using numerous usability testing and software testing.

2 INTRODUCTION

Digital repositories are described as tools to manage and store digital data [14]. Putting similar content into a single institutional digital repository allows users to preserve the content and gain the most use from it from a single location through functions such as search and retrieval. Digital repositories store items such as multimedia content, historical artefacts, and research material. All digital content is stored with the appropriate metadata. The metadata used is based on the functional requirements of a system, which can be established by looking at the requirements of an end user [17]. The content of the system is usually added to the repository by the creator of the digital artefact or by a third party [31]. The repository architecture is required to manage the content and provide baseline features such as search and retrieval.

Digital configurations of a system include all settings for the setup of a digital repository, such as which formats are allowed to be uploaded, paths for main directories that include CSS files, and which metadata can be searched for for all artefacts in the repository. A well-constructed interface gives admin users the ability to edit configurations of the repository with ease without extensive knowledge of how the system works. However, configuration and installation interfaces in earlier versions digital repositories were not a focus point for end users [13]. Administrators had to configure digital library tools themselves. This results in poor usability of administrative mechanisms making the system prone to many errors when trying to edit the configurations.

SimpleDL is a digital repository that is designed to work in low resource environments [24]. It is made to be simple and allows for long term access to the libraries when there is no current preservation and when a power failure occurs. Both a database and a database management system are absent. Flat files are used to store unstructured data, and XML files are used to store structured data. Incorporating a metadata collection and producing a webpage representation of the information are the primary functions of SimpleDL. Three phases are used to do this, each of which corresponds to a system application or script. Import, indexing, and generation are steps. Individual target XML files are created for metadata, index, and generation during import, which is the reading of metadata from source XML and spreadsheets. All user files and XML information are indexed for a data retrieval system that allows faceted searching. Applying the XSLT stylesheet causes all XML files (metadata, users, website pages, etc.) to be generated to be translated to HTML.

As it stands, SimpleDL's administrative system only offers a few features. Online file editing or configuration changes are not permitted by the interface. A file contained in an archive must first be downloaded, then it must be modified using outside software, and then it must be re-uploaded. Additionally, the administration system's existing user interface is confusing and challenging to operate. The administrative system of SimpleDL needs to have its existing functionality expanded. The ARCHMAN project team presents a solution of a revamped administrative system that is simple to use. The team consists of three members, and this report covers the configurations of the system. Fellow group members handle managing of files and editing of files online.

The current system configurations of SimpleDL exist in a perl, xsl and css files, with no interface on the frontend to view or change these configurations. The only way to access and edit the configurations is from the terminal or by opening the perl file itself. Editing this file directly may result in errors if the user does not have the adequate knowledge needed. Errors such as removing an important directory path will result in the CSS style sheets not being rendered and the site displaying a cached version of the SimpleDL site. Configurations can be complex to setup,and without proper usability techniques used to design the interface, users are restricted to how they can edit the configurations without adequate knowledge.

2.1 Solution outline

The development of a user-friendly interface that provides functionality to directly edit the configurations in the perl, xsl and CSS files. The interface will present the configuration items in a visually structured and organised manner. Users will interact with a

graphical depiction of the configurations rather than directly with the code, which makes it simpler to comprehend the relationships between various options. Each configuration will be task-oriented and align with familiar configuration changes such as changing directory paths, interchanging different lists of fields for browse and sort, and transforming the look and feel of the SimpleDL site.

This paper focuses on introducing a new interface for administrative configuration that is both intuitive and user-friendly while ensuring that the existing functionality is not changed and work as intended.

2.2 Report structure

The structure of the paper consists of first discussing related work and literature. System design discusses requirements gathering and system architecture. System implementation discusses implementation, development platform and implemented components of the system for both system configurations and UI configurations. The testing section breaks down which tests were performed on software and for usability testing as well as results and discussions. Conclusions cover future work and concluding comments.

3 RELATED WORK

For building an interface for administrative configuration tasks for a digital repository, there are many designs and reference models for digital repositories to be able to do so. Using these designs and models depends on the needs of the system and what configuration tools and properties are best suited.

3.1 DSpace

DSpace is specifically created for the preservation and management of research data, offering long-term access and visibility [20]. The system uses a web-based interface that caters to both user and administrator needs. Its user-focused screen enables searching and submission, while the administrator's screen allows system management [25]. DSpace's interface also supports multiple community and collection home pages, improving organization and accessibility. The inclusion of Dublin Core metadata within items improves accessibility, though it requires proper authorization configurations. DSpace offers a customizable workflow for various user needs, and its import/export features further enhances data management. [9].

3.2 Fedora

Fedora presents a framework for creating distributed digital repositories, with a emphasis on flexible architecture [9]. This architecture is beneficial for developing digital library systems and provides web services that facilitate dissemination. Fedora's APIs are described using the Web Service Descriptive Language (WSDL), showcasing their extensibility [21]. The system enables different presentations of objects through disseminations, and its design centers around templates for data objects, metadata, and links to software services. Services described by behavior objects grant users access to data objects, with metadata and operational information stored in behavior objects. Fedora's open APIs, available as web services, empower users to generate and manage digital content.

3.3 Greenstone

Greenstone is tailored to function effectively in low-bandwidth environments through a local web server setup. Importing digital items is simplified using configurable plugins [23]. It is particularly designed to be user-friendly for non-specialists, allowing them to compile digital objects into collections. Documents undergo a conversion process into Greenstone Markup Language, similar to HTML. The repository's structure includes separate directories for collections, each containing subdirectories for various purposes [28]. Unique object identification using hashing ensures mirroring data has no unintended consequences. Greenstone's suitability for searching and indexing is highlighted, especially for non-textual content like video and audio collections [18].

3.4 SimplyCT

Designed with a focus on low infrastructure and maintenance costs, SimplyCT offers a web-based Digital Repository System (DRS) built on an existing file hierarchy [10]. This system supports search and browse functionalities, and an administrator section adds control over access options. Drawing from past Digital Library Systems (DLSes), SimplyCT's design prioritizes automatic configuration based on data, eliminating the need for manual adherence to required structures [22]. Basic file access suffices due to low infrastructure requirements, and data preservation is ensured through directory copying. Searching relies on metadata and annotations, with no fixed services. Instead, peer subcollections contribute to building layers of meaning on stored digital objects.

3.5 Discussion on literature

For small libraries, SimplyCT is more suited as it is a repository with low maintenance requirement and minimum infrastructure. It places emphasis on a simple architecture and file organisation. Greenstone, on the other hand, concentrates on indexing and searching and is built for limited resources. It imports digital items via a programmable plugin and employs hashing to create distinctive identifiers for material. Its preservation component is weaker than SimplyCT's. Because it provides a flexible workflow, DSpace is ideal for big organisations. It uses the Dublin Core to facilitate easy access to digital artefacts. Large organisations can also employ Fedora, which outperforms DSpace by using a disseminator for digital assets. Overall, user-friendly interface configurations need more research because they are not the main focus of digital repositories.

4 PROJECT DESIGN

4.1 Requirements gathering

4.1.1 Process. Requirements gathering first took place by performing a careful review of the existing code base and the SimpleDL site. The requirements for this project were conducted with Emandulo users, our clients. Emandulo is an experimental system that discovers dispersed material about Southern African history and collects it in one place [24]. The purpose of this digital repository is to make the artefacts searchable for educational and researcher purposes. The Emandulo site makes use of SimpleDL as a digital repository.

The requirements gathering took place online via Google Meet with three expert users. Questions were established prior to the meeting to gain insight about the current configurations, what they would want in a new interface and what additional features participants would want to be in included in the system. Participants were shown a demo of what work had already been done due having a demonstration with our first and second reader at an early stage of the project.

4.1.2 Outcomes. Two of the three participants did not use the configurations but spoke from a usability perspective if they had to use the system as they would want it presented. The third participant had worked with the configurations more and gave more detailed requirements to reduce possible human error.

The requirements are as follows:

- Each configuration be separated from the others with detailed descriptions detailing what the function does.
- To assist with how the data was shown to users, participants suggested using drop-down boxes with key and matching value pairs, which would also reduce the amount of code shown to users.
- Each addition or change to a configuration array in the perl file should have immediate feedback to the user to show what has been updated and prevent updates if the change or addition will result in an error, such as adding a new administrator ID when the same admin ID already exists.
- An addition is to be able to change the way collection logos are uploaded; currently, one has to add the location of the logo through the code, but it should rather be done through the interface. This feature was to be implemented if there was time, as it is not a priority.

The feedback was used to refine the first iteration of the project, which was the demo, and address changes to the format of the way the interface was showing the data. Participants did not feel comfortable with the whole configuration code being on the interface that one can edit directly, separated by different options. The feedback was that when designing the interface, there should be an emphasis on users who don't have the appropriate background in computer science or configurations of digital repositories.

4.2 System architecture

The administrative system interface for SimpleDL consists of a two-tier system consisting of a client and server. The frontend client will include the configuration interface that users interact with, and the backend is the server that edits the configuration perl file, CSS and XSL files. Refer to Figure 1 for a diagram of the high-level system architecture where the highlighted part represents the configuration component.

4.3 Software artifacts

Unified modeling language (UML) was used to define and document the system through visuals [12]. This ensured understanding of user stories between us and the client. Refer to figure 3 for a use case diagram which shows user stories for system configuration and UI configurations, users can perform multiple functions on

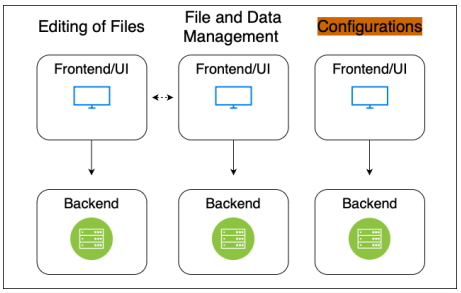


Figure 1: High-level system architecture

the system. Refer to figure 2 for an activity diagram for system configurations. The figure shows what functions users can perform as well as the process of the server receiving the data and updating the configuration file. Refer to figure A.1,5,6 and 8 in the appendix for more software artifacts that describe the process of the system.

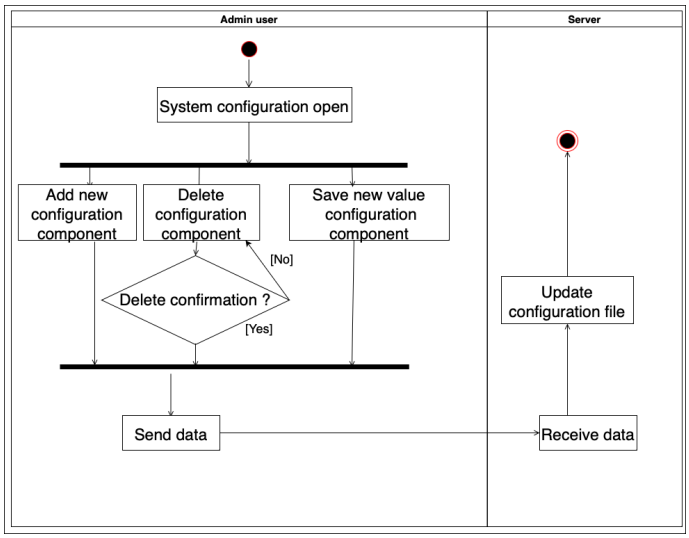


Figure 2: Activity diagram for system configurations

5 SYSTEM IMPLEMENTATION

5.1 Implementation strategy

The software development methodology used was the agile methodology. Given that we are a small team, agile allows us to manage our work more efficiently and effectively as opposed to traditional methods [5]. This gave us the freedom to prioritise the most important tasks first and change requirements midway through development. Software iterations could be released as quickly as possible to have users test the iterations, so the system could be continuously refined as much as possible. The methodology was well suited because of the short time given to complete the project.

The four core values of the agile methodology were adhered to strictly. The Emandulo users individual interactions drove the development of the system [5]. Weekly meetings with our supervisor

Figure 3: Use case diagram for add components of system configuration and UI components.

and co-supervisor were held to assure that our progress was on track and that the system was meeting the client's needs.

A focus was placed more on working software than comprehensive documentation [5]. The only documentation that was created was during the requirements gathering and usability testing stages for user stories and refinement of requirements gathering. To make sure that each component was working while developing, console logs were used for each function. To test whether the system was sending the appropriate data to update, delete, and add to the configuration file between client and server, acceptance tests and walk-through tests were used later in development for more exploratory testing purposes.

The short intervals between the user evaluations for each iteration allowed us to respond to change more easily [5]. User evaluations resulted in an evolving list of requirements. The changes were made as a priority to refine the functions before the next user evaluation.

The Kanban tool was used to aid in adhering to the agile methodology; it helped with improving the quality of the system at each task, encouraged coordination and communication and maximised efficiency amongst the team [30].

Implementation of the project took place in three stages.

- Phase 1 : Demo/first iteration: The first demo was done as per requirements discussed with our supervisor. Basic functionality was added, and options were added to the frontend that differentiate each configuration that exists in the configuration file. Only the formats for uploads were implemented, with the option to add a new file format. The rest of the configurations were added, but as an edit box to directly edit the code in the perl file. For the UI configurations, only changing the banner title was implemented. The changes did not reflect immediately on the frontend. Users had to reload the page each time a new format was added.
- Phase 2 : second iteration: The second iteration produced an improved interface with more detailed controls and more functions. Functions included add, delete and save for acceptable file formats, administrator accounts, separators, structure uploads based on a field, manageable directories, main directory layout, website template locations, locations of external programs and key files and vocabulary. The functionality for the UI configurations of the look and feel of the SimpleDL site were created in this phase to update the CSS file and XSL transform file with new titles, descriptions, SimpleDL logo and colours of components on the site such as collection box colour.
- Phase 3 : third iteration: Additional system configuration functionality were added being save, delete and update for list of fields for search and browse. The UI was refined. Configurations had immediate feedback of the system status following a change in the configuration and error checking messages to limit the amount of human error. Errors including mistakenly clicking on 'add' to configuration when an input box is empty, incorrect formats and duplicated entries for unique keys such as for admin id.

5.2 Development platform and technology

The system was developed as a web application and is compatible with all web browsers, such as Chrome, Safari, and Firefox. Visual Studio Code was used for the coding environment. A virtual machine was created to connect and setup SimpleDL on the local laptop due to SimpleDL working best on Linux operating system. A SSH client extension on Visual Studio Code made the connection, which allowed the virtual machine folders to open and be made editable from the local machine[15]. Node.JS and React.JS was chosen for the development of the project. Node.JS is a framework of JavaScript which handles the backend of the application [26]. React.JS is a frontend library based on JavaScript. Using these JavaScript frameworks is beneficial as it allows for the system to be faster and easy to use. JavaScript was chosen because of its ability for users to directly use the web applications dynamically without needing to reload the page every time a configuration is changed [7].

An HTTP-header-based system called Cross-Origin Resource Sharing (CORS) enables a server to specify an origin other than its own from which a browser should be able to fetch resources [8]. CORS was needed as the server needed to communicate with a web browser that was running on the a different machine such as

running the web application on the virtual machine but displaying the user interface on the local computer. The local browser prevents this due to security reasons but allows it when CORS is used.

The multer library from nodeJS was used for uploading images [2]. The express framework was used because of its broad features suitable for building a web application, such as using template engines to produce a HTML template to make dynamic items on a web application and aids in the agile methodology of having working code by making debugging easier to conduct [19]. To pull information out of the perl file and transform xsl, Node.js file paths were used. A JavaScript object notation format (JSON) was used to interchange data between the client and server side using key-value pairs [6]. Google drive was used for data backups and version control to prevent data loss and to retrieve an earlier version of the project if needed.

5.3 Implemented features

The components implemented are based of user requirements and user stories. The components are detailed as to what the feature entailed and how it was implemented.

5.3.1 General. Each configuration function follows a similar approach to updating respective configuration files. The web application interface been the client takes input from user through input boxes and when buttons of add, update or delete is clicked, the data is sent to the server using a 'post' request and updates the relevant configuration file with the new data.

5.3.2 Acceptable file formats to upload. Acceptable file formats is a configuration that dictates which file formats users are able to upload digital artifacts in. Examples of formats include pdf, mp3 and zip.Refer to figure 4 which shows the configuration in the perl file. The web application configuration interface presents these file formats in a list view with options to either delete or add a new format.

The implementation of the function consists of a state variable that gets updated with the formats that is sent from the server using a 'GET' request in a JSON response. This is done by the server reading the configuration file using fs, which reads the file in UTF-8 coding and tests it using callback functions to provide the user with the error or complete message. A regular expression is used to find the format fields array. The data is then extracted and split using a new regex to create the format array, which is sent to the client. The contents of the array are then shown to the user on the web interface.

The add feature uses an input box for the user to provide a new format and a button to add it to the array. The add button uses the same approach as section 5.3.1. The server uses a regex expression to find the format array in the perl file. The extracted data is formatted into an array, and the new format is appended to the array. fs.writeFile is used to replace previous data in the config file.

The delete function follows a similar approach, where the user states what to delete in the input box and clicks on the delete button. The array in server is split and filtered to remove the format, and the previous array is replaced with the updated version and written back to the file. Each edit to the format array updates dynamically on the frontend. Refer to the figure 5 which shows the final interface for formats.

5.3.3 Administrator accounts. The interface separates the administrator account functionality into two options. The first is to view listed admin ids with options to delete and add. The second is the administrator password, which only has the option to update.Refer to figure 4 which shows the administrator account configuration in the perl file

Variables are used to set the corresponding password and administrator IDs. The administrator ids are mapped as a list to the interface. Each ID is rendered with a delete button; when the delete button is clicked, the ID is sent to the backend. The server uses regex expressions to find the id array, and if found, checks if the element first exists, then removes the id from the array and replaces the previous one with a new one. A similar approach is taken to uploading a new id; it checks if the id exists first and adds the new id and replaces the array with the updated version. The 'Modal' libray is an accessible modal dialog component for React.JS. This is used to produce confirmation modals for deletion, save and addition to provide feedback to the user if their action is successful or not. The interface of the modals provides options to close the confirmation and warning messages to improve the usability of the system.

To update the password, the user inputs a new password into the input field, and the delete button is used in a similar way as in section 5.3.1. Refer to the figure 5 which shows the final interface for admin accounts.

```
# acceptable file formats for upload
@upload_accept = ('JPG', 'jpg', 'jpeg', 'png', 'tif', 'tiff', 'pdf', 'mp3', 'zip');

# administrator accounts
@administrators = (1);
$verifySalt = 135276;
```

Figure 4: Acceptable file formats and administrator accounts in configuration file

5.3.4 Separators. The separator configurations consists of variables that are used to define different separators that split and organize data in the collections into fields and sub-fields. The separator interface on the web application shows the four separators with their current value. Below the separators, there is a drop-down box that updates a selected option variable depending on which separator is chosen. In the configuration file, the separators are assigned to a variable similar to how the password for admin accounts is stored. The regular expression to find the separator assignments in the perl file is slightly adjusted to account for each separator. Each separator has its own regex expression. The data is then sent as a JSON array, where each element is the value of its respective separator. On the client side, each separator variable is assigned a value.

A combination of the selected option variable and the new character the user inputs into the input field is sent to the server using the

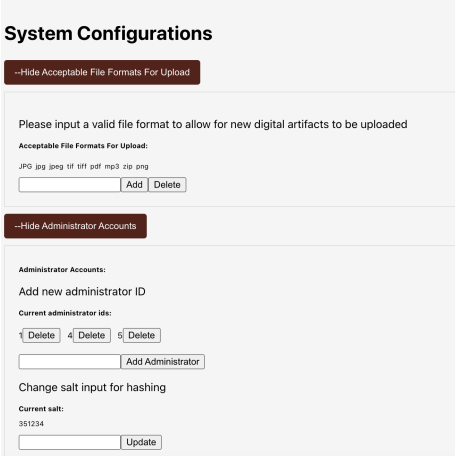


Figure 5: Acceptable file formats and administrator accounts final interfaces

approach described in section 5.3.1. The server uses the selected option value to append to the regex expression where there is a placeholder for the value. The expression extracts the relevant line containing the separator and updates that specific separator, and the new value is dynamically updated at the frontend by recalling the use effect function that fetches the data from the backend for this configuration. Refer to figure 6 which shows the final interface for separators.

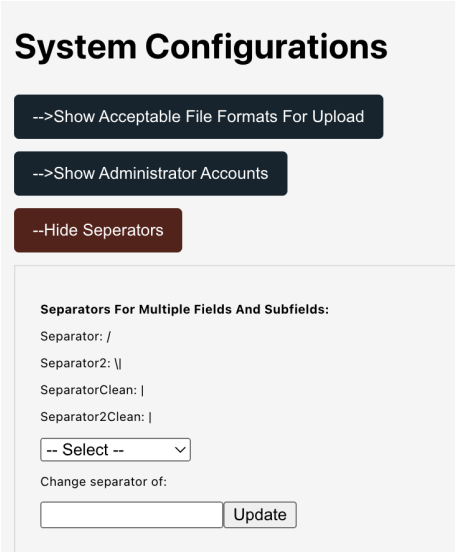


Figure 6: Separators final interface

5.3.5 List of fields for search and browse and sort. The configuration of list of fields to search and browse and sort, stores fields for search which is metadata which allows the system know where to look for the artifact when a search is made. The browse list contains terms that exist in the list of fields to search. These terms in browse help organize and present the digital artifacts to users for browsing and sorting. The configuration file stores the data in the form of a Perl hash, functioning as an associative array. Refer to figure 7 which shows the fields for search and browse in the perl file.

The 'search' field's data is presented to the user in a tabular format. This format uses two columns to display key-value pairs, enhancing the readability and user friendliness of the data. The data is extracted on the server side using a similar regex expression as in previous configuration implementations. Each key and value pair is clickable, which uses the 'modal' library to open a pop-up window. The pop-up window gives the user options to edit the value of the key that is open and send the edited value to the server to update the current array and write to a file. The pop-up was created with buttons to close the pop-up with 'cancel' to ensure that users can go back to their previous state if opening the pop-up was a mistake.

To add a new key and value pair to the array, there are input boxes for each, and the add button triggers the key and value pair to be sent to the server. The server uses regular expressions to search for that specific key and updates its value directly.

The delete function follows a similar approach, but only the key is sent to the server; the server then uses regular expressions to match the key and replaces the key and value pair with an empty string. The new addition is added with a comma at the end of the value at the top of the array list to prevent errors when importing the new edited configurations.

The field browse data is dependent on what metadata exist in the 'field search' data. The data in the configuration file exists in an array with three items per element. The server only fetches and sends the first item of each element. The metadata items are displayed in a list, one item below each other, and get dynamically updated when a user chooses to add a browse element to the array using the drop-down box. The drop-down boxes data consists of the metadata in the search fields. The add button places the new browse field into the browse field array in the configuration file with the correct format. Each browse element is rendered with a delete button to delete the element that contains the key on the server. Refer to figure 8 for the final interface for list of fields for search and browse and sort configuration.

5.3.6 Manageable directories, Main directory layout,website template locations, location of external programs and key files, vocabulary. The structure of these configurations is showcased to the user in the same way by having the function name of a configuration displayed first and under it a drop-down box consisting of the keys of the respective data. The data on the server is extracted using regular expressions and sent to the client as a string. The string is formatted and split by key and value pairs into two arrays. A function is used to show the corresponding value of the key element in an edit box below the drop-down, depending on which key is selected. The value can then be edited and sent to the server using the approach in 5.3.1 to update the value of the key in the configuration file.

```
'field_search' => {
title => 'fullrecord/item/title',
    creator => 'fullrecord/item/creator',
    contributor => 'fullrecord/item/contributor',
    coverage => 'fullrecord/item/coverage',
    date => 'fullrecord/item/date',
    description => 'fullrecord/item/description',
    format => 'fullrecord/item/format',
    identifier => 'fullrecord/item/identifier',
    language => 'fullrecord/item/language',
    publisher => 'fullrecord/item/publisher',
    rights => 'fullrecord/item/rights',
    relation => 'fullrecord/item/relation',
    source => 'fullrecord/item/source',
    subject => 'fullrecord/item/subject',
    type => 'fullrecord/item/type',
    lod => "fullrecord/item/levelOfDescription",
    comments => "fullrecord/comments/comment/*",
    subcollection => "fullrecord/filename/subpath",
    all => "weight=5 fullrecord/item/title;*//text()"
},
#FieldBrowseStart
'field_browse' => [
    [ title, 'Title', 'title' ],
    [ creator, 'Author', 'creator' ],
    [ publisher, 'Publisher', 'publisher' ]
],
```

Figure 7: List of fields for search and browse and sort in configuration file

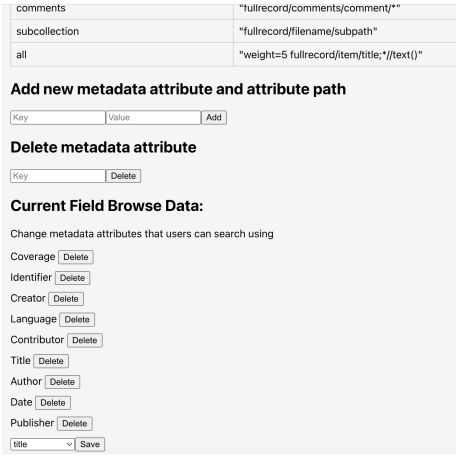


Figure 8: List of fields for search and browse and sort final interfaces

Users can define new key-value pairs using input boxes, subsequently leading to updates within the configuration arrays present in the config file. Additionally, a delete function is available, enabling the removal of specific key-value entries from the array. Refer to figure 9 for final interface for manageable directories and main directory layout. Interfaces for template locations, locations for external programs and vocabulary are presented in the same way.

5.4 UI configurations

The UI is generated from XML source files that are transformed by an XSLT style sheet into associated HTML files. This includes CSS styling files to style the SimpleDL site. The titles and descriptions of components of the SimpleDL interface in the XSL file is written directly within the anchor element. This was changed to storing the

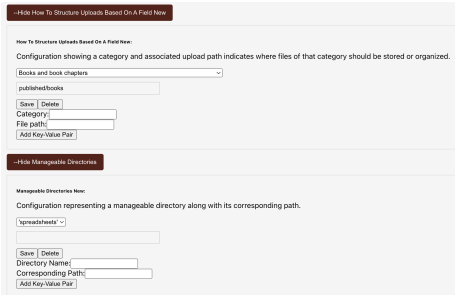


Figure 9: Manageable directories and Main directory layout final interfaces

value of titles and descriptions into XSLT variables and then only inserted into the anchor element. Refer to figure 10 for an example of the dynamic variable. This gave the application the ability to dynamically change the variables from the application frontend.

```
<xsl:variable name="dynamicTitleAdminHeader">Admin</xsl:variable>
  <title><xsl:value-of select="$dynamicTitleAdminHeader"/></title>
```

Figure 10: How dynamic variables are created

5.4.1 Titles. The current value of a title that exists on the SimpleDL site is shown to the user with the heading of what the title is for. An input box is provided for the user to type a new value for a title, and an update button handles the input in the same way described in section 5.3.1. The server uses regular expressions to match the pattern of where the dynamic variable pertaining to a title is located. The server replaces that line in the XSL file. The updated data is then written to the file, replacing the previous one. The latest value of the title is updated on the frontend without needing to reload the page. Refer to figure 11 which shows the final titles configuration interface.

5.4.2 Descriptions. The description of what the SimpleDL site is about is approached in a similar way to the titles. It uses a dynamic variable that needs to be updated by the user on the frontend. The update button sends the data in the same way as in section 5.3.1 which then changes the value of the description in the configuration file.

5.4.3 images. The configuration application interface features a drop-down box consisting of all the current image names in the image file path. The user has the option to select which image they would like to update on the SimpleDL logo on the site. Once the user clicks on update image, the image file path in the transform file is replaced with the new file path. There is an option to upload an image directly into the image folder; this updates the image drop-down box selector with the new image being added.Refer to figure 12 which shows the final interface for the image upload configuration.

5.4.4 CSS. CSS configurations allows users to change the look and feel of the site by changing the colours of various components. The server extracts the current rules of each CSS function and sends

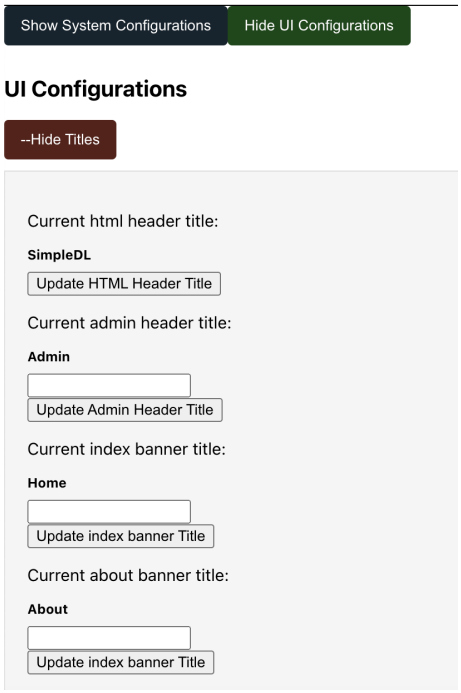


Figure 11: Titles final interface

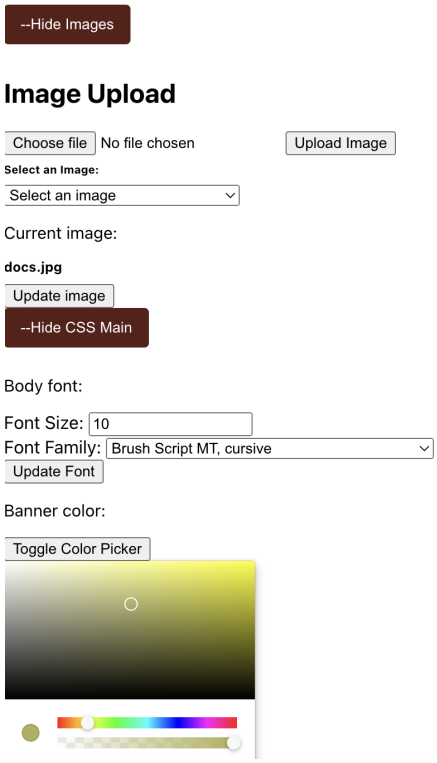


Figure 12: CSS final interface

them to the client. The client updates the relevant state variables with the current font, colour, and font size and buttons send the new data in the same way described in section 5.3.1.

The font size component features an input element that allows the user to pick a number through arrows to increase or decrease the value. The user can also type in the font size. To change the family of font, the user is given a drop-down box of options for family fonts. Once the required size and font family are set, the user can click on 'update' font.

To present different options for colours, the library 'ChromePicker' was used. It shows a colour wheel for users to interact with and pick a colour to update a specific component. The 'ChromePicker' is initialised with the current colour of a component sent from the server side, which gets it from the CSS. The user can either pick a colour from the picker or input a RGB Colour. Update colour sends the new data in the same way described in section 5.3.1, the new value of the RGB updates the previous colour variable in the CSS for that specific component. Refer to figure 12 which shows the final interface for body font and colour component configurations.

6 TESTING

Following the first demo/iteration, usability testing was conducted at each iteration and software tests were implemented. The meetings took place on Google Meet and in person.

6.1 Software tests

Software testing was conducted to validate and verify that all software worked as intended, prevent most bugs and increase performance [27].

6.1.1 Performance testing: Performance testing for a web application was chosen because of its ability to assess the overall compliance of the system [11]. Specifically, web application performance was used to determine how well the system performs in terms of speed and web server response time.

Chrome DevTools were used to assess the performance of the page. The performance tool provides a breakdown of the CPU activity using metrics for loading, scripting, rendering, and painting measured in milliseconds.

Tests were done for the loading of the landing page and some functions to test how long add, delete, and update take. The outcome concluded that the functions did load fast enough and that no issues were found that were slowing down the application.Refer to figure 13 which shows the update test. The update test showed a load time of 197 mm which is in the ideal range for load speed of 0-2 seconds [4].

Lighthouse was another tool used via Chrome to assess the performance of the system. The resulting score was 99, which indicated that the site is good for user experience [1]. However, an issue was

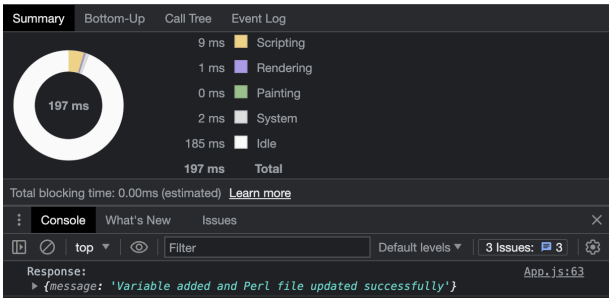


Figure 13: Update test performed

found with the cache of the site that was slowing down performance. Lighthouse had suggested that a long cache lifetime could speed up repeat visits to the site. Refer to figure A.7 in the appendix for the test result.

6.1.2 Browser compatibility: Browser compatibility was another software test conducted to ensure that the system runs on all browsers and displays correctly. Test cases were made to test if each function on the interface is consistent and the data is rendered correctly [29]. The website Lambdatest was used as a tool for performing the browser compatibility test [3]. The tool used a tunnel to the localhost to open the configuration interface in multiple browsers to check if they passed the test cases. The browsers that were tested on Lambdatest were Firefox, Brave, and Opera. Chrome and Safari were tested locally and not on Lambdatest. All browsers passed all the tests except Safari, which rendered a question mark when a image was selected in the drop down. Refer to figure 14 which shows the rendered question mark. This was amended by removing the display of the current image view in the code.

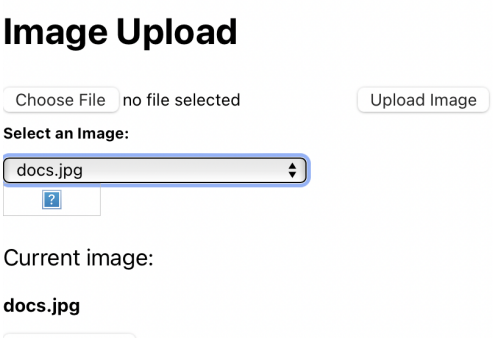


Figure 14: Browser test for Safari showing error

6.2 User tests - iteration one

Expert users who were shown the initial demo or iteration were contacted to perform heuristic evaluations of the system. The testing was done via Google Meet. The participants were briefed on how the testing was going to be conducted. Each team member had a chance to show their system and give time for participants to go through each of Nielson's 10 usability heuristics and rate it according to how much the system violated a heuristic. A score of zero to one indicated that there was little to no problem with the usability of the interface. A score of three to four indicated that the problem with the usability posed a severe issue that needed to be addressed [16]. Participants provided feedback on each rating.

6.2.1 Results. Results are listed below with each heuristic and its summary on comments and ratings.

- Visibility of system status. All participants did not feel as if they were informed of the outcome of their actions. Clicking on update, delete, or add did not provide appropriate feedback. However, the updated variables did update immediately on the frontend, which gave users feedback on their actions. The resulting average score was one.
- Match between the system and the real world. The users did not understand what the terms were for system configurations. This resulted in not knowing what each configuration meant, which made it difficult to use the system. The resulting score was a two.
- User control and freedom. Users were satisfied that they could undo an action by simply using the delete and add functions. Users scored this heuristic a one. This area can be improved to cater to 'emergency exists', such as undo buttons, but is not a priority.
- Consistency and standards. Testers did not find confusion in wondering if different actions meant the same thing. There was consistency throughout the system. Users rated this heuristic a zero.
- Error prevention. Users found no error prevention messages. Suggestions were to add error messages as a pop-up to ensure the user sees the message and minimise errors in the configuration file. Users rated this heuristic a two.
- Recognition rather than recall. Users found no issue with this heuristic, as all labels were clearly defined and visible. They acknowledged that the names of each configuration were easily retrievable when needed. Participants rated this heuristic a zero.
- Flexibility and efficiency of use. Participants felt that if they had thorough knowledge about each configuration, they could use the system faster. However, there are no shortcuts to the system without expert knowledge. Participants rated this heuristic as one. They had commented that they chose this rating as they can efficiently use the system at a good pace to perform tasks but could use it quicker if they had more detail on each configuration and what format it takes in.
- Aesthetic and minimalist design. Participants had mentioned that they agreed that there was no irrelevant information present on the interface. The interface is focused on the essentials. The overall rating was a zero.
- Help users recognise, diagnose, and recover from errors. Participants did not find any useful error messages indicating an error had occurred. They suggested having useful solutions when errors are generated that will aid users in avoiding the problem. Users gave this heuristic a two.

- Help and documentation. Users had commented on the fact that they needed additional help to use the system as they do not routinely change configurations. They had suggested providing descriptions of each configuration and what the format of the data should be. Users gave this heuristic a three.

6.2.2 Discussion. In the first round of usability testing, participants found many issues with the current system. The issues were mainly usability issues when it came to a lack of knowledge about SimpleDL configurations. However, they had mentioned that the design of the web application and how the configuration data is provided in drop-down boxes and edit boxes work well for them. Adjustments were made to the system following this testing to include error messages, confirmation messages to indicate outcome of actions, and more detailed functions. Refer to figure A.2, A.3 and A.4 in the appendix which shows the error messages and confirmation messages that were added. What was noticed was that the evaluation took place with two participants at the same time. When one participant gave a rating the other agreed with, there might have been bias due to not wanting to disagree with a fellow colleague. The second iteration had users perform usability testing separately to account for the bias.

6.3 User tests - iteration two

The second round of usability testing took place via Google Meet and in person with expert digital library users at UCT with three users. The participants evaluated the system using Nielson heuristics and were asked to perform tasks according to user stories in the form of a cognitive walk through.

6.3.1 Results. For visibility of system status users gave a zero as the system kept them informed of what the outcome was of their actions with respective changes updating on the configuration application interface itself and with confirmation pop ups. For match between the system and the real world, the rating had decreased from the first round to one as better phrases were used to describe key-value pairs and the functions. However, users still found difficulty understanding some jargon and wanted us to use more familiar concepts. Error Prevention. Evaluators were satisfied with the newly added pop ups for error messages however, wanted more error checking for other configurations. The overall rating dropped to a one. For flexibility and efficiency of use, the rating had stayed at one as users suggested that the current interface and functionality are efficient enough. For help users recognise, diagnose, and recover from errors, evaluators rated this a one as they still found difficulty in recovering form errors, but this was improved from the last iteration. Users had suggested using more detailed error messages. Help and documentation was rated a one as evaluators still believed that this section could be improved by using more detailed descriptions of each function although they had commented that the functions were more detailed from the first iteration .User control and freedom ,consistency and standards users, recognition rather than recall and aesthetic and minimalist design had an overall of zero as feedback was the same was iteration one.

Users were also asked to perform tasks based on user stories and the requirements of the system. These checks were done as a cognitive walkthrough. Tests were set to assess each component. Refer to table 1 for tests and results.

Table 1: Acceptance tests and results

Test	
Add and delete an acceptable file format	Pass
Add and delete an admin ID	Pass
Update administrator salt password	Pass
Change separator values	Pass
Edit value of search field keys	Pass
Add and delete search field keys	Pass
Edit, add and delete manageable directories	Pass
Edit, add and delete main directory layout	Pass
Edit, add and delete website template locations	Pass
Edit, add and delete locations of programs and files	Pass
Edit, add and delete vocabulary	Pass
Change description of 'about'	Pass
Change image of SimpleDL logo	Pass
Change colours of SimpleDL site components	Pass

6.3.2 Discussion. The second iteration demonstrated improvements from the first. Participants found little to no issues with the usability of the system. However, there was still confusion about what each configuration meant. A user had suggested implementing a 'help' function either using a hover function or pop out that describes to user what the function entails for future work. All components of the system worked as expected. Users were able to perform tasks with little help to no help.

7 CONCLUSIONS

The web application was created to provide an interface for the configuration of the SimpleDL. The system provides a user-friendly interface to mimic user interaction with the configuration Perl and XSL files. This is achieved by extracting each configuration in the files using regex expressions, updating the extracted data, and writing to the file. Functions to mimic user interactions include add, delete, and update. Testing was done with expert SimpleDL users; users were asked to perform heuristics and cognitive walkthroughs. The feedback was positive and satisfied the requirements for the system. The overall objective of the system was achieved that configurations can be given a user-friendly interface that users can use without expert knowledge of coding and with a reduced chance of error.

For future work the configuration interface can be explored to test different approaches to helping users navigate the system. Such as introducing an AI chatbot that can guide one on how to use the configurations. The configuration perl file can be changed to an xsl similar to the transformation file. The perl file made it difficult to extract configurations as compared to the xsl file. This would make the configuration updates, deletions, and additions more seamless and less prone to human error.

REFERENCES

[1] 2023. Google Lighthouse: What It Is How to Use It — semrush.com. https://www.semrush.com/blog/google-lighthouse/. [Accessed 14-09-2023].

[2] 2023. multer — https. https://www.npmjs.com/package/multer. [Accessed 14-09-2023].

[3] 2023. Next-Generation Mobile Apps and Cross Browser Testing Cloud | LambdaTest — lambdatest.com. https://www.lambdatest.com. [Accessed 14-09-2023].

[4] 2023. What Is Page Load Time on a Website and Why It Matters - Sematext — sematext.com. https://sematext.com/glossary/page-load-time/. [Accessed 14-09-2023].

[5] Samar Al-Saqqa, Samer Sawalha, and Hiba AbdelNabi. 2020. Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies* 14, 11 (2020).

[6] Tim Bray. 2014. *The javascript object notation (json) data interchange format.* Technical Report.

[7] Sanja Delcev and Drazen Draskovic. 2018. Modern javascript frameworks: A survey study. In *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)*. IEEE, 106–109.

[8] Nicholas Gibbins. [n. d.]. Cross Origin Resource Sharing. ([n. d.]).

[9] Douglas Christopher Gorton. 2007. *Practical digital library generation into DSpace with the 5S framework.* Ph. D. Dissertation. Virginia Tech.

[10] Stuart Hammar and Miles Robinson. [n. d.]. A Web-based Digital Repository System. ([n. d.]).

[11] Mohd Ehmer Khan. 2010. Different forms of software testing techniques for finding errors. *International Journal of Computer Science Issues (IJCSI)* 7, 3 (2010), 24.

[12] Hatice Koç, Ali Mert Erdoğan, Yousef Barjakly, and Serhat Peker. 2021. UML diagrams in software engineering research: a systematic literature review. In *Proceedings*, Vol. 74. MDPI, 13.

[13] Nils Korber and Hussein Suleman. 2008. Usability of digital repository software: A study of DSpace installation and configuration. In *International Conference on Asian Digital Libraries*. Springer, 31–40.

[14] Murugan Krishnan. 2016. *Digital Repositories: An Overview.* 209–219.

[15] Microsoft. 2021. Developing on remote machines using SSH and Visual Studio Code. https://code.visualstudio.com/docs/remote/ssh

[16] Jakob Nielsen. 2005. Ten usability heuristics. (2005).

[17] Jung-Ran Park. 2009. Metadata quality in digital repositories: A survey of the current state of the art. *Cataloging & classification quarterly* 47, 3-4 (2009), 213–228.

[18] MN Ravikumar and T Ramanan. 2014. Comparison of greenstone digital library and DSpace: Experiences from digital library initiatives at eastern university, Sri Lanka. *Journal of University Librarians Association of Sri Lanka* 18, 2 (2014), 76–90.

[19] Anubhav Sharma. 2023. Understanding react and express: (A comprehensive guide): Simplilearn. https://www.simplilearn.com/tutorials/react-tutorial/guide-to-understanding-react-and-express#:~:text=Express%20js%20is%20a%20node,fast%2C%20unopinionated%2C%20and%20lightweight.

[20] MacKenzie Smith, Mary Barton, Mick Bass, Margret Branschofsky, Greg McClellan, Dave Stuve, Robert Tansley, and Julie Harford Walker. 2003. DSpace: An open source dynamic digital repository. (2003).

[21] Thornton Staples, Ross Wayland, and Sandra Payette. 2003. The Fedora Project. *D-Lib Magazine* 9, 4 (2003), 1082–9873.

[22] Hussein Suleman. 2012. The design and architecture of digital libraries.

[23] Hussein Suleman. 2019. Reflections on design principles for a digital repository in a low resource environment. (2019).

[24] 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.

[25] 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.

[26] Stefan Tilkov and Steve Vinoski. 2010. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing* 14, 6 (2010), 80–83.

[27] James A Whittaker. 2000. What is software testing? And why is it so hard? *IEEE software* 17, 1 (2000), 70–79.

[28] 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.

[29] Lei Xu, Baowen Xu, Changhai Nie, Huowang Chen, and Hongji Yang. 2003. A browser compatibility testing method based on combinatorial testing. In *International Conference on Web Engineering*. Springer, 310–313.

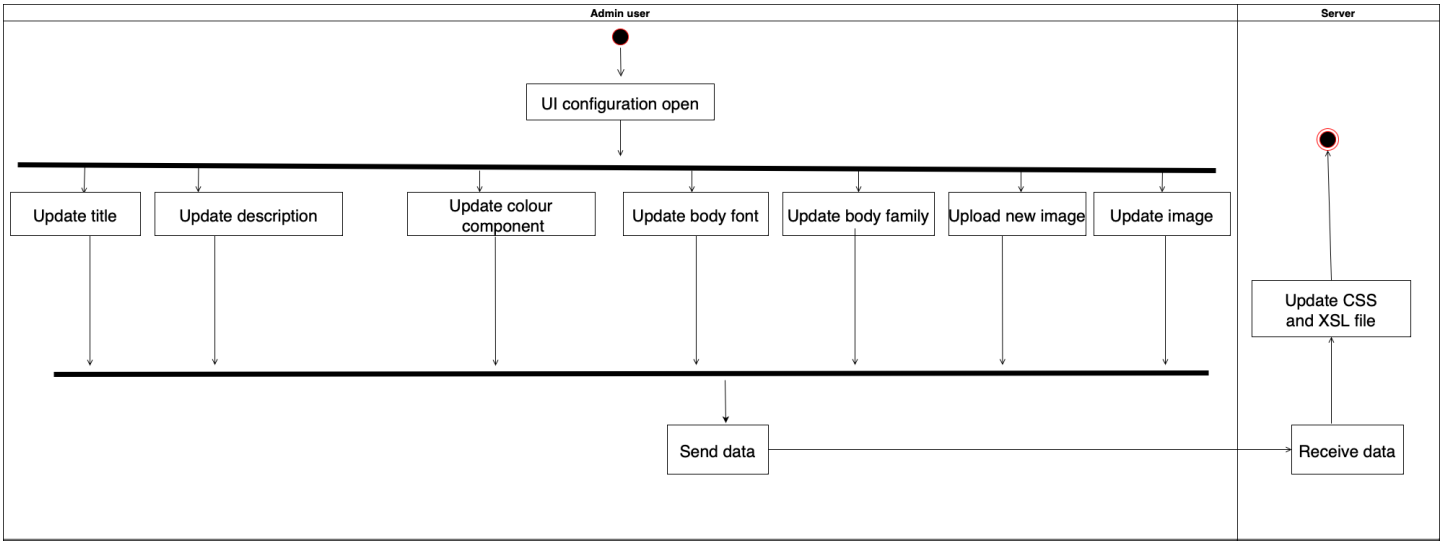
[30] Wael Zayat and Ozlem Senvar. 2020. Framework study for agile software development via scrum and Kanban. *International journal of innovation and technology management* 17, 04 (2020), 2030002.

[31] Alesia Zuccala, Charles Oppenheim, and Rajveen Dhiensa. 2008. Managing and evaluating digital repositories. *Information Research: An International Electronic Journal* 13, 1 (2008).

APPENDIX

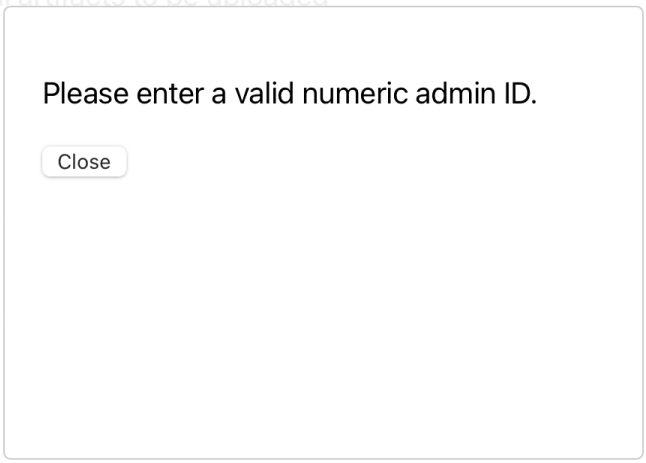
A.1 ACTIVITY DIAGRAM

Activity diagram showing UI configuration



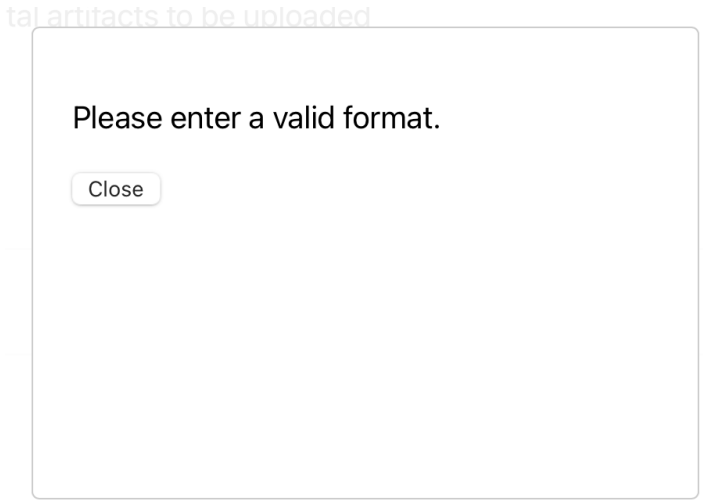
A.2 ERROR MESSAGE FOR ADMIN ID INPUT

Error occurs if user tries to add a letter instead of a number for admin ID



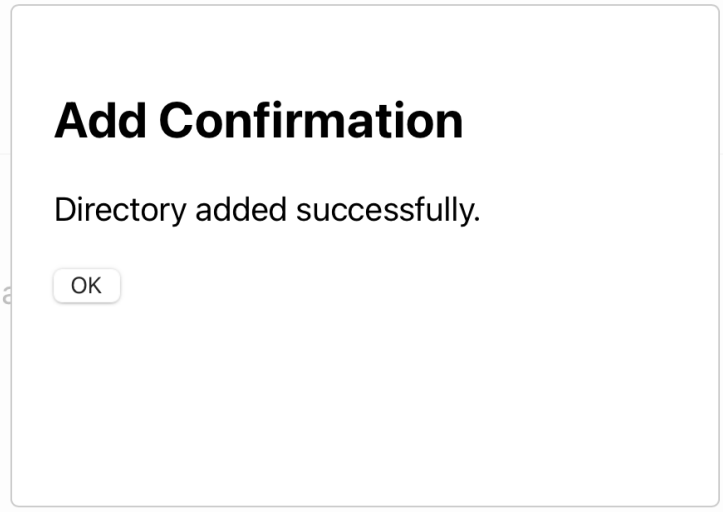
A.3 ERROR MESSAGE FOR INPUT FIELD

Error occurs if user tries to add a blank input



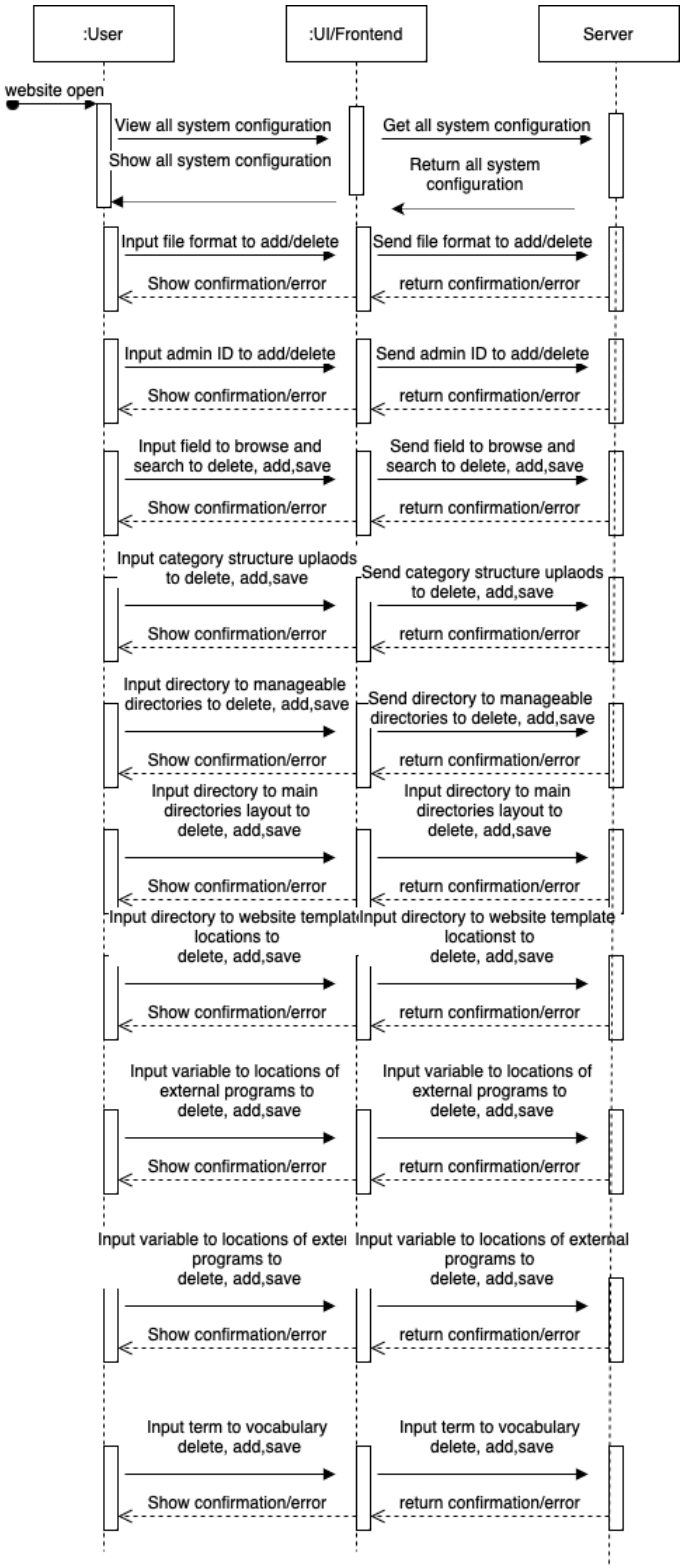
A.4 CONFIRMATION MESSAGE

Confirmation messages for adding a directory.



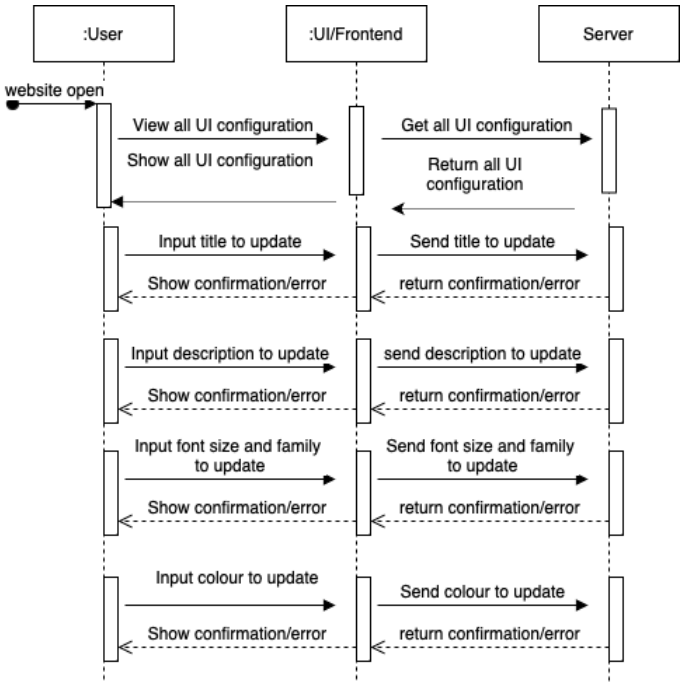
A.5 SEQUENCE DIAGRAM FOR SYSTEM CONFIGURATIONS

Diagram showing process of system configuration between the user, UI/frontend and server.



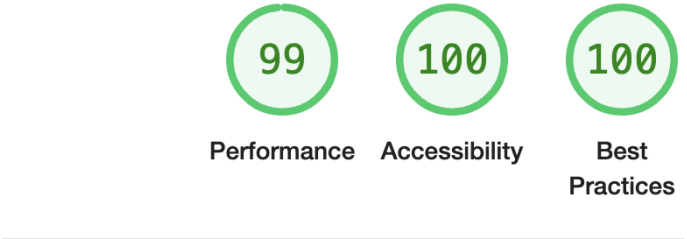
A.6 SEQUENCE DIAGRAM FOR UI CONFIGURATIONS

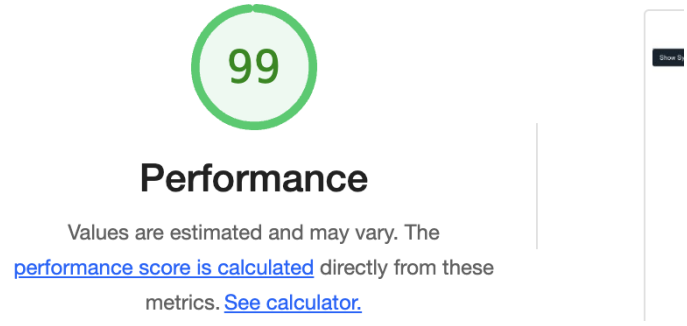
Diagram showing process of UI configuration between the user, UI/frontend and server.



A.7 LIGHTHOUSE TEST

Diagram showing performance test result.





A.8 DEPLOYMENT DIAGRAM

Diagram showing how software components and hardware components of system configurations component.