Tab 1

SEARCH ABERDEEN REGISTERS 2025

1495 UNIVERSITY OF ABERDEEN

Team Lima

CS3528 Software Engineering and Professional Practice

Search Aberdeen Registers 2025

Team Members

- Andreas Maita (a.maita.22@abdn.ac.uk)
- Caitlin Thaeler (c.thaeler.22@abdn.ac.uk)
- Dermot Stelfox (d.stelfox.23@abdn.ac.uk)
- Fariha Ibnat (f.ibnat.22@abdn.ac.uk)
- Haziel Osunde (h.osunde.22@abdn.ac.uk)
- Holly Sinclair (h.sinclair.22@abdn.ac.uk)
- Piotr Śmiałek (p.smialek.22@abdn.ac.uk)
- Rebekah Leslie (r.leslie.22@abdn.ac.uk)

Acknowledgements

We would like to express our sincerest gratitude to our guide, Dr. Leonardo Amado, for his invaluable guidance, support, and encouragement throughout the development of this project. His insights and feedback were instrumental in shaping our work.

We are also deeply thankful to our client, Dr. Jackson Armstrong, whose passion and vision for the project were both inspiring and motivating, and Phil Astley at the Aberdeen Council Archives where we had the privilege of viewing the Aberdeen Burgh Records in person.

Finally, we would like to thank the users who took the time to test our application and provide thoughtful feedback. Their contributions were essential in helping us refine the user experience and improve the overall quality of the system.

Project Video

https://youtu.be/wZQnMB77LEc

Project Link

https://sar2.andreasmaita.com/

1 Table of Contents

Introduction

About Aberdeen Registers Online

The Aberdeen Registers Online (ARO) corpus is a special scholarly resource comprising a completely digitised transcription of the first eight volumes of the Aberdeen Burgh Registers, the oldest surviving civic government record in Scotland. Spanning 113 years from 1398 (volume one) to 1511 (volume 8), these records document a wide range of civic and legal activities, including court proceedings, property transactions, elections, ordinances, and official correspondence. These volumes provide rare and invaluable insights into the laws, customs, and daily lives of people living in 14th-century Aberdeen, preserving the complex social and legal reality of a late medieval urban community. Their uniqueness lies not only in their age and scope, but also in the detailed administrative chronology they provide. These books contain the earliest and most comprehensive archive of civic government records in Scotland and, as such, give historians the unique opportunity to trace and analyse patterns in the legal proceedings of an urban city in the late Middle Ages over more than a century. Nothing highlights the historical significance of these records more than their inscription on the United Nations Educational, Scientific and Cultural Organisation (UNESCO) UK Memory of the World Register, a list that recognises documentary heritage of exceptional value: "This invaluable archive is a remarkable and rich insight into a small island's past and mark on the world. This is Britain's collective memory" (UNESCO, 2025). The digitisation and transcription of these records was carried out by the Law in the Aberdeen Council Registers (LACR) team, which has made this remarkable archive accessible to scholars, educators, and the wider public. By transforming fragile, century-old physical documents into searchable digital texts, the LACR team has preserved an irreplaceable part of Scotland's documented heritage for many years to come, as well as promoting new forms of scholarly analysis and public engagement.

In 2017, a prototype web application called Search Aberdeen Registers (SAR) was developed by Team Charlie in collaboration with the LACR team. Its primary aim was to support and enhance the lives of researchers working with the ARO corpus, providing a tailor-made tool for searching, filtering and viewing the digital transcriptions of the records. Before the development of the SAR search tool, the LACR team did not have access to adequate resources for their research, as no specialised tool existed; The need for a bespoke web application was obvious. SAR was built to improve research processes, increase the accessibility of the records, and allow researchers to better analyse the documents in the ARO archive. It addressed this gap by introducing essential functionality such as keyword search, filtering options, and result visualisation, significantly improving the research process. Advanced features, including fuzzy search matching for spelling variations and date-based filtering, further enhanced the usefulness of the tool. Team Charlie developed their version of the SAR tool using the Ruby on Rails web application framework. While this approach offered the advantages of rapid development and robust functionality, it later introduced technical challenges for the LACR team and the University of Aberdeen. The

university's technical infrastructure is predominantly based on Laravel, a PHP framework. As a result, maintaining the Ruby on Rails-based SAR application required specialised technical knowledge that is not readily available within the institution, potentially forcing the university to retrain staff or hire external developers. These limitations make long-term support impractical and expensive and highlight the need for a new application aligned with the university's technical ecosystem.

Through the completion of this project, Team Lima has redeveloped the SAR search tool, transforming it into a more robust, sustainable, accessible, and user-friendly solution for accessing the ARO corpus. The new version of SAR has been rebuilt using Laravel, JavaScript, and SCSS, a structured syntax that generates CSS, aligning the development process with current best practices and aiding future support and maintenance. This update has improved the web application's usability, functionality and maintenance, ensuring it remains reliable and accessible in the long term. It has also aligned with modern development practices to allow easier updates and support in the future. The primary critical use case of the new application is to emulate the functionality of the previous application: to enable the user-friendly search and navigation of the digital records stored in the ARO corpus. This includes providing both basic and advanced search functionalities, where users can perform keyword and phrase searches, apply filters such as page, date, language, and content type, and sort results by frequency, chronology, or volume and page. Additionally, the application will enable users to browse individual pages and perform XQuery searches. The new application also features a bespoke help page to help less experienced users utilise the SAR tool to the fullest degree. This aligns with Team Lima's goal of widening access to historical information and closing the digital divide. A well-designed front end, created following an evaluation of the existing SAR webpage, has been developed using Vue, JavaScript, and SCSS. These decisions were made to ensure a better user experience when compared to the original webpage, which has been validated through the completion of an A/B test. Laravel has been used to develop the back end of the webpage to define the server-side logic. This decision was made to streamline maintenance and adhere to the university's existing infrastructure.

Target Audience and Intended Usage

The primary audience for the SAR application includes academic researchers, historians, and students aged 18 and above, particularly those engaged in medieval studies, Scottish history, or digital humanities. This user group will benefit from access to high-precision search tools and filtering options that support the detailed textual analysis of the ARO corpus. By allowing fast and accurate access to specific records, the application promotes the analysis of the social, legal, and economic aspects of Aberdeen's medieval past. A secondary audience includes people with a general interest in history, such as amateur genealogists and the wider public, who may wish to explore the ARO corpus for personal or educational purposes. For these users, the SAR tool offers an accessible way to explore

Aberdeen's medieval burgh records, helping to bring history to life for audiences with limited access to such an extensive resource. Team Lima's redevelopment of the search tool strongly emphasised the importance of usability and accessibility to ensure that the application equally serves users with a range of technical skills and needs. Both user groups will benefit from our improved user interface (UI), which makes it easier to navigate and interact with the content, prioritising accessibility and ease of use. Additionally, a new responsive layout makes the SAR application accessible across devices. By redesigning the SAR webpage and search tool for 2025 and beyond, Team Lima has ensured that the registers remain accessible to diverse user groups for research, education and discovery.

Team Organisation

In Scrum, ownership and accountability of the project outcome are shared by everyone equally, rather than having a hierarchy. However, there are two main important roles involved in the framework. We nominated Andreas Maita to be our Scrum Master, and he guided the entire team throughout the duration of the project. He enforced and facilitated Scrum techniques and monitored everyone's progress throughout the sprints. Haziel Osunde was selected as our Product Owner, and she maintained a continuous relationship with the client to keep the entire team aligned with their needs and expectations, making sure we had a comprehensive and accurate requirements specification. All members of the group, including Caitlin Thaeler, Piotr Smialek, Holly Sinclair, Rebekah Leslie, Dermot Stelfox, and Fariha Ibnat, are part of the development team and were in charge of the design, implementation and testing of the product.

Following on from last semester, we have maintained the practice of employing effective processes rooted in the agile way of work, whilst also finding new ways to improve and build on what we know. Overall, our workflow remained largely unchanged. Consistent and established practices like weekly sprints and pair programming are central facts of agile methodology, which we employed last semester. As proven this semester and last, using these techniques has fostered collaboration, transparency among us, and efficient progress tracking. However, this semester required a shift in our team's structure, considering that our work is a continuation from last semester, our goals and responsibilities persisted, but minor goals for SAR, which were dictated by incoming feedback and our own reflections, changed.

With a change in minor goals, a structural change in our team was also needed. Division between front-end and back-end was something that was decided among us, as the tasks we needed to complete this semester fell under one of these categories. The members of the group had their sectors to work in. Working this way aligned well with the principles of pair programming whilst encouraging in-team collaboration. Despite the fact members of the team had their sectors to work in, everybody was proactive in helping each other where they could. This adaptability created a dynamic environment, enhancing team cohesion and making sure progress was not stalled in any area.

Project Management

Throughout the project, we decided to use different software tools to aid our collaboration and make the development process easier, such as:

- **Github**: this was used for multiple purposes, like storing code, version control, and working together with others. It kept all our work saved and secure in one place and was an excellent tool for collaboration. We set up the repository so that all pushes to the main branch require a pull request. Every pull request had to be approved by two other people, encouraging reviews and double-checking for errors before anything was made final.
- **Social Media**: This part was very important since we stayed connected using two different apps: WhatsApp and Discord. Both were essential tools because we stayed in touch with each other and offered frequent updates on our progress using WhatsApp. We used Discord to communicate more project-related issues and created different channels related to different aspects for better organisation. Discord was especially useful for online collaboration because we utilised its server-wide call function.
- **Visual Studio Code Live Share**: This extension allowed us to code collaboratively even when we could not be together in person. It was used multiple times while coding and allowed us to implement a key technique within agile, pair programming, speeding up the process and letting us work together remotely. It also encouraged us to use pair and mob programming during implementation, which reduced the need to have extensive code review meetings.
- **Figma**: This application was used during the initial design process to create wireframes, allowing us to start visualising the website and what changes we wanted to make to its layout. It allowed us to all be on the same page regarding the visual side of the project.
- **Jira**: A project management application which permits all team members to create, assign, and keep track of tasks effectively. This tool made it clear what tasks needed to be done, who was assigned to what, and the effort required for the tasks. This tool is currently used throughout the software development industry, so using it was not only a sensible option but one that would allow us to align ourselves with industry standards. Adopting this tool not only streamlined communication but it also ensured the visibility of tasks and made the team's goals for the week clear.

On the whole, the software used for management of the team and projects remained the same, aside from the introduction of Jira, which enhanced project cohesion.

Background

In the context of digital archive systems, particularly those focused on historical civic records, few tools offer a level of specificity required by researchers working with something like the ARO corpus. As such, the SAR application is a very specialised system tailored exclusively to the ARO archive and the needs of the University of Aberdeen. As such. It cannot be compared directly to any competitor systems except the original search tool developed by Team Charlie in 2017. The new SAR tool will exhibit substantial improvement in several key areas. The updated system incorporates modern development practices. Unlike its predecessor, which was built using Ruby on Rails – a framework the university's technical ecosystem does not use – the new SAR tool aligns with the university's technical expertise, reducing the need for external support. From a user perspective, the new SAR tool includes improvements to its UI and navigation capabilities. While SAR is indeed a specialised system, its extensive search functionality makes it a powerful tool for accessing and searching historical records, offering high utility for researchers, students, and the general public.

Related Products or Services

While the SAR application cannot be compared directly to any other system, it can be compared to similar systems that are used to search through large archives. One such tool is The National Archives:

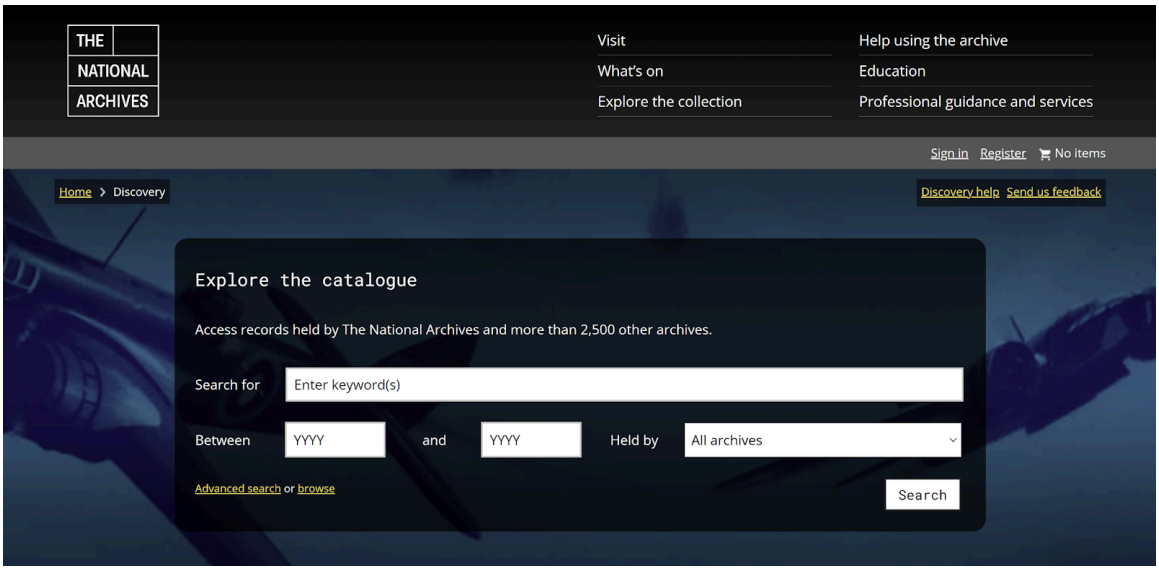


Figure 1: The National Archives home page (The National Archives, n.d)

Our group liked that the landing page contained the search tool, making it the first thing the user sees. This search tool also has basic and advanced search functionality, just like the SAR search tool. The advanced search button leads to a different page, which means the user has to wait for additional assets to load when using this function. It also has fewer filtering

options. In comparison, the SAR search tool's advanced search function is contained in a drop-down menu on the home page, making it easier and quicker to use. Our tool's search functionality is much more flexible and detailed, which will eventually result in searches that yield more precise results.

Another archive search tool that can be compared to SAR is the US National Archives:

Figure 2: The US National Archives home page (National Archives, 2008)

When comparing SAR to this search tool, a few notable differences can be identified. The US National Archives search bar is hidden in the top corner of the page, making it potentially difficult for users to find. There is also no clear advanced search function. In comparison, SAR is much more user-friendly and accessible.

Competitor Analysis

At the beginning of the redevelopment process, Team Lima evaluated the original SAR tool and identified key areas for improvement:

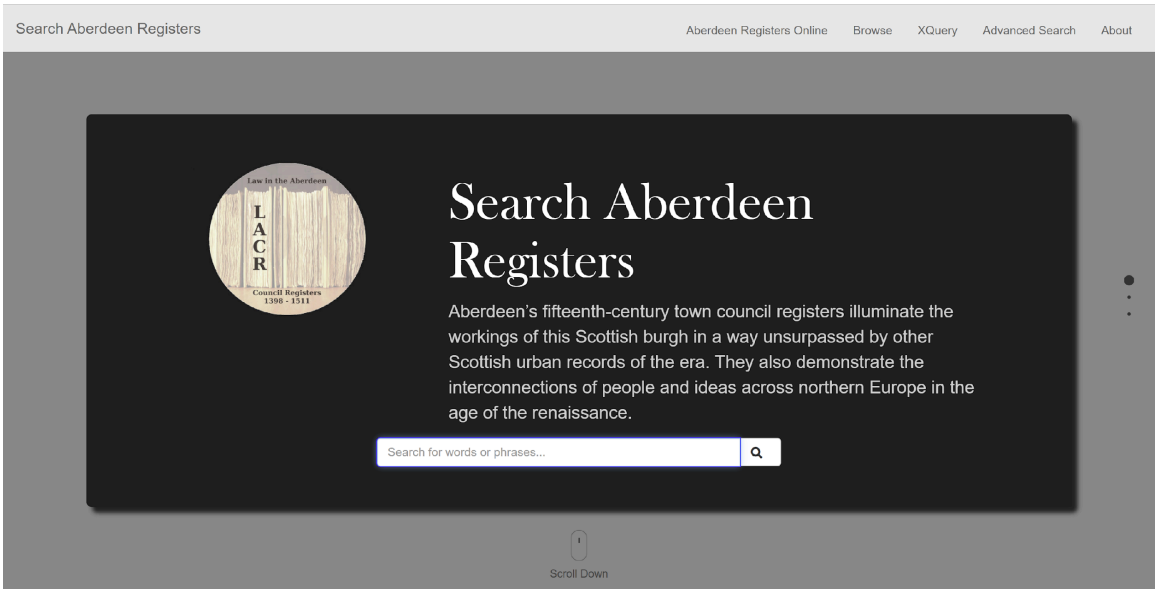


Figure 3: The Search Aberdeen Registers 2017 home page

- **UI Design**: The overall appearance of the site was described as dull and outdated. Members described animated transitions included in the browse section of the SAR 2017 site as unnecessary and stated that an improved colour scheme and layout would provide a more engaging experience.

- **Usability and Accessibility**: Navigation was described as inconsistent and confusing. After using the site, we discovered that the home page exhibited awkward scrolling behavior that hindered the use of advanced search functions. Using the browse page also required too much scrolling, and navigation could be improved with arrows or shortcuts. The majority of the site did not provide any guidance or instructions and there was no help page.

- **Maintainability**: The site was built using Ruby on Rails, which is not commonly used today and harder to maintain.

Identifying these issues helped us develop ideas that improved the tool as a whole. These improvements include:

- Combining and simplifying the search and results pages and the browse pages to reduce unnecessary steps.
- In the browse page, multiple pages were combined into one and navigation was streamlined. A page number dropdown menu was replaced with arrow buttons and a page input box.

- Tooltips were introduced to provide users with information about each element on the site, and a dedicated help page was added to provide more in-depth support.
- The site was redeveloped using languages that align with the University of Aberdeen's current infrastructure.

While the 2017 version of the SAR application provided users with the core functionality they needed to search through the records, analysis revealed that it lacked polish, guidance to aid accessibility, and a well-planned layout. After using the tool, members of Team Lima were left frustrated by its obvious shortcomings. However, this evaluation process was instrumental in informing us of our subsequent design and development decisions. Through the evaluation and redevelopment of SAR 2017, we have created an updated tool that focuses on user-friendliness, accessibility, and long-term support. Through the redevelopment of SAR, we have created an updated search tool that focuses on user-friendliness, accessibility, and long-term support.

Requirements

Analysis

During our initial meeting with our client, Dr. Jackson Armstrong, he emphasised that the central goal for the new version of SAR was to retain the functionality of the original system while improving its usability and maintainability. Most importantly, Dr. Armstrong explained that the main goal of this development process was to retain the existing functionality of SAR (2017). He also requested that we consider minor improvements and quality-of-life features that could be added to make the search tool easier for researchers and members of the public to use. Additionally, he informed us of the university IT department's preferences for implementation, particularly the languages and frameworks that should be used, and the technical standards that should be abided by, such as for security and accessibility.

This information, combined with a detailed team evaluation of SAR (2017), informed the following functional and non-functional requirements for the search tool's redevelopment. Throughout both semesters, the requirements specification has been constantly updated and refined as we learned new information or received feedback from the client and user testing.

Functional Requirements

Basic Search

The main feature SAR is its basic search interface in the form of a simple search bar, which allows users to quickly search for words, characters, phrases, and patterns within entries of the Aberdeen Burgh Registers.

- **Search Methods:** Users should be able to pick from a selection of search methods, which specify the type of input. These consist of the options keywords, phrases, regex or word start, word middle and word end.
    - If the user selects word start, word middle or word end, the search results should match entries that contain words with a specified string of characters either at their start, middle or end.
    - Keywords method will match individual words in the search to words in the entries. The keywords in the search should be separated with spaces, and each of these will be searched for individually
    - The phrases method matches consecutive words or expressions.
    - Selecting regex should allow the user to search for patterns in the entries using regular expressions.
- **Spelling Variance:** The user can select how much spelling variation or flexibility they want in their search results. Search results can be an exact match or have a

similarity equal to or higher than the specified variance. Variation can range from 0 to 40%, with 5 options available (None, Some, Moderate, Considerable, Significant)

- **Sorting:** Several search options should be available, including; frequency within result, volume and page, chronological and finally best match, which is a new addition not featured in the 2017 version of SAR
- **Case Sensitivity** is another newly introduced feature that can be toggled on or off.
- Autosuggest: the search bar should provide a typeahead suggestion.

Current Status: All features with the exception of autosuggest have been successfully implemented

Advanced Search

The advanced search functionality has been designed to give users the option to refine their searches using filters and specific search criteria.

- **Features and Filters:** The advanced search menu, accessible to the user via a dropdown button located below basic search on the home page, should include filters for parameters such as:
    - **Language:** A drop-down menu where the user can specify either Latin, Middle Scots, Dutch, Multiple or Any.
    - **Volume:** Users should be able to select from specific volumes using checkboxes. There is an additional button for toggling all options on or off. Volume 3 should be excluded as it does not exist.
    - **Page:** Users can select individual or multiple pages to search. The page numbers are to be separated by commas and do not need to be sequential. The user should only be able to search for valid, existing pages.
    - **Entry:** On 2017 SAR, Users can search for a specific entry (paragraph) in the corpus. However, this feature has been removed as per recommendation of the client.
    - **Date Range:** The user can input a from and to date (From: date To: date) to search within a specific date range (1390s to 1510s), selecting a year only, year and month or the full date.
    - **Entry ID:** Users can search for specific documents, or a set of documents using a unique identifier, or part of a unique identifier. For example, searching ‘ARO-8’ would produce all documents found in volume 8 of the ARO corpus.

Current Status: All advanced search filters have been implemented, however Entry ID does not currently allow for a search with only part of an identifier, only the whole ID.

Search Results

When the user submits a search query, the system should display the output generated by the search algorithms.

- **Output and Display:** Search results are displayed as a list of record transcriptions, complete with the relevant metadata (volume, page, date, language and entry ID).
- **Results Per Page:** The user should be able to specify the number of entries displayed on each page of the results, and toggle between these pages.
- **Linking to Browse Page:** When the user selects an entry while searching, they should be directed to the corresponding Browse page.
- **Additional Features:** The current edition of Search Aberdeen Registers does not allow the user to highlight and copy the transcription text in a search result so this should be amended in the new version as per request of the client.

Current Status: All of these features are now functioning.

Browsing

The browse functionality should allow users to explore the records systematically by volume and page, displaying a high-quality scanned image of the original manuscript, alongside detailed transcriptions of all the entries of that page with related metadata such as volume, page, date and document ID.

- **Side-by-Side Layout:** The image would be displayed beside its relevant transcription. This design ensures the reader can see how each transcription aligns with the content in the original manuscript, emphasising readability by presenting text and images together for comparison.
- **Zoom Functionality:** The user should be able to zoom in on an image by hovering over it with their cursor.
- **Export Options:** Users should be able to select entries and generate PDF downloads for specific transcriptions.
- **Toggle to XML:** Toggling between text and XML views of the entries

Current Status: While the majority of the above is implemented, and the ability to toggle between different volumes and pages is fully operational, the image displayed for each page is a placeholder image due to issues with getting access/rights to the images.

XQuery Search

More advanced or specialised users can perform XQuery searches on a dedicated webpage, replicating the functionality of the XQuery search function included in the original SAR system. This feature is for users that are comfortable with XML querying.

Current Status: The functionality for XQuery Search is fully implemented.

Non-Functional Requirements

Performance

Our users should never have to wait idly for the website to process. The system should be fast even when dealing with large files, thousands of images or complex searches. Through the development of our system, we have ensured that the search times are quick and responsive at the front end, and the website works rapidly when loading and switching between pages. This is mainly due to the lightweight nature of our application, having made sure not to run overbearing code or plugins.

Security

Security was an important consideration, especially because the application may eventually be hosted on the University of Aberdeen's online infrastructure. We read up on the OWASP Top 10 Web Application Security Risks (OWASP, 2024) and educated ourselves on the most significant security risks to web applications. The application does not feature any sort of account system, so there is no risk to user data, however, other security measures are in place, such as the routing system that Laravel provides.

Usability

The usability of the redeveloped SAR application was a huge consideration. As developers, Team Lima wanted to adhere to industry standards and guidelines such as the WCAG 2.1 requirements and the BCS Code of Conduct. We were also committed to developing an application that is usable by as many people as possible. Closing the digital divide was an important requirement. We made various changes to the UI after evaluating the original SAR web page and identifying areas for improvement.

Reliability

We found that the previous version of SAR was often not reliable, with features not behaving as expected. For example, form options that did not function, and inaccurate search results. Our website has been designed to be more dependable and consistent. Search results are also much more precise.

Accessibility

To meet the guidelines set out by the university, the WCAG 2.2 web accessibility standards were researched and have been followed to the best of our abilities (W3C, 2023). These guidelines, and how they have been achieved, are as follows:

Perceptibility

Text alternatives for all non-text content have been provided for images, icons and multimedia. This not only helps visually impaired customers who use reader screens but

also helps if some part of the server fails to display an image. An alternative can be presented regardless. This can help with maintainability, as well as bug fixing.

The website complies with contrast standards to help visually impaired users. This is tested on Google Chrome developer tools. It also helps create a flashy and bold design, using high-contrast features, white space, and minimalism.

Operability

All of the SAR website functionality is fully keyboard navigable, and the page has a logical tab order that makes sense to users who navigate web pages via their keyboard.

Understandability

The website is readable, we have used appropriate sizing, things are properly spaced out, and appropriate font sizes have been used.

Our website works in consistent and predictable ways; our team has worked on a consistent navigation bar that always stays at the top of the screen to provide accessible navigation, which works well with keyboard input handling.

There are so many ways to make a website more accessible to a wider audience. Throughout our development process, we strived to ensure that we adhered to as many standards as possible. However, there are still things to be aware of and areas to improve upon in the future, such as:

- Focus Indicators: Visual indicators for focused elements, such as buttons and links, can help users identify the current point of interaction within the application.
- Error Identification and Assistance: Form errors must be identified with concise, accessible error messages to help users avoid and correct mistakes.
- Screen Reader Compatibility: Dynamic content and user interactions should be designed with screen reader support, including proper use of HTML roles, labels, and ARIA (Accessible Rich Internet Applications) attributes where applicable.

Maintainability

During the development process, the maintainability of the new SAR search tool remained a top priority. It is the main driving force of the project as a whole and will ensure that SAR can continue to be used as a valuable resource well into the future.

The transition from Ruby on Rails to Laravel was pivotal in improving the maintainability of SAR. Laravel aligns with the University of Aberdeen's pre-existing technical ecosystem and redeveloping SAR using this framework reduces the need for external help and helps aid future updates. Additionally, in-code comments have been used to help future developers who may want to build upon our work. Ideally, the turnover should go smoothly.

Scalability

While the ARO corpus is finite and it is unlikely that it will be updated, the architecture of our application still supports potential expansion. Additional records could be easily incorporated into the online archive. Extra search classes and features could also be added to the existing functions if needed.

Interoperability

The application processes XML files in line with the Text Encoding Initiative (TEI) standards (Text Encoding Initiative, n.d.). This is a rule set which attempts to standardise how certain text (XML) is formatted and laid out such that meaning, paragraphs, metadata and other content are easily recognisable and addressable rather than each person having their own personal methods which may otherwise cause a lack of familiarity and incompatibility.

Design

Design Overview

Search Page

The search page design was changed according to the user's needs and to obtain a better look and functionality in regards to the previous website. Other than creating another page for the advance search we created a drop down button which opens all the functions of the advance search. This is done so we don't have to scroll down as the previous website was very laggy so we opted for a much better option. One of the main perks of our version is the url is responsive for both search and browse pages. In which the user can directly type the content they need in there. on top of that the searches are also more accurate.

Browse Page

We decided to redesign the browse page as it plays a big role in the updated version of the Search Aberdeen Register website ,also commonly referred to as SAR. We made this decision based on all the issues found in the earlier version of the site made by the previous group, and we tried to create a more intuitive and accessible user experience. To achieve this, we merged two pages, the browse page and the page viewer page, into one single interface. This helped us address several redundancies and usability problems that existed in the earlier version of SAR.

When we were testing the original site, we found a few inconsistencies and limitations that made it harder for users to access and interact with the data properly. The original browse page allowed users to choose a volume and a page number, but selecting multiple pages did not work as expected as it often failed to display the correct number of pages in the page view section. It does show the first page selected, but the problem occurs when the user tries to choose multiple, as the system doesn't even allow that and when it does it takes you to the last page selected. So the option is either choose one page or all the pages in that volume.

Also, the functionality between the browse and page view pages overlapped with each other, as both allowed users to select volumes and pages, but it was not clear what the intention was behind the browse page, as it didn't had much functionality other than browsing through each volume or giving the user option to go the page viewer. As first-time users ourselves, we found this confusing to navigate. Because after selecting the volume and pages, nothing happened until we clicked on the page viewer button.

We found the process confusing and misleading with multiple steps just to go to the designated page. This process for the researcher can be frustrating, especially when trying to browse through the pages quickly, even for the new users who are unfamiliar to this functionality can be quite time-consuming trying to understand the basic functionality. Another issue was with the mobile view. When trying to view the transcript and the image,

it only showed the volume and pages, not the image, which as a researcher, may cause inconvenience if they want to check something quickly.

To fix all the issues outlined earlier, we decided to merge the browse page and the page viewer page into one. As the functionality of both is quite similar. The layout is now much more intuitive and accessible as we tried to get rid of the redundant functionality and pages. So when the user selects a volume and page, they are shown the high-quality scanned image along with the corresponding transcription. By doing this we remove the need to switch pages and wait for separate views to load. This is helpful also to the system as it doesn't need to keep loading multiple pages and It improves usability and matches what users, especially researchers, would expect from a historical archive website. As this has been implemented through a single-page application, the waiting time is also reduced, making the experience for the users much more smoother.

With the new design, the application now has fewer pages to navigate, which makes everything feel smoother and less confusing. Instead of spreading functionalities across different places, everything works from a single interface. This avoids the confusion we experienced in the earlier version. This new implementation will be a great help, especially for the new users.

In the older site, users had to scroll through a long list of page numbers to find what they needed. Some volumes had pages ranging from three hundred to a thousand pages. This was a frustrating option for all users, especially those doing research. Now users can select a volume from a dropdown menu, move through pages one by one, jump straight to the first or last page, or type the page number and go directly to it. This helps users have more control without making the interface feel crowded. It has been designed to be easy to use and work well for people who are casually browsing or doing serious research.

Throughout the entire process, we kept accessibility in mind, as we wanted the experience to be as smooth as possible. One feature we decided to keep was the side-by-side layout, which shows a scanned manuscript image and its transcription. This was something the client liked from the old site, and we agreed it was very useful. It helps users compare the original handwritten text with the transcription. This is especially important for researchers who want to verify how accurate the transcription is in case any inaccuracy is found. Each transcription includes important data such as the volume number, page number, date, and document ID. The layout helps users see how the content connects between the scanned page and the transcription.

Another feature we focused on improving is zooming. Being able to zoom in on a scanned image is very important when examining historical documents, especially those written in old script or faded ink. On the desktop, users can hover their mouse over the image to zoom in. On mobile or tablet, they can zoom with their fingers to be able to see the high-quality image. We made sure that zoom works smoothly on all devices and that the controls match what users are familiar with. It may seem like a small feature, but it adds a lot to the user experience, especially for those studying the original handwriting or the layout of the manuscript pages.

We also improved how users can select and export transcriptions into PDF format. In the old system, this was done through a confusing dropdown list on the browse page tab, which was

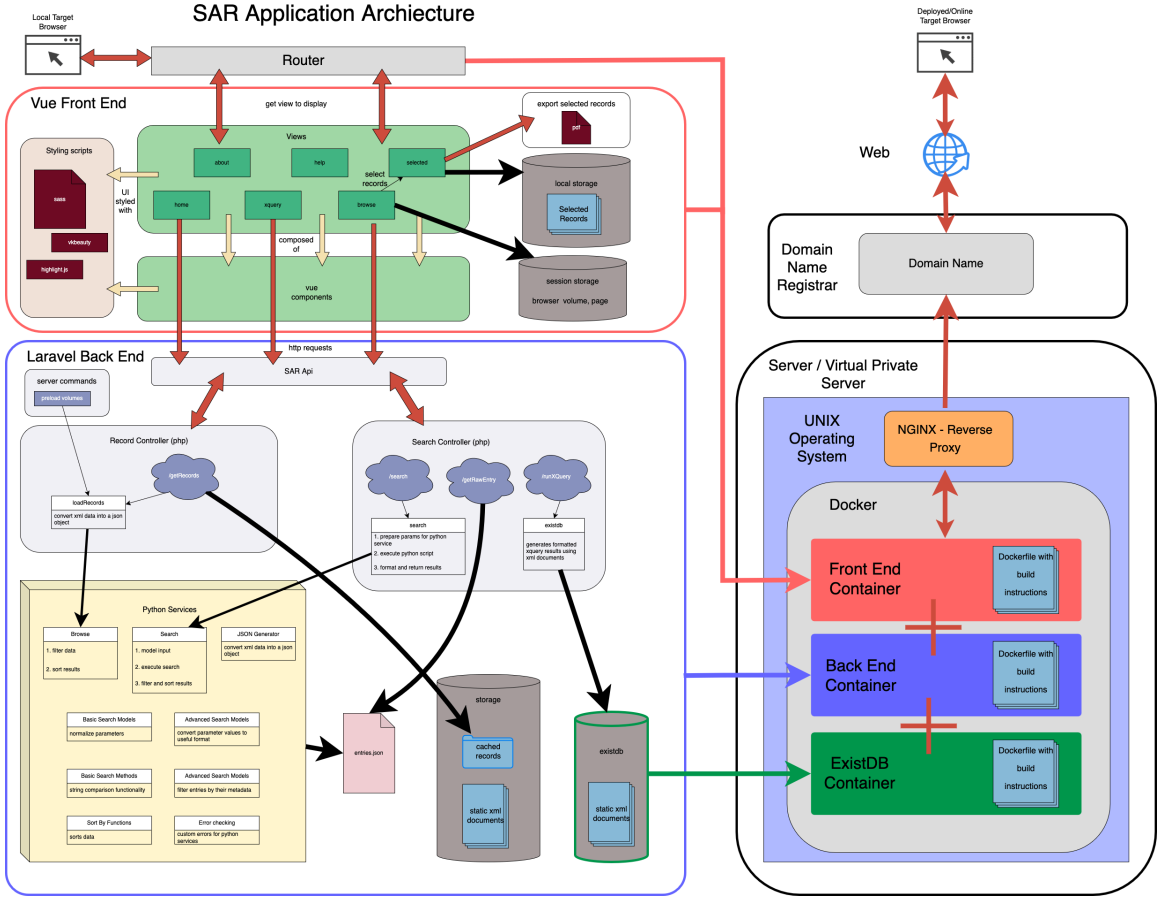
not very visible or easy for first-time users. In our version, each transcript now has a checkbox so users can easily select the entries they want. These selected entries are added to a dedicated tab section of the site called "Selected."

Once a user selects a manuscript, it gets added to that page, and a "Selected" tab appears in the navigation bar. If manuscripts or transcripts are removed from the selected page, the tab also disappears. This makes users more aware of when they are choosing or removing something. The selected page also has the option to remove all entries or download them as a PDF file. These changes make the process simpler and more transparent. Even users who are not technical can clearly see what they have selected and what they can do next. To make the whole experience more accessible, we used tooltips and other accessibility features to ensure everyone can use it and understand what each feature is meant to do. The tooltips appear when users hover over buttons, dropdowns, and other interactive elements. These small pieces of text help users understand what each feature does without needing to visit a help page. This is especially helpful for new users or anyone with accessibility needs. We wanted to create an interface that is self-explanatory and user-friendly so that users never feel lost while using the site.

One technical challenge we faced was hosting large image files. Right now, our prototype uses placeholder images stored locally because we do not have the resources to scale and host all the scanned images, which are more than sixteen gigabytes. The site we are using for hosting has a maximum limit of one gigabyte, which is already taken up by the rest of the website. However, we have planned a solution for full deployment. When the university takes over, they can store all the images on their own servers. Our code is already set up for this, and it only needs a few adjustments to connect to the image folder.

From the beginning, it was very important to match the client’s requirements. Since they were already used to the previous interface, we did not want to completely change how the site looked or worked. Instead, we focused on fixing what was broken and tried to make the experience better and easier for everyone.

Architecture



User Interface Design

Our UI design philosophy was guided by two complementary goals: to create an interface that showcases the historical significance of the Aberdeen Burgh Registers, and prioritise a modern, accessible, and functional user experience. The result is a website that bridges the past and the present, with a design that is visually rooted in the 15th century and an interface that fully aligns with 21st-century usability standards.

Design Vision

The aesthetic direction of the new SAR UI is inspired by the architecture of Old Aberdeen and the intricate calligraphy that can be found in the Aberdeen Burgh Registers. By incorporating stylized elements derived from the records themselves – such as carefully

traced text samples and colour palettes that evoke aged parchment – we created a visual interface that highlights the significance of the records. This is combined with modern UI practices like clear typography, a responsive layout, and easy navigation to create a sleek and functional application. Care was taken to preserve the readability and accessibility of the webpage, which meant avoiding the overuse of decorative elements that could overwhelm the user. A key guiding principle during the development process was: “design for all”.

Accessibility and Compliance with WCAG 2.1

Throughout the development process we aligned our decisions with the requirements outlined in the Web Content Accessibility Guidelines (WCAG) 2.1 at Level AA, the mid-range compliance level that promotes strong accessibility and addresses common barriers for users with disabilities and satisfies both A and AA criteria (Rivenburgh, 2021). These requirements included:

- **Colour Contrast**: All text and interactive elements meet or exceed the minimum contrast ratio.
- **Text Readability**: We used readable typefaces, logical font sizes, and avoided decorative fonts for body content.
- **Keyboard Navigation**: Interactive elements like dropdown menus, links, and buttons were tested to ensure full keyboard navigability.
- **Screen Reader Support**: Semantic HTML elements like <main> and appropriate ARIA roles were used to ensure compatibility with assistive technologies like screen readers.

Inclusive Design and Widening Access

To comply with the BCS Code of Conduct, we designed an application that is usable by the widest possible audience. We accounted for physical disabilities, different levels of digital literacy, and gaps in historical knowledge. To support users unfamiliar with archive search tools, we implemented the following features:

- **A Bespoke Help Page**: This page explains key functionality across the application, describing how to use basic and advanced searches, the browse page, and XQuery search.
- **Consistent Layout**: We implemented a consistent design and navigation layout to help all users navigate the site with confidence.
- **Tooltips**: The addition of dedicated tooltips is designed to help users unfamiliar with the SAR tool, even if they do not visit the help page.

- **Responsive Design**: The web application was developed with mobile users in mind and leverages responsive elements to ensure that the SAR tool is accessible on all devices.

Iterative Design and Agile Development

During the redevelopment process, the new SAR tool's UI design underwent many changes as part of our agile approach to development. Key design stages included:

- **Wireframes**: Low-fidelity wireframes were developed early to help us explore layout options and identify key navigation improvements. They were shared with our client to gather feedback.

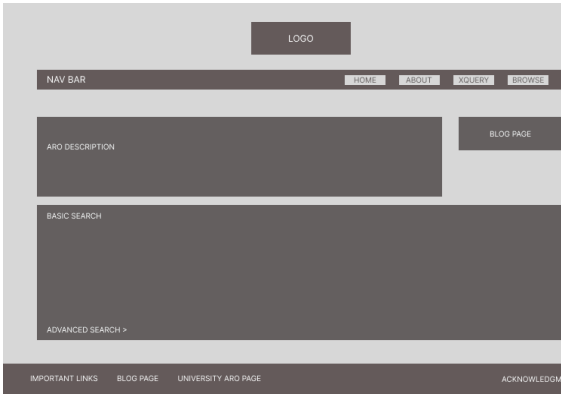


Figure 4: Homepage Wireframe

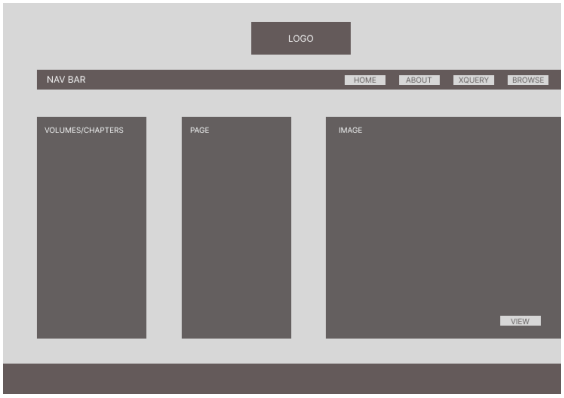


Figure 5: Browse Page Wireframe

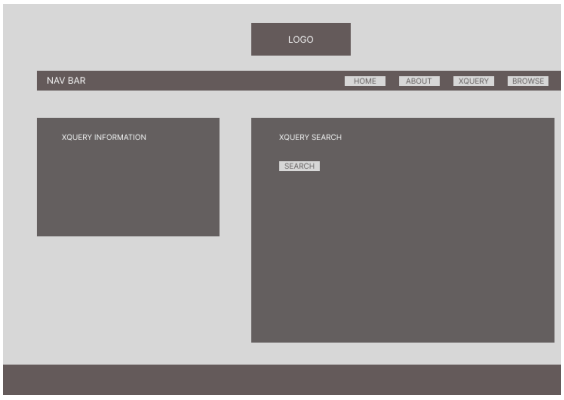


Figure 6: XQuery Page Wireframe

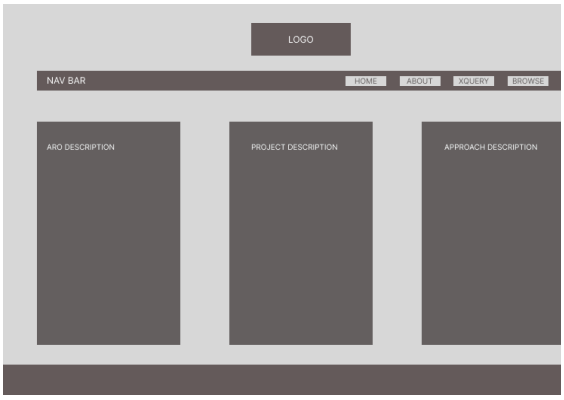


Figure 7: About Page Wireframe

- **Mood Board**: We created a mood board to establish the visual direction of our design, including a colour palette, fonts, and imagery.

- **Design Iterations**: More than 15 visual prototypes were created and modified by continuous feedback and the need to address issues like copyright, legibility, and layout accessibility.

A full visual history of Team Lima's design iterations (Figures X – X) documents the SAR tool's evolution from conceptual design to polished product. Each new iteration reflects the lessons we learned about usability, accessibility, and design. Significant iterations are shown below:

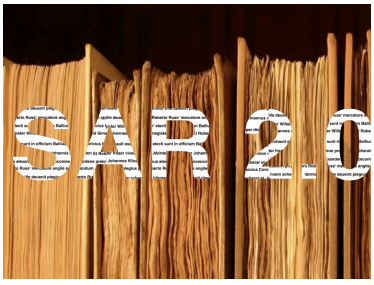


Figure 9: UI design iteration 1

Initial design featuring the original books. The logo lets

Figure 10: UI design iteration 2

Trying a more creative approach: using the text as

Figure 11: UI design iteration 3

Used the original text from the books to create a

you look inside to see the text.

a background, however, this came out very cluttered.

backdrop for the title, however, this looked quite cluttered and not very readable.

Figure 12: UI design iteration 4

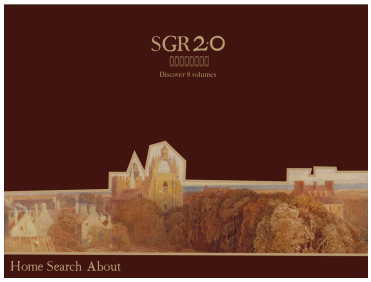


Figure 13: UI design iteration 5

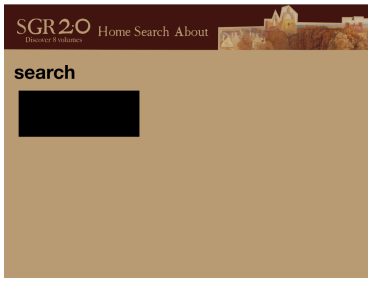


Figure 14: UI design iteration 6

Kept the logo and stretched it out, incorporating features of Old Aberdeen. Added a line around the feature to bring it into the design.

Made the logo smaller and added a nav bar, we felt that this design more closely aligned with our desired aesthetic.

We were satisfied with the design, but we didn't have the copyright for this image and did not find a suitable copyright-free alternative.

Figure 15: UI design iteration 7

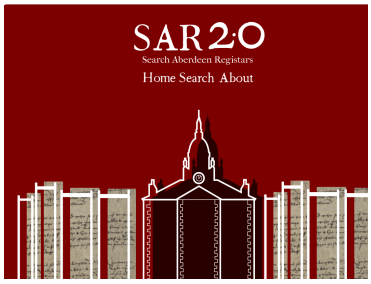


Figure 16: UI design iteration 8

Figure 17: UI design iteration 9

Instead, we decided to trace the townhouse where the volumes are stored to incorporate as a design element.

Changed the colour scheme to improve contrast and perceptibility. Incorporated elements of the original works.

Decided to simplify logo design bringing it into a circle, which made the webpage less cluttered and simpler to understand.

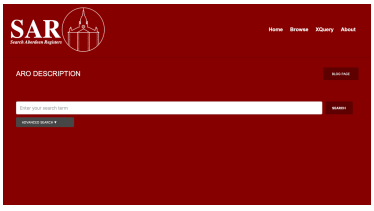


Figure 18: UI design iteration 10

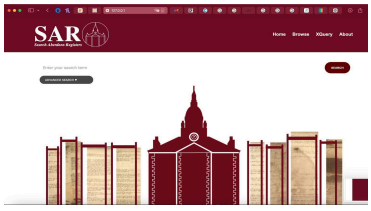


Figure 19: UI design iteration 11

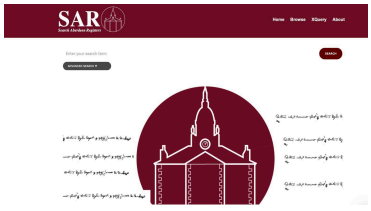


Figure 20: UI design iteration 12

Brought the new changes onto the partially functioning webpage, however the red colour did not provide good contrast between sections of the webpage.

Incorporated the design elements used earlier, and changed the content area to white, to improve readability and accessibility.

Found the use of the old books to look cluttered and distracting, decided to trace text from the original passages.

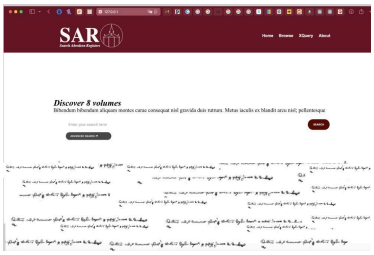


Figure 21: UI design iteration 13

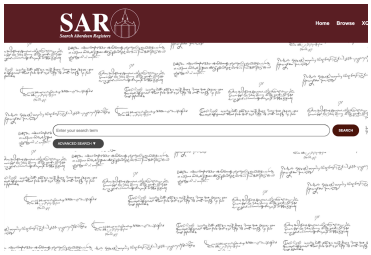


Figure 22: UI design iteration 14

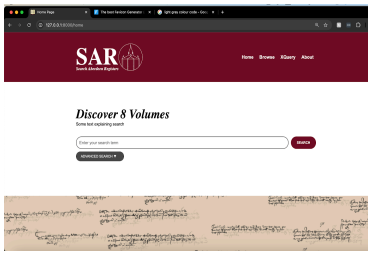


Figure 23: UI design iteration 15

Decided to remove the element in the middle and instead added more elements from the traced texts, this looks a lot more visually interesting and cohesive.

Tried it out as a background, however, was not happy with this iteration as it would not be very friendly to visually impaired users.

Brought the new changes just to the bottom to create symmetry with the top and create white space. Made the background of the text element match the colour of the book pages.

The final design of the SAR tool that can be seen on the deployed version of our application meets our original goal: to pay homage to the historical significance of the Aberdeen Burgh Registers while maintaining accessibility and usability.

Coding and Integration

Implementation Decisions

Docker Integration

One of the main reasons we switched focus from a purely local implementation was due to the benefits of being able to serve our project to the wider public. This allowed us to conduct more conclusive testing to discern the efficacy of our project via testing methodologies such as A/B testing to compare the previous implementation to ours (link here why deployment is good). Another reason as to why Docker was chosen was due to the goal of providing a ready-made "plug and play" system for the University tech. department, who have communicated with us that the current implementation uses Podman — which is a system built upon Docker and as such has similar skill sets and production-ready environments that need little to no alteration if Docker is fully implemented to port over to the more familiar system of Podman.

Docker is a system which allows an app to be split up into manageable compartments called "containers" such that if a certain aspect of the app fails, i.e., the back end, the entire system does not need to be re-made or rebooted, only the affected container. The way Docker systems work is a declarative system and even support for YAML via Docker Compose (link here). Such a feature allows an app to not only be infinitely reproducible such that it can be installed and ran identically on systems of the same type, but also in correct configurations it permits an app to be entirely system and architecture independent — which is an extreme advantage in a production environment where system hardware and/or architecture may be subject to change for a myriad of reasons such as upgrades, failing components or merely development on a different system.

Such a declarative environment allows for safe reproduction and mitigates risk of downtime or permanent data loss given that the containers contain a copy of the code and have no effect on the original data by default unless granted this in cases of updating databases etc (link here talking about how many real systems and companies use docker). As such, our project was very successful in replicating a real-world environment that is likely to be developed and experienced in the same manner, thus we believe the university should be well suited to replace the previous system with our updated implementation.

An image is a link placed in the declarative Docker files to a maintained set of installation instructions — which is what systems such as Node and Python are installed into containers with. One area of difficulty was to properly utilise the ExistDB image to replicate the same environment with the XML files required for XQueries — one of the reasons being that the official documentation as of writing today was not to a level of detail that would allow us to

understand how to integrate it into our system. A work-around for this was to install ExistDB into the root of the repository, gitignore it such that it isn't uploaded to GitHub, and then copy it into its Docker container.

Whilst this did produce the desired results of allowing the frontend and backend to communicate and XQuery to properly be executed, this goes against the declarative nature of Docker as if there were any updates to the version of ExistDB it would be more difficult to re-install this into the root as opposed to just changing one line in the Dockerfile for an update. However, logically this is the same thing as pulling the image but with extra steps and slightly inflating the size requirements.

Deployment

Deployment was conducted on a Virtual Private Server (VPS) (link here to talk about it) with similar hardware and software to that of the current implementation. We were told that the current system is on an operating system of Red Hat Enterprise Linux 9, which is based on Debian Linux - and as such we decided that due to the similarity in systems and the broad range of compatible software Ubuntu was the ideal choice for a development environment, this had the added benefit of our team members who were on Windows 11 Systems to carry out their workload via Windows Subsystem Linux and have a working environment. Our VPS was hosted with Oracle Cloud and located in the UK to benefit from both the lower latency but also the associated protections of a local system such as data protection laws. This VPS had 4 OCPU, 24 GB RAM and 50 GB HDD storage which is more than comparable to a production-ready system and gives us a good indicator on the performance and as such areas to improve upon for when this is to be deployed in and utilised in the stead of the original SAR system.

This system then only required NGINX, docker and a firewall as well as their dependencies to be installed. The use of docker allowed for fewer dependencies of the project, such as Laravel to be needed for installation on the VPS, thus making deployment much more straight-forward. The firewall used was Uncomplicated Firewall (UFW) which allowed for security measures to only expose required ports such as ssh (for development) as well as 80 and 443 for HTTP and HTTPS.

Cloudflare is the chosen Domain Name Registrar used which allowed us to pay for a domain and set up the IP of the VPS to be mapped to this domain via a DNS so that users would have to remember simple words as opposed to IP numbers, making it human-readable. This also aided in the strive for security as bad actors would have to work harder to find the server IP address than usual.

NGINX is a service used to expose a web app to an owned domain; a configuration file is edited with the appropriate chosen domain name as well as the ports forwarded etc. The certificates which allow you to verify that you own a domain and can deploy your app to it were acquired via a python-based tooling called Dependabot which then further allowed

SSL certificates to give modern secure HTTPS connections. The NGINX configuration file is detailed as to how to set up in the root of the repository through a file called Deployment_Instructions.md which will work with any domain and system as long as the ones that correlate to your VPS IP address and owned domain name are substituted.

LIBRARIES

PHP Laravel [https://laravel.com/docs/11.x/routing]

VUE [https://vuejs.org/guide/introduction.html]

AXIOS [https://axios-http.com/docs/intro]

SASS [https://sass-lang.com/documentation/]

existDB [http://exist-db.org/exist/apps/homepage/index.html]

SequenceMatcher
[https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher]

Python Levenshtein [https://pypi.org/project/python-Levenshtein/]

RapidFuzz [https://rapidfuzz.github.io/RapidFuzz/Usage/fuzz.html]

Team Lima SAR Repository [https://github.com/GroupLima/SAR-Repo]

Team Charlie LACR Search Repository
[https://github.com/team-charlie/lacr-search]

Instructions for Running the Laravel Project:
https://github.com/GroupLima/SAR-Repo/blob/main/README.md

Vue and Laravel

Vue.js and Asynchronous Handling:

Vue.js is a JavaScript framework for building front-end UIs. In Vue, you can start simple and then progressively add tools and features to build complex web applications. At its core, Vue provides a way to build components that encapsulate data or state in JavaScript and connect that state reactively to a template in HTML. These are called declarative views because the same data inputs always produce the same output in the visual UI.

Searching asynchronously has been implemented using Axios and Vue.js to collect data from the Laravel backend without the need to refresh the page or block the user interface. When a user is searching on the website, Axios works in the background by sending the data as a GET request. All this data is then processed in Laravel's SearchController, which validates the input and returns a JSON result which includes the search result or the validation errors. On the front end, when a search request is made, the response is processed dynamically and used to update the user in real time. This is done so the application remains interactive for the user - while waiting, the user is still able to use other elements present on the website. The main functionality of asynchronous request handling is that there is no interruption for the user, so the user can be satisfied with an easy and responsive experience.

Collecting data is made easier in Vue.js by using its reactive data object and the v-model directive, which binds user input (like text fields, dropdowns, checkboxes, radio buttons etc.) to Vue's state variables, which are stored inside the dataStore method, and to the backend by Axios. It makes sure that it goes through the specified validation rules before running search logic. This is done to ensure real-time updates to the collected data whenever an input is modified by the user.

Timers and callbacks have not yet been implemented but they can be introduced in the future to improve user interaction and to optimise server interaction. Some of the changes to make this happen can be using debouncing, which delays the search request until the user has stopped typing for a certain period. This is helpful as it avoids unnecessary requests for every keystroke. Another one that can be implemented is throttling as it ensures that requests are sent at regular intervals rather than flooding the server during user inputs. Additionally, callbacks can be used to define specific actions after a server response like updating UI or handling errors.

Loading data without reloading the page is another benefit of using Vue.js reactivity and Axios, which avoids reloading the whole page. When the user submits a search, the results are fetched asynchronously from the server and stored in Vue. Through Vue's templating engine, the results are rendered using the v-for directive in the DOM which allows easy updates. This is a very important method as it makes the user experience better by showing the updated version while still storing the current page state. As mentioned briefly in the previous paragraph, Axios asynchronous functions make sure the rest of the application stays functional while the data is being fetched, and the benefits related to this are reduced server load and bandwidth usage as only the important data is being transferred as it doesn't reupload the entire page.

SCSS

SASS is a popular framework used to manage CSS that allows developers to nest CSS elements inside one another, which can be used to mirror the structure of HTML. This makes style sheet code more readable and easier to maintain. With SASS you can also define global variables. For example, creating a consistent design with regulated body text is simple, just define a $body-font variable and assign it where required. SASS files are changed by developers who then generate a corresponding CSS file using the terminal. This process can also be automated using the "Watch SASS" command, which can be integrated into VSCode with the SASS extension. SCSS is a form of SASS that is more syntax-oriented, requiring the use of semicolons and brackets. This was Team Lima's preferred framework because it is more structured, making it easier to understand. The improved readability and maintainability of SCSS are why it is the university's preferred framework to manage CSS and why we chose to implement it during the development of our application.

Coding Integration

In this section we will go over exactly how each of the systems in the SAR web app was implemented, how they work together, and specific design decisions we made throughout development to uphold client requirements. To promote ease of future integrations, we aimed to create an infrastructure that's strongly decoupled, Our app is designed so that components can be easily switched out or updated individually without affecting the other parts of the system while still being cohesive enough to be maintained as a single deployed docker project.

Frontend

Single page application (SPA)
design to navigate between pages without having to reload the window. Means we can keep navbar and footer as a persistent layout (no need to remount), but selects which components to mount onto page depending on the route (search fields component if home page, browse content page if browse page, xquery stuff component if xquery page, etc)

Currently does unfortunately reload for search, but plan to eliminate window reload in the near future

A Modular Design

After submitting our first report, we decided to redesign the entire Search Aberdeen Registers website and rebuild the front end as a Single Page Application (SPA) using Vue.js. This decision was based on several factors, such as improved performance, easier navigation, and a more modern and responsive experience for users. Previously we also used Vue.js but used CDN (Content Delivery Network) in where we didn't need to install

Vue.js but not all the functionality were available, but it was a great way to enhance the html based components.

We chose to implement an SPA because it allows content to be rendered dynamically on the client side. This means users can navigate between pages and interact with features without constantly needing to reload the entire site. This approach is particularly useful for our website, which works with a large amount of data. Reloading the page every time would slow down the site significantly and negatively affect the user experience, which is something we wanted to avoid. It's also important because the site is research-focused, and users often switch between pages rapidly while working. Having to wait longer than they need may cause inconvenience to their work.

For the SPA structure, we organised the project using a component-based architecture. We separated our files into two main folders: views and components. The views folder contains the top level pages, such as BrowseView\.vue, HomeView\.vue, SearchView\.vue, and HelpPageView\.vue. Each of these view files is connected to a route defined in router/index.js. Vue Router handles the navigation between these routes, which allows users to move smoothly between different sections of the site, such as from Browse to Selected, without needing to reload the entire page. This makes the experience faster and more usable, as everything happens in the background without interruptions and is abstract from the user.

The components folder holds all the reusable elements used within the views. For example, BrowseContent.vue contains all the core functionality for the Browse page. Other components, such as Navbar.vue and Footer.vue, are shared across multiple views. This is done to improve the modularity of the code and makes the project easier to maintain and scale. It also helps avoid duplication, ensuring that the app stays organised and efficient as it continues to grow, as every single update adds up.

Browse: created reusable components such as Record and RecordList that can be nested inside one another to create html that is easy for the developer to navigate through and debug. Vue has v-models which are useful for intuitive binding between parent and child components, allowing data values to be passed down from parents emitted from children.

Script first, template second: in most of our vue scripts, we have a consistent structure using the setup composition api. All of our functions are organised neatly at the top, with an html template below the script which utilizes the script functionality. Having a consistent code structure has helped the whole team grasp the code being written, as each new component we create is somewhat familiar. It also speeds up debugging because we have a good idea of where to start looking when receiving console errors that involve either html, javascript, or sass.

Syncing Pages In Browse
Some of the pages from SAR records are missing, so this came as a challenge when implementing the navigation for pages in the Browse view. Our solution was to track both the index of the page and apply functionality to retrieve the actual page number according to the real document dynamically using a vue reactive state. As a result, we were able to update the records on the right hand side of the browse view in real time, where the page number in the input bar fetches the corresponding records as you type. We have tested it thoroughly as explained in our front end tests, to ensure that inputs are extremely failure proof. The code has been written to check for the nearest page when the user types in a missing page or mistypes a character or two. Furthermore, even the url query recognizes your input and handles the query value the same way, ensuring consistency across the application. Lastly, the browser cache stores your latest browse values in session storage, so the values in the browse input as well as the url are always up to date and as dynamic as possible.

Loading animation
Users will have more patience if they have something to be "entertained" by while they wait for records to load. Ever since record caching was implemented, this has rarely been needed though. To prevent users from trying to navigate pages while records load, we disable all the browsing navigation components and display the loading animation panel in its place. Makes it more obvious to the user that they need to wait and be patient for their records to load. Having input fields will tempt them to spam buttons that send useless requests to the server, creating more possibility for their browser or our server to crash, as we haven't dealt with rate limiting yet.

Date picker (frontend)
Before: we were using built-in date input, but this was set to 2025 and so the user has to either type in the dates or click back arrows in the calendar to reach the ancient Scottish dates which is annoying. The date input does not allow partial dates like just the year. It REQUIRES a year, month, and day, and will autofill if the user does not specify. furthermore it does not allow the developer to select a custom default date which is annoying, rather it fills it in with an arbitrary number such as month=4, day=16
After: created a custom DatePicker component which provides dropdowns of a custom range of year, month, and day. Allows users to specify partial dates, but obviously requires year if month is to be selected, year and month if day is to be selected, etc. the date picker range is from 1398-1512 to correspond with the range of dates in the registers. Allows both selection and typed input. The user can also specify dates or partial dates in the url. DatePicker component emits immediate changes in date to the SearchFields component using v-model.

Routing & Queries

Http Requests: FetchAPI vs Axios:

	axios	Fetch api
pros	supports a wider range of browsers (handles CORS (cross-reference resource sharing) and prevents CSRF) better than fetch, automatically parses responses into JSON format, simpler syntax, handles errors better, progress bar for download requests, offers built-in request cancellation	lightweight as it is native to Vue.js and has no library overhead, flexible in regards to handling files of different formats
cons	increases project size- but size shouldn't be a problem because the database isn't large, requires installation- but trivial	Need to automatically parse responses into JSON (crucial because it is required to convert the stream of data into usable objects, doesn't automatically enable CORS (need to install cors and obtain CSRF token).

*CSRF (cross-site request forgery) happens when a user attacks a server and changes its state which could prompt other end users to submit private or personal information that can then be looked at by the attacker.

BACKEND

SAR API

When a user makes a request to the server, it is first received by the endpoint in web.php. The SAR api requires the prefix '/api'. If the route exists, the code in web.php reroutes the user's http request to either the SearchController or RecordController. The SearchController deals with routes '/xquery', '/search', and '/getRawEntry'. RecordController handles the '/getRecords' route.

SearchController

/search

Php file parses url input from request, and we immediately convert the json key names to the names python uses for its corresponding attributes. Then the permitted parameters get bundled into a command where a python script is called on these parameters in the form of

a shell_exec function. The returned data is a JSON response object where the entries in the response are updated to have html highlighting and the language abbreviations are mapped to their full spelling.

/getRawEntry
Returns a single entry as a JSON object. It uses a passed in docId, which it then uses to get the entry data from entries.json, which is the json file compiled from the xml files.

/xquery
Accesses existdb to execute an xquery request on the xml files

RecordController
RecordController contains routes to request records or check if a record exists. When a request for a set of records is made, the server tries to fetch an on-server-load preloaded array of data from cache. If the volume isn't in cache, it will fetch the records with the function loadRecords. In this function, a python script called browse_search.py runs and returns all the records for one volume.

Reducing load times
'preload:records' is a custom command that is ran once every time the server starts up. It is executed at boot in AppServiceProvider.php. It preloads the return data for records for each volume and stores it in server cache. Preload time is around 2min and runs in background, loading 1 volume at a time, and it won't affect the speed of any http requests while it runs. As soon as one volume preloads, that volume can be fetched by cache. For example, say volume 1 preload is complete while volume 2 is still preloading. In this instant, the user can get volume 1 from cache, but trying to get volume 2 will result in actually loading for the user.

JSON Generator
Created to convert xml files into single json for easy parsing and better security. Is currently used for search and browse, allows direct indexing of an entry by its docId, instead of searching through the xml every time

Includes the xml path in each entry so the xml string can be accessed quickly. If the path of the xml files changes or is put in a database, this path can be easily modified or removed to adapt to the new storage method.

A substantial amount of information contained in the XML files does not serve any purpose in this application. Looking over these files to extract matches for a user query means lots of irrelevant information will be looked at. In combination with the additional tags wrapping the content, the search functionality performs suboptimally if we only utilise the XML files. Because an efficient search is highly

important, we decided to extract the relevant entry data from all the XML files and convert them into a dictionary that we stored in a single JSON file called entries.json.

There are many differences when parsing and indexing a JSON file compared to an XML file. Although XML files have more support in terms of schemas and namespaces, they require strict formatting. JSON formatting is independent of any programming language since it is tagless, so it is a good choice for future-proofing the application. It is also more readable and easier to debug than XML. The main reason for using a JSON file to store entry data is to simplify the development of search functions. Searching for content from a specific tag within the XML requires complex function calls that require identifying the correct namespace for the tag. Some of the tags use the TEI namespace while others use XML namespace. Meanwhile, searching for content within the JSON file only requires you to know the entry ID which is used to index the corresponding entry data object. Each entry data object is made up of key-value pairs of tags and their content. For example, if you want to get the entry's XML content, you can index the XML file path stored as a key-value pair in the entry data object. This is much faster than looping through all the XML files to search for the correct one.

The data stored in the entries JSON file can be easily updated through the JSONGenerator.py program where you can specify the directory of both the XML files and the JSON file.

Our Search Service

Data Models

Created models for basic search search methods and advanced search. Purpose of models is to serialize the user's input parameters. This means a parameter like volumes with an array string type can be converted to an array of integers, string dates can be converted into integer tuples, and the typed query itself can be normalized according to the search method (stripping white space, or combining characters with spaces into one word with no spaces). Before a search is made, the query parameters must be passed through a model and then the search uses the values produced by the model.

How To Execute Python Search?

We are using standalone python scripts with no framework or application layer. Although this means the script must be executed via shell command, it is very easy to use and can be integrated into other applications with no strings attached. As long as the target file can be located within the directory, you will be able to use basic or advanced search.

To run the python script inside a php function, you have to build the command as if you were writing to the console. For our command, Search.py takes in 2 arguments besides 'python3':

1. an exact file path, $python_search_file, which points to the python file, and a parameters object.

2. The parameters object, which we have called $permitted, is the object returned from converting the search parameters from request into python name versions.

3. $permitted is encoded into a shell argument using escapeshellarg(json_encode($permitted), and is stored in $permitted_params.

4. The command comes together to form a string $command = g 'python3 $filepath $permitted_params'

5. Now we can actually execute the command using shell_exec($command), store the result in $raw_output, and then decode the result by wrapping it in json_decode($raw_output, true)

6. Finally, the json readable results of our command can be indexed using $output["results"] where $output = json_decode($raw_output, true)

7. If results aren't as expected, use php Logs or echo command to debug possible errors.

Basic Search

Fuzzy Search: SequenceMatcher vs Python Levenshtein vs RapidFuzz

Finding matches involves getting substrings within the entry text where the spelling similarity of the substring complies with the user's specified variance. Fuzzy search is a technique used to compare strings and calculate their similarity. In order to select the most suitable method for implementing fuzzy search for the application's needs, we compared 3 possible contenders: SequenceMatcher, Levenshtein, and RapidFuzz.

- **SequenceMatcher** is a built-in Python library that handles general string matching including Fuzzy Search. However, its speed plummets with larger datasets, making it slightly inefficient for the client's medium-sized corpus.

- **Python Levenshtein** is an external library which is used for precise control over string operations such as substitutions, insertions, and deletions. It is much faster than SequenceMatcher and doesn't compromise speed as the dataset grows. The precise control is also more suitable for the tasks required.

- **RapidFuzz** is also an external library that has more advanced fuzzy matching features than Levenshtein but still contains functions for precise control. Both Levenshtein and RapidFuzz perform edit distance and similarity ratio calculations in large datasets incredibly quickly. However, RapidFuzz focuses slightly more on optimization and has been proven to outperform most other fuzzy matching tools in large corpus comparisons while still being capable of precise control over string operations. With its speed and scalability, RapidFuzz is undoubtedly the best option out of the three, so we chose to implement fuzzy search with RapidFuzz.

Parsing User Query

Finding matches

When a user searches, the user input content is passed through a search controller. For the first part of the search, a Python program looks through the text content of each entry in the JSON file. When looking for a match, it looks for text that either matches the search input exactly or is similar to an extent.

Adjustable variance per search method

Variance extent ranges from low variance to high variance. This variance is quantified according to the number or percentage of differing characters to the query. As another perk to control over our own search functions, we are able to determine exactly which values correspond with a certain variance. The implementation is different depending on what search method is selected. For regex, variance is completely ignored. For most methods, the rapid fuzz fuzzy functions produce desirable values. For phrases, we consider the number of words combined with the fuzzy values to determine the final variance value of a phrase.

Variance/Similarity conversion

Variance has values 0-5, but rapidfuzz uses a score out of 100, so we created a method to convert the desired variance value to a corresponding rapidfuzz percentage. The converted variance value can be calculated as the absolute value of multiplying the variance by 10 and subtracting 100 abs(variance*10-100).

Word Start, Middle, End

Word Start, Word Middle, and Word End are 3 of the main search methods. They use fuzz.ratio to calculate the variance score for each matching word in the entry text.

Regex

An entry with a regex pattern applied to it returns non overlapping sections of the text where the pattern finds an exact match.

Keywords

The query consists of multiple words that are each scored separately and the overall score for the entry text is the value of the highest scoring word.

Phrase

From slow to phast!
Optimizing search speed + preventing overlapping of results
Character vs word comparison: switching from comparing number of characters to comparing number of words improved performance. Less 'things' to count
String building vs indexing: at first we stored an array of words, along with a string of the concatenated words. Each time we appended or removed a word from the phrase, we reconcatenated the words for phrase comparison, even if we weren't doing string comparisons on the phrase, which is not the most efficient. Our most recent version of phrase searching involves iterating over the text using pointers to keep track of the part of the phrase we're checking, and only dealing with strings once we actually need to do string comparisons.

Eliminating useless comparisons.
To keep the comparison relevant, we only calculate scores for phrases where the number of words matched the number of words in the query, or is within the allowed range according to variance. For example, if the user selects 0 variance, we skip over any sequence of words that differs from the number of words in the user query. However, if the variance is > 0, we use a word margin that allows phrases with of +/- number of words than the query to be compared. We use while loops to grow and shrink the size of the window, sort of like a horizontal slinky movement, or a worm inching its way across the sidewalk.

How do we prevent overlapping results?
Because we're using a window stepping technique, this means we're often checking phrases where parts of the phrase are in phrases we recently compared. The program ensures that the best matches are prioritized by keeping track of the current best scoring phrase and only confirming it as a final match if no subsequent overlapping phrase has a higher score than it. For example, let's say the query is **'milk chocolate'**, text is **'i love milk chocolate milk delicious yum'**, and the variance is 3:

1. **'I love'** is checked first. The fuzzy score does not meet the variance threshold. There is not a previous best phrase to confirm either, so the program moves on.

2. **'love milk'** is checked next since, and the fuzzy score is above the variance threshold, so the best score and start/end indexes is set to **'love milk'**

3. Next **'milk chocolate'** will be checked. After its score is calculated, the program checks if the indexes overlap with the index of the previous successful score. It does! Furthermore, it has a better score than **'love milk'** as it matches exactly with the query. So, it changes the best score and start/end indexes to **'milk chocolate'**

4. Next **'chocolate milk'** is checked. After its score is calculated, the program checks if the indexes overlap with the index of the previous successful score. It does! However, it doesn't have a better score than the index of '**milk chocolate**'. So, nothing is saved, and the program moves on

5. Next **'milk delicious'** is checked. After its score is calculated, the program checks if the indexes overlap with the index of the previous successful score. It doesn't. So the program looks at the most recent best phrase, which is '**milk chocolate**', and stores the indexes of this phrase in the array of final results. Since **'milk delicious'** passes the variance threshold and does not overlap with **'milk chocolate',** its indexes can be set as the new best phrase.

6. Next, '**delicious yum**' overlaps and doesn't meet variance threshold

7. End of string, so **'milk delicious'**, the last best phrase, is confirmed as a result in final results.

8. Final results for **'milk chocolate'** allowing variance 3 = [**'milk chocolate'**, **'milk delicious'**].

Advanced Search

Advanced search does not use any string matching libraries like that of basic search. It is mainly used to filter out specific entries such as removing entries that are outside of a specified date range. This search object takes in or pulls from json_entries, and also establishes the following optional parameters: language, pages, volumes, entry_id, date_from, date_to. The data might not be formatted correctly, so it is passed through a data model called AdvancedSearchModel, where the return value of this model provides the parsed values of the original parameters. The parsed values are then used to filter out entries. Any parameter that hasn't been specified is simply skipped over during the filtering process.

We have created a separate file to store our advanced search methods, which allows us to keep our logic neat and organized. Advanced_search_methods.py contains methods to determine if an entry date is before or after a specified date. Now since date is an object with keys such as 'when', 'from', 'to', and 'cert', we created a separate static method to specifically where we can pass in the string value in the form of 'YYYY-MM-DD' of a key value pair of a date object, and receive an integer value of that date. The function uses bitwise operations to calculate values for year, month, and day and add them together.

Order By:

By default, sort_best_matches is applied to a basic or advanced search which sorts the results by their accuracy score first and match frequency second. Each sort method in sort_methods.py evaluates the json object input, and applies criteria to rank the results differently. They all require a json object as input and most functions also take in a matched

object. The sorting criteria can come from the match results object and/or the json entries. While most of the functions sort the matches, browse search currently uses a function which sorts on the json_entries only. If sorting matches with json_entries criteria, matches.items() = matches $\subseteq$ json_entries, where the linking key is their docId.

ExistDB

ExistDB is a highly performant and feature-packed XML database engine which is optimised for and allows the usage of XQuery. XQuery is the querying language of XML files. The design of ExistDB concerning XML is a direct parallel of SQLite to SQL.

Reasons for selecting BaseX:

Although the previous team Charlie had concluded BaseX to be their best option to work with in terms of both functionality and performance, we encountered some difficulties incorporating this system with the important feature set of both docker and Laravel PHP, the latter of which was a mandatory framework - and as such we went for a more suitable option.

It allows for those familiar with the last project to be able to immediately start working rather than learn an entirely new system.

If required, advanced GUI available on the localhost with its immense feature set (i.e., file uploading, user management etc.) GUI can be provided for maintainers for purposes such as debugging and/or extending the functionality of the project.

Other database/XML parser considerations:

- lxml is a very updated and well-documented Python library implementation, however, this was limited to XPath Queries and thus didn't offer the feature set that is desired (FLOWR expressions).
- PostgreSQL XML Database supported XML data type and accompanying functions, XML export format, and mapping XML to SQL databases however it too was limited to SQL, JSONPath and PL/pgSQL querying.
- Gate/Gate Mimir isn't primarily focused on delivery through XML so their query language is based on Annotation Query Language (AQL) and/or Ontologies which don't make it suitable for this project's implementation.

Further Considerations for XQuery Execution:

Given the advanced nature of XQuery, its usage is anticipated to be infrequent. Therefore, a potential optimisation could involve implementing an internal timeout mechanism. This mechanism would allow the server to remain active for a predefined duration (e.g., 12 hours) after an XQuery is executed. Once the timeout expires, the server automatically shuts

down to conserve system resources. While this approach minimises resource consumption over time, it introduces a trade-off: the initial query (and thus the user) following a restart would experience a delayed execution due to the server initialisation overhead.

TESTING

Automated Testing
We have implemented automated testing through GitHub Actions to ensure code quality and prevent features from breaking. The workflow file, main.yml, automatically runs all of the front-end test files whenever there's a push or merge request to the main branch. The workflow runs in Ubuntu Linux every time, providing a consistent testing environment which means that errors that avoid detection in local setups will be caught.

This automation serves as a quality barrier, preventing code that fails the tests from being merged into the main branch. By integrating testing directly into our development workflow, we ensure that all code changes are validated against our test files before they become part of the main branch.

Front End Tests
The implementation of comprehensive testing throughout our project has been essential to ensure system functionality, maintain a good user experience, and validate that components work together as expected. We have focused primarily on testing front-end components using the Vue Test Utils library and Vitest as our testing framework, which allows us to effectively evaluate the functionality of our system and identify potential issues.

A key aspect of front-end testing for our project was testing the functionality of the search operation. The goal of testing the search operation is to ensure that the input from the end user corresponds with the expected output. Furthermore, our tests allow us to compare our implementation to various requirements and specifications. Our testing strategy centred on verifying that individual components function correctly, allowing them to seamlessly integrate with other parts of the application. This approach helped us maintain code quality, evaluate the functionality of the system and discover bugs early in the development process.

Router Tests
We implemented tests to ensure navigation between pages works correctly throughout the application:

- Home/About Page Routes: Tests were implemented to ensure that the home and about pages were returning successfully. These tests ensure that the pages are loading as intended and the expected content is being displayed.

- Search Page Route: After using a 'router push' request for the '/search' page with query parameters, this test similarly verifies that the correct route is being received. This test also confirms that search queries are being properly managed, which is essential for the search page where query parameters must maintain integrity.

Search Functionality Testing

Given that search functionality is central to our application, we've implemented extensive tests for our search components:

- Search Fields: We thoroughly tested the advanced search fields to ensure proper form handling. Our tests verify that default search values are applied when the website loads, as well as confirming that user input updates the search appropriately. We also tested the advanced search dropdown elements extensively, such as the volume selection, confirming that it toggles individual volumes as well as 'all volumes'.

- Search Results: We implemented tests that used a mock version of axios to simulate API requests without actually calling the server, allowing us to validate that results are displayed correctly based on various query parameters. We also tested the search results' page changing functionality, with tests confirming that next and previous buttons work as expected as well as tests that ensure the final page of results can properly handle having fewer entries than a full page.

- Search Result Card: We tested the individual result cards that make up the search results, verifying that search result data is displayed correctly and that interactive features such as the copy button work as intended. Our tests confirm that key information such as volume number, date, and language are all properly displayed.

Back End Tests

Search Unit Testing

We created several python unit tests to ensure our python search methods work as expected. First we tested smaller functions from each search class with a variety of input to ensure our code is robust. The sar entry texts often include special characters, so it’s important that we make sure the tests work for special characters as well. Some of the search classes share similar functionality although slightly different, so they deal with the same input differently. For variance scores, tested less than, greater than, and equal to so that we know the score values are as expected. Phrase search required a lot of tests especially as it is a more complex searching function that has several functionalities layered on each other. We tested that the phrase overlapping function did not accept any non overlapping indexes within texts, and that calculating the margin with ceil returned the correct word margin according to the word length of each phrase. Finally, we tested the parent functionalities once all of the individual function tests passed. These were successful and confirmed that our search methods do indeed produce the results that we want according to different parameters and variance requirements for each search method.

Accessibility Testing

Figure 24: Lighthouse audit results (May 10, 2025)

While a perfect score in the Lighthouse accessibility audit is a good reflection of the SAR tool's usability and Team Lima's commitment to inclusive design, we also reviewed 10 additional manual checks recommended by Lighthouse to validate areas that may not be tested automatically. These areas include whether a page has a logical tab order and if the visual order on the page follows the Document Object Model (DOM) order. These manual checks were also cross-referenced during the NVDA screen reader compatibility testing process.

Minor issues were identified under the other audit categories (performance, best practices, and SEO). These include improperly sized images affecting load time, unused JavaScript, deprecated APIs, and a missing meta description. While these issues do not affect the accessibility of the SAR application, they have nonetheless been noted for future development.

NVDA

To test the SAR tool's screen reader compatibility, we conducted manual testing using the NVDA application, one of the most widely used open-source screen readers available free of charge. This confirmed that users with visual impairments, cognitive disabilities, and learning difficulties can fully navigate and interact with all aspects of the SAR tool. Key findings include:

- All headings, links, and buttons were announced correctly by the NVDA screen reader
- Form and select elements have appropriate associated labels and are accessible via keyboard tabbing.
- The structure of the content is logical and ensures clear navigation.

Where needed, ARIA roles and semantic HTML5 elements, like <main>, <header>, <footer>, or <label>, were implemented to improve the SAR tool's compatibility with assistive technologies. Importantly, the NVDA screen reader was able to read out each record entry in full, no matter what the language is. This approach to testing aligns with the BCS public interest principle, where they state that developers must "conduct your professional activities without discrimination" and "promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise" (BCS, 2022).

A/B Testing

To evaluate the usability improvements made in the SAR 2025 interface, Team Lima conducted an A/B test comparing it to the original SAR tool developed in 2017. The aim was to measure user satisfaction and determine whether the redesigned interface provided a significantly better user experience

A total of 30 participants took part in the study, split evenly between the two versions of the website. Participants represented both our primary audience (students) and our secondary audience (members of the general public). Each user completed equivalent tasks on their assigned version of the tool and responded to 10 Likert-scale questions (scored from 1 to 5) evaluating usability and overall satisfaction. The results and findings of the test are summarised below:

SAR 2017:

- 26.7% of users that evaluated SAR 2017 identified as members of the public, and 73.3% identified as students.
- The users had a range of experience using archive search tools, with 20% frequently using them, 20% occasionally using them, 33.3% rarely using them, and 26.7% never using them.
- Users expressed strong dissatisfaction with the navigation and overall layout of the original SAR tool developed in 2017. Feedback we received from users included:

"Scrolling made the screen glitch, and it was difficult to navigate and understand the browse section in particular."

"Having to scroll down to a particular page rather than being able to search for it was tedious."

"I hate the navigation."

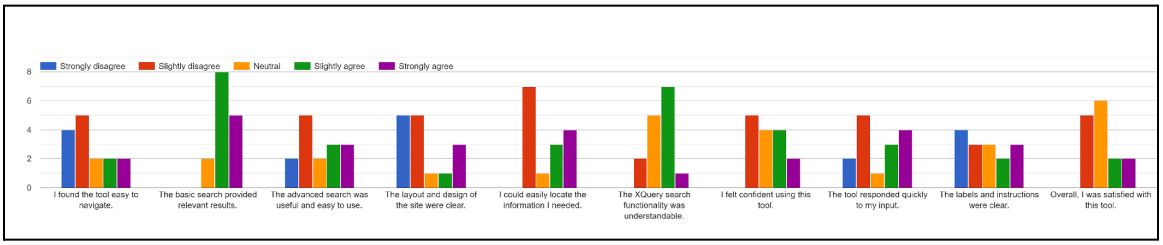


Figure 25: SAR 2017 A/B Likert question results

SAR 2025:

- 53.3% of users who evaluated SAR 2025 identified as members of the public, and 46.7% identified as students.

- More users who evaluated this version of the tool had no experience using archive search tools, with 13.3% frequently using them, 13.3% occasionally using them, 33.3% rarely using them, and 40% never using them.
- Some users left positive feedback and expressed a preference for the available dark mode, while others wanted some extra information. Feedback we received from users included:

"I appreciated the light and dark mode as I can switch between them depending on my environment."

"I think the only thing is more info on the XQuery page would be nice."

"Advanced search was a little bit confusing as I wasn't sure if it took effect immediately on the results already there, or whether I had to click search again to get it to complete the advanced search."

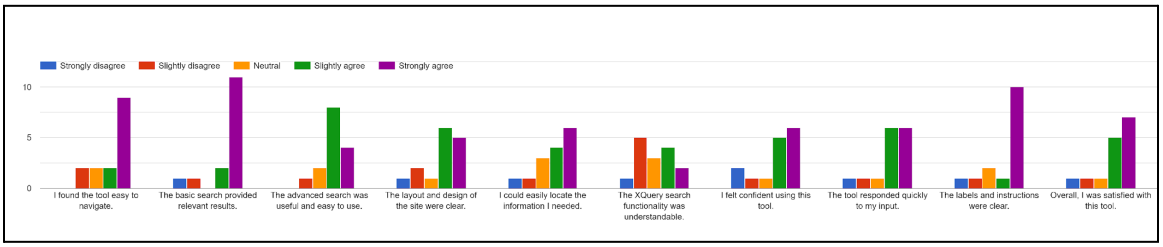


Figure 26: SAR 2025 A/B Likert question results

Analysis:

Following the completion of the A/B test, the results were analysed. SAR 2017 had an average total score per question of 32.93 (Standard Deviation ≈ 9.45), while SAR 2025 had an average score per question of 41.93 (Standard Deviation ≈ 7.93). This shows that users consistently rated SAR 2025 more positively. An independent t-test was calculated to find out whether the difference in scores was statistically significant. The result was 1 = -2.59, p ≈ 0.015, showing a statistically significant difference in favour of SAR 2025. To measure the importance of this difference, Cohen's d was calculated at 1.01, representing a large effect size. This shows that higher user ratings for SAR 2025 are both statistically and practically meaningful.

Additionally, the fact that users rated SAR 2025 higher, despite having less experience on average than users who tested SAR 2017, suggests that the usability of SAR 2025 is greater than that of SAR 2017. User feedback and comments have also been used to inform future work and improvements that can be made.

In conclusion, these A/B testing results prove that SAR 2025 provides significantly improved user experience compared to the original 2017 version.

Limitations

One limitation is that of the implementation of XQuery. Ideally the same database the previous team used would be utilised, however the issue with attempting to make BaseXClient work was too difficult for us to resolve in the time we had in the previous semester, so we opted to change to ExistDB instead. This allowed us to then locally implement full XQuery in FLOWR format as intended but as further development occurred and we extended the project to be deployed whilst also using docker, this further complicated things. One reason ExistDB was the latter choice was due to its less extensive documentation and use - this was one of the obstacles with docker as the documentation seems sparse and such a use-case isn't well discussed meaning that as first time users and developers of both docker and an XQuery database a situation like this was hard to navigate. As a result the official ExistDB docker image wasn't made to work and an alternative solution was developed where an installation of ExistDB with the xml files already pre-loaded would be copied into the root directory of the repository and then this would be able to make the suitable docker container to be interacted with. This works due to the database only needing to be queried and never updated, however this does effectively mean that the repository directory would have extra data unnecessarily inside of it which is against the purpose of docker.

Conclusions and Further Work

Project Outcome

Building on last year, significant progress has been in advancing and completing work that was either unfinished or simply not working.

One of the aspects being the elastic search, while the previous team used a paid query service, elastic search, we took it as an opportunity to implement a custom search algorithm that involved creating a range of modules to not only satisfy the current search requirements but further refine the logic so that results are more precise than elastic search. Development of the new search began last year but has now been completed. Not only does it match the original functionalities of the old search it surpasses it with new and minor customisations, which have strengthened the accuracy of the search.

Last year, we were unsuccessful in properly implementing XQuery, which we suspected was due to the terminal environment spawned through shell escape and the execution of a Python script. However, we recognised that a more elegant and robust solution was needed. This year, we decided to switch to eXist-db, as it is the only major native XML database that offers built-in support for XQuery execution, RESTful APIs, and seamless XML data handling, making it a much more suitable choice for our needs.

Future Changes

Keywords

Improve results of keyword search. Currently the algorithm assigns an overall score based on the match with the highest score, In keywords specifically, this means going with the highest scoring word in the entry. Further examination between the new and current SAR has sparked a desire for the functionality of the new SAR to more closely align with the current one. We plan to enhance the current score accuracy for keywords by averaging the matching word scores in each entry instead of returning the score of a single highest scoring word.

Fixing highlights

Highlighting currently being done in the backend. Eats up performance and doesn’t highlight everything it's supposed to. Backend adds html tags to data in response, increasing size of data object being transmitted, causing limitations ex. Searching ‘a’ does not give results because the return object is too big. Plan is to move highlights to front end which will reduce

results load time significantly and also it just makes sense to do formatting in front end in general as its only useful for the client side

Improving SPAness of Frontend
Get rid of reload on search
Cache search results using space that would've been used for elastic search.

Code Security
Eliminate hardcode of paths currently everywhere in application. Especially front end api requests

Autosuggest
A useful addition discussed was auto suggest, this feature being particularly useful for users who may not necessarily know what they are looking for. This feature would suggest words that they may want to investigate

Conclusion

Team Lima has been honored to work with Dr Jackson Armstrong on this project, helping to ensure that these exceptionally well-kept records – digitised by the LACR team to conserve them for future analysis – remain accessible for researchers and the wider public and continue to offer a glimpse into life in 14th and 15th century Aberdeen. We are excited at the prospect of the project being used by the University and displayed on its website. We are proud of what we have accomplished and built. This project has been a valuable learning experience and has left us with "real world" experience working with a client and developing a web application using the Agile development methodology. We are all proud of what we have achieved so far and look forward to seeing how our search tool contributes to the ongoing study of the Aberdeen Burgh Registers.

REFERENCES

(Use consistent citation style – APA, IEEE, etc.)

1. National Archives (2008). *National Archives*. [online] Archives.gov. Available at: https://www.archives.gov/.
2. OWASP (2024). *OWASP Top Ten*. [online] Owasp.org. Available at: https://owasp.org/www-project-top-ten/.
3. Text Encoding Initiative (n.d.). *The TEI Guidelines*. [online] tei-c.org. Available at: https://tei-c.org/release/doc/tei-p5-doc/en/html/index.html.
4. The National Archives (n.d.). *The Discovery Service*. [online] discovery.nationalarchives.gov.uk. Available at: https://discovery.nationalarchives.gov.uk/.
5. UNESCO UK. (n.d.). *Memory of the World – UNESCO UK*. [online] Available at: https://unesco.org.uk/portfolio/memory-of-the-world/.
6. W3C (2023). *Web Content Accessibility Guidelines (WCAG) 2.2*. [online] www.w3.org. Available at: https://www.w3.org/TR/WCAG22/.
7. Sethi, R. (2022). *Software Engineering, Basic Principles and Best Practices*. Cambridge University Press.

Appendices

User Manual

https://sar2.andreasmaita.com/help

The above link will direct you to the help page on our application, in which you find guidelines on how to use and navigate all features present in the website.

Maintenance Manual

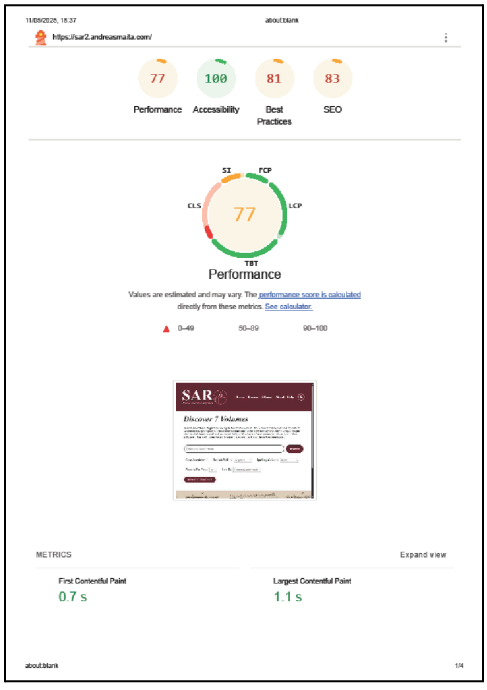
https://github.com/GroupLima/SAR-Repo/blob/main/README.md

The link provided above will direct you to the ReadME page located in our GitHub repository, which will provide an in-depth explanation on maintenance procedures guidelines.

Test Logs

Lighthouse

The full Lighthouse report is shown below. Double-click the object to view the report in full:



Tab 2