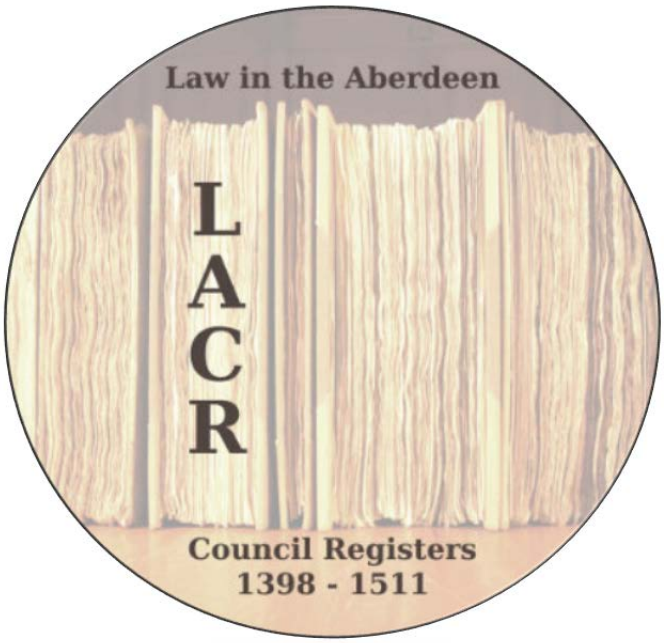


LACR Search

Technical Report

Team Charlie

Table of Contents

Introduction

About LACR Search

LACR Search is a Ruby on Rails powered web application that was developed for our clients: the Aberdeen Registers Project team, and the Aberdeen city archives. Both clients had different ideas of what functionality the application should have: The LACR team wanted a tool that could help them with their research by giving them a more convenient way to analyse documents. The Aberdeen city archives primarily wanted the application to make the transcriptions and images of the documents available to the wider public. It therefore fell to our team to combine these two perspectives into an application that could be used both for serious academic research and, a discovery tool that the people with limited knowledge of the source material. To achieve this we decided to create an application based around three different search tools, each with an increasing level of complexity. With increase in complexity comes an increase of functionality, and thus more concise and relevant search results. The first of these search tools is “simple search” this tool is aimed at users with the least knowledge. Users provide a string which will then be searched within the content of the documents. The second search tool is advanced search. Itis designed for user with a medium level knowledge. Along with the functionalities of “simple search” the advanced search gives the user access to a number of other fields that they can use to further the filter the results. These fields include - but are not limited to - date range, volume and page. The most precise search option is XQuery. It allows the user to create queries which can extract data that would not be possible with simple or advanced search. Around the applications search features, we also implemented other features, such as the ability to browse through all of the documents, and the ability for authorised users to upload and delete documents.

The LACR Team And Their Project

"Aberdeen has one of the richest surviving series of municipal registers in all of Europe. The first eight volumes are exceptional in a Scottish context – more survives for Aberdeen for the fifteenth century than for all other Scottish towns combined. The registers are in a near continuous sequence from 1398 to the present day. In 2013 the earliest eight volumes, covering 1398 – 1511, were inscribed in the UNESCO UK Memory of the World Register for their outstanding historical significance. They take the form of manuscript books, which record elections, the admission of burgesses, leases, correspondence, civic ordinances, but the bulk of the content of the council registers concerns court proceedings, dealing with disputes and the administration of justice. Thus they are a valuable record of law, and the evolving language of law, as they are written in Latin and Middle Scots. The Law in the Aberdeen Council Registers project is funded by the Leverhulme Trust and housed in the Research Institute of Irish and Scottish Studies (RIISS) at the University of Aberdeen. It is based on a collaboration between RIISS and the Aberdeen City & Aberdeenshire Archives. The project runs from 2016-2019 and is creating a digital text from the UNESCO-designated council register volumes."

Dr JACKSON ARMSTRONG
Principal Investigator and Project Director

LACR Search has been developed in part to complement a larger project being conducted by the LACR team. The first eight volumes of the Aberdeen council registers, covering the period 1398-1511, are being transcribed and annotated. The Aberdeen council registers are primarily legal records of court proceedings, ordinances, elections, dispositions of property and correspondence. The surviving registers from 1398 to 1511 miss only one volume (c.1415–c.1433). They are hand-written; mostly in Latin and Middle Scots. The project reaches to the end of volume 8 (covering 1501–1511). The team aims to produce a digital textual resource from the Aberdeen council registers in the format of full, accurate, versatile and online text encoding initiative (TEI) compliant transcription.

The tools LACR team currently have access do not fulfil their needs, slowing down research and the transcription process. The team expressed the need for a substantial piece of software tailor made for digital humanities research. This sentiment is echoed throughout the digital humanities community. We therefore concluded that this was a viable project to take up. While researching and transcribing the documents it is important that the project team can easily navigate and search within the transcribed documents. These services will be provided by our web tool. LACR Search allows the project team to upload transcripted XML documents alongside the corresponding scanned image. Once documents have been uploaded they can be quickly browsed or searched through, by using either a simple search, or a more refined method through advanced search options. That allows the user to specify date ranges, document ID's and proper names.

Advanced users also have the option to search through the XML tags of the uploaded documents using an additional search bar for XQuery.

About Digital Humanities

The LACR project is part of a quickly growing, ever more important field known as digital humanities. Contemporary Digital Humanities stands not in opposition to the past, but on its shoulders. It honours the pioneering labours carried out over the past seven decades in the form of statistical processing (computational linguistics), linking (hypertext), modelling (architectural and visual displays), and the creation of structured data (XML).[19]
Digital humanities initial focus is to preserve documents and information from the past in a digital form, granting it immortality and making it available to a much wider audience than the original author of the documents could ever imagine. The second focus of digital humanities revolves around everything from word frequency studies and textual analysis (classification systems, markup, encoding) to hypertext editing and textual database construction, contemporary Digital Humanities marks a move beyond a privileging of the textual, emphasizing graphical methods of knowledge production and organization, design as an integral component of research, transmedia crisscrossing, and an expanded concept of the sensorium of humanistic knowledge.

Team Strategy

Our team consists of 5 members: The team leader Radostin Stoyanov, deputy team leader Marcel Zak and the rest of our development team Cameron Beck, Jack Burn, and Jan Siemaszko. Being team leader Radostin took on the responsibility for organising team meetings, providing communication between our client and development team, organising meetings and demonstrations with the client and assigning tasks to team members. In the case Radostin was unable to uphold his responsibilities our deputy team leader Marcel was assigned with taking them. Work was distributed evenly throughout the team on a week by week basis. It was decided early on that each member of the team should take a specialised role, with each team member concentrating on a different aspect of the project. This allowed us to be able to work on different parts of the application in parallel, reducing idle time of individual members. Communication between team members was encouraged, so each member had a clear idea of the progress of development and what was needed to be done or improved.

Team Meetings

During the project our team had two weekly meetings scheduled at a time all members were available. Our meeting on Mondays consisted of all 5 team members in which we have

discussed our progress with tasks given the previous week and showed other team members progress we have made. Our second weekly meeting occurring on Fridays consisted of all team members and also our project guide, Dr Adam Zachary-Wyner. In these meetings we demonstrated to Dr. Wyner the progress we have made and discussed any future plans. Dr. Wyner being a part of the LACR Project team was in close contact with our client. Taking advantage of this, Dr. Wyner also proposed meetings with the LACR team and us. The second part of these meetings is very similar to our meeting earlier that week. However, this is the time in which our team leader Radostin will allocate tasks to each team member ensuring that everyone is given a fair workload and are happy with their tasks.

Meetings with the clients

We chose to follow the agile development method. Frequent communication with our client was significant in guiding our decisions throughout the design and implementation phases of the project. During the development phase of the project Radostin organised two major formal meetings with the LACR team, on which all members of the team were present. In the meetings we demonstrated the application, both gave and received questions, and feedback about the current features. Functional requirements were routinely scrutinised. As the weeks passed a number of the requirements were revised to meet the ever changing vision of the project in accordance with the client's needs. Although some of the changes led to developmental setbacks, they led us to create a more refined final product.
Towards the end of the project numerous other smaller drop in meetings were scheduled at short notice in order for us to get quick feedback about certain aspects of the project.

Team Management Software

Our team utilised several tools to aid in our development:

- GitHub: Web-based version control repository and Internet hosting service. It offers all of the distributed version control and source code management. And it is widely used tool in the industry.
- Trello: A collaboration tool that helps organise tasks, deadlines, communication with team members and progress of the work. It is easy to track who is working on what and where something is in a process.
- Gitter: An instant messaging service that was used to provide communication between team members. It is also connected to GitHub and Trello. Therefore, we have got immediate notification when tasks were assigned or when an issue with the code arose.

Competition

The majority of software in our sector that is, web interfaces to search through large historical corpora comprises mainly of in-house software, individual software packages that are tied to their particular project. They are also mainly offered as an offline solutions, and often only consist of very basic search functionality. Our web-tool is designed for much wider range of users from children and family historians to academics and researchers. The difference between these applications and our web-tool, is that our tool is not tied to a particular project and it needs only minimal modification to work in a relevant situation. It is standards-compliant (specifically, it uses documents stored in the HisTEI format), and therefore (given your documents are encoded in the correct format) it will work with them. Another important difference is the search functionality provided. LACR Search allows normal text searches over the whole corpora, as would be allowed with any other search tool, but it also allows advanced searches (such as within a date range), and even XQuery to be run over the body of the documents.

One impressive example of in house developed software is the “Deutsches Textarchiv” -German text archive-.[14] You can search for specific word or basic forms, and the query can be refined by word type, context or metadata filters. There are numerous options to refine search queries using DDC -a corpora and linguistics search tool that can search for words or sequences of words together with morphological patterns- in the corpora of the German text archive. The DDC's own syntax allows very complex searches. This bears a stark similarity to our XQuery search feature. Deutsches Textarchiv deals with a corpora that is TEI compliant, and display the documents in a very similar manner: there is an image of the document on one side of the page, the original text of the document and the transcribed XML of the document on the other. However our application differentiates itself with features such as search suggestions, auto complete and fuzzy search.

Another example of in house software is “Mecmua”.[13] An Austrian based historical text repository of early modern ottoman historical documents. Compared with our application their search capabilities are basic. They only have options for simple string match searches, but do however have filtering options such as: person and places. Mecmua deals with TEI compliant documents which are displayed in a manner similar to ours, but does not include the image of the documents. Mecmua displays the plain text of the document, a stylised version with identical HisTEI standard annotations to our application and the transcribed XML of the document.

Finally, another application which offers features similar to ours is: OpenSoNaR.[17] It is an online system that allows analysing and searching the large scale Dutch reference corpus SoNaR. Due to the size of the corpus (500 million words), OpenSoNaR aims to provide a user-friendly online interface to its data. Their search functionality could be considered akin to ours. For instance, they have an “expert search” that allows the user to write their own queries. The difference is that the queries are written in Corpus Query Language (CQL) instead of XQuery.

CQL is a full-featured query language introduced by the Corpus WorkBench and also supported by the Lexicom Sketch Engine.[27] It is a standard and powerful way of searching through the corpora for complex grammatical or lexical patterns or to use search criteria which cannot be set using the standard user interface. However, XQuery is much more powerful as it is general query language for XML annotated data. Although, it was originally designed to query XML databases, XQuery is the main query language used within the TEI community. [28]
OpenSoNaR also have an “extended search” which works like our advanced search. Users can specify data in a number of different fields to further filter the search results. They also have a simple search that acts as a basic string match search. This also mirrors features of our application. The OpenSoNaR project does not display any of the related images with documents. Our application also features other functionalities which we consider to vastly enhance the user experience. Be it autocomplete, suggestions and document browsing functionality.

Other than in-house developed applications there are only few products which aim to give users similar functionality as our application. These, however are mostly general search tools which do not take into consideration the content of the corpus, or the specific needs someone wanting to search through these corpora.

One such piece of software which could be seen as competition is Gate Mímir. Mímir is a multi-paradigm information management index and repository. Mímir has a web interface which can be customised to provide user-friendly interface [30] with search capabilities mainly focused on search for plain-text, annotations and semantics. However, it lacks fuzzy search, suggestions and autocomplete. Similar to our XQuery interface, the default Mímir web application [29] require the user to enter plain-text queries in order to use the search functionality. Therefore we believe our application is considerably more user friendly, allowing for users of all skill levels to carry out searches. [11]

BaseX is another piece of software that is similar to ours. BaseX is native XML database with an intuitive GUI which has several options for data visualisation. These data visualisation options are only available through a desktop application, so end users have to install and configure BaseX to be able to use these features. It allows users to search using a formal query languages and has an autocomplete function. However, BaseX’s search capabilities still fall short in a couple of areas when compared to our application. For instance, it does not support functionality similar to our “simple search”. This is full-text search, fuzzy search or suggestions in case of spelling error. [2]

Requirements

The requirements were devised early in the initial stages of the project, and were constantly revised throughout the duration of development to meet the LACR team's needs. The requirements are ranked in order of importance (highest to lowest).

Functional Requirements

1. The application should parse all XML tags required by the TEI standards, as well as those required by the namespace

   This requirement has been fulfilled, all tags are parsed server side before the documents are added to the database.

.

2. User should be able to search through a transcribed document
   a. User should be able to search for plain-text string

   Fulfilled in the simple search. The ability to search through the corpus via plain text is provided by Elasticsearch, and thus the entire body of the corpus may be searched through via plain-text key. This is also possible in the advanced search, since this also uses Elasticsearch.

   b. User should be able to search through documents by ID (volume, date, page)

   As with the plain-text search, Elasticsearch also provides the ability to search via document ID, volume, date, and page. However, unlike the previous requirement, we do not allow this in the simple search, if only because that would negate the point of the simple search.

   We also extended this requirement so that the user can search by paragraph and language in the advanced search. They may also modify the level of spelling variants allowed in the search - that is to say, how closely the spelling of the results must match the spelling of the queries.

   c. User should be able to search through documents by proper names

   This requirement has been fulfilled inasmuch as it is possible to do so. In the future, the intention of the LACR project is to annotate the documents with more tags such as annotations for proper names, however, at the current time, this has not yet been done. Therefore, the ability to search for a proper name will be possible via the XQuery search when the tags are added to the document. The ability to do so already exists in our application, even if none of the documents currently have the necessary data provided. This requirement has simply been subsumed by the next requirement.

   d. User should be able to search through documents by annotation

   This requirement has been fulfilled by the XQuery search. Although a complex tool, it allows very advanced searching of the XML documents by their structure, and specifically it allows searching for annotations (perhaps with a given context). We have

tried to make this search quite simple, even though XQuery is an innately complex tool.

3. User should see the search results and the ten words of context surrounding the result.

   We did not fulfil this requirement precisely - instead of providing ten words of context around the result, we provide thirty characters of context. This is because of the way in which the design of the Elasticsearch interface works, as it currently is, we just truncate the result from Elasticsearch down to thirty characters.

4. User should see the document ID in the search results.

   This requirement has been fulfilled. The ID of the document is displayed at the bottom of each search result, and is also displayed in the header of every paragraph in the paragraph display. The purpose of this display is so that paragraphs in the corpus may be easily referenced.

5. User should see the date of the document in the search results.

   This requirement has been fulfilled. The date is displayed to the right of the volume and page numbers in every search result. It is also displayed in the header of every paragraph in the paragraph display.

6. User should be able to view a list of documents in chronological order.

   This requirement has been fulfilled in the search results, under the order by option. By default, search results are ordered by relevance; however, this can be changed to chronological. We have also added the option to sort by volume and page ascending and descending order.
   When browsing all documents, they are ordered by volume and then page, in ascending order. This is by request of the LACR team.

7. The application should store the documents with a SHA-256 checksum.

   This requirement was deemed to be unnecessary during development, therefore the requirement was not fulfilled. Our reasoning was that the documents will not only be stored on our server.

8. The application should periodically check the stored documents against their checksum so that data can be protected against data decay.

   This requirement was deemed to be unnecessary during development, therefore the requirement was not fulfilled. This is because requirement eight requires seven to be

fulfilled, and it is not (see above for reasoning).

9. New data (XML files, translations, interpretations and scanned pages) should be imported. (via web interface, drag and drop function, filter out directories, only files which are supported)

    We partially fulfilled this requirement. Instead of allowing users to upload documents via a drag-and-drop interface, the client required to use the system's native file picker. Hopefully, this should be less confusing for the user. The file-picker allows the user to select multiple documents, but only xml files for the transcriptions and tiff files for page scans.

10. User should be able to download the XML file of desired document.

    We fulfilled this requirement. When looking at a specific search result, or browsing by page, the user may select an arbitrary set of pages and click on the "download" button to download the xml files containing the relevant pages. The pages are downloaded in a zip file.

11. User should be able to download the original scan of desired document.

    We partially fulfilled this requirement. Due to copyright concerns from the LACR team, it was decided that we should not make the original scans available for download. Having said that, the images would be available for download as a lesser-quality jpeg files with a watermark.
    The user may do this in a similar way to the previous requirement. They select some pages and click the "download" button. Then select the option to download the transcriptions with images. The xml files and images corresponding to these pages are downloaded in a zip file.

12. The application should be able to interface with the CLARIN project.

    This requirement could not be completed for a number of reasons; firstly, CLARIN requires a subscription that costs £1000 per year, (and the university is not currently subscribed). Secondly, the UK CLARIN representative was unresponsive and other Contacts involved in the CLARIN contact were difficult to come by.

13. User should be able to toggle all annotations on and off.

    This requirement was not fulfilled. Although there is no technical reason that annotations could not have a toggle, if we just showed the plain-text, it would include things such as deletions, and this is not what the LACR team wanted and the client required

changes.

14. User will be provided with documentation.

    This requirement is fulfilled in the user manual and video tutorial. Also, where we have felt that parts of the website could be particularly confusing for the end-user, such as in the XQuery search, we have provided a description of how it should be used.

Non-Functional Requirements

1. The web-application will be licensed under the Apache 2 license.

We have fulfilled this requirement because, with the exception of FontAwesome (which uses the SIL Open Font License), every external piece of code we use is licensed under either the MIT, BSD 2-clause, or Apache 2 licenses. All of these are compatible with the Apache license, and therefore there is no reason for us to be unable to license the complete package under the Apache 2 license.

2. The web-application must be suitable for searching queries, browsing the XML documents, translations, interpretation and scanned pages for these user groups:
    a. Secondary school children aged 11 years old or older;

    Although children this young may not have any knowledge of the languages, or the content the documents contain, the basic search functionality is extremely easy to use even for people in this age bracket. The basic search functionality resembles that of google and will feel familiar to children. The browse feature is also easy to use so would also be appropriate. We therefore feel that we have fulfilled this requirement.

    b. University and college students;

    University and college students will have a clear understanding of how the basic and advanced search work. They also may have an understanding, depending on their area of study, of the content and language of the documents. We also conducted focus group which contained eleven third year history class students of University of Aberdeen. Based on results, it is clear that we have fulfilled this requirement.

    c. Researchers;

    Researchers in the fields such as digital humanities will be the group best suited to take advantage of the applications more advanced features such as XQuery. They will also be able to use the advanced search effectively. Client's researchers regularly tested the

application and the last test was done on the second meeting for beta version presentation. The client approved that we have fulfilled this requirement.

d. The wider public

The wider public with average computer skills, and who are aged 11 years or older. We think that the simple search should be fairly accessible to the general public, and so they should also be able to use the product. We therefore feel that we have fulfilled this requirement.

3. The web-application must be accessible from
    a. Chrome version 50. +
    b. Firefox version 45. +
    c. Safari version 9. +
    d. Edge version 38. +

Apart from simple HTML/CSS/JavaScript, on the front-end, we are using JQuery, Bootstrap, and fullpage.js. JQuery supports the latest current browser version and its predecessor. Given that we started the project when these version were current, jQuery supports all of these. Bootstrap supports the latest versions, and thus the same is true for Bootstrap as for jQuery. Finally, fullpage.js also supports the latest browsers, so the same is true here as well. We can therefore expect our application to work from these browsers. Therefore, the requirement is fulfilled.

4. It will be possible to extend the application such that translation and interpretation views can be added.

This is fulfilled because all we really would have to do is add extra fields in the database, and add options for them within the user interface.

5. The web application will be optimised for these screen resolutions:
    a. Extra small devices, Phones: less than 768px
    b. Small devices Tablet: min-width:768px
    c. Medium devices Desktops: min-width:992px
    d. Large devices Desktops: min-width:1200px

This requirement is also fulfilled. See Testing and Evaluation section.

6. User should receive search results within 5 sec. if their device’s connection to the Internet has a download speed of 1 Mbps, and their upload speed is 256 Kbps.

Elasticsearch is heavily used for many large sites, and is therefore optimised for high

throughput. In our tests, it always delivers results in the order of tens of milliseconds. The same is true of the rendering time for the page on our server (given normal load). The only other variable on which this depends is server upload speed. This is out of our control, since the choice of ISP and connection speed is up to our client, but when deployed on the university server, we have always received results within this time period.

7. The system should be reliable and available at least 363 days in the year.
   See Testing and Evaluation section.

8. It is acceptable to restart the system in the event of failure at most once a week.
   See Testing and Evaluation section.

9. The web-application should support up to 1000 concurrent users. This requirement is bound with minimal hardware requirement specification.
   See Testing and Evaluation section.

10. The system can be restarted at most once a week for maintenance purposes and applying software updates.
    The system restart would normally take up to 5 seconds. It is used to apply software updates and patches.

11. The system should be able to store single file with size up to 1TB.

We support Ubuntu 16.04 (as seen below), which uses the EXT4 file-system. By default, EXT4 uses a block size of 4KiB, and therefore the maximum file size is 16TB.[18] Therefore, this requirement should be met even though we could not test it. See Testing and Evaluation section.

12. The system should be running on Ubuntu 16.04 LTS.

This requirement has been met, inasmuch as all the software used and written works on Ubuntu 16.04.

13. The system should be installed and tested within one week by our team.

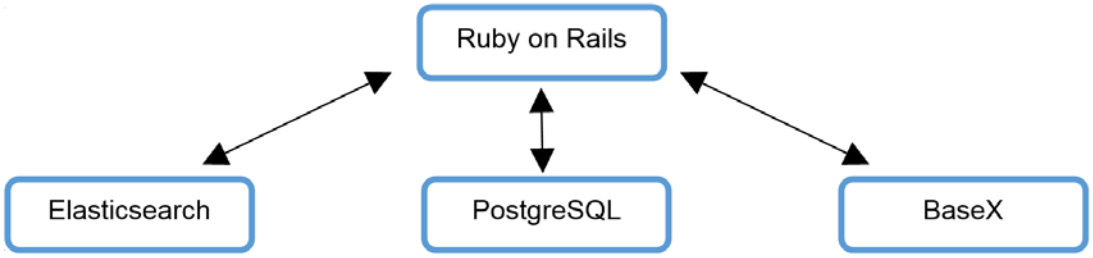
This requirement has been fulfilled by completed installation and testing on a virtual server hosted by IT Services UoA. (http://lacr-demo.abdn.ac.uk [accessible only via University's computers or https://vdi.abdn.ac.uk/])

Design Overview

[Figure 4: LACR Search architecture]

The design of LACR Search is made up of four main components:

- *Ruby on Rails* is the server-side web application framework providing the core structures for our web service and web pages.
- *Elasticsearch*, often initialised to *ES*, is a search engine that offers distributed, multitenant-capable full-text search, and provides functionalities such as suggestions and auto-completion.
- *PostgreSQL* is an object-relational database. Its primary functions are to store and return data securely in response to requests from the web application.
- *BaseX* is a native light-weight XML database management system and *XQuery* processor. Provides support for the standardised query languages *XPath* and *XQuery*.



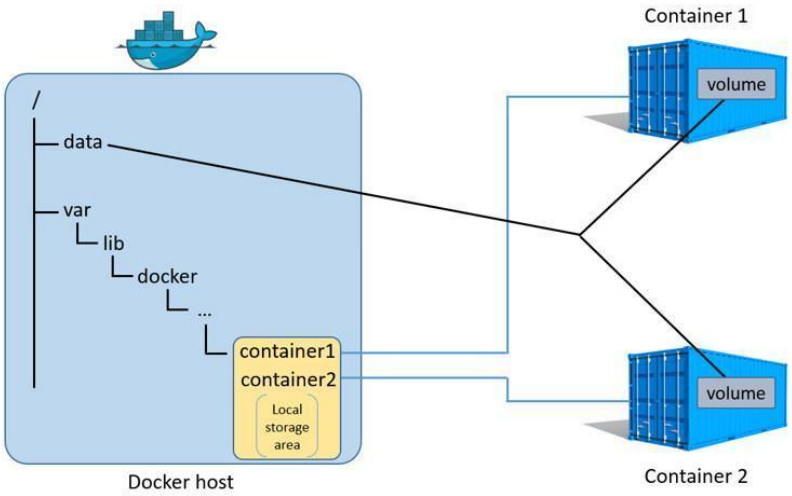
How the main software components relate to each other?

Key implementation decisions

Docker

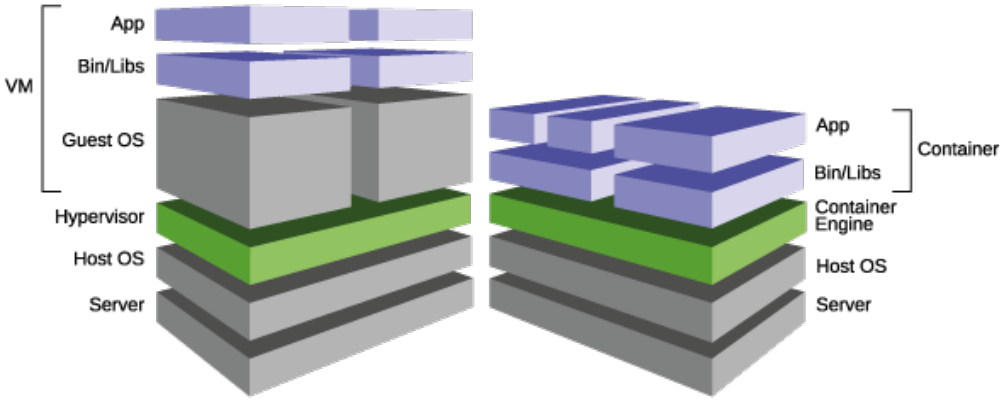
What is Docker?

An open-source project that automates the deployment of applications inside software containers.



What is container?

An operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel.



During development, we decided to isolate each of the key components of our product into a separate Docker container. A Docker container is a single Linux environment - it has its own filesystem, its own processes, and so on. This means that each container is separate from every other container (see the picture). We did this for the following reasons:

- A simplified deployment procedure – we created build recipes for each container in the form of Dockerfile. It contains:
    - Environment variables used for configuring the internal applications;
    - A list of software packages to be installed;
    - Network ports to be exposed; and
    - Files to be imported or copied.

- In addition, a Docker-compose file is used to build and run all containers simultaneously. The compose file also specifies dependencies, for example web-application depends on

database. It is also used for identifying appropriate configurations for different environments such as development, production, testing, and so on.

- Operating-system-level isolation – each container has its own resource limitations (such as CPU time, memory, disk space, network bandwidth, etc.); system administration (Linux Users/User Groups, Firewall rules, Installed software, etc.); and network configurations (IP address, Domain name, and so on). These separations are important to provide robustness of the system. Thus, a failure or stoppage in one container would not affect the work in another. For example, a halt of the BaseX container would not affect Elasticsearch, Rails, or Postgres. Furthermore, this feature allows for better recoverability - in the above case, the only action required is a restart of the BaseX container, which is done automatically by Docker-compose and normally would take no more than a second.

- Enhanced security is provided when different segments of the application are separated. For instance, exploiting a vulnerability in one component such as Elasticsearch would not allow the attacker to gain direct control of any other part of the system such as PostgreSQL.

- Scalability – The core components of the LACR Search system are independent from one another. Thus, each of them can run on different physical machines. Likewise, complete copies of each component can work in parallel, managed by a load balancer. For example, multiple Ruby containers which contain the web application can work in parallel due to being managed by a load balancer, such as Nginx.

- Reliability - Due to the lightweight nature of the container architecture, it is possible to store backup copies - they will be saved on the system, and will be ready to be started in case of system crash or failure. These live backups help reduce the downtime of the system.

Web Framework

The vital decision about the most suitable web framework was made by considering the project goal, time limit and knowledge required. There were four proposed options from which *Ruby on Rails* was chosen.

Ruby on Rails vs Django (Python)

The comparison between these two web-frameworks demonstrated similar advantages and disadvantages. They both have large repositories of open-source libraries, and both have well maintained bindings for APIs of external systems such as PostgreSQL, Elasticsearch, etc. One major motivation to prefer Ruby on Rails over Django was the proved web-development

community providing a plethora of examples and tutorials. The second reason was that two team members already had good experiences with Rails.

Ruby on Rails vs Laravel (PHP)

Better readability and writability are essential advantage of the Ruby programming language over PHP. Dominant characteristics of Rails compared to Laravel are the vast user and developer community as well as the estimated amount of development time.

Ruby on Rails vs Grails (Java)

The Grails framework is the foundation used for the web interface of Mímir. As Mímir provides rapid search for plain text and annotations, building on top of Mímir would have the advantage to provide stable basis for LACR Search. However, the development time and effort required to build the web-application utilising the Java framework would be significantly higher than starting from scratch with Ruby-on-Rails.

Full-text search

This design decision aimed to provide search functionality which allows the user to perform full-text search over the content of transcribed documents. Our choice was Elasticsearch due to its key functionalities:

- Fuzzy query search with configurable preciseness which retrieves similar matching words or phrases;
- Suggestions for misspelled words; and
- Autocomplete functionality

Elasticsearch (ES) vs Gate with Mímir

Mímir is a multi-paradigm information management index and repository which can be used to index and search over text, annotations, semantic schemas (ontologies), and semantic meta-data (instance data). It allows queries that arbitrarily mix full-text, structural, linguistic and semantic queries and that can scale to gigabytes of text. [11]

A comparison with Elasticsearch is not straightforward as they both provide different search functionality. In fact, they have equally good development support and documentation. Having said that, Elasticsearch has a huge amount of support behind it - it is used by Microsoft Azure, EBay, Facebook, Dell, The New York Times, Adobe, GitHub, SoundCloud and others.[5]

This means that any problems we ran into were *much* more likely to have been solved before. We chose Elasticsearch because the primarily functionality of LACR Search is searching for words, and Elasticsearch is very good at this.

Elasticsearch vs Neo4J GraphDB (Ontotext)

Neo4j is built from the ground up to be a graph database. The architecture is designed for optimising fast management, storage, and traversal of nodes and relationships.[8] Utilising a graph database as core component of LACR Search would have the advantage of customisable search functionality with great performance. Neo4j has a great community, good documentation, and development support, as well as many usage examples. The main disadvantage over Elasticsearch would be the requirement to transform the HisTEI XML format to GraphML, which is an XML-based file format for graphs.[6] Solving this problem would require us to change the focus on building well-structured graph representation of the original XML transcriptions. We believe this would have made the development time for the application significantly longer.

Searching for annotations

For this functionality, our project requires a quick XML processor with minimal configuration setup, hardware requirements, and the ability to provide efficient search through the whole corpus. Our choice was BaseX as it satisfies these requirements.
BaseX is a lightweight, high-performance, and scalable native XML database and XQuery processor. It uses a tabular representation of an XML tree structure to store XML documents. The XPath accelerator encoding scheme and staircase join operator are used to speed up XPath location steps. Additionally, BaseX provides several types of indices to improve the performance of path operations, attribute lookups, text comparisons and full-text searches.

BaseX vs Logstash

Logstash is a tool to collect and process log messages. It has collection of configurable input plugins including one for XML format. It integrates well with Elasticsearch and provides the ability to search for XML tags with the advantages this high-performance search engine. However, Logstash is designed to provide a server-side data processing pipeline which reads data from multiple sources. Thus, adapting Logstash to process static XML files and index the tags with Elasticsearch would not be efficient approach to search for annotations. [26]

[Figure 3: Logstash pipeline architecture]

BaseX vs PostgreSQL XML

PostgreSQL provides support for the following:

- XML data type and support functions
- XML export format
- Mapping XML documents to SQL databases

The implementation is based on the open-source library *Libxml* and can store the XML content in two forms: *Document*, and *Content*. Strictly speaking, the XML type does not accept string literals as input. The way to convert text strings to XML data and back are:

- `XMLPARSE (DOCUMENT|CONTENT value) --> xml`
- `XMLSERIALIZE (DOCUMENT|CONTENT xmlvalue AS varchar) --> varchar`

The DOCUMENT format is used with valid XML documents such as the transcriptions created by the LACR team. However, there are a few key disadvantages. The first is lack of support for XML functions from the ruby gem which implements the client for PostgreSQL. Solving this problem would require writing SQL queries which could be used to search with XPath by extending the user input. This solution has a high security risk, and would require time-consuming and exhaustive testing. The consequences of this would be limited search and perhaps inefficient functionality.

Another tricky part would be the separation of the data. For performance and security reasons, as well as optimal data design structure, the XML documents should be stored in separate database. Providing such data isolation has a conflict with Ruby on Rails. The design of this framework was not intended to work with more than one database. Solving this problem would require us to query the second database with our own interface implementation. However, this would not be an easy task for us because PostgreSQL does not implement a REST API.

BaseX vs ExistDB

Comparison between these two native XML databases is difficult, because they both implement same functionalities and offer similar interfaces. However, the quality of the documentation and of the user community of BaseX at the present time is more comprehensive than ExistDB.

BaseX vs Gate and Gate Mímir

Gate is a great tool for all types of computational task involving human language. The key problems our team met when considering adoption of the Gate framework were:

- How we could we integrate Gate with our Rails application?
- How do we provide the search interface?
- How do we import XML documents into Gate's data store?

Gate has a well-documented Java API, and the best way to solve these three problems would be to implement our own client utilising the Java API. One option for this is possibly using the Ruby Java Bridge (http://rjb.rubyforge.org/) that does most of the heavy lifting for us. It functions as a Ruby-to-C-to-Java wrapper. In comparison with BaseX the above three questions are addressed with a client implemented in Ruby which make use of the REST API. An example implementation of such code is provided by the BaseX Team. (http://docs.basex.org/wiki/Clients)

Ruby-on-Rails Architectural Design

The implementation of LACR Search makes the most of the Ruby on Rails web framework, and available open source gems. The design is based on the MVC architectural pattern, and gathers the components into one of three layers:

[Figure 2: Architecture Diagram]

- The Presentation layer – Contains the implementation of Views and Assets.
- The Application layer – Contains all Controllers and some of the Ruby libraries used.
- The Storage layer – Contains all Models which in some cases extend classes from utilised Ruby Gems.

The design of the system adopts some advantages from the MVC architectural pattern:

- High cohesion has been achieved by logical grouping of related actions on a controller together. For example, the Documents controller (which appears in the Application Layer) groups all methods related to the documents stored in the database (upload, destroy, retrieve all available documents, show specific document, and so on) together. A different group of functions is in the Download controller. These are the methods which retrieve the original XML files and images, stored on the hard disk as files. The retrieved documents are copied into a single zip archive, which is temporarily stored and provided for download.

- Low coupling, the very nature of the MVC framework, has been achieved by reducing the dependency between models, views, or controllers.

- Ensuring modification is guaranteed due to the nature of Ruby on Rails. For example, to extend the functionality of the application, the only modifications that need to be

done are:

- Add new routes to lacr-demo/config/routes.rb
- Create new views to lacr-demo/app/views/
- Create new controllers to lacr-demo/app/controllers/
- Create new models in lacr-demo/app/models/

User Interface

Development of responsive web user interface in the short period (10 weeks) assuming our experience, can be very time demanding. Also, there is no point of rewriting what has been properly done. Therefore, we decided to utilise open-source libraries and plugins. The fundamental technologies we use are Bootstrap and JQuery. Supplementary plugins for different functionalities were added during the development process. Some of them are described in the Coding and Integration section.

Data organisation

Uploaded files

The file upload is allowed only for authorised users with administrative privileges.

For the transcription of a document, we accept files with the MIME types of `text/xml` or `application/xml` which use the UTF-8 encoding, and which use the HisTEI XML namespace.

For the scan of the document, we accept files with the MIME types of `image/tiff` and `image/x-tiff`.

The uploaded files are processed and stored on the hard-drive, with information about their metadata and their path stored in the PostgreSQL database.

[Figure 1 Entity-Relationship model]

Each uploaded image is saved in three versions.

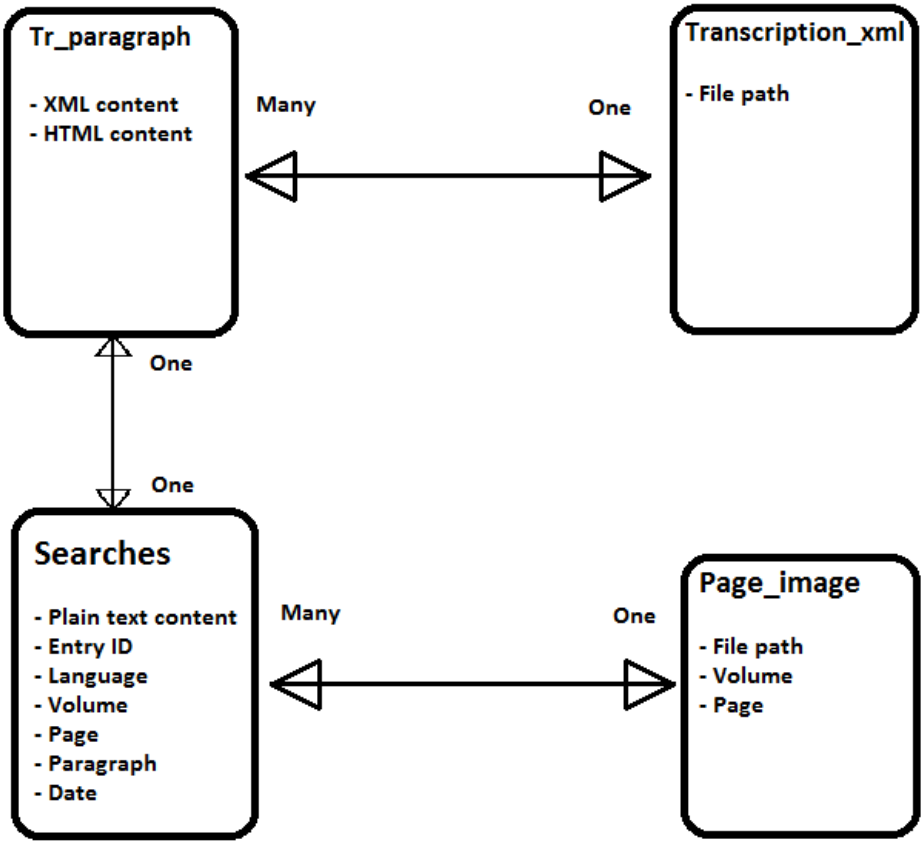
- The original `image/tiff` format is stored. It is accessible for download only by authenticated users with administrative rights.

The compressed images are resized twice: once, to a file with a maximum height of 1024 pixels, and second, to a file with a maximum height of 800 pixels. They are saved in *jpeg* format. Both versions are accessible for view and download to all users of LACR Search:

- The images with height of 800px are displayed next to the transcription.
- The larger versions are made available through JQuery zoom - a JavaScript plugin used to enlarge images on mouse-over. [7]

XML files are parsed during upload. The data needed by the search engine -Elasticsearch- and the query processor -BaseX- is extracted from the files and imported appropriately. In detail, the format required by Elasticsearch is plain-text with no XML elements, where BaseX requires well-formatted XML content.

Furthermore, the search results of Elasticsearch are linked with corresponding images and original XML the content. This is implemented with foreign keys within the PostgreSQL database.



On the above diagram `Searches` represents a database table used to store the data which is made available to Elasticsearch. It has one-to-one relation with the table `Tr_paragraph` which stores the original XML content for each paragraph as well as the equivalent content converted in valid HTML5 format. These two versions of each paragraph (XML and HTML) are

displayed from “Browse” and “View Page” functions. The “Plain text content” is displayed while searching with simple or advanced search.
The other two many-to-one relations are used to link the search results and individual paragraphs with the corresponding XML files and images. These relations are used to display the images, next to transcriptions as well as for the “Download” functionality.
The result is the original XML content is displayed with the plain-text matching the search query alongside with the corresponding image.

Database design

[Figure 1 Entity-Relationship model]

The Entity-Relationships diagram shows the types of information that are saved in the database.There are five main entities:

- XML files (**Transcription_xmls**) – Information about the uploaded XML files.
- Paragraphs (**Tr_paragraphs**) – HTML and XML format of the content of each paragraph.
- Data made available to Elasticsearch (**Searches**) – Plain text content of each paragraph with additional information (such as date, language, etc.).
- Image files (**Page_image**) – Information about the uploaded images and compressed versions.
- Authentication (**Users**) – Data required for user authentication and authorisation.

Relationship indicated as “**Grouped in**” relates Tr_paragraphs and Searches with Transcription_xmls. The data provided to ES and both content formats (HTML and XML) for each parsed paragraph are grouped by the XML file from which they have been extracted. This relation provides a direct link from the search result or stored paragraph to the original XML file. It is used to implement the download functionality.
The association “**Linked**” corresponds to relationship between Tr_paragraphs, Searches and Page_image. It links each XML paragraph with the matching search results from ES and the corresponding image.
The relationship “**Uploads**” corresponds to the file upload functionality, and is restricted to only authenticated and authorised users.

Summary

LACR Search is designed to provide functionalities for all relevant requirements and extend them further to improve the usability of the software. LACR Search can easily expandable with modular components.

- High cohesion of the software elements has been achieved by group similar functionalities within single rails modules. Our main development focus is utilising the Ruby on Rails

MVC architectural pattern. The design principle we followed results in the LACR Search consisting of five modules. Each of them has been described as an entity within the ER Diagram explained above.

- Low coupling has been achieved by separation of modules and guarantee independence between system components. The search functionalities based on Elasticsearch would not affect the query processing based on BaseX. The search is also independent from displaying the document transcriptions as well as the images as these data is stored within PostgreSQL tables and retrieved from separated methods and models.

Coding & Integration

JavaScript and CSS Libraries

JQuery

JQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. We decided to use JQuery for a multitude of reasons: Firstly it gave us the ability to write shorter, easier to read code. This is important in the scope of a group project because it decreased the time which was taken to understand other people's code, this was a huge time-saver. JQuery is a library that all members of the group are familiar with, and find it easier to write than pure JavaScript. JQuery is cross platform, and compatible with all of the browsers and devices we are required to support.

Bootstrap

Bootstrap is the most popular CSS and JS framework for developing responsive, mobile first projects on the web. Bootstrap easily and efficiently scales your websites and applications with a single code base, from phones to tablets to desktops with CSS media queries. Bootstrap allowed us to quickly develop our user interface by giving us access to a multitude of front elements that made creating an attractive and usable front end. Prototype the front end quickly in the early stages of the project outline the basic architecture of the website. This made implementing models, controllers and applying agile software development methodologies easier.

FullPage.js

A simple and easy to use JavaScript library to create fullscreen scrolling websites (also known as single page websites or one page sites). The library provides a straightforward way to

create full screen auto scrolling websites. A task that can seem simple initially but that gets quite complicated once you start covering issues such as browsers compatibility, touch devices, kinetic trackpad scrolling, URL linking, accessibility, responsiveness, call-backs etc. FullPage.js allows the creation of fullscreen scrolling websites, as well as landscape sliders inside the sections of the site. It is fully functional on all modern browsers. It also provides support for mobile phones, tablets and other touchscreen devices. FullPage.js was used to assist us with the front end design of the application. We decided early on in development that we wanted to have a full page vertical sliding landing page. By using this library we were able to dramatically reduce the amount of time it required to implement this feature. Once we had used the library on the homepage we discovered other areas of the website it could be utilised in. The horizontal slide functionality was used to implement the navigation between search results and the page with image and transcription, allowing the user to - go back - without need to refresh the web-page or use the Back button from the web-browser.

Ruby Gems

SearchKick

SearchKick is a gem for Ruby that was used prominently within our application. SearchKick is a popular gem that is used in conjunction with Elasticsearch. It allows for a simplified way to interface the search engine with simple import statement within a model. SearchKick has a number of built-in methods that allowed us to take advantage of complex Elasticsearch features with relative ease. Features such as highlighting, suggestions, autocomplete, spelling variants, pagination and specifying search method would have taken significantly more time and effort to be implemented without SearchKick. The advanced search functionality of the web-tool was developed utilising simple "*where*" conditions provided with this Ruby gem. Built-in functionalities similar to this made setting a range of desired values or other specific equivalence as filter for the search results in just few lines of code.

Devise

Devise is a popular and flexible authentication solution for Rails applications. It is a complete MVC solution based on Rails engines and it also allows to have multiple models signed in at the same time. The overall design of this gem is based on modularity concept, so that it can be specified exactly which module is required. We decided to use these modules:

1. Database Authenticatable - hashes and stores a password in the database to validate the authenticity of a user while signing in.
2. Recoverable - resets the user password and sends reset instructions.
3. Registrable - handles signing up users through a registration process, also allowing them to edit and destroy their account.
4. Rememberable - manages generating and clearing a token for remembering the user from a saved cookie.

5. Trackable - tracks sign in count, timestamps and IP address.
6. Validatable - provides validations of email and password.

Concerning security, the library is securing passwords with bcrypt hashing algorithm. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.

There is a number of reasons why we decided to use Devise. The major one is that it is open source, developed by security experts and used by many companies. People are always looking for vulnerabilities. When vulnerabilities are found, a patch is released. Because of that, we can reduce the amount of time we put into security and focus on the business logic.

Nokogiri

Nokogiri is the most popular open source Ruby gem for HTML and XML parsing. It parses HTML and XML documents into node sets and allows for searching with CSS3 and XPath selectors. It may also be used to construct new HTML and XML objects.
Nokogiri is fast and efficient. It combines the raw power of the native C parser Libxml2 (http://xmlsoft.org/) with the intuitive parsing API of Hpricot (https://github.com/hpricot/hpricot).[23]

Description of the algorithm we used to parse the XML files is the following:

1. Upload of XML files.
   See appendix 1.6

   1.1 The uploaded files are initially stored in the `/tmp` directory of the Ruby container.
   See appendix 1.7

   1.2 Validation of the uploaded files. After the first validation implemented in `xml_uploader.rb` the files are moved in the directory: `lacr-demo/public/uploads/xml/`
   See appendix 1.8

2. Store the information for the uploaded file in the database.
   2.1 Save the file path in PostgreSQL
   See appendix 1.9

   2.2. Parse the XML content of each file.
   2.2.1 Remove the Processing Instructions. These are comments which should not appear on the webpage.

See appendix 2.1

2.2.2 Find all `pb` (page break) elements and create the 1st paragraph with content "The page is empty."
See appendix 2.2

2.2.3 Separate individual `div` elements from the XML files.
See appendix 2.3

2.2.4 Obtain the XML content for each of them and convert it to valid HTML5 format. Store both the XML and HTML content in the `tr_paragraph` table.
See appendix 2.4

2.2.5 Find and extract the date, language, paragraph, page, and volume for each `div` element. Convert the XML format to plain text and store the plain text together with the extracted data (date, language, paragraph, page, and volume) in the PostgreSQL table `searches`. On save the gem Searchkick will upload this information into Elasticsearch.
See appendix 2.5

2.3 Save the XML content in BaseX
See appendix 2.6

2.4 Re-index the new content in Elasticsearch.
See appendix 2.7

Problems encountered during development

Our team ran into a few unexpected issues during the development process of the algorithm for parsing the XML format.

Line breaks `<lb />`

The first challenge was the `<lb />` (line break) tag in XML. The line break is a void tag which means it does not have content. Its role is similar to the `<br />` (break) tag with the only one difference; it is not a standard tag, thus is not recognised as breaking line annotation from web browsers and XML parsers. This problem leads to unexpected concatenation between two words separated only with `<lb break="yes">`. This problem also breaks the search functionality as Elasticsearch search within plain text content obtained by stripping off all XML tags. This problem

becomes slightly more sophisticated when we find out that words divided with <lb break=”no”> should be indeed concatenated.

This issue has been addressed with the following Ruby code:

```
entry_text=(
    Nokogiri::XML(
       entry_xml.gsub('<lb break="yes"/>', "\n")
)).xpath('normalize-space()')
```

The solution is to replace the string `<lb break="yes"/>` with the new line character. However, we met this problem again when we were visualising the XML on the web-page. Since `<lb />` is void tag the best solution is to replace it with the corresponding HTML5 valid tag `<br />`. Another replacement which also works is `<span style=”display: block”></span>`.

The problem has been resolved in the function `xml_to_html` implemented in the file `lacr-demo/app/models/transcription_xml.rb`.

This tag is also an obstacle when working with XQuery. The problem appears when the function string() is used. The result is a string which contains the text content of XML element and all sub-elements. The outcome is again unexpected concatenation of two words separated only with this tag.
Resolving this problem could be done in different ways. One of them is:

```
normalize-space( string-join( <XML ELEMENT>/text(), ' ') )
```

Page breaks `<pb />`

Another tricky part related with the XML format is the <pb /> tag. It is a void tag used to indicate the end of page and includes the page number as attribute. At first we ignored these tags since the page number is already included in the ID of an entry and there was no reason search for page breaks. However, the problem was pointed from the LACR team when they noticed that pages indicated as empty (using `<note></note>` tag) would not be found. It is important to display that a page is empty, otherwise the user might wonder why some pages are missing.
The problem is not trivial as page breaks might appear everywhere.

An example:

Green indicates the previous page.
Black represents the page break.
Blue is the next page.
Orange indicates the incorrect displayed content.

- Case 1: Page break appear between entries (This is not considered as a problem since it would not provide any additional information.)

```
<div xml:id=”ARO-5-0020-3” xml:lang=”lat”>
…
</div>
<pb n=”21” />
<div xml:id=”ARO-5-0021-1” xml:lang=”lat”>
…
</div>
```

- Case 2 has been solved in the following way: For each page break in the XML file a new entry in the database is created with paragraph number set to 1 and with content “Page is empty”. This has record is overwritten from the actual record if exists, otherwise the message “Page is empty” will appear when these document page is shown.

```
<div xml:id=”ARO-5-0020-5” xml:lang=”lat”>
…
</div>
<pb n=”21” />
<note type="editorial">Page is empty.</note>
<pb n=”22” />
<div xml:id=”ARO-5-0022-1” xml:lang=”lat”>
…
</div>
```

- Case 3 is has been solved by representing the paragraph (XML format) as a string which is split in two parts. The first part of the paragraph is stored with the corresponding page, volume and paragraph numbers, where the second part is appended at the beginning of the first entry on the next page.

```
<div xml:id=”ARO-8-0028-3” xml:lang=”lat”>
…
<pb n=”29” />
…
</div>
<pb n=”30” />
<div xml:id=”ARO-5-0030-1” xml:lang=”lat”>
…
</div>
```

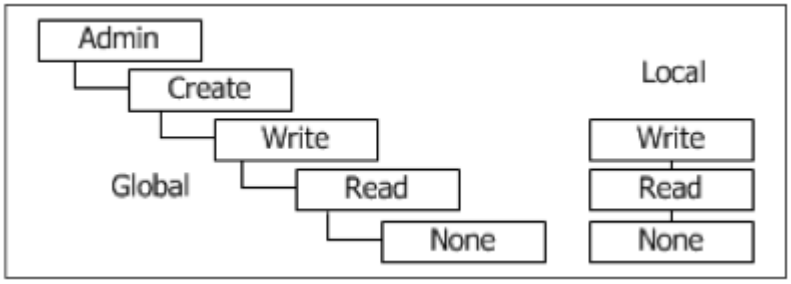
BaseX Client’s Security

The adoption of BaseX within LACR Search has been made at the later stage of the

development process. It is also an example of how the LACR Search could be further extended. The whole setup of BaseX at the current stage of the software works independently from the rest of the application. It has own Views and Controllers. Model was not needed as the interaction with this database has been implemented within a modular library `lacr-demo/lib/BaseXClient.rb`

The problem our team encountered was with restricting the user control. This issue has been solved with two key design decisions:

- The BaseX database is only used for the XQuery search and nothing else. This restrict the user input to only valid XQuery and XPath expressions. Everything else returns an error. This is also known as Query mode.
- When XQuery is used with admin privileges it allows full control over the XML database - create/modify/remove documents and users. This problem has been solved with the BaseX user management system. The user's XQuery expressions are executed from BaseX user with only read privileges.



Hierarchical representation of the User Management System of BaseX.

HisTEI Stylesheet

One of the most difficult issues our team encountered was: "How to style the XML content of the transcribed documents?"

The problem comes from the fact that the XML namespace (http://www.tei-c.org/ns/1.0) used by our client include large amount of XML elements. Creating unique style for each one of them is truly time-consuming task. Thankfully, Dr Mike Olson had developed an open-source framework add-on for the Oxygen XML Editor, licensed under the MIT license.

We have adapted the CSS styles and images from the HisTEI add-on with a few modification:

- Remove all Oxygen functions which are not appropriate.

  For instance the following function

```
-oxy-append-content:
        oxy_editor(
            rendererClassName,
"info.histei.contextual_info.ContextualEditor",
            swingEditorClassName,
"info.histei.contextual_info.ContextualEditor",
            edit, "@ana"
            contextual_type, "ann"
            contextual_filter, "phrase"
        )
;
```

- Replace all oxygen functions with equivalent CSS3 alternative.
- For example, the following:
  `-oxy-prepend-content: url('../images/bloc-16.png');`
  Must be replaced with:
  `content: url('bloc-16.png');`

- Change the style rules to define CSS class rather than element name. (Appending a dot at the beginning of the rule)

Testing and Evaluation

Tests are an important part of software development process. We began writing the tests simultaneously with the implementation. Fortunately, testing support was integrated into the fabric of Rails. Rails creates a test directory at the point the project is first created. During development we used three environments: development, test and production. This is useful to us and is another benefit of using Rails. Each environment's configuration can be modified. For instance, to setup our test environment we have to modify the options found in `config/environments/test.rb`. Also there can be separately defined databases for each environment in the configurations found in the file config/database.yml. Additionally, it is possible to specify particular gems which should be used in those environments. This can be done in the Gemfile. In our case there are these extra gems for test environment: Cucumber rails, database_cleaner, and selenium webdriver. See appendix 1.1

Since Ruby-on-Rails is reliant on the MVC pattern, all the tests are also split into separate folders which represent this pattern. The strategy for testing is performed in four steps: unit, integration, validation and system testing. We used Minitests, which are the default tool in Rails for unit and integration testing. For validation and GUI testing we decided to use Cucumber with Capybara. We used a bottom-up approach, first writing tests for models then controllers and finally views. All of those Black-box tests are designed to test equivalence classes, boundaries, random and impossible values. It is also important to mention that we used Git version control, and before every new commit or branch merge we run all the tests. This approach greatly reduced the overall

amount of bugs and conflicts in our code.

The first gem we installed -for the purposes of testing- was cucumber-rails which adds the possibility to write cucumber tests within the rails application, it also installs Capybara for GUI test automation. Our justification for choosing cucumber was that the tests are written as simple scenarios and it is fairly easy, even for non-developer to read and understand them. These scenarios with steps definition can be found in `features`. This is one example of scenario used for user log in testing. See appendix 1.2

This scenario is a plain English test with keywords which requires steps defined for every: “Given”, “When”, “Then” and “And” statement. Due to the help of Capybara we were able to automate the GUI testing. Selenium-webdriver gem is required only for JavaScript testing. The main advantage of using Cucumber is that we were able to rewrite some of the functional and nonfunctional requirements as test scenarios. This turns out to be very convenient, especially for validation testing. Here are steps definition for the scenario above. The statements need to be true in order to pass the test. See appendix 1.3

As you can see, there are no get or post methods because Capybara was developed to simulate what the user can see. Whereas those links, input fields and buttons can work without any problems while testing, they could be hidden, overlapping each other or have the same colour as a background. Therefore we had to also check the web interface manually, but overall those tests helped us reduce the amount of time required for manual testing.

These kind of tests clearly do not test the random or impossible values. These tests are designed to test the GUI’s links, responses and JavaScript functionality. For complete testing we developed separate tests for Model and Controller. As it was mentioned above, tests for Models are implemented using Minitests. They are designed to test all equivalent classes, boundaries, random and impossible values. Our approach was to test not only if exceptions were thrown from invalid input, but also to test the exception itself. This could be seen in the appendix 1.4 . The pass criterion is defined in the code. Here is a one short snippet of the code for testing the User Model. See appendix 1.4

Every single test has its own name that represents the case. Then you can notice strings in double quotes which are displayed only if the assert function fail. With help of these strings we are able immediately spot the problem and solve the bug.

The next step was tests for the Controller. The main advantage of separate testing of MVC is that we can very easily determine where a bug is. Tests for controller are independent from Views, but they rely on its Model. These tests were also designed to test equivalence classes, boundaries, random and impossible values. Here is a snippet of the test code for the User controller. See appendix 1.5

There are also tests which cannot be automated within a reasonable time and it is easier and faster to do them manually. One such example is that the web application needs to be optimized for the following screen resolutions:

(a) Extra small devices, Phones: less than 768px
(b) Small devices Tablet: min-width: 768px
(c) Medium devices Desktops: min-width: 992px
(d) Large devices Desktops: min-width: 1200px

This test was done manually and the pass criterion for each resolution was that all the elements on the web page must be visible and possible to click on them without using the horizontal scroll bar.

Finally, we need to mention tests which we could not conduct due to the technical restrictions we were facing. We could not test the application in terms of scalability. This means we could not even test one of our non-functional requirement that the web-application should support up to 1000 concurrent users. Despite that we are fairly confident that the application would handle this easily. Since we are using Docker, the application can be quickly deployed on another servers and load balancer can distribute the clients among these new copies of the application.

№	**Description**	**Test**	**Pass**
	Functional requirements	----------	-
1	The application should parse all XML tags required by the TEI standards, as well as those required by the namespace	We uploaded several TEI documents and checked the results in the database.	**OK**
2.a	User should be able to search for plain-text strings	We searched for plain-text string. Test passes if it displays desired string and only if the string exists in the uploaded files.	**OK**
2.b	User should be able to search through documents by ID (volume, date, page)	We searched by ID of a document. Test passes if it displays relevant result and only if some document has this ID.	**OK**
2.c	User should be able to search through documents by proper names	We searched for a proper name. Test passes if it displays relevant result and only if some document contains the searched proper name.	**OK**
2.d	User should be able to search through documents by annotation	We searched for an annotation. Test passes if it displays relevant result and only if some document contains the searched annotation.	**OK**
3	User should see the search results and 30 characters surrounding the search are displayed	We searched for “Johnson” in the corpus. Test passes if it displays relevant result and 30 characters surrounding this result.	**OK**
4	User should see the document ID in the search results.	We searched for “Johnson” in the corpus. Test passes if it displays relevant result and ID of the result.	**OK**
5	User should see the date of the document in the search results	We searched for “Johnson” in the corpus. Test passes if it displays relevant result and date of the document.	**OK**
6	User should be able to view a list of documents in chronological order. (they can also sort by volume, page ascending and volume, page	We searched for “Johnson” in the corpus and ordered the results based on chronological order. Test passes if the search results are in chronological order.	**OK**

	descending)		
7	Users can upload all relevant data (XML files, translations, interpretations and scanned pages)	We uploaded XML files (transcriptions) and tiff files (images). Only these files are relevant for now. Test passes if the application successfully upload and store the documents and images.	**OK**
8	User should be able to download the XML file of desired document.	We selected a few pages and tried to download the xml files. Test passes if we successfully downloaded the desired XML files.	**OK**
9	User should be able to download the jpeg with a watermark of desired document.	We selected a few images of the documents and tried to download them. Test passes if we successfully downloaded the desired images. The requirement of watermark was added at the last time by our client and it is included in our future plans.	**OK**
10	User should be able to toggle all annotations on and off.	This requirement has changed. Therefore is not fulfilled and tested.	**FAIL**
11	User will be provided with documentation.	We provided user manual and also video tutorial for a new user.	**OK**
	Non-functional requirements	----------	-
12	The web-application will be licensed under the Apache 2 license	The software is licenced under the Apache 2.0 license.	**OK**
13	The web-application must be suitable for searching queries, browsing the XML documents, translations , interpretation and scanned pages for secondary school children aged 11 years old or older	Not tested. We did not have enough time to conduct the test on this user group and evaluate results.	**FAIL**
14	The web-application must be suitable for searching queries, browsing the XML documents, translations , interpretation and scanned pages for university and college students	We held a focus group of eleven third year history class students from the University of Aberdeen. The evaluated results can be seen below in the section “Evaluation”.	**OK**
15	The web-application must be suitable for searching queries, browsing the XML documents, translations , interpretation and scanned pages for researchers	Client’s researchers regularly tested the application and the last test was done on the second meeting for beta version presentation. Test passes when client approve that the application is suitable for all these tasks.	**OK**
16	The web-application must be suitable for searching queries, browsing the XML documents, translations, interpretation and scanned pages for the wider public with average computer skills.	Not tested. We did not have enough time to conduct the test on this user group and evaluate results.	**FAIL**
17	The web-application must be accessible from Chrome version 50. +	We tried all the application functionality on Google Chrome Versions 50.0.2661.75 (64-bit) and current stable 56.0.2924.87 (64-bit). Test passes if the web application and its functionality works as expected.	**OK**
18	The web-application must be accessible from Firefox version 45. +	We tested all of the application's functionalities on Mozilla Firefox Versions 45.0 (64-bit) and current stable 52.0.1 (64-bit). Test passes if the web application and its functionality works as expected. Some Capybara tests for JavaScript are also using Firefox because of selenium webdriver.	**OK**

19	The web-application must be accessible from Safari version 9. +	We tried all the application functionality on Safari Versions 9.0 and current stable 10.0.3. Test passes if the web application and its functionality works as expected.	**OK**
20	The web-application must be accessible from Edge version 38. +	We tried all the application functionality on Microsoft Edge Versions 38.14393 (64-bit) with EdgeHTML 14.14393 (64-bit). Test passes if the web application and its functionality works as expected.	**OK**
21	It will be possible to extend the application such that translation and interpretation views can be added.	The application can be easily extended. Nothing is limiting this.	**OK**
22	The web application will be optimized for extra small devices, Phones: less than 768px	We tested this requirement on Google Chrome with device toolbar (Ctrl + Shift + M) where we set up the resolution to 412px x 732px (Nexus 5X). The pass criterion is that all the elements on the web page must be visible and possible to click on them without using the horizontal scroll bar.	**OK**
23	The web application will be optimized for small devices Tablet: min-width:768px	We tested this requirement on Google Chrome with device toolbar (Ctrl + Shift + M) where we set up the resolution to 768px x 1024px (iPad). The pass criterion is that all the elements on the web page must be visible and possible to click on them without using the horizontal scroll bar.	**OK**
24	The web application will be optimized for medium devices Desktops: min-width:992px	We tested this requirement on Google Chrome with device toolbar (Ctrl + Shift + M) where we set up the resolution to 1024px x 1366px (iPad Pro). The pass criterion is that all the elements on the web page must be visible and possible to click on them without using the horizontal scroll bar.	**OK**
25	The web application will be optimized for large devices Desktops: min-width:1200px	We tested this requirement on Google Chrome with device toolbar (Ctrl + Shift + M) where we set up the resolution to 1280px x 1024px. The pass criterion is that all the elements on the web page must be visible and possible to click on them without using the horizontal scroll bar.	**OK**
26	User should receive search results within 5 sec. if their device's connection to the Internet has a download speed of 1 Mbps, and their upload speed is 256 Kbps.	We tried to access the web page (lacr-demo.abdn.ac.uk) via University network where we specifically set the network speed to 1 Mbps / 256 Kbps and measured the elapsed time. Test passes if the average time required to load the page is below 5 sec. The average time was 3.48 sec, but it increased while loading high resolution images.	**OK**
27	The system should be reliable and available at least 363 days in the year.	Not tested. But during the whole semester we did not encounter any failure of the application. So we could assume that this requirement will be fulfilled in the future.	**FAIL**
28	It is acceptable to restart the system in the event of failure at most once a week.	Tested on the University VM. We did not encounter any failure. We've always kept production version separately and pushed the changes only after we tested the development version. Therefore we've reached high stability of the production environment.	**OK**
29	The web-application should support up to 1000 concurrent users. This requirement is bound	Not tested. This requirement could not be tested due to the technical restriction we were facing. The other thing is	**FAIL**

	with minimal hardware requirement specification.	that we host the application on the University's server and this test could be classified as DOS attack. Therefore we did not even try.	
30	The system can be restarted at most once a week for maintenance purposes and applying software updates.	We tested this requirement on the University's VM machine. There is no need to restart the system. The only reason when this has to be done is during the updates. This can be prevented with a copy of the web-application and load balancer to reroute the traffic during update	**OK**
31	The system should be able to store single file with size up to 1TB.	Not tested due to the technical restrictions. The VM does not have enough memory for this test.	**FAIL**
32	The system should be running on Ubuntu 16.04 LTS.	Tested. The VM is running this OS.	**OK**
33	The system should be installed and tested within one week by our team.	Manual test. We created a new virtual machine and deployed the OS and all the required software within an hour.	**OK**

Evaluation

As previously mentioned, the team used an agile development approach. This means that we have informed our client regularly about the latest improvements and collected feedback based on which we implemented proposed changes. This cycle repeated several times and we have kept regular email communication. Because we were able to get feedback very often and quickly, we did not waste too much of our time on developing undesired functionalities. This routine feedback loop also helped us to optimize the user interface.

In addition to the regular email communication with our client, we also held two major meetings. The first meeting was at the beginning of the semester and it was designed to introduce the alpha version of the application. The alpha version already had the basic functionality such as simplified search tool and simple user interface. Based on received feedback we redesigned the user interface and also decided to implement combination of Elasticsearch and BaseX into the application. Those two changes was probably the biggest ones among others. The second major meeting with the client was approximately three weeks before the end of the semester when we finished a beta version. This time, we also focused on acceptance testing where we demonstrated all the desired functionality specified in the requirements. At the end of the meeting we again received feedback which was positive but also includes a few minor changes. Since we deliberately chose to present the beta version with time reserve we had enough time to implement them.

On top of that we also held focus group two weeks before the end of the semester. The group consisted of eleven third year history class students. We provided brief introduction to the application and its functionality. Then they were asked to do tasks we prepared for them. We provided simple questionnaire which served as a guide to this exercise.

These were the tasks and questions we asked:

Section 1 (LACR Search)

Can you use the web-site to search for your own name in the documents? If your name gives no results, try using the name "Johnson".

- How easy was to do this? Scale from impossible 1..10 up to Extremely easy
- What did you find? Open answer.
- What questions did this prompted? Open answer.
- What could be improved or made clearer? Open answer.

Section 2 (Advanced Search)

Using Advanced Search try adding additional search criteria to your search such as volume, page or date.

- How easy was to do this? Scale from impossible 1..10 up to Extremely easy
- What did you find? Open answer.
- What questions did this prompted? Open answer.
- What could be improved or made clearer? Open answer.

Section 3 (Browsing documents)

Aside from searching you can use the web tool to simply browse through the content of the documents.

- How easy was to do this? Scale from impossible 1..10 up to Extremely easy
- What could be improved or made clearer? Open answer.

Section 4 (Try downloading some of the documents)

Select pages you would like to download using the checkboxes next to the page number. Then use the download button.

- How easy was to do this? Scale from impossible 1..10 up to Extremely easy
- What could be improved or made clearer? Open answer.

Section 5 (Design your own searches on a topic.)

For instance look up occurrences of a particular place name, or type of office-holder (e.g. alderman, provost, bailie), or product (e.g. salmon), or references to ships, etc.

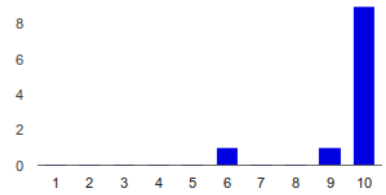
- What did you search for? Open answer.
- What did you find? Open answer.
- How your sequence of searches using the tool evolved through a few iterations?

At the end we did detail analysis of all responses. The results were clear. Majority of the responses evaluated the application as an excellent piece of work, very easy and intuitive to use. Moreover, we also received a few suggestions how we could improve the user interface and add some new functionality. This information will be used in the future development. We have learned

and seen that agile approach and regular feedback are very important and efficient way of developing software.

Simple Searching

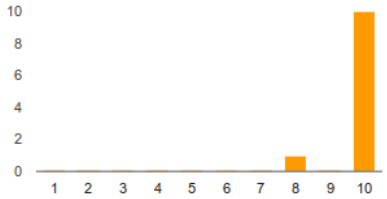
How easy was to do this?



Impossible: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**0**	0%
6	**1**	9.1%
7	**0**	0%
8	**0**	0%
9	**1**	9.1%
Extremely easy: 10	**9**	81.8%

Advanced Searching

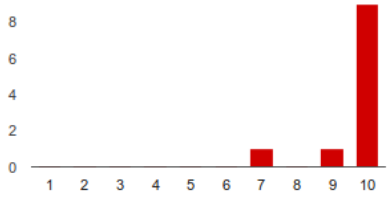
How easy was to do this?



Impossible: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**0**	0%
6	**0**	0%
7	**0**	0%
8	**1**	9.1%
9	**0**	0%
Extremely easy: 10	**10**	90.9%

Browsing documents

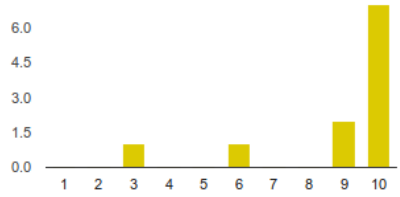
How easy was to do this?



Impossible: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**0**	0%
6	**0**	0%
7	**1**	9.1%
8	**0**	0%
9	**1**	9.1%
Extremely easy: 10	**9**	81.8%

Try downloading some of the documents

How easy was to do this?



Impossible: 1	0	0%
2	0	0%
3	1	9.1%
4	0	0%
5	0	0%
6	1	9.1%
7	0	0%
8	0	0%
9	2	18.2%
Extremely easy: 10	7	63.6%

Although the final product we delivered is working flawlessly, we had to face many problems and difficulties. One of them was the problem with Devise. The original database design was with two separate tables for “User” and “Admin” model. These models should inherit from “Patron” model. However, devise library had unexpected problems with this approach and it was necessary to change it. We have changed the design and all users are in one table and a Boolean attribute distinguish between user and admin rights. The next problem was with current version of Capybara and selenium webdriver. We have tried to use it but it has several issues and we had to swap to older version of Selenium webdriver 2.53.4 and Firefox 47.0.1.

Our team started to work intensively and immediately at the beginning of the semester. We realised the importance and need for this tool. Our client relied that we are the right team to develop and deliver the application in constrained time period. Teamwork was the key element which boosted the development process. It is also important to mention our guide, Dr Adam Zachary Wyner. The knowledge and guidance he has imparted upon us has been a great asset throughout this semester. We can critically evaluate and claim that we, as a strong and united team, have successfully fulfilled all of the relevant client’s requirements.

Comments from our client:

“The web tool has provided a prototype platform for the display of content from xml files, alongside the corresponding images. This alone has been useful in prompting discussion within the LACR project of aspects relating to our future intentions for hosting this material, and for prompting discussion of our own work processes. The prototype is impressive, in its clarity and ease of use. It enables browsing of xml content and images. Its most powerful feature is perhaps the searching function, both ‘simple’ and ‘advanced’, allowing for adjustable fuzzy searches across the corpus sample, and refinement of relevant search criteria. This will be a useful tool for the project team to interrogate the corpus. It also includes additional features, such as the download of selections in pdf format, which will be tremendously helpful as a research tool. I really value the work Team Charlie has put into creating this tool.”

Dr JACKSON ARMSTRONG
Principal Investigator and Project Director

“Team Charlie have created an excellent web-tool for the Aberdeen Council Registers. The tool allows browsing through the transcriptions that the LACR team have uploaded, with the original image being displayed alongside the transcription. This feature is useful to get an idea about what is included in the Council Registers and what they look like. In addition, the web-tool offers a variety of search functions. The ‘simple search’ is enhanced by allowing users to indicate the number of spelling variants of the search term. This functionality is essential when searching terms in non-standardised texts. The ‘advanced search’ allows users to additionally specify dates, volumes, pages, paragraphs and entry IDs. The possibility of specifying date ranges will be particularly useful for research on the Aberdeen Council Registers. The search results can then be displayed in a list and a page view, both allowing individual entries to be selected for further investigation. This means that researchers can create their own list of relevant entries, which can then be downloaded as .pdf files. This is particularly helpful for the distribution of specific material from the corpus. More complex queries can be carried out by using XQuery. Using regular expressions, this search functionality can be used to combine searches for XML annotations and search terms. For example, it allows researchers to see how many entries are in Scots or Latin within a certain period of time. This is particularly useful to investigate the change of linguistic varieties over time. Overall, the LACR team is very grateful for and satisfied with Team Charlie’s work. ”

Dr Anna D. Havinga
LACR Team

Even though, we are aware that this project is not at the end but at the beginning. Said that, we have to clearly state our future goals.

Future Plans

Our web-tool was designed in the scope of a larger project spanning over three years. The software, in its current state already has potential to further the projects teams' research: firstly by allowing them to search through documents which have been transcribed and uploaded. Secondly the XQuery is already being used to analyse the patterns of word use and frequency throughout the corpus. The XQuery functionality will have further use in a vast number of use cases for textual analysis of the corpus. However, as the LACR project continues as the project continues, and new possibilities for analysis the text emerge, it will become evident that our application currently has some shortcomings. The application can be easily expanded further down the line, either by removing, adding or modifying features. Our application was built with high modularity and comes with an exhaustive user and maintenance manual, meaning expansion would be relatively simple.

The CLARIN Network

One requirement which was left unfulfilled was integration with the CLARIN network. The CLARIN network is a Europe wide research network that works to archive language related documents within the fields of humanities and social sciences Scholars use CLARIN as a sort depository where they can exchange resources or work on joint development projects as well as exchanging information on standards and procedures for the archiving of research data. CLARIN surpasses our application in terms of scope, but its multi-faceted approach leaves our project a gap in the market to bring a much more specialised tool to the digital humanities community. Our application has a number of key features that greatly benefit this sector of the market. CLARINs revenue comes from a subscriptions it collects from each of its members, the subscription costs around £1000 per year.[16]

To enable researchers to search for specific patterns across collections of data, CLARIN offers a search engine called Aggregator. Aggregator searches across many different corpora that belong to various different projects that are part of the CLARIN network. All of the corpora are stored in external repositories. Aggregator uses CQL (contextual query language) which translates the user's query to a CQL query which is then transformed into a query the respective search engine can understand. It was originally planned for our project to be integrated with CLARIN so Aggregator could be used to query LACR Search. This was a feature that both the CLARIN team and our guide were keen to have us implement. It would have be beneficial for the project because our project would be featured on the CLARIN homepage and would be publicly available. The LACR team would also benefit from this by being able to partake in these forums where they can exchange resources or work on joint development projects.

It was our original plan to include our project with this network. We had contacts within CLARIN that we would be able to communicate with to include our project. It would be greatly beneficial to both the LACR team and our project as a whole. Our project would be featured on the CLARIN homepage and would be publicly available. The LACR team would also benefit from this by being able to partake in these forums where they can exchange resources or work on joint development projects.

However this idea eventually broke down. Our contact was not responsive when sent emails, this was incompatible with the timescale we had to develop this app. There were also 'catches' not previously known to the team. Namely an annual payment of £1000 to be a part of the network. However, since we have already gathered a lot of information about CLARIN, such as costs and how we could integrate our search with CLARIN, it now mainly comes down to the matter of funding and a small bit of code. We had a contacted with one of the developers from Clarin and discussed:

"*How we can integrate our project within the Clarin network?*"

"Your project sounds very interesting and I think the resources would be valuable to the CLARIN community. The exact efforts required to make your resources available discoverable within CLARIN depends in part on the availability of the resources; if I understand correctly these would be the TEI documents, and possibly also other resources such as scans? If they are going to be openly available there is no need to integrate with our single sign-on/federated identity infrastructure. If some kind of authentication is required, the story gets more complicated (or interesting depending on how you look at it ;)) not just from a technical point of view but also legally and organisational. I will include some links that explain this at the end of my message.

Making resources discoverable within CLARIN happens through our metadata infrastructure. You can provide either Dublin Core metadata [1] or "Component Metadata", the latter being the more expressive preferred standard within CLARIN. This is where the Component Registry comes into play, allowing you to define your own metadata schema within the component metadata framework (CMDI). You don't have to implement any APIs to produce CMDI, but depending on your needs you may need to design a custom metadata model. All of this is explained at <https://www.clarin.eu/content/component-metadata>.

At CLARIN we 'harvest' metadata from various sources using OAI-PMH [2]. Support for OAI-PMH is something you may have to implement unless you are using one of the existing off-the-shelf solution. More info is available at <https://www.clarin.eu/content/oai-pmh-cmdi>.

I can also recommend having a look at the CLARIN metadata FAQs <https://www.clarin.eu/faq-page/267>, which contain a lot of information about the full cycle of metadata modelling, creation and publication.

[1] http://dublincore.org/metadata-basics/
[2] https://www.openarchives.org/pmh/

”

Twan Goosen
Software Engineer
CLARIN ERIC

The explicit instructions provided from Twan Goosen were really helpful for us to understand the internal infrastructure of CLARIN and how we could make the LACR resources available through their network. These instructions could be used for the future development of LACR Search.

As of now plans are in place to, at a later stage in the project, gather the resources required and use the newly acquired contacts to include the project in CLARIN.

Latin and Old Scots Dictionary

Another feature we had planned to implement, but failed because of time restraints was the integration of a Latin and Old Scots dictionaries. The reason for implementing these dictionaries would be to provide translations for transcribed documents. We believe the best way to implement this would be for a popup box to appear when a user highlights a word. In the popup box a translation for the relevant word would appear.

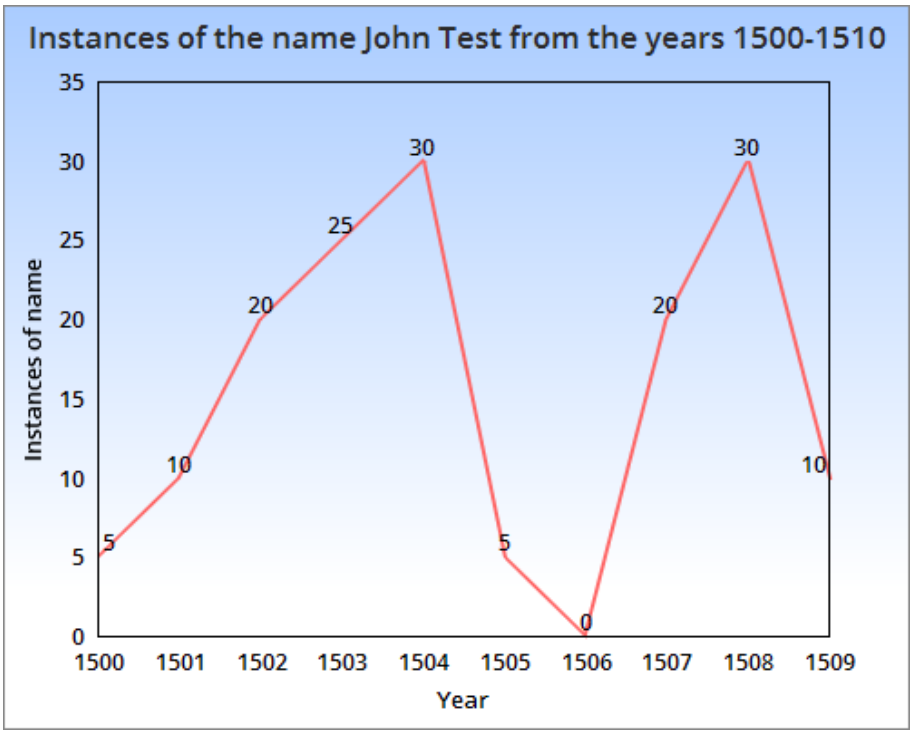
We conducted research on which dictionaries could be used and have a general idea of how they could be integrated with our web-tool. There are a number of Latin dictionaries available, and we believe the best option would be to use the Hunspell dictionary because it is open source and is already used by a number of large companies such as Microsoft and Openoffice. [32] For Scottish dictionaries there are significantly fewer options but, the best option we found was the “Dictionary of the Scottish Language. [31] The DSL is currently just a web resource, so the implementation of this could be difficult but there are a few options open to us. Firstly, contact could be made with the creators of the dictionary so a list of the words in the dictionary could be made available. Alternatively, integration could be achieved with making use of their own service.

Visualization of Data

One feature we had preliminary discussions about implementing was a data visualization tool. The feature would take a search that was performed by the user and present it to them in the form of a graph. For instance it could illustrate to the user how many instances of a certain name occurred over a specified time range. The LACR team expressed a desire for this tool as they believed it would help with later stages of their research, namely textual analysis.

If the user wanted a graph to illustrate the instances of the name John Test between the

years 1500 and 1510 it may look like the following:



In addition to allowing users to create these graphs the user could also be given the option to download these graphs This would be a useful feature as it would not only allow someone to review the findings of their own research but also the ability to present the research. Data visualisation would reveal new paradigms with the text that would not be obtainable without it. (E.g. frequency of occurrence of person's name, event or a place throughout the years.)

Comments

Another feature that we did not have time to implement into the app was the ability for a user to add comments about the documents. While it never saw the light of day the feature was subject of discussion during several team meetings and numerous concepts were thought of and explored. The first of two main ideas that was discussed by the team was a comment section similar to many successful sites with thriving communities such as Facebook and Youtube. Users are able to submit a comment to a particular document were it will leave the comment itself and the username of who left it. Users are also able to see all comments left under the document and who left them.

This would be beneficial to a user as it would allow users to discuss and debate findings within the documents with other users without being face to face. However after several discussions on this feature it was decided that this format of comments was not the most efficient for the app. Again taking inspiration from elsewhere the team landed on the second concept which takes inspiration from sites such as Google Docs. Unlike the previous idea, the comments are not placed under the

document. Instead they are left within the document itself. This is done by allowing the user to highlight text within the document before leaving the comment. When a user hovers their mouse over the highlighted text it will display the comment and the replies to the comment.

We believe that the second concept would be more suitable for this app as the topic of most if not all discussions will refer to specific parts of text within the document. By using the second concept there will be no confusion in what piece of text is being referenced as it is highlighted by the person who left the comment. It also means that users don't have to search through a list of comments before finding some comments on a particular section.

While this feature would greatly improve the app, our team simply did not have the time to implement it. However it is our hope that the research and discussions we have made about the idea will allow the implementation of this feature to become a reality in the future.

Conclusion

We feel delighted our contribution to the Law in the Aberdeen Registers project was found helpful by the LACR team. It was an extremely enlightening experience being given the opportunity to be part of such an interesting project. The application being part of such an expansive domain, - digital humanities - gave our team the chance to build something that we believe can have a big impact. We hope our application continues to benefit the team with their research and analysis. There is clear a vision of how the application can be built upon and used as the basis for much greater things.

Appendices

Appendix 1.1

```
group :test do
      gem 'cucumber-rails', :require => false
      # database_cleaner is not required, but highly recommended
      gem 'database_cleaner'
      # for testing JavaScript, require older version selenium and
      #Firefox version 47.0.1 for more info check:
      # https://github.com/teamcapybara/capybara#drivers
      gem 'selenium-webdriver', '~> 2.53.4'
end
```

Appendix 1.2

```
Feature: User Log in
      In order to log in
      As a valid user
      I need to fill in log in form and submit.

Background:
      Given I am on the home page
      And I have chosen to log in

Scenario: Successful login

      Where someone tries to login with valid credentials

      When I enter my email
      And I enter my password
      And I click on "Sign in"
      Then I should see "Signed in successfully."
      And I should be on the home page

Scenario: Invalid password

      Where someone tries to login with incorrect password

      When I enter my email
      And I enter invalid password
      And I click on "Sign in"
      Then I should see "Invalid Email or password."
```

```
      And I should be on the login page

Scenario: Registration does not exist

      Where someone tries to login with email which is not registered

      When I enter an email which is not registered yet
      And I enter valid password
      And I click on "Sign in"
      Then I should see "Invalid Email or password."
      And I should be on the login page
```

Appendix 1.3

```
Given(/^I am on the home page$/) do
      if User.find_by_email('testuser@test.com').nil?
            User.create!(email: 'testuser@test.com', password: 'password',
            password_confirmation: 'password')
      end
      visit '/'
      page.current_path == '/'
end

Given(/^I have chosen to log in$/) do
      click_on 'Login'
      page.current_path == '/login'
end

When(/^I enter my email$/) do
      fill_in 'Email', with: 'testuser@test.com'
end

When(/^I enter my password$/) do
      fill_in 'Password', with: 'password'
end

When(/^I enter invalid password$/) do
      fill_in 'Password', with: 'password2'
end

Then(/^I should be on the login page$/) do
      page.current_path == '/login'
end

When(/^I enter an email which is not registered yet$/) do
      fill_in 'Email', with: 'testuser1@test.com'
end
```

Appendix 1.4

```
test "should not save duplicate users" do
      user = User.new(email: 'user@test.com', password: 'password',
```

```
      password_confirmation: 'password')
      assert user.save!, "Did not save the user with legal entries"
      user = User.new(email: 'user@test.com', password: 'password',
      password_confirmation: 'password')
      assert_not user.valid?
      assert_raises(ActiveRecord::RecordInvalid) do
            assert_not user.save!, "Saved the duplicate users"
      end
end
```

Appendix 1.5

```
test "user should be able sign_up" do
      get "/users/sign_up"
      assert_equal 200, status
      post "/users", params:{ user: {email: 'user@test.com', password:
      'password', password_confirmation: 'password'} }
      follow_redirect!
      assert_equal 200, status
      assert_equal 'Welcome! You have signed up successfully.',
      flash[:notice]
end
```

Appendix 1.6

- File: lacr-demo/app/controllers/documents_controller.rb
- Implementation:

```
      def upload
         if user_signed_in? and current_user.admin?
            …
         end
      end
```

Appendix 1.7

- File: lacr-demo/app/controllers/documents_controller.rb
- Implementation:

```
if params.has_key?(:transcription_xml)
    xml_files = xml_upload_params
    xml_files['xml'].each do |file|
       …
    end
end
```

Appendix 1.8

- File: lacr-demo/app/uploaders/xml_uploader.rb
- Implementation:

```
def extension_whitelist
  %w(xml)
end
```

```
        def content_type_whitelist
          ['application/xml', 'text/xml']
        end
```

- File: lacr-demo/app/controllers/documents_controller.rb
- Implementation:

```
# Check the XML namespace
nokogiri_obj = Nokogiri::XML(File.open(file.path))
namespace_list = nokogiri_obj.collect_namespaces.values
if (namespace_list.include? TranscriptionXml::HISTEI_NS)
   …
else
   @unsuccesfully_uploaded[:xml].push("HisTEI namespace not found:
#{filename}")
end
```

Appendix 1.9

- File: lacr-demo/app/controllers/documents_controller.rb
- Implementation:

```
# Show message when overwrites
output_message = TranscriptionXml.exists?(filename: filename) ? "Overwritten
#{filename}" : filename
# Create new or find existing record
t = TranscriptionXml.find_or_create_by(filename: filename)
# Store the information about the uploaded file
t.xml = file
# If the save was successful
if t.save!
   @succesfully_uploaded[:xml].push(output_message)
   # Store file content and filename for import to BaseX
   xml_files_content.push([filename,nokogiri_obj.to_xml])
   # Proccess the XML file
   t.histei_split_to_paragraphs
else
   @unsuccesfully_uploaded[:xml].push("Unexpected error on saving:
#{filename}")
end
```

Appendix 2.1

- File: lacr-demo/app/models/transcription_xml.rb
- Implementation:

```
doc = Nokogiri::XML (File.open(xml.current_path))
# Remove Processing Instructions. Tags of type <? .. ?>
doc.xpath('//processing-instruction()').remove
```

Appendix 2.2

- File: lacr-demo/app/models/transcription_xml.rb
- Implementation:

```
# Create empty pages
page_breaks = doc.xpath("//xmlns:pb[@n]", 'xmlns' => HISTEI_NS)
# The volume number is extracted from the xml:id of div
volume = splitEntryID(doc.xpath('//xmlns:div[@xml:id]/@xml:id', 'xmlns' =>
HISTEI_NS)[0])[0]
page_breaks.each do |pb|
   begin
      # The attribute @n is assumed to be the page number
      page = pb.xpath('@n').to_s.to_i

      # The note contains information about the page
      # it content is usually "Page is empty."
      if not Search.exists?(page: page, volume: volume, paragraph: 1)
         s = Search.new
         s.volume = volume
         s.page = page
         s.paragraph = 1
         pr = TrParagraph.new
         pr.content_xml = “<note xmlns=\"#{ HISTEI_NS }" \
                           type=\"editorial\">Page is empty.</note>”
         pr.content_html = "<span class=”xml-tag”>Page is empty.</span>"
         pr.save
         s.tr_paragraph = pr
         s.transcription_xml = self
         s.save
       end
    rescue Exception => e
       logger.error(e)
    end
 end
```

Appendix 2.3

- File: lacr-demo/app/models/transcription_xml.rb
- Implementation:

```
# Extract all "div" tags with atribute "xml:id"
entries = doc.xpath("//xmlns:div[@xml:id]", 'xmlns' => HISTEI_NS)
# Parse each entry as follows
entries.each do |entry|
   …
end
```

Appendix 2.4

- File: lacr-demo/app/models/transcription_xml.rb
- Implementation:

```
# Recursive function to convert the XML format to valid HTML5
  def xml_to_html(tag)
    tag.children().each do |c|
```

```
      # Rename the attributes
      c.keys.each do |k|
        c["data-#{k}"] = c.delete(k)
      end
      # Rename the tag and replace lb with br
      c['class'] = "xml-tag #{c.name.gsub(':', '-')}"
      # To avoid invalid void tags: Use "br" if "lb", otherwise "span"
      c.name = c.name == 'lb' ?  "br" : "span"
      # Use recursion
      xml_to_html(c)
    end
end
```

Appendix 2.5

- File: lacr-demo/app/models/transcription_xml.rb
- Implementation:

```
# Go to the closest parent "div" of the entry and find a child "date"
# and extract the 'when' argument
date_str = entry.xpath("ancestor::xmlns:div[1]//xmlns:date/@when", 'xmlns' =>
HISTEI_NS).to_s

# Fix incomplete date format to make search functionality work
if date_str.split('-').length == 3
   entry_date_incorrect = nil
   entry_date = date_str.to_date
elsif date_str.split('-').length == 2
   entry_date_incorrect = date_str
   entry_date = "#{date_str}-1".to_date # If the day is missing set ot 1-st
elsif date_str.split('-').length == 1
   entry_date_incorrect = date_str
   entry_date = "#{date_str}-1-1".to_date # If the day and month are missing
set ot 1-st Jan.
else
   entry_date_incorrect = 'N/A'
   entry_date = nil # The date is missing
end

# Convert the 'entry' and 'date' Nokogiri objects to Ruby Hashes
   entry_id = entry.xpath("@xml:id").to_s
   entry_lang = entry.xpath("@xml:lang").to_s
   case entry_lang # Fix language standad
   when 'sc'
      entry_lang = 'sco'
   when 'la'
      entry_lang = 'lat'
   when 'nl'
      entry_lang = 'nld'
   end
   entry_xml = entry.to_xml
```

```
   # Replace line-break tag with \n and normalize whitespace
   entry_text =(Nokogiri::XML(entry_xml.gsub('<lb break="yes"/>',
"\n"))).xpath('normalize-space()')
   xml_to_html(entry)
   entry_html = entry.to_xml

   # Split entryID
   volume, page, paragraph = splitEntryID(entry)

   # Overwrite if exists
   if Search.exists?(page: page, volume: volume, paragraph: paragraph)
      s = Search.find_by(entry: entry_id)
      # Get existing paragraph
      pr = s.tr_paragraph
   else
      # Create new search record
      s = Search.new
      s.entry = entry_id
      s.volume = volume
      s.page = page
      s.paragraph = paragraph
      # Create TrParagraph record
      pr = TrParagraph.new
   end

   # Save the new content
   pr.content_xml = entry_xml
   pr.content_html = entry_html
   pr.save

   # Create Search record
   s.tr_paragraph = pr
   s.transcription_xml = self
   s.lang = entry_lang
   s.date = entry_date
   s.date_incorrect = entry_date_incorrect
   s.content = entry_text
   s.save
end
```

Appendix 2.6

- File: lacr-demo/app/controllers/documents_controller.rb
- Implementation:

```
# Upload xml content to BaseX
begin # Catch connection error
   session = BaseXClient::Session.new("xmldb", 1984, "createOnly",
"1a85637cd2ba8c936306c0fa2438b9fcf23dcfa1")
   session.execute('open xmldb') # Open XML database
```

```
   xml_files_content.each do |file_name, file_content|
      begin # Catch document creation error
         session.replace(file_name, file_content)
      rescue Exception => e
         logger.error(e)
      end
   end
session.close
rescue Exception => e
  logger.error(e)
end
#End of uploading to BaseX
```

Appendix 2.7

- File: lacr-demo/app/controllers/documents_controller.rb
- Implementation:

```
      # Generate new index for Elasticsearch
      Search.reindex()
```

Figure 1: Entity-Relationship model

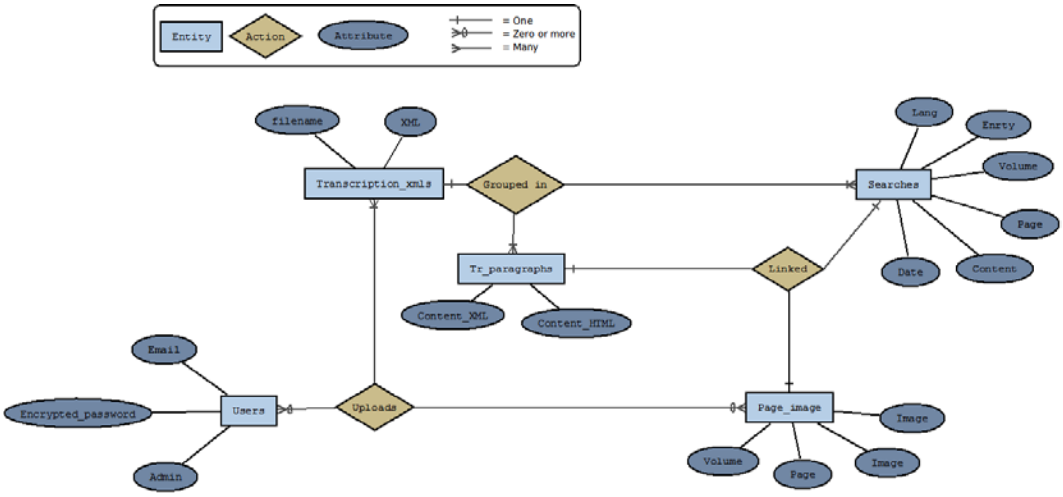


Figure 2: Architecture Diagram

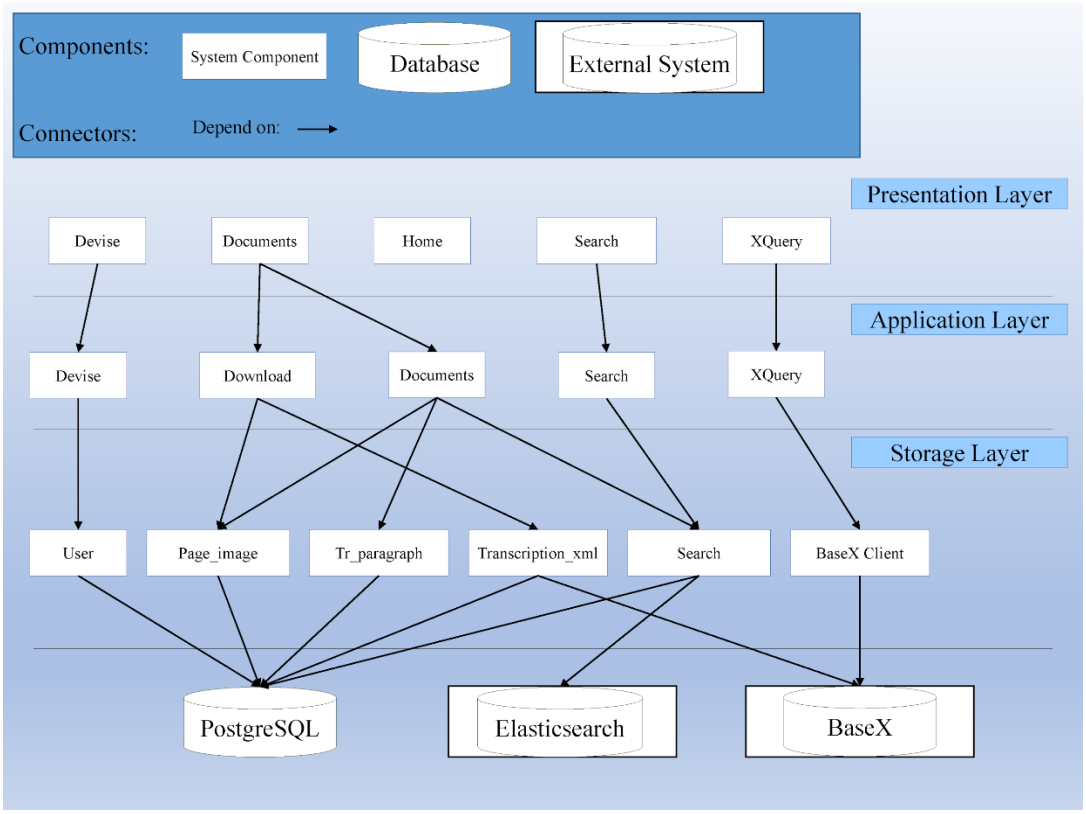


Figure 3: Logstash pipeline architecture

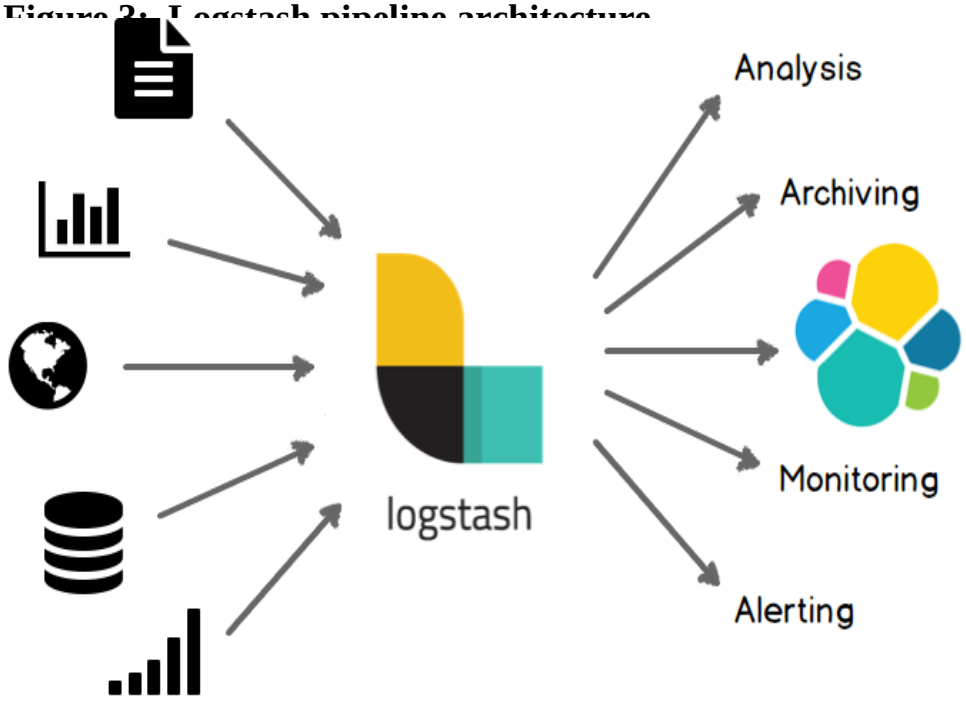
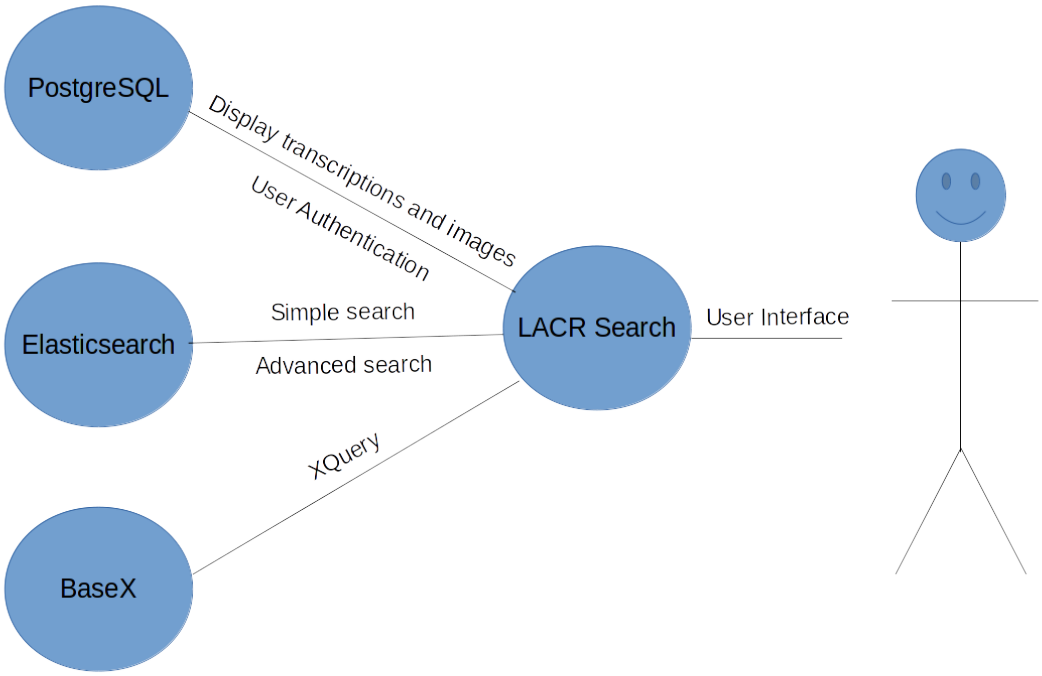


Figure 4: LACR Search architecture



References

[1] BaseX Clients (2017, March): http://docs.basex.org/wiki/Clients
[2] BaseX Documentation (2017, March): http://docs.basex.org/wiki
[3] BaseX User Management (2017, March): http://docs.basex.org/wiki/User_Management
[4] CarrierWave Documentation (2017, March): http://www.rubydoc.info/gems/carrierwave
[5] Elasticsearch Customers (2017, March): https://www.elastic.co/use-cases
[6] GraphML FIle Format (2017, March): http://graphml.graphdrawing.org/
[7] JQuery Zoom. (2017, March): http://www.jacklmoore.com/zoom/
[8] Neo4j Documentation (2017, March): https://neo4j.com/docs/developer-manual/
[9] Nokogiri Documentation (2017, March): http://www.rubydoc.info/github/sparklemotion/nokogiri/
[10] Searchkick Documentation (2017, March): https://github.com/ankane/searchkick#reference
[11] Gate Documentation (2017, March): https://gate.ac.uk/mimir/doc/mimir-guide.pdf
[12] Gate MiMiR and cloud services (2014, June): https://gate.ac.uk/sale/talks/gate-course-jun14/module-4-mimir/module-4-mimir-cloud.pdf
[13] Mecmua. Early Modern Ottoman Culture of Learning. (2017, March): https://mecmua.acdh.oeaw.ac.at/index.html
[14] German text archive. (Deutsches Textarchiv): http://www.deutschestextarchiv.de/
[15] Aggregator, CLARIN's search tool: https://weblicht.sfs.uni-tuebingen.de/Aggregator/
[16] CLARIN: https://www.clarin.eu/
[17] OpenSonar word repository: https://www.clarin.eu/showcase/opensonar/
[18] Fedora Project - developer interview: http://fedoraproject.org/wiki/Interviews/EricSandeen/
[19] A short guide to digital humanities: http://jeffreyschnapp.com/wp-content/uploads/2013/01/D_H_ShortGuide.pdf
[20] Devise documentation (2017, March). Retrieved from GitHub: http://www.rubydoc.info/github/plataformatec/devise/
[21] Rails Guides (2017, March). Official Rails 5.0 documentation: http://guides.rubyonrails.org/
[22] RailsAdmin Gem (2017, March): https://github.com/sferik/rails_admin/
[23] Hunter Powers, "Instant Nokogiri", Packt Publishing Ltd, 2013
[24] Capybara Gem documentation (2017, March). Retrieved from GitHub: https://github.com/teamcapybara/capybara
Cucumber Documentation (2017, March). Retrieved from GitHub: https://github.com/cucumber/cucumber/wiki/ruby-on-rails
[25] Logstash Documentation (2017, March) https://www.elastic.co/guide/en/logstash/current/
[26] Logstash Documentation (2017, March) https://www.elastic.co/guide/en/logstash/current/
[27] BlackLab, Corpus Query Language (2017, March) http://inl.github.io/BlackLab/corpus-query-language.html
[28] TEI, Guidelines (2017, March) http://www.tei-c.org/Vault/P5/1.0.1/doc/tei-p4-doc/html/SG.html
[29] GATE Mímir, BBC News Demo (2017, March) http://demos.gate.ac.uk/mimir/gpd/search/index
[30] GATE Mímir, People in the news Demo (2017, March) http://demos.gate.ac.uk/pin/
[31] DSL, what is DSL? (2017, March) http://dsl.ac.uk/about-dsl/what-is-dsl/
[32] Hunspell, About (2017, March) http://hunspell.github.io/