Chronicling America Parser

Cole Silvernail, Angel Chacon, and Ricardo Pulido

Department of Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

Abstract — The goal for this project is to deliver a functioning ALTO XML parser and a website with a back-end database. The parser needs to be able to separate newspaper pages into separate articles using the information in ALTO XML files. The database for this project must be able to hold all required information pertaining to user accounts, their queries, and the parsed newspaper data. Because of the quantity of newspapers and articles that will be in the database, the query tool needs to be robust, so that the server will not get overwhelmed by fetching too many articles. The query tool needs to be efficient and give the user many options to use in order to find their desired results without being flooded with irrelevant data. Proper indexing of relevant fields should help to decrease lookup time. Results should be chunked as to reduce load to both the server and the client,. but quickly allow navigation to any chunk. The users need to be able to get specific about what they are querying. The website will allow users to save their queries, as well as download a heat map or CSV file of returned query results.

I. Introduction

In order to understand this project, it must first be understood what Chronicling America is. Chronicling America is a project aimed at collecting and digitizing old newspapers that can be indexed and searched online. It provides access to data regarding historic newspapers and some of their digitized pages.

It is produced by the National Digital Newspaper Program (NDNP), which is a partnership between the National Endowment for the Humanities (NEH) and the Library of Congress, and is an effort to create a queryable database of US newspapers. The Library of Congress is dedicated towards developing and forever maintaining this valuable resource. [1]

This project's parser will parse through the newspaper pages in ALTO XML form that can be obtained from Chronicling America. Once they have been parsed they will be input into a database and will be made searchable from a powerful query tool.

It is important to understand that this project is not meant to replace the Chronicling America website. Rather, it will be considered a useful research tool that will work in tandem with Chronicling America. The features of this project's website will act as powerful research and analytic tools for academics, journalists, and other similar researchers. These features include being able to download CSV files, a heat map of results that match a user's query, and others.

II. Broader Impacts

The results of this project will show our sponsors new areas that they can explore to improve the project, such as supporting other languages, handling newspapers from outside the Library of Congress, and/or providing analytics of how the project is used. Being able to obtain historic newspaper information can improve the teaching tools of education faculties across the country.

Journalists and researchers can benefit from a one stop shop location to aid in finding old or obscure articles. A vast amount of new information from decades prior will be available to anyone who uses these tools, which can be analysed to gain a deeper understanding of how it influenced the people and communities who read it, as well as opens the door into big-data research such as language processing of historical and geographical writing styles.

Also, we hope this will turn into a powerful research tool for academics. We hope the new features along with the user interface will make the web app easily accessible and convenient for researchers to gain new insight into the times and issues they are studying.

III. Main Components

This Senior Design project consists of several parts. There are four main parts, and three optional stretch goals. These parts are: the website, the database, the query tool, and the parser. These parts will be connected in different ways, mostly to the database.

A. Parser

The first task is to create a parser that can read and separate sections of a newspaper into articles that will be indexed and stored in a database. Articles will consist of different parts, but the main ones will be their title and content. Exactly how the title and content are determined will be part of the challenge. These articles must also be searchable from a website.

The parser will be connected to the database. It will take in newspaper pages and insert the relevant data to store with the article into the database. This process will also be batched to speed up the process. This will be a completely

isolated task between the parser and the database server, and so there should not be any user-facing issues.

B. Query Tool

The second task is augmenting and building the search engine. One of the requirements is to add more sophisticated search features. This includes AND, OR, and NOT operators along with proximity matching, search using metadata, and explicit string search, along with the ability to combine these in different ways. The user must also have the ability, if they choose, to save these queries to the their account so they can be conveniently used or referenced later.

The query tool will be connected to the database by an inverted index. It will get a user query from the website, fill in any information that is necessary, search for the keywords in the inverted index, and find which articles contain those keywords. Next, the results will be ranked using the user query specifications and then returned to the user and displayed on the website.

C. Database

The third task is to create a database for storing the data for users and articles. As mentioned in the first task, the articles table will store the title and content, along with some metadata. The user table will store the user's username and/or email, password, and their search queries, as mentioned in the previous task.

D. Website

The fourth and last task is to design a website that allows users to register an account, save their search queries, and view a heatmap for various articles based on the location of the newspaper they come from.

The website will be connected to the database and the query tool. When logging in or registering, the website will have to look up accounts, create new accounts, or access/edit an account's information from the database. Access to the query tool is necessary for the website in order for the user to search for articles, and to receive all the content and metadata for each article.

IV. Website Specifications

A. Front End - ReactJs

React is an increasingly popular JavaScript library for building wonderfully complex user interfaces. Originally developed by Facebook and open-sourced to the community, React employs the following ideals that will aid in our project:

- **Declarative:** React makes it painless to create interactive UIs. We can design simple views for each state in our application and React will efficiently update and render just the right components when your data changes. Declarative views make your code more predictable, simpler to understand, and easier to debug.
- **Component-Based:** By building encapsulated components that manage their own state, we can compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, we can easily pass rich data through the app and keep state out of the DOM.
- **Learn Once, Write Anywhere:** React does not make assumptions about the rest of the technology stack, so we can develop new features in React without having to rewrite existing code.

We will use ReactJS to render all HTML to the client.

B. Back End - Laravel

Laravel is an open sourced cross platform web application framework that aids in the creation of elegant websites through expressive and creative code. Laravel attempts to handle the workload of common tasks to ease development of projects. Laravel comes with an extensive documentation set and video series that allows fast learning of the framework. Laravel's features include:

- Simple, fast routing engine
- Powerful dependency injection container
- Multiple back-ends for session and cache storage
- Expressive, intuitive database ORM
- Database agnostic schema migrations
- Robust background job processing
- Real-time event broadcasting

We will utilize Laravel only as a Restful API. The routing will be handled by React router. Rather than serving HTML or PHP views, we plan to use Laravel to deliver powerful JSON APIs

C. Authentication

We will use JSON Web Token (JWT) for user authentication. Laravel Restful API will generate the

token and send it back to the user. Laravel will verify the JWT when a user makes an API request.

V. Query Tool

Part of our project requirements is a query tool that allows boolean operators, quoted phrases, and other common query inputs. This query tool also needs to work without crashing our server or taking too long to return results.

The query tool will be searching through a database of thousands of articles. Because of this, we need to use a robust query tool that will weed out less relevant articles and will push the most relevant and useful articles to the top of the returned results. Without a formidable query tool it is possible a query input could return every single article in the database. This would not only make the results useless, but could also crash the server.

We can learn a lot about how an efficient search tool works from Google. Without search engines like Google it would be near impossible to parse through all the information on the internet and find what you are looking for. Google is a very popular web search tool because it has a powerful query tool, and it does a good job at returning relevant websites to the user.

This project also needs a query tool that will return relevant articles quickly, therefore it may be useful to know how Google's search tool and algorithm works. Google does not share everything about how their search algorithm works, but there are certain parts of it that are known.

A. Apache Solr

Solr is a popular search platform used by some of the most popular websites on the internet. Some of the websites and applications that use Solr are Netflix, Instagram, DuckDuckGo, and eBay [2]. First created in 2004, It is a powerful tool that will be able to fulfill all the needs of the query tool for this project. It is also open sourced, so no license is needed to use this platform for this project [2]. It was created using Lucene, which is a java-based library that is used primarily for building search tools [3].

One of the reasons that Solr is so popular is that it has been proven to be able to handle high traffic volumes and has many useful features [2]. Some of these features would be ideal for this project including inverted indexing and a plethora of potential search options.

B. Inverted Indexes

Solr uses inverted indexing in order to make searching large databases much faster [4]. An inverted index is a data structure that includes a mapping of a database's contents and where those contents are located within it [5].

In the context of this project an inverted index would have a list of each unique word within the article and then would note each article where that word was used. If an inverted index is used in this way a query would no longer need to search the text of every article in the database, and instead would be able to only search by the keywords inputted by the user. This results in a faster and more efficient way of searching a large database with a substantial amount of text in each record. This is particularly beneficial to this project because of the potential for an extensive amount of text in each newspaper article that will be within the database.

C. Results Ranking

Once a search is performed and all the relevant articles are gathered, Solr will then rank the results in a way similar to Google's and many other search tools' ranking systems. Solr uses an organized set of considerations to decide how to rank the relevancy of each document.

First, it uses the frequency of each inputted keyword to determine ranking, which is a simple and commonly used method [4]. It will also use inverse document frequency. This means that the less a unique word is used across the database, the more that word can potentially increase an article's relevancy ranking [4]. On top of these, a coordination factor is considered. This will take into account how many different search terms are found in an article, causing the articles with more keywords to rise in rankings [4]. Lastly, Solr penalizes articles that contain more words than average, meaning that the probability that the article will include a search term is more than average. This is called considering the field norm [4].

These are the basic variables Solr takes into account when ranking search results. It becomes more complicated when other search options are added to the query. For example, search term boosting can be used.

D. Basic Search Options

The Solr search options are plentiful and diverse. For this project there are some basic search options that are needed, and would be included by default on any well developed search tool. The Solr search platform provides all these search options including:

1) Boolean Operators - AND, OR, NOT
2) Search by Phrase/Direct Quote
3) Proximity Matching
4) Searching by Metadata
5) Wildcard Searches

Solr searching is also customizable, making it possible to specify which fields to use so that we can search by every metadata option that is included in the database [2].

E. Our Query Tool vs. Chronicling America's Query Tool

For this project it is important to have a better query tool than Chronicling America's query tool. For that reason it needs to be seen how this project's query tool plan stacks up to Chronicling America's existing query tool. If this project's query tool is found lacking when compared to Chronicling America's tool, then a new plan needs to be made in order to ensure this project's quality and practicality.

Let us start by comparing the boolean operators that will be available for use in each query tool. As you can see in the table below, both Chronicling America's search tool and this project's search tool with both have ways of using the AND and OR boolean operators.

The difference between the two tools in this case is that this project's search tool will make the NOT operator available to the user, while Chronicling America's tool does not [6]. In this category, this project's search tool has the upperhand in a big way. The NOT operator is an essential tool for many search tools, including Google's search tool. It offers the opportunity in the ranking of results to penalize articles for containing specific search words.

The next issue being used to compare the two query tools is searching by metadata. Both query tools will be able to search by state, however only this project's query tool will be able to search by city. Both query tools will be able to search by the name of the newspaper that published the article, year range, and date range.

The major advantage that this project's query tool will have over Chronicling America's query tool is that it will be able to search by words in articles instead of pages. Chronicling America does not have its newspapers separated into articles, while the main objective of this project is to parse through the newspapers and distinguish between articles. By separating pages into articles smaller portions of text can be analyzed for relevance by our query tool. Overall the query tools are pretty evenly matched in this category.

F. How the Query Tool Will Act

Now that many critical parts of the query tool have been researched and decided on, the plan for how the query tool will actually work can be laid out. Even though Solr will be handling many functions within the query tool, the process of querying search words needs to be planned.

First, when a user begins using the query tool they will need to decide on whether they will want to use a basic search or an advanced search. Once they choose this, they will enter their search terms and criteria, including the metadata that will narrow down results.

Once the user clicks the "search" button their query will be lightly translated into a form Solr understands by adding fields and necessary operators. Once it is translated, it is sent to Solr to parse through the query text.

Next Solr will search for the search terms in the inverted index. The inverted index will tell Solr which articles in the database contain the search terms. Once Solr retrieves the full article content and its information, it will analyze them in order to rank the articles by relevance using Solr's own ranking system and the criteria the user entered. Once the articles have been ranked, their information will be returned to the user allowing them to choose which article they wish to look through.

VI. Parser

The article parser is one of the most important aspects of this project. The parser will take the ALTO XML file generated by the Chronicling America website, read the data in the file, separate it into articles, and insert those articles into the database.

A. The ALTO format

ALTO is an XML file format defined and maintained by the Library of Congress. In this file describes what a newspaper page looks like, the text on the page (through the use of character recognition software), where the text is, what fonts and styles it uses, etc. This data will be read by the parser and interpreted in a way that allows for splitting it into discrete articles with a title and content.

The text data is separated into TextBlock elements, which consist of vertically stacked TextLine elements. Those TextLine elements can then consist of three different elements: String for text, SP for whitespace, and HYP for words that are hyphenated and split along two lines. Separate from that, there are TextStyle elements that describe the text size, font, and its decorations such as whether it's bold or underlined. All this information can be used by the parser to differentiate between articles.

B. Reading the File

Because the ALTO files served by Chronicling America can be hundreds of kilobytes in size, of which only some of the data gets used, we needed a way to quickly read the information we needed without using up too much memory.

There different types of XML file parsers:

- DOM parser

- SAX parser
- StAX parser

A DOM parser loads the entire document into memory and constructs a DOM tree in-memory with all the data stored in the file. This tree can then be accessed to get the information we need.

A SAX parser loads the document portion by portion. As it reads the file, it fires off events that include data about parts of the document, such as the start of an element, text content, or the end of an element. These events can then be read for relevant information which we then store ourselves.

A StAX parser only loads the relevant parts of the document when specifically requested to do so, and is considered to be much simpler in general.

Each one of these parsers has their benefits and drawbacks. However, for our use case, we determined that the SAX parser was the best choice. It reads the document all at once and allows us to keep only the necessary data while saving memory and processing time.

C. Processing the Articles

Articles consists of a single title and some textual content (along with some other metadata stored with it in the database).

In our testing, we've found that a significant majority of ALTO files contain well-organized TextBlock elements which contain whole articles. Rather than mess that up, we've decided to preserve the original structure of the text and only perform minor modifications where they won't introduce any problems. We use the first line of the TextBlock to be the title of the article, and the remaining lines are stored as the content of said article.

After all the articles are parsed locally, we open a database connection to our database and store the data. It is estimated that to store all the articles, about 140 gigabytes of space will be needed for the database. Also, after several optimizations, the estimated amount of time for parsing all pages is almost 4 years.

VII. Conclusion

Once this project is completed it will certainly be a powerful and efficient research tool for academics who want to use the information from Chronicling America. Using the several complex parts of this project, a research tool can be constructed. It is important to remember that this project's purpose is not to replace the Chronicling America website, but instead is meant to build an add-on tool that makes the information contained within the Chronicling America website more easily analyzed by researchers.

By using the parser designed and built during this project, it will be possible to break each page contained within the Chronicling America website into articles in a relatively accurate manner. This allows the information within the pages to be queried for more easily, and ensures queries will better reflect the data within the texts of these newspapers.

Once the pages are broken down into articles they can be easily analyzed using various programming tools because users will be able to access query results in CSV files. On top of the CSV files, users will be able to visually see the geographic locations that query results originate from using this project's query heatmap that can be downloaded as a PNG file. These are important fragments of this project, because they further cement this project as a useful tool for serious academic research.

This project's query tool will also make slight enhancements to Chronicling America's query tool. First, because the pages will be separated into articles before being entered into the database and inverted index, our query tool will be allowed to analyze smaller portions of text for relevancy to inputted search terms. This should make queries more accurate and relevant to what users want. This project's query tool will also allow users to save their searches, allowing them to refer back to them while conducting their research. By using an inverted index and a prudent relevancy ranking system, search results will be returned quickly and will be ordered in a way that users will find convenient and useful. Also, this query tool will allow users to narrow down their results using a generous amount of search options, further ensuring the relevancy of the results that will be returned to them. This query tool will be put together using Solr which will allow it to run smoothly and efficiently on any server it will be run on.

This project's website is also organized and built in a logical manner that will make researching easier and more pleasant for users. It is also responsive, meaning that it will adapt to the user's screen size. This allows users to access the website on their computers, phones, or tablets and still have a similar experience without having to zoom in on parts of the website, making their experience more convenient and pleasurable.

The three main portions of this project are the parser, the website, and the query tool. On their own, they do not amount to a useful project, but when they are united they create a sum larger than their individual parts. By working in sync with one another they create a comprehensive and effective research tool for users wanting to analyze historical data.

References

[1] “Chronicling America - About.” Chronicling America, chroniclingamerica.loc.gov/about/

[2] Apache Software Foundation. https://lucene.apache.org/solr/

[3] Tutorials Point. Lucene Tutorial. https://www.tutorialspoint.com/lucene/

[4] Tan, Kelvin. 2018. http://www.solrtutorial.com

[5] Jain, Saurav. Geeks for Geeks. Inverted Index. https://www.geeksforgeeks.org/inverted-index/

[6] The Library of Congress, Chronicling America: Historic American Newspapers site. https://chroniclingamerica.loc.gov/help/