# IDEAL Spreadsheet Final Report

CS 4624 - Hypertext and Multimedia
Professor: Edward Fox

Client: Mohamed Magdy, Phd. Student

Group Members:
Anthony Ardura, Austin Burnett, Shawn Neuman, Rex Lacy

# Table of Contents

# Abstract

The IDEAL proposal encompasses an incredibly vast infrastructure of technology intended to be used by people of varying backgrounds. The analysts and researchers who will be familiar with the data presented through many aspects of the IDEAL project may not be familiar with the means of accessing it from the differing resources. The purpose of this project is to provide non technically-skilled personnel with the ability to access data in a easy to use and intuitive way.

The data this project focuses on are tweets, photos, and webpages found on web-archive files, or ‘warc’ files. These warc files are comprised of a few, to several hundreds of gigabytes, making a manual search to find specific information near impossible. Instead, we use a Cloudera VM as a prototype of the cluster used in IDEAL, and demonstrate how to load WARC files for Hadoop processing. That allows parallel big data processing with several software tools, supporting database and full-text searching, text extraction, and various machine learning applications.

Our project goal to present relevant data in an attractive, useful, and intuitive way was achieved through the creation of a web based spreadsheet-like service. While the exact use goes on in greater detail below, the overarching plan was to provide the user with an easy to use spreadsheet, which takes input from the user and returns the relevant data in spreadsheet cells. The other functionality requested by the client for special jobs such as ‘all images’ or ‘word count’ led to other features.

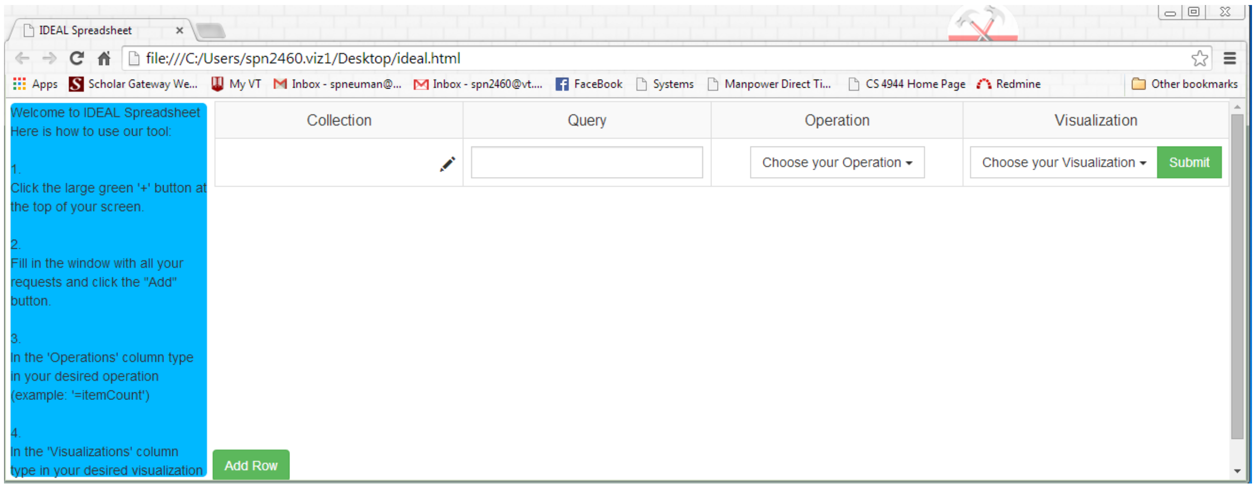
To summarize, this project intends to provide a web service to provide IDEAL researchers with the means to retrieve relevant information from warc files in an intuitive and effective manner. The project called for several technologies and frameworks which will be elaborated on below, and this project paves the way for increased future development in the IDEAL project mission.

# User Manual

The IDEAL Spreadsheet interface is designed to be an html style page that the user can access from anywhere.  The user will be able to choose from existing web archive files and query these files for relevant information. Additionally, the user will be able to perform different queries on the same file, similar queries on multiple files, or perform multiple queries on multiple files. This backbone version is only designed to support the word count operation and show the results in word cloud format.

All of the searchable web archive files are categorized by type of disaster. To expedite the search process, the user will first select a type of event, such as “Fire”, “Shooting”, or “Hurricane”. This allows our program to filter out those events that do not match which significantly reduces the possibility that the user will become overwhelmed with the sheer volume of events that can be chosen from. Once the choice has been made, a new drop down menu appears, that is populated with only those events that fall under the chosen category.

First, the user should add a row to the table by pressing the green button labeled “Add Row”. This will add a row directly underneath the header row with a section for adding to the collection to be searched, a section for adding a specific query to search for, a section to apply supported operations to the search collection, and a section to select the type of visualization for the results. This figure shows the results of pressing “Add Row” one time.



Welcome to IDEAL Spreadsheet
Here is how to use our tool:

1.
Click the large green '+' button at the top of your screen.

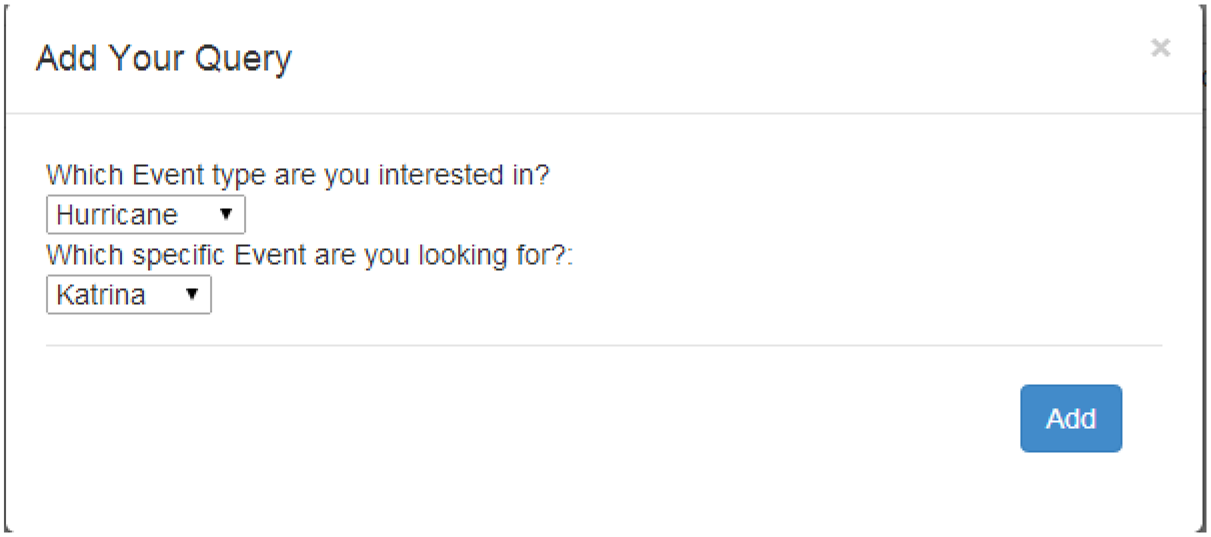
2.
Fill in the window with all your requests and click the "Add" button.

3.
In the 'Operations' column type in your desired operation (example: '=itemCount')

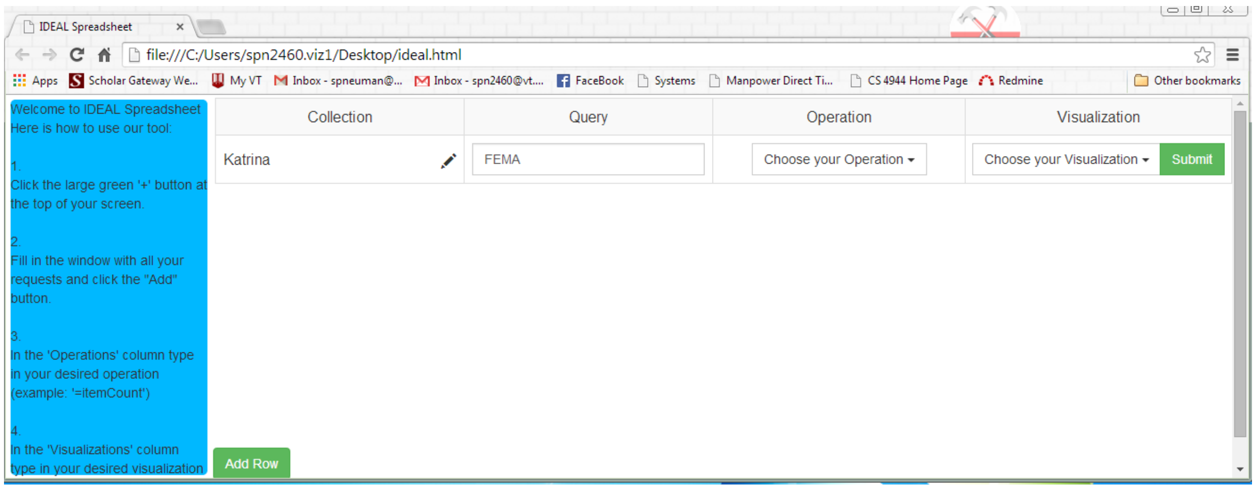
4.
In the 'Visualizations' column type in your desired visualization

| Collection | Query | Operation | Visualization |
|---|---|---|---|
| | | Choose your Operation | Choose your Visualization Submit |

Clicking on the pencil icon in the box located in the “Collection” column will cause a dialog box to appear. This is where the user will first select an event type, and then select the specific event to be queried. This will ultimately return a large list of files that match the criteria. Currently, we have hard coded some values to demonstrate the usability of this tool. The figure below shows the result of having chosen “Hurricane” from the “Event Type” drop down menu, and “Katrina” from the “Specific Event” drop down menu.  Note, the user must click the "add" button to bring this selection into the collection box.


Add Your Query
×
Which Event type are you interested in?
Hurricane ▼
Which specific Event are you looking for?:
Katrina ▼
Add

Now we see that "Katrina" has been added to our "Collections" and we can begin to query all collections related to Hurricane Katrina.  Next we will refine our query by providing a keyword to search for.  We entered FEMA in an effort to find articles related to FEMA's involvement in the wake of this disaster.



Welcome to IDEAL Spreadsheet
Here is how to use our tool:

1.
Click the large green '+' button at the top of your screen.

2.
Fill in the window with all your requests and click the "Add" button.

3.
In the 'Operations' column type in your desired operation (example: '=itemCount')

4.
In the 'Visualizations' column type in your desired visualization

| Collection | Query | Operation | Visualization |
|---|---|---|---|
| Katrina | FEMA | Choose your Operation | Choose your Visualization |

The user can now choose the type of operation to perform on this particular query.  This version only supports the operation "word count" which will return the number of times the word "FEMA" appears in documents relating to Hurricane Katrina.  The user can also choose the way in which this information in visualized. Currently, word cloud is the only supported visualization strategy. After both the operation and visualization strategy have been chosen, simply hit the “Submit” button and the results will appear.

Future versions of this tool will add additional supported operations. One important operation that needs to be supported is “file count”. This feature should return the number of files that meet the search criteria. This will help the researcher get a better idea of how many relevant hits there are. It will also aid the researcher in refining or modifying the search criteria in order to return a more manageable list. A “Filter” button should also be added. this should allow the user to enter certain keywords that may carry more weight in the search process. A “help” button should also be added. This feature will give the user some assistance in how the tool works, along with some helpful hints on using SOLR based searching.

Network based visualization may be of benefit for the researcher to see how different elements of the results are connected. Dr. Polys, graduate student Peter Radics, and undergraduate researcher Shawn P Neuman, have developed a tool for applying three dimensional layouts to network ontologies. This tool could be integrated as one of the potential visualization methods and may be of some significant value.

# Developer’s Manual

To understand the current development that has been done in order to further contribute to the build out of this tool, there are a few core concepts:

Working with WARC Files

Hadoop Configuration

Hadoop Jobs

Spring Framework

Front End Development

Workflow

These will be covered in depth in the next sections to make this project readily available to be extended.

## SECTION 1: WORKING WITH WARC FILES

The IDEAL Project has done an excellent job in collecting a plethora of event data in the form of web archives. The accessibility and availability of the Internet has contributed to the sheer size of these web archives.

The data provided by the IDEAL Project is in the form of a .WARC file, the Web ARChive file format. This format was developed in light of web crawlers and provides a standard for collecting various forms of content that are encountered on the Internet (WARC, Web ARChive file format). These files contain all of the content from a given domain portrayed as WARC records (WARC, Web ARChive file format). While there are eight types of WARC records, the one we’re interested in for this project is the response type (The WARC File Format). This signifies a full HTTP response and provides the entire content of that request.

As I mentioned before, these WARC files contain WARC records of eight different types of which one we have interest in. In order to focus on the data that we want, we must extract and filter these files to get the underlying text.

While it is not discouraged to develop your own tool to process these files and associated records, there are some available tools to aid with this extraction. We used Hanzo WARC Tools which includes a tool for unpacking WARC files by extracting all of the HTTP responses and exporting them into a file structure with an associated index file with information about each file. In addition to this, some of the other groups from the Spring 2014 edition CS4624 Multimedia, Hypertext and Information Access developed some more fully developed tools using this package.

While extracting the HTML files is the first and most important part of extracting these records, HTML files contain information that we both do and don't want. This project focused mainly on text, so we had to extract the raw text from these files and remove all of the tags from the document. Again, there are many tools that do this. While we used html2text, some of the other groups used Beautiful Soup, both of which excel at HTML parsing. Beautiful Soup has more tools that can be used to fully parse a document and extract information from certain tags to be used further. An example of this would be looking at links to other pages contained in the document or images contained or linked to in the document. Please reference Figure 1.1 for the number of .warc files we used for development.

After we have processed and extracted the necessary information from these WARC files, we can continue to actually look at this information to provide the user with some information about these collections.

| **# of .warc files processed** | **size of .warc files** | **# of HTML files collected** | **size of HTML files collected** | **size of text files collected** |
|---|---|---|---|---|
| 70 | 5.6 GB | 13,421 | 3.7 GB | 3.1 GB |

**Figure 1.1**: Metrics from our .warc file consumption

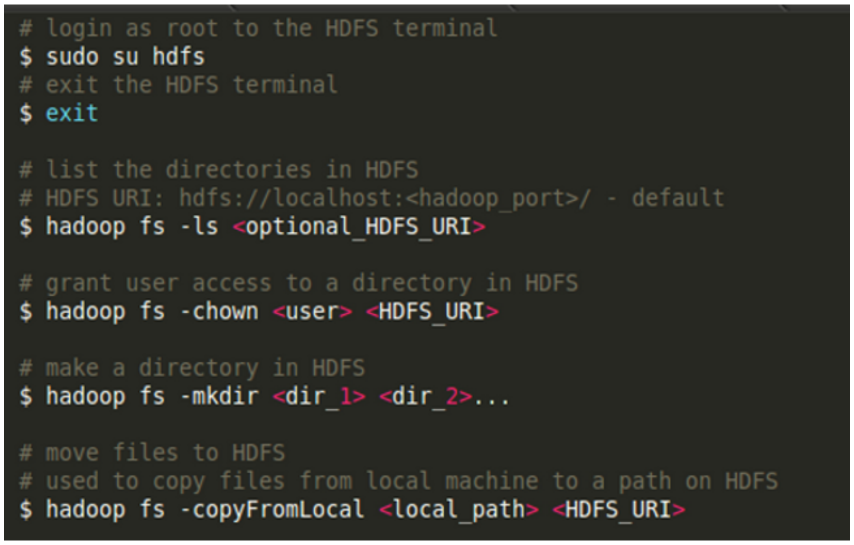
## SECTION 2: HADOOP CONFIGURATION

With the amount of data provided from these archives, we need a tool to assist in the processing and extraction of data from these collections.

Hadoop is a piece of open-source software that provides, “reliable, scalable, distributed computing” (Apache Hadoop). To put that in layman's terms, Hadoop allows us to process large amounts of data in efficient ways. The core pieces of Hadoop that were used in this project were HDFS, or Hadoop Distributed File System, and Hadoop MapReduce. HDFS is the core of any Hadoop application. It is really just a file system, like the one on your personal computer, but “is well-suited for distributed storage and distributed processing” (Apache Hadoop). For this section, we will discuss more in depth about HDFS and its configuration for this project. MapReduce will be touched on in the Hadoop Jobs section.

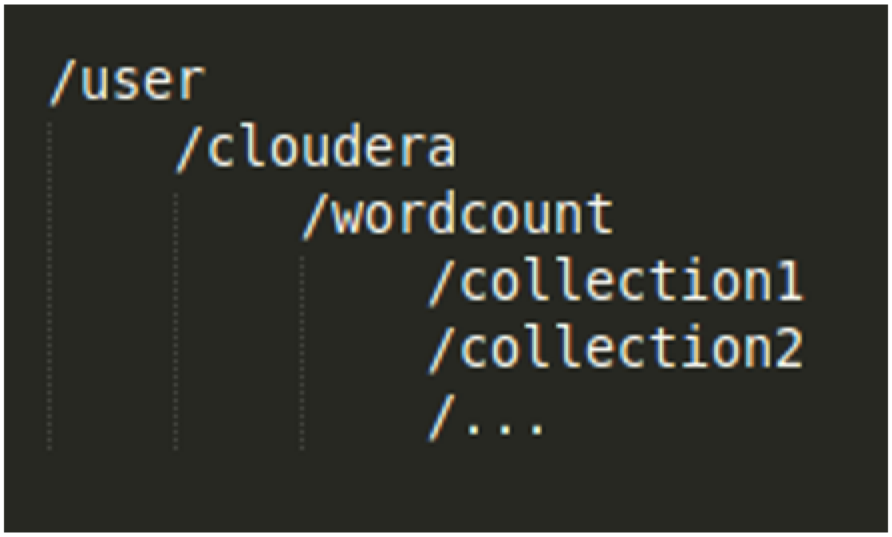
Our project relies on HDFS to process the web archives. For the remainder of this section, I will assume that Hadoop has been configured on your machine. For more information on how to set up Hadoop, you can reference the Single Node Tutorial provided by the Apache Hadoop Wiki. If you’re not interested in installing Hadoop on your machine, reference the Workflow section of this document for more information on ways to do this.

Further assuming that you have extracted some WARC files, we must get this data into HDFS to be processed. The Hadoop Wiki provides a File System Shell Guide with all the information you need to interact with HDFS. For those familiar with a UNIX command line interface, many of these commands will seem familiar or provide similar functionality. As you can see below in Figure 2.1, I have provided a list of the commands we used and their purpose for your convenience. In Figure 2.2, you can see the file structure we used, which can be modified according to corresponding instructions in the next section.



```
# login as root to the HDFS terminal
$ sudo su hdfs
# exit the HDFS terminal
$ exit

# list the directories in HDFS
# HDFS URI: hdfs://localhost:<hadoop_port>/ - default
$ hadoop fs -ls <optional_HDFS_URI>

# grant user access to a directory in HDFS
$ hadoop fs -chown <user> <HDFS_URI>

# make a directory in HDFS
$ hadoop fs -mkdir <dir_1> <dir_2>...

# move files to HDFS
# used to copy files from local machine to a path on HDFS
$ hadoop fs -copyFromLocal <local_path> <HDFS_URI>
```

**Figure 2.1:** A list of HDFS shell commands



```
/user
    /cloudera
        /wordcount
            /collection1
            /collection2
            /...
```

**Figure 2.2:** The filesystem structure we used

## SECTION 3: HADOOP JOBS

To process the data, we use MapReduce Jobs to parse through it in a distributed fashion. the distribution is a combination of the basis of HDFS and the concept of MapReduce. These MapReduce jobs consist of both a mapping and reduce function. The map function takes key/value pairs to process intermediate key/value pairs that will be merged by the reduce function (Dean & Ghemawat).

An example of this can be seen in the `Jobs.java, WordcountMapper.java,` and `WordcountReducer.java` classes. In the `Jobs.java` file, you will notice that at the top, we have the main location for our `wordcount` directory that has all the subdirectories of our collections. From our experience, you have to stay one level above the directory that you're interested in running jobs on. For instance, when we specified the `HDFS_DIR` to be `hdfs://localhost/user/cloudera/wordcount/`, it, for no apparent reason, chopped off `wordcount/`. This is why on lines 50, 51, and 57 we had to specify that directory as an extension.

In the `WordcountMapper.java` file, we can examine the map function to our WordCount functionality. As you can see in the `map` function, we receive a key/value pair along with a context. The context keeps track of our intermediate key/value pairs. The value we receive is the line in a file. While we have more tokens within that line, we add to the context the word, removing some common punctuation, and note that we saw an occurrence.

In the `WordcountReducer.java` file, we see how we take these intermediate key/value pairs and merge them to one. Each key has associated values with it that denote an occurrence. The for-loop sums these occurrences and writes to the shared context the key, or word, and the total occurrences. As you can see, this job tallies the frequency of the words in a dataset.

Further filtering could be applied in the map function to remove more punctuation, skip common words or only focus on certain driven topics.

To further expand on the `Jobs.java` file now that we have a better understanding of the MapReduce paradigm, we can talk about how to run wordcount. In the file, you'll notice the `wordcount` function. The `id` parameter is the collection id provided in the `event_meta.tsv` file. We stored all collections by id so that they can be easily referenced. In this function, we set the input to be the collection directory and specify an output for this as well. Once the job finishes on line 55, we read the results. By convention, jobs specify the results by a `part-r-00000` file containing all words and their frequency in the set. These results are mapped so that they can be easily transformed to JSON for our response to the client-side webpage.

The other function listed, `writeToHDFS`, is for SOLR integration. This function allows you to accept a `String` response and write it out to a file. This is necessary because Hadoop jobs need a file on HDFS to read. This function isn't used because we did not get this far and therefore doesn't have very useful information in it. The overall functionality is easily demonstrated and can be further expounded upon to expose this functionality.

## SECTION 4: SPRING FRAMEWORK

To provide an API to interact with our collection, we used the Spring Framework, a Java framework that provides data access to our underlying data on a server. For our sake, it provides endpoints to connect with via AJAX requests to access the output from our Hadoop jobs. The syntax is very easy to understand with a little knowledge of how HTTP requests work.

Looking at the `WordcountController.java` class, you can see just how this works. Each function has an associate URL ending that gets concatenated onto the origin URL and an associated method that corresponds directly with HTTP request methods. By passing in an `HTTPServletRequest` to our function, we have access to the request headers and parameters. In the case where we run the wordcount job, we extract the id from the request to process and return the job for that collection. We return a mapping of the results which is easily handled and converted to JSON format. This map is an instance of `WordcountMap.java`, which is just an abstraction around a HashMap and could be extended for more functionality. This class also allows to POST. This was to allow from the output from SOLR to be passed back server side and processed, but this was not fully implemented.

The `CollectionsController.java` class provides an API endpoint to access our collections provided in `event_meta.tsv`. One problem we encountered was how to keep up-to-date references to the available collections easily. Maintaining this file provides its own complexities and maintenance, but so does hard-coding values in the front-end. We decided to hard-code, but this functionality is provided in case the future developers would rather use this. This builds a collection of `Disaster`'s, which contains the id, name, type date and location of the disaster, all available in the `event_meta.tsv` file.

The `HomeController.java` class hosts our home page. The only reason we kept this is that for local development, you cannot hit out http://localhost:8080 endpoints from a local HTML file. This causes a Cross-Origin Access error which directly conflicts with the Same-origin policy. We hope that if these are both migrated to a server, this will resolve the conflict and allow for modularity between the HTML file and Spring services.

## SECTION 5: FRONT-END DEVELOPMENT

Front-end development applies to the HTML, CSS, and Javascript that helps drive the user interface presented to the user. Relating to the same-origin policy aforementioned, we actually have two near-identical files - `home.html` and `home.jsp`. I say near-identical because their references to the CSS are different based on location of the corresponding CSS file.

The structure of these files represents the Javascript dependencies at the top followed by our script for the Javascript we wrote. In addition, we have references to CSS dependencies and our CSS. Following this is our HTML. Our dependencies include Bootstrap, a CSS library, and a Javascript library, which assists with the aesthetics of the site and provides more functionality, like our modals. We also use jQuery, a popular library that makes DOM access much easier and provides easy syntax for AJAX requests and event handling. We also use D3.js, a library that creates data models by taking data as input and does some nifty crafting with SVG to represent visual representations of the data. We specifically use D3.js to construct our word cloud visualization.

In our HTML/JSP files we provide a fixed interface at first with a sidebar, a table and an "Add Row" button. Upon clicking "Add Row", `onAddRowClicked()` is called. This appends a row to the table with ids and attributes that correspond to the row number, which allows us to use CSS selectors based on the row that is clicked to get the necessary data. Now, you'll see a blank row with cells to add a collection, query, operation and visualization.

Upon clicking the collection pencil, a modal appears for you to select your collection. Ideally, future work involves resetting the modal each time so that the event selection is fresh each time you go to add a new collection. Query is not yet hooked up, but there is a naive check on line 174 to handle the query and POST the data as described earlier. The only operations supported in the dropdown for Operation and Visualization are Word Count and Word Cloud respectively.

Upon clicking 'Submit' for the visualization, the handler is called on line 166. The initial if-statement assures that a collection is selected and there is an associated operation and visualization. If all requirements are met, we issue an AJAX call for the word count of that collection. Upon finishing the request, this response object is parsed and transformed into the format that D3.js requires. This is a JSON object with both a text and size field. Upon finishing this translation and also applying some filters on common words, we create a modal that is associated with this row. This is because rendering the word cloud in the table requires this to be smaller and it distorts the table beyond what we thought users would find practical. After creating the modal, we pass the word count to D3.js to handle and append to the modal. Additionally, we append a link to the visualization box that corresponds to opening the associated modal with the word cloud.

The most confusing part about this file is the event convention that we used. Due to having multiple developers working on this and not specifying a given convention, our event handlers differ throughout the file. For instance, some elements have an onclick attribute that corresponds to a function in our script while others have jQuery handlers.

To further develop the front-end of this project, it would be helpful to become familiar with Javascript in general. Further, it would be helpful to become familiar with jQuery, Bootstrap, and D3.js. All of these libraries are well-respected in the web development communities and have extensive APIs and tutorials available. While I won't go into the details of our CSS, it is helpful to understand the style properties associated with the elements and understand what they do. Good resources for this are CSS - W3Schools and CSS-Tricks. All of the CSS is contained in `home.css`.

## SECTION 6: WORKFLOW

In this section, I'll describe some of the tools that we used in our development that made things go by much more smoothly and will allow things to work out-of-the-box.

The first tool that we used is the Cloudera Quickstart Virtual Machine. This provides a working virtual machine with all of the tools you need to get started. Cloudera provides some tutorials to learn how to interact with HDFS as well. Using these commands along with the ones provided earlier, you can easily load your collections to HDFS to be readily available.

The Spring Framework provides a customized version of Eclipse to provide a local server mainly. Spring Tool Suite provides Pivotal tc Servers and Maven integration which allow you to import our project, right-click on the project to select "Run On Server…", and see everything work right in the browser.

In addition to these tools Sublime Text is a convenient text editor for HTML, CSS and Javascript. There are many community plugins available to further aid in this. This editor is completely optional but can be customized to be an incredibly useful tool.

## SECTION 7: DEPLOYMENT

To deploy our application on the cluster, there are two simple steps in doing so. First, we must integrate the HTML/CSS and then deploy the Spring project.

To deploy the HTML and CSS, you can simply add `home.html` and `home.css` to your existing interface. If you are using some sort of templating, make sure that you add our dependencies: jQuery, D3.js, and Bootstrap to your page's `head` section. Additionally, it may be a good idea to export our `script` that we wrote that is included in the `head` section of `home.html` to an external script. This would potentially be more preferential.

To deploy the Spring project, you must first update some of the references in the project. As mentioned before, you must configure the `HDFS_DIR` variable in the `Jobs.java` class to reflect the location of your collections. If you have a different naming convention for the collection directories than using the ids as we did, you'll have to update the static collection arrays in the `script` of the `home.jsp/home.html` files. After this, you're ready to deploy the project.

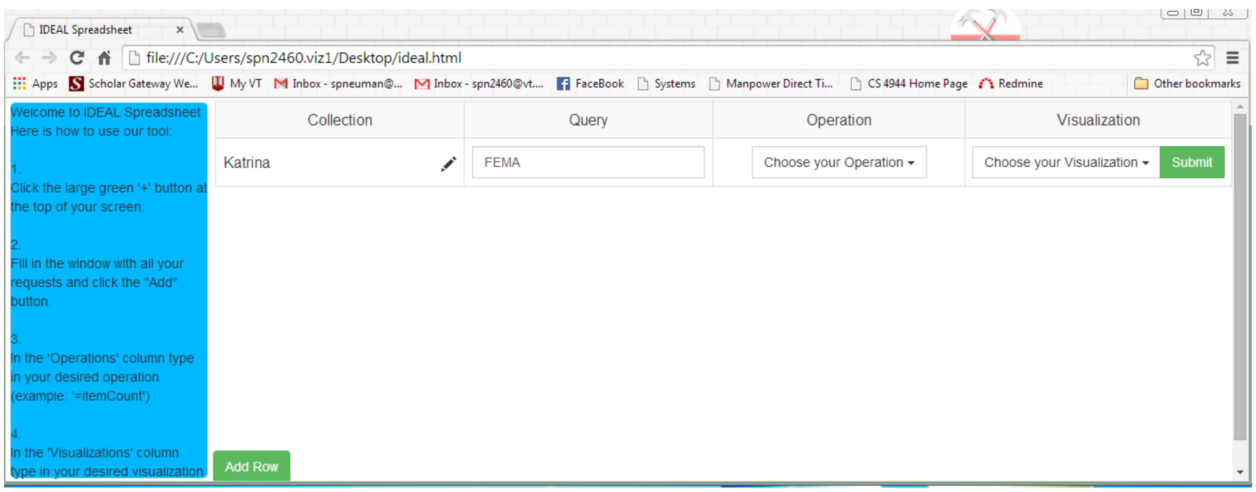
If you're using Spring Tool Suite as mentioned in the Workflow section, it will automatically build a .war file for you. Using traditional Eclipse, you may have to export this project as a .war file. From here, you can deploy this on a Tomcat server. Details on how to do this are listed here. Depending on the port the server is deployed on, you may have to rewire some of the AJAX calls to reflect this.

# Future Work

For those interested in continuing on the work of our project: we had several features that we were not able to implement within our project cycle. These include several more iterations of the user Interface and including more functionality.

## SECTION 1: UI ITERATIONS

With future iterations of the UI we hope our successors can further work with the IDEAL team to get to the specifications that they desire. The IDEAL Team made a few critiques of our interface at the end of the project cycle that we were not able to implement.. They wished that the UI was more intuitive. To make it more intuitive they wished that the blue help bar on the left side of the page held more detailed instructions for first time users so they can quickly become acclimated to the system quickly.



Welcome to IDEAL Spreadsheet
Here is how to use our tool:

1.
Click the large green '+' button at the top of your screen.

2.
Fill in the window with all your requests and click the "Add" button.

3.
In the 'Operations' column type in your desired operation (example: '=itemCount')

4.
In the 'Visualizations' column type in your desired visualization

| Collection | Query | Operation | Visualization |
|---|---|---|---|
| Katrina | FEMA | Choose your Operation | Choose your Visualization |

To aid in this aspect, they suggested making the info bar on the left side larger so it is easier to read to gather instructions. They also suggested the inclusion of a help button so that further information can be gathered on how to use the system. The use case they described involved using the built-in modals we are already using from Twitter Bootstrap and opening a new modal on-click of the help button. The user could then search the modal for all information they should need. Our IDEAL Spreadsheet team suggest making a ‘search’ text box in the modal so faster recognition of all information can be achieved. Finally, the IDEAL Team suggested changing the headings under Query and Operation, since the vocabulary seems to be geared more to our underlying understanding of the system, and not necessarily intuitive for a user. They wished to change the title of the ‘Query’ column to ‘Keyword’ because that is how it actually functions as a keyword filter on the collection to get the relevant documents from Solr. They wished to change the ‘Operation’ column to be titled ‘Analysis’ because it is the the specific analysis you wish to do on that filtered down collection.

## SECTION 2: FUTURE FUNCTIONALITY

The IDEAL Team showed interest in some functionality aspects which we were not able to complete. First, they wished that queries which are computed a lot are saved so that a user would not have to individually create queries they have consistently used in the past. This would allow a user to leave the web application and come back and continue research that they were doing before. Right now, our application does not hold state between uses, however, to implement this functionality would only require using a Database to back the information to store User data and User-specific query data. Further, the team expressed interest in being able to edit the ‘stop words’ of the word cloud. Right now, the word cloud picks up occurrences of common words, such as the word ‘the’. This decreases the usability of the application, by not providing relevant data to the user. Although, we already filter out some of these common words, it would be good to provide the functionality as part of the Keyword or Analysis column. It would allow for more relevant visualizations.

The last two expansions in functionality they wished to implement deal heavily with Solr. They wished to see more options for our Analysis and Visualizations. Currently, we were only able to retrieve analysis on the word count and display it as a Word Cloud. The IDEAL team was not exactly sure other Analysis or Visualizations they wanted us to perform during our project cycle, however, our implementation allows for the expansion of both these features so that in the future the application can improve. These would involve more back end development. New Analysis features would require different search criteria for Solr.

# Lessons Learned

One of the largest problems we faced during our project cycle was trying to set specific requirements with our clients. The original idea was for our project was to implement an Arcspread like interface for the IDEAL project. Those requirements were ambiguous due to the lack of our domain in the project. We had never seen or were able to acquire Arcspread, and our direct client was had not used it extensively either. Also, we were completely new to the IDEAL project itself and had to get up to speed incredibly quickly. For this reason, we were left very vague on requirements and functionality of our application. This taught us to set specific and clear requirements early for our implementation later.

We also learned to put more research into the implementation early. We researched, but determined that most of our work would be involved on the back end of our application interfacing with Solr and the Hadoop Cluster. In fact, that was one of the easier parts of our application. We spent a lot of time working on the front end design bogged down by the ambiguity I already talked about in the previous paragraph. The ambiguity led us to waste time trying to nail down functionality that the IDEAL team would want which stopped us as we had to wait to know what Solr queries to implement.

The final problem that was tough for us to overcome was to understand the scope of the project as it related to the entirety of IDEAL. Understanding not only our part, but all other active groups in the overall implementation of the IDEAL System helped guide our efforts, and allowed us to gear our efforts to completing tasks that were not distributed to other teams.

# Acknowledgements

We would like to thank a few people for helping us throughout this process. First and foremost we would like to send our deepest thanks to our client Mohamed Magdy for all his work aiding us in our completion of this project. Further, we would like to thank the entire IDEAL Team (NSF IIS - 1319578: Integrated Digital Event Archiving and Library) for allowing us to aid them in their research pursuits by creating this application.

# REFERENCES

Apache Hadoop. (n.d.). *Hadoop - Apache Hadoop 2.3.0*. Retrieved May 2, 2014, from http://hadoop.apache.org/docs/current/index.html

Dean, J., & Ghemawat, S. (2004, December 1). Google Research Publication: MapReduce.*Google Research Publication: MapReduce*. Retrieved May 4, 2014, from http://research.google.com/archive/mapreduce.html

The WARC File Format. (2006, January 1). *IIPC Framework Working Group: The WARC File Format (Version 0.9)*. Retrieved May 3, 2014, from http://archive-access.sourceforge.net/WARC/WARC_file_format-0.9.html#anchor3

WARC, Web ARChive file format. (n.d.). *WARC, Web ARChive file format*. Retrieved May 4, 2014, from http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml#sign