Tweets Metadata

04.28.2015

Presented by

Virginia Tech
Computer Science

CS4624: Multimedia, Hypertext, and Information Access
Taught by: Dr. Edward A. Fox

Client: Mohamed Magdy

Group Members:
Christopher Conley
Alex Druckenbrod
Karl Meyer
Samuel Muggleworth

Table of Contents

Team Members, Personnel, and Roles

Dr. Edward Fox	[Professor, Principal Investigator]
Mohamed Magdy	[PhD. Candidate, contact, Client]
Chris Conley	[Student, Front End]
Alex Druckenbrod	[Student, Back End, JSON]
Karl Meyer	[Student, Communications, Manager]
Sam Muggleworth	[Student, Front End, Back End]
Sunshin Lee	[Graduate student contact]
Carlos Castillo	[Contact at QCRI]

Project description

The previous CTRnet and current IDEAL projects have involved collecting large numbers of tweets about different events. Others also collect about events, so it is important to be able to merge tweet collections. We need a metadata standard to describe tweet collections. We need software to carry out the merger, and to prepare summaries and reports about the respective collections, their union, and their overlap, etc. Preliminary work on this was carried out by Michael Shuffett (Ref. iv). That should be tested, extended, applied, further documented, and disseminated.

- Project deliverables:
    - a standard for tweet sharing and for tweet collection metadata
    - methods for merging such collections and delivering reports
    - Web application implemented with the aforementioned methods
- Expected impact of the project: better collaboration among projects working with tweets, especially related to emergencies and investigations for research purposes
- Required skills of the team engaging in the project: interest in tweets, metadata, collection merging

Program Skeleton

- The main goal of the program is to merge two specific collections of data and produce a summary of what happened within that merger process.
    - The actual merged data is not the goal – there are many tools to merge two databases.
    - We are mainly concerned with how the merge happened, i.e., how many duplicates were there, how many total tweets, etc.
- When the user wants to upload a collection to the database, they will be prompted (**Figure 1**) to fill in information about the collection
    - This includes the names that they applied to any of the Twitter standard metadata they collected for tweets as well as the names of any additional types of metadata. This allows for proper mapping of data during merges.

- The only required fields for any given tweet are the TweetID and UserID[1]

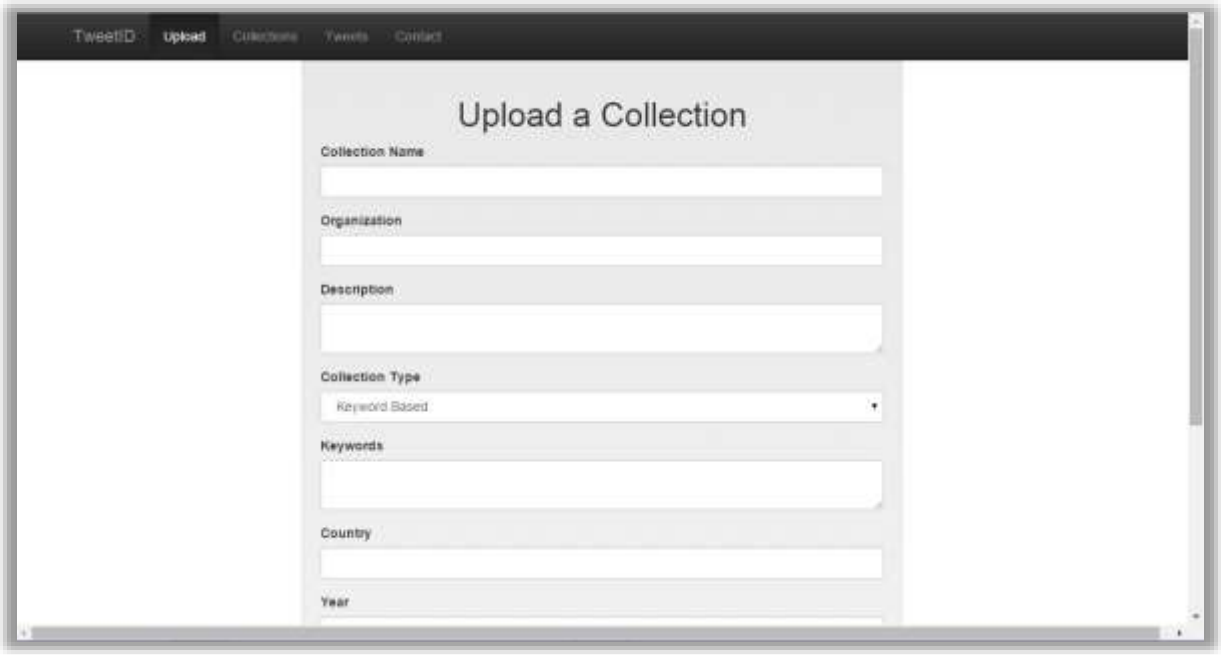

Figure 1: Michael Shuffett's Upload information page

- Merging occurs from the page(s) displaying the collections in the database (**Figure 2**)
  - After selecting two collections to merge, the merge can move forward
  - The request is completed server side and then sent back and the merged collection is made available for download and available to be merged with additional collections.
  - The response sent back contains a report of the success (or failure) of the merge, (**Figure 3**)

[1] This is due to changes made in 2014 to the Twitter Terms of Service and Developers policy (ii, v)

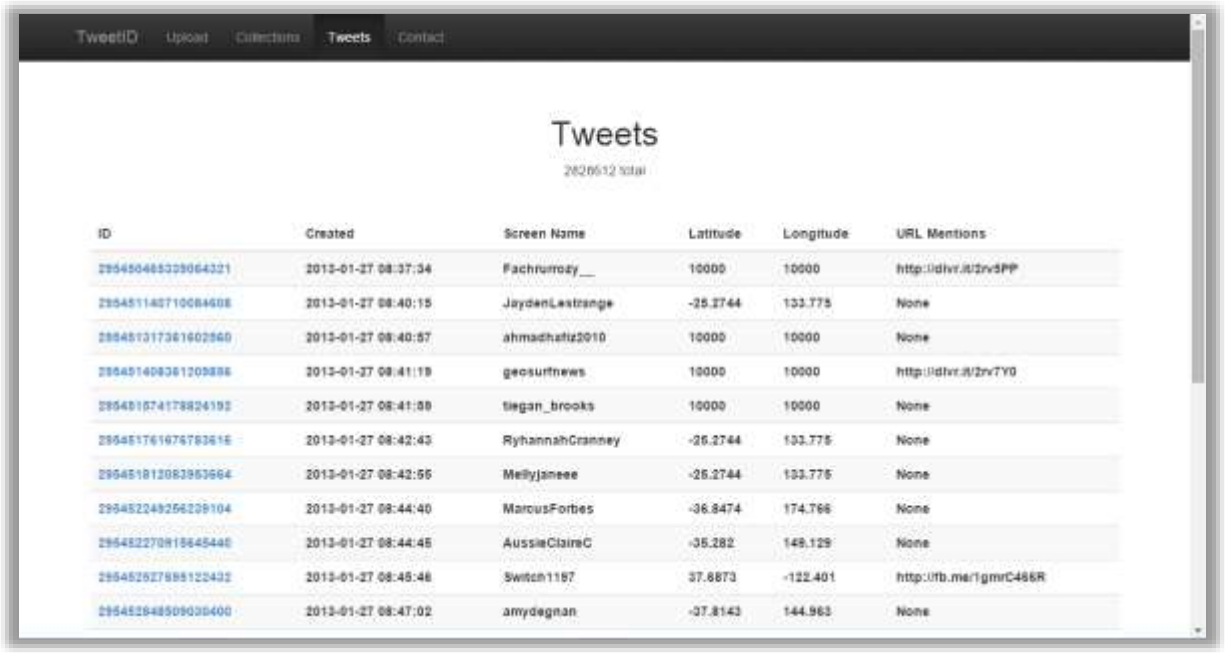

Figure 2: Shuffett's tweet listing page

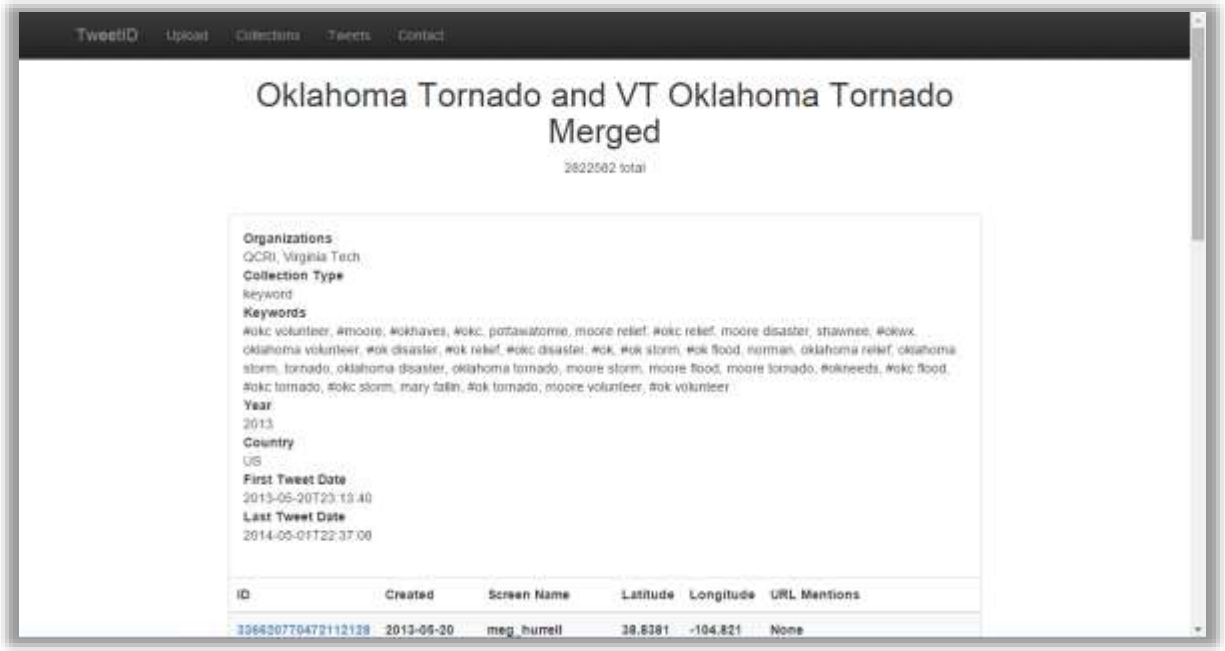

Figure 3: Shuffett's Archive information and listing page

- The following was the proposed color scheme and potential base layout for the web tool (nav bar and injection frame) (**Figure 4**)

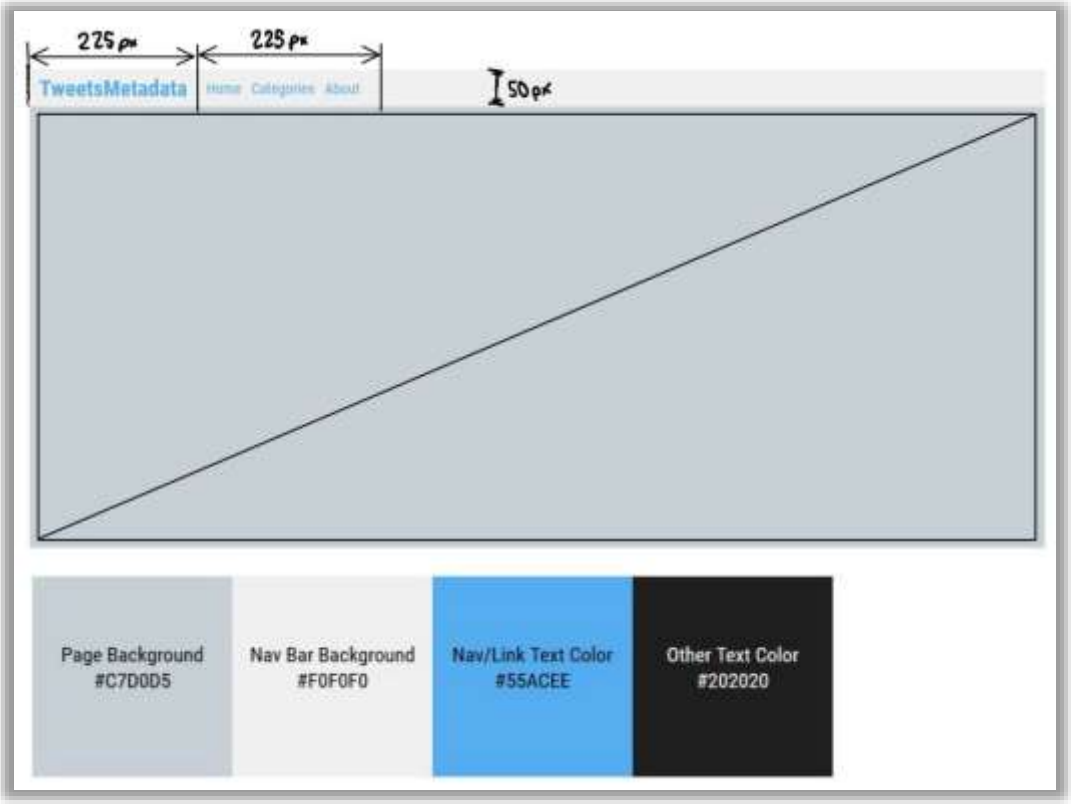

Figure 4: Our mock-up for the future web app and its color scheme

Note: Merges are not persistent – unless merging additional collections, after navigating away from the summary page the merge status is cleared.

Timeline

Plans: give milestones at least twice per month

- **February 5** - Have contract signed by both the client and instructor and Tweet level standard approved
- **February 19** – Standard Collection level metadata approved
- **February 27** – Submit Requirements/Design Report
- **March 5** - Storyboard for upload tool and GUI layout design
- **March 19** - Upload tool ready and Storyboard for merging/summary tool
- **April 2** - Merging tool ready and storyboard for gluing tools together
- **April 16** - Glue checkup and report any modifications made
- **April 30** - Continued testing
- **May 6** - Deliver product

Future Development

- The software should be able to upload any file with tweets in it. Right now there is only one format that is accepted: it needs to have 6 fields with tab-separated values.
- The software only takes TSV-formatted files, and it should take: JSON, CSV and TSV files in order to upload any type of reasonable collection
- Right now, it does not allow headers, but the new software should allow for this type of differentiation so as to allow for collections that have been manually altered to display their headers as well
- Future development should continue to only require the TweetID and UserID in the collected tweets metadata.[2]
- Possible: to make an autonomous tool, this could avoid the necessity of the user manually mapping archival information to make the merge possible
- Possible: to use a schema mapper tool like the one Dr. Fox used for archaeological dig information (Ref. iii)

[2]As before, due to changes to the Twitter ToS and Developer Policies (Refs. ii, v)

Design

When the user goes to this tool, they will land on the home page (**Figure 5**).

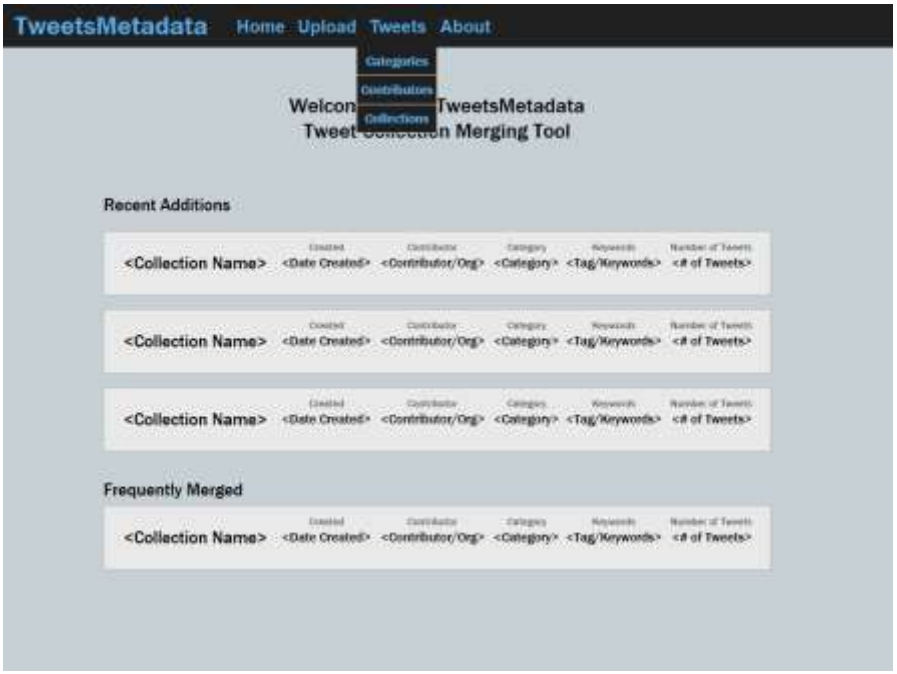

Figure 5: Home Page

Use Cases/Scenarios:

1. Uploading a collection: When uploading a collection, the user would click on the upload link at the top of the page.
   a) The first step of the upload would render as a modal[3] over the current page (**Figure 6**).
   b) After selecting a file of appropriate format, the user would click next. The current modal would resize and display the content for providing collection information (**Figure 7**).
   c) After filling in the required information and any additional information the user would click next again and the modal would change to reflect the last step of providing field mapping information (**Figure 8**).

[3] A modal being a user interface element similar to a dialog window that takes focus over the rest of the page. http://en.wikipedia.org/wiki/Modal_window

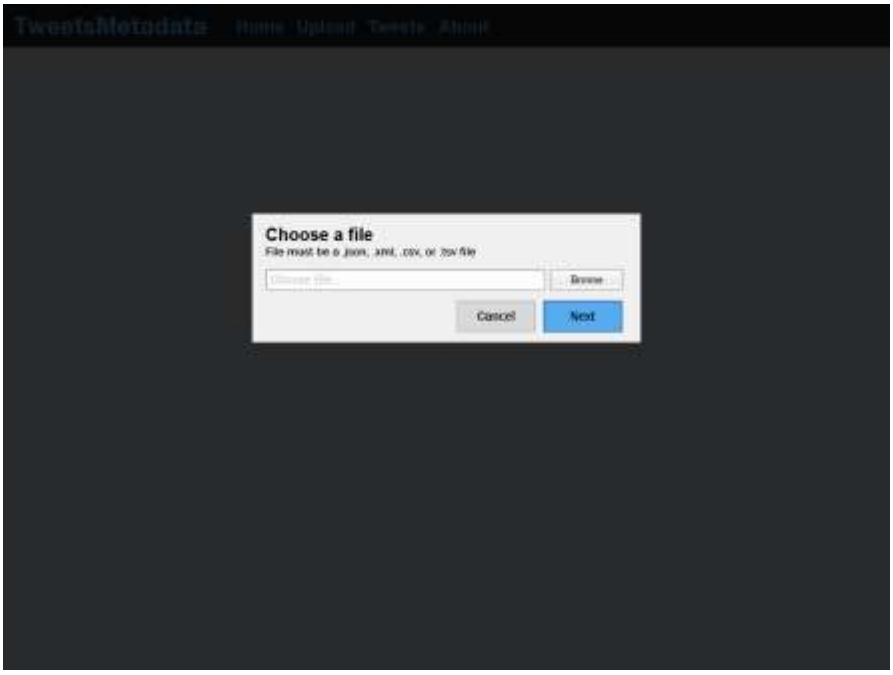

Figure 6: Choosing a file.

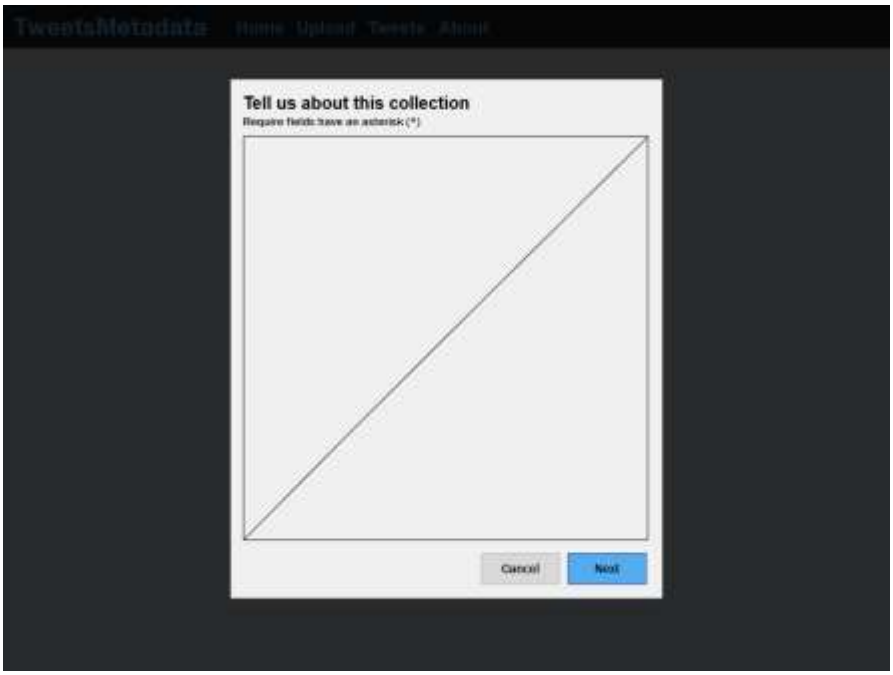

Figure 7: Providing information about the collection.

Figure 8: Providing information for mapping field names.

2. Viewing list of Categories/Contributors: To view the list of Categories or Contributors the user would: put their mouse over the Tweets link in the menu bar to bring up the additional options for viewing tweets and click either the Contributors or Categories links displayed,
   a) Then the page for that list format would come up as in **Figure 9**.
      i. Note that when changing the page, the status of the merge builder would be carried over (i.e. if a collection had been selected for merging it would still be selected on the new page)

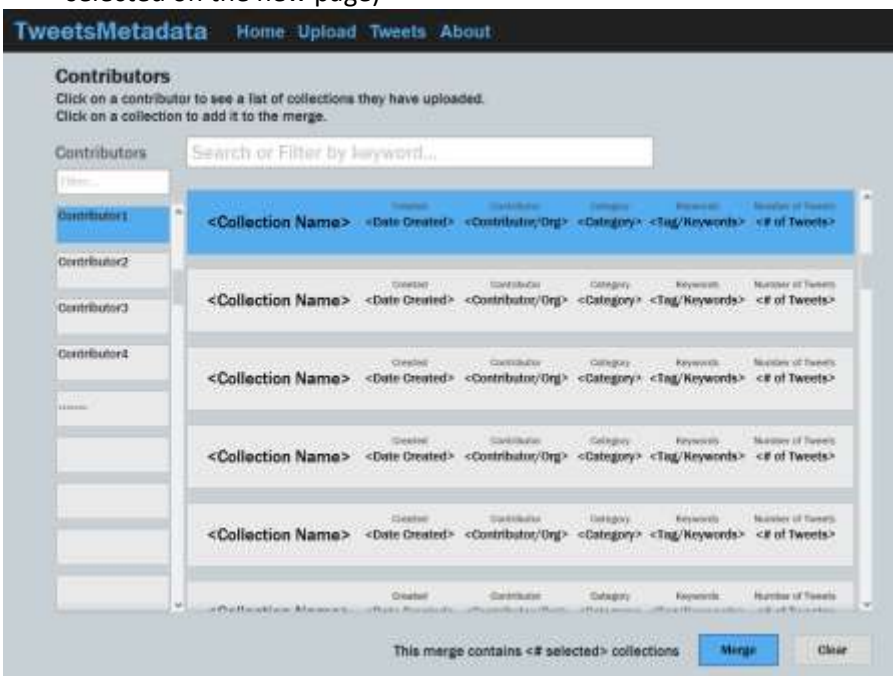

Figure 9: Categories/Contributors Page.

3. Viewing list of Collections: To view the list of Collections, the user would: Click the Tweets link at the top -or- mouse over and click collections in the drop down
   a) The page for the list of collections would come up as in **Figure 10**.
      i. There will be a way to refine search by date created, contributor, or category

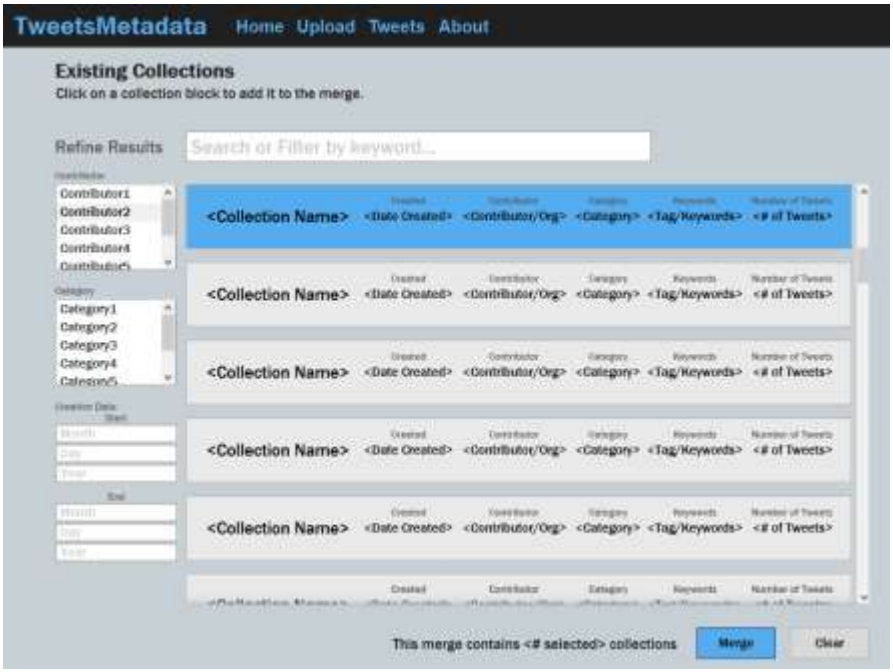

Figure 10: Collections Page. This page will be similar to the Categories/Contributors Pages, however it will have additional filters for keywords, contributors, and categories.

4. Preparing/Executing a Merge: (occurs from list of Categories, Contributors, or Collections). To request a merge, the user can select Collections to include from any of the list view pages (**Figures 9** & **10**).
   a) To start a merge, the user clicks the "Merge" button found at the bottom of the page.
      i. To cancel/clear merge status, they click the clear button.
      ii. To remove individual collections from a merge, they deselect them from the menu.
   b) The request is sent to the server which processes the query and returns the resulting merge.
   c) When received, the merge summary and report page is brought up, displaying the results of the merge. This page also provides the opportunity to download the resulting merge and/or print out the summary. (See **Figure 11**).

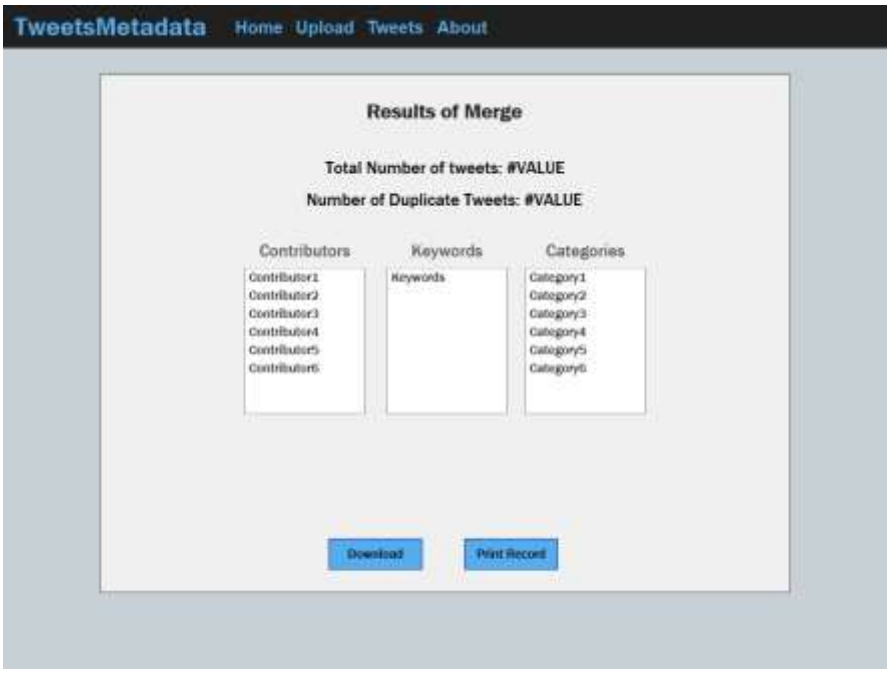

Figure 11: Merge summary and report page. This page contains all information gathered about the merge including total number of tweets, overlap of collections, keywords used for the entire merge, etc.

Implementation

The TweetsMetadata project is intended to provide a tool for the merging and compilation of collections of tweets surrounding various events or topics. This report discusses the deliverables of the project, the technologies being used, and the management approach for implementation. The deliverables include the standard for describing tweet collections and a tool for executing and summarizing merges of collections. The implementation and preparation of these deliverables will be cooperatively executed based off our roles (i.e. front end developers focusing on front end development).

Introduction

Our project goal is to establish a twitter metadata collection standard, to be used for a web application tool, which will merge two archives of tweets through user-defined manual-mapping, and produce a metadata summary of the merge giving the user an opportunity to save the results of that merge into a database or to download the results of the merge.

Background

The previous CTRnet and current IDEAL projects have involved collecting large numbers of tweets about different events. Researchers, investigators and others are also collecting tweet metadata about events, so it is important to be able to merge tweet collections seamlessly, giving the user well-defined control over the merge and the mapping of metadata. We need a metadata standard to describe tweet collections. We need software to carry out the merger, and to prepare summaries and reports about the respective collections, their union, and their overlap, etc. Preliminary work on this was carried out by Michael Shuffett (Ref. iv). That should be tested, extended, applied, further documented, and disseminated.

Project deliverables (Ref. i)

- a standard for tweet sharing and for tweet collection metadata
- methods for merging such collections and delivering reports
- Web application tool implemented with the aforementioned methods

Expected impact of the project: better collaboration among projects working with tweets, especially related to emergencies and investigations for research purposes.

Required skills of the team engaging in the project: interest in tweets, metadata, collection merging, database management, graphic design, user interface.

Technologies and Software

The software and technologies used for this project were largely decided by the provided code from Michael Shuffett's project as we are expanding his work from May, 2014 (Ref. iv).

On the front end, Shuffett had been using jinja 2 (templates), WTForm/Flask-WTForm (forms/uploading info), Bootstrap/Flask-Bootstrap (style/CSS), and jQuery/Flask-Script (client-side scripting). These tools make it easy to create form and list based tools such as this one. **Figure 12** shows an example of how easy it is to build a dynamic section of a page.

```
{% for collection/tweet in collections/tweets %}
      <div>
            <ul>
                  <li>collection/tweet.name</li>
                  <li>collection/tweet.created</li>
                  ...
            </ul>
      </div>
{% endfor %}
```

Figure 12: Code snippet

As demonstrated above, jinja2 makes it very easy to recycle templates and create dynamic pages that can render large sets of data with minimal programmer effort. With the other technologies, Flask specific versions of each have been included allowing for guaranteed cooperation and compatibility with the server side as it is using Flask for its server software.

The backend of the web application is written in Python. We are specifically using the Flask library for easy server integration. All code has been uploaded to a team GitHub page, where each member on the team has access to pull and push to the source repository. See **Figure 2** for the plan/phase breakdown.

There are mainly three ways where the front and back ends will interact:

1. When a user requests a page (ie. collections list page),
2. When a user uploads a collection to the tool, or
3. When a user requests a merge of collections.

In the first case, on page load a RESTful call will be made to populate the page with whatever it needs (list of collections or contributors, associated information, etc). This call (GET) is a simple query for a list or additional information returned in json format. It is then broken down and displayed using jinja2 templates.

In the second case, the user has clicked the upload button, fills out the form ([Flask-]WTForm), and attaches the file to upload. Then the form is compiled into a packet and, if it is properly filled out (validated), POSTed to the server (again, RESTful).

In the last case, the user has decided to merge two collections. The collections chosen, the user clicks merge and a query is sent to the server to merge (POST). At the server an algorithm comparing all collection fields is used to create a temporary list of similarities. There will be an option to store this list permanently on the front end. If the user decides to keep this merge permanently then the server will create a new database entry for this merged collection. When completed, the merged collection is sent back to the user and made available to download along with a summary of the report (also compiled server side; part of the return packet). The packet is parsed and the merge success page shows the summary of the resulting collection (eg. what was merged, duplicate tweets).

Note: At the server the uploaded file will be parsed and inserted into a database containing tweet collections. Since we are continuing to use code given from last spring we are going to keep the initial database schema. Michael Shuffett's database design sufficiently adds a new collection to an existing organization's collections or simply creates a new organization in the database and matches the collection with the newly created organization. When finished, the server will respond with a success or failure message. The client side displays a message corresponding to the server's response.

Development Phases

The work for this project will be handled simultaneously, where the front end and the back end will be working towards a middle ground of implementing our minimal goals for this project.

Table 1. Estimated Development Phases

	March	March	March	April	April	April	April
Front-End	collection standard & mapping info	Style modi-fications	Mods cont'd	repackagin g tweets	testing		
Back-End	Upload & parse CSV files	Upload & parse JSON files	Upload & parse XML files	testing			
Both	obtaining req'd & addt'l metadata			obtaining req'd & addt'l metadata	merge font with back	generate summary	testing

Table 1 represents the order our team is imagining accomplishing our development goals within each month remaining on the project. This ordering was produced by each individual and compiled by Karl Meyer into a timeframe. There is always room for improvement and room for additional goals to be added, which makes this chart something that will be amended regularly. Our team is utilizing the website: seenowdo.com to create project flow and to establish who is doing what and when and when our deadlines are.

For story and task tracking, we decided to use the online tool “SeeNowDo” which is a simply tool for Agile development. **Figures 13** & **14** show the back and front end storyboards and the tasks that we broke each one into.

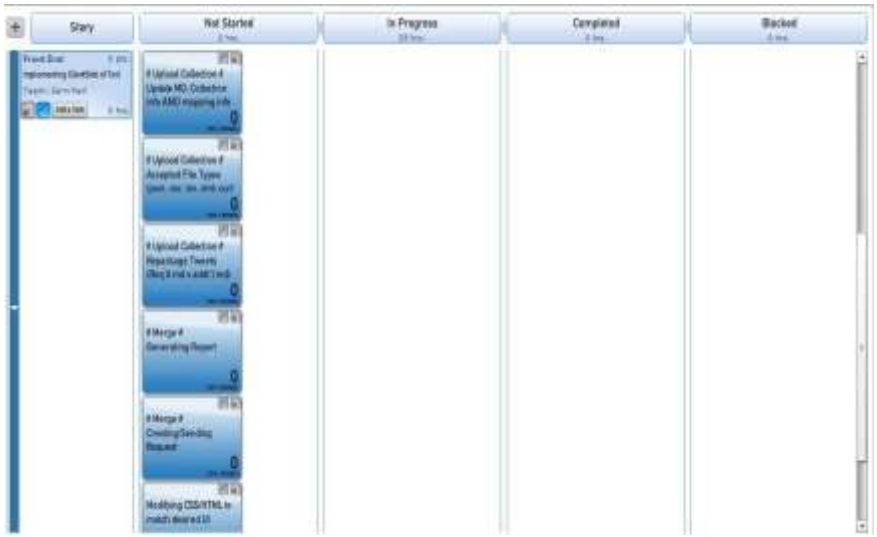

Figure 13: Screenshot of Front End storyboard. The story is broken up into smaller tasks.

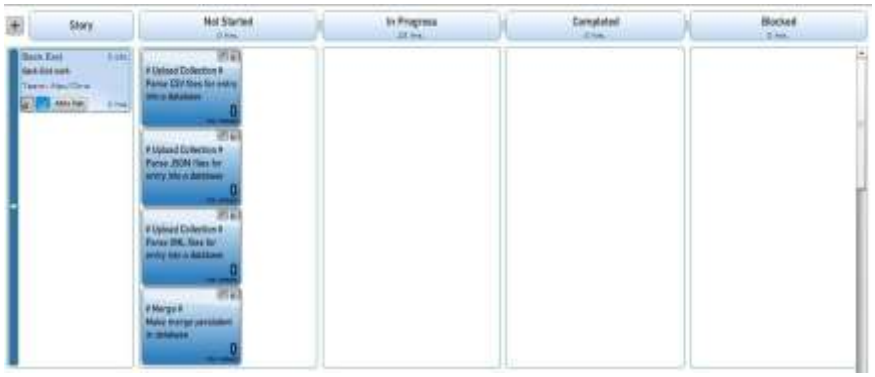

Figure 14: Screenshot of Back End storyboard. This story has less tasks planned, however they are more time consuming.

As previously mentioned, our group has set goals that will accomplish the task set out for us by Dr. Fox. This goal is to have the web-app tool merge two twitter archives by allowing the user a manual mapping capability to match their metadata standard to ours and have it produce a summary about the merge. Our reach goal is to make the process autonomous and to allow the user to simply press upload with any file format, and our tool will utilize the principles of a schema mapper to relate the metadata standards to ours and to allow for a smooth merge that could be automatically called by another program.

Management Approach

Each team member has a specific task summarized below:

Alex:	back-end development work, coding to accept all file types, and on making merge persistent in the database
Chris:	back-end development work, coding to accept all file types, and on making merge persistent in the database, writing up pertinent information into implementation report
Karl:	communication between group members, gantt chart, research
Sam:	front-end development work on uploading a collection, refitting wireframes and user interaction flow charts

From a management perspective, it is necessary to reach the goals of the customer, but, it is also pertinent to provide them with potential alternative solutions or services that may be of relevance to their desired end-goals. For this project, we sat down with the client and ensured that the goals of the project itself would be met, but also showed how we might be able to improve upon the suggested solution by offering an autonomous solution. It is clear that this is a reach goal as we are highly constrained by time, however, it would be of great use to the researcher's goals.

Quality Assurance

Another important aspect to consider is how to ensure the quality of our end product. Two things that are of the most importance are: how we are meeting the requirements and how we are going beyond them (or might be able to). To do this we must come up with our own scoring of how we will judge our product's readability and transparency by ensuring that both our Principal Investigator Dr. Fox and Mohamed can utilize and understand our tool and its code. This will eventually require some form of product testing by both the client and maybe a test group unrelated to the project so as to ensure ease of use (as the product will be available on the internet for anyone to use).

Security/Safety Risk Management

Our project requires attention to tool use and legal risks. One is the approach to the use of the Twitter. Currently, we are operating on the advice of our Principal Investigator and we are developing the project according to the law and under the presumption that the web-app tool will not only be used for lawful purposes, but also be operated by users according to the Twitter Terms of Service agreement.

Second, how we are going to approach interacting with the users and researchers who will be utilizing this tool. We plan on generating a message that will be present on the web-app page which spells out the way the tool works and specifically mentions how to use the tool according to the twitter terms of service. It might also be pertinent of us to say in this message that the development team should not be held responsible for the use of the tool by any party other than themselves.

Third, what our terms of service will say, and what type of information we want about ourselves or the team on the web page. Something to consider is how responsible will we make ourselves for the content of the application and database.

Conclusion

Our implementation plan has been developed in a way that we believe is reasonable to be completed in the time allotted. The management and phase development of this software has been developed by each team member and then discussed as a group to ensure that we are not attempting to do either too many things or something that requires the full development of another phase of the project to be efficiently accomplished by the final submission date. It is the responsibility of each team member to report to the manager their progress and the problems they encounter when working on each development phase so that the team manager can quickly apply resources and effect action on those parts of the project that are requiring the most amount of effort at the time.

Prototyping

Our efforts, as of this milestone date April 8th, have been focused on getting the upload to work. As of this checkpoint, we can only upload a tsv file, however we are testing csv files now. We hope to have xml and Json working shortly after our issues regarding csv files are resolved and fully tested and integrated. Our goal for our next prototype refinement report is to have a fully operational merge process. Currently, we can only merge based on static fields, the end goal is to force the merge to use our established fields for all merges. The idea is to scan their schema and ask the user to match their schema to ours. After this step is completed, we will merge as normal. Our reach goals for our prototype refinement is to have an autonomous schema-mapper with possible perk of being able to save a mapping by a specific user so that the manual mapping would only be necessary for extraneous schema entities that had not been recorded or evaluated by the web-app tool during previous merges.

As a part of the prototyping phase, we had begun switching the base code written by Shuffett from Python 2.7.9 over to Python 3.4.3. Fortunately, the other technologies being used for this tool (Flask, Jinja2, sqlite3, and jQuery) are all compatible with versions of Python greater than or equal to 3.3. This means that changes being made to port over are largely syntactic. For one, in 2.7.9, the print command is written as a statement and not a function. That is to say, 2.7.9 says, print “Message”, and 3.4.3 says, print(“Message”). This means that updating to the newer technologies and code efficiency will be possible in the near future, thus providing ample room for further improvement of this tool as tweet collection and metadata analysis will be heavily utilized in the near and possible farther future (whether it be Twitter data or some other type, the concepts are similar enough to adapt the given code.

However, upon further investigation and development, it was realized that there are too many bugs in the newest versions of Python (3.0 and later) to switch the system over at this point in time. Before April 8th, we had been in the process of debugging the collection upload process. There was a conflict in the syntax and use of the csv reader method in Python 3.4.3. Upon meeting with Mohamed Magdy, we determined that because Python 2.7.9 was being used for tools with the IDEAL project, it was in our best interest to revert back to that version of the software.

As far as front end design, our prototype finds itself toting an updated color scheme and the beginnings of a shift towards the anticipated final design. Using a responsive, bootstrap-based design gives us flexibility in building out the new design from where Shuffet left off. His well commented code and use of the Jinja2 will make it easy to continue work in updating the GUI. The show where some of the changes have been made. Right now one of our biggest concerns in the backend is what type of files the tool can process. Right now the files are limited to tab separated values. Our desired goal is to extend that to include comma separated, XML, and JSON. CSV implementation was relatively straightforward. Right now we have code alone that converts XML and JSON to TSV. We are currently trying to integrate this functionality into the main program. Our goal is to have that completed relatively soon.

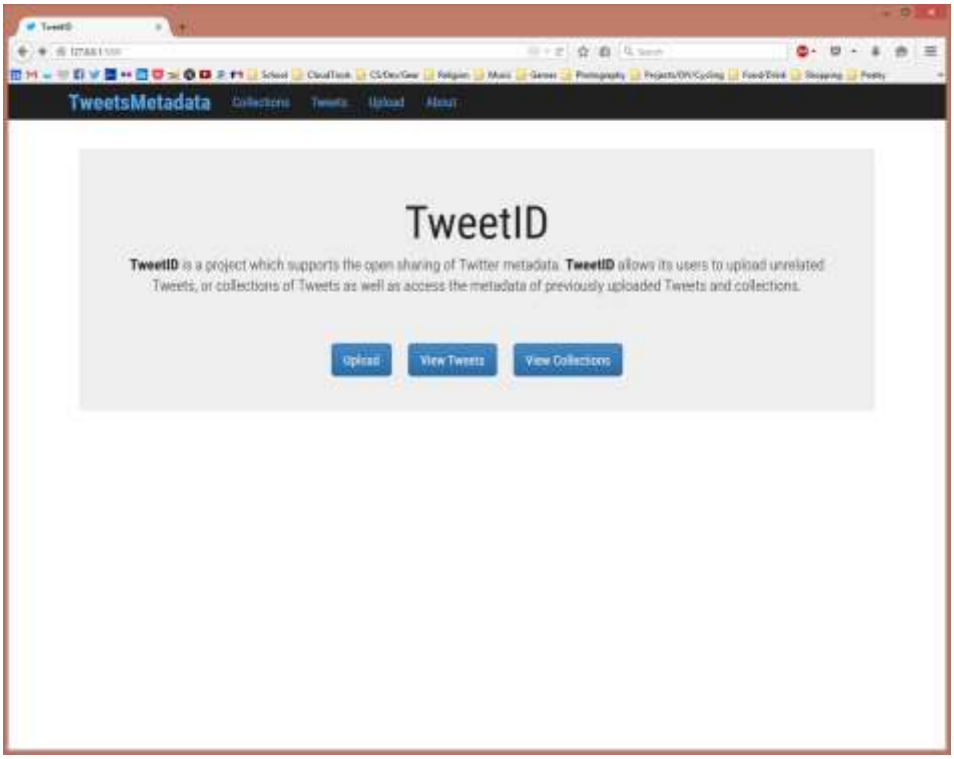

Figure 15: On the home page, we can see the update Nav bar and a sample of the new font (Roboto Condensed).

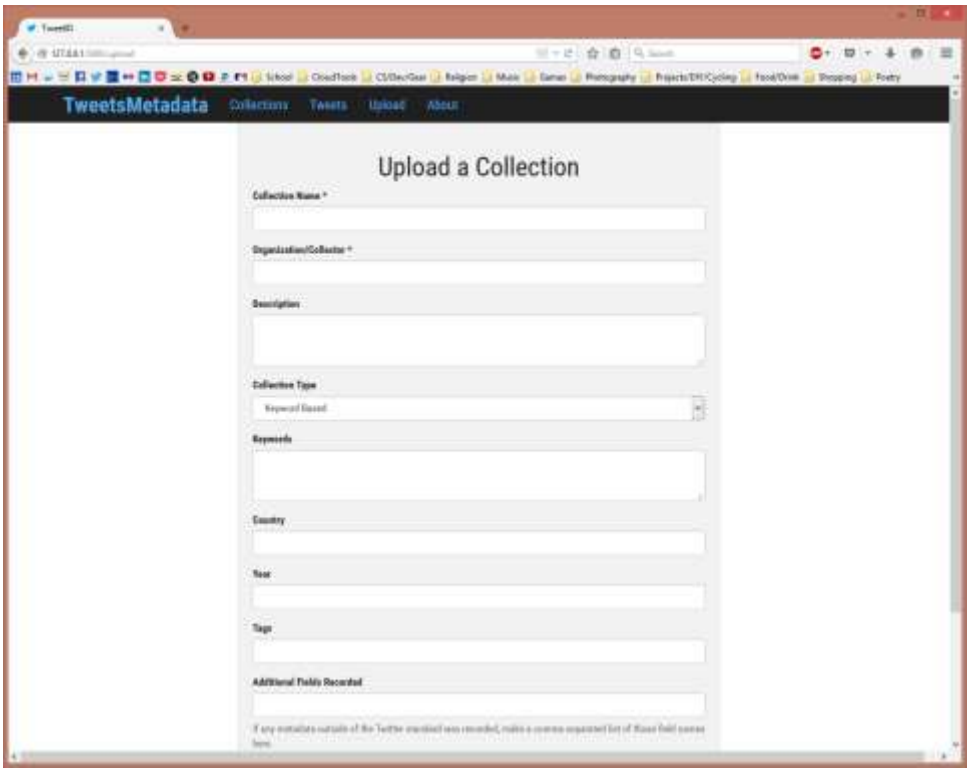

Figure 16: The Upload form has a new field for making a record of any fields outside of Twitters standard. More fields will be added for mapping field names used by the collectors.

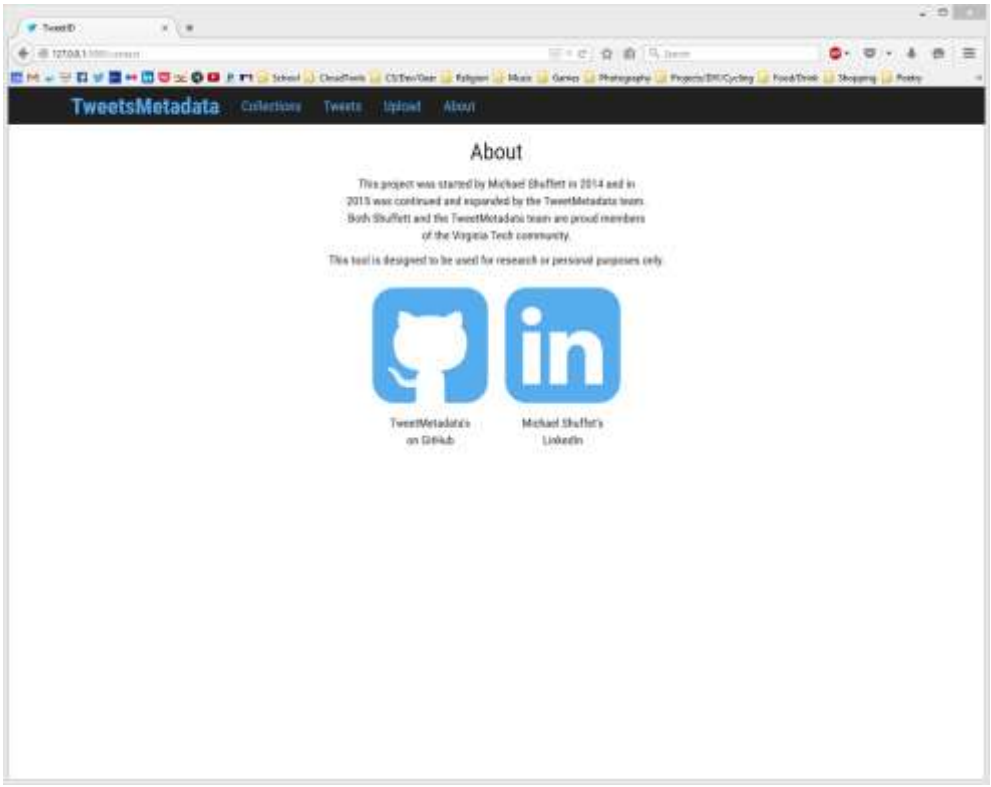

Figure 17: The About page has replaced the Contact page and contains links to the GitHub repository and Michael Shuffett's LinkedIn.

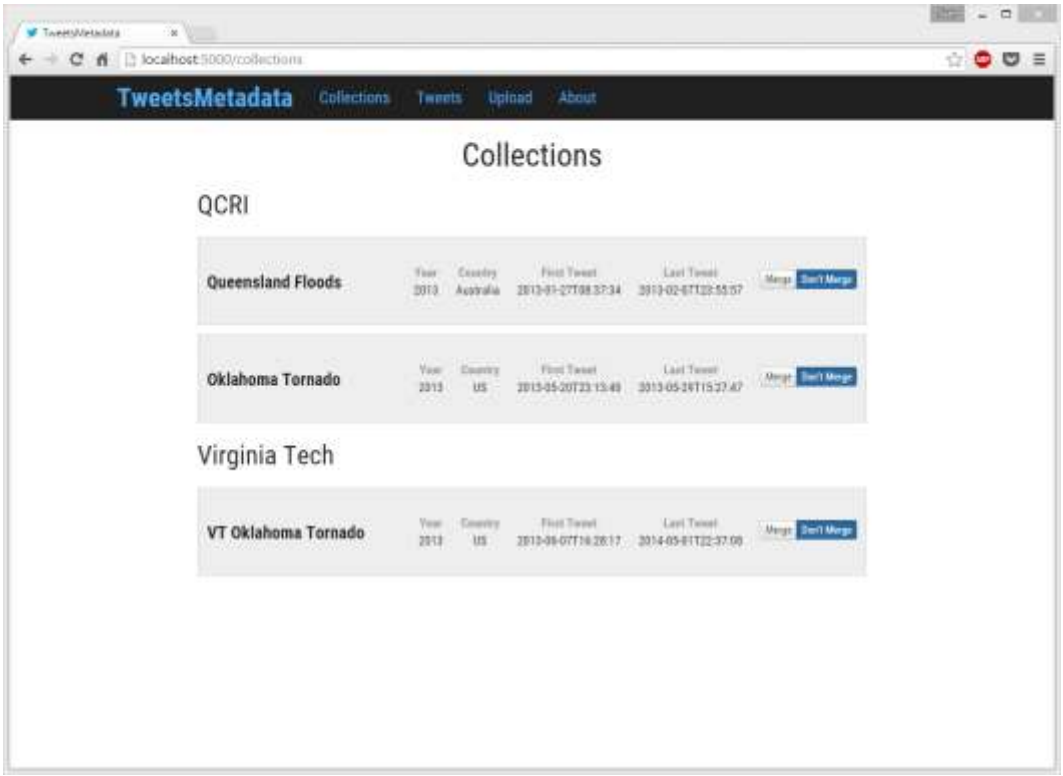

Figure 18: The Collections page is moving towards the wireframed design.

Testing

Our prototype has required some maintenance and upgrades which we are now bringing online. Some of the updates include: the manual mapping of schemas and implementing a way to keep track of individual IDs for collections that have been uploaded.

Currently, we have a full wireframe ready to go for the manual mapping part of our intended end-goal. For the final report, our team will need to clean up the design for collection details and implement the tweet mapping page in conjunction with the back-end development. In addition, there will need to be testing done for use cases to ensure that the flow of the tool is as expected.

As for the back-end, the team has modified the database to include a unique ID for collections uploaded. This is a better implementation than what we had because collections now can have the same name but they don't have to be the same collection. For instance, two organizations may have a collection with the same name, this would normally have produced an error with the previous implementation. Now there should not be any errors during upload of a collection which is ideal for the user because from our experience running tests on the code as is, there are a lot of collections with very similar or the same names. In order to implement this, the team had to modify the original code for listing so that it would list by collection at the base level compared to listing by organization then by collection from said organization.

To ensure the accuracy of this change in implementation, the team has added, and begun testing, sorting by name, organization, and year. We are utilizing files given to us by our client, Mohamed Magdy, which are supposed to be accurate representations of the types of files that might be uploaded to the tool when in use by researchers.

Through our initial testing of our newest implementation, we were able to decrease load time for the collection page because the listing is now sourced directly from the database, which is by collection, instead of first sorting by organization. Overall, this is a better design and allows more flexibility for the User.

Presently, our team has XML and JSON conversion working when we run our test files, but only in the case that their upload file is directly formatted to our specifications. Also, the conversion to a TSV file type only works smoothly for our pre-formatted test cases. This means that we need to create a larger collection of test files to further debug our issues when it comes to having files that are not exactly like the files provided to us, and which may have some variations. Some of these issues arose from our utilization and development upon Michael Schuffet's code, on which our project is built, which has some limitations due to its implementation and its exactness in being able to work only for specific situations and file types. Jagged (different object have a different number of objects in a field) XML or JSON also cause issues when uploading, however, we are currently working towards a process which can handle less perfectly-formatted files easier. To do this, our back-end development team is going to create fake data for the database because as of right now, the data we have will not suffice for the presentation/final report and the final tool that we are trying to develop according to the specifications laid out by our client.

One of the major changes briefly discussed above is shown below in **Figure 19**. This shows an ideal manual mapping wireframe where the tabs allow the user to navigate the two collections and their

metadata fields, before attempting to merge them together and produce a summary of that merge in the 3rd tab. Our front-end development team is unsure if this is the best way to implement this idea, but it is one that we have already created and have partial code implementation for. An alternative solution would be to have the two collections side-by-side on the web page with a Merge button at the bottom which would then take the User to another Merge Summary page. This is one of the things that we will be discussing with our client, as we are trying to be considerate of the ease of use of the tool by the User – which was one of the main complaints of the previous implementation of this tool.

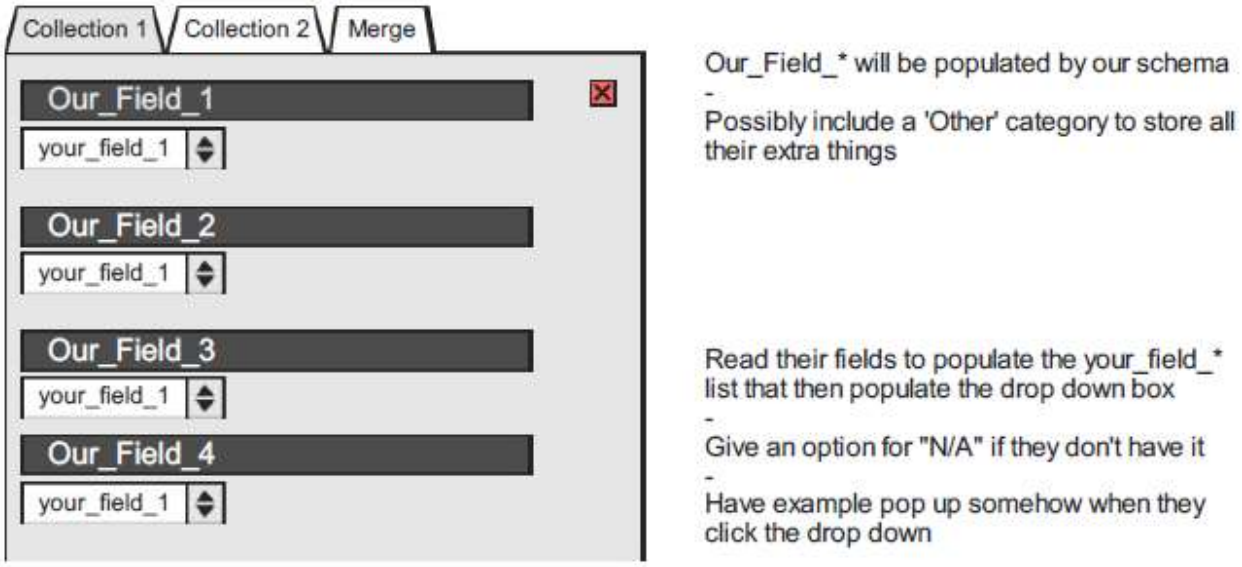

Figure 19: Wireframe of potential field mapping implementation

Our plans for future testing are three-fold. First, we plan on manually exploring our integration of the front and back-end development. This would be followed by our second phase of testing where we will take our implemented tool directly to the clients and have them use the tool, in any way that they see fit, so as to provide our team with more feedback and items to change. Our main focus at this stage will be usability testing and working towards a good flow between front and back end. Our third phase will be stress testing our tool, which may require attempting to upload/merge out of the ordinary files, large files, empty files, incorrect file types, and other such stress tests. After we have completed the first round in this manner, there will surely be another iteration of testing in this manner, if not more, so as to not only ensure the quality of our end-product, but also ensure that we have met our client's specifications and exceeded their expectations.

Table of Figures

References

i. CS_4624_12303_201501: Wiki->ProjectsS15->TweetsMetadata <https://scholar.vt.edu>

ii. "Developer Policy." *Developer Policy*. Twitter, Inc., 22 Oct. 2014. Web. <https://dev.twitter.com/overview/terms/policy>.

iii. "EtanaViz: A Visual User Interface to Archaeological Digital Libraries." *EtanaViz: A Visual User Interface to Archaeological Digital Libraries*. Virginia Polytechnic Institute and State University, 1 Oct. 2005. Web. <https://vtechworks.lib.vt.edu/handle/10919/20193>.

iv. Shuffett, Michael. "Twitter Metadata." *Twitter Metadata.* VTechWorks, 10 May 2014. Web. 09 Feb. 2015. < https://vtechworks.lib.vt.edu/handle/10919/47949>.

v. "Twitter Terms of Service." Twitter, Inc., 8 Sept. 2014. Web. <https://twitter.com/tos?lang=en>.