Crowdsourcing Transcription Platform

Da Kim, Daniel Martel, Edson Rios, Dimitri Rivera, Joshua Wozniak

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

***Abstract* — Maintaining records of historical manuscripts is a necessary part of preserving history. However, keeping an archive of the original documents is only one step of this process. It is common for handwriting in historical documents to appear as illegible. It is important for someone to transcribe these hard to read documents into something more accessible. Our objective is to provide a tool that allows this task to be crowdsourced to the public.**

I. Introduction

In this paper, we present our crowdsourcing transcription platform web application. This project was sponsored by Dr. Rosalind Beiler and Dr. Amy Giroux. The main aim of this project is to gain and make public the knowledge of past events as seen through the eyes of those who lived through them. By generating public history through mass collaboration, our application helps academic communities and the general public as a whole.

For the general public, public history is a great way to learn about our past. It also gives us context to our current status as a species, shapes our identity, and allows us to analyze our own stories and their importance to future generations.

As for the academic community, our application gives them new information as well as giving them more hands for their work. With the public's participation, historians are given more time to work on more large scale projects that require their expertise.

Our application helps create public history by engaging the general public in the process of transcribing old documents. It makes transcribing easy and intuitive for those that are experienced and provides teaching material to those that are not. As this process occurs, historical records are being digitized which make it easier to preserve, reproduce, and share.

II. Related Works

There has been software created in the past to accomplish similar goals to our application. One example is the Shelley-Godwin Archive [2]. Some features on our project were inspired by this website, like the side-by-side image viewer and transcription panel. Shelley-Godwin also has a transcription status for each document. The difference between that status and our status is that we have several more statuses than Shelley-Godwin (such as Editing and Encoding). Shelley-Godwin has a search page, but it is just a single-field search. Our search feature is much more advanced.

Another similar website is From the Page [3]. This website is more broad and offers a wider range of items to transcribe. For example, not only do they have collections of letters to be transcribed, but there is also a collection of a Stanford University professor's notebook and a handwritten playlist from Stanford's radio station. The text editor for transcribing these collections is very basic, offering only a plaintext editor. For our application, we were asked to incorporate a text editor with more features, such as underline, italics, and strikethrough.

III. Background

A. Frontend

The Crowdsourcing Transcription Platform is a single-page, web application that provides the user with a streamlined interface for communicating with the database. The application is built using the React library and Semantic-UI Framework.

The application flow starts with assigning a role to the user through a registration or sign-in process. The sign-in information, used to validate the user, is stored using cookies for persistence as well as security. This allows the user to access different components in the application such as for transcription, editing, encoding, uploading, and editing metadata. Without an account on the application, the user can only browse through the documents using either the browse or search component.

To participate in the transcription pipeline, the user must first authenticate themselves and check-out the document. Depending on the user's roles, the user can then edit, transcribe, or encode the transcription data to completion. If the user has metadata expert or administrative privileges, the user can also alter the metadata of the document. Furthermore, the user can upload their own documents using images, PDFs or external links and manually input corresponding metadata or upload them in a large batch using an XLSX file.

Outside of the core functionality of the application, the user can also utilize the user page to view and edit their account information and manage the documents checked out under their account. The user can also choose to view the tips page for transcription resources and help.

B. Backend

Our web application will accommodate six classes of users in a hierarchy structure of the following order starting with the lowest ranking user: general users, transcribers, editors, encoders, metadata experts, and administrators. Each user, except for the metadata expert and administrator, has the ability to request to rise in the ranks of our system. General users can register to become transcribers, transcribers can request to become editors, editors can request to become encoders, and encoders can request to become metadata experts. However, if a user is trusted, they can skip the ranking system by having an administrator update their role to whatever role is agreed upon.

First time visitors to the web application, or the general users, will be able to view and search for documents and the current progress that has been made in transcribing the document. They do not begin participating in the transcription process until they create an account on our application. Once they create an account, they start out with the role of a Transcriber.

Transcribers will be the first level of user that will be able to contribute to the transcription process. After logging in, a transcriber can check out a document and begin transcribing what they see in the document's image. In the case of a foreign document, transcribers will also be able to translate the document into English. After they have completed their transcription or translation, they can submit their work to the system for review. They will also be able to check in their documents if they give up on said document for other transcribers to finish. When checked in, their work will still be saved for other users to see and use.

If a transcriber applied and was promoted to an editor role, they are able to do all that their previous role can do, along with some added functionalities. When a transcriber submits their transcription or translation, editors check out a copy of the document and review the transcriber's work. While reviewing, editors can make changes to the transcriptions and re-release the document for more transcribers to work on, if need be. However, if they are happy with a transcription or translation and consider it completed, they finalize the document and submit it to the database. The document is then open for encoders or higher level users to work on.

If an editor applied and was promoted to an encoder role, they are given more permissions on top of their current permissions. After an editor submits a final version of a transcription to the database, an encoder will go in and tag keywords with specific codes. These codes are internationally agreed upon values that distinguish between certain people, places, organizations and more. After this final round of processing, transcriptions are fully completed and all processes for editing any part of the document will be locked unless a higher ranked user releases it again.

When an encoder is promoted to a metadata expert, they are given every privilege in the system besides the ability to affect user data. They are able to set the status of a document as they see fit. This includes releasing a previously completed document to any part of the transcription process or disabling any part of the transcription process for the document. This class of user is also able to access and edit the research notes of any document. This field is hidden from the users in the ranks below metadata experts. Metadata experts have full access to the web application's database as well. They may add, delete, or edit documents and their metadata in our application's database. They can do this in one of two ways, by manually entering the data for a single document, or submitting an excel sheet filled with data for multiple documents.

Finally, when a metadata expert is promoted to an administrator role, they have every permission in the system. Their privileges encapsulate all user type privileges such as transcribing documents, editing transcriptions, encoding documents, and accessing the web applications database. Along with this, administrators can accept or deny promotion requests from transcribers, editors, and encoders. Administrators can give any user any role that they see fit. However, they may not affect other administrators.

C. FusionAuth

When developing a web application with the ambition of thousands of users from different geographical locations you are going to need a robust auth solution.

Our needs consisted of authentication, authorization, user role management, and storing user data. FusionAuth checked every box. It has an extensive feature set and the ability to scale as large as we need it. They have their own APIs as well as client libraries that we were able to take advantage of. There is also a very user friendly admin panel that our sponsors will be able to use with no technical background needed.

FusionAuth's monetization strategy is hosting and support, so we were able to self host this entire solution for free. Set up is a breeze and it is also simple to containerize with docker and bundle with our application.

The community for FusionAuth is growing and right now they have a Slack channel for announcements and general support as well as feature requests. In the future they plan to have a forum on their website. However when developing with their APIs we never experienced the need

to take advantage of the community resources because their documentation is excellent and very developer friendly.

IV. Technical

A. Frontend

The web application was created using React with JavaScript, HyperText Markup Language (HTML) and Cascade Style Sheets (CSS). When planning the tech stack we would be using for development, we were mainly split between choosing using React or Angular for creating the frontend.

Some reasons why we chose to create the frontend portion of our application using React rather than Angular include its flexibility and easier learning curve. React is also much more popular and has multiple resources to help with development. Since the project was to be completed within a limited amount of time, the ease of use was a very important trait that React had over Angular.

A development framework is the set of tools and components that can be used for developing a web application. When deciding on a development framework we might want to use, we had to think about the desired appearance of the application and the functionality of the tools given by the framework. We decided to use Semantic UI as the development framework.

Our web application is a transcription website, so two necessary components when building our website is a viewer to display the documents that are being transcribed and a text editor. When searching a text editor, we looked for something that was efficient, developer friendly and included the requested features such as italicization, strikethroughs, bolding, changing letter colors, changing font size and more.

The third party provider we chose for satisfying these conditions we had set was QuillJs. QuillJs is free to use in both personal and commercial projects asking that we credit the authors. Documentation to customize and configure QuillJs can be easily found.

To create the pdf viewer, we used a combination of different react libraries. We used React-PDF to display the document to the user and React-zoom-pan-pinch to allow it to sit comfortably in a viewport for the user.

B. Database

For our database management system, we decided to use MySQL. This service comes with a whole host of benefits such as a $0 price tag and a companion application called MySQL Workbench. Other benefits include the large supporting community and the small learning curve of the SQL domain language. We used a SQL system instead of a noSQL system, such as MongoDB, because the data we are storing is very structured and is defined by its relationship to other information in the database. Therefore, the data fit very nicely into the SQL structure making for a relatively straightforward database design.

Since our web application is primarily based around documents, it is only fitting that our database would reflect that. The document table is the center of the database with all the related data connected in a spider-like structure. The only tables not connected to the document table are the request, individualnotification, and notification tables. While these tables are essential to our web application, they do not directly describe or relate to any other entity in our application. They instead are used purely to store notifications for our administrators when changes to the database are made, as well as changes made through the FusionAuth portion of our application.

C. Backend

For the backend portion of our application, we decided to use Spring Boot. Spring Boot is a Java based framework that is used to build spring applications that run independently. We chose this service mostly out of comfortability. Spring Boot uses the Java programming language which is a language that our entire team was experienced in. This helped our schedule because we had about half the knowledge necessary to start working with Spring Boot. With this powerful service, we were able to create the functionality necessary for our application. Below you will find the functionalities of our system as seen by the backend.

- Add a single document to database with an optional set of images
  - Endpoint address: /documents/add-one-doc
  - The first thing that happens is a quick check to see if the user making the request is at least a metadata expert with a cookie value. Then we check to see if a set of images was uploaded. If one was, we gather them all up and combine them into a PDF. We then store that file in our environment and create a personalized URL that is used for accessing the image. That URL is then entered into the document's information under the column name PDFURL. To add a document, a special JSON structure is required. This JSON contains the document, all people relationships, all organization relationships, all place relationships, and all related letter relationships.

All the data goes through strict error checking including checking if required fields of each entity have been entered. If all is well, all entities are added and linked in our database. Finally a notification is created and sent out to all administrators.

- Add a batch of documents with an excel
  - Endpoint address: /documents/add-excel
  - The first thing that happens is a quick check to see if the user making the request is at least a metadata expert with a cookie value. This endpoint takes in a .XLSX file with formatted sheets for each entity to be added. With this endpoint, we parse the entire excel sheet before anything is added. All the data is validated very similarly to the adding of a single document. If an error is found in the excel sheet, a notification is made and presented to the user that tried to submit the excel sheet. However, if all the data was fine, we simply add the information to the database and link the necessary entities. We then send a notification to all administrators.
- Update data in the database
  - Endpoint address: /documents/update-document, /person/update, /org/update, /place/update
  - These endpoints first check if the user is at least a metadata expert. For the document, the update is relatively similar to adding a new document. We do error checking on the data and make sure that the document the user is trying to edit exists in the database. We then delete all relationships and update the relationships based on what was sent. People, organizations, and places can be added from this endpoint if a new one was entered. All data is added and linked if no errors were thrown. For a person, organization, and place, the data is checked for errors and saved to the database if no errors were found. We then send a notification to all administrators.
- Delete document
  - Endpoint address: /documents/delete-document?docID=id, /person/delete?personID=id, /org/delete?orgID=id, /place/delete?placeID=id
  - All of these endpoints are relatively simple and similar and all require a user that is at least a metadata expert. We check to see if the entity we want to delete exists in the database. If it does, we delete it and create a notification for all administrators.
- Search/Browse for documents
  - Endpoint address: /documents/user-search?pageNum=int&pageSize=int, /documents/all?pageNum=int&pageSize=int
  - These endpoints can be accessed by any user. The first endpoint, the search functionality, takes in a mock document with the information that the user wants to search by. We first filter based purely on the metadata associated with a document. Then we filter the results based on the relationships that a document has, such as a person linked to the document. Once all that is sorted out, we sort the returned list based on an input value that is a part of the JSON sent to the endpoint. The counterpart to the search, the browse, is much more straightforward. We simply return all the documents in our database and sort them with the most recently added document showing up first.
- Get document data
  - Endpoint address: /documents/{id}
  - This endpoint takes in an integer that represents the id of the document that needs to be retrieved from the database. With this id, we simply search the database for a document with this id and return said document.
- Check out a document
  - Endpoint address: /transcriber/checkout, /editor/checkout, /encoder/checkout
  - Verify that the document has the proper status and make sure the user hasn't exceeded the check-out limit. Then create a DocumentEdit and set the userID to their ID. Tie the Document to the DocumentEdit. If the document has been worked on previously, set the documentText to the documentText of the most recent DocumentEdit associated with that document.
- Save work for later
  - Endpoint address: /transcriber/save, /editor/save, /encoder/save
  - Search the database for the DocumentEdit. This can be identified by the documentID and userID. Verify that the user has the document checked out currently and the document has the expected status. Then save the documentText to the database.
- Submit work
  - Endpoint address: /transcriber/submit, /editor/submit, /encoder/submit
  - Search the database for the DocumentEdit. Verify that the user has the document checked out currently and the document has the expected

status. Then save the documentText to the database. Change the status of the document to the next stage and disassociate the user with the document. Finally, create a notification for each admin.

- Return document
  - Endpoint address: /transcriber/return, /editor/return, /encoder/return
  - Search the database for the DocumentEdit. Verify that the user has the document checked out currently and the document has the expected status. Then save the documentText to the database. Change the status of the document to the previous stage and disassociate the user with the document. Finally, create a notification for each admin.
- Get list of all checked out documents
  - Endpoint address: /edits/{userID}
  - Query the DocumentEdit Repository for DocumentEdits with a matching userID and a completedDate of null. From this list, create a list of Documents that the DocumentEdits are attached to and return that list.
- Request a promotion
  - Endpoint address: /requests/make
  - Make sure that this request hasn't already been made and is still pending. Verify that the role is valid. Create a request with the userID and role and add it to the database.
- Change the status of a document
  - Endpoint address: /change-status
  - Verify that the status is one of the following: Needs Transcribing, Needs Editing, Needs TEI Encoding, or Completed. Find the document and change its status to this. If a user has the document checked out, return it. Create a notification that the status of the document has been changed.
- Change a user's role
  - Endpoint address: /roles/assign
  - Verify that the user is an admin. Delete all roles from the user that are greater than the new assigned role (if the user is being demoted). Add all roles that are less than or equal to the new role (if the user is being promoted). Create a notification that the user's role has been changed.
- Get list of all requests
  - Endpoint address: /requests/all
  - Query the Request Repository where granted is null. Sort them by dateRequested and return the list.
- Grant user a requested role
  - Endpoint address: /requests/approve
  - Verify that the user is an admin. Find the request in the database. Add all roles that are less than or equal to the new role. Set granted to true. Create a notification that the user's request has been approved.
- Deny user a requested role
  - Endpoint address: /requests/deny
  - Verify that the user is an admin. Find the request in the database. Set granted to false. Create a notification that the user's request has been denied.
- Deactivate a user
  - Endpoint address: /deactivate
  - Verify that the user is an admin. Return all of the user's checked out documents. Deactivate the user on FusionAuth. Create a notification that the user has been deactivated.
- Get list of all notifications
  - Endpoint address: /all
  - Find all Individual Notifications with the given userID. Sort the list in descending order by the individualNotificationID and return it.
- Mark all notifications as seen
  - Endpoint address: /mark-as-seen
  - Find all Individual Notifications with the given userID where seen is false. Mark all of them as true.

D. FusionAuth

FusionAuth provides developers with a direct API with opinionated measures for using the functionality of the application. Originally we were using this, however it became apparent that we had enough of our own necessities that we would benefit from developing a custom API ourselves.

We were able to pivot over to developing with their Java client library quickly. This enabled us to make a base url the source of all of our entire backend API. The decision simplified the configuration for both the backend and the frontend. All we had to do was add a maven dependency in our existing Spring Boot application and begin developing. This made it extremely simple and more performant for communication between the FusionAuth controller and PrintDB controllers. The FusionAuth client was injected as a spring bean and we were able to develop an encapsulated API for both the front end to call as well as a multitude of methods for the backend to use. FusionAuth was configured to use MySQL as its database as well, simplifying our infrastructure needs.

A design choice in FusionAuth is to abstract away the users from the application they are registered to. This enables larger scale operations to have a consistent user set with different user data attached to various separate applications. We opted to hide this design choice from the front end to simplify the approach for our use case. We recreated some of the basic API's such as logging in and creating users, as well as developed more advanced ones for our unique use cases.

Bundled with FusionAuth is Elasticsearch. This provides us with a powerful search engine to query user data based on any specific attributes. Our unique identifier for users is their email as well as an auto generated UUID. We were able to base many of our administrator only API's around the user's email instead of dealing with the UUID assigned to each user. This makes it more straightforward for the front end to make it user friendly for any administrative users. Dealing with user roles is a very common theme in this application, but with Elasticsearch we were able to make performant requests to promote, demote, or retrieve users based on their roles.

E. Containerization

The future of this project can go down multiple different paths regarding hosting and scaling. In collaboration with our sponsors we decided that containerizing the application with Docker would be a great hedge against the uncertainty. We were able to bundle the React frontend and Spring Boot backend into a single application with maven. With a simple "mvn clean install" command at the top of the backend directory you have generated a runnable front end and backend contained within a single jar file. An accompanying Dockerfile with some configurations builds a Docker image. All that's left to run the entire application is a docker-compose.yml file that will run the various images required. Docker will auto pull images from DockerHub on a first time run too. One other image needed is MySQL with a Docker Volume. The volume is to maintain persistent data even if the MySQL container goes down or is restarted. The other image needed is the FusionAuth base application and Elasticsearch. All that is required to run these containers together is Docker installed on a host machine. The portability that this provides completely solves our problem and makes the application easy to hand off.

V. Conclusion

The crowdsourcing transcription platform delivers on its promise of being an intuitive application for the benefit of public history. It abstracts away complexity and provides a single source of data for the public as well as a tiered role system for future contributors. This allows everybody involved to focus on their own role in public history. The leaders will be able to dedicate more time to being engulfed in their passion, the history, and less time in administrative and organizational efforts. Participants will be able to focus on their education while avoiding the mess of finding copies of the information they are seeking and passing off their work when the time comes. Our queue-like system will allow transcriptions to smoothly flow through the stages of completion and prevent unnecessary duplications. Non-participant users will still be able to view the public data they are entitled to and hopefully learn something new about our past.

References

[1] "FusionAuth v1 Documentation", Fusionauth.io, 2020. [Online]. Available: https://fusionauth.io/docs/v1/tech/. [Accessed: 20- Apr- 2020].

[2] "Shelley-Godwin Archive", Shelleygodwinarchive.org, 2020. [Online]. Available: http://shelleygodwinarchive.org/. [Accessed: 20- Apr- 2020].

[3] "Stanford University Archives | FromThePage", Fromthepage.com, 2020. [Online]. Available: https://www.fromthepage.com/stanforduniversityarchives. [Accessed: 20- Apr- 2020].