

Group 3

Audio-Transcript Database Implementation

Summer Martin - [martis36@tcnj.edu](mailto:martis36@tcnj.edu)

Jared Schmidt - [schmij12@tcnj.edu](mailto:schmij12@tcnj.edu)

Kyla Ramos - [ramosk10@tcnj.edu](mailto:ramosk10@tcnj.edu)

Lalima Bhola - [bholal1@tcnj.edu](mailto:bholal1@tcnj.edu)

Aly Maahs - [maahsa1@tcnj.edu](mailto:maahsa1@tcnj.edu)

## **Executive Summary**

While researching the Trenton Library and the Trentoniana Project, our group realized a simple need: accessibility. Currently, the users of the Trentoniana website have a difficult time trying to utilize the site to find audio files and the transcripts associated with them. It is disorganized and hard to navigate. For example, there is no way for users to search for audio files based on attributes such as specific dates, names or topics. This alone makes researching the history of Trenton a difficult task. Even further, the Trentoniana website solely has audio files. There are no transcripts or captions. This makes it extremely difficult or even nearly impossible for people who are deaf, hard of hearing or auditory processing impaired to utilize the files, and makes having transcripts a priority for this group of users and so many more. Moreover, we found that there isn't a way for visitors to address problems and make suggestions to the site. This also creates a problem for the moderators of Trentoniana; without a way to know what problems exist, there is no way for them to solve it. The need for user accessibility and easy navigation was the need that drove our project to completion.

Our approach to solving these accessibility problems was to create a user friendly application with an easy to use search feature that easily allows someone to search for transcripts and audio files together, as well as allow administrative users to identify any problems, gain analytics on which transcripts are popular, and update the website as needed. We also created a database schema to store these transcripts, as well as other data such as view counts or comments. Audio-transcript files are tagged with topics, names, dates for a convenient search option as well as a search bar on the main search page. Users can search by name, date, tags which are based on the topic of the file and like count or view count. Users can easily see all the comments other users have made to an audio file and easily make comments on a transcript themselves. Users can also make suggestions on our readily available suggestion page. Overall our approach was to make the transcripts as easily accessible as possible and searchable by as many means as possible. Users with hearing impairments now have access to the transcripts and all users can benefit from and utilize the search, comment, suggestion and transcript features.

There are many benefits to our solution. Firstly, compared to the previous setup, our application is composed of an elegant but simple and straightforward UI, making it easy for any users to utilize the website to its full potential. With our database schema, we have created a way for audio-transcript files to be tagged with topics, names and dates for convenient search options. We have also provided a way for users to comment on and like transcripts, allowing Trentoniana moderators to know which transcripts are popular, helping them to possibly create programs or promotions of those pieces of history while also providing a way for users to know which transcripts are of value to others, helping them in their research. Additionally, more popular transcripts can be prioritized for suggested changes or updates. Lastly, unlike competitors, one can easily know what updates are needed for the website because of user submission boxes for new audio files, records, or suggestions in general. While our implementation does have similar features to other websites, our user interface is simple and does not clutter up the space nor is it

overloaded with extra unnecessary features which could cause the user to become confused or overwhelmed with options.

The cost to the stakeholder would be very minimal. All the software we used is open source and it would only need to be installed on the users local machine. Our application would be an extension of the Trentoniana website created to hold the transcripts of the audio files instead of the current link that leads to only the audio files on archive.org without the transcripts. Thus the only cost would be having to have a server to store the data of the transcripts. The stakeholder may also need administrative users and moderators to oversee the database and implement suggestions made by other users.

### **Elaboration: Project Proposal and Specifications**

Revisions and additions from Stage IIa to Stage IIb are underlined. These revisions were made to add detail or further clarity. No revisions were necessary after the Stage IIb submission.

- Problem statement.

The Audio & Visual section of the Trentoniana website is currently a work in progress. There is a way for people to access the audio files and our classes have the transcripts of the audio files, but there are several problems with the current system. To start, the audio files are not easily accessible. They are hosted on a page that isn't user friendly and they are not easy to navigate to even when you are looking for them. The audio files are also not linked to the transcripts in any way and the transcripts are not currently accessible to the public. Furthermore, neither the audio files, nor the transcripts, are organized and there is no way to search for specific topics, dates or names. A few of the audio files are titled with names, but there is no other identifying information about the topic and several appear to only have a number. The files are also not tagged or labeled in any way that would make searching easier. Additionally, currently only having audio files makes it difficult or impossible for those with hearing impairments or those with auditory processing and other disabilities to use the files. Finally, there is no way for people to submit suggestions or concerns for newly discovered files, changes or errors. The system needs an upgrade where these problems are addressed and fixed so that users have a better experience when navigating the files and using the site.

- Objective of the module.

The objective of our project is to create and provide a more efficient and user friendly system for searching and finding audio files on the topics the user is looking for. It is also our goal to combine the audio field and the transcripts so they are both accessible to the public and have the ability to view the transcript that corresponds to the audio file while the audio is playing. We want to provide a user friendly interface for searching and viewing these files based on desired topics, years, or names. Providing an straightforward way to interact with these files

and a system that would allow for public users to submit new files to be transcribed would encourage interaction between the user and those behind the scenes to collaborate and improve upon existing systems. Better user interaction makes the user feel like their suggestions are valued and their interaction with our system is important. Also, adding the transcripts would make the system more inclusive for those with hearing impairments who would not be able to listen to the files. We also want those with higher access to the database to be able to improve and make changes, like adding relevant tags, information and new files, to the system more easily.

- Description of the **desired** end product, and the part you will develop for this class.

The audio-transcript files would be sorted and arranged in a neat, elegant user friendly user interface and would be tagged or labeled with topics, names, dates, and other relevant information so they would be able to be searched for more easily, made easy through tagging the files with relevant information. Each audio file would also display the number of views each file has had and the number of comments and “likes”. There would be a search bar at the top of the page so users could easily search for their desired topic or a specific year. If selected, the user would be taken to a different page where they could listen to the audio file and have the option to view the transcript as a sort of drop down option, where the transcript is hidden until the user indicates they would like to see it. When viewing the transcript, the user would see the entire transcript at once, maybe with the current part, the part being read at the time, highlighted. On the main page there would be a link to the transcript below each audio file in case the user would rather only view the transcript. There would be some feature on the main page, towards the bottom, which would be a submission box for new audio files, records, or suggestions. For example, if someone found an error in the transcript or wanted to make a suggestion for improvement or a new submission to be transcribed.

**\*\*Note:** while this was the original idea and overall goal that we based our final product after. The final product did not meet these exact specifications due to changing our minds as we created the product and our actual capabilities as to how far we could go and what we had time to do. This was not updated/ revised in the document since we felt this was the original proposal and ideas do tend to change after the original concept is thought up and proposed, so to show what our original intention was we kept the proposal as it was.

- Description of the importance and need for the module, and how it addresses the problem.

Since the current system only contains the audio recordings, but not on a main page and only in a way that you have to go through several pages to access them, we want to add the audio files to the main page and include transcripts to make information more easily accessible. A user may not want to listen to the entire audio file to find information so having a transcript will be helpful for that reason. For other users, it may be difficult to only listen to an audio file and a

transcript will help them follow along better. For hearing impaired users, listening to the audio file may be impossible or extremely difficult. Our module will allow users to view a transcript as the audio file plays or separately if desired. The module can also include a search function, a search bar, to make it easy to find a specific audio file since, at the moment, there is no way to know what is contained in an audio file until you listen to it. This application addresses the issues of user inaccessibility. We also want to make it visually appealing through our user interface so people want to explore the history of Trenton and promote the website itself. Our program will make the files easily accessible, searchable, user friendly, and better organized.

- Plan for how you will research the problem domain and obtain the data needed.

In order to research our problem domain and obtain the needed data we will examine similar applications of audio file use, accessibility and the correlating transcripts to see how these methods work, which work best and how we could improve on existing ones. We will see if some platforms ever have performance issues and do research as to why these issues occur and do our best to avoid them. We will also research other websites that host historical transcripts like historical archives, and see how they present information. We may look at other libraries' websites to see how their user interfaces are set up and presented for public use and make sure we allow for similar ease of access and user friendly methods for viewing historical files and transcripts.

- Other similar systems / approaches that exist, and how your module is different or will add to the existing system.

In sites such as ted.com, they have the audio playing while the corresponding transcript appears below the video. Our site will have a similar feature where users will be able to follow along to the transcript as the audio file plays. They would also be able to click a specific instance of the transcript, which would take them to that moment in the audio. Our module can also include a feature for displaying views, likes, and comments for the audio files and transcripts. By allowing users to like and comment on data, over time, the searches might be able to be filtered by "popular" or "top" files. The comment section would allow for debate or conversation on varied topics. This would create a better sense of community within the Trenton area and users of the Trentoniana site. Some sites that host historical audio files do not include transcripts in their system. Our method will add to systems like those by providing the transcripts for all of the audio files and allowing users to follow along with them while they listen.

- Possible other applications of the system (how it could be modified and reused.)

Since our module will allow for greater user accessibility, it can be applied to other locations where there is a lack of inclusivity in the distribution of information. For example, sites containing audio/video files without subtitles or transcripts will limit those who are hard of hearing or deaf from proper access to this data. The module will also contain a clear user interface/place to enter searches based on categories. Once implemented, the module can transfer

to similar sites to allow for a comprehensive user base. Our module could be modified for any set of data or files with need of an organized, user friendly system, especially for those pertaining to audio files.

- Performance – specify how and to what extent you will address this.

In order to optimize performance, we will create precise methods of querying and sorting data. For example, audio files should have the option of being sorted by length as well as content. Transcripts will have concise queries and filters/tags to allow users to find their data more accurately. We can also allow users the option of filtering data by exclusion. This can include excluding specific years or keywords from their search. Another option, instead of a direct search box, is to designate fields for an individual's first name, last name, year, length of video, etc. This would also assist in creating an accessible user interface. We will ensure our module has a reasonable performance level to guarantee an optimal user experience. To further determine if our module has made a significant improvement as compared to the current system, we will host user testing sessions in which we ask users to perform the same task on both systems. We will then ask the users which version they prefer to use and why. With the data we collect from our user tests, we will be able to quantifiably determine if our module performs before, as well as make any changes that users suggest to improve our project even further.

- Security – specify how and to what extent you will provide security features.

To provide security features for this application, we would set up a User ID and password function for the managers of the system to give them access to more advanced and private features of the database. These would be stored in the database if or when a user chooses to make an account. To provide security for the user, the passwords themselves would be encrypted so they are not stored in a plain text form (which would be a large security risk should that information be leaked in some way). \*\*

We would implement three types of user for the application: basic user, moderator, and administrator. (However, the only two users that actually require the use of a user ID and password are the moderator and administrator.) This basic user would have the least amount of power, and not be able to do more advanced things such as directly edit or delete a transcript. They would however have access to the general functions of the application such as suggesting an edit to a transcript.

The second tier of account would function as a moderator. These have more power than a basic user, but still do not have as much as the administrator of the app. They would have the ability to do more advanced things such as review any edits made to transcripts and either accept or deny them, as well as moderate a basic user's activity in the event they are being harmful towards others on the site. However, they wouldn't have direct access to the database unless given permission to by the administrator of the system.

This administrator would be the highest tier of user. They have all the permissions of the previous tiers, as well as having direct access to all database information. They can also create moderator accounts for other users to utilize, and can choose whether or not those moderators will have access to the database as well.

**\*\*Note:** We ended up deciding that hashing the passwords would be better than encrypting them, but we weren't able to implement this for our final web app. If this implementation was going to be actually used, we could use python/flask for this and use a hashing function like bcrypt to hash and store the passwords in the database, as well as when the user would try to sign in.

- Backup and recovery – specify how and to what extent you will implement this.

We will make sure that there are private backup copies of all the files inaccessible to regular users in case of file corruption or deletion. Since they are backed up, they can be recovered and reuploaded by those with higher access. New files will be automatically copied to a secure location which would not be accessible to the public and normal users. In the case of file corruption, which would have to be detected by a user and noted by a system manager or high level user, the copied file would have to manually be reuploaded to the site.

- Technologies and database concepts the team will need to learn, and a plan for learning these.

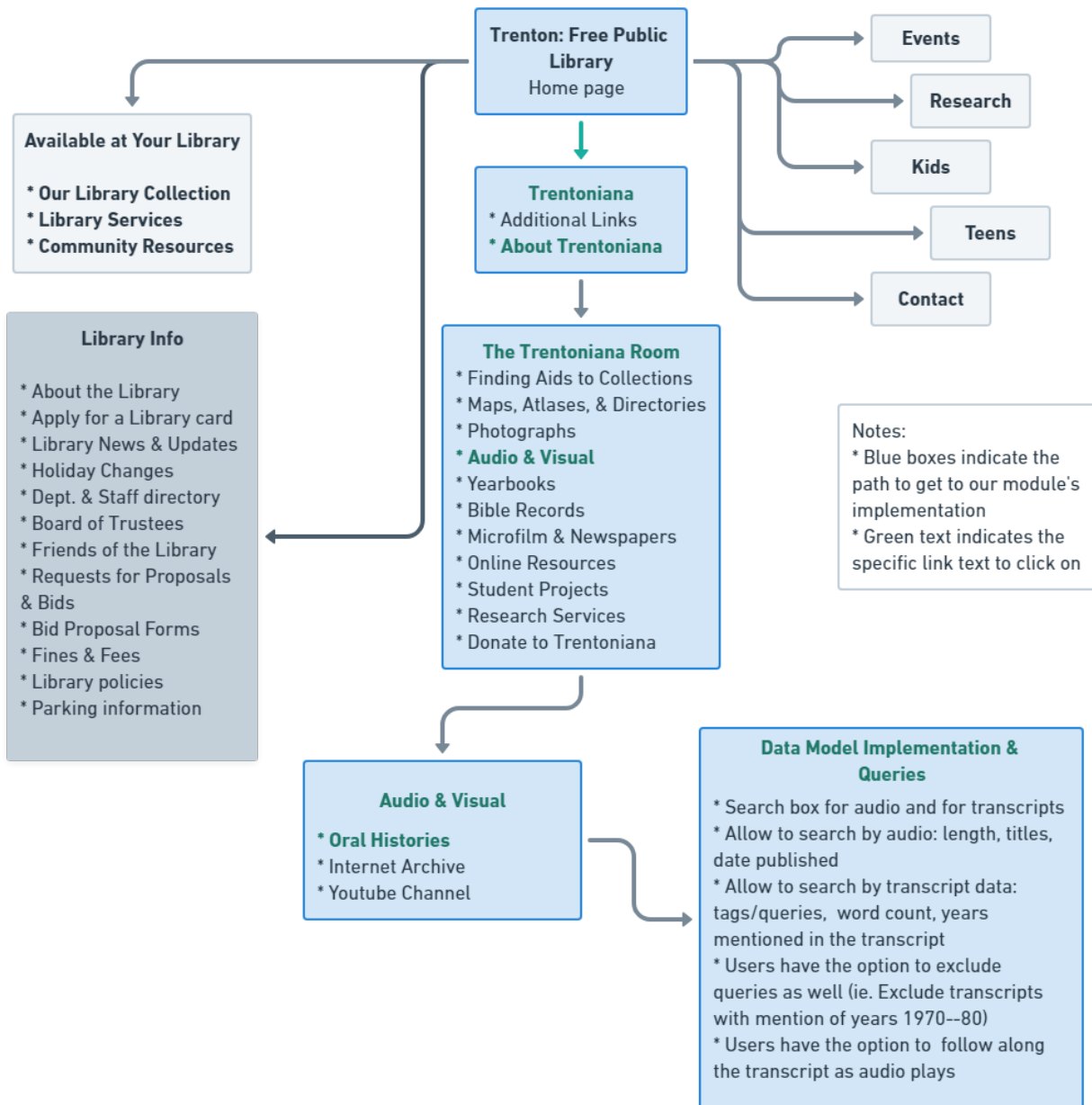
We will need to utilize PostgreSQL to create the database. In creating the web application for the project, we may also need to utilize PHP to link our database to the website. We will also utilize Figma to create prototype designs of what we want our final project to look like, and following that utilize HTML and CSS to carry out the look and feel we finalized for our web UI.

To accomplish our design goals, we will use online tutorials as well as the prior knowledge or experience of members on our team. A member of our team is currently learning about web design in a class and another completed a class on web design last semester as well as having prior design experience. The rest of the group members can learn from these two team members as well as from any online resources such as w3schools to carry this out.

There are several database concepts our team will need to learn in order to properly implement our program. Some of these concepts include what it means to put something into a database, how to implement security measures in a database, how to backup a database and its contents and make sure they are recoverable. We would also need to learn what the structure of a database is and how to implement one in code.


**\*\*Note:** While we originally did plan to and consider using PHP or Figma to create our final product, we ended up using Python and Flask. We did not revise that in the original proposal document because we felt that it was a proposal to outline our original goals and ideas and not set in stone.

- A diagrammatic representation of the system boundary that specifies what data you will model and which queries you will implement.





- 1-page quad chart;

 <b>Audio-Transcript Database Implementation</b> Group 3: Lalima Bhola, Aly Maahs, Summer Martin, Kyla Ramos, Jared Schmidt	
<b><u>Need</u></b> <ul style="list-style-type: none"> <li>• A better way to access the audio files from Trentoniana website as well as access to the transcripts for the audio files.</li> <li>• Customers with audio and hearing disabilities needs include transcripts for those.</li> <li>• An organized and sorted view of all the audio files with a search capability.</li> <li>• User friendly interface with better accessibility.</li> </ul>	<b><u>Approach</u></b> <ul style="list-style-type: none"> <li>• Have the audio files arranged in a unique, elegant user interface and have the audio files labelled and tagged based on topic</li> <li>• Have a search bar so the users can search for a specific topic or a specific date</li> <li>• An option for the user to view the transcript by itself or with the audio, potentially highlighting the text to match along with the audio</li> <li>• Give the user the ability to recommend new audio files to be transcribed or changes to be made</li> </ul>
<b><u>Benefit</u></b> <ul style="list-style-type: none"> <li>• Having a database for these Trentoniana audio files that is much better than the current Internet Archive</li> <li>• Making the transcripts available to the public</li> <li>• The program will support those with hearing and auditory processing disabilities by including the transcripts making our system inclusive and comprehensive.</li> </ul>	<b><u>Competition</u></b> <ul style="list-style-type: none"> <li>• Make our database look better and be easier to use than other implementations</li> <li>• Will be much better than the current Internet Archive's implementation as ours will have the transcripts available and have the audio files sorted, as well as having a search feature (where users can search by audio and/or transcript data)</li> </ul>

02/10/21

### Elaboration: Design

This consists of the materials submitted for Stages III and IV, with revisions clearly identified.

#### Stage III

##### Initial database size (approximate number of records)

- About 74 audio files

##### Types and average number of searches

- Types of Searches
  - Location
  - Religion
  - Year
  - Name
- Average number of searches
  - Maximum of 25 queries per day, most likely around an average of 10-15 queries

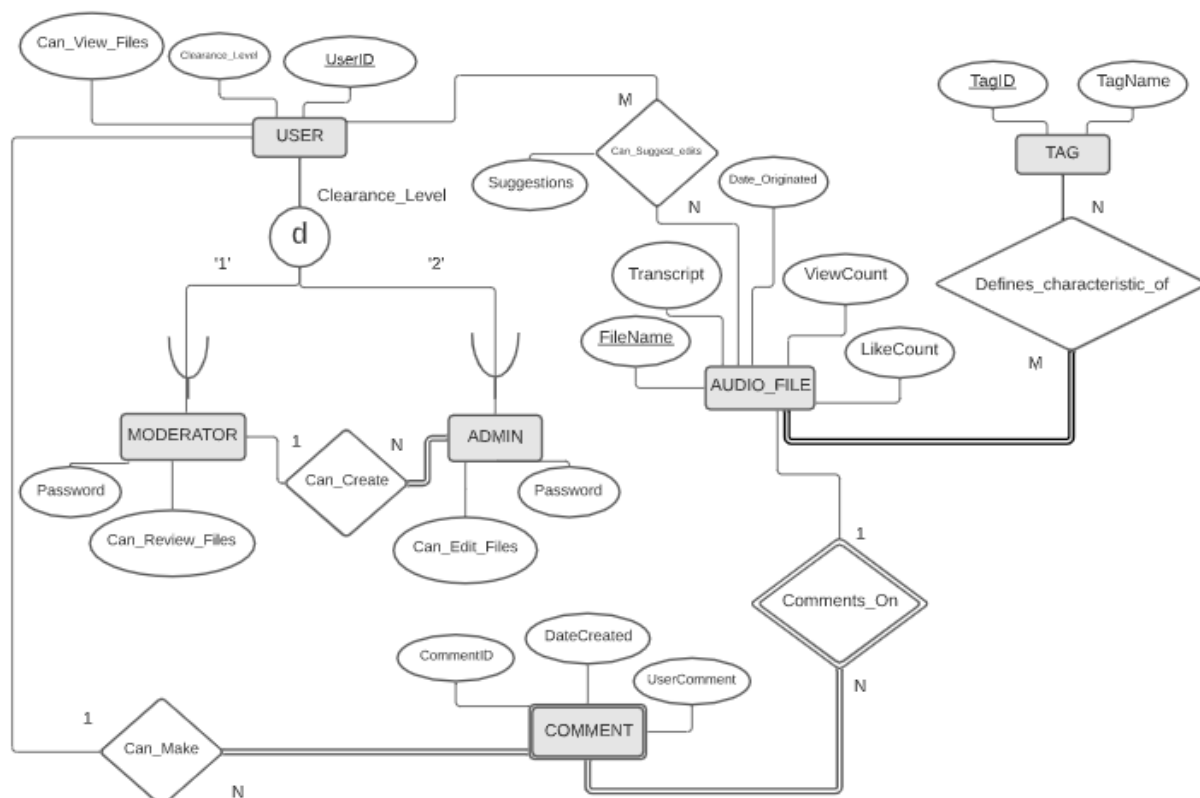
## Relationships:

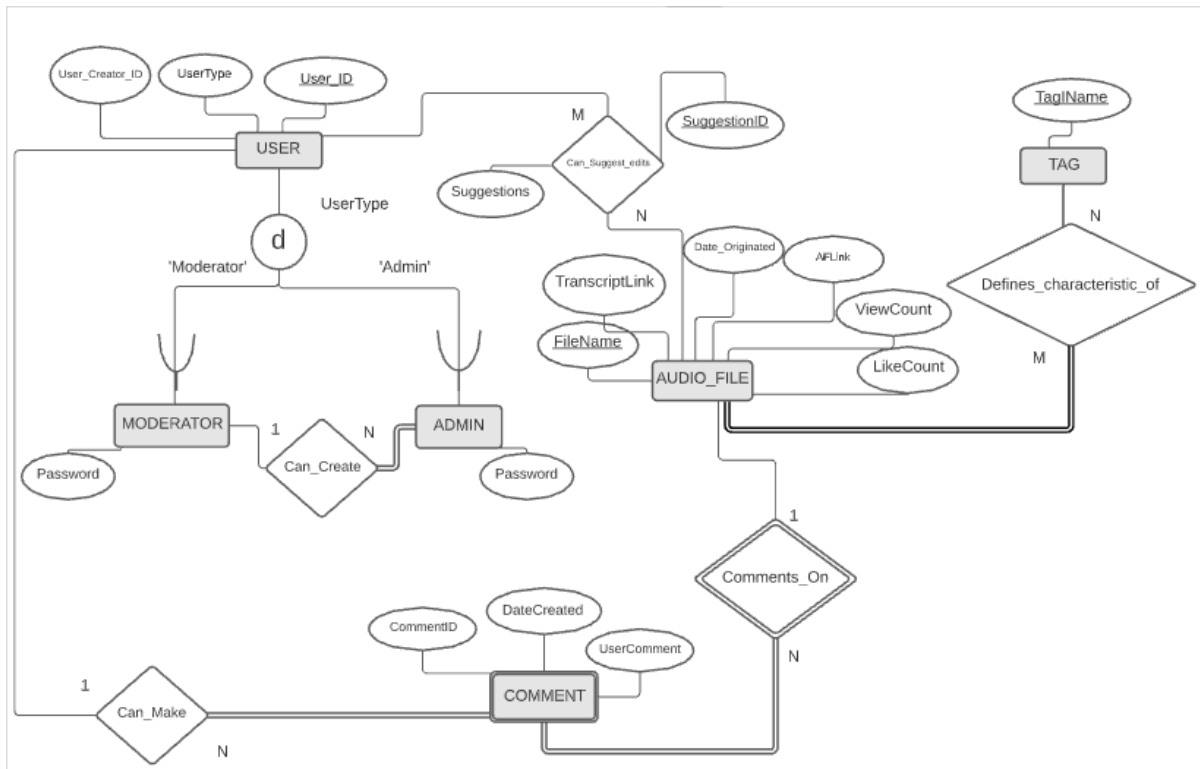
- Users CAN MAKE Comment
  - 1:N
- Comment COMMENTS ON Audio\_File
  - 1:N
- Users CAN SUGGEST EDITS ON Audio\_File
  - M:N
- Admins CAN CREATE A Moderator
  - N:1
- Tag DEFINES CHARACTERISTIC OF Audio\_File
  - M:N

## Documents:

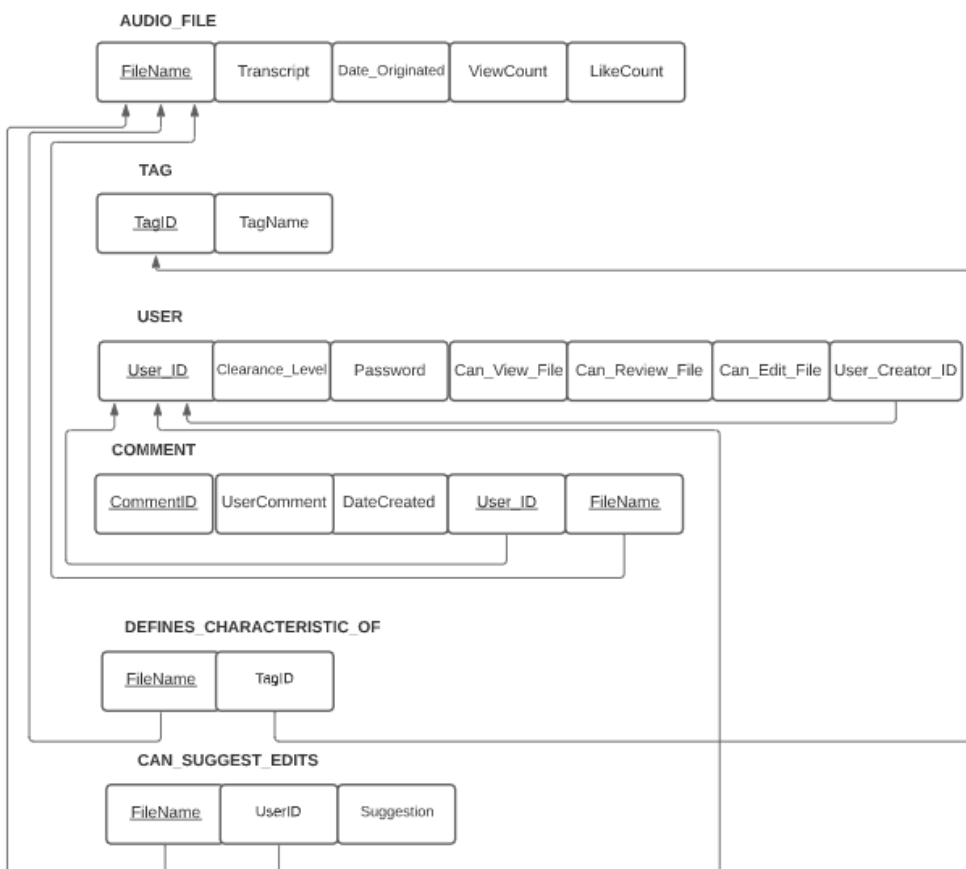
### ER Diagram

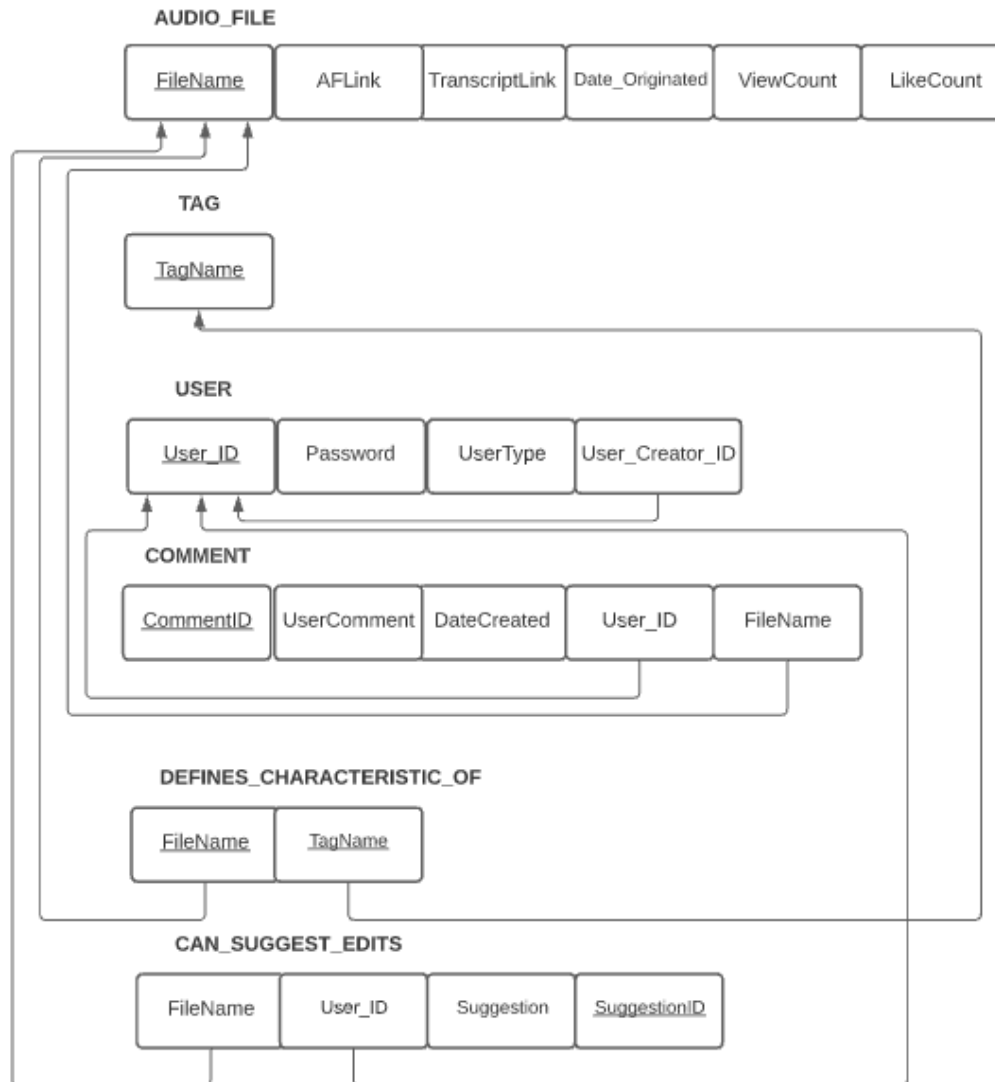
The first image is our original ER diagram and the second image is our final ER diagram with revisions. The changes that were made from the original version to our final include changing the transcript attribute to a transcript link attribute and adding in the audio file link- afile attribute. TagID was removed as an attribute from Tag as it was unnecessary and TagName was made the primary key. Several attributes were removed from the user table because they were unnecessary. File\_Name was removed as the primary key from the Comment table because it was not needed in addition to having CommentID. Lastly, in the table Defines\_Characteristic\_Of, TagID was replaced with TagName which was also made a part of the primary key and SuggestionID was added to Can\_Suggest\_Edits and made the primary key.





## Relational Schema:





The first relational schema image shows the original relation schema and the second shows the final relation schema with revisions. The changes that were made from the original version to our final include changing the transcript attribute to a transcript link attribute and adding in the audio file link- aflink attribute. TagID was removed as an attribute from Tag as it was unnecessary and TagName was made the primary key. Several attributes were removed from the user table because they were unnecessary. File\_Name was removed as the primary key from the Comment table because it was not needed in addition to having CommentID. Lastly, in the table Defines\_Characteristic\_Of, TagID was replaced with TagName which was also made a part of the primary key and SuggestionID was added to Can\_Suggest\_Edits and made the primary key.

## Stage IV

### Normalization:

Demonstrate that all the relations in the relational schema are normalized to Boyce–Codd normal form (BCNF).

- For **each table**, specify whether it is in BCNF or not, and explain why.

- **AUDIO\_FILE**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.

-- It is in second-normal form because all of the non-prime attributes are fully functionally dependent on the primary key FileName (which is the only primary key) and there are no partial dependencies. It is also fully functionally dependent on the candidate keys transcript link and audio file link.

-- It is third normal for because there are no transitive dependencies.

-- It is in BCNF because for any functional dependencies, the left side is a super key. For example, either FileName, the primary key, or Transcript, the unique key, is a superkey.

- **TAG**

-- It is in first normal form because each column is unique and contains one element. All of the attribute values are indivisible.

-- It is in second normal form because there is only one attribute which is the primary key TagName. There are no non-primary attributes or other candidate keys.

-- It is in third normal form because TagName is the only attribute so there are no transitive dependencies.

-- It is in BCNF because TagName is the only attribute.

- **USER**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.

-- It is in second normal form because there is only one attribute used as the primary key, User\_ID and all of the non-prime attributes are fully functionally dependent on User\_ID

-- It is in third normal form because there are no transitive dependencies. None of the non primary key attributes could determine any of the other attributes.

-- It is in BCNF because besides the primary key, User\_ID, there are no other prime attributes and no functional dependencies that do not have the primary key as the super key.

- **COMMENT**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.

-- It is in second normal form because the primary key contains only one attribute, CommentID and all of the other non-prime attributes are fully functionally dependent on Comment ID.

UserComment could not be a candidate key because there could potentially be two of the same comment.

-- It is in third normal form because there are no transitive identities. None of the non primary key attributes could determine any of the other attributes.

-- It is in BCNF because there are no functional dependencies that do not have a superkey on the left side. There are no prime attributes other than the primary key and no functional dependencies where a prime attribute is dependent on a non superkey.

- **DEFINES\_CHARACTERISTIC\_OF**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.

-- It is in second normal form because FileName and TagName are the only attributes and they are both part of the primary key. You must have both to determine the instance of Defines\_Characteristic\_Of. So, because there are no non-prime attributes, we can say that all of the non-existent non-prime attributes are fully functionally dependent on the primary key.

-- There are no attributes that are not part of the primary key so there are no transitive dependencies and as such, it is in third normal form.

-- It is in BCNF because there are only two attributes, both of which are needed for the primary key, so the relation only has the primary key. There are no functional dependencies where the left side is a non superkey and the right is a prime attribute.

- **CAN\_SUGGEST\_EDITS**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.

-- It is in second normal form because the non-prime attributes are fully functionally dependent on the primary key SuggestionID.

-- This is in third normal form because there are no transitive dependencies. For example {FileName, User\_ID} could not determine Suggestion and so neither of them alone could either. Suggestion also could not determine {FileName, User\_ID} or either of them separately.

-- It is in BCNF because there are no other prime attributes, other than the primary key, because theoretically, a user could leave the same suggestion on an audio file twice or more times. As such, the only prime attribute is SuggestionID and none of the other attributes, together or apart, could determine SuggestionID for the reason stated previously. This means that there are no nontrivial functional dependencies where the left side is not a superkey.

For each table that is not in BCNF, show the complete process that normalizes it to BCNF.

- We did not need to normalize any of our tables because they were already in BCNF.

## **Views:**

Define the different views (virtual tables) required. For each view list the data and transaction requirements. Give a few examples of queries, in English, to illustrate.

- Admin can: Select, Insert, Update, and Delete from any Table
  - For example, they can insert a new user into the user table, they can change the user type of a user, they can delete an audio file or update the actual audio file or transcript file, and they can select all the users they have created from the user table by selecting the tuples where their user id was the user\_creator\_id.
- Moderator can: Select, Delete Comments, Delete and manage suggestions, Update some attributes of audio files but not update the transcript or actual audio file, Insert/Update Tags
  - For example, they can delete comments from any file, delete suggestions from user suggestions, insert new tags for audio files, and update tags for audio files to keep the data relevant
- General User can: select and view all audio files, make suggestions, insert comments, update their own comments and suggestions.
  - For example, they can select audio files based on different parameters like view count or a certain tag or attribute, they can insert a new suggestion tuple for some audio file, they can update or delete a comment or suggestion they have made for some audio file.
- We will need the view Audio\_File\_Tags with the data from the tables Audio\_File joined with Defines\_Characteristic\_Of to do the transaction to search for/select all of the audio files and their information for a particular tag.
  - CREATE VIEW Audio\_File\_Tags  
AS SELECT \*  
FROM AUDIO\_FILE AS AF \* DEFINES\_CHARACTERISTIC\_OF AS DC  
WHERE AF.FileName = DC.FileName
  - For example, they could use the view to search for all of the audio files with a date originated before 2000 that have the tag education.
- We will need the view User\_Comments for the data from the tables User joined with Comment if we want to do the transaction to see all the comments a user has made and see the users type or other information at the same time.
  - CREATE VIEW User\_Comments  
AS SELECT \*  
FROM USER AS U \* COMMENT AS C  
WHERE U.User\_ID = C.User\_ID
  - For example, they could use the view to see all of the comments made by moderator users.
- We can create the view Audio\_File\_Comments for the data from the tables Comment and Audio\_File to do a transaction to select the tuples with all the comments made on a particular file and we want to see all the information for that file as well.

- CREATE VIEW Audio\_File\_Comments  
AS SELECT \*  
FROM AUDIO\_FILE AS AF \* COMMENT AS C  
WHERE AF.FileName = C.FileName
- For example, you can use this view to Search all the comments made on Audio\_File with more than 100 likes.
- We can have a view Creator for the data from the tables User joined with User where User\_Creator\_ID = User\_ID to do a transaction to select all the tuples with users that were been created by another specific user (Admin) if we want to check the information of both the creator and the users they are creating.
  - CREATE VIEW Creator  
AS SELECT \*  
FROM USER AS U JOIN USER AS U2  
WHERE U.User\_ID = U2.User\_Creator\_ID
  - For example, you can use this view to see how often a certain admin makes other admin users, or how many moderators they have made.
- We can have a view Suggest\_File\_Edit for the data from the tables from Can\_Suggest\_Edits joined with Audio\_File to do a transaction to select all the suggestions made for Audio\_File and the Audio\_File information so we can edit/update the file based on suggestions.
  - CREATE VIEW Suggest\_File\_Edit  
AS SELECT \*  
FROM AUDIO\_FILE AS AF \* CAN\_SUGGEST\_EDITS AS CSE  
WHERE AF.FileName = CSE.FileName
  - For example, you could use this view to see all the suggestions made on files with view counts > 100 so that you can work on those first because they get the most traffic

### **Queries Submitted for Stage IV:**

#### **Select Queries:**

```
SELECT * FROM Audio_File_Tag WHERE Date_Originated < 2000 AND TagName = 'Education'
```

```
// Select audio file by tag
```

```
SELECT FileName FROM Audio_File_Tag WHERE TagName = 'Education';
```

```
// Education is an example. You can replace the word education with any (relevant) word
```

```
// Find all the tags an audio file has
```



```
SELECT FileName, Tag FROM Audio_File_Tag WHERE FileName = <file name>;
```

```
// Select audio file by view count
```

```
SELECT FileName FROM AUDIO_FILES WHERE ViewCount < 100;
```

```
// ViewCount < 100 is an example. You could replace the where condition to be any relevant
```

```
// amount of views, such as > 100 as another example.
```

```
// Select audio file by like count
```

```
SELECT FileName FROM AUDIO_FILES WHERE LikeCount < 100;
```

```
// LikeCount < 100 is an example. You could replace the where condition to be any relevant
```

```
// amount of likes, such as > 100 as another example.
```

```
// Select audio file by date originated
```

```
SELECT FileName FROM AUDIO_FILES WHERE Date_Originated = '1/12/1950';
```

```
// Date_Originated = '1950' is an example. You could replace the where condition to be any
```

```
// relevant origin date such as 'February 1920' as another example.
```

```
// Find the number of comments an audio file has had.
```

```
SELECT COUNT (*) FROM Audio_File_Comments WHERE FileName = 'World War II';
```

```
// FileName = 'World War II' is an example. You could replace the where condition to be any
```

```
// relevant file, such as 'Economy' as another example. You could also use a different attribute
```

```
// such as LikeCount to see the comments for files with a specified amount of likes
```

```
// Find the total number of views of all audio files
```

```
SELECT SUM(ViewCount) FROM AUDIO_FILE;
```

```
// Search for by user type to get all the admin users or moderators.
```

```
SELECT * FROM USER WHERE UserType = <admin or moderator>;
```

```
// Search Comment by User_ID to find all the comments made by some user.
```

```
SELECT * FROM User_Comment WHERE UserType = <desired type>;
```

```
// Search for a Comment by FileName
```

```
SELECT * FROM COMMENT WHERE FileName = <file name>;
```

```
// Search the Audio_File_Comments by FileName (a view created in the previous part) to find all
```

```
// the comments for a file and all the audio file information.
```

```
SELECT * FROM Audio_File_Comments WHERE FileName = <file name>;
```

```
SELECT * FROM Audio_File_Comments WHERE LikeCount > 100;
```

// Search Comment by date created to get all the comments from one day.

```
SELECT Comment FROM COMMENT WHERE DateCreated = <date>;
```

// Find all the suggestions made on an Audio File with > 100 views

```
SELECT File_Name, Transcript, Suggestion FROM Suggest_File_Edit WHERE ViewCount > 100;
```

// Find all the users made by some other user

```
SELECT User_ID FROM Creator WHERE User_Creator_ID = <user ID>;
```

```
SELECT * FROM Creator WHERE U.UserType = U2.UserType;
```

### **Insert Queries:**

```
INSERT INTO TAG (school) VALUES ('Education');
```

// Education is an example. You can replace the word education with any (relevant) word

```
INSERT INTO AUDIO_FILE VALUES ('World War II', '<transcript file here>', '9/15/1980', '1000', '999');
```

```
INSERT INTO COMMENT VALUES ('004567', '<usercomment here>', '4/1/2021', '001234', 'World War II');
```

```
INSERT INTO USER VALUES ('000010', 'Y$pU1gZdaH', 'Moderator', 'collegeStudent04');
```

// User\_ID would either be default or serial. Only Admin users can make a user admin or a

// moderator (see update user)

```
INSERT INTO DEFINES_CHARACTERISTIC_OF VALUES ('Cold War', 'history');
```

```
INSERT INTO CAN_SUGGEST_EDITS VALUES ('Gubernatorial Election', '000030', '<suggestion here>');
```

### **Delete Queries:**

```
DELETE FROM AUDIO_FILE WHERE FileName = 'World War II' ;
```

// FileName = 'World War II' is an example. You could replace the where condition to be any

// relevant file name, such as 'Economy' as another example.

```
DELETE FROM CAN_SUGGEST_EDITS WHERE SuggestionID = <some number>;
```

// Delete tuple once suggestion has been handled.

```
DELETE FROM TAG WHERE TagName = 'Education';
```

```
DELETE FROM USER WHERE User_ID = '001234';
```

```
/* Delete a user account in case an account is no longer necessary, for whatever reason.*/
```

```
DELETE FROM DEFINES_CHARACTERISTIC_OF WHERE FileName = <name of file>  
AND TagName = 'Education';
```

```
/* removes a tag from an audio file, education is an example*/
```

```
DELETE FROM COMMENTS WHERE User_ID = '004567';
```

```
/* Delete all comments of a certain user. */
```

```
DELETE FROM COMMENTS WHERE FileName = <name of file>;
```

```
/* delete all of the comment for some audio file*/
```

```
DELETE FROM COMMENTS WHERE CommentID = '000123';
```

### **Update Queries:**

```
UPDATE USER
```

```
SET UserType = <Admin, Moderator, General>
```

```
WHERE User_ID = <user_ID number>;
```

```
/* update usertype for a certain user */
```

```
UPDATE AUDIO_FILE
```

```
SET Transcript = <Transcript>
```

```
WHERE FileName = <name of file>;
```

```
/* update transcript for a certain audio file */
```

```
UPDATE COMMENT
```

```
SET UserComment = <updated userComment>
```

```
WHERE CommentID = <ID of comment>;
```

```
/* update UserComment content for a specific comment (users can edit their own comments after  
posting) */
```

```
UPDATE USER
```

```
SET Password = <new password>
```

```
WHERE User_ID = <user_ID number>;
```

```
/*updates the password of a certain user*/
```

```

UPDATE CAN_SUGGEST_EDITS
SET Suggestion = <edited suggestion>
WHERE SuggestionID = <suggestionID number>;
/* updates a user's suggestion if they want to make changes to something they said*/

```

### **Construction: Tables, Queries, and User Interface**

This consists of the submissions for Stage V, with revisions clearly identified.

#### **Stage Va**

#### **DDL**

/\*The following commands create all the tables in our database. They specify the attributes and their data types, the primary key, foreign keys and what they reference, any checks necessary, default values, not NULL constraints and what to do ON DELETE and ON UPDATE \*/

```

CREATE TABLE AUDIO_FILE (File_Name text, AF_Link varchar (2048), Transcript_Link
varchar (2048), Date_Originated int CHECK ((Date_Originated > 1800) AND (Date_Originated
< 2050)), View_Count int DEFAULT '0', Like_Count int DEFAULT '0', PRIMARY KEY
(File_Name));

```

```

CREATE TABLE TAG (Tag_Name varchar (15) PRIMARY KEY);

```

```

CREATE TABLE USERS (UserID SERIAL PRIMARY KEY, Password varchar (255),
User_Type varchar (9) CHECK (User_type in ('Admin', 'Moderator', 'Guest') ) DEFAULT
'Guest', User_Creator_ID int DEFAULT '100000', FOREIGN KEY (User_Creator_ID)
REFERENCES USERS (UserID) ON DELETE SET DEFAULT ON UPDATE CASCADE);

```

```

ALTER SEQUENCE USERS_UserID_seq RESTART WITH 100000;

```

/\* This line is used to insure that the automatic increment for SERIAL starts at 100000, instead of the default value of 1, which is the starting value for userid's in this database \*/

```

CREATE TABLE COMMENT (Comment_ID SERIAL PRIMARY KEY, User_Comment
varchar (280) NOT NULL, Date_Created DATE, UserID int, File_Name text, FOREIGN KEY
(UserID) REFERENCES USERS (UserID) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (File_Name) REFERENCES AUDIO_FILE (File_Name) ON DELETE
CASCADE ON UPDATE CASCADE);

```

```
CREATE TABLE DEFINES_CHARACTERISTIC_OF (File_Name text, Tag_Name varchar
(15), FOREIGN KEY (Tag_Name) REFERENCES TAG(Tag_Name) ON DELETE CASCADE
ON UPDATE CASCADE, FOREIGN KEY (File_Name) REFERENCES AUDIO_FILE
(File_Name) ON DELETE CASCADE ON UPDATE CASCADE);
```

```
CREATE TABLE CAN_SUGGEST_EDITS (Suggestion_ID SERIAL PRIMARY KEY,
File_Name text, UserID int, Suggestion varchar (280), FOREIGN KEY (UserID)
REFERENCES USERS (UserID) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (File_Name) REFERENCES AUDIO_FILE (File_Name) ON DELETE
CASCADE ON UPDATE CASCADE);
```

/\*The following commands create necessary views for our database. Users may want to view files, tags, users, etc. based on certain attributes where joining two tables is necessary to receive the desired search results. The views provide a premade (by this file) "table" which are joins of other tables in the database to make searching easier\*/

```
CREATE VIEW Audio_File_Tags AS SELECT * FROM AUDIO_FILE AS AF NATURAL
JOIN DEFINES_CHARACTERISTIC_OF AS DC WHERE AF.File_Name = DC.File_Name;
/*joins audio files and tags based on the name of the audio file*/
```

```
CREATE VIEW User_Comments AS SELECT * FROM USERS AS U NATURAL JOIN
COMMENT AS C WHERE U.UserID = C.UserID;
/*Joins comments and users based on userid*/
```

```
CREATE VIEW Audio_File_Comments AS SELECT * FROM AUDIO_FILE AS AF
NATURAL JOIN COMMENT AS C WHERE AF.File_Name = C.File_Name;
/*joins audio files and comments based on the name of the audio file*/
```

```
CREATE VIEW Suggest_File_Edit AS SELECT * FROM AUDIO_FILE AS AF NATURAL
JOIN CAN_SUGGEST_EDITS AS CSE WHERE AF.File_Name = CSE.File_Name;
/*joins suggestions and audio files based on the name of the audio file*/
```

```
CREATE VIEW Creator AS SELECT U.UserID, U.User_Type, U2.USERID AS CREATOR_id,
U2.User_Type AS CREATOR_TYPE FROM USERS AS U JOIN USERS AS U2 ON
U.User_Creator_ID = U2.UserID;
/*joins users and users. This view was created to allow you to see the user type of the creator of all the other users. this provides an easy way to check that all created users are created by an admin*/
```

## DML

/\*This line inserts several tag tuples into the Tag table\*/

```
INSERT INTO TAG (Tag_Name) VALUES ('Trenton'), ('History'), ('Culture'), ('Photograph'),
('New Jersey'), ('Ewing'), ('Yearbook'), ('School'), ('Church'), ('Club'), ('Library'), ('Interview'),
('Old Trenton'), ('Neighborhood'), ('Sport'), ('Game'), ('Shooting'), ('Safety'), ('Crime'), ('Kids'),
('Art'), ('Music'), ('Money'), ('Law'), ('Rules'), ('Life'), ('Childhood'), ('Social'), ('Society'),
('Politics'), ('Economy'), ('TCNJ'), ('College'), ('Museum'), ('House'), ('Immigration'), ('Jewish'),
('Liquor'), ('Lawyer'), ('Family'), ('Business'), ('Philadelphia'), ('Immigrant'), ('Europe'),
('Development'), ('Youth'), ('Truck'), ('Improve');
```

/\*The following lines inserts several user tuples into the Users table in the database. Notice it is not necessary to enter a userID because the database will give the user one automatically. The user\_creator value also has a default so it does not necessarily have to be specified\*/

```
INSERT INTO USERS (Password, User_Type) VALUES ('dfvjh348', 'Admin');
```

```
INSERT INTO USERS (Password, User_Type) VALUES ('vwlijd827', 'Moderator');
```

```
INSERT INTO USERS (Password, User_Type) VALUES ('cerulian298', 'Moderator');
```

```
INSERT INTO USERS (Password, User_Type) VALUES ('gernal9823', 'Guest');
```

```
INSERT INTO USERS (Password, User_Type) VALUES ('random2984', 'Guest');
```

/\* The next lines are some examples of comment tuples which we insert in the database\*/

```
INSERT INTO COMMENT (User_Comment, Date_Created, UserID, File_Name) VALUES
('this is an example comment', '2021-04-10', '100001', 'Joel Millner');
```

```
INSERT INTO COMMENT (User_Comment, Date_Created, UserID, File_Name) VALUES
('another comment text', '2021-04-20', '100000', 'Rosenthal, Minerva');
```

```
INSERT INTO COMMENT (User_Comment, Date_Created, UserID, File_Name) VALUES
('first comment', '2021-04-11', '100004', 'Joe & Ida Klatzkin');
```

/\*The next command inserts several tuples into the defines\_characteristic\_of table which assigns some tags to each audio file. Note that both the audio file name and the tag name must be specified \*/

```
INSERT INTO DEFINES_CHARACTERISTIC_OF (File_Name, Tag_Name) VALUES ('Joel
Millner', 'Immigration'), ('Joel Millner', 'Jewish'), ('Joel Millner', 'Truck'), ('Joel Millner',
```

'History'), ('JHS 55 Samuel Rudner', 'Immigration'), ('JHS 55 Samuel Rudner', 'Liquor'), ('JHS 55 Samuel Rudner', 'Lawyer'), ('JHS 55 Samuel Rudner', 'Jewish'), ('Rosenthal, Minerva', 'Immigration'), ('Rosenthal, Minerva', 'Family'), ('Rosenthal, Minerva', 'Business'), ('Rosenthal, Minerva', 'Jewish'), ('Joe & Ida Klatzkin', 'Philadelphia'), ('Joe & Ida Klatzkin', 'Immigrant'), ('Joe & Ida Klatzkin', 'Europe'), ('Joe & Ida Klatzkin', 'Ewing'), ('Dr. Paul Loser', 'Development'), ('Dr. Paul Loser', 'Improve'), ('Dr. Paul Loser', 'School'), ('Dr. Paul Loser', 'Youth');

```
/* The next lines are some examples of suggestion tuples which we insert in the database*/
INSERT INTO CAN_SUGGEST_EDITS (File_Name, UserID, Suggestion) VALUES('Joe & Ida
Klatzkin', '100000', 'Line 42 should be changed');
```

```
INSERT INTO CAN_SUGGEST_EDITS (File_Name, UserID, Suggestion)
VALUES('Rosenthal, Minerva', '100001', 'I heard this instead of that');
```

```
INSERT INTO CAN_SUGGEST_EDITS (File_Name, UserID, Suggestion) VALUES('Dr. Paul
Loser', '100003', 'A cool suggestion');
```

### **CopyAFT File:**

/\*This file is designed to read lines from the AFT.csv file and transfer them into the database table Audio\_File. More lines may be added to AFT if desired, to include more instances of audio files. Superuser permissions must be activated for your account in order to use the Copy function\*/

```
COPY AUDIO_FILE(FILE_NAME, AF_LINK, TRANSCRIPT_LINK, DATE_ORIGINATED)
FROM '/home/lion/Documents/AFT.csv' DELIMITER ',' CSV HEADER;
```

### **AFT file contents:**

fileName,fileLink,transcriptLink,dateOriginated,like\_count,view\_count  
Joel

Millner,<https://archive.org/details/JHS13SideA>,[https://docs.google.com/document/d/18FNKen3YMKXLiiQvKzh791cwKLFAI2\\_MGkjDopXuYlY/edit?usp=sharing](https://docs.google.com/document/d/18FNKen3YMKXLiiQvKzh791cwKLFAI2_MGkjDopXuYlY/edit?usp=sharing),1995

JHS 55 Samuel

Rudner,<https://archive.org/details/JHS55SideA>,[https://docs.google.com/document/d/158zonmSrM92\\_J\\_YzVdJhlA7yCYq711vc5fAv3eXKPZY/edit?usp=sharing](https://docs.google.com/document/d/158zonmSrM92_J_YzVdJhlA7yCYq711vc5fAv3eXKPZY/edit?usp=sharing),1955

"Rosenthal,

Minerva",<https://archive.org/details/JHS05SideA>,<https://docs.google.com/document/d/17qD8zZNfONmbYCD36EYFUFhLc30aqWcGZ61bZve2t9k/edit?usp=sharing>,1996

Joe & Ida

Klatzkin,<https://archive.org/details/JHS08SideA>,[https://docs.google.com/document/d/1eSGVzOzxXYplQy9XhMs\\_Vh-7xg0ME6I\\_jfVLxC0U8b8/edit?usp=sharing](https://docs.google.com/document/d/1eSGVzOzxXYplQy9XhMs_Vh-7xg0ME6I_jfVLxC0U8b8/edit?usp=sharing),1988

Dr. Paul

Loser,<https://archive.org/details/OralHistroyWithDr.PaulLoserSideA,https://docs.google.com/document/d/1Vkr-8NJkRIT1P6WIXqX-nwi9uWixL3epEIW42jGE5fs/edit?usp=sharing,1976>

### **Queries Submitted for Stage Va:**

#### **Select Queries:**

```
SELECT * FROM Audio_File WHERE Date_Originated > 1960;
```

```
SELECT * FROM Tag;
```

```
SELECT * FROM Users WHERE User_Type = 'Moderator';
```

```
SELECT File_Name FROM Audio_File WHERE View_Count < 100 ORDER BY View_Count DESC;
```

```
SELECT File_Name, Tag_Name FROM Audio_File_Tags WHERE File_Name = 'Joel Millner';
```

```
SELECT File_Name FROM Audio_File WHERE Like_Count > 100;
```

```
SELECT File_Name FROM Audio_File WHERE Date_Originated = '1955';
```

```
SELECT COUNT (*) FROM Audio_File_Comments WHERE File_Name = 'Joe & Ida Klatzkin';
```

```
SELECT SUM(View_Count) FROM AUDIO_FILE;
```

```
SELECT * FROM COMMENT WHERE File_Name = 'Joe & Ida Klatzkin';
```

```
SELECT User_Comment FROM COMMENT WHERE Date_Created > '1995-12-31';
```

```
SELECT File_Name FROM Audio_File WHERE File_Name ILIKE 'Dr%';
```

```
SELECT min(Date_Originated) FROM Audio_File;
```

```
SELECT * FROM Audio_File ORDER BY View_Count;
```

```
SELECT File_Name FROM Audio_File WHERE NOT EXISTS (SELECT * FROM Audio_File WHERE Date_Originated = '2021');
```



```
SELECT File_Name FROM Audio_File WHERE Like_Count > ALL (SELECT Like_Count  
FROM Audio_File WHERE File_Name = 'Joe & Ida Klatzkin');
```

```
SELECT * FROM Audio_File_Comments WHERE FileName = 'Joe & Ida Klatzkin';
```

```
SELECT File_Name, Transcript, Suggestion FROM Suggest_File_Edit WHERE ViewCount >  
100;
```

```
SELECT User_ID FROM Creator WHERE User_Creator_ID = '100000';
```

### **Update Queries:**

```
UPDATE USERS
```

```
SET User_Type = 'Guest';
```

```
WHERE User_ID = '100001';
```

```
UPDATE AUDIO_FILE
```

```
SET Transcript_Link =
```

```
'https://docs.google.com/document/d/1eSGVzOzxXYplQy9XhMs_Vh-7xg0ME6I_jfVLxC0U8b  
8/edit'
```

```
WHERE File_Name = 'Rosenthal, Minerva';
```

```
UPDATE USERS
```

```
SET Password = 'soccer05'
```

```
WHERE User_ID = '100001';
```

```
UPDATE CAN_SUGGEST_EDITS
```

```
SET Suggestion = 'shorten this transcript because there is redundant info'
```

```
WHERE Suggestion_ID = '1';
```

```
UPDATE COMMENT
```

```
SET User_Comment = 'excellent interview'
```

```
WHERE Comment_ID = '3';
```

### **Delete Queries:**

```
DELETE FROM COMMENT WHERE Comment_ID = '2';
```

```
DELETE FROM AUDIO_FILE WHERE File_Name = 'JHS 55 Samuel RudnerI';
```

```
DELETE FROM TAG WHERE Tag_Name = 'Europe';
```

```
DELETE FROM USERS WHERE User_ID = '100002';
```

```
DELETE FROM CAN_SUGGEST_EDITS WHERE Suggestion_ID = '1';
```

```
DELETE FROM COMMENT WHERE User_ID = '100003';
```

```
/*deletes all comments for a user*/
```

```
DELETE FROM COMMENT WHERE File_Name = 'Dr. Paul Loser';
```

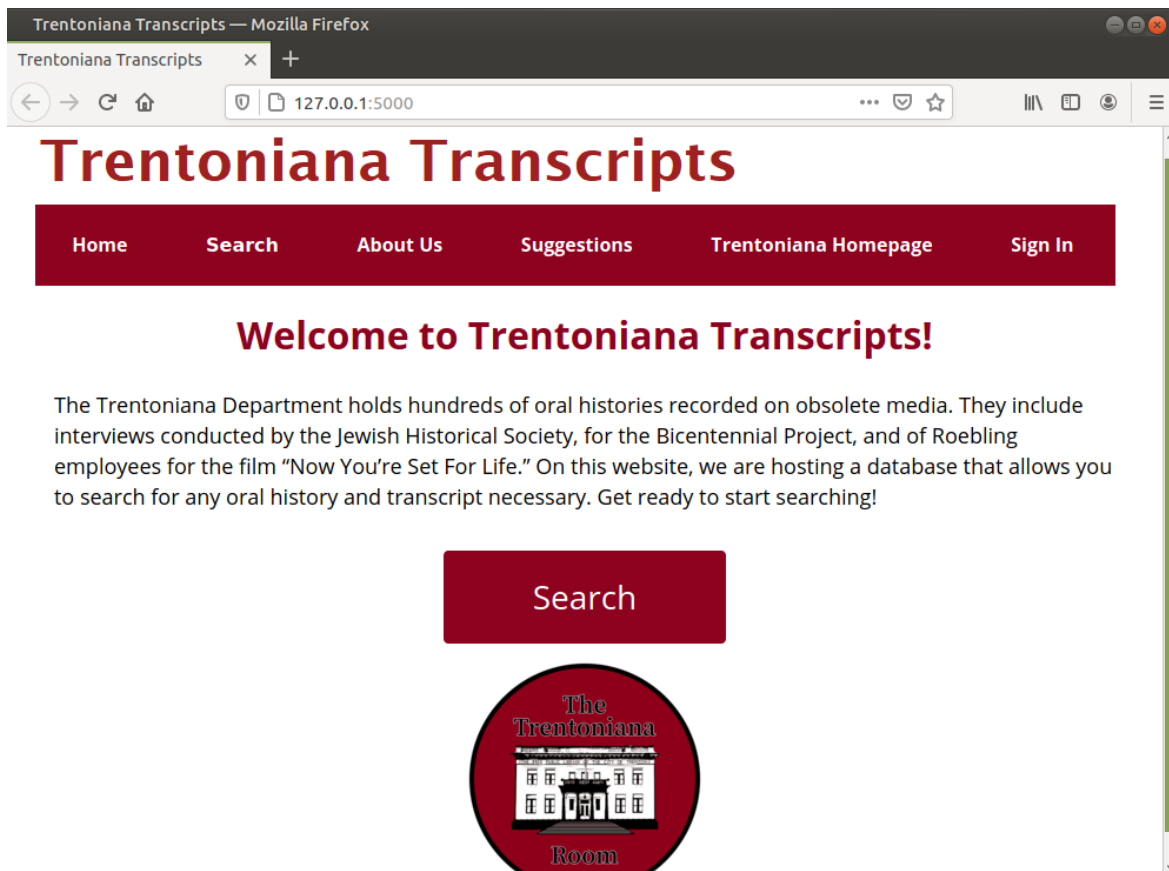
```
DELETE FROM COMMENT WHERE Comment_ID = '3';
```

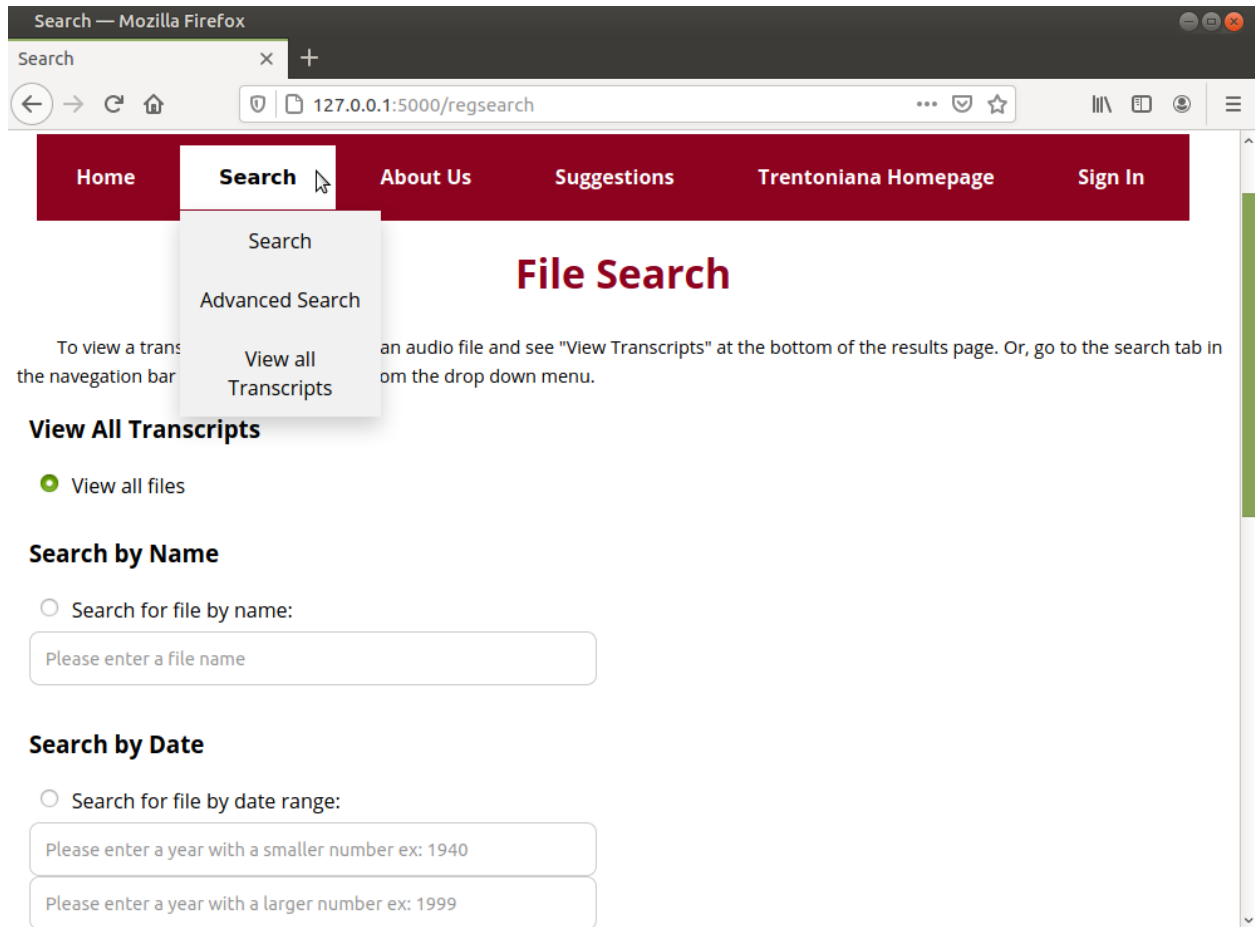
```
DELETE FROM DEFINES_CHARACTERISTIC_OF WHERE File_Name = 'Dr. Paul Loser';
```

## Stage Vb

### User Interface

These are two example pages of our user interface. It includes the home page and the top of the search page.





### **Maintenance**

All of our source code is thoroughly commented and documented. We uploaded all files onto GitHub and included clear and complete instructions for how to download and implement our project. This report and all other Stage V files can be found at:

<https://github.com/TCNJ-degoodj/stage-v-group-3>

### **Product Hand Over**

Our public GitHub repository can be found at the following link:

<https://github.com/martis36/transcriptDB>