

Creating a Music Database





Group 7 Project Final Report

Mario Arduz, Jack Charbonneau, Xiaoshuai (Maksim) Li, Geng Zhao

CS 542. Database Management Systems

9 December 2019

Contributions

| Member | Contribution % | Signature |
|-----------------------|----------------|---|
| Mario Arduz | 23.33% |  |
| Jack Charbonneau | 30.01% |  |
| Xiaoshuai (Maksim) Li | 23.33% |  |
| Geng Zhao | 23.33% |  |

Abstract: This project consists in building a web application to present information about music releases, songs, and artists. The information was obtained from a webpage that contains millions of instances of audio recordings and is stored in a PostgreSQL database. The functionality of the application is able to create, read, update and delete information with a Python based backend, and frontend supported by Jinja templates plus the regular resources of Bootstrap, CSS, JavaScript and HTML. Further improvements of the front end development are needed to have a fully functional application.

Keywords: Music Database, Flask, PostgreSQL, Web Page CRUD Application.

| | |
|--|-----------|
| Overview | 3 |
| Background Material | 3 |
| Front End | 3 |
| JavaScript | 3 |
| HTML/CSS | 3 |
| Bootstrap | 4 |
| jQuery | 4 |
| Jinja | 4 |
| Server | 4 |
| Python | 4 |
| Flask | 5 |
| SQLAlchemy | 5 |
| Database | 5 |
| Resources | 5 |
| Methodology | 6 |
| General Approach | 6 |
| Design | 7 |
| UML Use Case Diagram | 7 |
| Use Cases | 7 |
| ERD | 12 |
| Implementation | 13 |
| Issues | 13 |
| Track URLs | 13 |
| Partial Dates | 14 |
| Render Information from the Back-End to the Front-End | 14 |
| Write a front end that is consistent with the Get and Post Methods | 14 |
| Validation | 14 |
| Lessons Learned | 15 |
| Individual and Specific Lessons | 15 |
| Member Contribution | 17 |
| Conclusion | 17 |
| Future Work | 18 |
| Appendix | 19 |
| Gantt Chart | 19 |
| Schedule | 19 |
| Meeting Notes | 20 |

Overview

This project consists of building a web application to present information about music releases, songs, and artists. This information will be stored in a PostgreSQL database, retrieved by the server, and presented on the front end in the user's browser. This application will allow users to search for information, make modifications to the data, write reviews or create personalized playlists. The database will have a minimum of nine tables to support this functionality.

The functionality of the project is primarily focused on the actions performed by the user. The database will initially be populated with data from Discogs and Pitchfork, but the user will be able to update, delete and insert information from the tables. The primary function will be for the user to browse all of the artists, releases, tracks, and reviews that are present in the database. The web page will allow the user to search for this information based on a variety of filters such as artist name, release title, release rating, genre, etc.. The user will also be able to create an account, write reviews for albums, and create playlists of their favorite songs.

Whenever the user performs an action, it will be documented in a Log table in the database. Such actions include creating, deleting or updating a review or playlist. Upon carrying out any of these actions, a trigger will fire in the database to insert a row into the Log table containing the username of the user who performed the action, the date and time of the action, and a brief message defining what the action was. This log will help to keep track of activity within the application and may be useful for debugging in the event that errors occur.

Background Material

Front End

JavaScript

The team will mainly use JavaScript in HTML to build front end web application. JavaScript is really popular in HTML and has many advantages. JavaScript is very easy to implement in HTML and could work on web user's computers even when they are offline. Plus, it allows people to create highly responsive interfaces that improve the user experience and provide dynamic functionality, without having to wait for the server to react and show another page.

HTML/CSS

Html & CSS are really important in Web page. HTML is a language for describing the structure of Web pages. By using HTML we could publish online documents with headings, text, tables, lists, photos, etc. And retrieve online information via hypertext links, at the click of a

button. It also includes spread-sheets, video clips, sound clips, and other applications directly in documents.

We use CSS for describing the presentation of Web pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. This is referred to as the separation of structure (or: content) from presentation.

Bootstrap

This framework will be used to treat styles that are defined under CSS. The appearance of the website will be modified by implementing the most commonly used forms of user interface. For that matter, elements such as search or scroll bars will use the style pages provided by Bootstrap. This action is carried by defining links to the style pages inside the HTML file.

jQuery

jQuery is a library of JavaScript that will help to develop the functionality of the application. Its purpose is similar to the one of Bootstrap but focused on the manipulation of the page. This library is used as a complement to React because it is easier to implement while handling the structure of the page directly. The use of this library is also similar to Bootstrap, in the sense that it is included as a link within the web page.

Jinja

The Jinja templates are used to use the information that is retrieved from the backend to the front end. This language is useful because it is oriented to Python developers and works in a complementary way to the Flask code. This framework is partially used at both sides of the application to simplify the dynamic display of the information requested by the user.

Server

Python

The team will be using Python to program the back end of the web application. It is advantageous for our team to use Python when possible because it is the language that we have the most experience with. Because it is a very high level language, Python is also well suited for getting projects off the ground without using up huge amounts of time for development. Since the main focus of this project is the database, we are happy to save time on the development of the front end and server when possible. Python is widely used in server development and the team is confident that there will be an abundance of resources available to guide our learning in the early stages of the project.

Flask

Within Python, our team will also be making use of the Flask library. Flask is among the most popular libraries for creating web applications in Python. It provides a simple but flexible interface for creating endpoints and serving data to the front end. Flask is also well suited for creating REST APIs which can help our team stay consistent about the expectations for server programming and functionality.

SQLAlchemy

Another important Python library that our team will be using is SQLAlchemy. SQLAlchemy provides Object Relational Mapper that allows us to create Python classes corresponding to our database tables and then automatically generate tables based on those classes. SQLAlchemy also provides functionality for writing queries based on the object models. This library will be very useful to us because it provides a very simple and efficient way to handle server endpoints and database querying in a cohesive manner.

Database

Our team will be using PostgreSQL as our database of choice for this application. PostgreSQL has been widely used in web development. PostgreSQL is held in high regard because of its long history as a free and open source software. PostgreSQL also adheres strongly to the standards of the SQL Core Conformance. Given its extensive record of industry use and abundance of features, the team is confident that PostgreSQL will provide all of the functionalities that we need in order to implement our application.

Although the team only has a little bit of experience in using PostgreSQL, we are excited to take this project as an opportunity to learn another database technology in addition to Oracle which has been used for class assignments. We are confident that the large amount of widely available documentation for PostgreSQL and its fundamental similarity to other SQL databases will make it possible for us to quickly become proficient in using PostgreSQL.

In addition to PostgreSQL, we will also be making use of pgAdmin. pgAdmin is a graphical administration and development environment for PostgreSQL. It is available on all common operating systems and, like PostgreSQL, it is open source and widely used. The availability of a feature-rich GUI further increases our confidence in our ability to learn this new technology.

Resources

Official PostgreSQL Documentation: <https://www.postgresql.org/docs/>

Official pgAdmin Documentation: <https://www.pgadmin.org/docs/pgadmin4/latest/index.html>

Official Python3 Documentation: <https://docs.python.org/3/>

Official Flask Documentation: <http://flask.palletsprojects.com/en/1.1.x/>

Official Flask SQLAlchemy Documentation: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>
Udemy Flask Tutorial: <https://www.udemy.com/course/rest-api-flask-and-python/>
REST Tutorial: <https://www.restapitutorial.com/index.html>
Mozilla JavaScript Documentation: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
Mozilla HTML Documentation: <https://developer.mozilla.org/en-US/docs/Web/HTML>
Mozilla CSS Documentation: <https://developer.mozilla.org/en-US/docs/Web/CSS>
Official Bootstrap Documentation: <https://getbootstrap.com/docs/3.3/>
Official jQuery Documentation: <https://api.jquery.com/>
GitHub Guides: <https://guides.github.com/>
W3 Schools (quick reference for many web technologies): <https://www.w3schools.com/>
Python Pitchfork API Documentation: <https://pypi.org/project/pitchfork/>
Jinja Documentation <https://jinja.palletsprojects.com/en/2.10.x/>

Methodology

General Approach

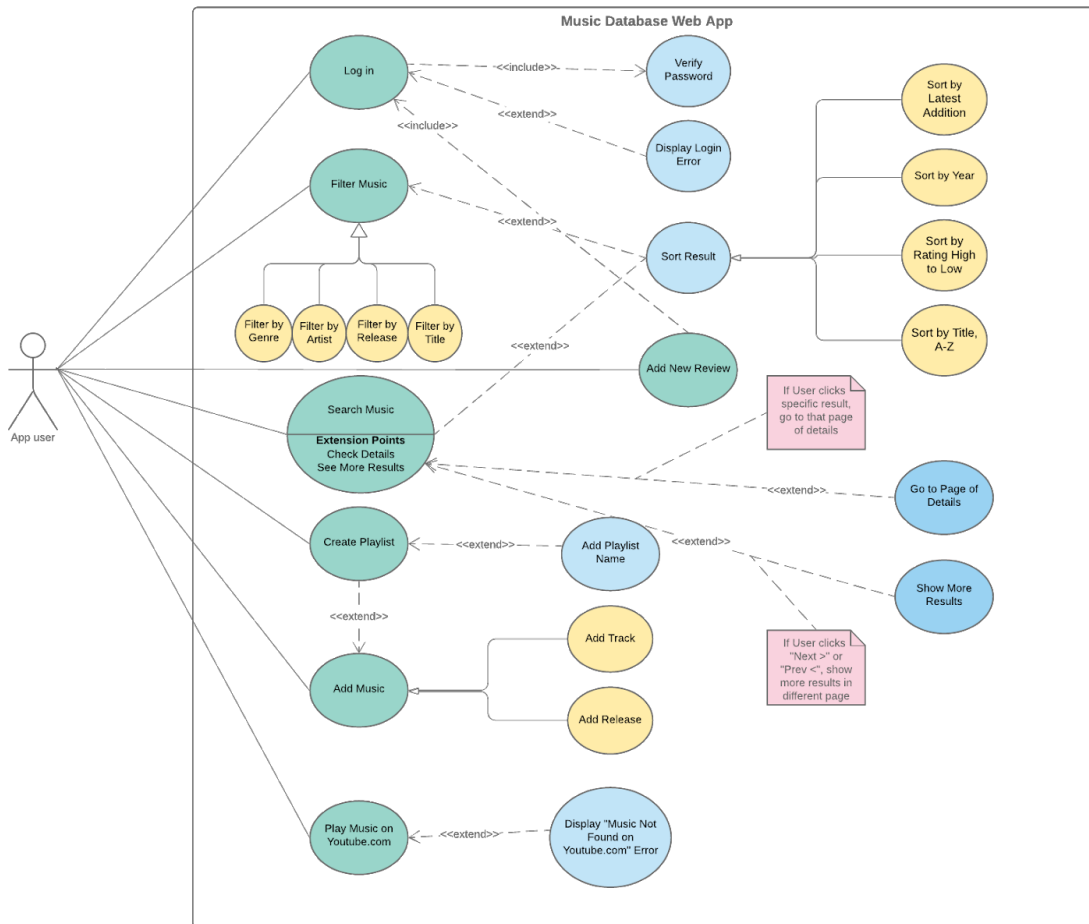
The general approach to develop the application consists in regular meetings to assess the progress of the project and to define the next tasks that should be carried to complete it.

The meetings started with the definition of the structure of the database. Discussions were held to agree on one ERD model, as well as its constraints and general functionality. Later, the team was divided in two groups that focused either on the back or front-end development.

The final stage of the project will be focused on having a complete merge between the front and back-end, as well as testing the main functionalities of the application and debugging the code when necessary. The approach to undertake the rest of the tasks will also be based in regular meetings and discussions.

Design

UML Use Case Diagram



Use Cases

| | |
|-----------------|---|
| Use Case Title: | Application user filters music |
| Actors: | Application User |
| Description: | This use case begins when a user, while using our web application, starts to filter by either "Music Genre", or "Release Name", or "Artist Name", or "Title Name", etc. |

| | |
|--|--|
| | <p>Basic Path:</p> <ol style="list-style-type: none"> 1. While using our music database web application, the application user selects a filter (“Music Genre”, or “Release Name”, or “Artist Name”, or “Title Name”) from a drop-down list, filtered result will be presented on the same page in default alphabetical order. 2. User has options to see the result in different order by picking up a sort from a drop-down list (including “Latest Addition”, “Year”, “Rating High to Low”, “Rating Low to High”, “Title, A-Z”, “Title, Z-A”, etc.). Filtered results are going to be sorted and represented. 3. The user may remove the filter or change the filter, the main result section will be represented. 4. The user can click “Next >” or “< Prev” to see more results. 5. The user clicks any specific result and then will be directed to a new page with details. <p>Alternate Paths:</p> <p>At any time during steps 1-4 the customer may exit the use case without actually viewing the details of any release or track info.</p> |
|--|--|

| | |
|-----------------|---|
| Use Case Title: | Application user searches music |
| Actors: | Application User |
| Description: | <p>This use case begins when a user, while using our web application, starts to enter a keyword of target artist name, or release name, or specific track name, or title name of company, etc.</p> <p>Basic Path:</p> <ol style="list-style-type: none"> 1. While using our music database web application, the application user enters a keyword of target artist name, or release name, or specific track name, or title name of company, etc. in the main search bar on the main page of our web application. 2. After entering the keyword, the user clicks the search button and then all the search results will be presented on the same page. 3. The User has the option to see the results in particular order by picking up a sort from a drop-down list (including “Latest Addition”, “Year”, “Rating High to Low”, “Rating Low to High”, “Title, A-Z”, “Title, Z-A”, etc.). Filtered results are going to be sorted and represented. 4. The user can click “Next >” or “< Prev” to see more results. |

| | |
|--|--|
| | <p>5. The user clicks any specific result and then will be directed to the related page with details of the result.</p> <p>Alternate Paths:</p> <p>At any time during steps 1-4 the customer may exit the use case and will be directed back page either by clearing the keyword in search text bar and then hitting return button on user's keyboard or just simply clicking the website logo.</p> |
|--|--|

| | |
|-----------------|---|
| Use Case Title: | Application user adds new review |
| Actors: | Application User |
| Description: | <p>This use case begins when a user, while using our web application, starts to add review while reviewing a specific page of release or track details.</p> <p>Basic Path:</p> <ol style="list-style-type: none"> 1. While using our music database web application, the application user moves to the review section located at the bottom of the page and starts to write reviews in the text box of review. 2. The user may be prompted to log in if the user hasn't logged in yet (Review text box is greyed out if the user hasn't logged in or using our web application as a tourist.) User will be directed to either log in or new user register page. 3. After logging in, the user can add the review or comment in the textbox. 4. A user can move on writing the view and click "Post Review" button to submit the review only if at least 10 words of review have been entered. "Post Review" button is greyed out on default and will be available to be clicked only when words count is over 10. 5. User clicks the "Post Review" button and review will be added and presented on the top of reviews section. <p>Alternate Paths:</p> <p>At any time during steps 1-4 the customer may exit the use case without actually submitting the review.</p> |

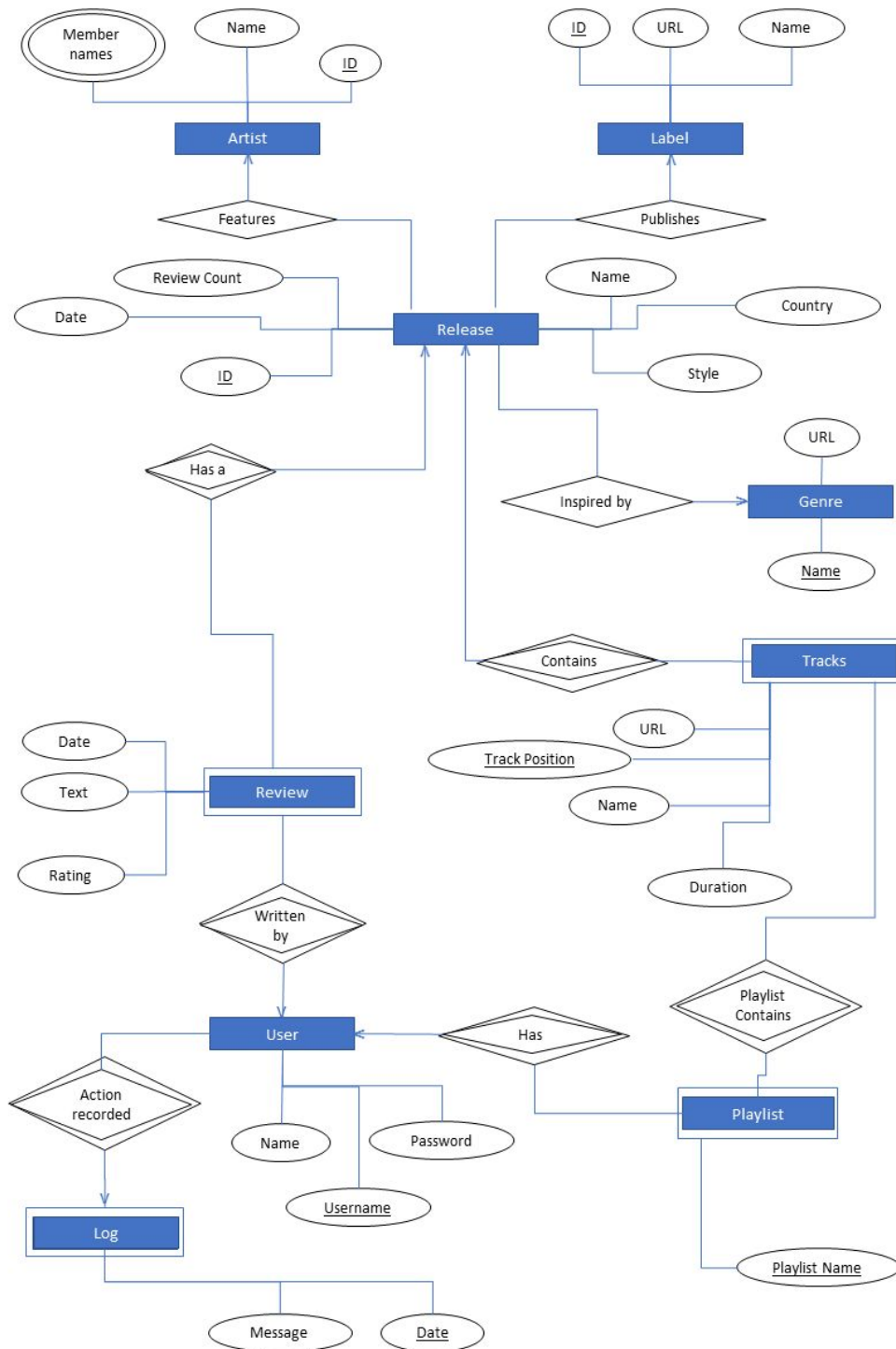
| | |
|-----------------|--|
| Use Case Title: | Application user creates playlist and adds tracks or a whole release |
|-----------------|--|

| | |
|--------------|--|
| Actors: | Application User |
| Description: | <p>This use case begins when a user, while using our web application, starts to create a new playlist and add tracks / a whole release.</p> <p>Basic Path:</p> <ol style="list-style-type: none"> 1. While using our music database web application, a user starts to create a playlist which will contain tracks from a single or various release(s). The user may be prompted to log in first if the current user hasn't logged in yet. The user will then be directed to either login page or new user registration page. The user creates a new playlist from scratch by clicking the "Create playlist" button on the page which will initiate the creation process. 2. The user will be asked to name the new playlist. If the playlist is not provided a new name by the user, a default name will be assigned. After the user clicks the "Yes" button on the creation confirmation window, a new playlist will be officially created. 3. The user can add certain tracks to the playlist while reviewing the details of tracks. There's a "Add this track" button inside music details page for each track. 4. The user also has the option to add a whole release into the playlist. "Add this release to playlist" button will be provided on the page of each release details. 5. The user has options to add the track or whole release to current existing playlists or to create a new playlist first and then add the track. If there's no existing playlist for the current user to add the track or release, the client will be prompted to create a new playlist first before adding tracks. <p>Alternate Paths:</p> <p>At any time during steps 1-4 the customer may exit the creation of a new playlist by clicking "No" when asked to confirm the creation or simple leaving the page without actually creating a new playlist.</p> |
| | |

| | |
|-----------------|---|
| Use Case Title: | Application user tries the desired track on Youtube.com |
| Actors: | Application User |

| | |
|--------------|--|
| Description: | <p data-bbox="427 207 1382 281">This use case begins when a user, while using our web application, starts trying to listen to the desired track.</p> <p data-bbox="427 312 591 344">Basic Path:</p> <p data-bbox="524 380 1354 453">While using our music database web application, the application user clicks the “Listen on Youtube” button for a certain track.</p> <p data-bbox="524 485 1386 516">The user will be directed to the page of that track on Youtube.com.</p> |
|--------------|--|

ERD



Implementation

In order to implement this project, our team is making use of a PostgreSQL database. This open source database is widely used and is compatible with all major operating systems. It also has a useful GUI tool in the form of pgAdmin. Rather than directly inputting SQL DDL statements, our team is making use of SQLAlchemy. SQLAlchemy is a Python library that acts as an Object Relational Mapper (ORM) which allows DDL statements to be derived from Python classes. In order to use this ORM, the team is converting the DDL statements described in the project proposal into model objects. This is a relatively simple task as the library is well documented and widely used. Once all of the tables are represented in the ORM, the Python script can be run to automatically add all of the tables to the database.

Once the tables are created, the team is using pgAdmin's import tool to import data from CSV files. The CSV files are created by parsing the raw XML files in Python and outputting the relevant data into a CSV. Each line in a CSV corresponds to one row in a table.

Once the database is created and populated with data, the team is once again using SQLAlchemy to handle querying. SQLAlchemy provides functions for the database model objects that can be used to represent SQL queries. As one example, a query for selecting tracks with the name 'example' would look like `Track.query.filter_by(tName = 'example')`. The team is converting the normal SQL queries from the project proposal into function calls like this.

These queries are attached to endpoints in the server which are handled by flask. As one example, a `/tracks` endpoint may carry out a query to select some tracks from the database based on certain criteria and then render the appropriate html template along with the data that was returned by the query. On the front end, the template will be filled out with the query data to show the user the results in an appealing way.

Issues

Track URLs

One of the first issues that the team ran into involves the `tURL` attribute of the `Track` table. When we initially examined the raw data, we thought that video URLs were mapped to their corresponding song. Upon further investigation, we have come to realize that video URLs are only tied to the release. This makes it impossible to map which URL corresponds to which Track. For the time being, the team is working around this problem by simply leaving all URLs null. In the future we would like to implement a more proper work around to match Track name and video title in order to determine which video URL goes with which Track. Although the results would not be perfectly accurate, it would be preferable to completely removing this feature.

Partial Dates

Another data parsing issue that we ran into is that of partial dates. Each release has a release date associated with it and the team was intending to store these values in the database as a date datatype. When we tried to do that we ran into an issue where some dates are in a format like "1999-0-0". Of course, 0 is not a valid month or day so the date datatype rejects this value. The date datatype will also not support a truncated value such as "1999" since it is not a complete date. Furthermore, the dates in the data take a variety of formats such as "yyyy-mm-dd", "mm/dd/yy", "yyyy", and so on. For these reasons, the team has decided that the most sensible solution is to simply store these values as strings rather than true dates.

Render Information from the Back-End to the Front-End

The main way the data is presented at the web page is through the incorporation of different rows into a table that is already defined at the html code. The first trials at the front end were using information in the form of a tuple of tuples. In consequence, the rows that were displaying the data inside in the html form could be iterated without any problems to show the instances rendered by the back-end. However, the later trials had another structure of information. In the later case, it was a dictionary of dictionaries that contained the data under the same keys, but in different dictionaries. This problem was solved by applying a modification to the dictionary to make an iterable object, so that the iteration at the front-end with Jinja could output the same information.

The mostly parallel development of the front and back end was another problem with the progress of the project. It requires a significant amount of consistency on both the declaration of the variables and also in the logic of the architecture to avoid consuming time in trying to merge both parts of the application.

Write a front end that is consistent with the Get and Post Methods

At the review page, it was necessary to implement a get method first, to obtain the information of all the reviews of a particular user. If the user submitted the form, a post method was needed to send the information of a new or updated review. Since this functionality had to perform consecutively at the same bottom of the page, it was problematic to implement both actions.

Validation

All the data definition was completed at the backend. Tables have been created and populated with data in the database. The majority of the work left to be done involves refining the interactions between the front and back ends to create a smooth user experience for the elaboration of reviews and playlists.

We can currently serve pages and data from the backend, and the front end user input functionalities (buttons, text entries, dropdowns, etc.) have been linked to the backend for the

searches of songs and albums. As it was mentioned earlier, it was not completely possible to implement these functions with the reviews and playlists.

One of the major methods we have used to validate functionality of our application is running tests for our application. We have done an initial set of tests to evaluate the functions and features of the music database application. We were only capable of doing white box testing, focusing primarily on verifying the flow of inputs and outputs through the application for different modules. The goal of this set of white box testing within our group members was to accomplish the functionality as well as improve the usability of our application. The limitation to perform a complete implementation of the application hindered the possibility of doing black box testing of the application.

Here is an example of white testing we did during the development. We did the testing with the help of use case scenario. The tester here is a developer from our group who wants to test the basic query function of our application. The developer wants to get a list of 10 tracks which include “untitled” in their track titles. Since user inputs aren’t supported yet, we create a dummy page for testing at “/untitled”. We then create a server endpoint to query the database for tracks named “untitled” and render the “untitled.html” page along with the data from the query.

In this test, we are able to verify that we have working functionality to store data in the database, query the database using SQLAlchemy, render html templates, and display templates filled with the data obtained from querying.

Lessons Learned

The lessons learned are diverse and correspond to the different backgrounds of team members. Nevertheless, the first stage of the design of the project provided an opportunity to learn how to structure a database applying the main concepts gained in class. Collective discussion led to the creation of a database that avoided redundancy and that included constraints, triggers and queries. More significantly, a better understanding of these concepts was obtained by means of group discussion.

The second stage of the project consisted in the implementation of the front and backend, where a pair of group members worked at each part. For that matter, each pair gained more expertise in either the side of frontend or backend of the application. However, the later steps of the project are going to provide a more homogenous understanding of both parts since the application needs to connect both features.

Individual and Specific Lessons

Xiaoshuai (Maksim) has been continually working on back-end object-relational-mapper (ORM) for the music database web application since the last project milestone. One major challenge for him in this stage of the project was completing creation of all tables by Python Flask-SQLAlchemy code and taking care of different kinds of problems when dealing with full

dependencies among different tables. Program testing and code revision sometimes could be a daunting and time consuming task. There are some back and forth before populating the final cleaned data into the tables created over Postgresql database. However, they all got sorted out after a huge amount of collaboration between Xiaoshuai (Maksim) and Jack who was in charge of data cleaning. Beside the database side, Xiaoshuai also has been working on creating backend codes to handle the requested raised from front end and send the requested data back to front end. All possible cases should be taken into consideration to make sure the application doesn't get interrupted. The biggest challenge encountered in this stage was the implementation of all concept and knowledge about database learned from the first half of the semester. Some of the tasks did take much time to be implemented over certain database platform. Xiaoshuai has learned that sometimes the final solution should be achieved by accomplishing various smaller sub tasks. Communication among group members and great plans are keys to the success of the whole project.

Geng Zhao has worked on completing front-end. He has learned how to create different kinds of forms, buttons and send request or get information from back-end. The big challenge is when you want to get or update some information, you need to consider all variables in both front-end and back-end. Find out the difference between back-end and front-end and debug could be hard and time consuming. Thus, he learned a little about how flask works because sometimes he misunderstood other teammates' suggestion and it's hard to merge the front end and back end smoothly. One of the most important things Geng learned is that front end is connected closely to back end. Only focus on front end will make you having a narrow view about this project. Also, Jack and Mario helped him a lot in handling the connection part. After the final part of this project, Geng learned that for a group project, merge all the parts from members is the hardest thing during the whole project. Meanwhile, good communication among group members is really important to the success of the project.

Jack has worked primarily on the data cleaning and back end development but has done some work on the front end as well. During data cleaning Jack learned how to use Python to parse XML data and how to work with large volumes of such data. Since XML is a hierarchical data format, parsing large quantities of data can sometimes result in the creation of massive trees that must be stored in memory. Jack learned that it is sometimes more efficient to break a large XML file into several smaller XML files and work on it that way. As for back and front end development, Jack learned a lot about using Flask and how to send data back and forth to allow pages to update dynamically.

Mario Arduz has worked establishing the core elements of the frontend application. He has a better understanding of the use of frameworks related to user interface and also to the exchange of information from the front end to the back end. The use of Bootstrap substantially helped the design of the interface, and a better knowledge of Flask helped to define the notifications for the insert, delete and update functions inside the html code. He was able to learn the general functionality of the back end that exchanges the information to the user page, and to apply functions similar to the declarations at Python inside the HTML with Jinja

Templates. All the skills learned at this project, beginning from the DDL, and complementing with the elaboration of the web page and a general understanding of the backend are totally new concepts. He is more aware that the learning curve of developing an application is rather a floor instead of a continuous function.

Member Contribution

| Task | Contributors |
|----------------------------|---|
| Parsing Raw Data | Jack (full effort) |
| Front End Development | Mario (full effort), Geng (full effort), Jack (partial effort) |
| Back End ORM Creation | Maksim (full effort), Jack (partial effort) |
| Back End Endpoint Creation | Jack (full effort), Maksim (full effort), Mario (small effort), Geng (small effort) |

| Member | Contribution % |
|-----------------------|----------------|
| Mario Arduz | 23.33% |
| Jack Charbonneau | 30.01% |
| Xiaoshuai (Maksim) Li | 23.33% |
| Geng Zhao | 23.33% |

Conclusion

The team was successful in planning, designing, and creating a robust database application. The team was able to design a database to suite the needs of our application and implement the database using PostgreSQL. The team was also able to parse data that was found online and load it into the database. The database successfully implements a variety of different data types and relationships. The team was able to use Python's SQLAlchemy library to query the database and obtain correct results. The team was also able to implement triggers that function correctly using SQL (as opposed to accomplishing this through a library).

Furthermore, the team was able to accomplish the creation of a full stack web application to allow users to interact with the database. The application is capable of accepting

data from the user via the front end, running queries based on the user's input, and displaying the results of the queries. It is possible for the user to Create, Read, Update, and Delete from the database via the front end.

The greatest challenge in completing this project was the web development itself as opposed to the database development. This is because the team had very little web development experience prior to starting this project. As such, one of the biggest objectives that remains to be accomplished is the ability to maintain a user session on the front end. As the application currently stands, a username is hardcoded to give the appearance of being truly logged in. When logging in, the database does verify that the username and password are valid, but once the user leaves the initial login page the 'session' is lost.

Since the team spent so much time on the web development, we were not able to complete everything we desired for the database, namely the creation of views. Since they were not necessary for the function of the application, we did not make them a priority for completion.

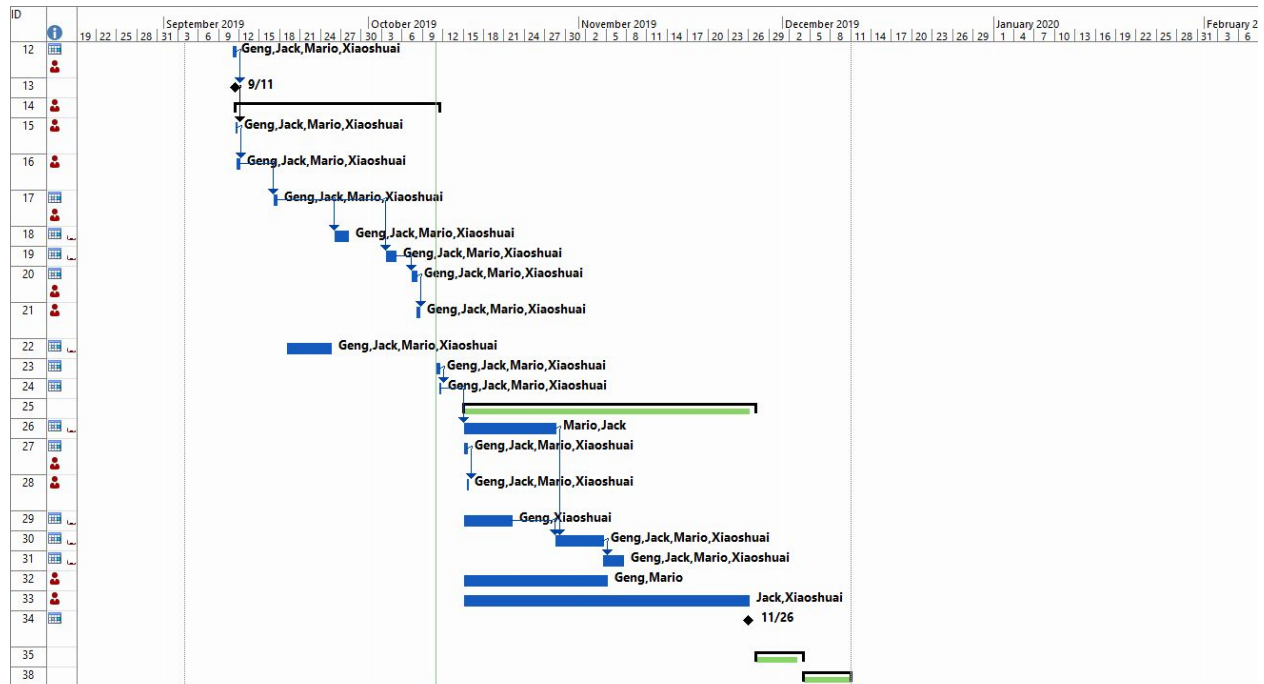
Future Work

A fully practical application needs further development of the front end in relation with maintaining a user session, the playlist and review functionalities. It is necessary to hold the information of which username is requesting or updating the information to render the data. Similarly, it would be needed to have a definition of the functions that it is more correspondent to the Get and Post methods, since most of them work only under the later concept.

In the future we would also implement some views and a 'statistics' page to display stuff such as artists and their longest song, release and number of tracks, or genre and number of releases.

After a complete implementation of the application, it would be possible to think about improvements of the back end with indexes. Since most of the queries are based on search functions at the songs and releases pages, it would make sense to elaborate a hash index on the titles of the songs, albums and names of the artists. Similarly, It would be interesting to compare the estimated theoretical cost (under a regular search of the heap files and with the hash index) between execution time of the queries.

Gantt Chart



Schedule

The following is the updated version of project schedule for the rest of the semester.

| Date | Mission |
|---------------|--|
| 11/06 - 11/12 | <ul style="list-style-type: none"> ● Collect and analyze feedback from professor regarding project progress report. ● Continue working on both front end and back end of the rest of web application. ● Communicate between front-end and back-end sub development groups. ● Merge the work of front-end and back-end. ● Debug and run test within sub development group. ● Apply advanced database tools to optimize the performance of database. |

| | |
|---------------|---|
| 11/13 - 11/17 | <ul style="list-style-type: none"> • Import remaining data for secondary tables • Implement SQL statements from project proposal • Debug and run white box tests for the whole project. • Prepare for deployment of the application on cloud. |
| 11/18-11/22 | <ul style="list-style-type: none"> • Deploy the web application to Heroku.com to run it on cloud. • Debug and test the functionality of the application on cloud. • Start working on final project documentation. |
| 11/23 - 12/3 | <ul style="list-style-type: none"> • Design and perform Acceptance Test (Alpha test and Beta test) for the web application by involving external test group. • Modify and finalize the development of the application. |
| 12/4-12/10 | <ul style="list-style-type: none"> • Finalize the group project final documentation. • Summarize the project and evaluate the performance of group members. • Prepare for final group project presentation. |
| 12/10-12/13 | <ul style="list-style-type: none"> • Present the project in class. • Evaluate, end and review the project. |

Meeting Notes

| | |
|-----------|--|
| Time: | Sept 9th, 2019 - Monday - 5:00 PM - 7:00 PM |
| Location: | Library Main Floor Discussion Area |
| Works: | <ul style="list-style-type: none"> • Worked on project intent. • Created UI mockups for the group project. • Created group project intent presentation slides. • Prepared for the presentation of project intent on Wednesday's class. |

| | |
|-----------|---|
| Time: | Sept 27th, 2019 - Friday - 3:00 PM - 6:00 PM |
| Location: | Library Tech Suite 111 |
| Works: | <ul style="list-style-type: none"> • Further discussed the group project details. • Formally assigned project work among all group members. • Adjusted the project completion plans. • Prepared for ERD design for the project. |

| | |
|--|---|
| | <ul style="list-style-type: none"> Assigned group project proposal to each member. |
|--|---|

| | |
|-----------|---|
| Time: | Oct 4th, 2019 - Friday - 3:00 PM - 6:00 PM |
| Location: | Library Tech Suite 111A |
| Works: | <ul style="list-style-type: none"> Reported progress from each team member on project proposal. Reviewed sample dataset. Worked together on creation of ER diagram for the database. Assigned work to be accomplished by each member before the next meeting. |

| | |
|-----------|--|
| Time: | Oct 7th, 2019 - Monday - 5:30 PM - 7:00 PM |
| Location: | Library Main Floor Discussion Area |
| Works: | <ul style="list-style-type: none"> Discussed and revised ER diagram. Group members updated their own project progress. Planned for next formal group meeting on Thursday. |

| | |
|-----------|---|
| Time: | Oct 24th, 2019 - Thursday - 3:00 PM - 6:00 PM |
| Location: | Library Tech Suite 111A |
| Works: | <ul style="list-style-type: none"> Assigned front-end and back-end to group members. Designed detailed schedule for group work by next project milestone. Revised database design. |

| | |
|-----------|--|
| Time: | Nov 1st, 2019 - Friday - 3:30 PM - 6:30 PM |
| Location: | Library Tech Suite 122 |
| Works: | <ul style="list-style-type: none"> Worked in sub groups on front-end as well as back-end development. Collaborated and communicated the progress of each sub group. Planned for future programming to accomplish more function. |

| | |
|--|---|
| | <ul style="list-style-type: none"> Assigned unfinished writing part of Progress Report to group members. |
|--|---|

| | |
|-----------|--|
| Time: | Nov 4th, 2019 - Monday - 4:00 PM - 7:00 PM |
| Location: | Library Tech Suite 120 A |
| Works: | <ul style="list-style-type: none"> Worked together to combine codes of front-end and back-end. Figured out the problems when combining the work together. Went over the current version of Project Progress Report documentation and identified remaining part of the report. |

| | |
|-----------|---|
| Time: | Nov 5th, 2019 - Tuesday - 6:00 PM - 9:00 PM |
| Location: | Online Meeting with Slack |
| Works: | <ul style="list-style-type: none"> Communicated and distributed the debugged codes among group members. Tested functions of applications which will be demonstrated during the class. Finalized the Progress Report. |

| | |
|-----------|---|
| Time: | Nov 5th, 2019 - Tuesday - 6:00 PM - 9:00 PM |
| Location: | Online Meeting with Slack |
| Works: | <ul style="list-style-type: none"> Communicated and distributed the debugged codes among group members. Tested functions of applications which will be demonstrated during the class. Finalized the Progress Report. |

| | |
|-----------|---|
| Time: | Nov 21st, 2019 - Thursday - 6:00 PM - 9:00 PM |
| Location: | Tech Suite 202 |

| | |
|--------|---|
| Works: | <ul style="list-style-type: none"> • Tested the code of database creation from Python Flask to Postgresql database. • Imported all cleaned data file to database created on Postgresql. • Sub groups worked on front-end and back-end. |
|--------|---|

| | |
|-----------|--|
| Time: | Dec 6th, 2019 - Friday - 3:00 PM - 6:00 PM |
| Location: | Foies Innovation Lab |
| Works: | <ul style="list-style-type: none"> • Merged the code from front-end and back-end together. • Assigned the rest of the project to group members. • Tested functionality of merged code together. |

| | |
|-----------|---|
| Time: | Dec 7th, 2019 - Saturday - 3:00 PM - 6:00 PM |
| Location: | Tech Suite 112 |
| Works: | <ul style="list-style-type: none"> • Merged the code together. • Tested functionality of merged code. |

| | |
|-----------|--|
| Time: | Dec 8th, 2019 - Sunday - 6:00 PM - 9:00 PM |
| Location: | Tech Suite 316 |
| Works: | <ul style="list-style-type: none"> • Merged the code between front-end and backend-end. • Tested functionality of triggers in PostgreSQL database. • Further tested the code for the whole project. • Finalized the final documentation. • Finalized the presentation slides. |