Alex Useloff, Jordan Butler, Jason Richards, Jacob Engelbrecht

Eamon Johnson

Introduction to Database Systems

May 2, 2019

Datify Report

Overview:

Datify is a web application where unique users can curate songs, organize playlists, and connect with others. The core intention of Datify is to provide a music repository for people to build their connections. We achieved this by collecting a set of (copyright free) music which users can add to their personal libraries. Users are first prompted to sign in with an email and password which uniquely identifies them. Once logged in, users have the ability to stream songs without permanently storing the song on the users' computer. Beyond this, they can create playlists to organize subsets of their music. As a music based service, Datify focuses on the users experience and ability to interact with music.

Through library and playlists, users are able to add and delete songs from these collections. As song files are stored once and then connected to users through relations, our server is constantly creating and deleting entries to show these changes. For overall metrics, whenever a song is played, we are able to update the song to track the total amount of plays. We are constantly reading from the database to show the songs in a users library, the songs in a given playlists, and the playlists associated with a user. Our database is constantly creating, reading, updating, and deleting to match the needs of its users.

To better the user's experience, we provide song recommendations based off of our entire user pool's song trends. We do this by analysing songs recently listened to by the user and then looking for tends all users have between that song and other songs. We aggregate this data to only recommend the songs the most users commonly listen to in common with the current users listening habits.

Requirements:

More explicitly, we constantly read from the database. All displays of library, playlists available to a user, songs in a playlist, and even checking login credentials are reading a database. Anytime we add a song to a library, add a song from a library to a playlist, or create playlists we are creating new entries into the database. We are constantly updating songs to keep track of the total number of plays. Whenever songs are removed from libraries, playlists, or playlists are removed all together, we delete entries from the database.

We use aggregate functions when recommending songs to find the song most played following a given song to recommend to a user. This utilizes the count aggregate function in sql, as well as group by and ordering by characteristics. Additionally, we welcome the user with the song which they have played the most, which uses aggregate functions to recommend.

The more direct requirements were achieved with being able to host the server on the Case EECS servers, thus being accessible anywhere on the case network through port 34108 (whenever we have someone hosting the server). We are using a MySQL Database hosted on this server to function as our storage server. This server is connected by a python MySQL connector to the python backend for the server. The python backend uses a Flask framework to serve the

web requests, and a configuration parser to initialize both the MySQL connector and Flask ports. Our frontend runs off of HTML, CSS, JavaScript and Jinja.

Google Chrome has been used as our web browser of choice to test our programs, and everything has run well. These should scale across all common browsers without issue.

Beyond the Requirements:

There have been a lot of areas where we paid extra attention to detail. The first and most obvious is the visual appearance of the website. For none of us being graphic designers, we created a very visually appealing site that is enjoyable and easy to navigate. Consistent color schemes, navigation bars, standard layouts were all points of detail we noted. Utilizing hover commands made interacting more fluid and enjoyable. It visually appears as you would expect a legitimate web server. We designed a logo which appears on all pages as well as the favicon for the web page.

Other areas where we paid attention to detail was the search bar, as we secured ourselves from sql injection attacks by parsing the string. We analyze all searches looking for a ‘ to be sure no one is tampering with our database. Through implementing this we learned many other public database website do not do this.

Another area is smart architecture designs of the SQL database with cascade on delete command and enums. Cascade on delete helps remove no longer useful data to reduce storage overhead and keep the database clean as playlists are deleted. This also helps us enforce songs needing to be in the user’s library to be in a playlist as if we delete a song from the library, it will automatically be removed from all playlists.

Additionally, we used the enum data type for instances of strings which should only exist of one or two values. This could be done on the backend with converting tiny integers to correlated strings; however, for readability, we decided to utilize the enums.

Finally we future-proofed our design as much as possible. We implemented a naive search functionality, but have plans for a more advanced search implementation. Friends and searching for other playlists are in the process of being implemented. There is always more to do with projects, and while we achieved our original plans, we were sure to design our project with the ability to expand in the future.

Member Contributions:

Jason:

Lead on the front-end design. Designed all webpages, including the HTML, CSS, Jinja, and JavaScript. Jason also did the connections between the back-end and front-end. Created all Flask app routes and implemented SQL queries in server's Python code. Implemented SQL query execution in Python and data management. Handled errors that arose throughout the process of connecting front-end and back-end. Learned about and implemented Flask sessions for differentiating between current user on a webpage.

Jordan:

Lead on SQL query design. Wrote majority of SQL queries used on the backend. Helped design aggregate implementation as well as webpage navigation structure.

Alex:

Lead on database design and data implementation. Created the vast majority of the sample data that was entered into the database. Helped design more complex theories for sql structure. Helped design queries for recommendation engine and sample data to test those queries. Learned how to use github in the process. Also learned a lot about virtual environments and servers.

Jacob:

Team leader as well as whatever needed to be done. A lot of initial planning and work on proposals and data flow. Designed initial basic schemas and designed full initial working prototype for the project to be build off of. Helped teach and encourage the use of Github repositories to store and share work. Organized team meetings and helped facilitate conversation and work amongst all members. Implemented the use of the server both for the database and eventually for the backend web host as well. Designed plan for recommendation engine. Helped with some python helper methods. Generated sample data relevant for working song entries in the database. Assisted with query design when needed.

Overall:

We were a matched group of two pairs, and we ended up working really well together. We had strong communication and helped each other learn and think through ideas better as a group. Jason definitely would have been the lead developer if this was a scrum team, as he

orchestrated most of the programming implementation. We all were active and worked many hours in person as well as individually to provide a project we are happy with now.

Things Learned:

So much. We are all so much more familiar with GitHub, something that will be very practical in the future. Gaining exposure to using terminal, virtual machines, linux machines, and / or ssh has been beneficial for us all. We learned about sql process management and how to kill active processes that hung for one reason or another and then induced a deadlock. We learned how to efficiently divide work among users, as well as the benefits of working in small groups to bounce ideas off of each other especially when designing queries.

We learned so much about all the languages we used as well. Many SQL errors improved our query writing, the loose type structure of python was very relieving and helpful compared to Java which we also considered using, and figuring out Flask sessions to track basic user information.

All websites now look a little different to us. With an understanding of how databases often connect and store information to web pages, we see one layer beyond the obvious. Learning how different websites handle and optimize searches is something we notice, and we are now often diving into the developer tools of Google Chrome for a better understanding of how websites are designed. This has been a really wonderful project which helped us gain a better understanding of how the web operates, as well as improved our ability to create, design, and impact an entire web designed project.

Information:

When the run.py is active by any user on the server, the webpage can be accessed here. http://eecslab-9.case.edu:34108/

To run as what is intended, you need updated Jinja, Flask, MySQL connecter, and config parser. These installations are noted in the setup file provided. Data for the server should be input first from setup.sql, and then data.sql for sample data (note: only some songs have .mp3 files provided for copyright issues). The dump file could also be loaded for the same database, but errors would be harder to diagnose.