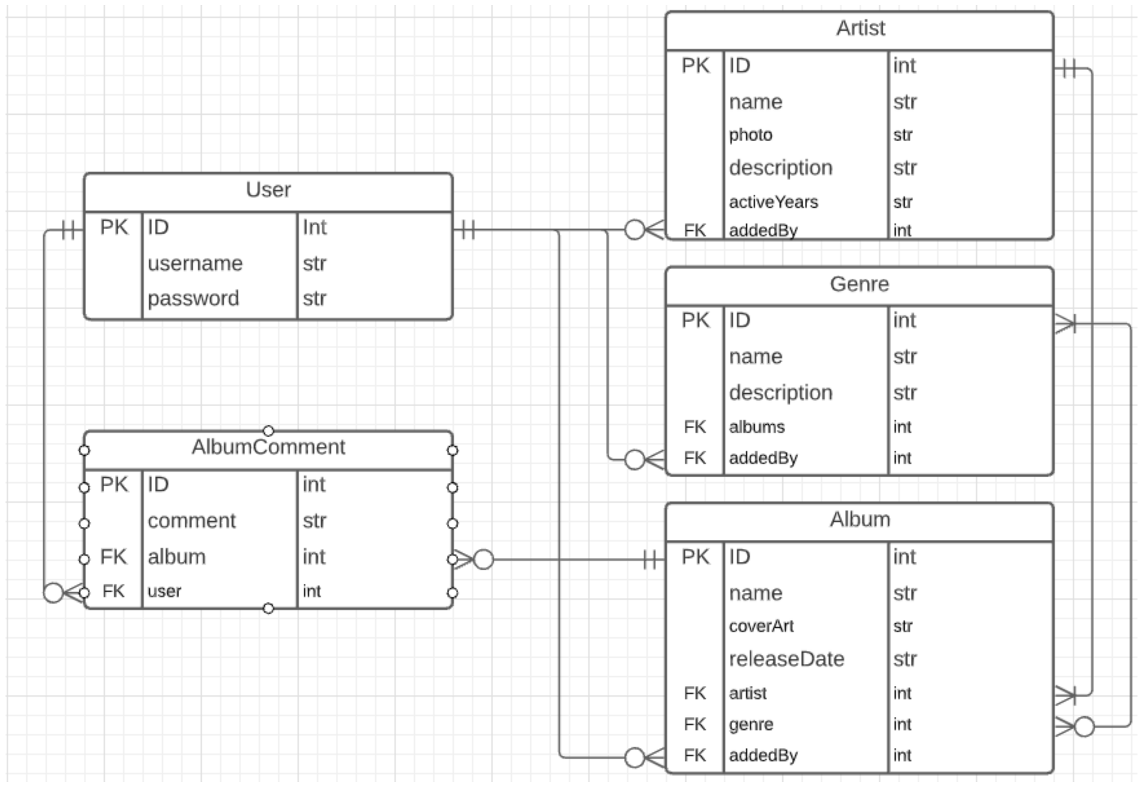
Github: https://github.com/jamesw1307/musec/

My project is a website that lets its users add genres, artists and albums to the database, to get other users to see other music tastes and discover new music. I'm making this website both for people who want to share their favorite music and those who want new music to listen to.

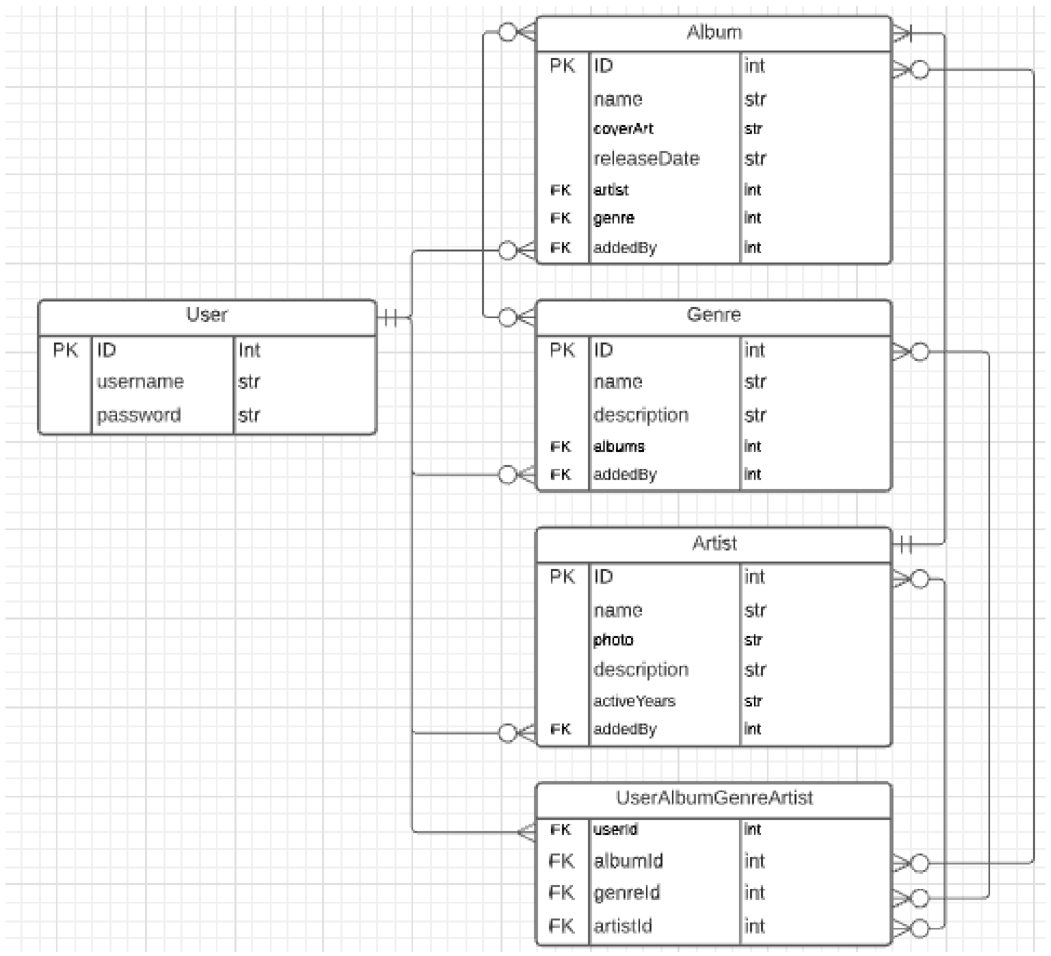
Database design

Plan design



First revision

My first problem with my planned database design is that there was no linking table, which I would need in order to store data for a many to many relationship between the Genre and Album table. For my first revision I decided to make a linking table for data for all tables. My thought process was that this would make it easier if I ever wanted to see all data relationships in one place. I also removed the album comments, as it wasn't necessary for the purpose of the website and also not easy to add with the time I had to complete the project.



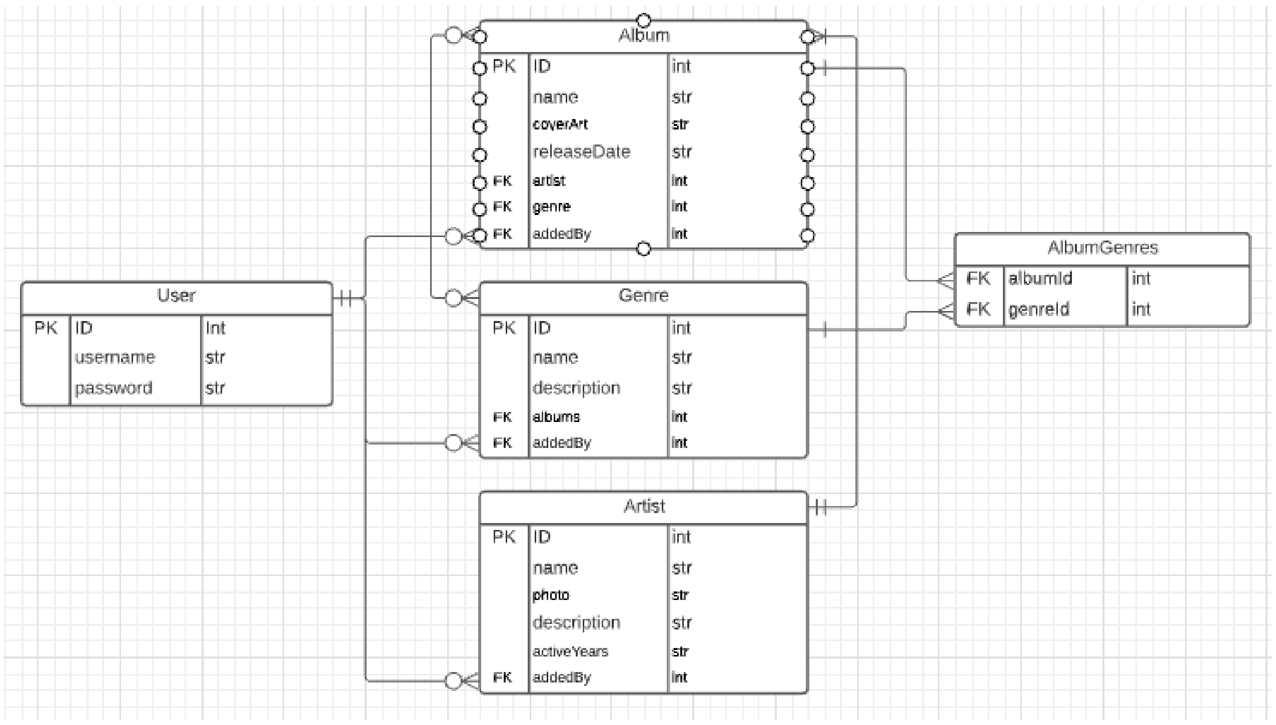
Many to many between genre and album:

This was done by using a db.relationship for both tables that both referenced to a linking table that contained album and genre id's. This made it so, for example, an album id would have multiple duplicates but each one would have a different attributed genre id (and vice versa). This way the many to many is fully functional, does not have unnecessary links between tables that are not necessary for my website, makes it much easier to manage data, and avoids errors that do not affect the website.

The db.relationship uses the linking table as secondary and back_populates references the other relationship in the opposite table.

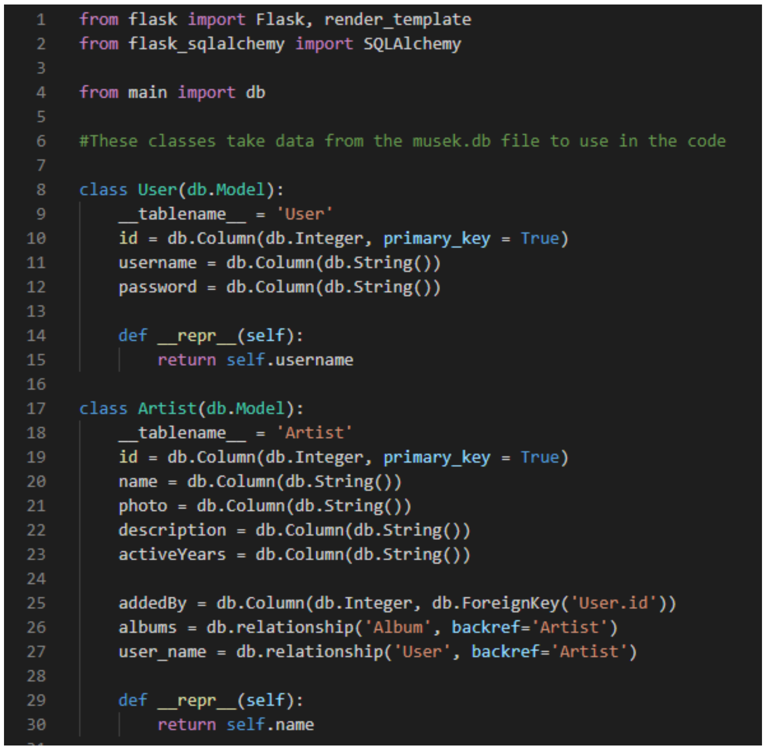
Second Revision

After getting my many to many relationship to work using db.relationship in sqlalchemy, which I did with a unique AlbumGenre linking table, the UserAlbumGenreArtist table was unnecessary and had a lot of data that was difficult to manage and not necessary to use for my websites functionality. So I removed this table completely, and instead just kept the AlbumGenre linking table. This was the last change I made to my database aside from repr functions and one to many relationships that allowed me to show data on my website as strings instead of lists.

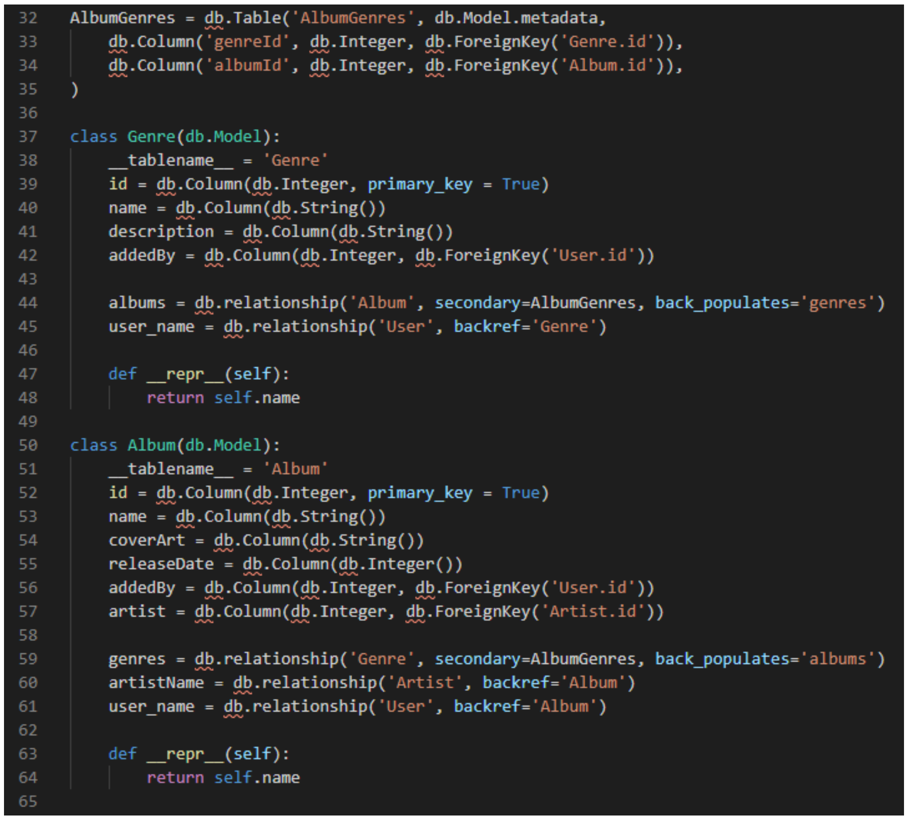


Models functions:

Below are the classes made for each table in my database after the final database design revision.



```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

from main import db

#These classes take data from the musek.db file to use in the code

class User(db.Model):
    __tablename__ = 'User'
    id = db.Column(db.Integer, primary_key = True)
    username = db.Column(db.String())
    password = db.Column(db.String())

    def __repr__(self):
        return self.username

class Artist(db.Model):
    __tablename__ = 'Artist'
    id = db.Column(db.Integer, primary_key = True)
    name = db.Column(db.String())
    photo = db.Column(db.String())
    description = db.Column(db.String())
    activeYears = db.Column(db.String())

    addedBy = db.Column(db.Integer, db.ForeignKey('User.id'))
    albums = db.relationship('Album', backref='Artist')
    user_name = db.relationship('User', backref='Artist')

    def __repr__(self):
        return self.name
```

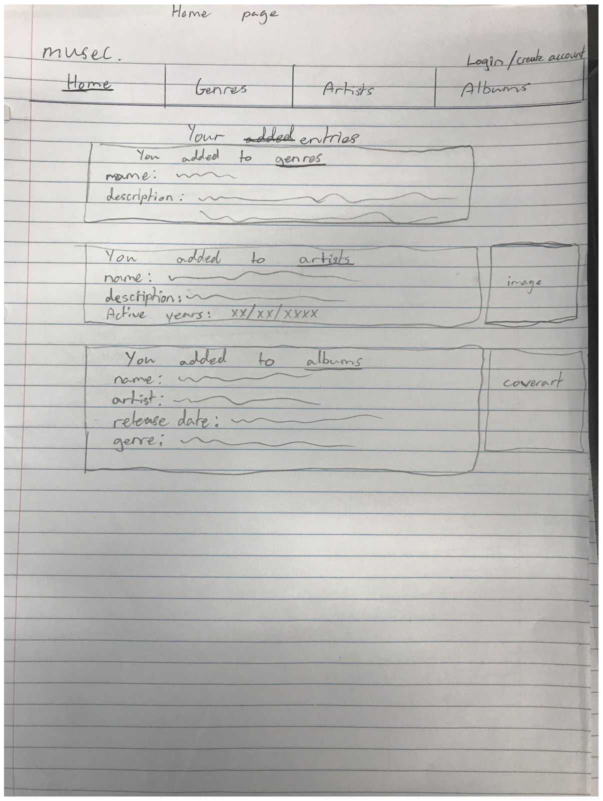


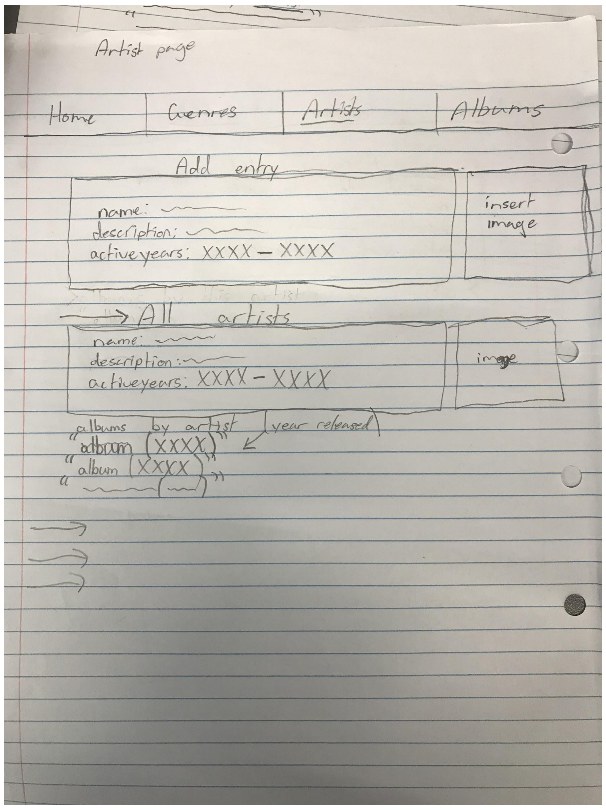
```
AlbumGenres = db.Table('AlbumGenres', db.Model.metadata,
    db.Column('genreId', db.Integer, db.ForeignKey('Genre.id')),
    db.Column('albumId', db.Integer, db.ForeignKey('Album.id')),
)

class Genre(db.Model):
    __tablename__ = 'Genre'
    id = db.Column(db.Integer, primary_key = True)
    name = db.Column(db.String())
    description = db.Column(db.String())
    addedBy = db.Column(db.Integer, db.ForeignKey('User.id'))

    albums = db.relationship('Album', secondary=AlbumGenres, back_populates='genres')
    user_name = db.relationship('User', backref='Genre')

    def __repr__(self):
        return self.name

class Album(db.Model):
    __tablename__ = 'Album'
    id = db.Column(db.Integer, primary_key = True)
    name = db.Column(db.String())
    coverArt = db.Column(db.String())
    releaseDate = db.Column(db.Integer())
    addedBy = db.Column(db.Integer, db.ForeignKey('User.id'))
    artist = db.Column(db.Integer, db.ForeignKey('Artist.id'))

    genres = db.relationship('Genre', secondary=AlbumGenres, back_populates='albums')
    artistName = db.relationship('Artist', backref='Album')
    user_name = db.relationship('User', backref='Album')

    def __repr__(self):
        return self.name

```

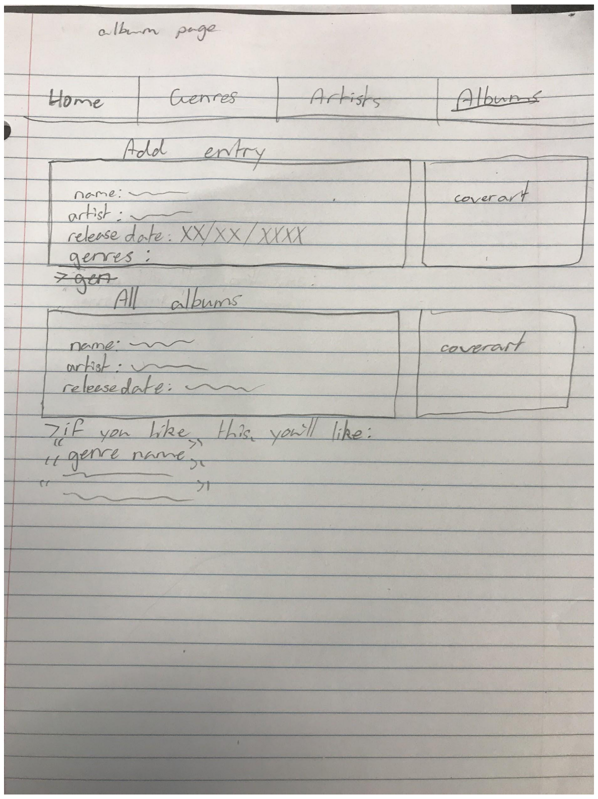
- Website design notes
  - Four different pages: Home/profile that shows all your own entries and album comments ordered by recently added. Genre page that shows all recently added genres. Each addition has its own container box containing the information, and below that is a list of all albums that have that genre attached. Artist page showing all recently added artists, each artist has a container with a sub list of all albums that artist has to their name.
    Last page is the albums page where each album just has one container box with info but users can add comments to the albums.
  - All entries are ordered by recently added to encourage looking at new entries.
  - Top of genre, artist, album pages will have an empty container with the necessary text boxes to fill out and make an entry. Filling all will add it to the database.
  - Top right corner of the home page will have one button, create/login. When clicked, a page appears with a username text box and password box. When an entered username is new, it is added to the user database, along with the password attached. If the username and password are in the database, the user can create entries and comments with that user ID.
  - Under a genre entry will be a sub list of albums with that genre.
  - Under an artist entry is a sub list of albums by the artist.

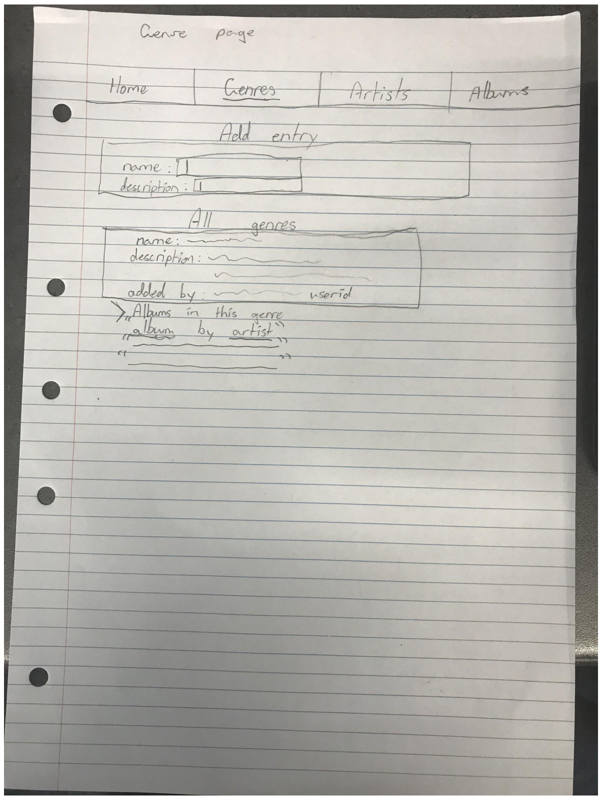
- Under an album entry is a sub list of genres the album is attributed to ("if you like this you might like…")

- Page layout diagrams for your webpages


Home page
musec.
Login / create account
Home
Genres
Artists
Albums
Your ~~added~~ entries
You added to genres
name:
description:
You added to artists
name:
description:
Active years: XX/XX/XXXX
image
You added to albums
name:
artist:
release date:
genre:
coverart


Artist page
Home
Genres
Artists
Albums
Add entry
name:
description:
active years: XXXX – XXXX
insert image
All artists
name:
description:
active years: XXXX – XXXX
image
albums by artist
year released
"album (XXXX)"
"album (XXXX)"



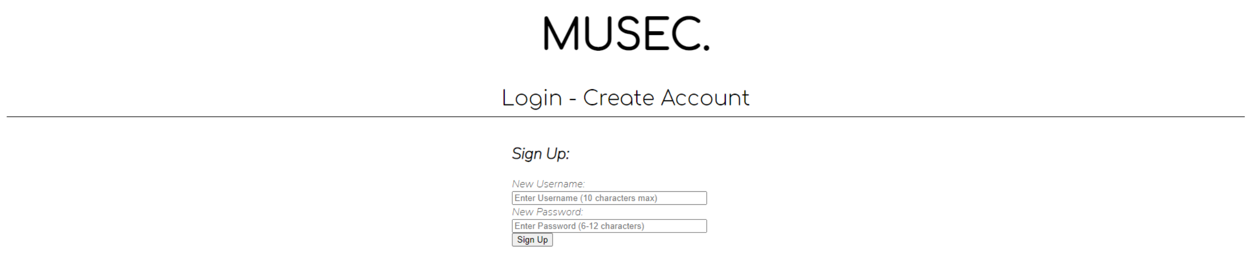


Final layout:

Website Layout Changes

My original plan was to have a single login box in the top corner of the profile page, but after designing my website to need user info in order to display associated data and in the interest of functionality, I instead made my login and create account separate pages all together.

[def create_account():] [@app.route('/')]


MUSEC.

Login - Create Account

Sign Up:

New Username:

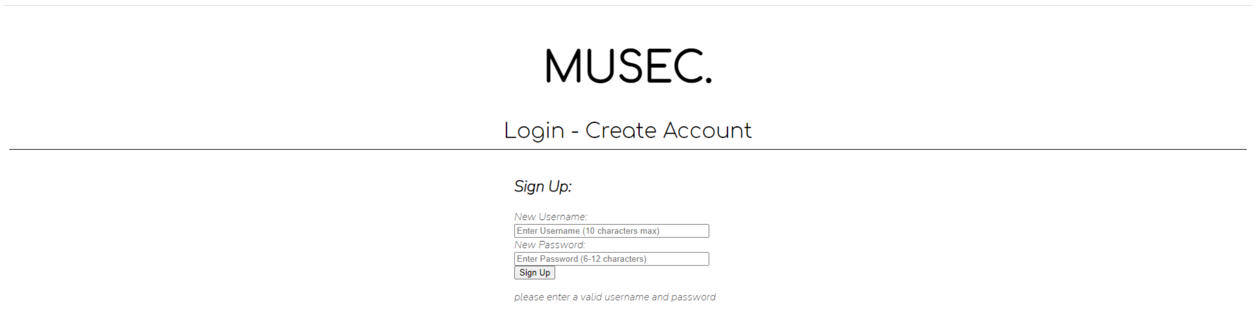
Enter Username (10 characters max)

New Password:

Enter Password (6-12 characters)

Sign Up

If nothing is entered on clicking Sign Up


MUSEC.

Login - Create Account

Sign Up:

New Username:

Enter Username (10 characters max)

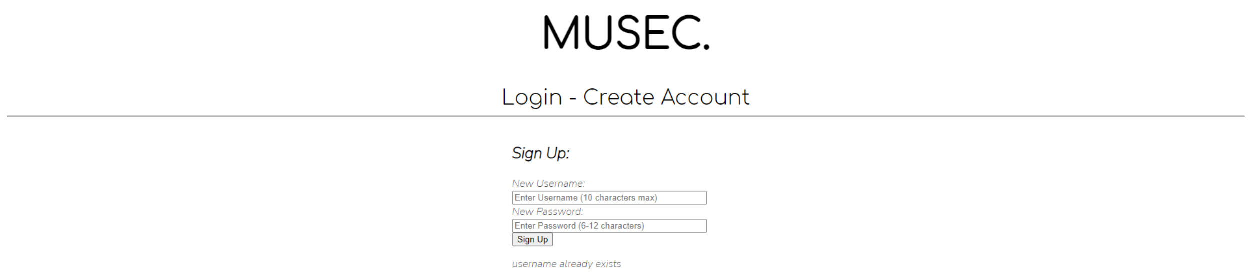
New Password:

Enter Password (6-12 characters)

Sign Up

please enter a valid username and password

If username already exists in the database


MUSEC.

Login - Create Account

Sign Up:

New Username:

Enter Username (10 characters max)

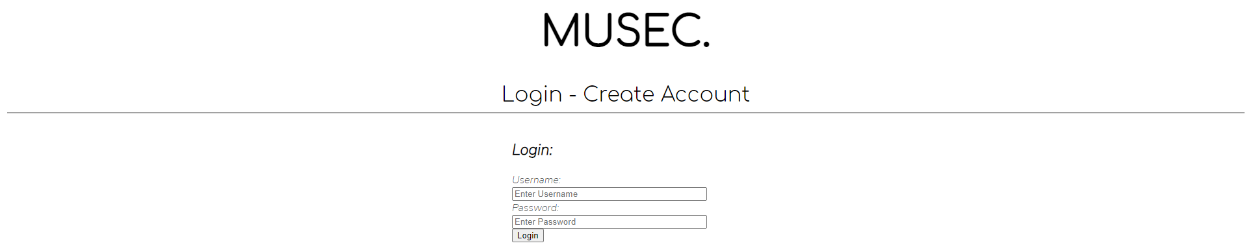
New Password:

Enter Password (6-12 characters)

Sign Up

username already exists

[def login():] [@app.route('/login')]



MUSEC.

Login - Create Account

Login:

Username:

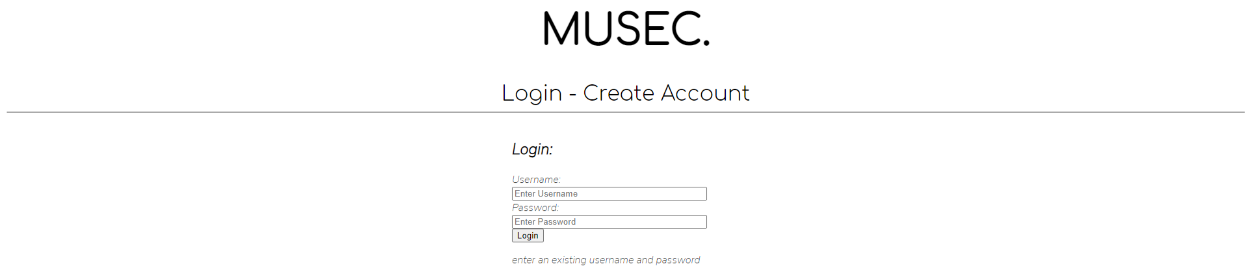
Enter Username

Password:

Enter Password

Login

If username or password is wrong/doesn't exist in database



MUSEC.

Login - Create Account

Login:

Username:

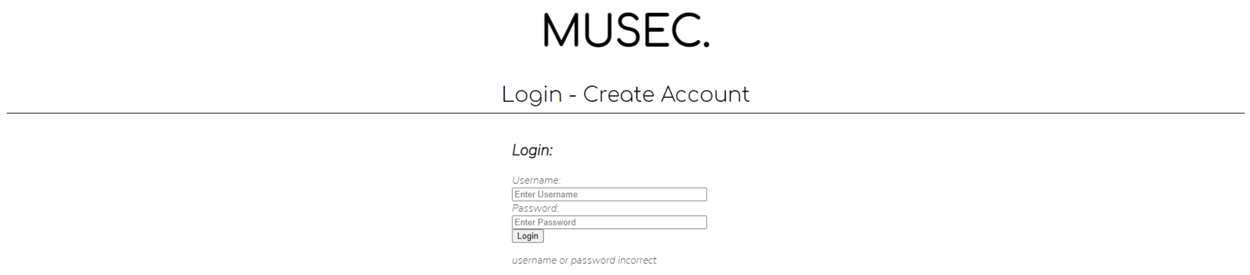
Enter Username

Password:

Enter Password

Login

enter an existing username and password



MUSEC.

Login - Create Account

Login:

Username:

Enter Username

Password:

Enter Password

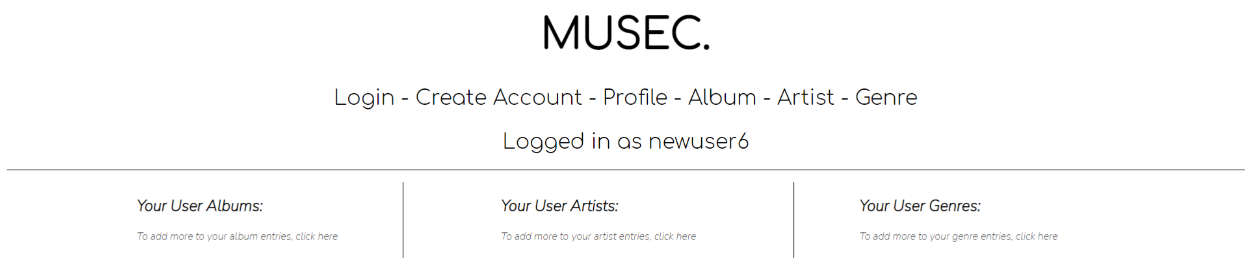
Login

username or password incorrect

As for the profile page, seeing as there are three separate tables that the user can add to, I made it so that there are three columns, for albums, artists and genres respectively. This makes a better user experience by not having to scroll through a bunch of unwanted information before getting to what the user wants, and also takes up more space on the page so there is no unnecessary blank space being unused.

[def profile():] [@app.route('/profile')]

For a new user


MUSEC.

Login - Create Account - Profile - Album - Artist - Genre

Logged in as newuser6

Your User Albums:

To add more to your album entries, click here

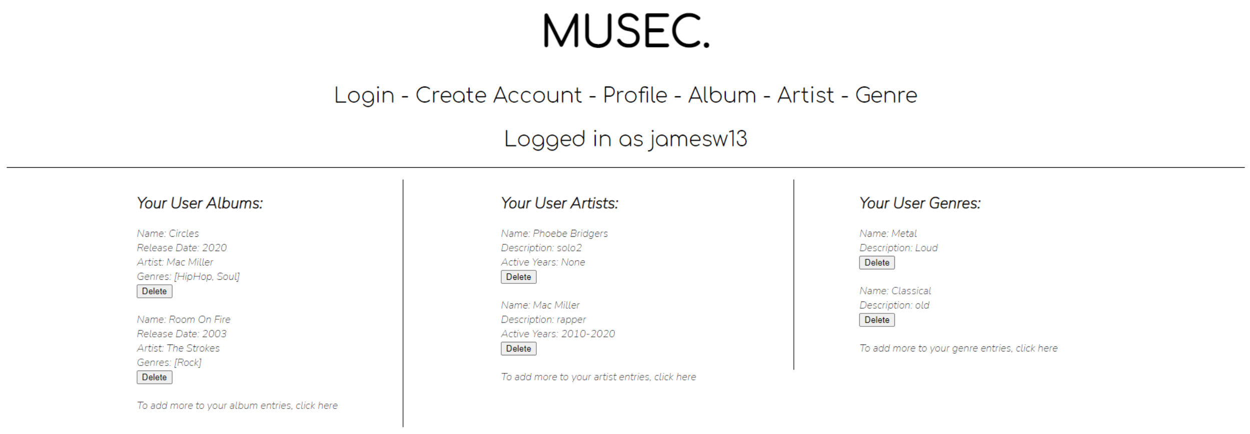
Your User Artists:

To add more to your artist entries, click here

Your User Genres:

To add more to your genre entries, click here

For user with some entries


MUSEC.

Login - Create Account - Profile - Album - Artist - Genre

Logged in as jamesw13

Your User Albums:

Name: Circles
Release Date: 2020
Artist: Mac Miller
Genres: [HipHop, Soul]
Delete

Name: Room On Fire
Release Date: 2003
Artist: The Strokes
Genres: [Rock]
Delete

To add more to your album entries, click here

Your User Artists:

Name: Phoebe Bridgers
Description: solo2
Active Years: None
Delete

Name: Mac Miller
Description: rapper
Active Years: 2010-2020
Delete

To add more to your artist entries, click here

Your User Genres:

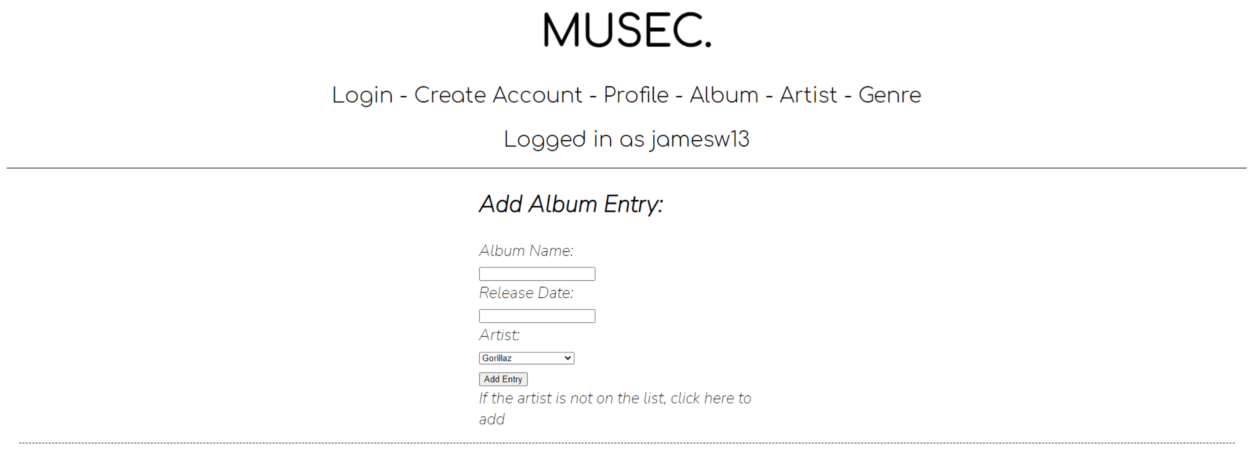
Name: Metal
Description: Loud
Delete

Name: Classical
Description: old
Delete

To add more to your genre entries, click here

For the album, artist and genre pages, the entry box is in the top centre as originally planned, and is separated from the entry list by a dotted line. Again, instead of lining everything directly under each other, I lined entries up into groups of threes. Each page has a fundamentally identical layout.

[def album():] [@app.route('/album')]


MUSEC.

Login - Create Account - Profile - Album - Artist - Genre

Logged in as jamesw13

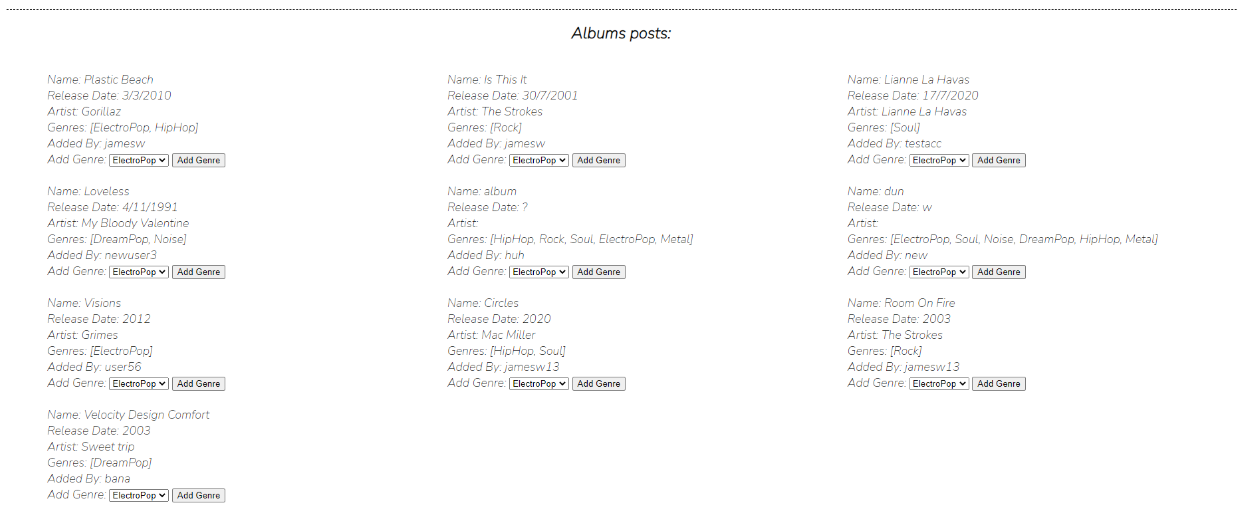
Add Album Entry:

Album Name:

Release Date:

Artist:
Gorillaz
Add Entry

If the artist is not on the list, click here to add


Albums posts:

Name: Plastic Beach
Release Date: 3/3/2010
Artist: Gorillaz
Genres: [ElectroPop, HipHop]
Added By: jamesw
Add Genre:

Name: Is This It
Release Date: 30/7/2001
Artist: The Strokes
Genres: [Rock]
Added By: jamesw
Add Genre:

Name: Lianne La Havas
Release Date: 17/7/2020
Artist: Lianne La Havas
Genres: [Soul]
Added By: testacc
Add Genre:

Name: Loveless
Release Date: 4/11/1991
Artist: My Bloody Valentine
Genres: [DreamPop, Noise]
Added By: newuser3
Add Genre:

Name: album
Release Date: ?
Artist:
Genres: [HipHop, Rock, Soul, ElectroPop, Metal]
Added By: huh
Add Genre:

Name: dun
Release Date: w
Artist:
Genres: [ElectroPop, Soul, Noise, DreamPop, HipHop, Metal]
Added By: new
Add Genre:

Name: Visions
Release Date: 2012
Artist: Grimes
Genres: [ElectroPop]
Added By: user56
Add Genre:

Name: Circles
Release Date: 2020
Artist: Mac Miller
Genres: [HipHop, Soul]
Added By: jamesw13
Add Genre:

Name: Room On Fire
Release Date: 2003
Artist: The Strokes
Genres: [Rock]
Added By: jamesw13
Add Genre:

Name: Velocity Design Comfort
Release Date: 2003
Artist: Sweet trip
Genres: [DreamPop]
Added By: bana
Add Genre:

- routes* / function signatures** for each page
  - @app.route(‘/’), def create_account()
  - @app.route(‘/login’), def login()
  - @app.route(‘/profile), def profile()
  - @app.route(‘/genre’), def genre()
  - @app.route(‘/artist’), def artist()
  - @app.route(‘/album’), def album()
- Fonts / colours used
  - Nunito sans for all titles and text
  - Comfortaa for website header and navbar

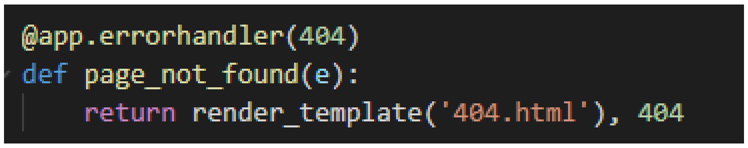
Testing table

What is being tested	**How you are testing it**	**Expected results**	**Actual results**	**Pass/fail/notes**
Showing info from the database on the website that is filtered by the users id (e.g album added by user1 is shown when url is	The user page is set as user/<int:id>, and shows all info that so for example when the url has /user/1, all entries from user	All results from the specified tables filtered by the user id, will be displayed on the website.	Data is displayed, but as values instead of strings, showing each entry as a list.	Pass, but I need to use some repr functions and add some name values to my models.py in order to display the data in a sensical way that

/user1)	1 can be seen. This is done with an sqlalchemy filter_by query.			is understandable to the user.
Add user on create account	User and password is entered in text boxes. Password is hashed on submit. Then both are added and committed to the database.	On click submit.	Works as intended	Pass
Storing user info in a session on login.	Using flask login manager and session, I can take the user's id and username on signing in and store it in a session to use across all my page functions. I can now use values from the session to display the same results without using a url. I am printing both the user id and name from the session on the page so I can clearly see if the session has worked.	On clicking login	The user name and id are stored across all pages, but username is displayed as a query.	Pass, but must add repr function to models to display username correctly.
Error messages for invalid create accounts or login attempts	In html there is an if statement determining if ‘errorMessage’ has a value. Under each submit function is an else statement that will set this value depending on what the error is (user exists in database, fields are empty).	On login, page will reload with a new message displaying an error.	Works as intended	Pass
Adding entries to the database using forms.	For my albums page, under add entries there is a text box for	On clicking the submit, the page will reload and underneath you	Works as intended.	Pass

	name, release year and a select field for artist, which contains every artist in the database. Underneath is a submit button that validates the entries and reloads the page.	can see the entry added to the list of entries, which is a simple sqlalchemy query that fetches all entries. The data should be visible in the database.		
Deleting entries	For each user post under the user profile, I added a delete button. This was part of a form, which the main code would get the value of the id that the post had. Depending if the post was an album, artist or genre, the code would match that id and delete the entry from the database.	On clicking delete, the page would reload and the post would no longer be visible and would be removed from the database.	Works as intended.	Pass.
Adding genre to an existing album and vice versa	After setting up my many to many between albums and genres, I added a drop down list for each album entry, showing all the genres. In my html, I have a hidden value that finds the id of the album. Then after clicking the submit button, the genre selected is added to the album that is of the same id that the submit is an element of.	On clicking submit, the page will reload and the genre will be appended to the list for the queried album.	Works as intended.	Pass, repeat the process for adding albums to a genre.
Links to urls	In header.html there are links to every url on my website using href = ‘/url’.	When hovered over with a mouse the link will be blue and underlined and	Works as intended.	Pass

	There is css that makes each link blue and underlined when hovered over with the mouse.	will redirect to the url linked.		
404 page	Make a basic 404.html, then the below code to add a 404 page.	When the url does not match an existing page, redirect to 404 page	Works as intended	Pass



```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404
```

Stakeholder feedback:

User experience:

On my first run through marking a few things were suggested that would improve user experience with my website. First was making some more hand holding features for new users, as at the time, when a new user is redirected to their profile, three lists for albums, artists and genres are shown but no data is shown because the user hasn't got anything attached to their profile. Therefore a new user has to go out of their way to look for a way to add to their posts. I addressed this by adding labeled links with descriptions telling the user what to do on my profile page. The links take the user to each album, artist and genre page, which all have entry fields in large font size at the top centre of the page which is what the user will first see.

At the time of my feedback, the way users could add a relationship between albums and genres was by going to the genre page, and under each genre entry would be a select field where the user would select an album existing in the database. It was suggested that this could be done vice versa for the album page (add a genre under an album entry). This would make the website more flexible, and considering that the site had a strong user base, most genres would have been added quickly. Adding this feature was simple and quick to do, because the many to many relationship was already functional, and I only had to recreate the same function, changing the values.

A classmate suggested that I should add an 'are you sure' box when deleting entries in order to make decisions easier and reversible for users. This was simple to implement and made the user experience better.

Relevant implications: Database

Privacy:

Considering that this website requires an account to use, and this includes a username and password, I had to address security in my database. Making sure that I could not see other users' passwords for privacy reasons was very important. I addressed this by generating a hashed code for a new user whenever they create an account, and store that in the database so anyone with access to the database cannot see it. I did this by importing werkzeug.generate_password_hash, and made it so that to log in the code would check the entered password and if it matched the generated code in the database, the user is successfully signed in.

End User Requirements:

It is important for my website to have a suitable database that is efficient and meets the end user requirements. Addressing the structure of my database in order to achieve these requirements. I first addressed this by integrating a linking table between the album and genre table, in order to establish a many to many relationship to address the complexity requirement. Originally I was planning to use a linking table between every table in the database but after trying to display this info on the website, I found that there was too much data being unused and was unnecessary for the end user requirements. All the data is shown on the website at some point aside from privacy sensitive information like passwords. My model classes are all flexible, and adding new functionality to the website is simple to implement because of it.

Relevant implications: Website

Usability:

In order to make my website usable for new users and easy to navigate, I included some links to tell the user where to go to add new entries. For example, when a new user is first logged in, they will be redirected to their profile page, where all their entries are logged. But a new user will not have any entries entered, so I needed to implement some HCI principles to make sure a new user is not confused on what to do next. Under each column (album, artist and genre) there is a link to the associated page labeled "To add to your __ entries, click here". This link is also highlighted blue and underlined when hovered over with a mouse, making it clear it is a link and is clickable (as is every link on my page). Each page has its entry field in large text at the top centre of the page, which is what the user will first see.

The album page also requires the user to choose from the available artists in the database, so I added another link to the artist page right under the select field to show the user how to add an artist before adding an album entry.

Privacy:

When a user first creates an account, or is logging in, they should not be worried about someone seeing their password as they enter it into a text field. I addressed this by making any password fields on the page censor the users input, so nobody can see the users password as they enter it.

I secured my users profile and info on the website by storing their information in a session. Using a session instead of a specific url or global values, allows the user's account details to be called into any function across different pages and reloads, and also prevents other users from accessing their account by just entering a url. When a user logs in, their username and id in the database is stored in a session, and each page function calls that session to use the users id to display the matching user data (all the users added entries). The users password is not stored in this session as it is unnecessary to use, and the user should not worry about their password being stored and easily seen by anyone with access to the database or session.