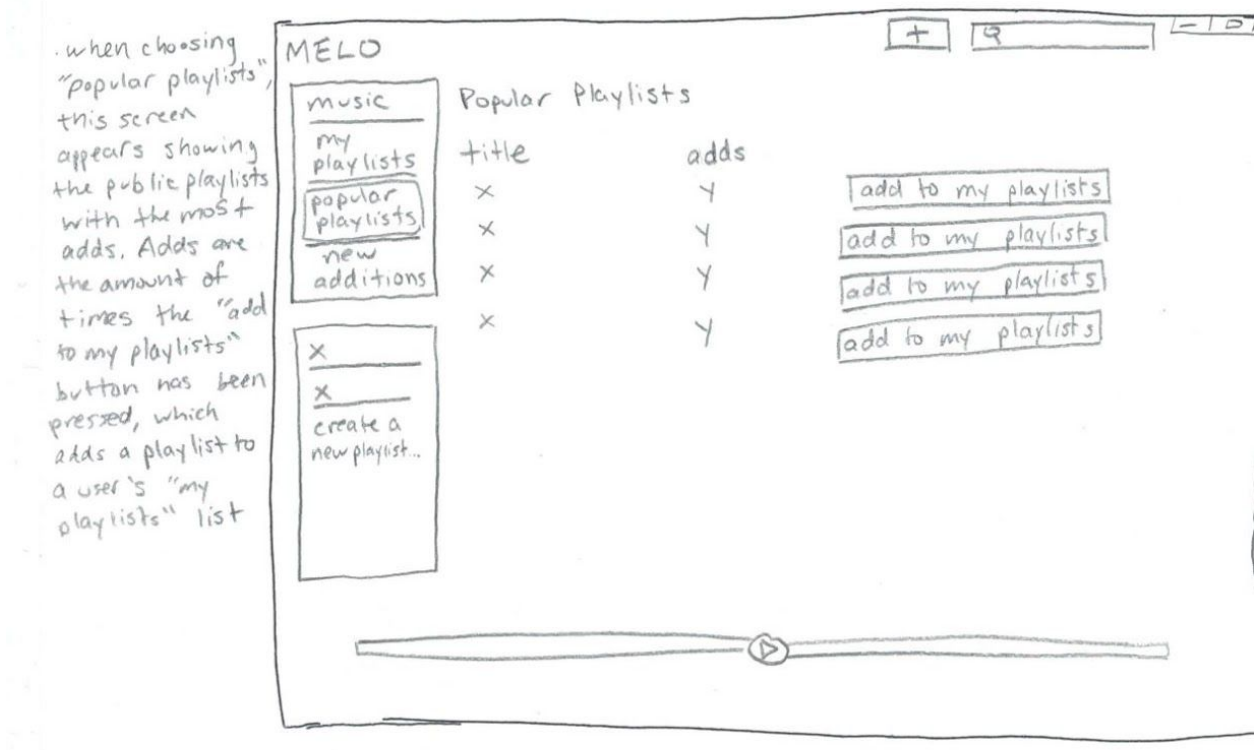
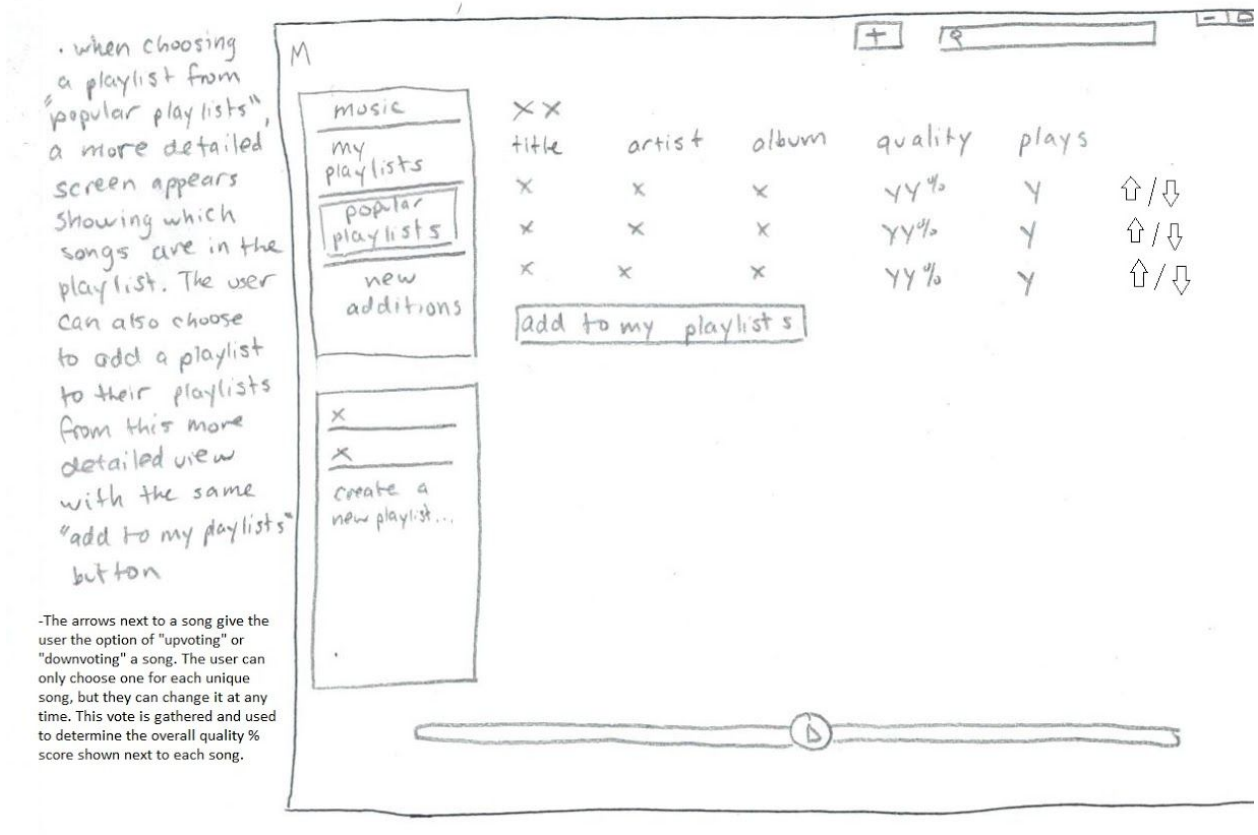
Nick Schillaci
Rex Cummings
Connor Lyons
Carson Murray
Drew Whittaker

MELON HEADS
Design Document

I. SUMMARY SECTION: The goal of our group's project is to create an application that is able to provide a local, central hub for collaborative, external music playlist building. It will be an all-in-one desktop music application designed to create and manage playlists built from music hosted on a variety of sites. The program will accomplish this by taking in links to music and their respective metadata and storing them in a database which the user indirectly interacts with through a front end desktop application that allows them to build playlists and play the music only using the application. The program solely stores links and does not handle actual music files.

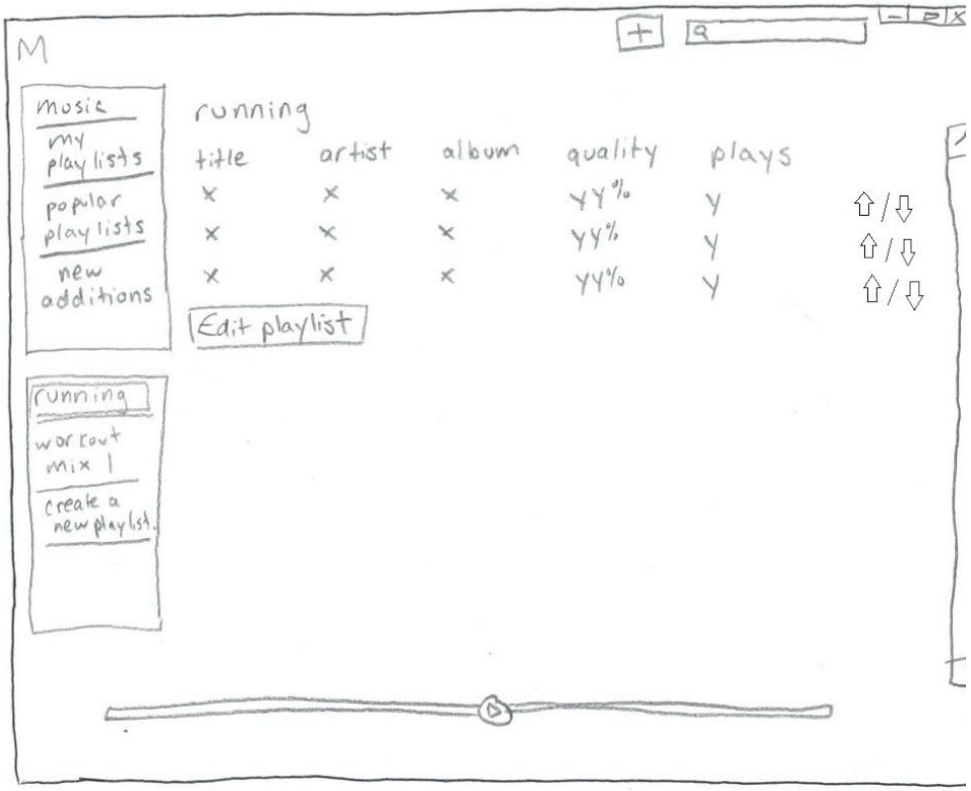
II. SCREEN/VISUAL SECTION:



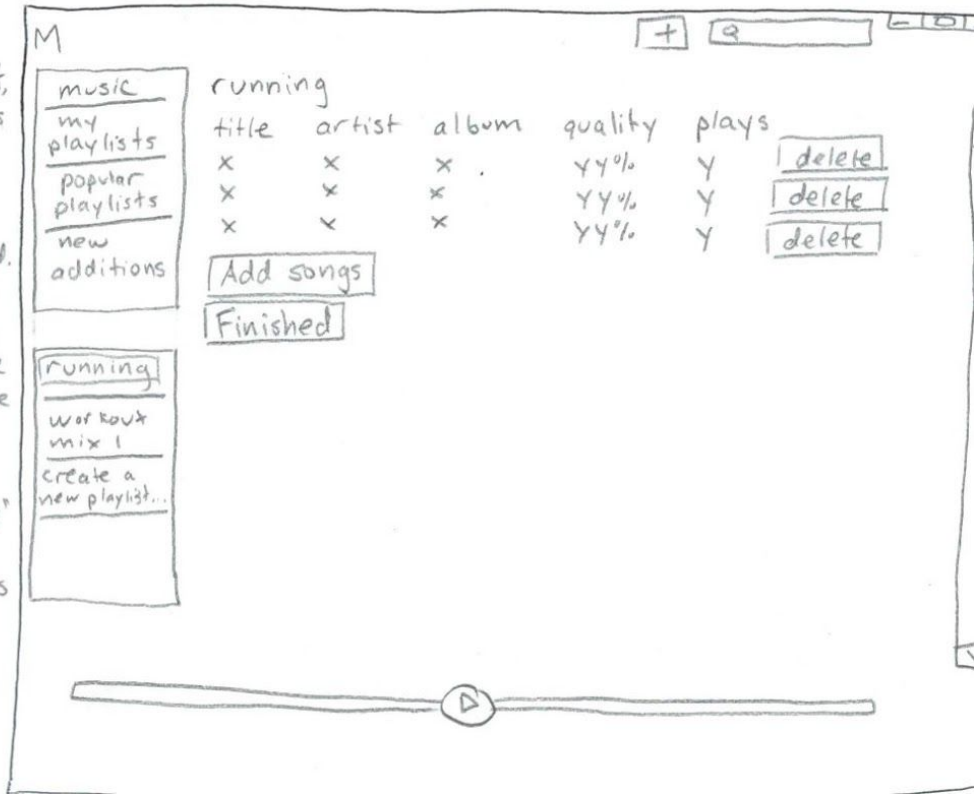


-The arrows next to a song give the user the option of "upvoting" or "downvoting" a song. The user can only choose one for each unique song, but they can change it at any time. This vote is gathered and used to determine the overall quality % score shown next to each song.

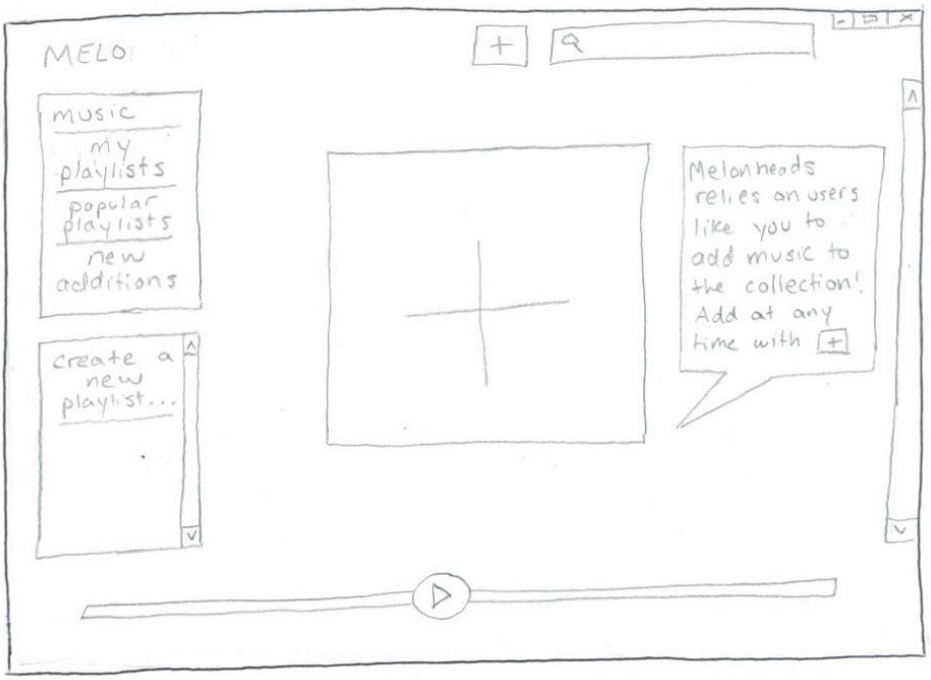
- When a playlist is selected, this screen is the result. Choosing "edit playlist" results in screen below



- After choosing to edit a playlist, the screen changes to this, where existing songs are given the option to be deleted. Add songs takes the user to the "music" tab where songs they choose and click are added to the playlist. "Finished" exits from editing playlists

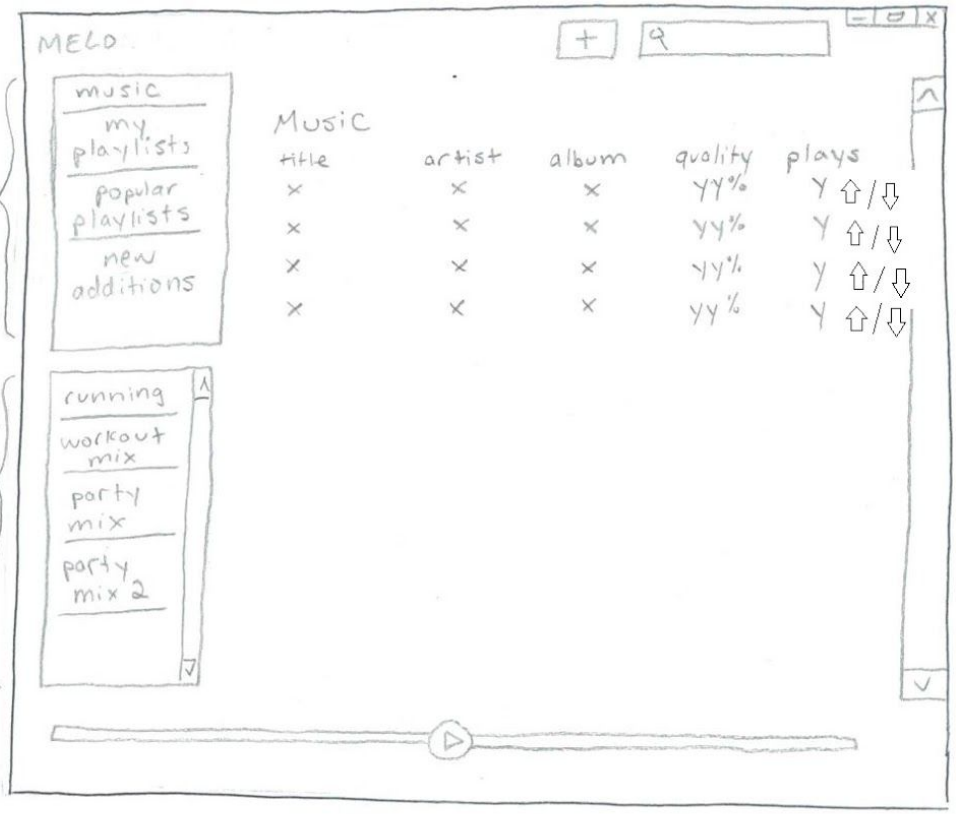


- The initial start-up screen when the application is opened each time. Clicking anywhere but the + removes it and the text bubble from the screen so you have access to the info underneath it.

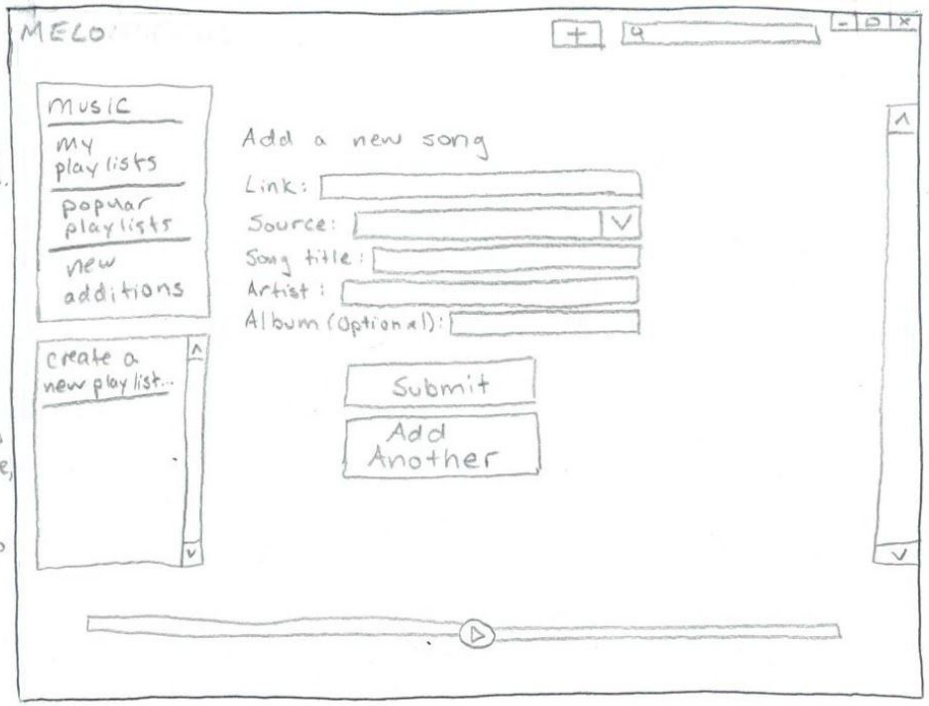


This is the quick reference box. It always contains these 4 item selections, which when chosen display in the center. The example here is "music" has been chosen

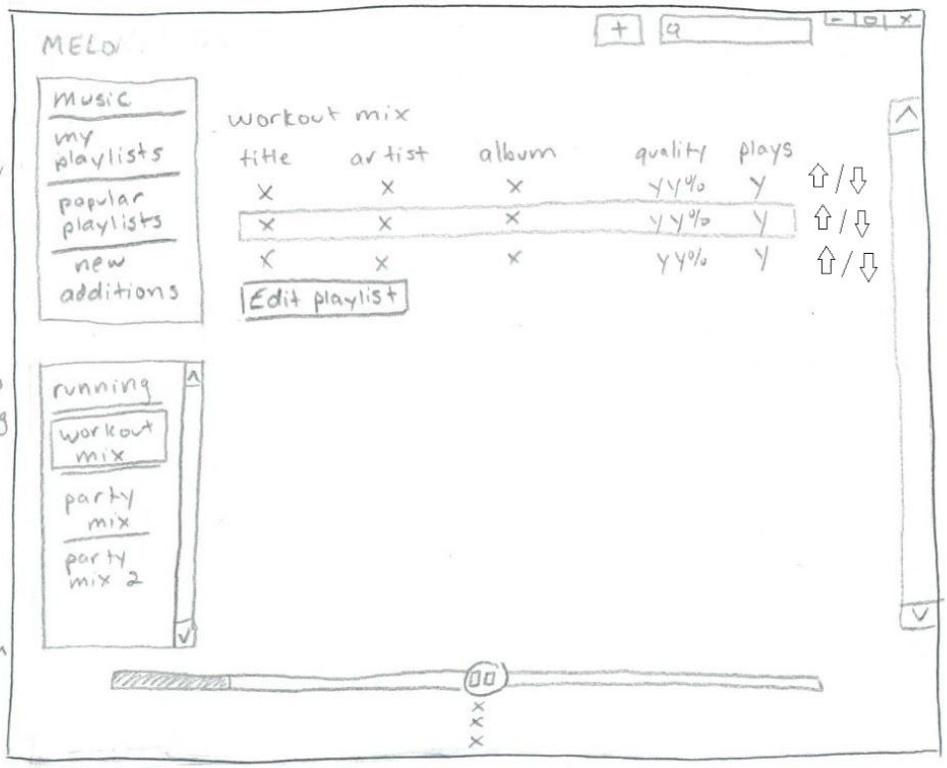
This is the playlist reference box. User created playlists will be displayed here and can be accessed/ brought to the main in one click

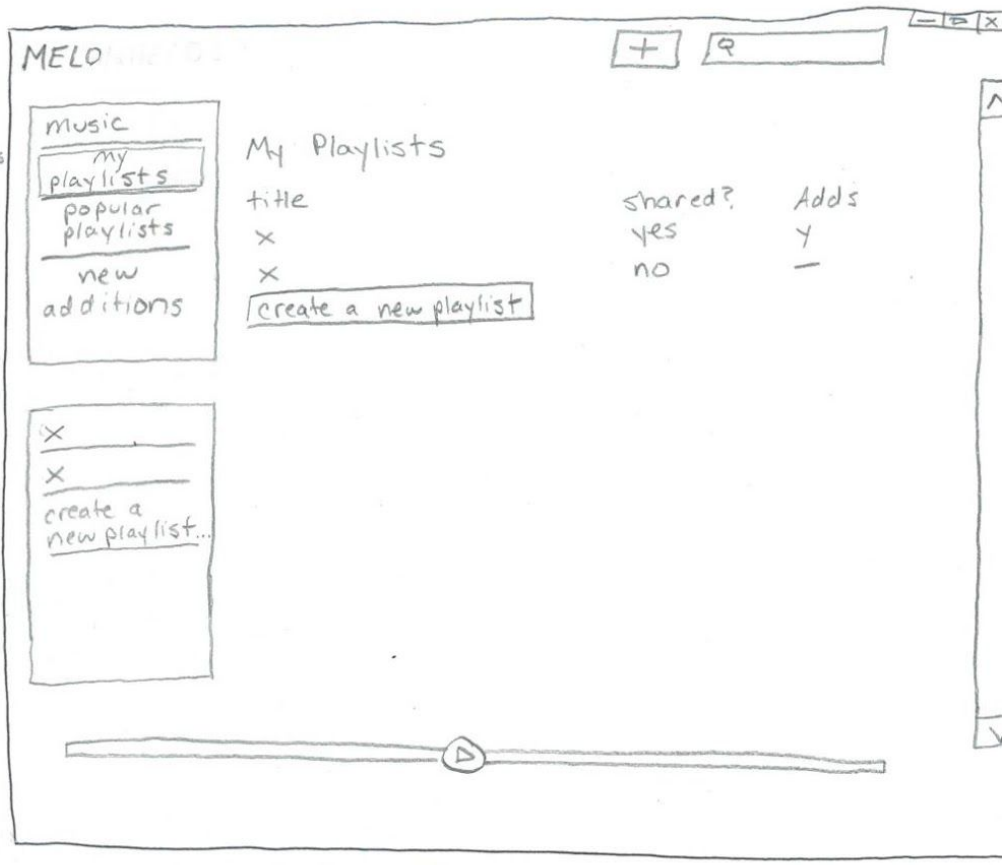


Rough sketch of what it looks like to add a song by clicking [+]. Link, title, artist, and choosing a source from the dropdown box are required to add a song. Submit submits song to database, add another submits and clears fields to quickly add another.

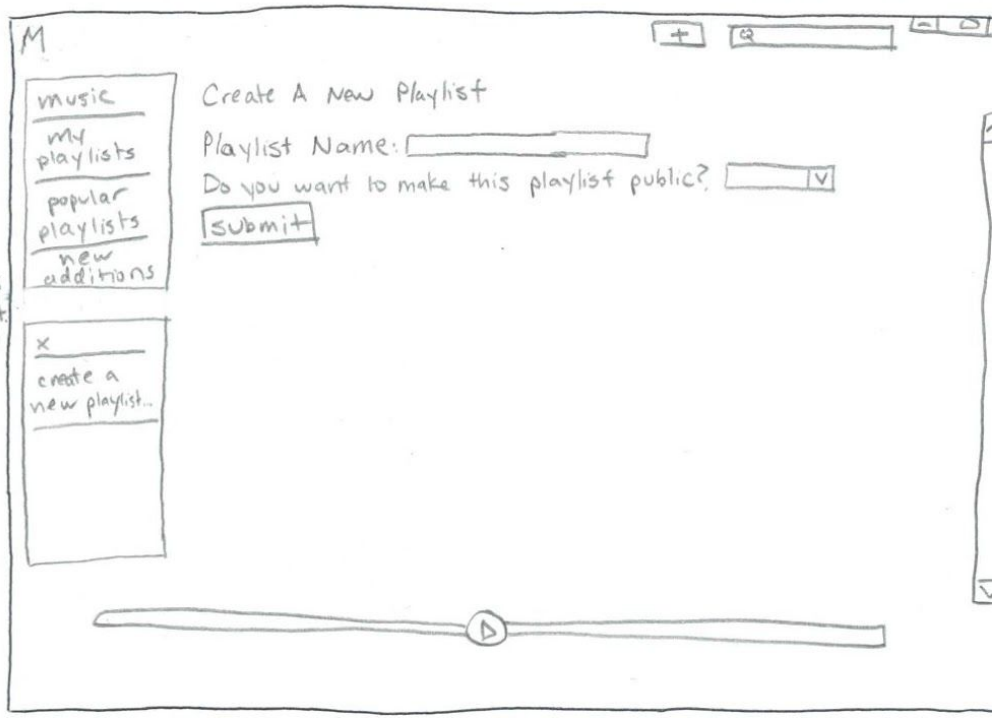


Rough sketch of what it looks like when a song is being played, in this case from a playlist created by the user. The play button turns into a pause button and the progress bar tracks along with the duration of the song. The song's title, artist, and album if available are shown beneath it.





·when choosing "my playlists", the main screen displays the names of playlists in a user's "my playlists" list, stored client side. It also displays whether that playlist is "shared", meaning, if yes, it has been added to the MelonHeads database of public playlists accessible by all users. "Adds" will only display the number of times the playlist has been added to a user's "my playlists" list



·when choosing the "create a new playlist..." option in the "my playlists" section or from the playlists reference box, this screen is the result. The user is prompted for the playlist name and if they wish to make the playlist public for other users to find and listen to. Submit commits the form and returns to my playlists and adds playlist

Use Case #	As a...	I would like to...	Completed by:
1	User	Add songs to the MelonHeads database	Using the “+” button that is visible on the interface at all times, the user can input the song link and metadata to add the song instantly to the database as long as it is accessible through one of the services our application can interface with.
2	User	Listen to music	By accessing the “music”, “my playlists”, or “recent additions” tab on the quick reference tab, the user will be given a list of music available for listening through MelonHeads.
3	User	Create playlists for quick listening	By clicking on the “create a new playlist” option in the playlist reference box, the user can create, name, and add music to a new playlist. By clicking on the name of an existing playlist in the playlist reference box, the user can edit their playlists at any time as well.
4	User	See what music has been added to MelonHeads recently.	By accessing the “recent additions” tab, a user can see the most recent additions

			to the MelonHeads database, which they can listen to or add to a playlist.
5	User	Mark a song as bad quality/mislabeled	The downvote arrows located next to every song will contribute to a song's overall quality score which can be viewed by everyone, notifying users of a song's poor quality
6	User	Mark a song as good quality/I enjoyed it	The upvote arrows located next to every song will contribute to a song's overall quality score which can be viewed by everyone, notifying users of a song's superior quality
7	User	Save a popular playlist created by another user for myself for later listening	When browsing through "popular playlists", sorted by the amount of times someone has done exactly this, you have the option of adding the playlist to your "my playlists" list, allowing for easier access at a later time
8	User	Share my playlists with other users	When creating a playlist, there exists an option to make the playlist public and share it to the Melonheads database in the playlist table. From here, other

			users can view, listen to, and add to their collection your playlist.
9	Admin	Remove song/playlist.	Admin will directly access and modify the database to remove songs or playlists.

III. RESTFUL SERVICE SECTION:

Endpoint:

- POST /songs

  Purpose: Create a new song entry in the database with the given inputs, or update an existing song item

  Inputs: title, artist, album, URL, source

  Output: Create new song entry in the database, or update existing one with the same name

  Implementation: Join the song's information into a single entry in the 'songs' database table

- GET /songs/:filterType:filter

  Purpose: Retrieve a particular song (by ID) or songs falling under a certain filter

  Inputs: filterType, filter

  Output: JSON representation of songs returned from the database and converted to Song objects with respective information

  Implementation: query the database for a list of songs that fall under a given filter (categorized by filterType)

- DELETE /songs/:id

  Purpose: delete song will be used to remove a song from the database.

Inputs: id (must have the specific song ID)

Output: deletion query for the specific song in the database

Implementation: Used for management purposes, a specific song (references by unique ID) is deleted from the database

IV. DATABASE SECTION:

Table: Songs

Song ID	Title	Artist	Album	URL	Quality	Plays	Source
Numerical value assigned to a song, it is a unique integer based solely on when the song was added relative to all other songs. IE first song ever added is 1, second is 2, etc.	The song title, as inputted by user.	The artist responsible for the song, as inputted by the user.	OPTIONAL: The album the song resides on, as inputted by the user.	The URL the song resides at from one of the pre-determined sites and services MelonHeads can interface with.	Quality refers to the score given to a particular song by user votes. Users are given an option to upvote or downvote a song. The upvotes and downvotes are then combined and form a percentage of upvotes to total votes and displayed as a percentage to all users.	The number of plays a song has received. One play is added every time a song is played from a playlist or clicked on by the user.	Where the song resides and where it is accessed by MelonHeads. IE Youtube, Soundcloud. This is chosen by the user in a dropdown menu, as the options are pre-determined and limited.

Table: Playlists

ID	Songs	Adds
Unique playlist ID #, assigned in the same way song ID # is assigned.	A list of song ID #s that the playlist contains	A number indicating the number of times a playlist has been accessed by users. This determines its ranking in the "popular playlists" tab.

V. IMPLEMENTATION PLAN SECTION:

1. Front-end: (Divided amongst Connor, Carson, and Drew)
   Note: The functionality mentioned below is worded for when the product is fully operational. For our midway point we aim to purely just create the shell of the application. The usability of buttons and actions won't be fully operational by this point. Functionality will be added incrementally and in mass once the database and API are integrated thoroughly.

   Homepage contains a **top**, **bottom**, **central display console region** (the largest region), and a segmented **left region**. For each region the buttons and tables will be visible on by the midway point, however each one will not work as designed yet. Our application will be able to add a song using the top region's add button but it will not be displayed in the center region. All tables will be purely visual and not functioning yet. The goal is to be able to add songs to the database, then in the next phase display the updated playlists and remaining components.
   a. **Top region** contains an add button and a search bar.
      i. The add button directs the user to a page from which they can add a song to the database by inputting the URL, title, artist, source, and optionally an album name.
      ii. The search bar will search through all the music added to the database from this tab. In the future, we hope to add the ability of the search bar to auto-generate possible candidate songs based on what user has currently input.

b. **Center region** will display the sidebar entry that is currently chosen. For example, if "popular playlists" was chosen, then a table of the popular playlists would be displayed in table format. The console provides the user with the option to edit the desired playlist to their preference. Editing involves adding or removing songs from a playlist of choice. In the future, we hope to allow the user to modify the name of the playlist they created themselves.
    i. A scroll bar will be available for sidebar entries that contain enough elements to warrant the use of the scrollbar.
c. **Left region** will be two separate sidebars, options list and playlist list.
    i. The top side bar will contain entries for music, my playlist, popular playlist, and new additions.
    ii. The bottom side bar shall contain a current list of all the created playlists.
        1. If there are no custom playlists, then only a create playlist entry will be available.
        2. A scroll bar be available as the list of playlists get populated.
d. **Bottom region** has a progress bar based on if a song is being played and shows how much time has elapsed. Also, contains a play/pause icon in the center of the progress bar.

2. Back-end: Functioning database (Divided amongst Nick and Rex)

The MySQL database will be hosted on AWS. Will contain two tables named song and playlist. We will populate the database with sample data for testing purposes to test functionality later on. Putting in raw data will enable us to test in case errors arise when users add new songs via some external source such as Youtube. For the midway point, we will add all the fields to the table as displayed in the Database section above. Also, using test data we will simulate a user adding a new song to the database.

a. Song table:
    i. Primary key for a song: song_id, song title, artist
b. Playlist table:
    i. Primary key for a playlist: playlist_id, song list

3. Back-end: Functioning RESTful API: (Divided amongst Nick and Rex)

At the midpoint of the implementation phase, our application will be able to utilize Youtube's official embedded player functionality and retrieve data about the song a user wishes to play from our RESTful API. The user will be able to add a song using Youtube as the source. The database will update accordingly depending on whether the user inputs a valid song entry.

VI. TECHNOLOGY STACK:

The programs GUI will be written in C# developed in Visual Studio. It will connect to API via HTTP requests and utilize official source embedded players to access and play music.

The Database will be written using MySQL and hosted on Amazon AWS. For the API it will also be hosted by Amazon AWS VM and updated via git repository. Our RESTful API will be built in Java and utilize SparkJava libraries. It will also handle request with HTTP.