

# Motivation

**Brief description:** A music app that chooses songs automatically for groups of listeners based on each group member's preferences.

**Problem:** People listening to music together want music that suits everyone's tastes.

**Deficiency in current solutions:** Pandora, Spotify, and YouTube are the most common music-listening platforms. The closest existing solution is Spotify's collaborative playlists, but these take time to make, and are generally targeted to smaller groups of 2 or 3 people. Thus, the common options when people listen to music together are to choose one person's playlist (which only makes one person happy), or to create a playlist specifically for this group of people (which takes a lot of time, is inconvenient, and not very general).

**Solution:** Automatically-created shared/collaborative playlist made and updated in real time as people join.

**Purposes:**

- 1) To keep track of and save information about people's music preferences: We want to allow users to input information about what type of music they like. They can also edit these preferences at any time. WeTube remembers these preferences when selecting music later.
- 2) To allow users to specify songs or artists they would like to hear played in gatherings: Then WeTube knows to play songs either in their "liked songs" list, or songs by artists in their "liked artists" list.
- 3) To specify people whose music tastes should be taken into consideration when WeTube chooses songs for the group: This means only users actually in that location will get to hear songs they like.
- 4) To allow users flexibility in which songs are played at a certain time: Users may find certain songs offensive or dislike certain songs; WeTube allows users to skip the current song when enough users dislike it.

# Concepts

**User:**

*Definition:* A person who has a WeTube account.

*Purpose:* To keep track of and save information about people's music tastes.

*Operational Principle:* When a person creates an account on WeTube, they are now able to input music tastes and join gatherings.

**Music Profile:**

*Definition:* Information about a user's music tastes, specifically, a list of songs they like, and a list of artists they like.

*Purpose:* To allow users to specify songs or artists they would like to hear played in gatherings they join.

*Operational Principle:* When users are in a gathering, WeTube will choose songs that are either in the members' lists of favorite songs, or songs by artists in the members' lists of favorite artists.

**Gathering:**

*Definition:* A group of people in the same location who want to listen to music together.

*Purpose:* To specify people whose music tastes should be taken into consideration when WeTube chooses songs.

*Operational Principle:* When a gathering is created, users can join the gathering. The music played for that gathering will be based on the joint music preferences of these users.

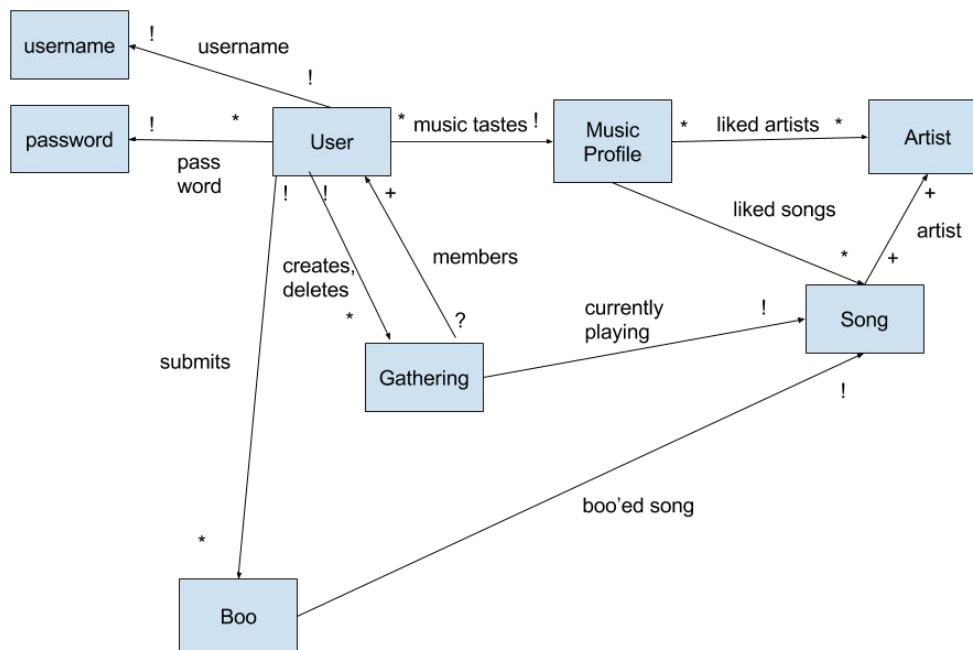
**Boo:**

*Definition:* A vote by a user indicating that they do not like the currently playing song

*Purpose:* To allow groups to skip songs they're uncomfortable with or dislike.

*Operational Principle:* Hosts can choose to enable booing or not; if enabled, and sufficiently many users boo a song (the exact value of the threshold is up to the host), the song will stop playing and WeTube will choose a new song. This song will not be chosen again for the gathering.

## Data Model



Textual Constraints:

A user can only submit a boo for a song that is currently playing for the gathering that the user is a member of.

A user who creates a gathering must be a member of that gathering.

The song that is currently playing for a gathering must be in the music profile of one of its members.

## Security Concerns

### Key Security Requirements:

- User's accounts and passwords should be secure.

### Preventive Measures:

- We will hash the passwords with a salt on our server to prevent against rainbow chain attacks.

### Mitigating Standard Attacks:

- XSS/Injection: We will escape all user input such as their favorite songs and artists.
- CSRF: We will use CSRF Tokens to prevent people from forging identities.

### Threat Model:

- Because there is no money flowing through the site or confidential personal information stored on the site, advanced hackers are unlikely to want to attack our site. Our attackers therefore are probably just everyday programmers who want to figure out someone else's music preferences.

# User Interface

(drawing below)

1. sign in page
2. My profile, my gathering, finding other's gatherings tab
3. my profile to set name and preferred songs, artists
4. My gathering to see current gatherings I'm in or to create a new one
5. Other's gathering to see enter and see their current and next songs, and its playlist
6. Find gathering by searching the host name or its url+shoutkey

PA

---


USERNAME

PW





sign in  
REGISTER





sign out

my profile my gathering other's gathering



MY PROFILE  sign out

NAME: MIRI CHO





SONGS:    

ARTISTS:    

Top-left songs in highest priority.

  ...

MY GATHERING sign out


CURRENT:    

END  
END  
END

CREATE NEW!

clicking leads to  
NEXT PAGE →

AQUA GATHERING sign out

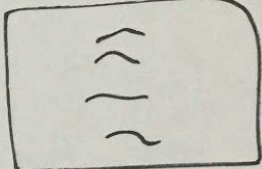


current:

NEXT:

PLAYLIST:

Playlist 1 sign out



CREATE NEW sign out

NAME:

showkey:

INITIAL:

IMAGE:

FIND GATHERING sign out

NAME:

HOST

OR

showkey:

# Design Challenges

When designing the app, we had to make important decisions about what data to store and how to store it, as well as what interface/functionality to provide the user. When making these decisions we had to take into account factors such as ease of implementation, ease of use, and fulfillment of purpose.

The first decision was *\*what\** data we should store. We need enough data to do an adequate job of song selection, but no extraneous data. For the somewhat simple song selection system we intend to implement for the MVP (namely, randomly select a user then randomly select a song), it's sufficient to store a list of favorite songs for each user. More complicated models of song selection, which make use of such information as favorite artists, boo history, and so on, require more data. Since songs will all have the same format (title, artists, etc. being the essential attributes) we decided to store the data in a MongoDB collection; as an added bonus, we are already familiar with Mongo.

The next big decision was what functionality to provide the user with. At first we had several ideas for how to deal with the biggest misfit - playing songs that certain members of the gathering *\*really\** don't like. We introduced the idea of a "boo" to enable members of a gathering to provide real-time corrective feedback to WeTube, which would skip a picked song that displeased enough of the members of the gathering. Precisely what is meant by "enough" - e.g. a fixed number or a fixed proportion of guests - we leave to the host. In fact, whether boos are enabled or not is left to the host.

We also originally included the concept of a ban list, where users can list songs that are objectionable to them (racist, explicit, etc.) that would be prevented from playing at any gatherings they were in. However, without checking whether a song is "objectively" offensive, this essentially enabled users to unilaterally perma-boo a song, and we decided that it would be too difficult to solicit input from the whole user base to decide if a song was offensive or not. Thus ultimately, we removed the ban list concept.

Finally, we had to decide on a way for users to join a gathering. We didn't want all gatherings to be public, since then random people could (adversely) impact the song selection of a gathering they were not invited to. On the other hand, requiring users to log in to access a gathering would require the host to coordinate their (e.g, Facebook) invitations with invites on WeTube, which is quite cumbersome. As a compromise, we decided to use a shoutkey-type system.