WeTube Revised Design

November 19, 2015

Alice Zhan, Jimmy Zhang, Miri Choi, Dhroova Aiylam

Motivation

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

Key Purpose:
The primary purpose of WeTube is to generate playlists that are amenable to the tastes of everyone in a group. WeTube will maintain information about individuals' music preferences, and use this information to select songs that are best for the group.

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.

Concepts

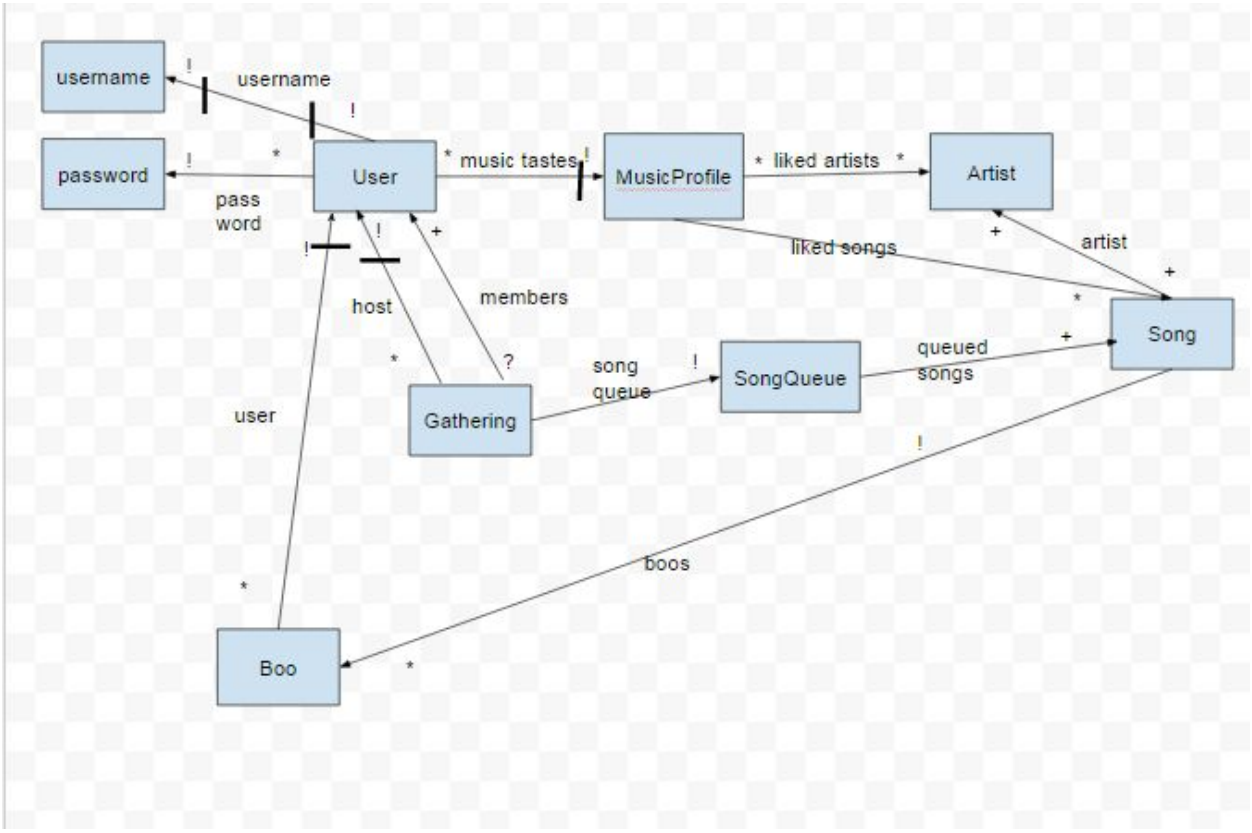
Music Profile:
Definition: Information about a user's music tastes, specifically, a list of songs they like, and a list of artists they like.
Purpose: To allow users to specify songs or artists they would like to hear played in gatherings they join.
Operational Principle: When users are in a gathering, 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 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.

Song Queue:
Definition: The next N songs, in order, which are to be played at a gathering
Purpose: To display, organize, and ultimately play, the songs WeTube has chosen for a gathering.
Operational Principle: A gathering displays the next N songs which are to be played; the exact value of N depends on the particular gathering. Users are able to boo songs which appear on the queue, and if a song receives enough boos it will be removed.

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 in the song queue for the gathering that the user is a member of.
- A user who creates a gathering must be a member of that gathering.
- The songs in the song queue must be in the music profile of one of its members.
- Note that if a user likes an artist and a song by that artist, that song should only be counted once.
- Only users who are the host for a gathering should have special rights (i.e., setting the boo limit, deleting the gathering, sending out invites).

Insights:

- Boos are associated with songs rather than users since they are used to remove the song. However, boos still need to record the user that created them to prevent a user from double-booing a song, or to use this data to better understand their music preferences.
- A liked artist is essentially just a set of liked songs (a subset of songs by that artist).

Security Concerns

Key Security Requirements:

- User's accounts and passwords should be secure.
- Users should not be able to use the website without logging in.
- Users should not be able to perform actions they are not allowed to perform (such as double-booing a song).

Preventive Measures:

- We will hash the passwords with a salt on our server to prevent against rainbow chain attacks.
- We will check on both the UI and the server endpoints for what actions the user is allowed to perform, and what actions the user is not allowed to perform. For example, if a user already booed a song, the button to boo that song should not appear for that user, and a request sent to the server should fail.

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.

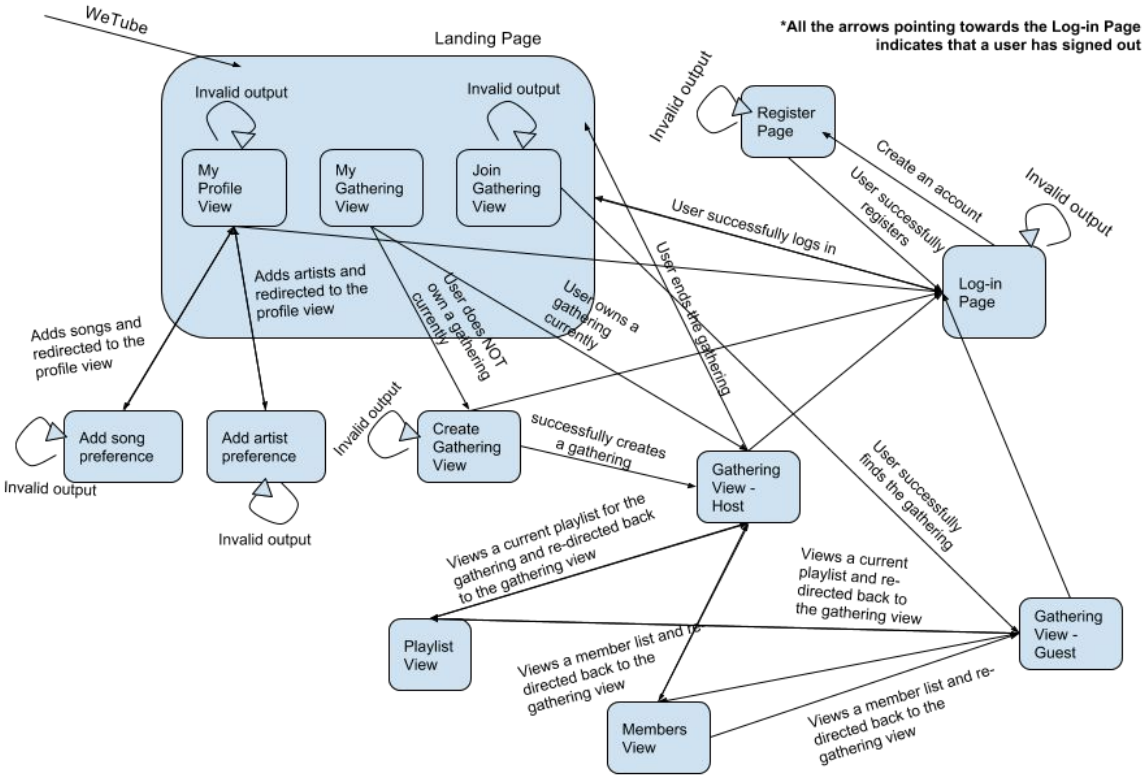
Threat Model:

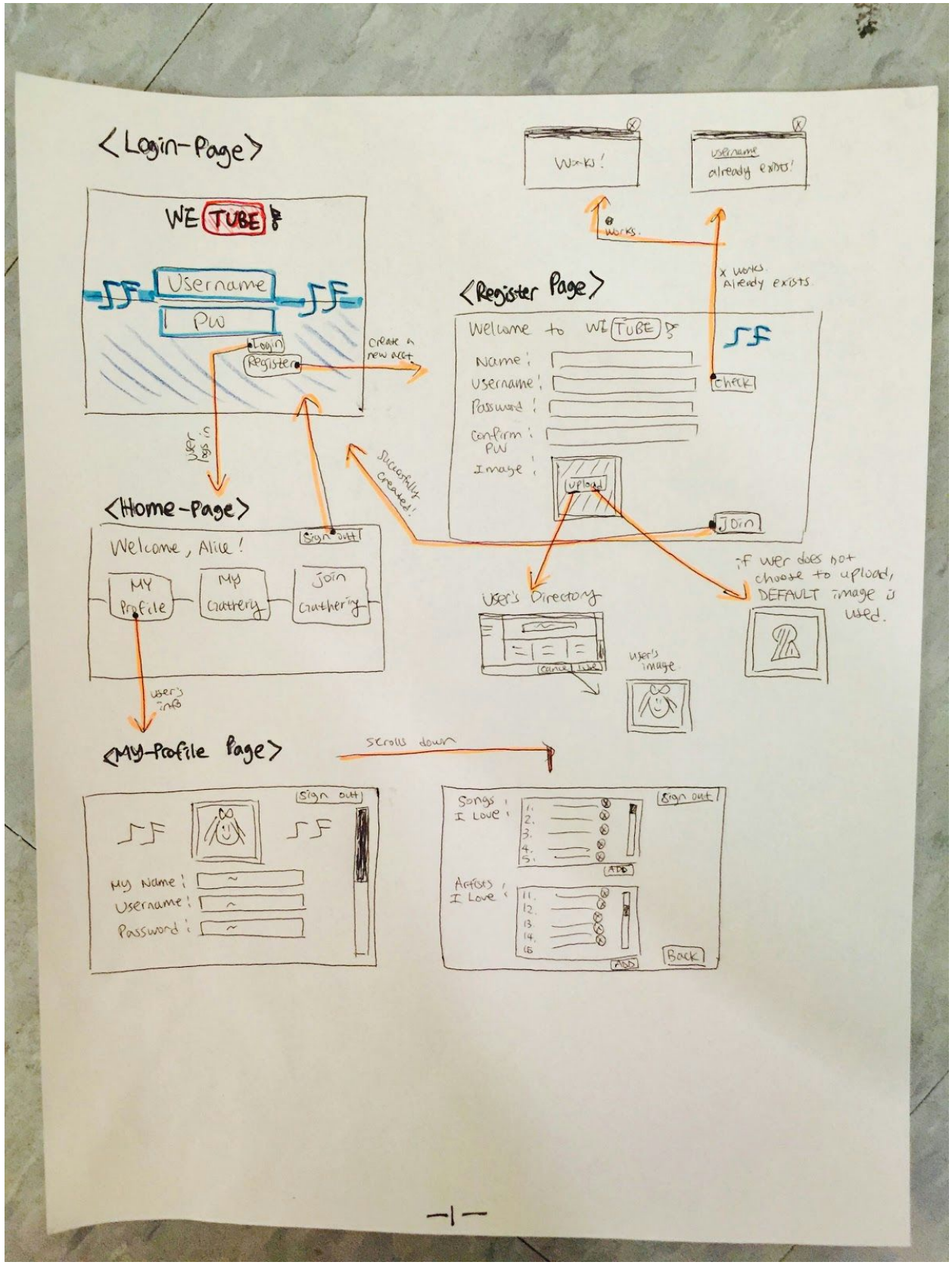
- Because there is no money flowing through the site or confidential personal information stored on the site, advanced hackers are unlikely to want to attack our site. Our attackers therefore are probably just everyday programmers who want to figure out someone else's music preferences.

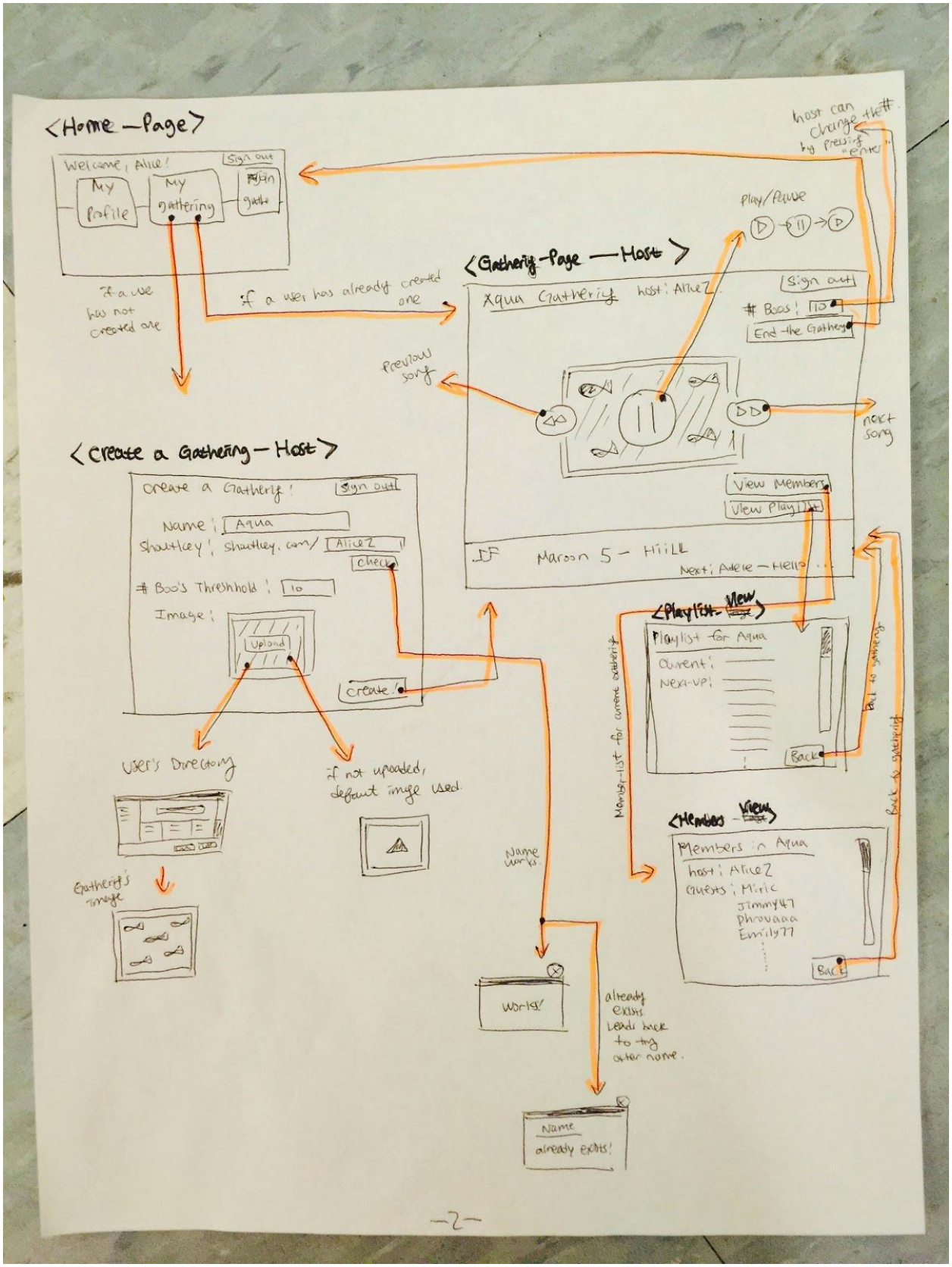
User Interface

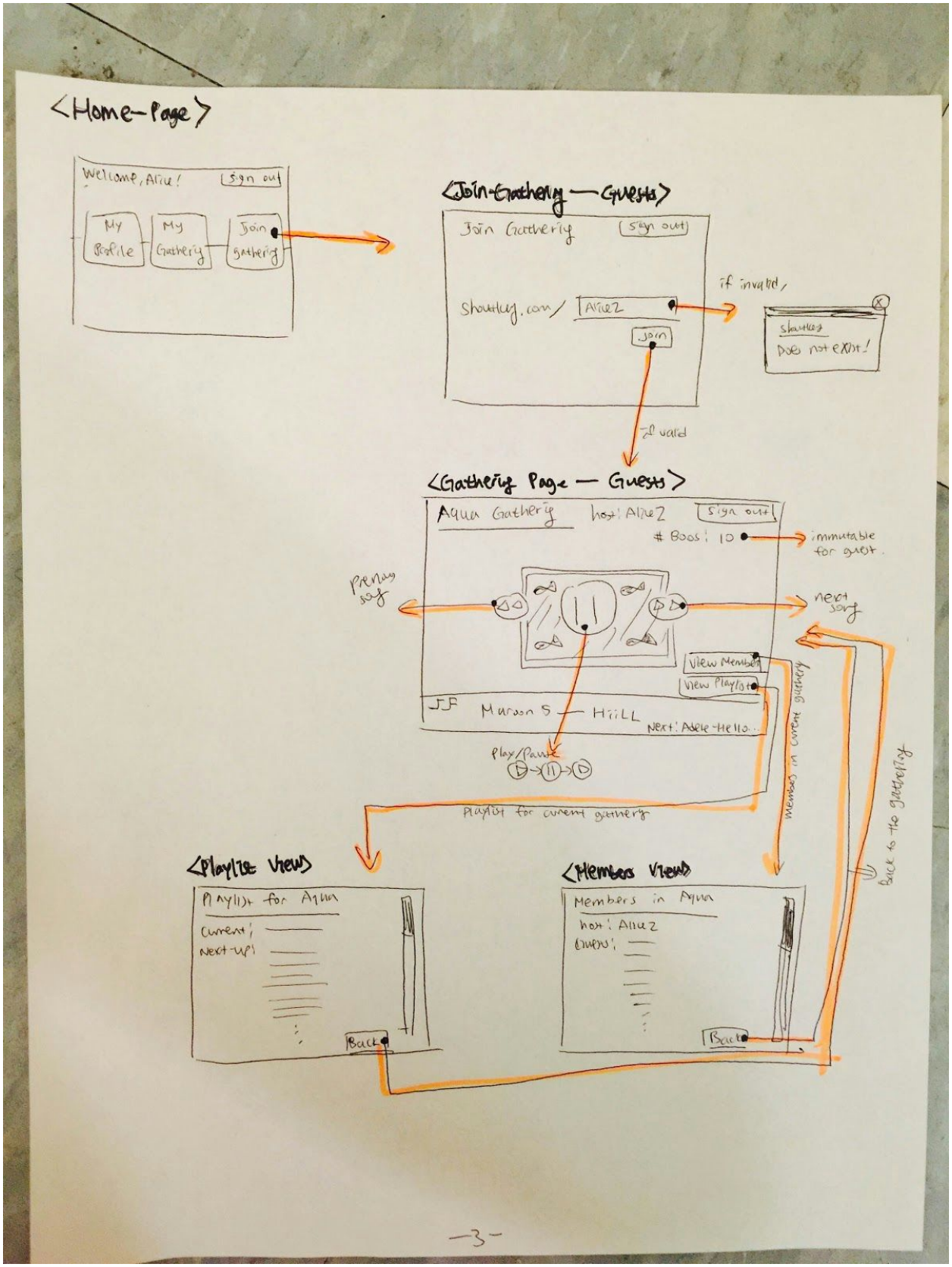
(drawings of state diagram/ UI below)

1. sign in/ register page
2. My profile, my gathering, joining other's gatherings tab
3. my profile to set name, username, password, image, preferred songs and artists
4. My gathering to see current gatherings I'm in or to create a new one
5. Join gathering and see enjoy the current playlist, and view its playlist and member list
6. Find gathering by searching the url+shoutkey


WeTube
Landing Page
*All the arrows pointing towards the Log-in Page indicates that a user has signed out
Invalid output
Invalid output
My Profile View
My Gathering View
Join Gathering View
Invalid output
Register Page
Create an account
User successfully registers
Invalid output
Log-in Page
User successfully logs in
Adds artists and redirected to the profile view
User does NOT own a gathering currently
User owns a gathering currently
User ends the gathering
Adds songs and redirected to the profile view
Add song preference
Invalid output
Add artist preference
Invalid output
Invalid output
Create Gathering View
successfully creates a gathering
Gathering View - Host
User successfully finds the gathering
Views a current playlist for the gathering and re-directed back to the gathering view
Views a current playlist and re-directed back to the gathering view
Gathering View - Guest
Playlist View
Views a member list and re-directed back to the gathering view
Members View
Views a member list and re-directed back to the gathering view


<Login-Page>
WE TUBE
Username
PW
Login
Register
create a new acct
user's info
<Register Page>
Welcome to WE TUBE
Name:
Username:
Password:
confirm PW:
Image:
upload
Join
check
Works!
Username already exists!
works.
x works already exists
successfully created!
<Home-Page>
Welcome, Alice!
Sign out
MY Profile
My Gathering
Join Gathering
user's info
if user does not choose to upload, DEFAULT image is used.
User's Directory
cancel
user's image
<MY-Profile Page>
scroll down
sign out
My Name:
Username:
Password:
Songs I Love:
Artists I Love:
ADD
Back


<Home-Page>
Welcome, Alice!
Sign out
My Profile
My Gathering
Join Gathering
If a user has not created one
If a user has already created one
host can change the # by pressing "enter".
Play/Pause
<Gathering-Page — Host>
Aqua Gathering host: Alice2
Sign out
Boos: 10
End the Gathering
Previous song
next song
View Members
View Playlist
IF Maroon 5 - HIILL
Next: Adele - Hello ...
<create a Gathering - Host>
Create a Gathering!
Sign out!
Name: Aqua
ShareKey: shareKey.com/ Alice2
check
Boo's Threshhold: 10
Image:
Upload
create
User's Directory
if not uploaded, default image used.
Gathering's image
Name works.
Name already exists!
already exists. Leads back to the enter name.
<Playlist View>
Playlist for Aqua
Current:
Next-up:
Back
Back to Gathering
Member-list for current Gathering
<Members View>
Members in Aqua
host: Alice2
Guests: Miric
Jimmy47
Phouvaaa
Emily77
Back
Back to Gathering


<Home-Page>
Welcome, Alice!
Sign out
My Profile
My Gathering
Join gathering
<Join-Gathering — Guests>
Join Gathering
Sign out
Shoutlog.com/
Alice2
Join
if invalid,
shoutlog does not exist!
if valid
<Gathering Page — Guests>
Aqua Gathering
host: Alice2
Sign out
Boos: 10
immutable for guest
previous song
next song
View Members
View Playlist
JJF
Maroon 5 — Hiill
Next: Adele-Hello...
Play/Pause
Playlist for current gathering
Members in current gathering
Back to the gathering
<Playlist View>
Playlist for Aqua
current:
next-up:
Back
<Members View>
Members in Aqua
host: Alice2
Guests:
Back

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. The alternative, a SQL database, is not as scalable and does not perform as well on large volumes of data as its NoSQL counterpart. 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. Our first idea was 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.

Instead, 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.

After deciding on the boo, we realized that boos should not apply to the currently playing song; if they did, every time an unpopular song came on it would play for 30 seconds until enough people veto it to skip it. This seemed just as bad, if not worse, than just listening to the song to completion. We therefore proposed the idea of a song queue, which would display in order the next N songs that WeTube has chosen to play. Users can boo any of the songs in the song queue (but not the playing song), and if a song is boo'd sufficiently many times it is removed from the queue.

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.