CS 30700

**MelodyMunk**

Team 24: Design Document

horizontal line



Matt Twardowski, Brooks Van Buren, Elizabeth Stallter, Tristan Pickett, Shrikar Peri, Avnish Bablani

Table of Contents

horizontal line

[**Purpose**](#_qh3vr47ht3cx) **3**

[Functional](#_z4hgs1u2myaz) 3

[As a party host](#_ydbiam34ttsp) 3

[As an admin](#_a343jj4dpfka) 4

[As a professional DJ (all sub-bullets if time allows)](#_5thdf5ljpejm) 4

[As a general user](#_dw6v1jvlyau2) 5

[As a party guest](#_3on23hxqw3ij) 5

[Non-Functional](#_he0x9uk9lay2) 5

[**Design Outline**](#_ti6jkphldw1s) **7**

[High Level Overview](#_la1ci1p1zp8t) 7

[Components:](#_gbz9ipllako6) 8

[Client](#_y76c3lotzw3g) 8

[Server](#_q8klh3ipfo0y) 8

[Database](#_byoty4jj18q6) 8

[**Design Issues**](#_5gnutgyzjbku) **9**

[Functional](#_grb86kvuawxz) 9

[Non-Functional](#_321l0qj83pst) 10

[**Design Details**](#_igm8ua3l019b) **13**

[MelodyMunk Data Class Level Design](#_d69g2ii6s283) 13

[MelodyMunk Database Schema Mockup](#_b2narbv7ld54) 14

[MelodyMunk Classes and Database Overview](#_9ov7f2yvcw64) 15

[Activity Sequence Diagrams](#_dz6gkj7424t4) 19

[User Authentication (MelodyMunk Credentials)](#_7vi01ir220bv) 19

[User Joining Playlist](#_d5qw887cdzs5) 19

[Playlist Admin Changing Current Song (Spotify)](#_w22d8geqh1m2) 20

[User Playlist Contribution](#_73yk86tvburh) 20

[Design Layout (UI mockups)](#_ezbjnimaut98) 21

[Playlist UI Mockup](#_w7t2wh97mktb) 21

[Room info UI Mockup](#_cxnqbdoh08ai) 22

[Login UI Mockup](#_yr1h6zvfur1g) 23

[Chat UI Mockup](#_x7klxdgsg9x) 24

# Purpose

Catering to the musical tastes of everyone at a social gathering can oftentimes be difficult. MelodyMunk, our webapp, strives to provide everyone the opportunity to share what they want to listen to. The primary feature is a live playlist offering dynamic changes that others can contribute to. It will increase the audience’s connection to the music being played as everyone has a say in the process. This also gives the host extra confidence in knowing that the guests are satisfied.

### Functional

###### As a party host

* + I would like to play songs my guests like
  + I would like the option to restrict songs to non-explicit versions
  + I would like the option to restrict songs to certain genres, artists, or Youtube channels (e.g. VEVO channels)
  + I would like the option to show the video feed from the current song if there is one (youtube videos)
  + I would like to be able to log in with a Facebook account, and have preferences accounts)for rooms saved
  + I would like to be able to restrict room access to specific users (requires
  + I would like the option to enable a text chat for the room
  + I would like to be able to restrict chat to only admins (for announcements)
  + I would like to be able to specify a list of actions guests and admins have permission to take
  + I would like a constant flow of music
  + I would like to set a suggested genre of music to be played (classical, pop, kids, dance, etc.)
  + I would like to be able to easily share a link to join a created room (QR code, URL)
  + I would like the option to display a visualizer for the currently playing music
  + I would like for songs to autoplay if the playlist is empty
  + I would like to make users into admins
  + I would like to notify people that I have to started a party via Facebook, Twitter etc
  + I would like to set a mood for the room

###### As an admin

* + I would like to be able to ban specific users from chatting
  + I would like to be able to ban specific guests from modifying the playlist or suggesting songs
  + I would like to remove undesired songs (Think... What’s New Pussycat playing 21 times in a row, Rickrolls, etc.)

###### As a professional DJ (all sub-bullets if time allows)

* + I would like to advertize my skills
  + I would like to share clips of my work
  + I would like to have aggregate reviews available from past performances
  + I would like live reviews to come in to gauge audience reactions

###### As a general user

* + I would like to have niche music available to play (if time allows) (Use Spotify, Soundcloud, Youtube)
  + I would like to have a mobile-friendly website interface
  + I would like to be able to search for songs
  + I would like to be able to vote on songs
  + I would like to be able to write reviews on the party host or DJ
  + I would like to be able to report other party guests to the admins

###### As a party guest

* + I would like to sign a guestbook to leave a note when I’m leaving the party (if time allows)
  + I would like to suggest songs
  + I would like to join a room quickly and easily (QR code or URL)
  + I would like my song to be played in a reasonable amount of time
  + I would like suggestions for songs that I may be interested in adding
  + I would like to be alerted when my song is going to be played next (If time allows)

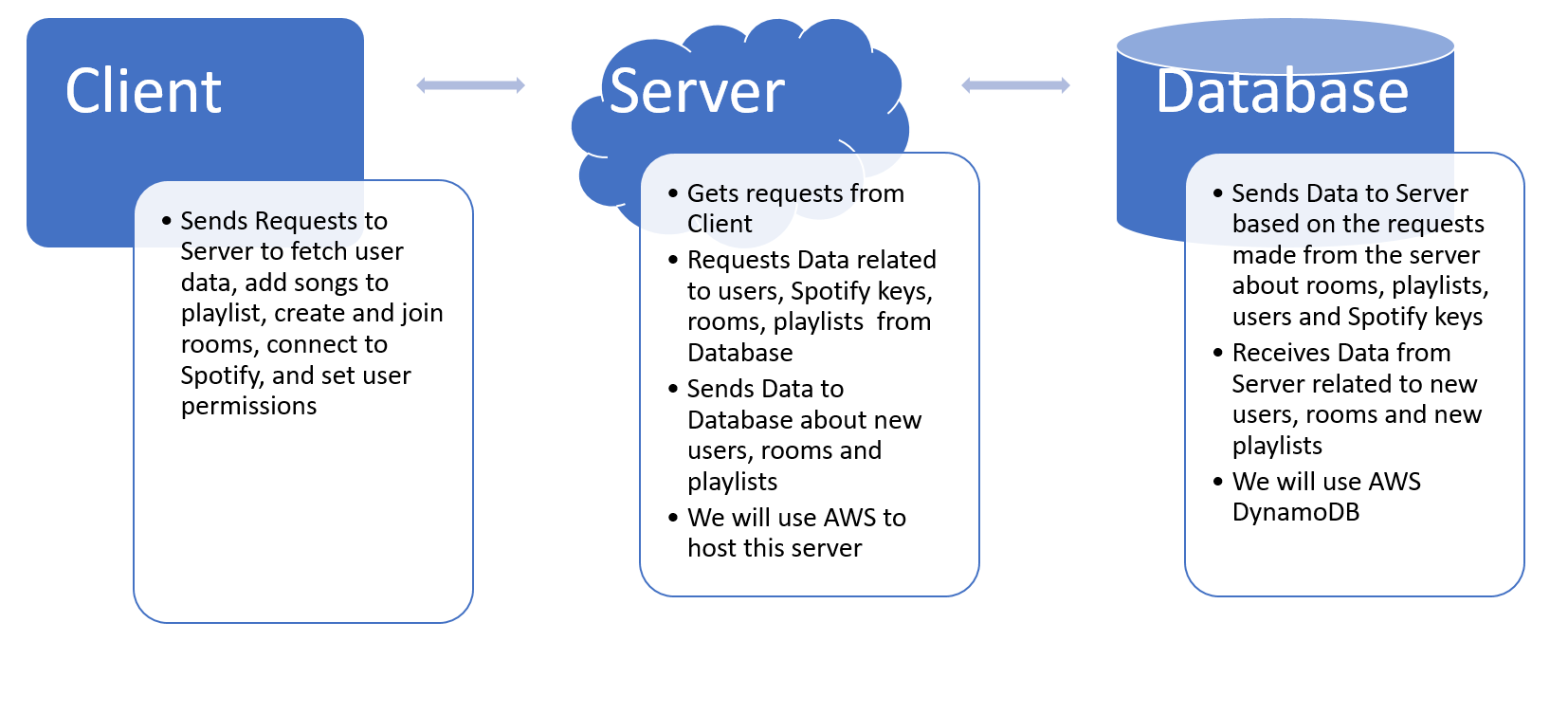
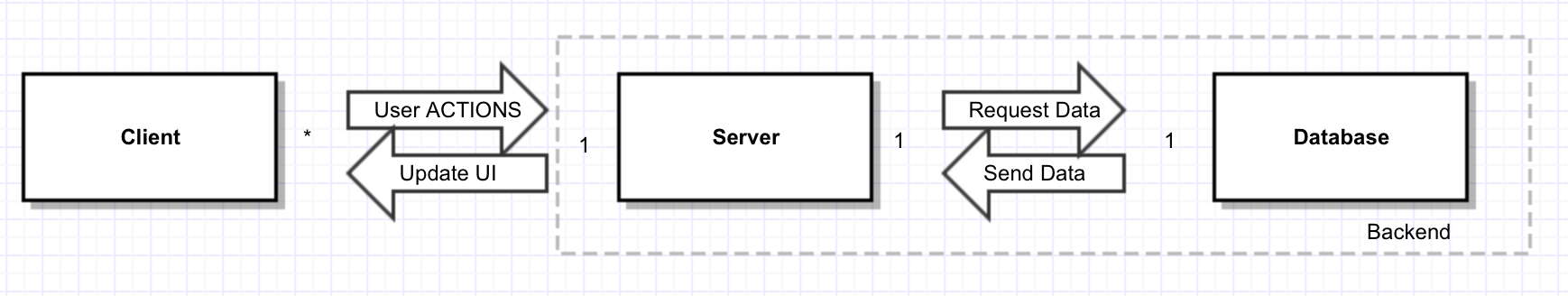
### Non-Functional

* Run smoothly without delay from the website in accordance to limitations set by 4g cell service
* Secure user profiles and user data
* Banning specific users from rooms can utilize IP bans or website profile bans
* Website must be able to handle at least 250 users at any given time
* We would like to have high quality music playing (minimum 128 kbps bitrate)
* A way to test whether or not the service is functioning properly

# Design Outline

### High Level Overview

Our project will implement a client server model for use in combination with an individual's personal device and an AWS server. The approach will allow for multiple users to be connected to the server at any given time. Clients will be able to access an AWS database to find friends using the service and invite them into their personal music rooms on the fly. Requests to add songs will be filtered through the Spotify library and if found then the webapp will update the host’s queue.



### 

### Components:

#### Client

1. The client will be where the user will be able to interact with our service.
2. The client will have the webpage where users can login and join the room they want to join.
3. The client will show the UI where users can join and create rooms.
4. The client will show the UI where users can add songs to the playlist

#### Server

1. The server will manage all of the interactions between the web application and the database.
2. The server will send all user, room and playlist information to the database from the web application and vice versa.
3. The server will handle all of the different users trying to access the web application and send and receive their information from the database.

#### Database

1. The database will store all of the data related to user, room and playlist information.
2. The database will have tables that manage information such as vote totals, access tokens, playlists, users and rooms.

# Design Issues

### Functional

1. Do users need to login to use our service?
2. Connect with an already existing service such as Facebook.
3. No user login is required to use our service.
4. Create our own way to login to service with its own userid and password.

Decision: We will allow users to logon to our service using their Facebook and Twitter account, and we will also provide the option to create their own MelodyMunk account. Guests will also be allowed to use our service but some features may not be made available to them.

1. How we will manage advertisements that will play while playing music?
2. Require users to have a Spotify Premium account.
3. Let the ads play if the user doesn’t have a premium account.
4. Use an ad blocker to not let the advertisements play.

Decision: We will recommend that users have a Spotify Premium account, but if they don't we will play the ads between songs that Spotify sends us.

1. How we will we let hosts invite people to their room.
2. Give them a shareable link that will give people access to their room.
3. Make the person who wants to join the room provide a room name and password to join.
4. Provide a qr code that will send a user to a specific room.
5. Give the host a list of people who could join the room and automatically invite the users the host selects.

Decision: We will give hosts the option to use a shareable link, qr code and also a password protected room service. Hosts will be able to use all three of these options.

### Non-Functional

1. How will we play music from Youtube, Spotify, Soundcloud?
   1. Embed in the page?
   2. Use a widget?
   3. API? (Do they all have one?)

Decision: We decided on a mixed approach with this. The preferred method for any service is to use an API since if it’s readily available, which allows us much more nuanced control over the whole system. If there is no fully fleshed out API for a service though we will then default to simply embedding the music into the page.

1. When banning users from chatting or suggesting songs, how will we enforce it on anonymous users?
   1. IP ban
   2. Also check against device being used

Decision: If we use IP bans, it may interfere with other uses that may be using the same internet connection. Because this would likely be a common case when people are at a gathering, we are going to require that if a host wants to ban guests from certain actions, it must also be required that guests sign in to join the room.

Another step we will take to mitigate this issue is to give host’s the ability to define separate permissions to guests that are signed into accounts and guests that are anonymous.

1. What language will we be using for our backend systems?
   1. PHP
   2. Python
   3. JavaScript
   4. .NET

Decision: PHP is a robust language that has been proven as effective in backend web development for over 20 years. This reliability gives us confidence that the platform will remain a stable one to work with years into the future while also meaning that there now exists a number of robust tools to help build our platform. Some of the members also have an extensive background working the PHP, overall making it the ideal language for our backend system.

1. How will the service be hosted?
   1. AWS
   2. Azure
   3. Personal Server

Decision: We will go with AWS since the tools and reliability they provide are greater than what can be found on a personal server. Our team also already has access to AWS service cutting costs from what would have to be paid for Azure. Furthermore, the members of our team have more experience working the AWS than Azure which slashed learning time that can now be spent elsewhere.

# Design Details

### MelodyMunk Data Class Level Design

### 

### 

### 

### MelodyMunk Database Schema Mockup

### 

### 

### MelodyMunk Classes and Database Overview

MelodyMunk allows users to create live playlists that are dynamically changing based on contributions from other users. In order to create a dynamic cloud-based application for users, MelodyMunk must use a database in order to store user and playlist information. A MySQL database will provide MelodyMunk with a simple relational data storage solution. In order to streamline the development process, we will be using Doctrine as an Object Relational Mapper (ORM) to persist changes to the database using objects and generate complex queries with ease. Below is a description of the classes and database tables that can be found on MelodyMunk's database. However, as we move further into the development process, our database schema is bound to change to accommodate for new features being implemented.

1. **User** 
   1. Every account on MelodyMunk will be stored as a row in the User’s table.
   2. In order to keep query execution time at a minimum, this will only store basic user information such as personal information, username, email, password and user roles.
2. **User API Tokens**
   1. MelodyMunk allows users to link their music streaming services directly to their account. In order to make calls to the respective music service API, it is vital to store the user's access and refresh tokens to make API calls throughout the application.
   2. The **type** column will distinguish each row based on the music service, this will be stored as an integer.
   3. The **user\_id** column will map the tokens from each service to the user which the tokens belong to. This will be an ManyToOne ORM relationship and stored as an integer in the database.
3. **Playlist** 
   1. In order to allow users to create playlists that are accessible to other users, MelodyMunk must store the playlist information in the database.
   2. This table will contain basic playlist information and will be mapped to the user who created it.
4. **Playlist Admins**
   1. In order to allow the user who created a playlist the ability to give admin privileges to more than one user, a table in the database is used to map users as admins to a playlist.
   2. This will contain columns that map to a user and a playlist. This offers the ability to create a simple query to determine whether a user is an admin for the playlist.
5. **Playlist Queue** 
   1. In order to allow users to contribute to the playlist, each song in the queue will have a row in this table.
   2. This will contain the song information and data needed to create a dynamic queue such as number of votes.
6. **Playlist Queue Vote**
   1. In order to prevent users from being able to vote on a song more than once, we need to store the votes in the database. This will allow the application to check whether a user has already voted on this song.
   2. This will contain columns that map to a user and a playlist queue object.
7. **Playlist Suggestion** 
   1. In order to allow users the ability to suggest songs to the playlist host, a table in the database will store the song suggestions so that the playlist admins can view the suggestions.
8. **Playlist Message** 
   1. In order to provide a playlist with a text chat between users, a table in the database will store messages from users.
   2. This will contain columns that map to a user and playlist to ensure that messages are only accessible for the playlist it was intended to.
   3. A **date\_created** column will store a timestamp of when the message was created to allow the application to pull the messages in descending order and achieve a chat room.
9. **Playlist User Ban** 
   1. In order to store which users are banned by admins from contributing to a playlist, a table in the database will hold that data.
   2. This will contain columns that map to a user and playlist in which the user is banned from.
   3. A **type** column will store an integer to distinguish between the type of restriction the user has in the playlist. (Chat ban, vote ban, full ban, etc.)
10. **Playlist Song Restriction**
    1. In order to allow admins the ability to remove undesired songs from being added to the queue, a table in the database will hold info about the song that is undesired.
11. **User Host Review**
    1. In order to allow users the ability to leave a review on a party host or DJ, a table in the database will be used to store user reviews

### Activity Sequence Diagrams

#### User Authentication (MelodyMunk Credentials)

#### User Joining Playlist

#### Playlist Admin Changing Current Song (Spotify)

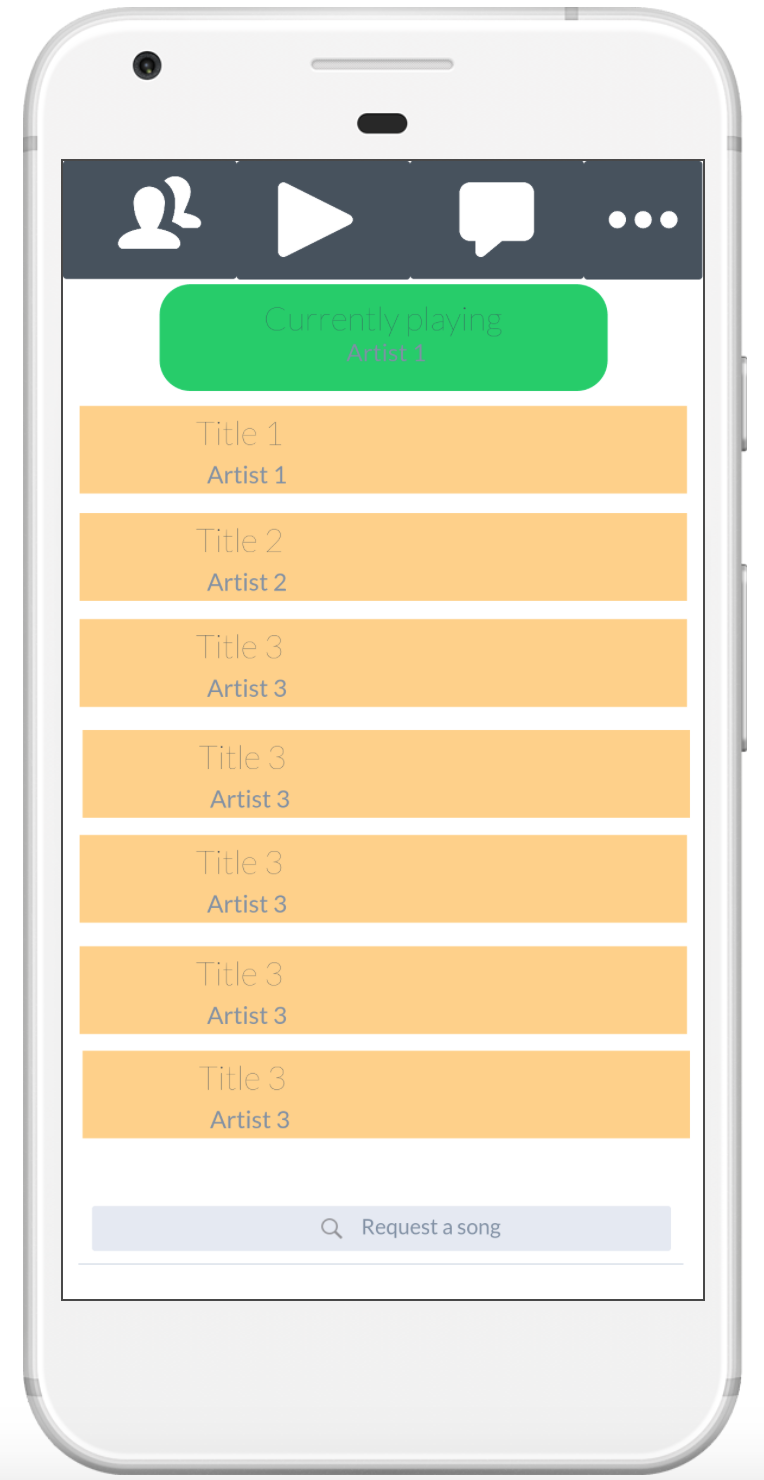
#### 

#### User Playlist Contribution

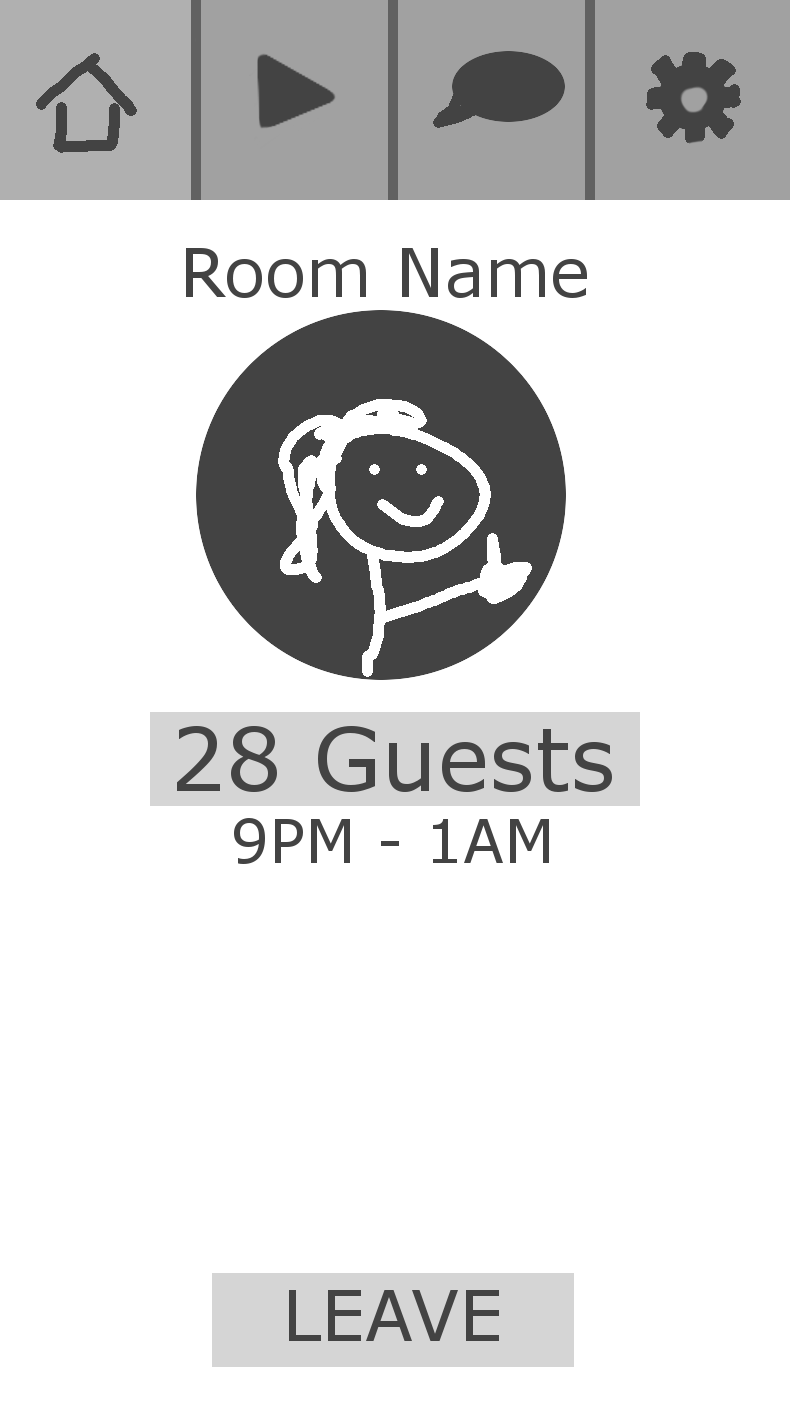
### Design Layout (UI mockups)

Because the webapp is primarily expected to be used on mobile, we are focusing on the design for the mobile webpage.

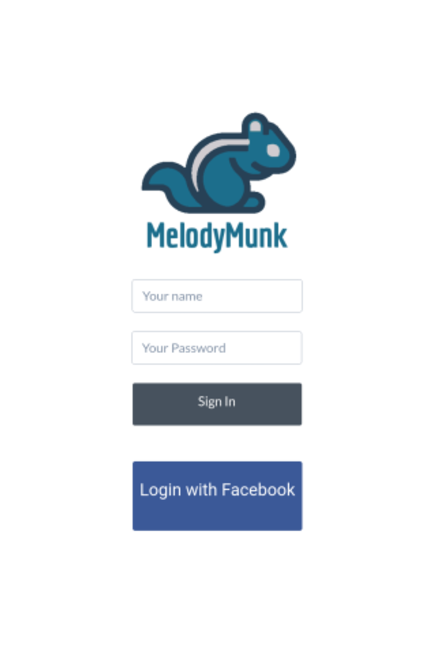
#### Playlist UI Mockup



#### Room info UI Mockup



#### Login UI Mockup



#### Chat UI Mockup

