**Team14 Project Specification**

**Project Proposal we are implementing**

Hotpot Music Share

We want to create a web application that allows multiple users to vote for and listen to songs together in real-time. The web application would allow users to create their own rooms that other users can join.

There would be two types of rooms’: a ‘private’ room and a ‘public venue’ room.

A private room would be for users at different locations, on different devices. This kind of room would be for friends who want to listen to the same songs together, virtually and in real-time. The virtual room will be in charge of selecting the song and playing it across all distributed devices. The room can play songs in a fair, round-robin fashion (popping the top song of each person’s song list). Users in the room can ‘like’ or ‘dislike’ the current song being played and the current song can be skipped if there are too many ‘dislikes’ (to discourage trolling). The room creator can configure various options for the room (voting process, anonymous likes/dislikes, a default playlist, kicking people out, etc.)

A public venue room would be for users at the same location. This kind of room would be for places like restaurants, bars, clubs, cafes; where the music would be played through a single medium, probably the room creator’s device (i.e. the public speakers that everyone can hear at a cafe). Users can join public venue rooms to vote for songs they’d like to be played. Our application would ensure that people are actually at the public venue they are voting for. Just like for private rooms, the room creator can configure various options for the room (songs only from a certain playlist, voting, likes/dislikes, song filters, clean/explicit songs, chatroom, etc.).

**Technologies to Use**

**Language**

Python

JavaScript

**Framework**

Django

**API**

For music playlist:

1. Spotify Web API:

Object models: <https://developer.spotify.com/documentation/web-api/reference/object-model/>

(we may want to look at the [playlist object](https://developer.spotify.com/documentation/web-api/reference/object-model/#playlist-object-full), [paging object](https://developer.spotify.com/documentation/web-api/reference/object-model/#paging-object), [playlist track object](https://developer.spotify.com/documentation/web-api/reference/object-model/#playlist-track-object), [track object](https://developer.spotify.com/documentation/web-api/reference/object-model/#track-object-full), [external URL object](https://developer.spotify.com/documentation/web-api/reference/object-model/#external-url-object), [Spotify URL](https://developer.spotify.com/documentation/web-api/#spotify-uris-and-ids))

[Reading a Playlist](https://developer.spotify.com/documentation/general/guides/working-with-playlists/#reading-a-playlist)

1. Youtube Web API:

IFrame Player API(embed YouTube video in website and control the player using JavaScript): <https://developers.google.com/youtube/iframe_api_reference>

(we may want to look at [how to load a video](https://developers.google.com/youtube/player_parameters#listType) and [how to load the playlist](https://developers.google.com/youtube/player_parameters#listType))

For user location:

1. GeoIP2 in Django <https://docs.djangoproject.com/en/2.1/ref/contrib/gis/geoip2/>
2. Geo-location <https://developers.google.com/web/fundamentals/native-hardware/user-location/>

**All Functionality**

**(Xiaoyu) Creating venues:**

* Create rooms (public or private)

Only logged-in users can create a room, and a room code is generated automatically. This user is the host (super user) of this room.

* Join room

Users (logged-in or guest) can enter a room with room code.

* Localization(figure out where the users are)

If a user wants to join a public room, he/she need to be very close to the host.

If a user wants to join a private room, he/she need to be approved by the host (by sending a message).

* Room configurations

In a public room, songs are ordered first by number of likes, then by the time they are imported. Total number of likes will be subtracted by number of dislikes. Users can like/dislike a song to manage the order of songs. Top 3 liked songs in one hour will be played again in the next one hour.

In a private room, songs will be played from the playlist of each user.

* Kick people out

The host can assign other users as administrator, and both host and administrators can manage the order of songs and kick people out.

**(Han-Yu) Manipulating song list:**

* Upload song list (local upload, upload from Music App like Spotify, upload from Youtube link)

After entering the room, user can choose to import song list from Spotify, Youtube or upload files from their device.

* Play songs simultaneously on different device (for private room)

For private rooms, the server needs to stream songs simultaneously to device connected to the same room.

* Prioritize song order (for private room, single song list)

User can change the order of songs in their own song list.

* Skipping songs

When dislike for a song exceeds a threshold, that song would be skipped.

**(Kai) Interactions:**

* Like/dislike songs: User could vote for the songs in the playlist in public room, then host could order the songs in playlist according to the number of like of each songs. Songs most liked(top3) in an hour will be played again in the next hour.
* Change song order: Only host could choose to change the order of the songs in playlist. Host could drag the song directly to adjust the order.
* Add user to rooms: Host and administrator will get notification and choose whether or not to approve new private room members. For public rooms, each user could enter the room code and the system would get their position and decide whether to add them in directly
* (Real-time chat stream): User could comment on the song and the real-time comment would float and move on the song or video interface

**(Xiaoyu) (Visualization):**

* Include youtube video

If songs are imported from a youtube playlist, youtube video will be streamed while playing the song.

* Default visualization

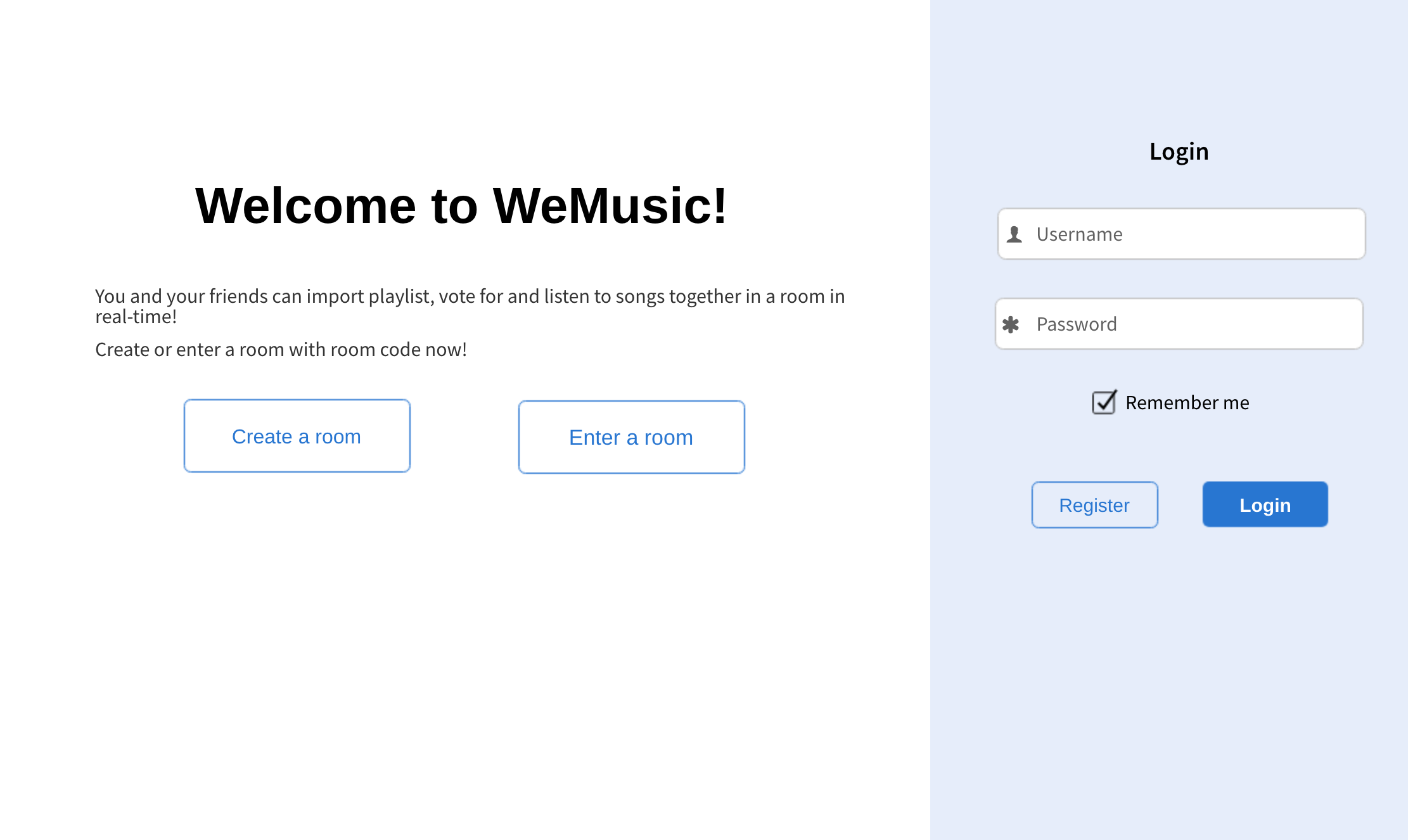
If songs are imported with other methods, default visualization will be played.

**Draft Implementation of Data Models**

|  |
| --- |
| from django.contrib.auth.models import User class User(AbstractUser):  playlists = models.ManyToManyField(Playlist, black = True)  location = models.CharField(max\_length = 500)  class Meta(AbstractUser.Meta):  pass  class Rooms(models.Model):  status = models.IntegerField() *# 0 = public, 1 = private*  number = models.IntegerField()  host = models.CharField(max\_length = 200)  administrator = models.ManyToManyField(User)  members = models.ManyToManyField(User)  playing = models.CharField(max\_length = 500) *# name for current playing song*  dislike = models.IntegerField()   def \_\_str\_\_(self):  return '%s' % (self.status)  class Playlist(models.Model):  name = models.CharField(max\_length = 500)  import\_time = models.DateTime()  room = models.ForeignKey(Room)    def \_\_str\_\_(self):  return '%s' % (self.name)  class Song(models.Model):  name = models.CharField(max\_length = 500)  artist = models.CharField(max\_length = 500)  link = models.URLField(max\_length = 500)  playlist = models.ForeignKey(Playlist)  num\_like = models.IntegerField()   def \_\_str\_\_(self):  return '%s, %s' % (self.artist, self.name)  class Comment(models.Model):  room = models.ForeignKey(Room)  text = models.CharField(max\_length = 500)  color = models.ColorField()  user = models.ForeignKey(User)  time = models.DateTime()   def \_\_str\_\_(self):  return '%s, %s, %s' % (self.user, self.text, self.time) |
|  |

**Wireframe or HTML mock-ups**

Welcome page



Music playing page

