# Document Information

|  |  |
| --- | --- |
| **Project name:** | Spotify Groups |
| **Date:** | Date of writing: 29/1/2021 |
| **Author:** | Group Project Team 11 |
| **Owner** | Prof. Jason Quinlan |
| **Document code:** |  |
| **Version:** | 0.4.1 |

Note: this document may not be valid anymore. Please check the configuration management system for the latest approved version of the document.

# Approval

|  |  |
| --- | --- |
| **Date** | **Name and Signature** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Notes

# Definition

|  |  |
| --- | --- |
| **Background:** | Spotify is an integral part of life for many young students. Whether it is used while doing a computer lab, walking down the street or at a party, it’s allowed music to be everywhere. Spotify Groups combines this universality with the shared nature of post-pandemic things like Netflix Party or Disney+ Watch Together. |
| **Main Goal:** | To create an app allowing people to create a shared chat room that allows for simultaneous Spotify playback (one user presses play and all users hear playlist). If time allows, an instant messenger function will be implemented to allow communication. |
| **Desired Outcomes:** | Knowledge of new programming languages – at the moment that consists of at least React JS and Django Python. Create an app that is simple, fast and easy to use. Get deliverables in on time. |
| **Constraints and Assumptions:** | Time limit of 7/8 weeks, manpower and knowledge of university students, malleability of Spotify Web SDK |
| **Interfaces:** | No similar projects attempted before, although Sebastian has developed some Django storefront projects before. |
| **Project Approach:** | As deliverables are announced our team will assign roles to complete them, looking at the success or failure of meeting previous deliverables successfully. Each member’s prior knowledge will be used, i.e. Sebastian’s experience with Django and Allan’s experience with JS. There will be weekly scrum meetings with Jason Quinlan, but also regular meetings outside of timetabled hours held over Teams. |
| **Project Product Description:** | * As of 0.3: * UI that can create a ‘room’ and have other users on difference machines join or leave rooms. Once inside room, a Spotify prompt can be searched to find a song, which is then played to all users. * Possible groupchat message function is being discussed. |

# Outline Business Case

One of the first apps that exploded during the halcyon days of the first lockdown was Netflix Party. This Chrome extension synched video playback in Netflix/Disney+/HBO Max (requires cheeky VPN). Like Zoom, it had existed beforehand but gained a spike in interest due to the current global situation.

To our knowledge, however, there has not been a widely-adopted similar application for music streaming.

One business benefit would be for private listening parties. A popular pledge tier for fundraising platforms (i.e. Kickstarter, Patreon) is access to private community events with content creators – portfolio reviews for artists, exclusive access to IM forums with content creators. Spotify Groups would allow for a lightweight way to organize private once-off events – for example, a musician could create a private room for people who donate €20 or more, send the link to those people, join the chat and interact with their community.

In terms of business risk, there’s a popular Discord bot that can be commanded to scrape YouTube and play it in browser. Combined with Discord’s forum-like functionality, that can create a similar experience to this product.

There are a couple of key differences: Firstly, rooms are non-permanent, rather than the permanent nature of Discord servers; its transient nature could be useful for listening parties and once-off social occasions that don’t require a permanent groupchat. Secondly, Discord’s Rhythm bot is a rough scraper that can come up with the wrong version of a song if they share the same title. Spotify’s search function is more specific.

Spotify has also released a ‘Spotify Group Sessions’ function in beta. These allow up to six people to join a synchronous group, which shares a queue. Like theirs, our Spotify Groups will be available for anyone with Spotify Premium and a login code. However, there are two differences: Firstly, Spotify’s app allows any participant to control the queue, while ours requires a number of votes to skip (set by host on room creation). Secondly, Spotify’s app requires either a QR code scan or a Spotify link; our requires a simple four/six letter code to enter a room. It is also worth noting that Spotify preceding us does not mean it will be more successful; music label giants Universal Music Group and Sony Music released an online music store named PressPlay two years before iTunes, but it ended up forgotten.

# Key Stakeholders

|  |  |
| --- | --- |
| **Major Stakeholder** | **Notes** |
| **Jason Quinlan** | Product owner, will be surveying progress and testing eventual product. |
| **Programming team** | Will serve as a good example of a group project on GitHub for future employment, as well as an educational experience. |
| **Possible users** | Might be useful if people want to use it – a Python/React program could end up more lightweight than Discord, for example. |

# Project Objectives

|  |  |  |
| --- | --- | --- |
|  | **Target** | **Tolerance** |
| **Scope** | New application in Django/React allowing for shared Spotify listening and possible direct messaging. Documentation describing code and process. | Theorised possible additions to scope include the aforementioned messages option, emoji reacts and simple games, but are dependent on time. |
| **Time** | 7/8 weeks | Not much, as placement will be starting. |
| **Cost** | Free in currency, ~20 hours per week | If project is finished early cost in time will be smaller, if it requires more time to complete cost will be greater. |
| **Quality** | Fully functional with minimal bugs. Sound quality should be the same as playing through Spotify’s Web Player, and entry into rooms should be frictionless with six-letter code. | Having a finished, functioning product is more important than certain factors i.e. high-quality graphic design for UI. |
| **Risks** | Not getting project finished in time, project sprawl with added features, project being more complex than expected. | Sprawl can be added if basic features are completed, and can be rolled back to previous versions if necessary. Complexity is not necessarily bad, as long as project team has time to adjust. |
| **Benefits** | Small and simple application at end, allowing for connected experiences New knowledge of languages, team skills and apps such as Slack | Benefits speak for themselves – not much of a tolerance margin here. |

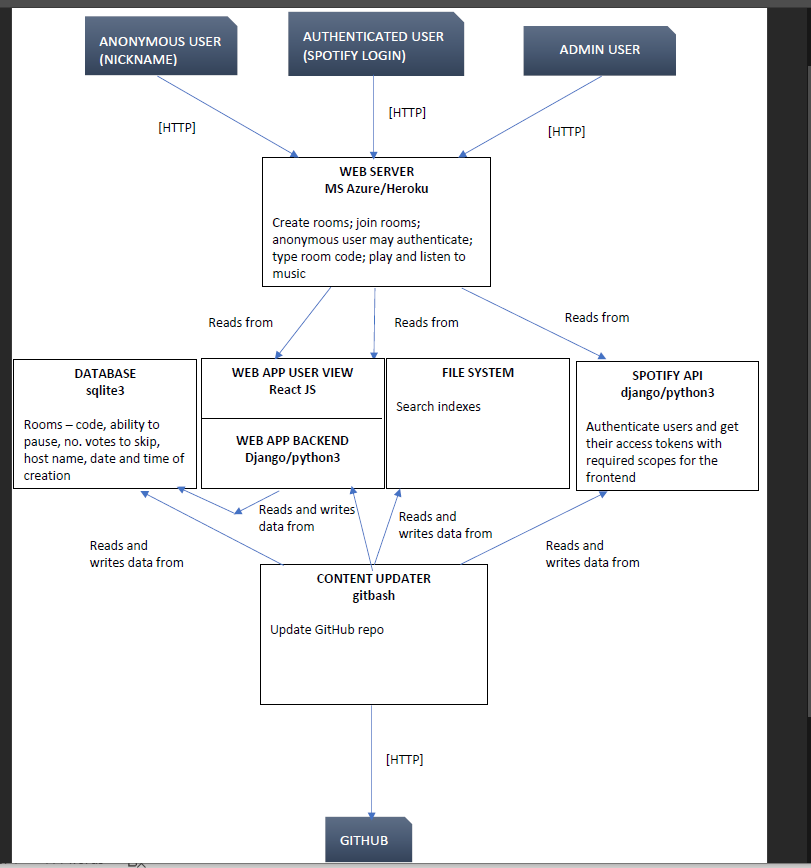
## Project Objectives – MoSCoW Prioritisation

|  |  |
| --- | --- |
| **Must have:** | * Ability for users to create and join rooms with each other using a password * Spotify API functioning, allowing users to log in with their accounts, play/pause it and listen to it with each other synchronously * Users should be able to add new songs to queue * Django views and URLs linked to React JS functionality * Updated README describing installation and running of app |
| **Should have:** | * An instant messenger function for each room, with each user giving a nickname on entry * The ability to choose a set number of votes on room entry, have users vote on skipping a song, then have that song skipped * The ability to choose whether guests can skip songs or not * Ability for host to update room settings * Spotify API-React connection allowing a logged in user’s name to be displayed * Ability for anonymous users to set own nickname on entering room * List of users currently in room |
| **Could have:** | * Web hosting on the UCC CS department servers * Web server coded with Heroku or Microsoft Azure * Emojis/simple games * Ability to exit room without leaving webpage * Ability to change nickname |
| **Won’t have:** | * Integration with sending images * Mobile website * Social media integration, i.e. sharing what song you’re listening to on Facebook/Twitter * Ability to cast audio over to other sources (i.e. from laptop to phone with Spotify app) * Integration with non-Spotify sources of audio on the internet, i.e. Bandcamp/YouTube |

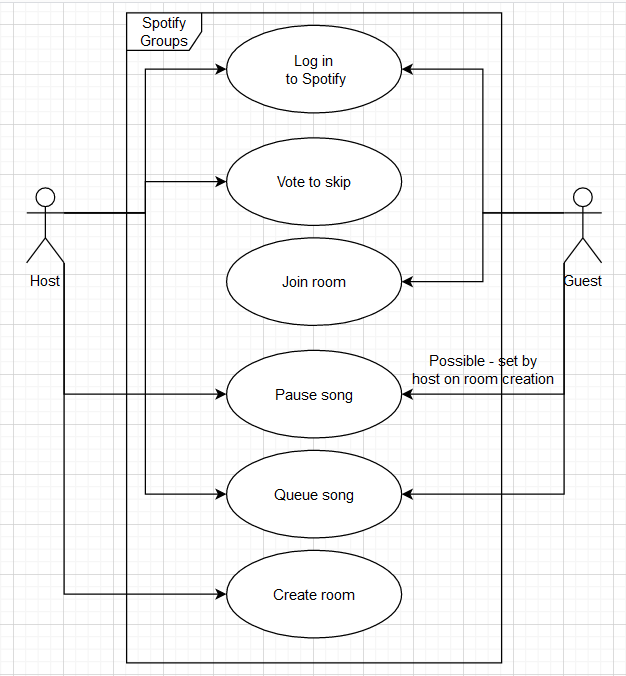
# Project Management Team

|  |  |  |
| --- | --- | --- |
| **Role** | **Reports to** | **Appointee** |
| **Spotify API Integration** | Jason Quinlan | Sebastian Racki |
| **Documentation, bug fixing** | Jason Quinlan | Cathal Donovan O’Neill |
| **Implementing visual representation of frontend code, error handling and redirecting** | Jason Quinlan | Naina Nair |
| **Implementing visual representation of frontend code, error handling and redirecting** | Jason Quinlan | Allan Barry |
| **NO ROLE** | Jason Quinlan | Bernard Grabarczyk |

# Architecture Diagram



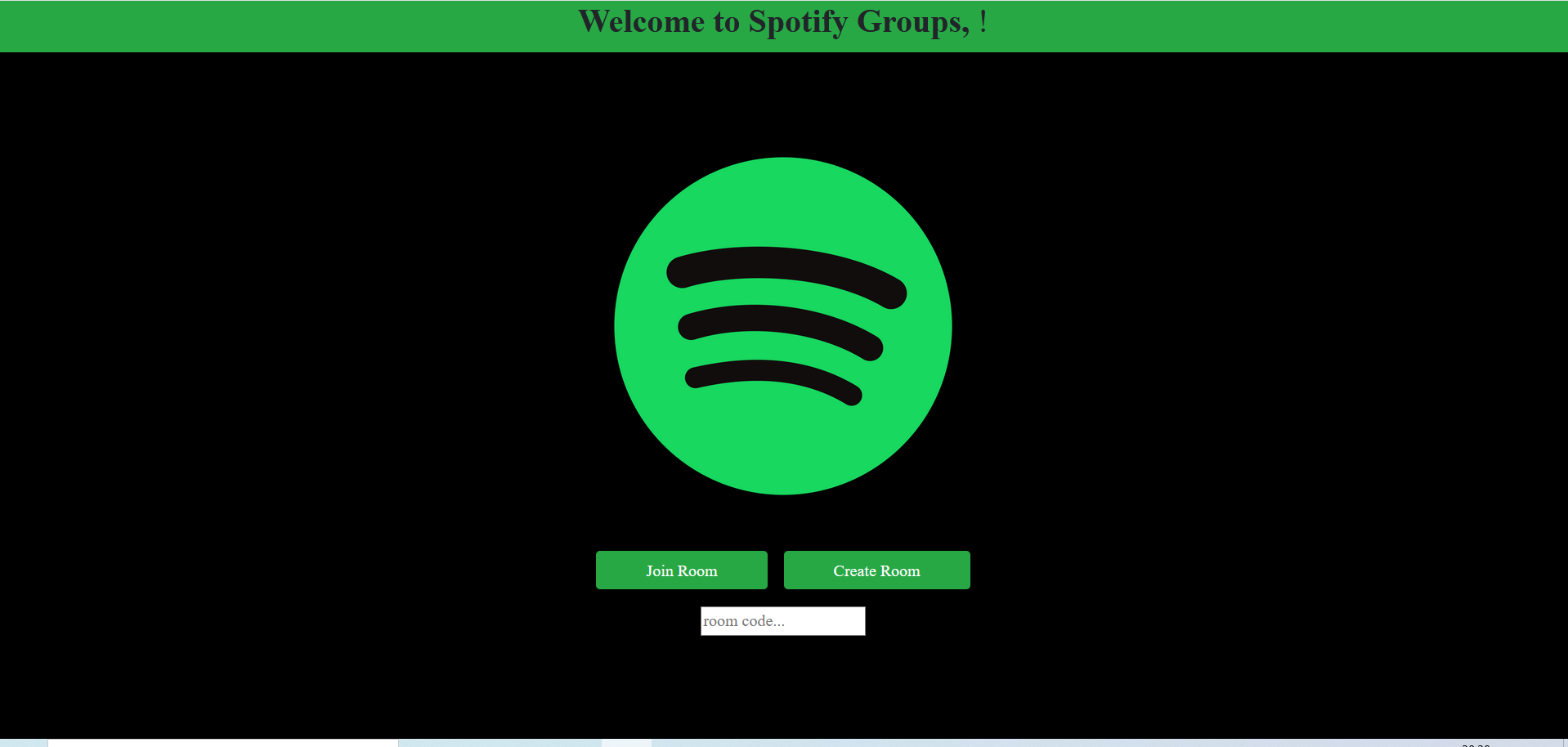
# Use Case Diagram



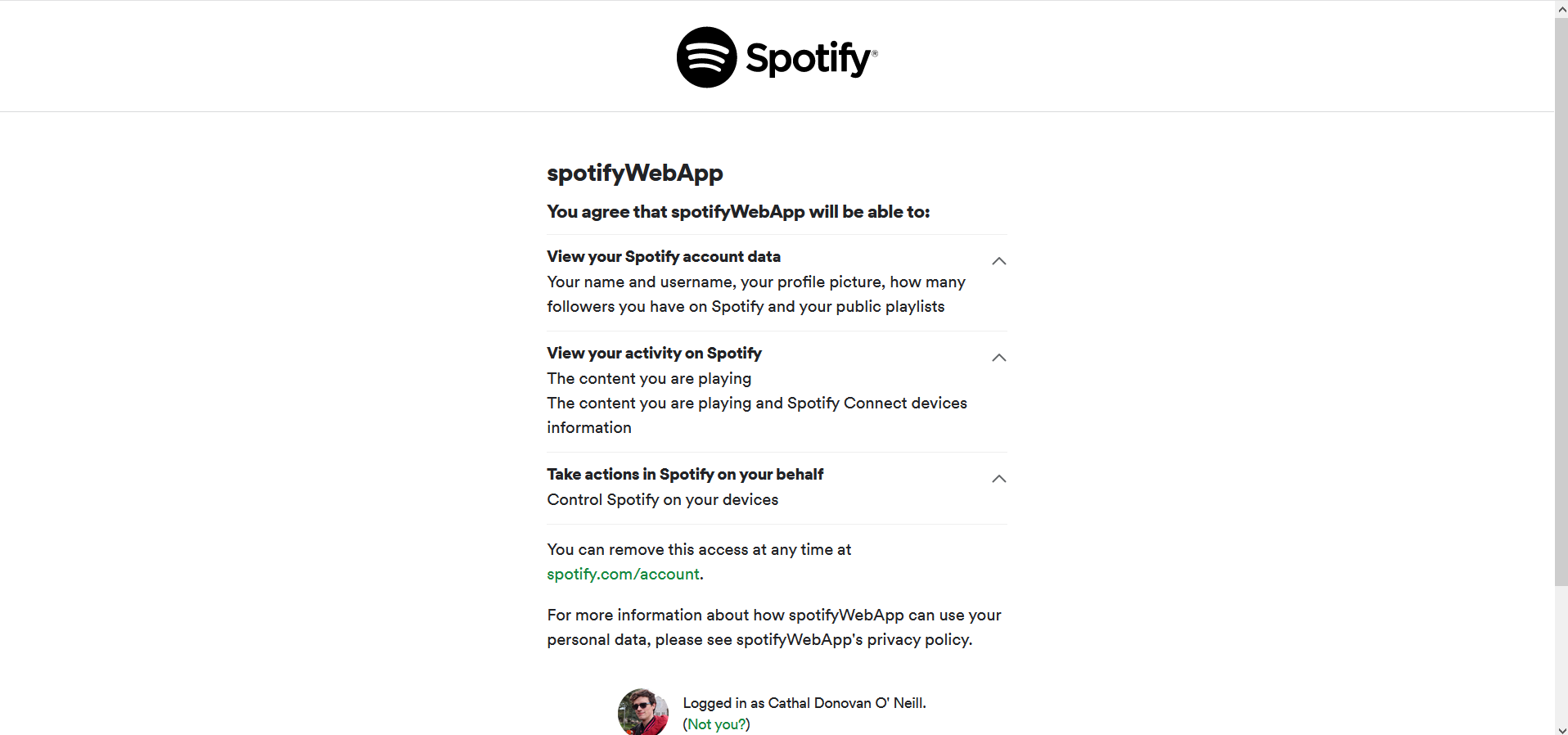
# Project updates

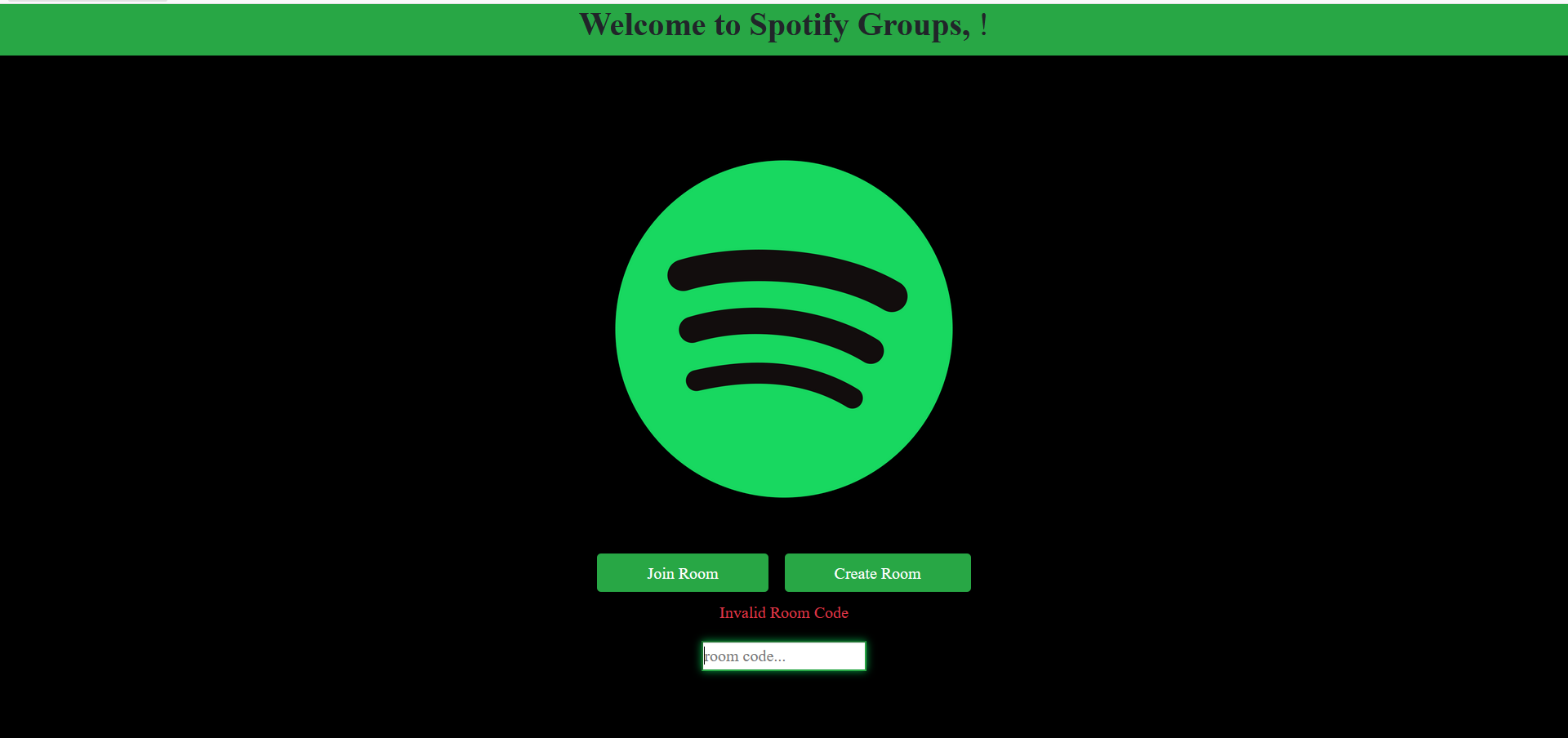
## Week 3:

### Current screens:

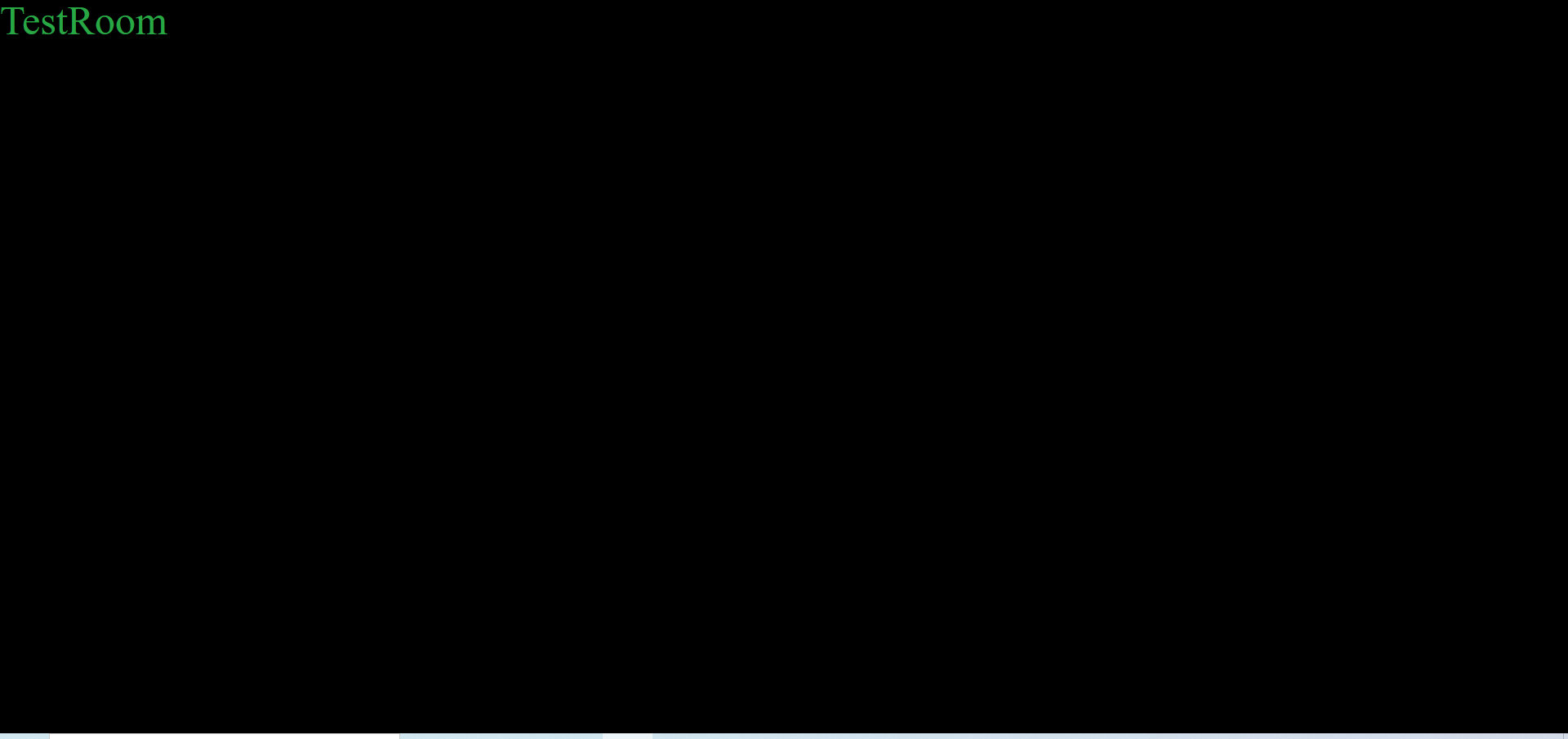
Home page:

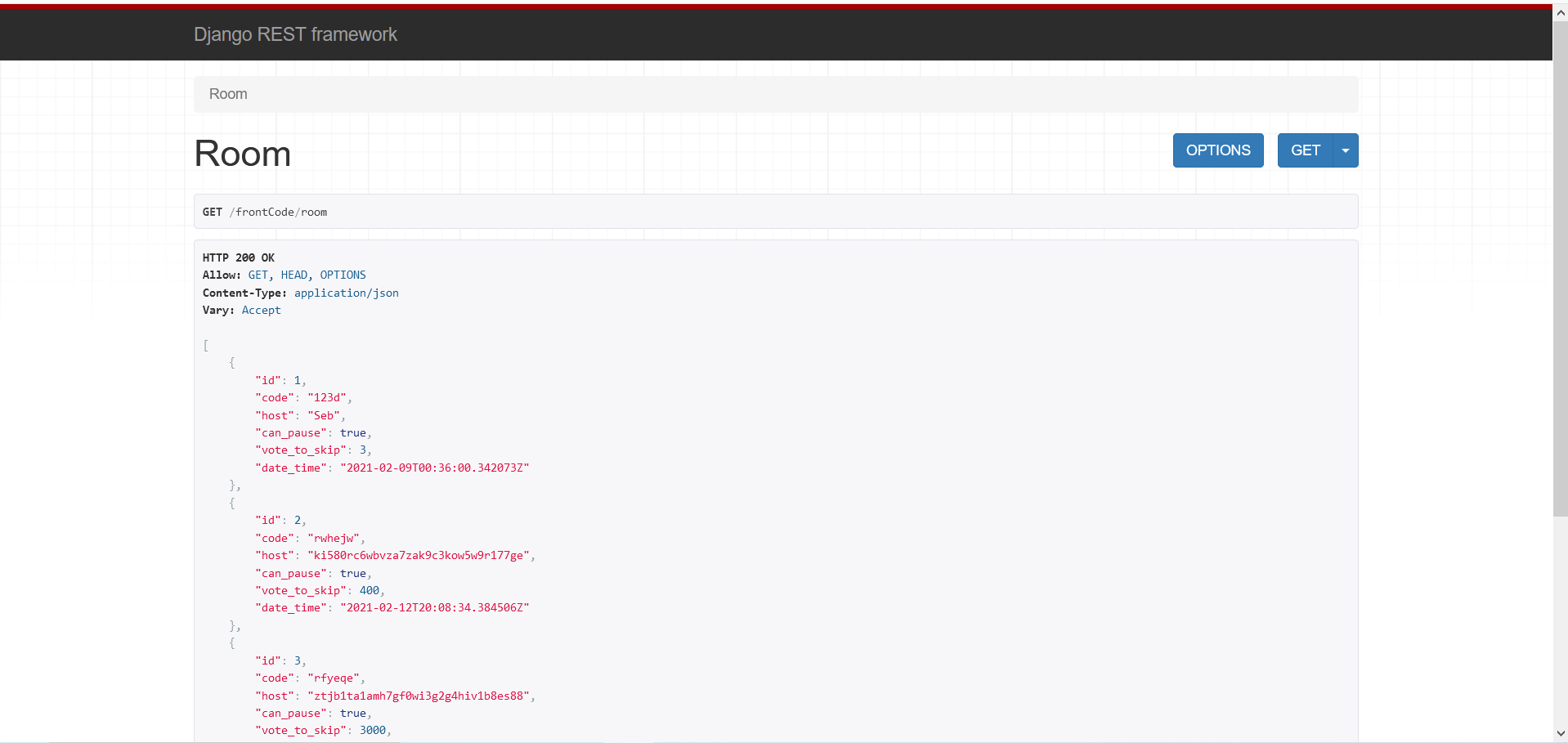
Click create or join room and be prompted for Spotify logins:



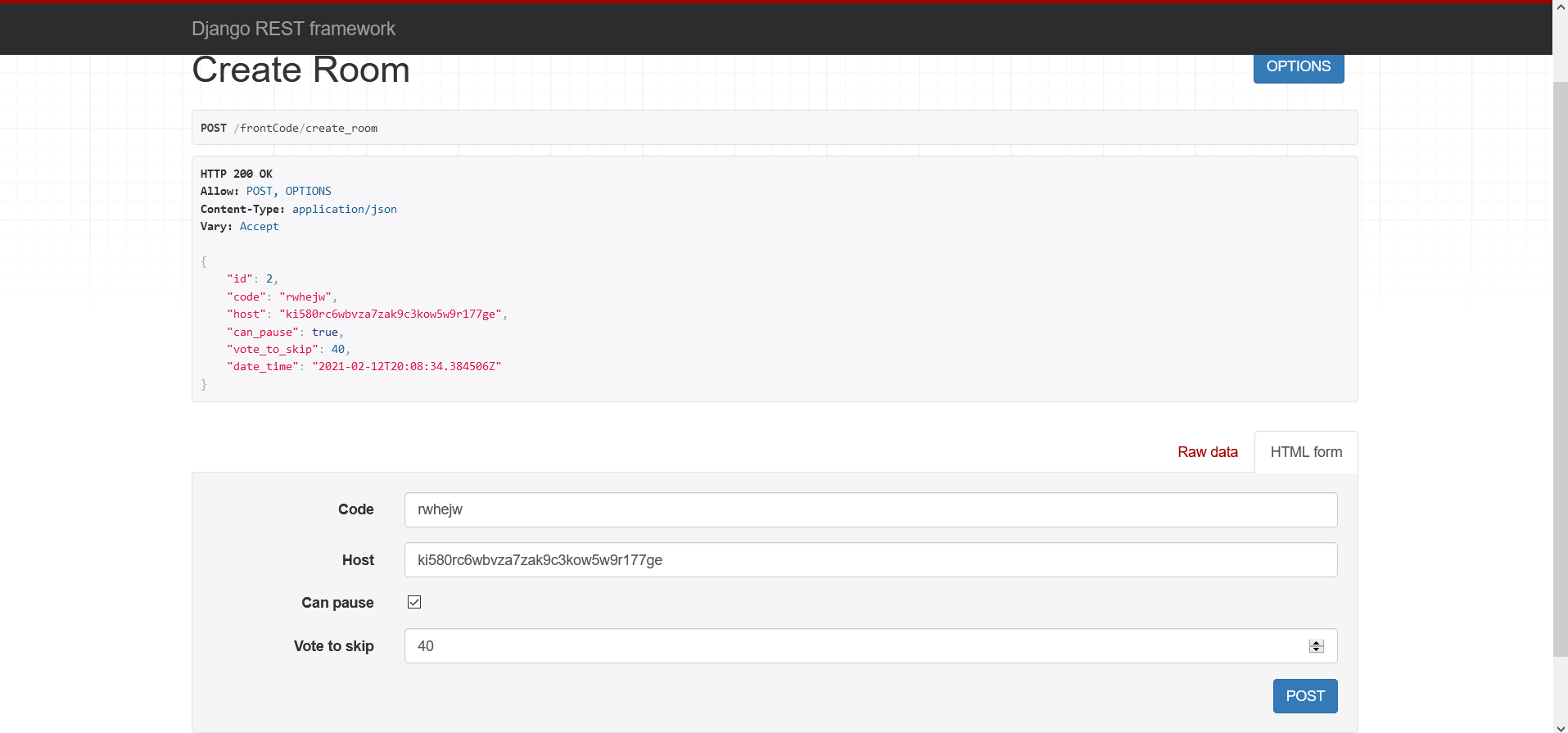
Clicking ‘join room’ requires a valid code: 

Placeholder for room in ReactJS (currently redirected to by Join/Create Room):



Current backend for collection of rooms: 

Current backend for creating a room:



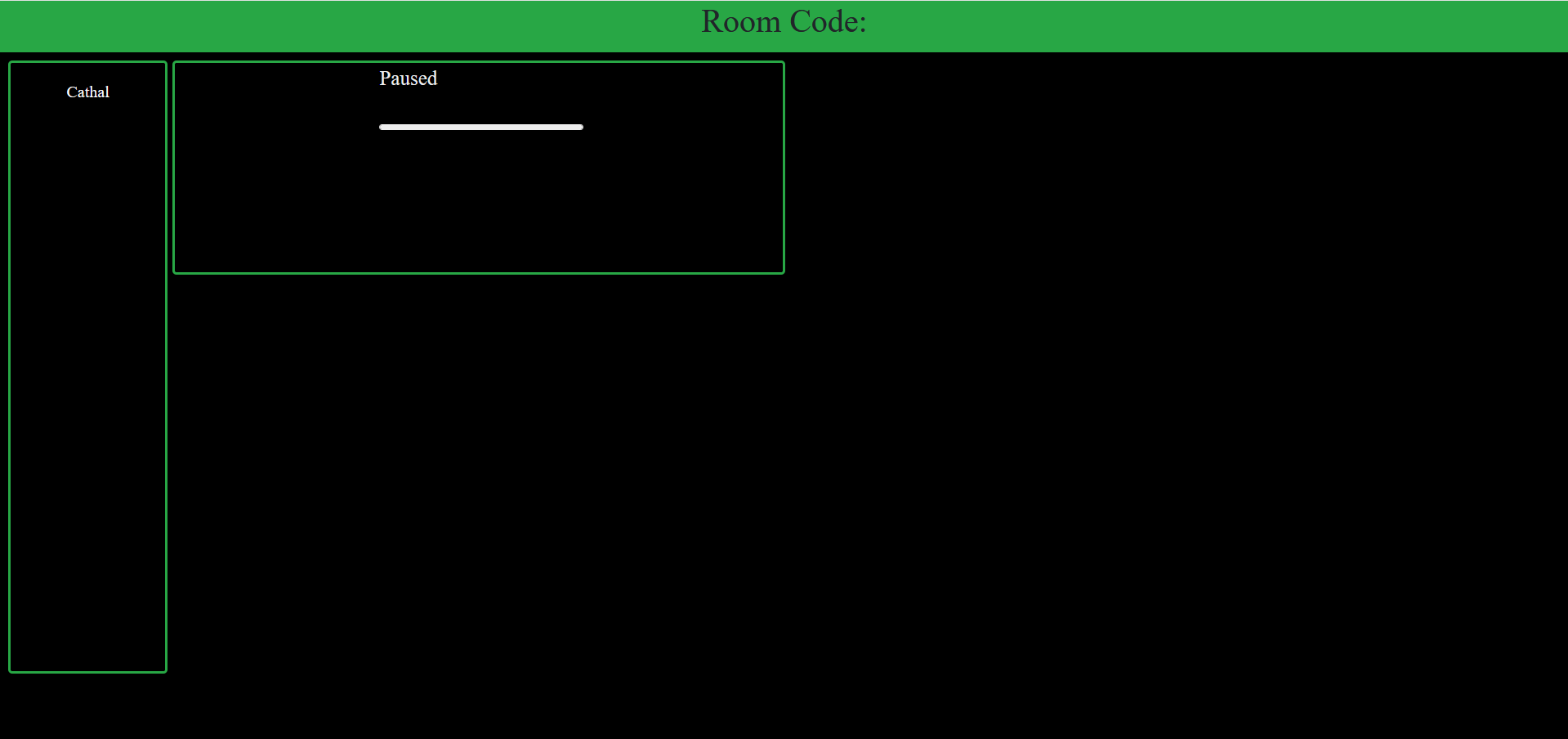
### Challenges and solutions:

* Allan:
  + Challenge: Finding a way to properly load the contents of another page when a button is clicked. Reloading a page is easy with the use of link elements, but had to make sure that buttons could call functions and perform other actions before deciding whether to reload the contents of the page. Links could not do this, so another way had to be found that did not lose out on the strengths of dynamically reloading pages.
    - Solution: Use a React Redirect function that would be called repeatedly when a page is loaded, but unless certain criteria was met would not do anything. Upon the click of a button, the function call in the page would be replaced with a <Redirect /> element changing the contents to that of another JavaScript file.
  + Challenge: Loss of CSS after the frontend was integrated with the backend.
    - Solution: Re-import the Bootstrap CSS file into the index.js file. This necessitated reinstallation of several other modules once complete.
* Naina:
  + Challenge: Unfamiliarity with Django and React
    - Solution: Doing mini-courses on them, becoming familiar enough to begin coding
  + Challenge: Redirecting to and from Spotify authentication/login page upon clicking join/create room
    - Solution: Changing some of the link tags, editing backend code, adding code to frontend, adding the redirect URI to the Spotify project’s page on the Spotify Developer’s website
* Cathal:
  + Challenge: Updating project brief. Received feedback from project owner and peer review requesting more detail on continuous project development along with better layout of project management team, use/sequence diagrams and information on a new competitor, Spotify’s proprietary group sessions.
    - Solution: Kept track of challenges and solutions throughout the week to better describe them in brief. Analysed Spotify Group Sessions to compare to our own, put together use/sequence diagrams and implemented layout/word choice feedback from peer review.
  + Challenge: Unfamiliarity with Django and React
    - Solution: Reading resources in the Slack chat and reviewing earlier weeks allowed me to gain better knowledge of the language, along with analysing the current code. Now bug fixing Django.
  + Challenge: Fixing Create Room function. The Django backend was not displaying in browser/allowing POST requests/HTML-format input.
    - Solution: Added code to the urls.py to allow access to the create room function. Edited the post request function to fix variable-Room attributes, allow POST requests and fixed the JSON-->Django serialiser.
* Sebastian:
  + Challenge: Resolving merge conflicts in git.
    - Solution: Delete branch and reimplement separate segments of code until merge conflicts resolved
  + Challenge: Splitting up the work between the team, making sure tasks distributed equitably.
    - Solution: Assigning tasks to teammates based on people’s strengths, interests and weaknesses.

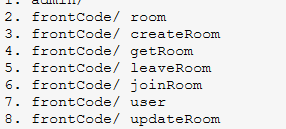
## Week 4:

### Screens:

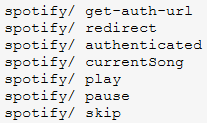
Updated room page with nicknames/unfinished player:



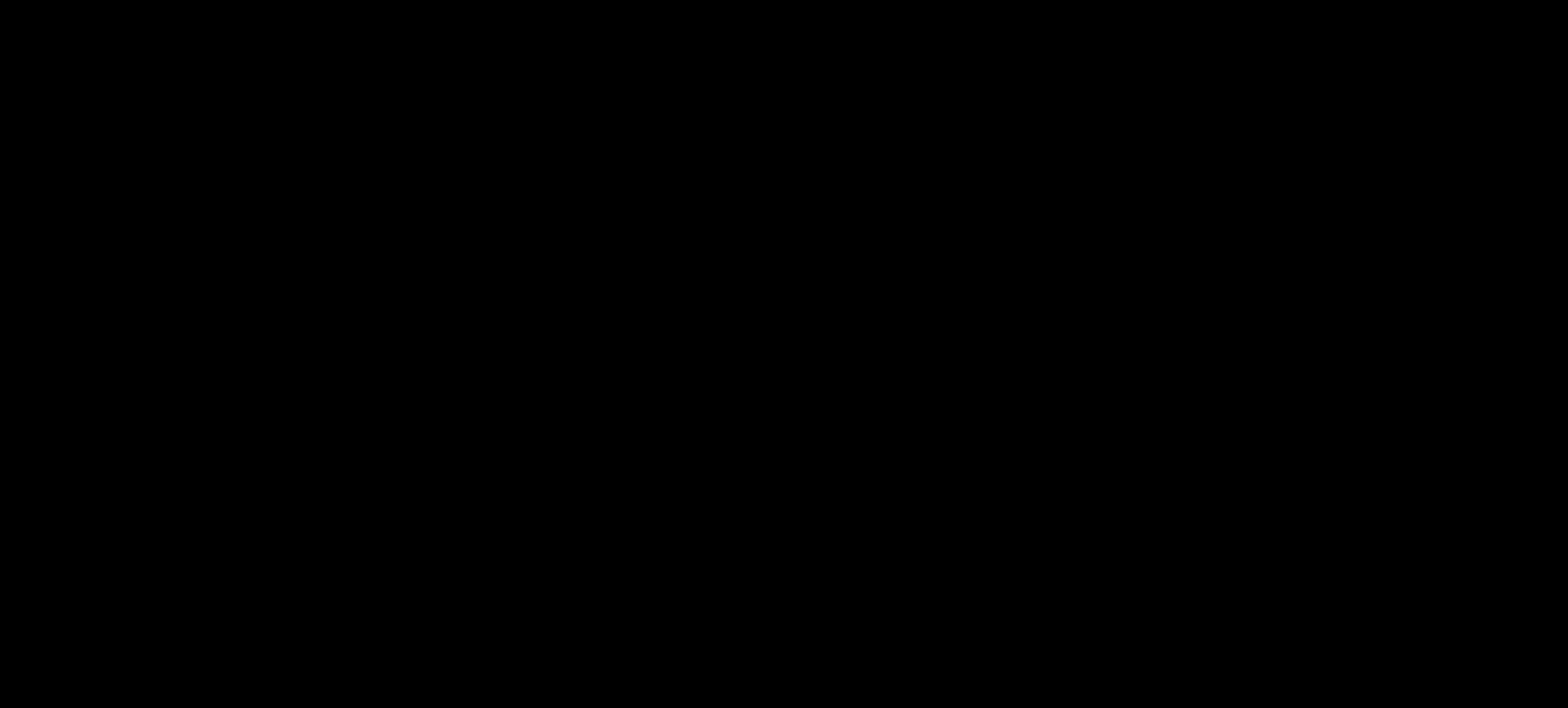
Updated backend pages (No current visual display):



Updated Spotify pages (No current display):



React rendering of join/create lobbies are not yet completed, so black screen:

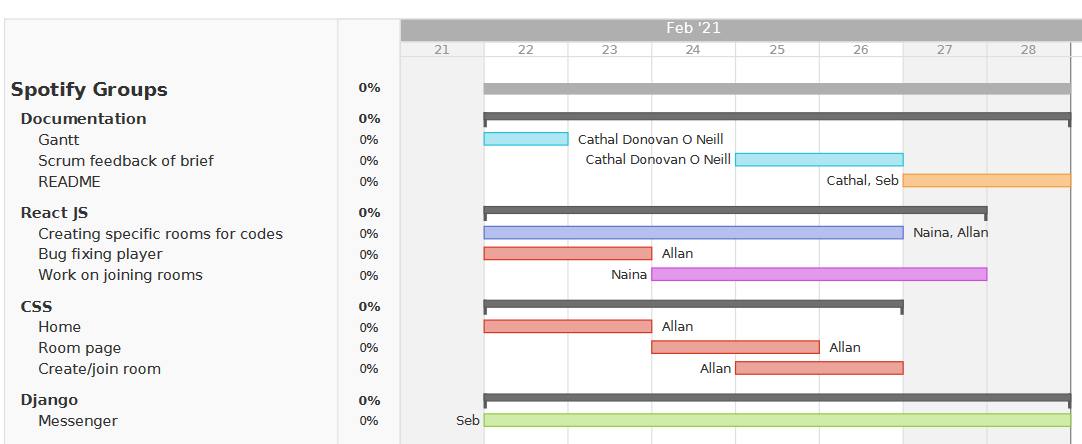


Rest are as week 3.

### Challenges and solutions:

* Cathal
  + Challenge: Update the document with response to feedback given: Fill out the MoSCoW form more, add an architecture diagram combining all team’s individual architecture diagram, update with screens, challenges and solutions, add Gantt chart.
    - Solution: Updated documents, organized meetings to discuss diagrams and combined them into one, organized meeting to discuss Gantt chart for week 5.
  + Challenge: Continued bug fixing in the Django views and URLs.
    - Solution: Fixed bugs, mostly focusing on syntax and consistent variable names. Finished looking into Rooms errors and mostly finished in Spotify API, intending to move to active development next week.
* Sebastian
  + Challenge: Finish planned Django views and URLs for updating room settings and leaving rooms.
    - Solution: Completed; now working on resolving some code conflicts and merging.
* Naina
  + Challenge: Trying to get the website to redirect to the ‘join room’ page when the ‘join’ button is pressed.
  + Challenge: Similarly, get the create button to redirect to the create room page when it is pressed before it takes the user to the actual room.
    - Solution for both: First had to code the actual ‘create room’ and ‘join room’ pages, after which had to resolve some slight inconsistencies with the frontend and backend code. Then, had to change some things in the homepage to make sure the buttons redirected to the right pages: Initially the buttons went directly to the room, so they had to be redirected to their separate pages. The ‘enter code’ text box was moved into the join room lobby rather than the home page.
* Allan
  + Challenge: Getting the Player working was a bigger challenge than expected. Accessing the song data was fairly simple but playback of said music could not be achieved without specific permissions/scopes used to get the access token. Up until now, for development purposes, I had been using hardcoded access tokens obtained from Spotify's website, however one of the required scopes for the Player was not available to be set when requesting a token like this.
    - Ongoing work: Now is working on obtaining the access token for the current session through working with both the backend and some complex React functionality.

### Gantt Chart for week 5



**- - - - PRINCE2® Templates - - - -**

A guide for this template, it’s latest version, and all other templates are available at [mplaza.pm/templates](https://mplaza.pm/templates)

Also, you may be interested in using our PRINCE2 eLearning Courses available at [mplaza.pm](https://mplaza.pm/)

Copyright 2018, Management Plaza

You can use this document for free in your projects and for your personal purposes. Redistributing this document or using it for training requires permission from Management Plaza.

This document is based on AXELOS PRINCE2® material. Reproduced under licence from AXELOS. All rights reserved.