# Document Information

|  |  |
| --- | --- |
| **Project name:** | Spotify Groups |
| **Date:** | 21.3.21 |
| **Author:** | Group Project Team 11 |
| **Owner** | Prof. Jason Quinlan |
| **Document code:** |  |
| **Version:** | 1.0 |

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.  It should allow for a fun new listening experience, whether listening to your favourite music with your housemates or playing games over Zoom. We wanted to combine our skills and learn new technologies to create a product we think people would be interested in. |
| **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.  The web app should be hosted online through Heroku and GitHub, and implemented with React and PythonJS. We want to deliver a quality product at the end of the eight weeks, hitting regular production milestones along the way. |
| **Desired Outcomes:** | * Knowledge of new programming languages – at the moment that consists of at least React JS and Django Python. * Familiarity with including and calling to outside APIs (i.e. Spotify) in a web application using HTTP calls. * Create an app that is simple, fast and easy to use. * Fulfill all of the must-haves in our MoSCoW diagram and as many should-haves and could-haves as possible, recording it as we go. * Get deliverables in on time. * Gain experience with collaborative lines of communication while adapting to the online world, through Slack, regular Teams meetings and direct messages. |
| **Constraints and Assumptions:** | * **Constraints:** * Time limit of 7/8 weeks * Manpower and knowledge of university students * Malleability of Spotify Web API * **Assumptions:** * No team members will suffer major hardware crashes over the course of development, rendering them unable to participate in the project. * Technology required will stay within the limits of React, Django and Spotify’s API, and new languages will not be needed |
| **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. * Regular to-do lists from scrums and bug reports are put up and discussed on the Slack. Discussions also include where users are related to each other to allow for better documentation in the brief. * Programming team works on their separate tasks as defined below, and push/pull/merge their code with their GitHub accounts. This is noted on the Slack, which notifies the whole team when someone has merged their code. |
| **Project Product Description:** | * UI that can create a ‘room’ and have other users on difference machines join or leave rooms. * Once the host creates a room, they start Spotify on their laptop app or the web browser Spotify and start streaming. * All users in room will be able to synchronously hear the same music stream that the host is playing. Users can vote to skip a song. * On creating a room, the host sets the room settings (Whether or not guests can pause the group stream and how many votes the song takes to skip.) |

# 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. Sets schedule for weekly milestones, such as regular project brief updates and inclusions (i.e. MoSCoW, Gantt chart) and provides project management resources such as UCC Entrepreneurship Seminars, educational YouTube videos and academic papers. |
| **Programming team** | Sebastian, Naina, Cathal and Allan, bringing our skill sets together to accomplish the goal of launching Spotify Groups. Will serve as a good example of a group project on GitHub for future employment, as well as an educational experience, with Sebastian and Allan improving their pre-existing knowledge of Django and React respectively, while Naina and Cathal get to come to grips with the new programming language and technologies. |
| **Possible users** | * Online gamers – would allow Spotify to be shared between people in an online gaming session, providing a shared experience over the internet in a time where people can’t sit on the same couch as each other. * Families and friends – There are few art forms as intimate and personal as music, and like shared playlists, listening rooms would allow families and friends divided by oceans to experience the same album at once. * Music fans and music creators – Could be used for album launch parties, as described in business case. Fans could schedule a time to listen to an album as a group, share the link with each other, and go through it together at the same time. |

# Project Objectives

|  |  |  |
| --- | --- | --- |
|  | **Target** | **Tolerance** |
| **Scope** | * New application in Django/React allowing for shared Spotify listening and direct messaging. * Documentation describing code and how to use it, functioning as an instruction manual of sorts. * README describing what the project does and how to install it. * HTML/CSS done in Bootstrap. * File describing the directory system of the project and how to move around it. | * As the project moves along in time, the scope tolerance will narrow as core features become the focus. |
| **Time** | * 7/8 weeks * Weekly check-ins with product owner Jason Quinlan * Weekly group and feedback tasks * Beta release 7/3 | * Not much, as placement will be starting. * While exceptions may be given in extenuating circumstances, milestones in the form of assignment deadlines are fixed and will likely be non-negotiable. |
| **Cost** | * Free in currency * 20 hours per week * Cost of a Spotify Premium subscription | * If project is finished early cost in time will be smaller, if it requires more time to complete this cost will be greater. * Costs in terms of time must be carefully managed. * A Spotify Premium free trial is available for those who have not yet subscribed, or a friend’s account could be borrowed. |

|  |  |  |
| --- | --- | --- |
| **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. * All circumstances described in the use case should exist. | * Having a finished, functioning product is more important than certain factors i.e. high-quality graphic design for UI. * Certain functions are more relevant to the product than others. These are prioritized in the MoSCoW below. |
| **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 * New project to display on our GitHub accounts | * Application can be scaled up or down depending on time and features, as prioritised in MoSCoW * Time spent learning new technologies must be managed as not to spend too much time learning, not enough implementing |

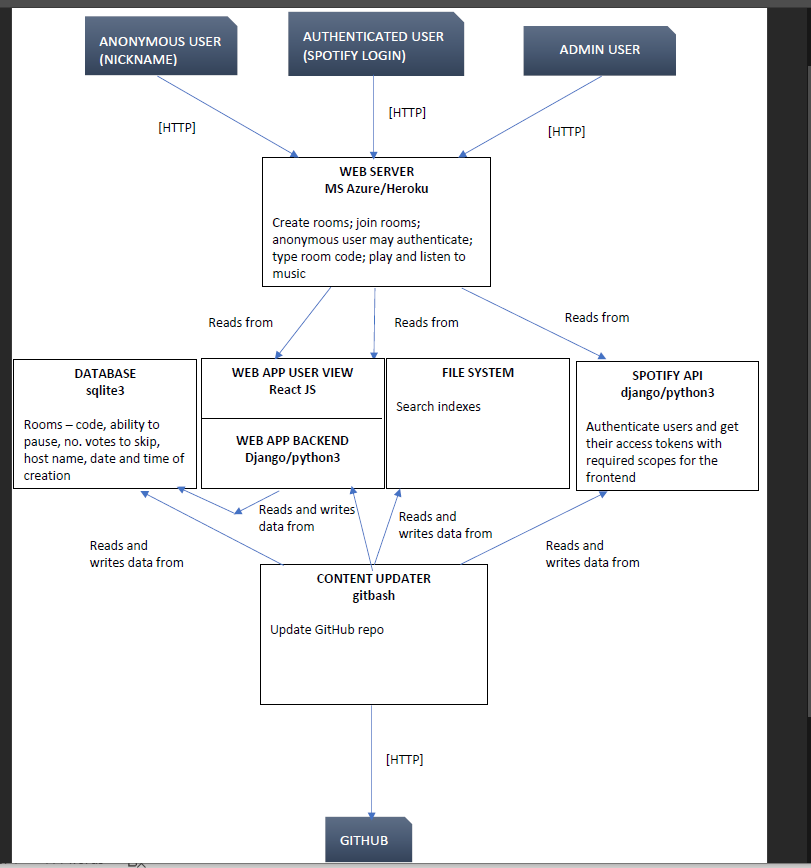
## Project Objectives – MoSCoW Prioritisation

|  |  |
| --- | --- |
| **Must have:** | * Ability for users to create and join rooms with each other using a password * Spotify functioning, allowing users to log in with their accounts, play/pause it and listen to it with each other synchronously * Django views and URLs linked to React JS functionality * Updated README describing installation and running of app * The ability to choose whether guests should be able to skip songs or not * Spotify token readable, allowing a logged-in user’s name to be displayed * The ability to choose a set number of votes on room entry, have users vote on skipping a song, then have that song skipped. |
| **Should have:** | * Ability to cast audio over devices (i.e. from laptop to phone with Spotify app) * List of users currently in room * Ability for anonymous users to set own nickname on entering room |
| **Could have:** | * Emojis/simple games * Ability to change nickname * Ability for host to update room settings |
| **Won’t have:** | * Integration with sending images * Mobile website * Social media integration, i.e. sharing what song you’re listening to on Facebook/Twitter * Integration with non-Spotify sources of audio on the internet, i.e. Bandcamp/YouTube * Web hosting. * An instant messenger function for each room, with each user giving a nickname on entry |

# Project Management Team

|  |  |  |
| --- | --- | --- |
| **Role** | **Reports to** | **Appointee** |
| **Django development** | Jason Quinlan | Sebastian Racki |
| **Documentation, bug fixing** | Jason Quinlan | Cathal Donovan O’Neill |
| **React development** | Jason Quinlan | Naina Nair |
| **Spotify API development** | Jason Quinlan | Allan Barry |
| **NO ROLE** | Jason Quinlan | Bernard Grabarczyk |

# Architecture Diagram



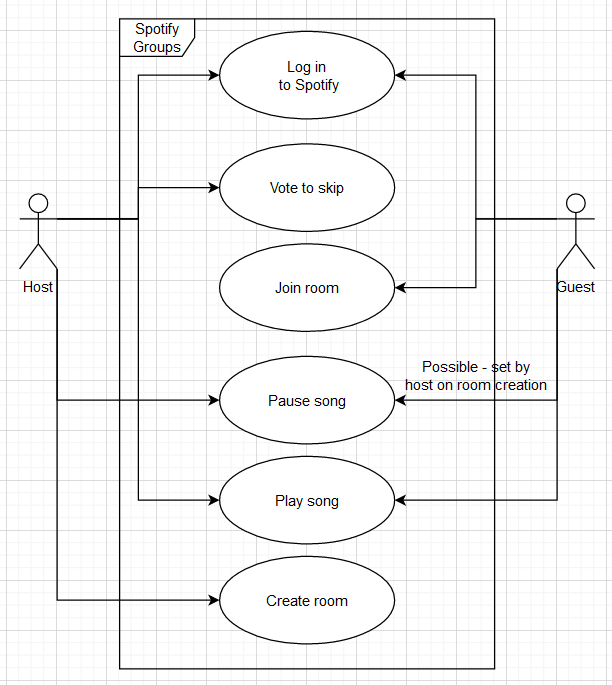
Architecture as of halfway through semester

Previous skills of members of the project team led to us choosing to use ReactJS and Django as our frontend and backend language respectively. Likewise, SQLLite was the familiar SQL variant for the member with pre-existing Django experience, and was thus chosen. We needed Django-database interaction to hold room information, and the templates for our models. Using Spotify required using Python, as it involved dealing with Django’s HTTP abilities and cookie-setting to login to Spotify and gain access to Spotify’s library of songs. The app currently reads from the user’s file system to load the HTML/CSS and React for the site. GitHub was set by the product owner as the content updater of choice.

Overall, the architecture of this project grew naturally from previous experience and the requirements of the project.

The web server implementation was a stretch goal for the project – it has instead been implemented over a local LAN.

# Use Case Diagram



Use case diagram for Spotify Groups as it stands, describing what we want the app to do.

# Project updates

## Week 3:

### Challenges and solutions:

On the React side, the main task this week was properly loading 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. Link tags could not do this, so another way had to be found that didn’t lose out on the strengths of dynamically reloading pages. A React Redirect function was used that would be called repeatedly when a page is loaded, but would not do anything until certain criteria was met. Upon the click of a button, the function call in the page would be replaced with a <Redirect /> element changing the contents to another JavaScript file.

The Spotify authentication/login page was implemented upon clicking the ‘Create’ or ‘Join Room’ pages. This required changing some of the link tags, editing backend code, adding code to the frontend, adding the redirect URI to the Spotify project’s page on the Spotify Developers dashboard.

Some merge conflicts came up on Git, which needed to be resolved; this required deleting a branch and implementing separate segments of code until conflicts were resolved.

Several team members were unfamiliar with the technology when we started. To circumvent this, they took mini-courses, reading resources on the Slack and reviewing earlier code.

Bug fixing: The Create Room backend function was not displaying correctly in the browser, allowing POST requests, or allowing HTML-format input. This was solved by adding code to the urls.py allowing access to the Create Room function. The post request function also had to be edited to fix variable Room attributes, allow POST requests and fixing the JSON🡪Django serialiser.

The CSS was lost after the frontend was integrated with the backend, which required reimporting the Bootstrap CSS file into the index.js file. This necessitated the reinstallation of several other modules once complete.

On the documentation side, feedback was received from project owner and peer review, requesting more detail on continuous project development along with better layout of the project management team, use/sequence diagrams and information on a new competitor, Spotify’s Group Sessions. Better track was kept of challenges and solutions throughout the week, diagrams were implemented, and Spotify Group Sessions was analysed in the business plan section.

## Week 4:

### Challenges and solutions:

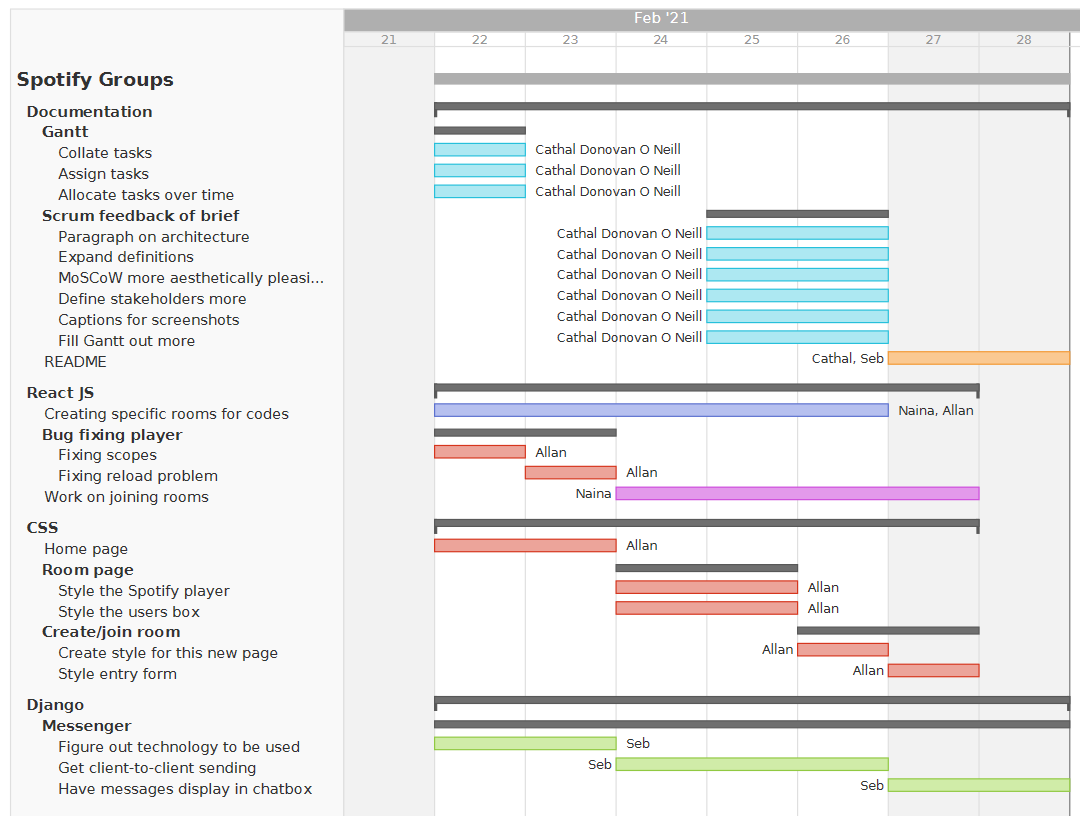
All Django views and URLs planned for updating room settings and leaving rooms now implemented. Work is ongoing on resolving merge conflicts. Current Spotify Views as of week 3:

Getting the player working was a bigger challenge than expected. While accessing the song data was fairly simple, playback of said music could not be achieved without specific permissions/scopes used to get the access token. Up until now, for development purposes, hardcoded access tokens had been obtained from Spotify’s website, but a specific scope could not be requested while using the web player. This is ongoing, and will require working with both the backend and some complex React functionality.

Bug fixing continued. On the Django side, syntax errors and inconsistent variable names were fixed, and other files in the SpotifyAPI folder are being double-checked. The join and create room pages were not redirecting to their respective lobbies when pressed. New lobbies had to be coded, which required resolving some slight inconsistencies with the frontend and backend code. Some homepage links then had to be changed to makes sure the buttons redirected correctly (initially the buttons went directly to the rooms, skipping the lobbies). The ‘enter code’ text box was moved into the ‘Join Room’ lobby rather than the home page.

On the documentation side, the brief was updated in response to product owner’s feedback: The MoSCoW report was changed to include more; each member created an architecture diagram, and one was created aggregating the individual ones; regular challenges and solutions were added; a Gantt chart was added. Meetings were organized to discuss and create diagrams and Gantt Chart.

### Gantt Chart for week 5



Planned schedule for the week ahead

## Week 5:

### Challenges and solutions:

Ongoing work on the Spotify player was completed. Scopes have been fixed with the web player and playback now works.

As another result, homepage now displays user’s Spotify username.

Bug fixing continued, with Cathal on the Django side and Naina on the React side.

Work started on the proposed messenger function. Django channels was chosen to get messages sending between client and backend. Work on the frontend is ongoing – while the messages send successfully between clients on the backend, the chatbox on the frontend is not currently outputting received messages to user.

On the documentation side, Gantt chart tasks were expanded; explanatory captions were added to all screenshots; MoSCoW was modified for aesthetic purposes and to keep track of goals; ‘Stakeholders’ section was expanded; explanation of system architecture was expanded; introductory definitions were greatly expanded. Brief was prioritized due to deliverable deadline.

## Week 6:

### Challenges and solutions:

The main goal of this week was to secure a beta release. The project was converted into a full web app instead of an app working over the LAN, which required deploying on GitHub through Heroku. We ran into cache issues in Heroku, which required redeploying the project several times. This work is ongoing; our local code is correct, but is not showing up on the website.

Besides that, bug fixing was the name of the day. Joining rooms now fully works online, as does creating them, although displaying the user list is having problems. While the caching error created problems for the beta release, stream-sharing over the room now fully works online.

On the documentation side, feedback from last week was implemented. The project objectives and introduction section were expanded, and the weekly check-in was completely restructured for ease of readership. Images are now used to provide context and demonstrate code, rather than having separate sections for screenshots and sections.

A new project document was created for the beta release. While the caching error caused problems with the streaming sections of it, the create/join rooms still are effective and the entire document will be useful for final project submission. A README was also developed giving a quick overview of the project.

## Weeks 7 & 8:

Weeks seven and eight focused on taking our feedback from the beta release and our unimplemented functions, and turning those into a finished product.

Web hosting became a distraction from completing our MoSCoW and use cases, so focus was instead put into making the LAN version of the app the best that it could be. Bug fixing was the main task of these two weeks.

The group came together to work on our final screencast, setting meetings to work out who would cover what parts, filming them and editing them together. Hope you enjoy it!

New documentation was also created: The README had to be extended to detail LAN installation, rather than simply providing a web link. A new product document/instruction file was created to incorporate beta feedback and the current state of the application. A file structure walkthrough was included.