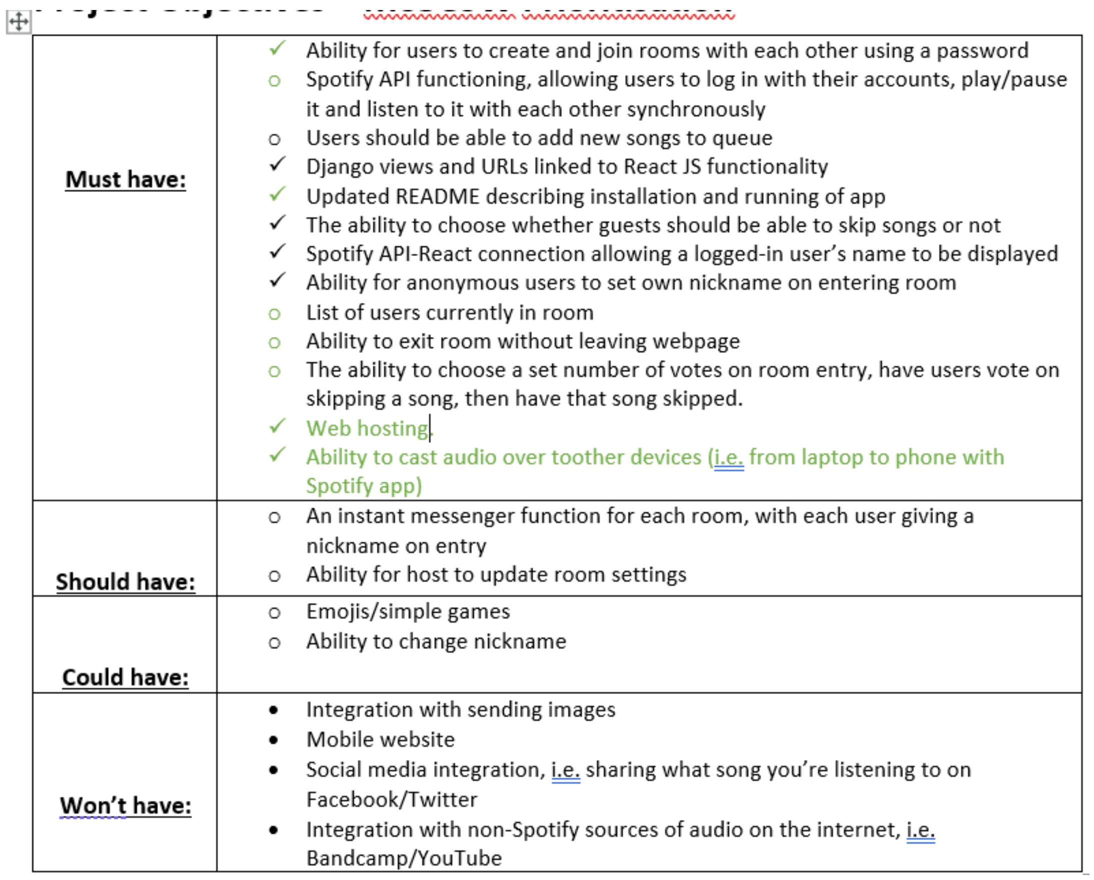
Group number 11: Spotify Groups

Cathal Donovan O'Neill: 118495052, Naina Nair: 118100268, Sebastian Racki: 118459412, Allan Barry: 117343896

Looking back on our goals

Overall, while our goals were ambitious, we feel that most of them were within reach. Our primary goal for this project was to create a fully-functional website that allows a group of people to create a shared chatroom that could be used for synchronous Spotify playback. As it stands, although we still have a good bit to do before we have our final product ready, we are nearing completion of our goals. It took a while for us to get accustomed to the team dynamics and new frameworks, but our primary goal is still definitely attainable. As of now, all of our initial MoSCoW must-haves (as seen in fig. 1) have been implemented or are within reach, with the majority of work left to do being bug fixing.



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 ✓ The ability to choose whether guests should be able to skip songs or not ✓ 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 ○ Ability to exit room without leaving webpage ○ The ability to choose a set number of votes on room entry, have users vote on skipping a song, then have that song skipped. ✓ Web hosting ✓ Ability to cast audio over toother devices (i.e. from laptop to phone with Spotify app)
Should have:	○ An instant messenger function for each room, with each user giving a nickname on entry ○ Ability for host to update room settings
Could have:	○ Emojis/simple games ○ 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 • Integration with non-Spotify sources of audio on the internet, i.e. Bandcamp/YouTube

Figure 1: Our MoSCoW chart. Having a MoSCoW chart helped with prioritising our goals.

The project taught us a lot about prioritisation of goals. Overall, most goals were met in the project. That said, in the beginning, there was some overestimation of the project's difficulty. A lot of learning was needed, both through tutorials and trial and error. For example, we had to learn through experience that Heroku doesn't support Spotify's web SDK due to being too new to the market. Having to pivot away from Heroku to Microsoft Azure for deployment costs us valuable development time, and will likely affect some of our MoSCoW should-haves (i.e. the chat function).

Another example of this overestimation took place in our Spotify player. We originally attempted to create our own Spotify player, using React to fetch the data on the Room's host's currently playing song, but it wasn't until after all that had been accomplished that we found out that Spotify cannot stream music on our website without using a JavaScript library that had only been written and released recently. *This* library came with its own version of a player that was not easily customisable: Creating our own player previously had turned out to be a waste of valuable time and effort. Luckily, the new player turned out to be versatile — we ended up needing it to hide certain buttons from non-users — but at the time, we had no idea that this would be the case.

With all that in mind, however, in eight weeks our team of just four members managed to produce an application that has taken Spotify a year to roll out, learning new frameworks and APIs in the process. There are still some kinks to work out but all of our must-have goals have been met or are within reach, along with many of our major could-haves. We feel that this is a testament to our ambition in goal-setting. It's a two-sided lesson — while you may not complete all your goals, big ambitions can bring big rewards.

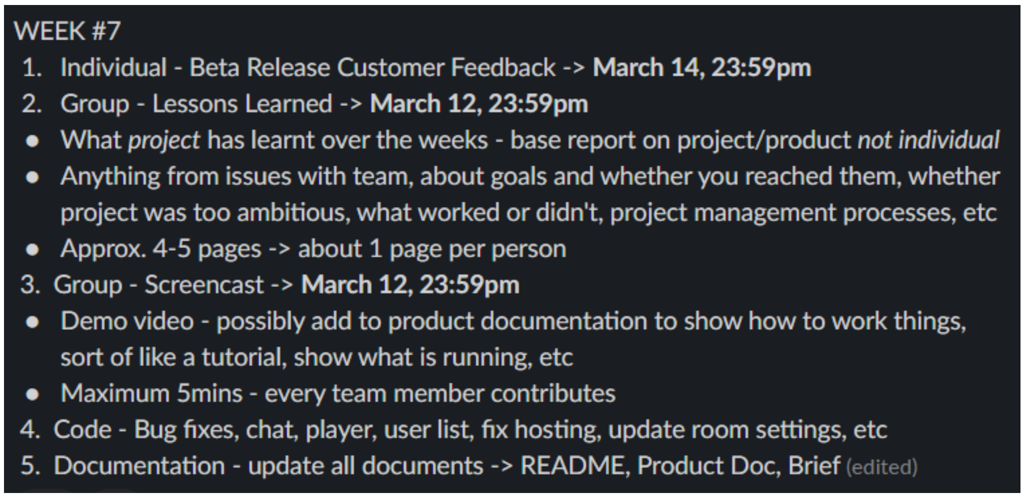
Project management

Communication was a major area from which we learned lessons. A lot of time could have been saved in the first two weeks had we communicated more as a team and asked each other questions about how the code we had written individually worked. This also led to one problem where code ended up being overwritten, due to members of the team not realising that they were working on the same files. However, we did grow closer and began to communicate much more often as the project started to come together. As the beta grew closer, the Slack chat was hopping with queries and collaborations. With the benefit of hindsight, we would have organised more meetings early on where we could go through the various issues we were having individually and use them to help better understand each other's work.

On a similar note, while it was not a big problem, the asynchronous nature of online collaboration did create some difficulties. There were a handful of times where people were not available to work on or help with the project, leading to some sitting around and waiting for replies to questions that might end up missed. It may have been a great help to us if we had perhaps dedicated some time everyday to help each other with the project, whether through regular code-focused scrums or setting some hour of the day where all team members had to have Slack open in the background.

That said, not all of the lessons we learned about communication were negative — we learned that giving and receiving feedback can be significantly beneficial to the work we did as a team. Whether it was feedback from the group or a customer (i.e. our peer reviews), it ultimately helped us to work harder towards our goal of producing a better final product.

Scheduling was also an issue. Initially we prioritised coding over organisation, which led to some amount of rushing into it before planning a proper schedule. This may have led us to get a little flustered in the later weeks once we started getting other assignments and things to work on, and would be a definite thing to change if we had the opportunity to start the project over. Nevertheless, we were able to come together eventually and get our group better organised with regular team meetings and updates, and by making dedicated to-do lists for each week to ensure all the work got done systematically. An example of these can be seen in fig. 2.


WEEK #7

1. Individual - Beta Release Customer Feedback -> **March 14, 23:59pm**
2. Group - Lessons Learned -> **March 12, 23:59pm**
- What *project* has learnt over the weeks - base report on project/product *not individual*
- Anything from issues with team, about goals and whether you reached them, whether project was too ambitious, what worked or didn't, project management processes, etc
- Approx. 4-5 pages -> about 1 page per person
3. Group - Screencast -> **March 12, 23:59pm**
- Demo video - possibly add to product documentation to show how to work things, sort of like a tutorial, show what is running, etc
- Maximum 5mins - every team member contributes
4. Code - Bug fixes, chat, player, user list, fix hosting, update room settings, etc
5. Documentation - update all documents -> README, Product Doc, Brief (edited)

Figure 2: An example of our to-do list for this week.

Technology

We learned a number of technologies through this placement that are used by professionals, from gaining greater experience with Git, to the React webpages and the Django backend. All of us greatly improved our familiarity with these languages over the course of this project, even if we were not specifically working in those areas. These technologies will come in handy for future employment and project work; in particular, our current hosting technology, Microsoft Azure, is needed for one of our members' work placement.

We also learned where our preferences lied with regards to certain technologies. React proved incredibly complex to use, but the project could not have been completed with any other framework. Sebastian learned that he would literally rather die than use Docker, Heroku or anything else to do with hosting. Some of our groups enjoyed the frontend React/Spotify API, while others preferred the Django side of things, and will be working in those areas in future projects.

That said, we had to upskill very rapidly. For example, it was very difficult not only trying to understand the various features of React, but also learning about all the different modules and libraries required to build our website. This also led to some delays: Before we could get started on the frontend, some degree of Django literacy had to be attained by the frontend developers. This proved to be a more time-consuming task than originally thought and led to slowing down the frontend production.

Making the technologies work together also proved difficult at times. One specific aspect of blending the code for the Django and React led to a whole day of work attempting to reorganise the file structure; both Django and React have their own preference of file structures, and require certain files to be located in specially-named folders. To solve this, half the team had to do a lot of research in figuring out how we could use React in such a way that it would allow the allowed file system for Django, including finding a way to tell React to look in different paths for certain files. We learned that it helps in projects not just to have people skilled in specific areas, but to have people with pre-existing knowledge of how those areas work together.

On the bright side, however, this upskilling allowed us to find which ways we learned best when educating ourselves. From short courses on educational websites to YouTube screencasts to the ever-trusty StackOverflow, different resources were used to improve our coding knowledge. Some people preferred visual and aural methods (i.e. YouTube), some written (StackOverflow). Knowing where to find this information, and how best we learn, is an important skill in learning outside of the college experience.

All in all, this project reinforced the advantages of working with frameworks and languages that you have previous skill in. If we had known more about Django, React and the Spotify API before starting this project, it would have helped us to prevent issues like the frontend delays from happening in the first place. Now that we have those technologies, however, they will be valuable for future projects.

Conclusion

Spotify Groups has taught all of us lessons in the process behind projects. At the beginning, members had a more do-it-yourself ethos that led to less communication, but as the project went on we grew closer and trusted our teammates to change the code without messing things up. We've learned the benefits of scheduling, communication and organisation. We've also taught ourselves modern frameworks and APIs in a short space of time, to make a web application that we are all proud of. While there are always ways that a project can go more smoothly and goals a project cannot meet, each bump on the road has been a lesson learned for our future endeavours.

Figure 3: The traditional end Rick