House Party - CI601 The Computing Project Report

By Alexander wood – 22827042
Supervisor – Jarod Locke
Second Reader – Andrew Blake

Word count – 6801
GitHub - https://github.com/acegoal07/HouseParty

Contents

Abstract

This project demonstrates how I designed and developed ‘House Party’ to target a gap I found in the market. ‘House Party’ is aimed at group music listening sessions and focuses on a website that can be used without the need for a music account or the installation of an app. The purpose is to allow people to access to add music to an existing queue at the same event whilst providing a fee service and allowing the host to manage it. I achieved this by building the website using the free to use ‘Spotify API’ which requires only the host of the party to have a ‘Spotify Premium Account’. My aum is that this interactive website will allow for more varied and inclusive events without the need for constant changing of devices, and apps.

1. Introduction

In this dissertation aspects of the research, design and development of the house party project are broken down into sections, explaining what was done and used through the project. The final product of the project will also be broken down explaining the parts needed for it to work and function while also explaining what they offer and how the user will interact with it.

1.1 The idea

The idea for the project is to create a website that allows users to log into their Spotify accounts and create a session which they can invite guests to, those guests invited will not need a Spotify account. Once the guests have joined the party, they will be able to use the website to add songs to the queue of the hosts Spotify account allowing for a joint listening experience.

1.2 Motivation

This idea emerged from the fact that there are multiple music streaming services, each with its own method of allowing people to share and control each other’s music, but these platforms do not support each other. At a party, there is no way to invite someone who uses a different platform, hence the concept of creating a website that allows people to add songs to the queue while eliminating the need for a platform-specific account or app.

1.3 Project aims and objectives

- The website should have a simple and easy to use design
- The website should be accessible
- The website should store any sensitive data securely
- Host can invite guests to their hosted parties through multiple methods
- Guests can join parties easily

- Guests can search for songs
- Guests can add songs to the host's queue
- Guests don't need a Spotify account
- Guests don't need an app
- Guests can view the song currently playing
- Host can restrict what songs are added such as not allowing explicit content or blocking duplicate songs
- Host can invite people using a QR code
- API security

1.4 Minimal viable product

Sign in to Spotify account

Allow the user who wants to host a party to login into their Spotify account easily.

Create and manage a party

Allow someone who is logged into their Spotify account to create a party which meets the needs of the party and manage the settings for the party.

Join a party

Allow guests to join a session hosted by a user that they have been invited to participate in.

Add songs to the host's queue

Allow guests using the party to add songs to the hosts queue through the website

2. Methodology

2.1 Project management techniques

The project methodology which was used to design this website is the waterfall methodology. This will be used for the design process to layout the requirements and the features that will and won't be in the final product. Then when it comes to the development process of the website, the waterfall methodology will be combined with a 'Kanban Board' to help manage and know what is being worked on and what's left to be worked on.

2.2 Testing techniques

While developing the website there were multiple different ways of performing tests to help find and fix bugs and improve the UI for the user's experience the testing types that were used are White Box', 'Black Box' and 'Unit Testing' each having their own benefits.

White box testing:

White box testing was performed as development happened and was used to test elements of the system as they were being implemented to make sure the implementation was correct and worked as expected before the changes were pushed to the GitHub.

Black box testing

Black box testing was performed by taking a small group of users and having them use the website. This testing was done to further identify bugs and issues that could arise while using the website. This testing also allowed for changes to the UI and new features to be discussed further, helping to improve the website and its usability.

Unit testing:

In additions to all the tests that were being performed using human input there were also tests put in place that run automatically to perform checks to make sure the page acts as it should depending on different variables, these tests were built using a package called 'Playwright'. An example of a test created can be seen in **Appendix 2.** Appendix 3 contains the results page that showing the results of the tests carried out on different platforms and browsers. This data is then further broken down and Appendix 4 shows where failures occurred in the code.

Some tests have been disabled due to security put into place in the API preventing any external users from accessing the API this means the tests can't be run locally and must be run on the server.

The unit tests were also linked to the GitHub repository so that when a pull request was requested between the dev branch and the stable branch the tests are run to make sure that there are no changes that have broken any functionality of the website.

2.3 Development tools and software

Visual Studio Code (VS Code):

For the coding and development of the project, it was decided to use the programming Integrated Development Environment (IDE) Visual Studio Code. This IDE was chosen due to familiarity and its extensive customization options, which allow for a more efficient and comprehensible development process.

GitHub:

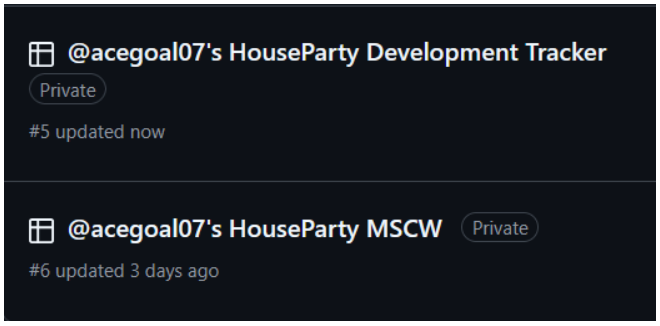
GitHub was used to store a backup of the code and allow for version control, this method creates a repository and uploads all code and files to it, storing them securely and enabling the creation of branches when features are added or changed which can

cause major issues, using GitHub provides the opportunity to undo any changes without losing or changing the code.

By using GitHub, changes can be controlled through branches, allowing the separation of new changes from a working and functional build by having a stable branch and a development branch. Once testing confirms all changes pushed to the development branch are functional, these changes can then be merged into the stable branch.

GitHub Projects:

To keep track of the project GitHub projects was used as it allows for both the code and progress tracker to be kept together as both are stored on GitHub while also offering the same features as other project progress trackers *(GitHub No date)*.



Brighton Domains:

To host the website, the decision was made to use Brighton domains which is already free to use and accessible. Allowing the website to be hosted on it straightway with no delay. It also allows for the backend to also be hosted straight away as it supports all the required tools. These include automatic PHP files, database events, databases and the ability to host HTML, JS and CSS meaning everything can be kept together on one server.

Draw.io:

To make wireframes for the website draw.io was used as it offers a wide range of features while also being free, easy to use and accessible online meaning no software must be installed making it easy to switch between devices.

OneDrive:

To make it easy to work across multiple devices and to provide an extra backup of the work OneDrive was used as it provides active backups meaning any changes being made to the project are automatically backed up to the cloud without needing any input providing an extra level of security to the files. OneDrive also has the benefit of being able to sync multiple devices to one OneDrive, meaning any file changes on one device are then updated on the second device allowing making it easy to switch between the two.

Playwright:

To perform unit testing on the website and its front-end functionality, Playwright was used to write automatic tests which perform a range of checks and then returns the outcome from the tests in a user-friendly way, making it easy to understand.

Bootstrap and Font awesome icons:

The font awesome and Bootstrap icon libraries were used throughout the website. These icon libraries are easy to use and provide a wide range of icons to use along with built in customizations for the icons *(Font awesome No date) (Mark Otto, J.T. No date)*.

Qrcode.js:

To generate QR codes for the website which are used to invite guests to the party a package called Qrcode.js was used. This package is a free to use package which was downloaded and hosted locally on the server with the website to make accessing it easy *(davidshimjs No date)*.

Google fonts:

The website uses a custom font called Roboto which was downloaded from google fonts and hosted locally within the website.

Lighthouse:

To help test the website performance and accessibility a tool called lighthouse was used which is a browser extension that runs a set of tests on the returning statistics on load time, performance and accessibility while also returning this data it provides information about areas which could be improved to improve the scores *(google 2016)*.

2.4 Project risk analysis

Risk Description	Likelihood of risk	Impact of risk	Action taken
Spotify API outage	LOW	HIGH	No action has been taken for this as there is nothing that can be done for this risk
Brighton domains outage	LOW	HIGH	No action has been taken for this as there is nothing that can be done for this risk
Externally hosted packages becoming unavailable	LOW	HIGH	To prevent any issues from external packages becoming unavailable instead of using third party hosting services the packages are hosted locally with the website
Exceeding Spotify API request rate limit	MEDIUM	LOW	There is no way to prevent this issue from occurring but instead there have been a process put in place to manage exceeding the rate limit, so the user experience is not affected

Loss of the project files from file corruption etc	LOW	HIGH	To prevent this from affecting development there were multiple backups put in place. These include GitHub, OneDrive and a copy hosted on Brighton domains allowing for the code to be recovered
A bug being pushed to the stable branch and hosted on the servers	LOW	MEDIUM	To prevent this issue from occurring all the code goes through multiple tests and checks before leaving the dev branch of the GitHub this stops any code containing bugs from being hosted on the servers

3. Research

3.1 Literary review

Spotify API terms of service:

This website breaks down the terms of service for the available APIs provided by Spotify showing what's allowed to be made using the APIs and what isn't. This website was easy to navigate and easy to understand making it extremely easy to check that what's being made meets the requirements of the terms of service.

Spotify Web API documentation:

This website breaks down the functionality of the Spotify web API showing how to use it and how to gain access to it. The documentation in some areas was complicated and took a lot of time to understand fully, but in some areas of the website it allows for you to try out API requests showing you what attributes you can send followed by showing you what the returned data looks like.

Spotify Design & Branding Guidelines:

This website broke down the requirements of the website regarding how it should look and what needs to be displayed on the website for it to be approved and taken out of development mode by Spotify. This website was clear and easy to understand giving examples and screenshots of what is expected to be shown on the website while also providing downloads and access to required content such as different Spotify logos for different scenarios.

Apple Music API documentation:

This website explained the capabilities of the Apple Music API and how to gain access to the API functions. The website was quite comprehensive but that made it complicated to navigate and understand at first; after exploring the website it became easier to navigate and understand.

Soundcloud API documentation:

This website broke down and explained the capabilities and functionality of the Soundcloud API. The website is quite simple and provides as much information as possible while making it clear with snippets of code and screenshots making it extremely easy to read and understand.

YouTube documentation:

This website explained the functionality and capabilities of the YouTube API. This website is simple in some areas but in others quite complex and hard to understand, screenshots of examples returned are provided and this helps to understand the expected outcomes of using the API functions

3.2 API research

Before development started a lot of time was put into researching the Spotify API and its functionality to find out how to make the Spotify login work and use it in the website. This was followed by research into how other functions from the API which are required for the website work and how they should be implemented in *(Spotify No date)*.

Once research around Spotify was completed, research around other music streaming platforms began. This research was carried out to discover if it would be possible to add support for multiple platforms to the website, however, the research proved that it would not be possible to add support for other platforms as they do not have the functions required for the website to work built into their APIs making it impossible.

3.3 Proof-of-concept

To learn and figure out how to use the Spotify API a proof-of-concept Node.JS was made to help learn how to use the API to perform the actions required for the website. This also helped to make sure it was possible to make what was envisioned *(Spotify No date)*.

3.4 User feature brainstorming group

To help find out what people thought about the idea a small group of people was put together and three questions were asked:

1. Would you use this service,
2. what features would you want this service to have
3. when would you use this service.

The group was used to test if the idea would be used and what features something like this would need for people to be interested. Noted below are some of the ideas/features the group came up with.

- The ability to vote on the order of songs
- Restricting how many songs can be added by each guest
- Ability to block duplicate songs
- Hosts song queue automatically clears after the party ends
- Ability to join the party using a QR code

Some features were included in the final product. Due to time constraints not all the ideas/features were included but there are also some that could not be added due to limitations of the Spotify API limiting access.

3.5 Similar services

Spotify offers a similar service called Spotify jam which allows users to invite people to control their music. This service allows guests more control over the music enabling guests to remove, add, pause and skip songs. These are positive features but can become an issue as this allows all users full control, meaning all users can remove or skip other user's songs. To use Spotify jam, all users joining the session require a Spotify account and access to the Spotify app. Without these users are unable to join. This can restrict who can join and add songs to the queue due to users using multiple different music platforms *(Spotify no date)*.

3.6 User testing

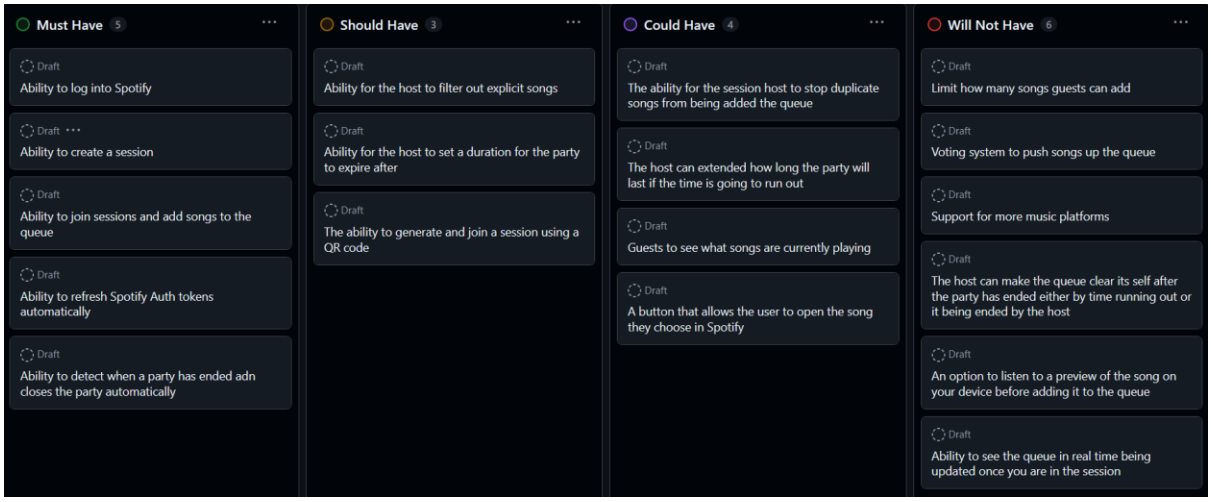
Before the final prototype was finished a small group of users were put together and asked to test the website this was done to see how users interacted with the website in order make any necessary improvements to ensure that it was intuitive and user friendly, the testing also helped identify bugs that were not found during unit testing and Whitebox testing.

4. Product description

4.1 Planning

To plan the development of the website the MSCW method was used to help prioritize the features and functionality to be added to the project a screenshot of the MSCW made can be seen below. This method categorizes the importance of each feature into four different categories:

- must have,
- should have,
- could have,
- and won't have *(ProductPlan 2024)*.



After the MSCW was made it was then taken and used to create a progress tracker which included the main functionality and design of the website but also listed the features that were to be added to the website separate from the main functionality this progress tracker was also used to line out the order in which parts of the website were going to be made in a screenshot from the GitHub pages can be seen in **Appendix 9.**

4.2 Designing

Following the planning phase work on the design of the website was started, the first step was to create a simple sketch of the layout and where things would go on the page. These sketches were just black and white. An example can be seen in **Appendix 5**.

The basic sketches were then taken and made into wireframes using draw.io adding more details and flushing out what the user interface would look like including features. This can be seen in **Appendix 6**.

Design Decisions

Languages Used:

When it came to the languages used to build the system, JS, HTML, CSS, and PHP were chosen. These languages were selected because Brighton domains support hosting of these types of files, and there is existing knowledge on how to use these programming languages.

Mobile First Design:

Due to the system mainly being used on mobile devices the decision was made to focus primarily on a design for mobile devices which does not have a separate view for larger screens, instead the view is scaled to the screen size. By doing this, it allowed for time to be saved and, pre focus on features that are more important.

Minified Files:

To help improve performance of the website, minified JS and CSS files were used to enhance loading speeds and deliver a smoother user experience.

Basic Design:

A basic design was chosen for the website to make it easy for anyone to pick up and use. This decision was also influenced by the environment in which the app would be used, user would not want complicated interfaces with fancy designs which make it harder to use.

Micro Services architecture:

When building the system, a micro services architecture was chosen to allow parts of the website to be reused in other areas, as they work independently. This also improves maintainability because the code is easier to locate and update, and any modifications to one component are automatically applied to all parts of the website using the update function.

Spotify:

Spotify was chosen over other music platforms for several reasons.

1. access to a Spotify account, whereas there is no access to accounts from other platforms.
2. Spotify's API is free to use and has the required functionality needed to build the website, which other platforms do not offer or require a subscription.
3. Spotify has a large user base, allowing more user to access the website compared to using a less popular music platform.

Spotify guidelines:

Some parts of the website's design had to be changed from the original plan due to Spotify's guidelines around the design and interaction with the website. These guidelines must be met for the website to pass inspection by Spotify before it can be taken out of development mode. An example of some of the guidelines required by Spotify are the need to have an authorised Spotify logo presented along with any data retrieved by the API this also needs to be followed by a button or link which can take you to any songs that you have retrieved data about from the API the link must take the user to the Spotify's website where they can listen and view the song further another example is the specific specifications for border radius on album art Spotify provides specific radius sizes for large and small screen these are 2px for small screens and 4px for large screens.

Server security:

When adding security to the server side of the website, it was decided to restrict external connections to the database or API, allowing only those that originate from the same server. Along with restricting who can send API requests there was also data sanitizing implemented into the API this helps to prevent SQL injection further by removing any characters that could be used to initiate a SQL injection improving the security of the API and protecting sensitive data further this security was implemented

using the built in mysql_real_escape_string which is a built in function within PHP (*Hypertext preprocessor No date)*.

Why a website:

The decision to make this a website was because of ease of accessibility, users would not be required to download an app they simply visit the website to use it.

Hosted locally:

To help improve performance and reliability all files and tools used on the website was hosted on the server including all fonts and external libraries, by doing this, it is guaranteed that all files used would always be accessible, therefore if an external library stopped working this would not prevent the website from working.

4.3 Prototyping

During the development process of the website there were 2 prototypes: the first prototype was used to evaluate the development of the backend of the website. During this process a simple UI was introduced which allowed for easy testing. The second prototype was to develop the UI to finalise the design for users. These 2 prototypes were separated allowing for the focus to be on a specific section of the website, allowing for focus to be on one thing at a time thereby improving performance and allowing for testing to be done on one specific part at a time providing more accurate testing.

4.4 Development

The development of the website was broken down into pages due to the website being built using a micro service architecture this meant that each page was being focused on independently and only one page was being focused on at any one time. This worked well as none of the pages depended on each other as they all worked independently. Alongside the development of the pages the API was also being worked on. This is because all the pages used different API functions due to them performing different actions. This type of development also applied to the micro services, for example the ModalHandler. This service was being used by multiple pages, meaning it was being added to and modified during the development of other pages to add more functionality to it but also to enhance and improve the service.

Once development of both the front end and back end of the website was complete and both were combined into the final functional product, was allowing for improvements and enhancements to be made to both components improving performance, adding more debugging, adding more error handling, improving maintainability, and improving functionality this was mainly done for the backend as during the process of making the front end areas for improvement were found in the API's.

4.5 Problems during development

Login and verification issue:

A major problem that arose during the development of the website involved the information used to verify the hosts identity which is both stored in the hosts cookies and the database. When a user logs into the website a refresh token is returned to the server. This combined with the hosts user ID is used to verify their identity before performing and involving managing or creating a party. It was noticed that occasionally the refresh token was returned when logging in and would change from the previous one provided, meaning that on occasion a user created a party logged out, making them unable to manage their party.

The solution to this was to add a check into the API that handled the login. The check would see if the user has created a party with a refresh token already and if the new refresh token did not match the old one the user would instead have the old one added to their cookies instead of the new one, making no difference to the users experience but making it so the verification issue would not happen again.

API time miss match:

An issue which affected the development was a time mismatch. The website was designed to use the UK time zone to work out when to close expired parties, but parties would always end an hour before they should. With further investigation it showed that the server was always an hour behind. The solution to this was when the expiry time is stored an extra hour is added on to counter the issue, allowing everything to work as intended.

Spotify extension submission:

Once there was a working prototype the website was submitted to Spotify to be taken out of development. This process can take a minimum of a month to be completed so a submission was made early so any issues found could be fixed. The submission that was made was declined due to an issue with how the Spotify logo was being displayed and that there was no link to the song on Spotify. These issues where fixed quickly once they were brought to my attention, but a resubmission was not sent to Spotify due to time constraints and how long it takes to get a reply from the Spotify team, but this is planned to be done in the future now that all the changes have been made to open it up to more users.

Unregistered user being able to login:

An issue was discovered involving the Spotify login API this issue involves the Spotify login API allowing none registered users from logging into the website this causes the websites functionality to be unresponsive because to use the website while it’s in

development mode you need to be registered by the owner on the Spotify website, but the API does not distinguish between these users automatically to fix this more checks were added to the websites login API to catch non registered users and then send them to the login error page.

Tab functions unresponsive in popups

During user testing a user pointed out that in some scenarios while using the website the tab function to navigate would not interact with the popups this was fixed by adding a focus grabber to the modalHandler which meant when the popup was opened the focus would be forcefully moved to the popup and locked to only interact with it until the popup was closed.

Music not playing error

During user testing an error was discovered involving the Spotify API this error is that you are unable to add songs to the hosts queue unless they have music playing on Spotify to fix this an extra API request was put into place to send a request to Spotify to check if there is music playing on the hosts account before attempting to add a song to the queue preventing any API issues and allowing for a specific message to be shown when this occurs.

4.6 Limitations

Brighton domains:

Plans were made to use a WebSocket with Nodejs to handle requests between the user and Brighton domains. This was chosen to improve the performance of the user's device and put less strain on the servers. However, as Nodejs is no longer present on Brighton domains, a polling system was used instead. Although this system has some performance drawbacks due to constant requests being sent to the server to keep the data up to date, it serves as a suitable replacement for the WebSocket as it provides the same result that a WebSocket does *(Kilbride-Singh, K. 2023)*.

Time constraints:

To make sure that all the features planned in the progress tracker would be completed along with the website not all the features that were envisioned for the website were added this was done to make sure that the website would be finished to a high standard without any bugs and at a later date the features that did not make it into the prototypes could be added. An example of a feature that was not add due to time constraints was the ability to restrict how many songs guests can add to the queue which would have offered the host more control over the number of songs added.

Real world testing:

The idea of performing real world testing was considered, but due to testing only being possible with user from within the same university module, doing so in the real world would be unethical and break multiple rules imposed on the assignment. Consequently, this kind of testing was not conducted, and instead, more unit testing and Whitebox testing were used instead.

Spotify rate limit:

Due to Spotify implementing a rate limit into their API preventing loads of API requests from being sent over and over it was decided that some features would not be implemented into the prototype of the website in case it caused the rate limit to be hit preventing the website from functioning for a certain amount of time. One such feature was the ability to see what song was currently playing this feature was not implemented as it would require constant API requests to function.

4.7 Final product

The final product of the website is built up of multiple parts including both server side and client-side code. The server-side codebase includes two automated PHP scripts. One of the scripts handles closing parties which have expired and the other handles refreshing the party's authentication token for Spotify. The server-side also includes a database which is used to store all the necessary information about the party including verification information, settings and authentication tokens. A screenshot displaying the layout of the database can be seen in **Appendix 8**. The database also has a built-in automatic function which runs every second to check for expired parties and close them, this performs the same purpose as one of the automated PHP scripts as it allows for the automatic functions to be setup in different ways depending on the server.

The client-side codebase consists of many parts including the HTML for the pages included on the website. These pages are also linked to a JavaScript file specific to the page which handles all the functionality of the page. All the pages are also linked to a CSS file which handles the styling for the pages, controlling how they look and making sure all page's match when it comes to design. The client-side codebase also contains multiple JavaScript tool to perform certain actions which is shared across multiple pages. An example of one of these tools is the cookies.js which is used across multiple pages to handle and manage the user's cookies.

Along with the codebase a logo was also designed for the website as a way for people to identify the website. This logo appears when you share the link to the website and when an invite to the party is shared, a screenshot of the logo can be seen in **Appendix 7**.

The website is then comprised of six HTML webpages all serving a different purpose for the website these pages are the homepage, 404, loginerror, join, dashboard and party.

Homepage:

The homepage is the first page that any user going to the website will see unless the user has been invited to the party by another user. This page offers the ability to navigate to the joining page, to login to Spotify account, to navigate to the dashboard page if logged into their Spotify account, a screen shot of the page can be seen in **Appendix 10.**

404:

The 404 page is shown when accessing a page which does not exist in website a screenshot from this page can be seen in **Appendix 11** this page has no functionality other than the ability to return to the homepage.

Login error:

The login error page is used to tell the users about any issues that occur while they try to login into their Spotify account through the website an example of one of the errors can be found in **Appendix 12** the page can show 4 different error screens involving no premium account, in development mode, API request limit reached and a general error screen.

Join:

The join page is used for guests with a party join code, users can enter the code and join the party, this page also allows the user to navigate back to the homepage a screenshot from this page can be seen in **Appendix 13**.

Dashboard:

The dashboard page is separated into 2 screens a create party screen which can be seen in **Appendix 14** and a party management screen which can be seen in **Appendix 15**. The create party screen offers the user the ability to enable settings such as blocking duplicate songs and allowing explicit songs it also offer the user the ability to control how many hours the party will stay active for.

The party management page offers the ability for hosts to adjust settings for their active party these included being able to either enable or disable settings, extend the parties duration and being able to end the party early before the duration they set has run out. The page also has information about how the website and party works and contains information about joining parties and displays a QR code which can be used for guests to join and other important information about joining and other functions which can be used to invite guests.

Party:

The party page is where invited guests can add songs to the hosts song queue this page also offers the guest the ability to search for any song available on Spotify through the

search function it will then return all the available results, a screenshot from the page can be found in **Appendix 16**.

Critical review

The house party website was created to fill a gap in the market of group music apps and to help bring people together while they are listening to music in groups. It allows all invited guests to add songs to the group's song queue, while restricting control over all the actions of music player just to the host.

House party has met most of the aims and objectives which were set out for it at the beginning of the document. One of the aims of for the website was for it to have an easy-to-use UI which house party has done very well with a very basic, clear to understand UI making it easy to use and accessible to everyone, while still providing all the functionality that was planned to be added to it. The basic design also helps it to meet the aim to be accessible as it makes it easy for tools like screen readers to interpret the website. Furthermore, a lot of development time while working on the UI was spent making sure all aspects of the website would be accessible, meaning that all parts of the website can be used by everyone. Furthermore, the accountless experience lets invited guests gain access and helps to make the website easy to use as invited users do not need to do anything except either enter a join code or go to a URL to access the party before they can start searching for songs and adding songs to the queue.

House party was also made to combat the need for users to have an account or app which house party does successfully, offering a free and accountless experience for guests. Unlike other services this means it is not restricted to just users who use one music platform, allowing anyone to be invited to add songs to the music queue.

The final prototype of the website does not have all the features and tools that were planned for the website. This was planned to allow for the main features and functionality of the website to be perfected and finalised so the final prototype would be of a high standard, but the features that weren't available in the protype could be added later to build upon the final product. An example of a feature that did not make it into the final product is the ability to restrict how many songs a guest can add to the queue. This feature was not added in this version due to time constraints pushing the focus on to the main features of the website.

The security Implemented into the API and system is good and effective, but they could be built upon and improved as these are more basic security measures and more sophisticated ones could be implemented with more time and resources.

The development of house party has led to a lot of lessons being learnt around development and research which in the future can be applied to jobs and more projects

that are undertaken. An example of a lesson that was learnt during the development of house party was the ways to implement security into the API built in PHP. This information is extremely helpful and helps to enhance the pre-existing knowledge around PHP and building APIs by adding knowledge which will be needed in the future while working with these systems. This knowledge also helped to improve the APIs that were built by showing the standard way to build them and implement everything that's required for it to work.

The user research done both before and during the project was very important and helped a lot during the design and development process, but more could have been done to help further the project more and improve both the user experience and the features featured in the website. If time would have allowed, more research could have been done to help refine the design and features more to better implement and improve them.

For the hosting of the website Brighton domains was used because it was free and required no setting up to gain access to it. This was an effective way to get it active easily, but a more powerful hosting service could be used like AWS if funding was acquired. By using AWS, it would reduce the risk of any outages affecting the website due to the service having system in place to prevent outages and downtime. By using a different hosting service this would also come with the benefit of being able to choose features that the server would have allowing for future versions to use a WebSocket instead of a poling system meaning it will have better performance.

During the project GitHub was used for version control and to always keep a backup of the project. This was further built upon by having two branches on the GitHub repository: the dev branch and the stable branch. By doing this any code that could contain errors or changes that could affect the websites functionality were kept on the dev branch until after testing was performed, which is then when it was moved to the stable branch after the code was cleared of bugs and it was known that the code broke no functionality.

Conclusion

In conclusion the project was completed successfully and to a high standard presenting research and a working prototype of the website that was designed. There are areas of the project that could be improved and built upon adding more features and improving the user interface and experience for the users. This can be done by adding more control features for the host allowing for them to have more control over the party and how the party works and by improving the websites API to make it more efficient and more robust against errors and attacks, improving the website for the users by making is safer and more secure for users to use. Along with improving the code more user-based

research and testing would also help with the design of the website and would be an effective way to validate the usefulness of the project along with helping to identify areas in which the prototype can be improved.

References

Spotify Web API (no date) Web API | Spotify for Developers. Available at: https://developer.spotify.com/documentation/web-api (Accessed: 20 September 2024).

Spotify developer terms (no date) Spotify Developer Terms | Spotify for Developers. Available at: https://developer.spotify.com/terms (Accessed: 20 September 2024).

Start or join a Jam (no date) *Spotify*. Available at: https://support.spotify.com/uk/article/jam/ (*Accessed: 20 September 2024*).

API reference | YouTube Data API | google for developers (no date) Google. Available at: https://developers.google.com/youtube/v3/docs/ (Accessed: 10 December 2024).

Apple Music Api (no date) Apple Developer Documentation. Available at: https://developer.apple.com/documentation/applemusicapi (Accessed: 10 December 2024).

SoundCloud for developers (no date) API - Guide - SoundCloud Developers. Available at: https://developers.soundcloud.com/docs/api/guide (Accessed: 10 December 2024).

Moscow prioritization (2024) ProductPlan. Available at: https://www.productplan.com/glossary/moscow-prioritization/ (Accessed: 11 December 2024).

Design & Branding Guidelines (no date) Design & Branding Guidelines | Spotify for Developers. Available at: https://developer.spotify.com/documentation/design (Accessed: 13 February 2025).

Roboto (no date) Google Fonts. Available at: https://fonts.google.com/specimen/Roboto (Accessed: 14 February 2025).

About projects (no date) GitHub Docs. Available at: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects (Accessed: 15 February 2024).

Mark Otto, J.T. (no date) Bootstrap, Bootstrap · The most popular HTML, CSS, and JS library in the world. Available at: https://getbootstrap.com/ (Accessed: 14 February 2025).

Font awesome (no date) Font Awesome. Available at: https://fontawesome.com/ (Accessed: 14 February 2025).

davidshimjs (No date) Qrcode.js. Available at: https://davidshimjs.github.io/qrcodejs/ (Accessed: 14 February 2025).

Introduction to lighthouse ： chrome for developers (2016) Chrome for Developers. Available at: https://developer.chrome.com/docs/lighthouse/overview (Accessed: 15 February 2025).

Fast and reliable end-to-end testing for modern web apps (no date) Playwright. Available at: https://playwright.dev/ (Accessed: 15 February 2025).

Hypertext preprocessor (no date) PHP. Available at: https://www.php.net/manual/en/mysqli.real-escape-string.php (Accessed: 17 April 2025).

Kilbride-Singh, K. (2023) Long polling vs WebSockets - which to use in 2024, Ably Realtime. Available at: https://ably.com/blog/websockets-vs-long-polling (Accessed: 23 February 2025).

Appendices

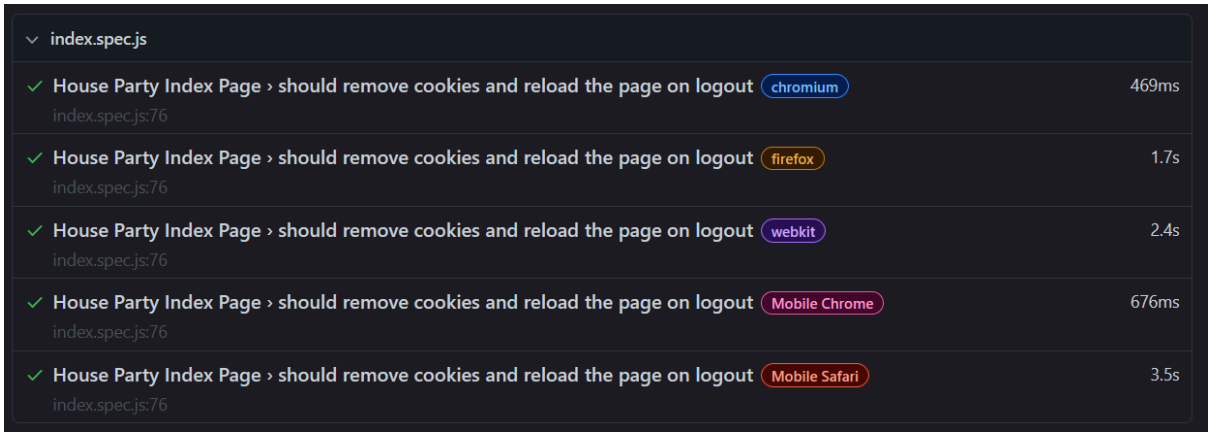
Appendix 1:

Date	Meeting notes
26/09/2024	• Spoke about project idea • Talked about the legal side of the project • Spoke about what research needs to be done into the terms of service
03/10/2024	• Updated on legal research • Updated on project progress
10/10/2024	• Progress update on designing the project • Did ethical form • Updated on project progress
24/10/2024	• Updated on project progress • Spoke about submitting the Spotify API forms to have the project recognised by them
13/11/2024	• Spoke about the interim report focusing on what I have in It already and what I could add to it • Updated on project progress
18/02/2025	• Updated on development of the project • Shown a demonstration of a prototype working • Updated on the status of Spotify Application • Discussed features that could be added or discussed in report • Discussed the report of the dissertation

Appendix 2:

```
test('should remove cookies and reload the page on logout', async ({ page, context }) => {
    await context.addCookies([
        {
            name: 'refresh_token',
            value: 'dummy_token',
            domain: '127.0.0.1',
            path: '/',
            expires: Math.floor(Date.now() / 1000) + 3600
        },
        {
            name: 'host_id',
            value: 'dummy_host',
            domain: '127.0.0.1',
            path: '/',
            expires: Math.floor(Date.now() / 1000) + 3600
        }
    ]);
    await page.goto(basePath, { waitUntil: 'load' });
    const logoutButton = page.locator('button#logout-button');
    await logoutButton.click();
    const cookies = await context.cookies();
    expect(cookies.find(cookie => cookie.name === 'refresh_token')).toBeUndefined();
    expect(cookies.find(cookie => cookie.name === 'host_id')).toBeUndefined();
    await expect(page).toHaveURL(basePath);
});
```

Appendix 3:

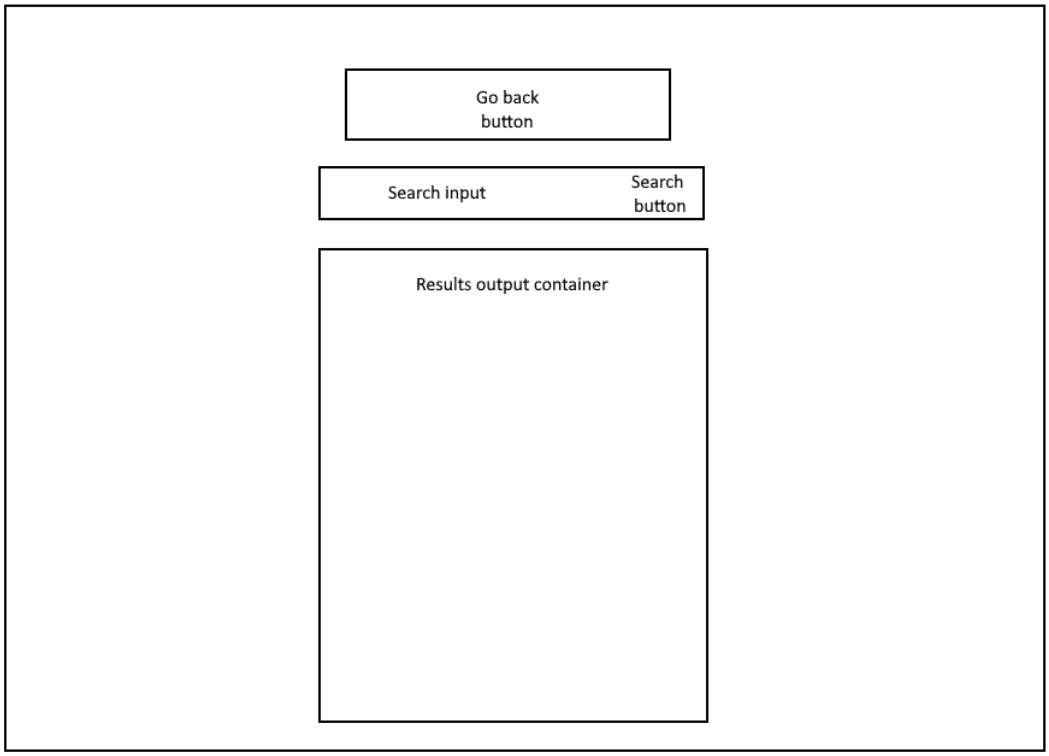


Appendix 4:

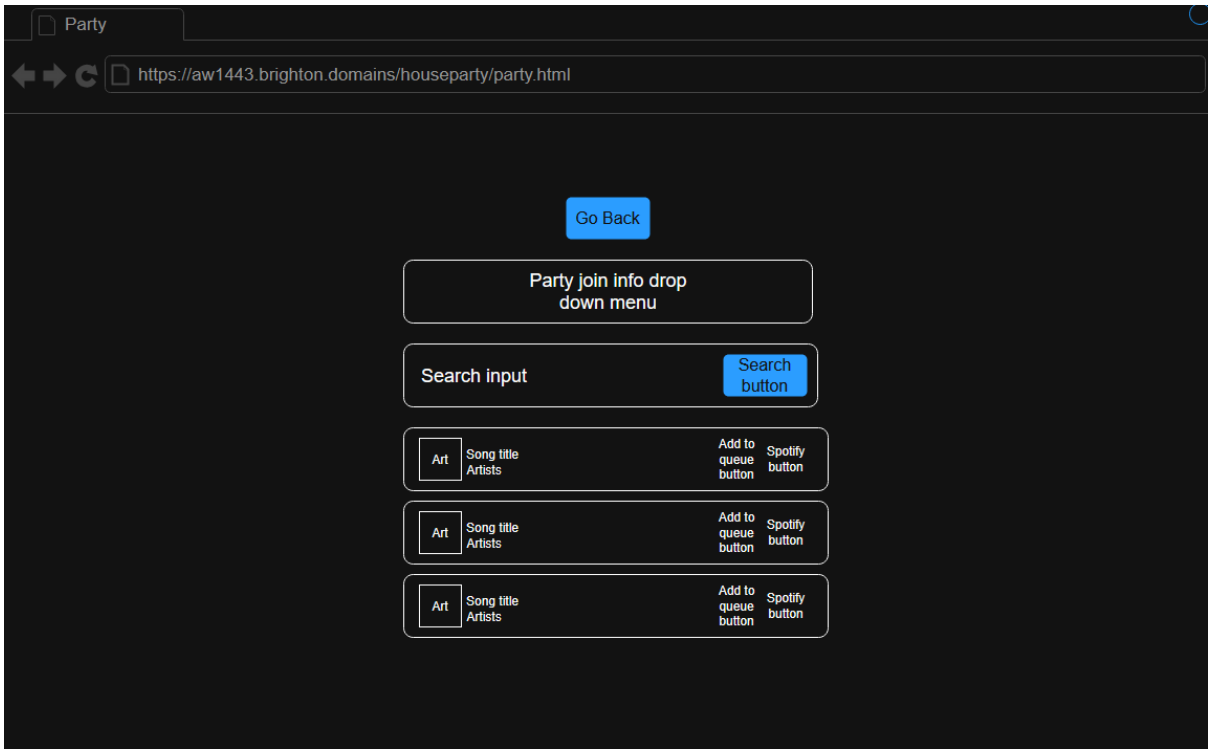
Test Steps

Before Hooks	425ms
browserContext.addCookies — index.spec.js:77	4ms
page.goto(http://127.0.0.1:3000/index.html) — index.spec.js:93	78ms
locator.click(button#logout-button) — index.spec.js:95	149ms
browserContext.cookies — index.spec.js:96	2ms
expect.toBeUndefined — index.spec.js:97	1ms
expect.toBeUndefined — index.spec.js:98	0ms
expect.toHaveURL — index.spec.js:99	28ms
After Hooks	14ms

Appendix 5:



Appendix 6:

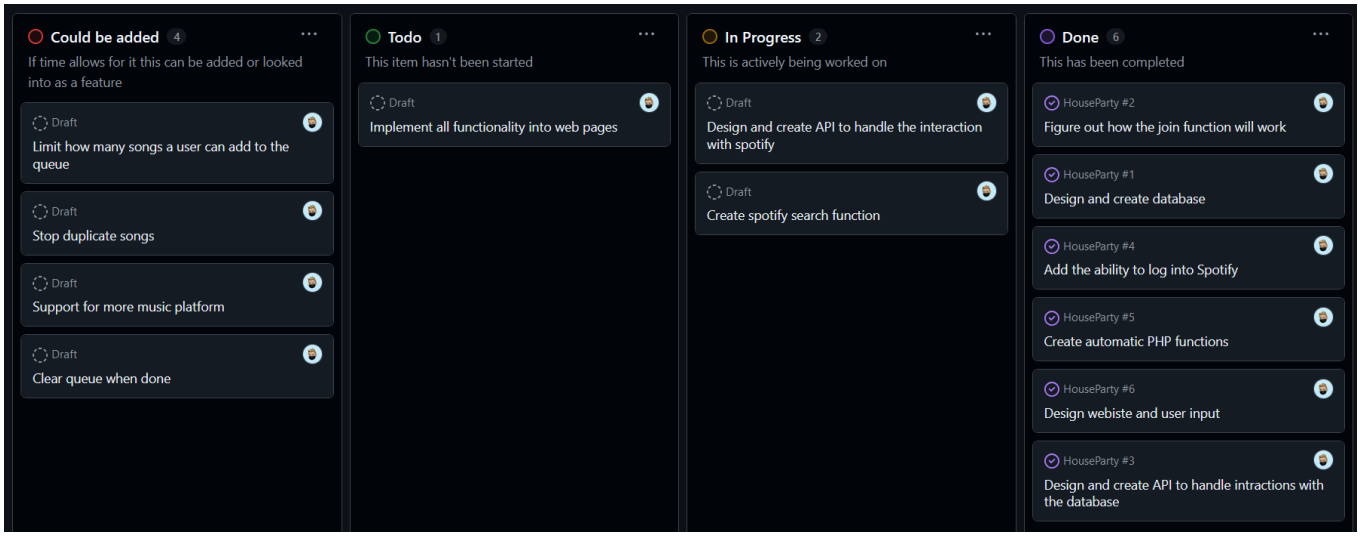


Appendix 7:

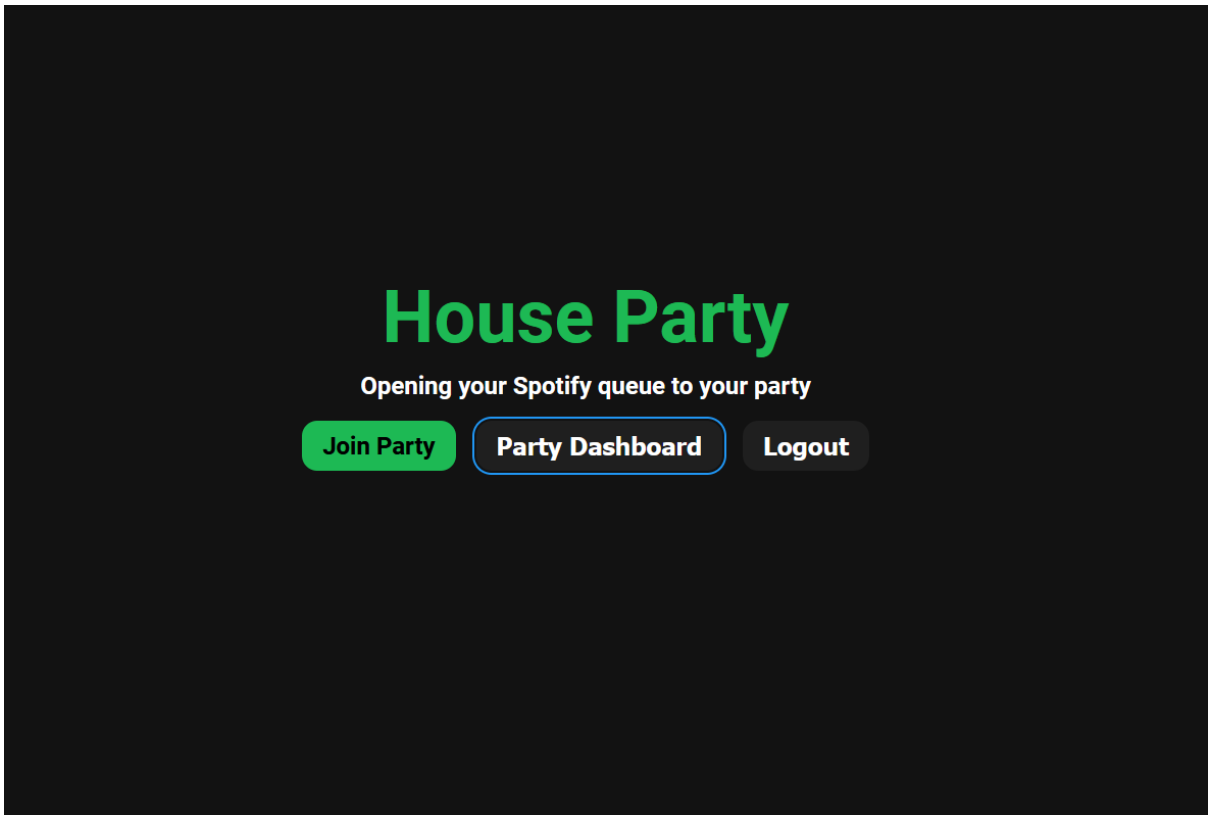
Appendix 8:

- `parties`
  - `party_id` - Id which is going to be used to join the party (Primary key)
  - `host_id` - The id of the spotify user
  - `access_token` - Token used to send api request for the host
  - `refresh_token` - Token used to refresh the access token after it expires after 1 hour
  - `token_expires_at` - Timestamp the access token will expire at used to determine when to refresh the access token
  - `party_expires_at` - Timestamp for when the party expires and needs to be closed
  - `explicit` - Whether or not to allow explicit songs to be added
  - `duplicate_blocker` - whether or not to allow duplicate songs to be added to the queue

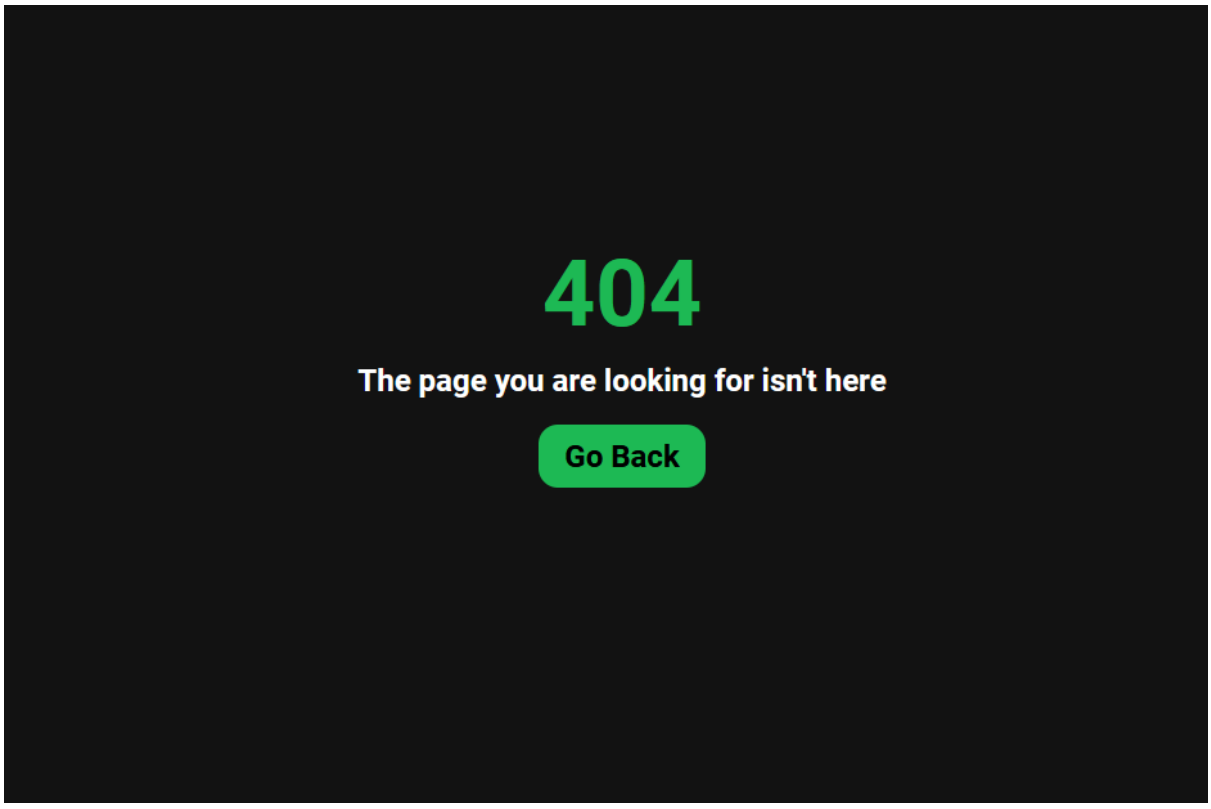
Appendix 9:



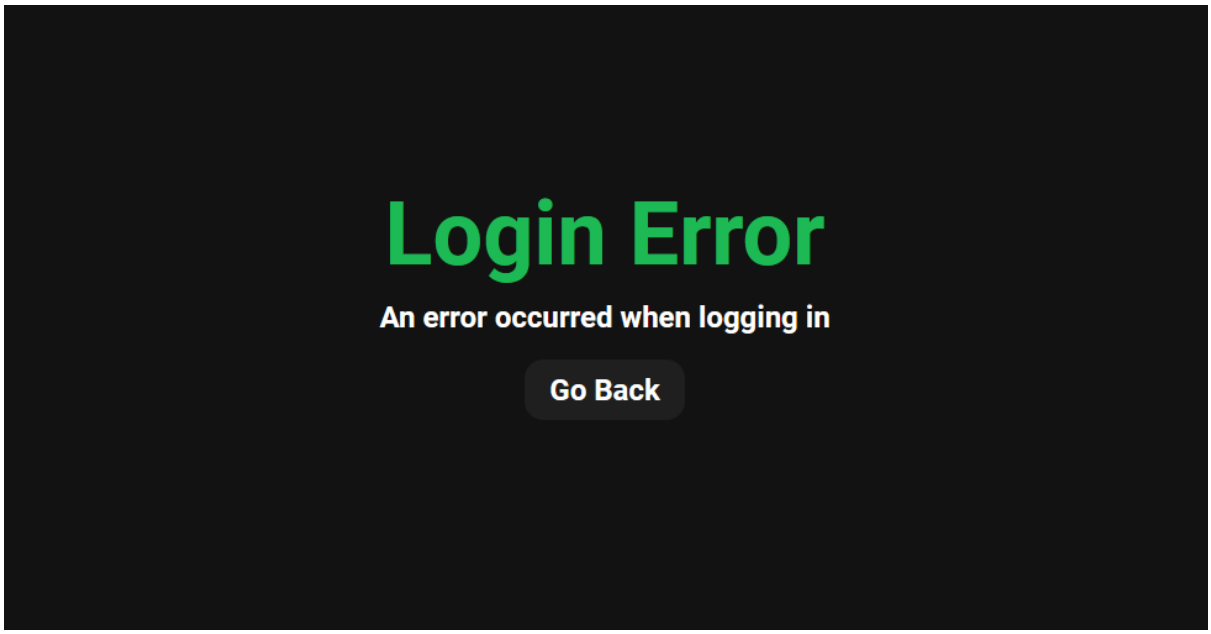
Appendix 10:



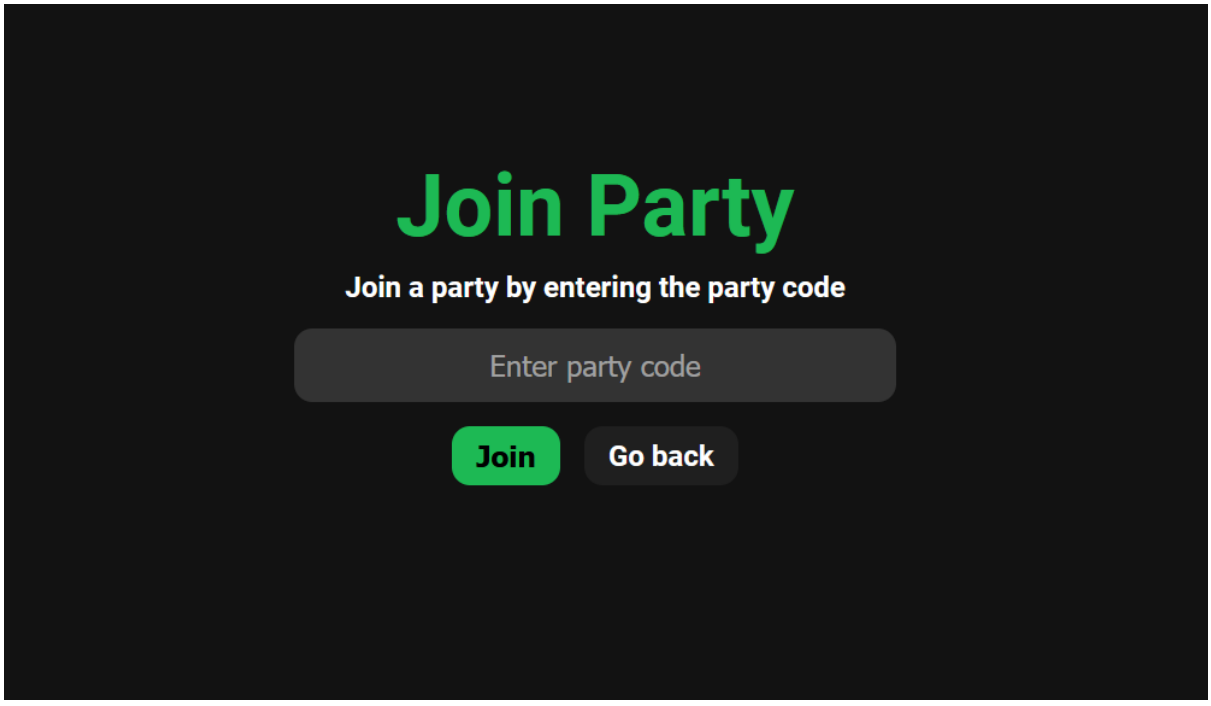
Appendix 11:



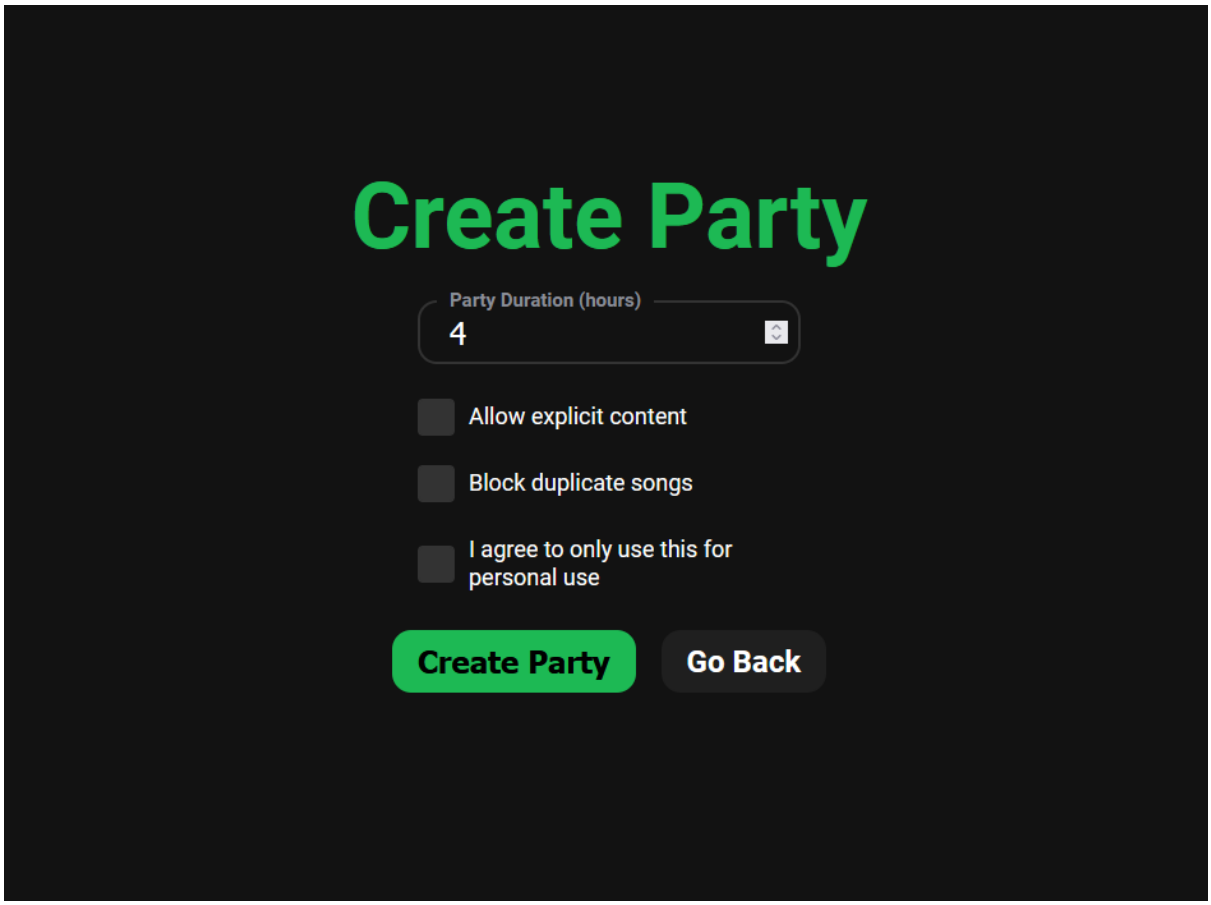
Appendix 12:



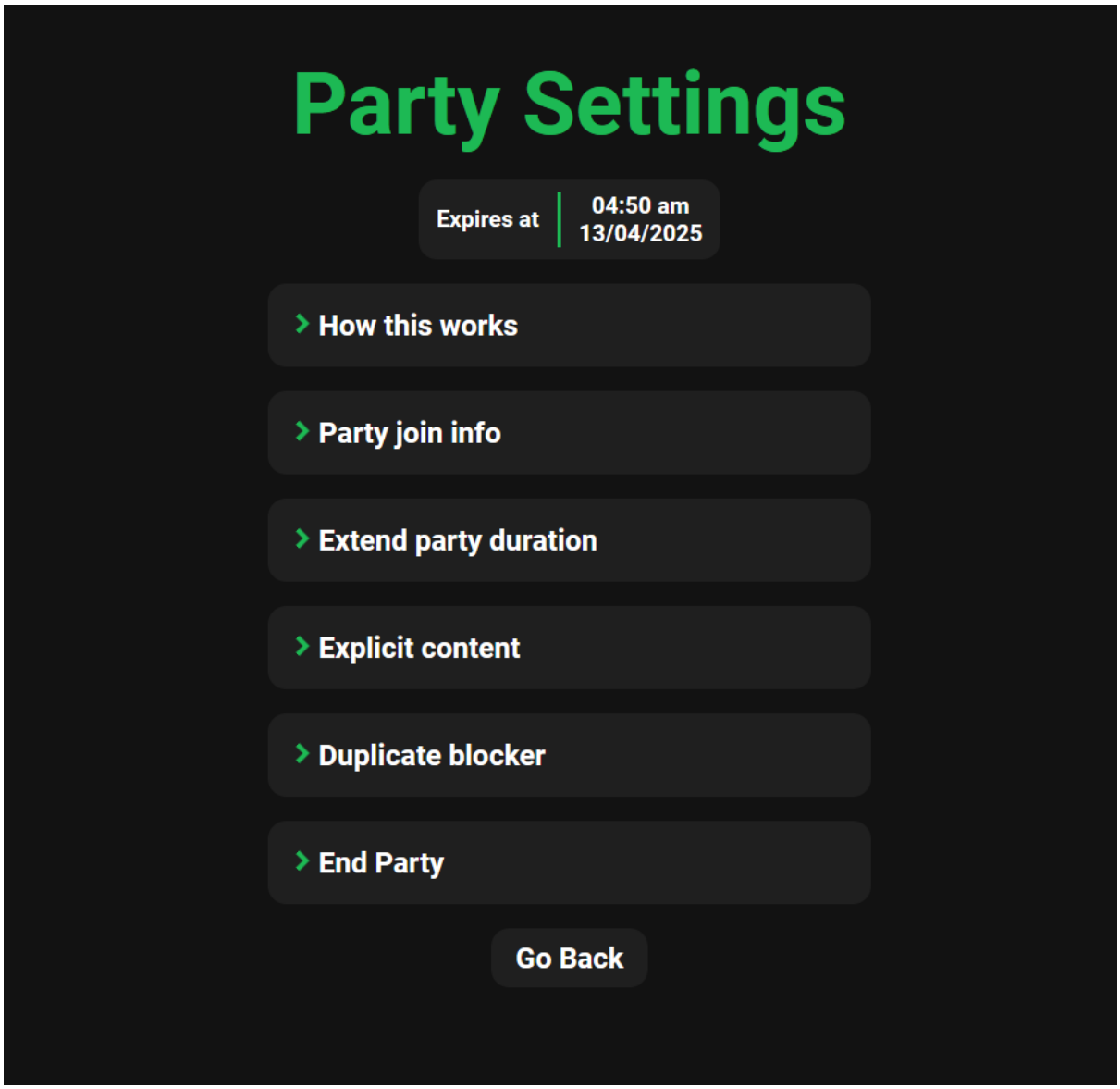
Appendix 13:



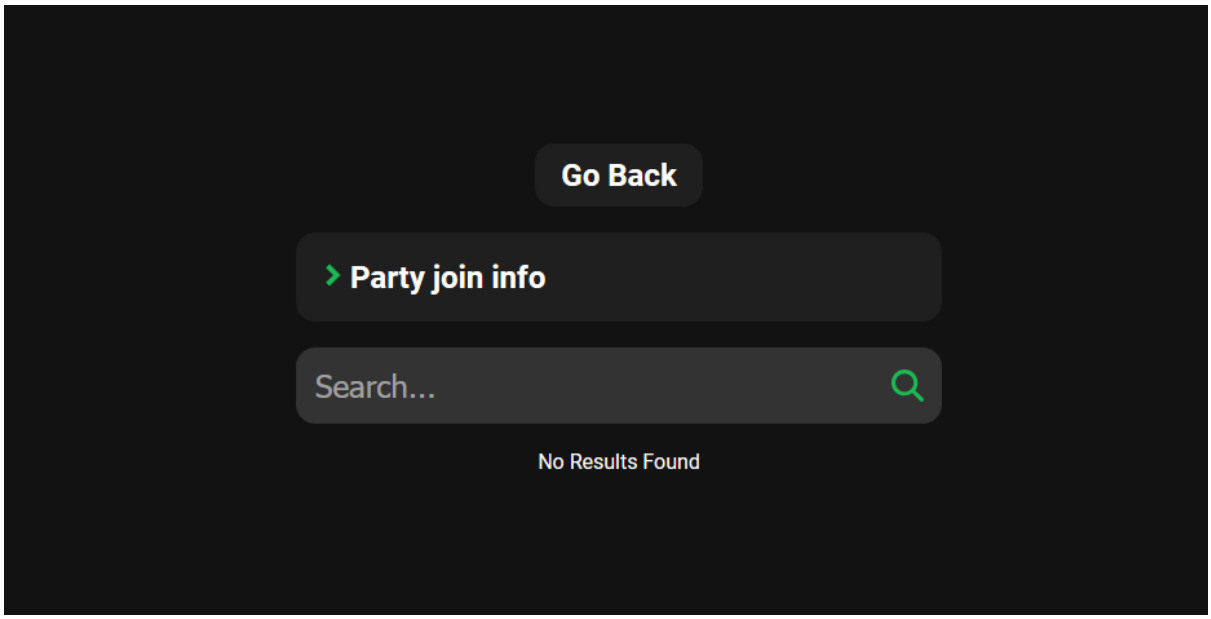
Appendix 14:


Create Party
Party Duration (hours)
4
Allow explicit content
Block duplicate songs
I agree to only use this for personal use
Create Party
Go Back

Appendix 15:


Party Settings
Expires at
04:50 am
13/04/2025
How this works
Party join info
Extend party duration
Explicit content
Duplicate blocker
End Party
Go Back

Appendix 16:



Bibliography:

Spotify Web API (no date) Web API | Spotify for Developers. Available at: https://developer.spotify.com/documentation/web-api (Accessed: 20 September 2024).

Spotify developer terms (no date) Spotify Developer Terms | Spotify for Developers. Available at: https://developer.spotify.com/terms (Accessed: 20 September 2024).

Start or join a Jam (no date) *Spotify*. Available at: https://support.spotify.com/uk/article/jam/ (*Accessed: 20 September 2024*).

API reference | YouTube Data API | google for developers (no date) Google. Available at: https://developers.google.com/youtube/v3/docs/ (Accessed: 10 December 2024).

Apple Music Api (no date) Apple Developer Documentation. Available at: https://developer.apple.com/documentation/applemusicapi (Accessed: 10 December 2024).

SoundCloud for developers (no date) API - Guide - SoundCloud Developers. Available at: https://developers.soundcloud.com/docs/api/guide (Accessed: 10 December 2024).

Moscow prioritization (2024) ProductPlan. Available at: https://www.productplan.com/glossary/moscow-prioritization/ (Accessed: 11 December 2024).

Design & Branding Guidelines (no date) Design & Branding Guidelines | Spotify for Developers. Available at: https://developer.spotify.com/documentation/design (Accessed: 13 February 2025).

(No date a) Every noise at once. Available at: https://everynoise.com/ (Accessed: 13 February 2025).

(No date) Tuner. Available at: https://tommygeiger.com/tuner/ (Accessed: 13 February 2025).

Roboto (no date) Google Fonts. Available at: https://fonts.google.com/specimen/Roboto (Accessed: 14 February 2025).

About projects (no date) GitHub Docs. Available at: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects (Accessed: 15 February 2024).

Mark Otto, J.T. (no date) Bootstrap, Bootstrap · The most popular HTML, CSS, and JS library in the world. Available at: https://getbootstrap.com/ (Accessed: 14 February 2025).

Font awesome (no date) Font Awesome. Available at: https://fontawesome.com/ (Accessed: 14 February 2025).

davidshimjs (No date) Qrcode.js. Available at: https://davidshimjs.github.io/qrcodejs/ (Accessed: 14 February 2025).

Introduction to lighthouse ： chrome for developers (2016) Chrome for Developers. Available at: https://developer.chrome.com/docs/lighthouse/overview (Accessed: 15 February 2025).

Fast and reliable end-to-end testing for modern web apps (no date) Playwright. Available at: https://playwright.dev/ (Accessed: 15 February 2025).

Hypertext preprocessor (no date) PHP. Available at: https://www.php.net/manual/en/function.filter-var.php (Accessed: 17 April 2025).

Hypertext preprocessor (no date) PHP. Available at: https://www.php.net/manual/en/mysqli.real-escape-string.php (Accessed: 17 April 2025).

Kilbride-Singh, K. (2023) Long polling vs WebSockets - which to use in 2024, Ably Realtime. Available at: https://ably.com/blog/websockets-vs-long-polling (Accessed: 23 February 2025).