Project Planning Report

Due: November 1st, 2024

GitHub Repo: https://github.com/CMPT-276-FALL-2024/project-22-vines

Murat Guler | 301461628

Greg Kuchta | 301479235

Roi Lee | 301560420

Brandon Chattha | 301579323

Table of Contents

APIs - Roi & Brandon

Spotify

Spotify is our major API, which our project will mainly depend on. It includes many beneficial features such as searching for music, controlling what is being played and managing the playlist. Spotify is one of the most used music platforms and includes so many different genres of music that it will be perfect for our project.

Feature 1: Search for Music and its details

The major function of music-recommending programs is to be able to search for music. Spotify API allows the user to search for them based on the genre that they are looking for, artists or songs that they like and create a recommended list of songs related to it. Another important part of the API is that the user can see all the details of the song that they find, such as the artist, its album and its release date. Therefore, this function will be the most important part that we include in our project.

Feature 2: Control / Interact with the playback

One of the 10 Heuristics is to be able to give the user control and freedom (3), and this function allows that perfectly. Spotify API allows the user to control what is being currently played on the screen and gives the user the control to do anything they want with the music. Users can mix the playlist, go to the previous song, stop or start the music, skip the music or loop the music. There are so many features that Spotify API has to allow the best user experience, that this is an amazing feature that we are going to implement on our project.

Feature 3: Manage personal library

For users who found the perfect music for themselves, it would be very annoying if they had to search for the music again on their music platform to save it. However, Spotify API provides the function to save the song, create a new playlist or add the song to the previously created playlist. This function is very important for this project because it applies the 7th heuristic of usability which is flexibility and efficiency of use. Therefore, we will surely use this function of the API to allow the users to have experience in a very efficient way.

Napster

Napster is more our general-use API. Overall, there are features we definitely want, but not all can be implemented using the Spotify API; this is where Napster steps in. Napster used to be very popular before for streaming music until platforms like Spotify took over. Considering how it was once used to distribute audio files, there’s no question as to why Napster has as much information as it does.

Feature 1: Demo clip of songs

The chorus of a song is what many people would consider to be its core or generally the most powerful section. The chorus is what the majority of listeners will get “stuck in their head” and is likely the most catchy. That being said, when finding a new song, the chorus is usually what listeners will pay the most attention to. In order to streamline this searching process, we will implement a function that plays ~30 seconds of the song (chorus) so users can determine whether or not they would like to listen to it more.

Feature 2: Retrieve a song based on data (e.g BPM)

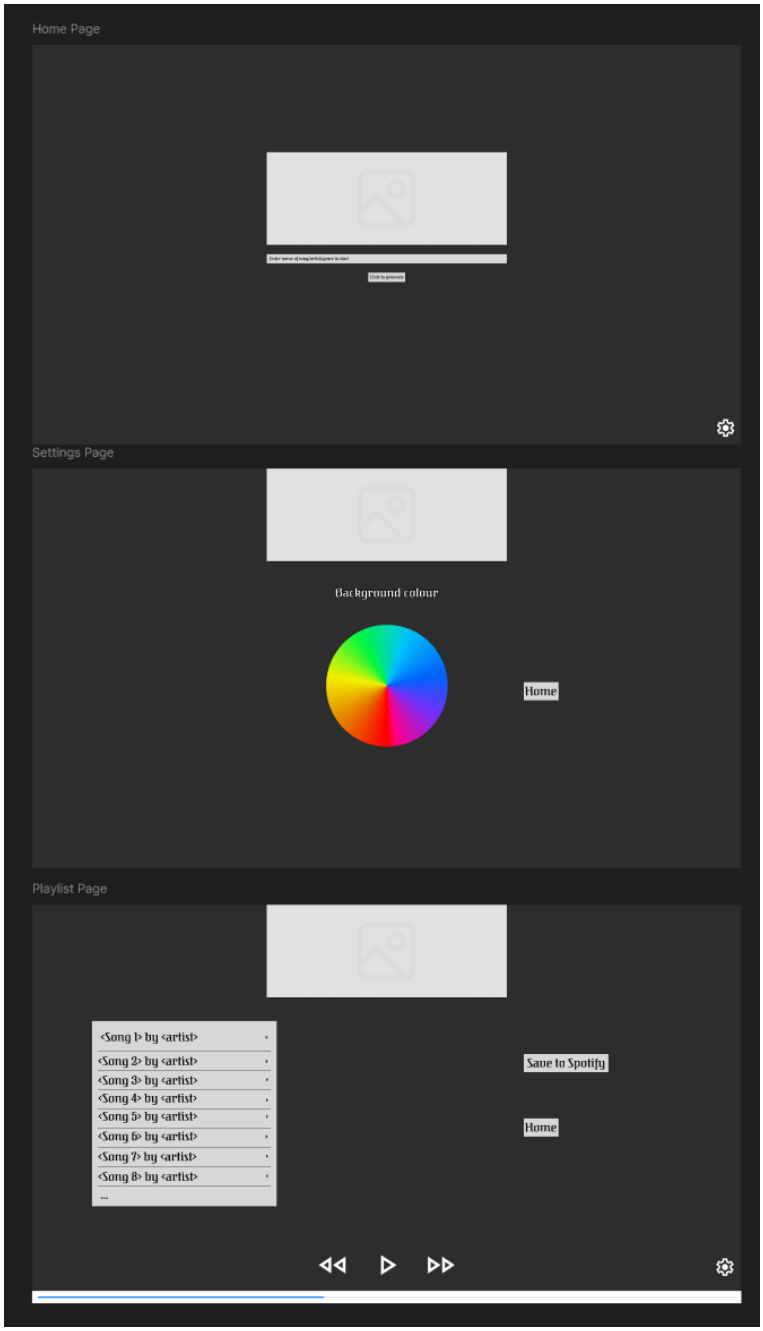
For users who have specific needs, only certain requirements meet said needs. Those who run, for example, are likely to want a playlist contrived of only songs at a certain tempo (speed), a tempo that matches the user’s running speed. With our specialized function to create a playlist based on criteria such as this, constant scrolling through album after album in an attempt to find the right tempo will be completely useless.

Feature 3: Suggest songs based on similarity

In order to actually create stylized playlists for users based on certain needs, similarities will need to be drawn between songs to determine whether two songs are fit to be placed in the same playlist. By integrating this feature into our playlist generation, we won’t just have playlists containing music from a single artist. Each genre of music has several sub-genres, which all have their own subcategories. By having playlists with songs from different artists, almost all artists having their own unique sound or style of music, users can explore more niche categories of music embedded into their initial request of genre.

Prototype - Murat

https://www.figma.com/proto/TMnmfmJDZzxeqXG7xwDYfv/Untitled?node-id=15-15&node-type=frame&t=FcljEKxQrnV2vSgz-0&scaling=contain&content-scaling=fixed&page-id=0%3A1



SDLC Mode (Kanban) - Roi

For this project, we will be using Kanban as our Systems development life cycle. The reason why we chose this model is because;

1. The Kanban board has significance in visualizing the process.
    - Easily see who is working on what
    - Which task has problems/issues
    - Which tasks are finished

2. Efficiency and Productivity
    - Focuses on continuous delivery
    - Limiting the work-in-progress elements reduces the effect of bottlenecks, teams are more productive

3. Problem-solving and Risk Management
    - Kanban allows contributions from all the members by allowing them to be able to make suggestions for further improvements, discuss issues and risks
    - Share responsibility for work

4. Collaboration and Experimentaiton
    - Kanban's significant visualization allows the team members to know what is happening.
    - It promotes changes that are incremental and increases the team's motivation

WBS - Brandon

Create the Music Generator

1. Create UI
    a. Determine how many pages will be needed
        i. Realistically one for each function
    b. Determine what each page will look like
        i. Take the previously designed storyboard and turn each sketch into a wireframe
            1. Take each wireframe and turn it into a prototype
        ii. Turn what will be the page from a physical drawing into a digital design
    c. Create each page with buttons, drop-down menus, scroll bars etc.
    d. Link pages together using buttons like “Generate Playlist” which will take the user to the page containing the generated playlist
2. Create the interactions between APIs and user
    a. Take user input and use it to filter songs
    b. Return a list of songs appropriate for settings set by user
        i. Test API correctly returns a list of songs fitting the genre given by user
    c. Use Spotify API to create the playlist
        i. Iteratively go through the list of songs and place them into a new playlist with a fitting title
    d. Return playlist to the user through screen output
3. Test that same input returns a playlist with different songs
    a. Plug in the same input multiple times – “Jazz” for example – and assert the two lists aren’t the same in terms of content
4. Implement API/User interactions into the website
    a. Text box gets user input
    b. User input used as filter
    c. Drop-down menu shows playlist

Project Schedule - Greg

Week 1 (Deadline: Nov 6)

- **API 1 Integration**: Initial setup and basic integration of API 1.
- **Interface Design**: Complete wireframe and initial layout design for the app interface.

Week 2 (Deadline: Nov 13)

- **Finish UI design:** Webpage layout should be completed with placeholders for incomplete features
- **API 1 Feature Completion**: Full functionality of API 1 completed, tested, and integrated with the app.
- **Interface Development**: Create an interactive interface for API 1 features.
- **Testing Setup**: Automated tests created for API 1 features.

Week 3 (Deadline: Nov 15) – Midpoint Check-In with TA

- **Check-In**: Present completed API 1 integration, interactive interface and tests.
- **API 2 Planning**: Finalize specifications, data structure, and details for API 2.

Week 4 (Deadline: Nov 20)

- **API 2 Integration**: Full integration and functionality of API 2.
- **Comprehensive Testing**: Automated and end-to-end tests for both API 1 and API 2.
- **Interface Refinement**: Finalize the interface for both API features.

Final Week (Deadline: Nov 27)

- **Final Debugging**: Resolve any outstanding issues and conduct final testing.
- **Documentation**: Complete user and technical documentation.
- **Deployment**: Ensure the app is ready for deployment and submission.

Risk Management - Everyone

Low-Risk

1. Group member(s) getting sick and being unable to do work.
    a. Low-risk because people don't get sick often and our group is already organized enough that if one or more members do get sick, the rest of the group will be able to recover pretty quickly
    b. This would cause everyone else to do their designated work on top of what we have already assigned for ourselves.
    c. Mitigation: Work will begin much earlier than it needs to be.
        i. If someone does get sick, their remaining work will not be as large is it would normally be
        ii. Everyone else will have less work to do before helping out leaving us enough time to create a well-done project
2. The group generally not being able to implement 1 or more of the features
    a. Low-risk because all members are well-versed in programming and should none of us be able to implement a feature, ChatGPT is an appropriate and intelligent tool we can use
    b. This issue would cause us to have to pivot and choose another feature to implement
    c. It could set us back by a fair amount of time as we would have likely already been trying to implement the feature for some time before giving up
    d. Additionally, we would have to think of a new feature and implement it from scratch taking up even more time
    e. Mitigation: Begin the more complicated tasks, like implementing more difficult features first
        i. If a feature cannot be implemented, we'll have more time to recover
        ii. Set our sights lower to begin: the feature doesn't need to be the best in the world, just something that efficiently serves its purpose.
3. One of our APIs shut down
    a. Low-risk because both Spotify and Napster have such huge reputations and have been around for so long that the possibility they get shut down for any reason is low
    b. Should either API shut down, our group would have to either find a replacement with features similar enough to Napster or Spotify to implement our current plans or find a new API and new features with little or no overlap
    c. Mitigate: Have a backup API that can implement the same or similar features that Napster and/or Spotify can
4. Member's lack of skill/knowledge
    a. Low-risk because it is very simple for a person to take a different role when it comes to a lack of skill in one of the criteria. If a member announces that they are incapable of certain things, they can do a different part that they are capable of.
    b. This has almost no impact on the project unless there is nothing else to do for that member other than the problem from before.

    c. Mitigate: Each members only take what they are capable of or be assigned according to their capabilities
5. Too many team members for available programming tasks
    a. This can lead to uneven distribution of work and lack of contribution from some members
    b. Mitigation: Use Agile development framework (Kanban) so members can clearly see which tasks are left and can pick them up as needed
    c. Create a project header with all the function declarations so the inputs and outputs are known ahead of time
    d. Assign different tasks such as UI development, API implementation, and testing

Medium-Risk

1. Scope Creep
    a. Mid-risk because although this rigid structure as to how the project will be handled is fairly detailed, certain features like finding related songs or songs with certain qualities can be difficult to do. For example, what defines related songs other than being in the same genre of music? How can one define whether a song is in a genre of music and how can that be applied to a computer algorithm without creating/adding to a backend data structure?
    b. This would cause us to either implement a completely different feature or reduce the reach of the initial
    c. This is less of a future risk and more of a current risk: our features may already be too ambitious.
    d. Mitigation: Before implementing, see what information is held about songs in our APIs (Spotify and Napster) and see how said information can be applied to each complicated feature
        i. Additionally, we can use AI systems like ChatGPT to see if algorithms for finding such specifications exist before attempting to implement them so we don't end up wasting time
2. Lack of consensus regarding project's final form
    a. Lack of consensus can occur when team members have differing visions for the project, priorities, or preferred approaches to implementation.
    b. This can lead to misunderstandings, inefficient workflows, duplicated or conflicting efforts, and ultimately a product that does not meet initial goals or satisfy all team members.
    c. This can be mitigated by ensuring that all team members take an active role in project development
    d. Another way to mitigate this is to discuss core functionality during weekly meetings so all members have input

3. Possible limitations on the Spotify API without user subscription
    a. We might run into an issue where we're not able to implement one or more features using the free API.

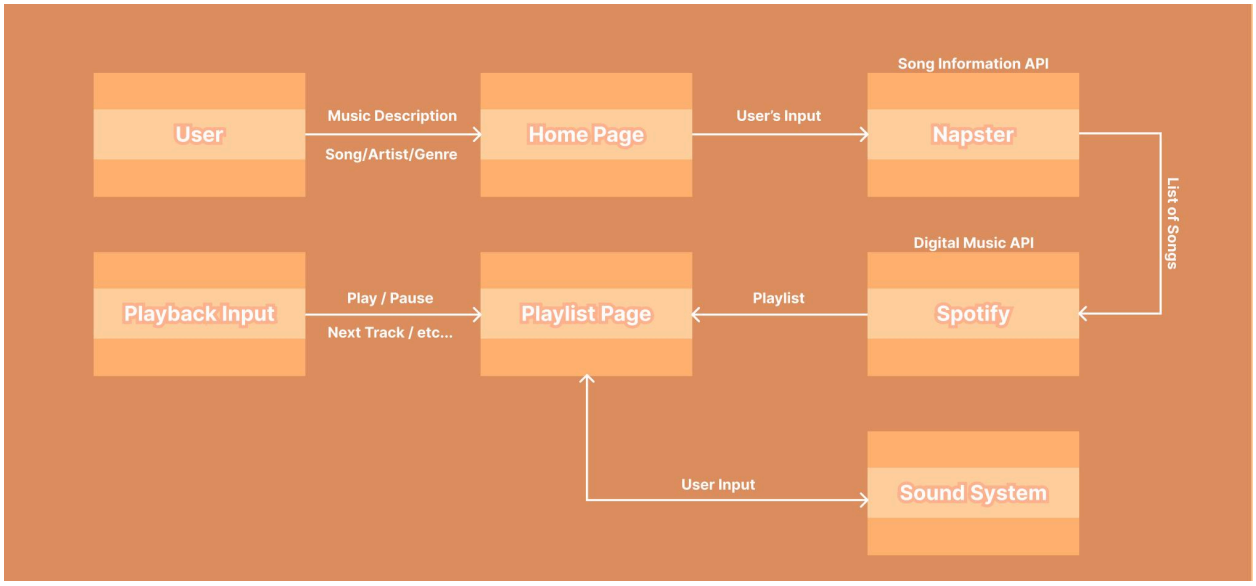
b. Mitigation: Implement and test API features early to ensure free API is sufficient for our project
c. Run the API using one of the group's Spotify subscriptions if necessary

High-Risk

1. Communication problems
    a. When it comes to a group project, communication is very significant and when there is a problem, it is considered a High-Risk for it is possible for the entire project to become a failure due to it. For example, if 2 members do not communicate properly and make 2 different parts of the project completely different to the point that they cannot be used together. This would mean that at least one or sometimes both of those parts must be remade.
    b. The impact of this risk could vary depending on the situation but it could scale from a very small adjustment to a redo of the entire project.
    c. This is a risk overall regardless of whether it is current or future. Miscommunication can create a problem from a very small part to a very significant part.
    d. Mitigation: Everyone needs to talk to each other and communicate before doing anything. Although the problem of miscommunication is something that is certainly going to happen at least once in the process of the project, this way can reduce the impact to a minimum.

2. Performance problems
    a. When the project needs to be finished, however, if the performance is significantly worse than what we originally designed, it is a High-Risk for it is not possible for the team to work on it anymore and must be finished as an incomplete project. If the deadline is due the very next day but the implementation of some parts is not finished, there is no possible way to achieve the goal anymore, making the whole project incomplete work.
    b. The impact of this risk is very large, it is something that cannot be fixed anymore and there is no other possible way to resolve this issue.
    c. This is more of a future risk for it is only a possible case nearing the end of the project deadline.
    d. Mitigation: Every team member must be aware of what they are working on and must follow/keep up with the milestone timeline. If there are any delays or miscommunication, these could also cause problems in the performance of the final product.

Data Flow Diagram - Brandon/Roi



MVC Model - Roi