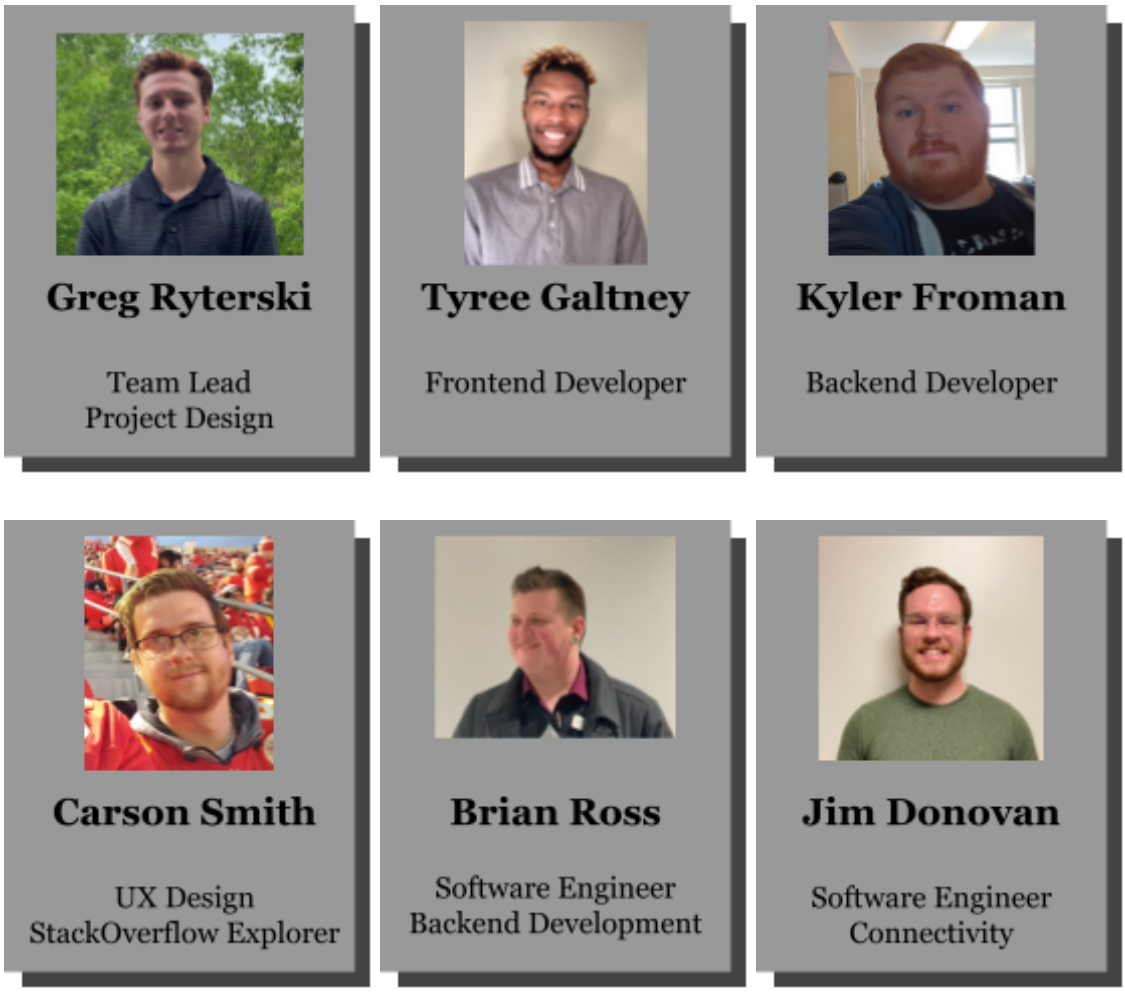
MusicBit: Final Project Doc

By: Jim Donovan, Kyler Froman, Tyree Galtney,

Brian Ross, Carson Smith, Greg Ryterski

Team Information:

- **Mission Statement**
  - To match the music in your heart to the music in your ears
- **Team/Project Name:** MusicBit



Mentor

The group does not currently have a mentor. The group brainstormed and came up with a few names to ask. These professors were: Dale Musser, Ronny Bazan Antequera, and Aaron Scantlin. These could be good options, but the group is not at a final conclusion. The group has reached out to Dr. Musser and is currently waiting to hear back from him.

Introduction

- **Project Statement**
    - People currently spend too much time trying to figure out a song to match their mood. So, we aim to create an application that can match your music to your mood. This would streamline the user experience by generating a song recommendation based on the current heartbeat. MusicBit also aims to let the user choose the desired heart rate that they want to go to. This will help with exercise, quiet contemplation, and everything in between.
- **Problem Resolution**
    - To provide a song based on the heartbeat of the user by utilizing Fitbit's BPM monitoring and Spotify's API.

Benefits of Solution

- By being able to match the user's current heart rate with a suggested song at the BPM provides many benefits. One of those includes keeping the user focused on the activity that they're doing. For example, with working out most people tend to get distracted and stare at their phones because they don't have something telling them to keep going. With MusicBit the application will suggest workout songs to the user by providing them with a desired heart rate range bar. This will help solve the issue of unfocused people at the gym because music BPM translates toward a person's current heart rate and motivation.

Requirements

- **Hardware**
    - AWS hosted website
    - Fitbit Versa 2
    - Computer with an open network for pairing the devices
        - Needs to be able to run chromium-browser
- **Languages to look into**
    - Java & XML
    - Python
    - C#
    - Javascript
    - Typescript (Angular & Ionic)
- **Non-User/Non-Functional Requirements**
    - Have app recognize connected heartbeat sensor
    - Receive suggested song based upon current heartbeat
    - Have the ability to choose if the desired song is a good suggestion
    - Mode to match heart rate and mode to push the user toward the desired heart rate with a given song choice
    - Allow users to define preferred genres
        - Request and integrate Spotify-curated recommendations
- **User/Functional Requirements**
    - Dashboard for information
    - Current heart rate
    - Desired heart rate
    - Slide bar to put move the desired heart rate
    - Login for Spotify token
    - Music player

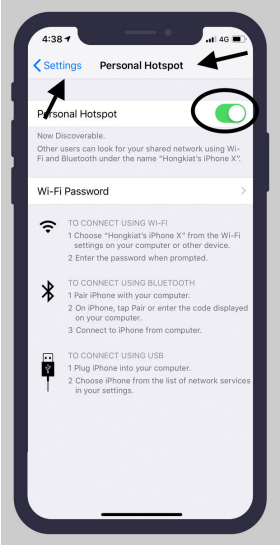
Methods

The following section pertains to the methodologies the group plans to use in order to accomplish the goal of recommending users' songs based on telemetric data. In order to do this, the group will first need to design a system by which the system can receive said telemetric data from a device made to track and report this data. The group will then need to design a system by which the system can get song recommendations from a 3rd party using API calls. Next, the group will purge the list of generated songs aforementioned based on the telemetric data by filtering out the farthest songs from the target bpm. The group will then make a system that passes this purged song list back to a 3rd party web-based song player embedded in the web application.

Hardware

Fitbit Versa 2

- Weight: 40 grams
- Case Material: Aluminum
- Bluetooth v4.0
- Wifi capability: Wi-Fi 802.11 b/g/n
- 5ATM water-resistant
- 4GB ROM
- Regular: 5 days, Standby: 8 days+
- Battery Type: non-removable Lithium-polymer
- Charging Type: Charging dock
- Charging Time: 2hrs



or

Phone or Computer with an open network (hotspot)

Phone/Computer Minimal Specs

- Wifi capability
- Bluetooth compatible

Connectivity:

The Fitbit & phone/computer need to both be connected to the same local internet source. This allows there to be communication between the two devices and enables the system to grab the API calls and transform them into a song recommendation. Expect an open network in order for communication to happen.

How we plan on doing this

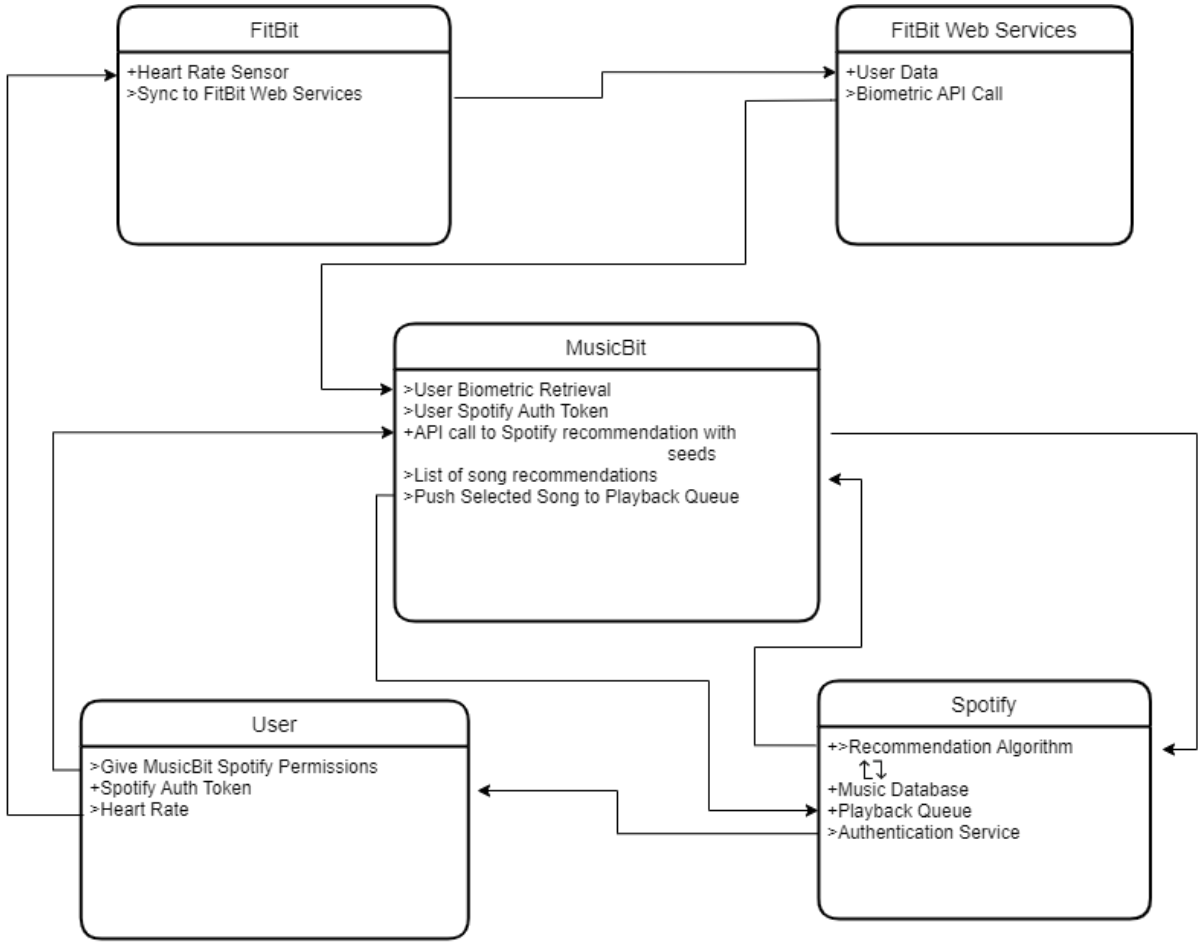


Figure 1: System Architecture Diagram, showing interaction between components of system

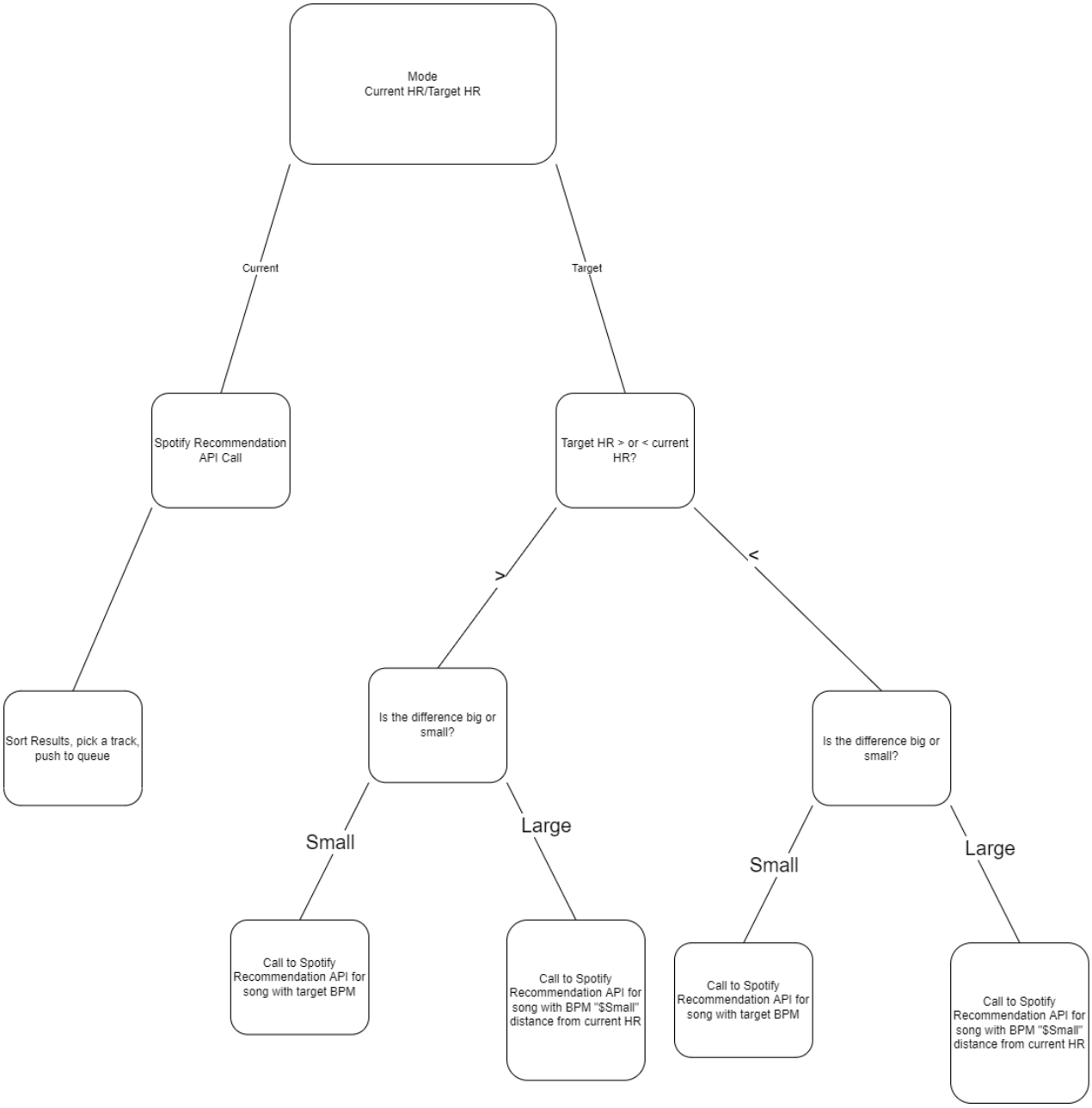


Figure 2: System Decisional Flowchart

As displayed in the above diagrams the plan is as follows:

1. Acquire a token to the user's Spotify account to allow for personalized song selection via Spotify’s API
2. Have the user select the mode of the application either to match the current heart rate or aim for a target heart rate.
3. Make a call to the API associated with the users connected Fitbit device to acquire heart rate
4. Get a list of songs recommended to the user by Spotify via their token acquired in step 1 and the Spotify developer API and save these in a list.
5. Calculate target BPM based on both the heart rate found in step 3 and the mode selected in step 2
6. Use target BPM calculated in step 5 to purge the list of songs acquired in step 4 by removing songs with the farthest BPM from our target.
7. Add a purged list of songs from step 6 to the queue to be played by Spotify via their API.

API Endpoints

Get Heart Rate from FitBit

By utilizing FitBit's developer account, their Intraday series API is able to be called. Within the Intraday API call, the data is returned in a JSON file containing the date of the heart rate log, the number of calories burned, the resting heart rate, the time the heart rate was recorded, the value of the heart rate at that time and the interval of how often the dataset is updated upon the request [2]. The data that is specifically important is the heart rate at each interval.

Get Track's Audio Analysis from Spotify

One endpoint of Spotify's API call is track audio analysis. This API call returns a JSON response containing a multitude of data regarding the duration of the track, overall loudness of the track, tempo, what key the track is in, and more [3]. The main response that will benefit the project would be the tempo response. It returns a float of the estimated tempo of the track in beats per minute. This will help suggest songs based on the user's heartbeats per minute and the track tempo.

Get Recommendations from Spotify

Endpoint of Spotify's API call is to get recommendations. These recommendations are generated based on the parameters passed and respond with a list of tracks that fit those parameters. It returns a JSON object that contains tracks, artists, id for the artist, the uri for the track (which is the id for it), the url for the track, and more [4]. This endpoint is needed in order to apply the user's heart beats per minute to a track that has a tempo matching that. After getting the response, filtering the tracks to get the best fit will be the next step, followed by adding it to the queue.

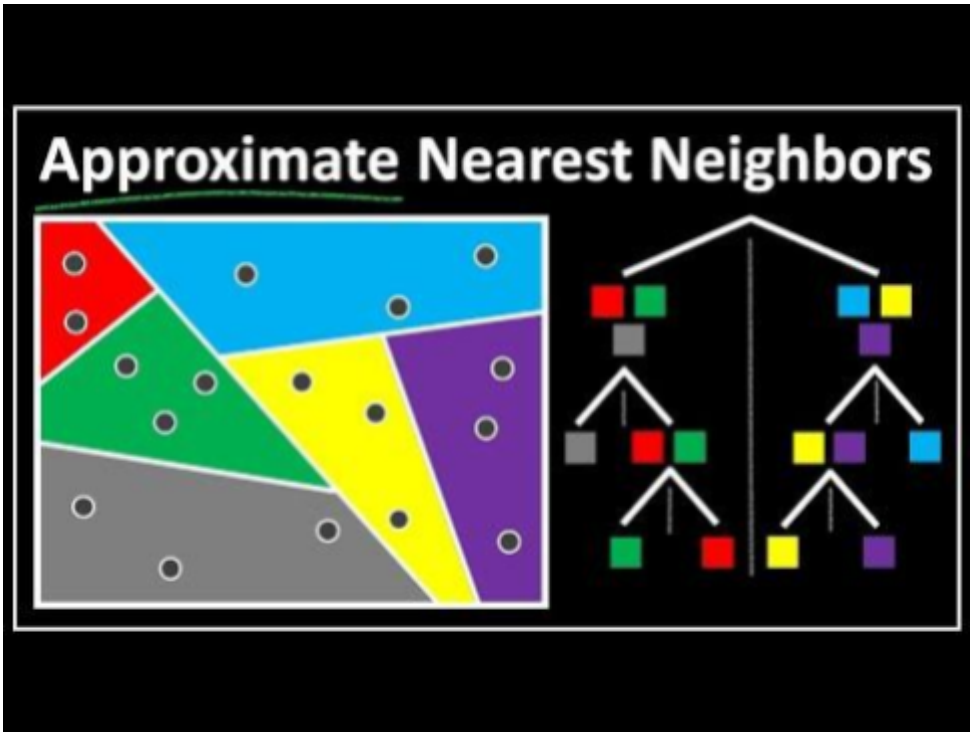
Add Item to Playback Queue from Spotify

Another endpoint of Spotify's API call is adding an item to the playback queue. This is a call to update the user's queue. This is going to be needed after getting the user's heart beats per minute and a specific track tempo (Get Recommendations Endpoint).After getting that data, the next step is to add the song to the queue. In order to do so, the API call does need the track uri which is in short the ID to a song.

Algorithms

Approximate Nearest-Neighbor

Spotify currently uses the approximate nearest-neighbor search algorithm when suggesting recommended songs. The group plans to use Spotify's algorithms rather than creating a new one for recommending songs to the user. With this, the Approximate Nearest-Neighbor will be used (by Spotify) to help provide users with songs that match the BPM and are previously 'liked' songs from the user's previous listening sessions. Below is a youtube video that does a good job of explaining the algorithm.



Double click the image to open the drawing. Then click the image in the drawing to play the video. Here's the link as well: https://www.youtube.com/watch?v=DRbjpuqOsjk

Also, Spotify's recommendation takes into consideration how long a user listens to a song. If a user listens to a song for more than 30 seconds then the sound is counted as positive feedback for the recommendation and if the user skips it before 30 seconds it's counted negative [1]. This is performed in addition to the approximate nearest neighbors algorithm which helps provide a different branch of songs if the user didn't enjoy the suggested one.

Website Prototype

The following is a mock up of what the MusicBit website would look like and the features it would include. Firstly the application starts with a login for the users spotify, this is to get the users token for use with the API. Next there is the application's main screen. Here there's a bar at the top to drag to find songs at that BPM. It would only suggest the 'Desired BPM' if the 'Live' was not on and turned red. While 'Live' is toggled on the user can click the big circle in the middle that would suggest a song based on the user's 'Current BPM.' In order to change the mode for the 'Live' display all the user would simply need to do is tap the 'Toggle' button located in the middle left of the screen. Lastly there is also a button the would lead to a stretch goal of the groups this being a live graph tracking a comparison of the users heart rate and the bpm of the songs suggested by the application over time. The group is looking to create this web application with the use of Angular though this will require some members of the group to learn how to use angular from little to no experience. The group plans to implement the backend in java due to familiarity.

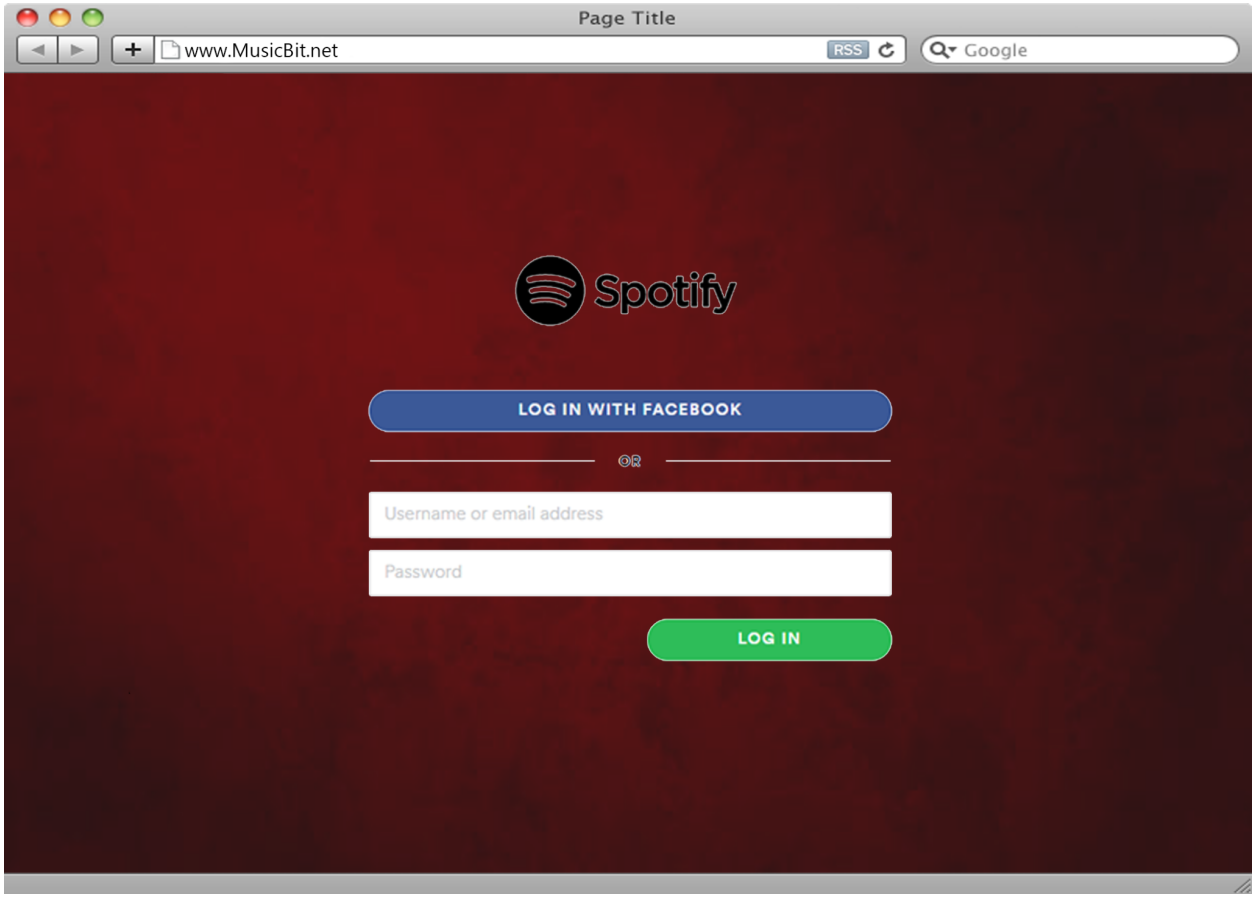


Figure 3: Login protoype for MusicBit

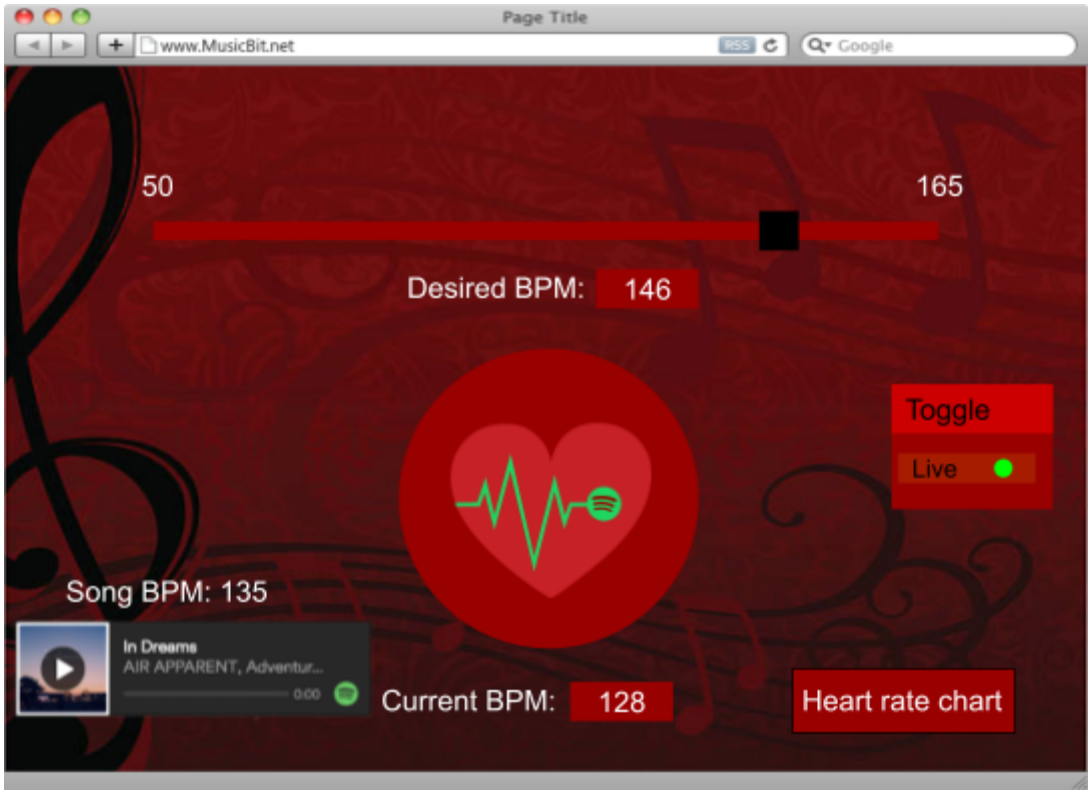


Figure 4: Home screen prototype for application

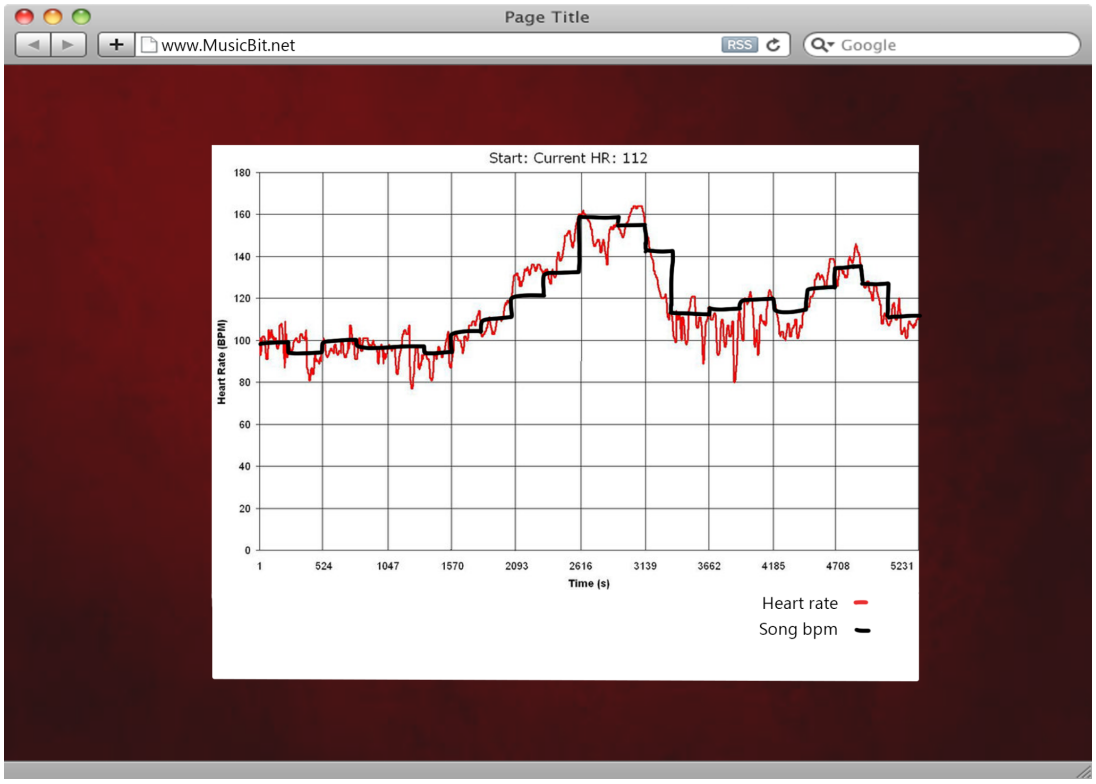


Figure 5: Heart rate history chart prototype

Research Documentation

Introduction

There are only a few aspects of life that beats the feeling of listening to music. Music helps to calm someone down or it lifts them up and provides them with more energy. This is where our group, MusicBit, steps in and creates a new way of achieving this idea. Our idea is to retrieve data from a FitBit or similar device, more precisely the heart beats per minute, to suggest songs to listen to. There are many small details that need to be figured out including which device to use, the benefits of listening to music matching the heart beats per minute, and song suggestion algorithms that are usable for this idea.

Topic 1: Retrieving Data from Heart Rate Sensors

Researchers: Jim Donovan, Tyree Galtney

One of the main concepts that needs to be implemented is getting the current heart rate from the user. When researching which device to continue the project with, documentation and accessibility to the data were key points. The question remained, which device has the documentation for getting the current heart beats per minute and does the device fit our budget?

After researching which brand of device to continue the project with, the group has elected to use FitBit. Compared to other brands of heart beats per minute measuring devices, FitBit and its API made for an easy choice. This choice took much research and digging through the FitBit API. One alternative to FitBit that was explored was AmazFit. The price for a new AmazFit device could be as low as $59.99 while the lowest price of a new FitBit device is $79.99. In this way, AmazFit certainly looks more attractive. Other than FitBit and AmazFit, other options included Apple Watch or Samsung Watch. Apple Watch could be as low as $159.99 for refurbished Series 3 and $229.99 for a new one. A new Samsung Watch can be found for $199.99 on sale. The price points for both the Apple Watch and the Samsung Watch push the boundaries for the budget allowance, thus pushing them out of contention. Obtaining the data that AmazFit and FitBit measures was the next step.

This step required significant discussions and research to understand the data. It was found that AmazFit does not have any official API that can be called on to obtain data. A work around could be to store the data in GoogleFit, then access the GoogleFit API. This strategy would work, but has too many steps, especially when compared to FitBit. FitBit has a direct API that can retrieve the heart beats per minute as well as other data points for a specific device. The API calls to get this data require the device in question to be paired with a developer account. The FitBit API contains an endpoint called “Heart Rate Time Series”. After initially reviewing this API it was unclear how to obtain real time data, as the data in this endpoint

seemed to be mainly historic with very little granularity. This API returned data about activities including resting heart rate, but no mention of the real time heart rate data that was desired [6]. The group was at a loss and tried thinking of ways to force this endpoint to work. The group was brainstorming methods to work around this issue, however, after some more digging, found exactly what was needed. Another FitBit API endpoint that includes Heart Rate is the Intraday API endpoints that return Heart-Rate data for the current day with a high level of granularity and detail [5]. The endpoint can return data from the previous few minutes or seconds if we desired. This API is a normal REST API communicating with JSON objects. These Intraday endpoints are the exact endpoints that are needed to create our application, meaning FitBit is a clear winner based on the API alone. The device will be a little pricier for the project, however, this cost will be worth it as we will be able to use the well documented API that FitBit provides.

	FitBit	**AmazFit**
Price	New device as low as $79.99	New device as low as $59.99
API	Has official API and documentation	Does not have official API documentation but can be added to GoogleFit

Topic 2: The Benefits of Matching BPM to Heart Rate

Researchers: Kyler Froman and Greg Ryterski

Why would anyone ever want or need a song to match their heart rate? Well, there are many advantages of listening to music that matches your current heart rate or having songs push to you a certain heart rate. Listening to music can provide many perks to the user such as making them feel relaxed, energized, happy, creative, etc.

Kesha said it best, “I hear your heart beat to the beat of the drums.” MusicBit aspires to do just that. Die Young is one of the many songs that puts any listener into that happy throwback mood. Maybe it’s because of the memories it brings back or it could even be the tempo as to which it’s played. Die Young has a measured tempo, BPM, of 128 which is similar to many of those amazing throwback songs such as Good Feeling (Flo Rida) and We Found Love (Rihanna, Calvin Harris) [10]. Clearly, there’s a correlation between the song and putting people into an energetic mood, and in fact, studies have proven that listening to music can affect heart rate, increasing or decreasing it depending on the BPM [8]. This is beneficial to any user as it will help put the listener into a mood or keep them in a desired frame of mind.

People typically prefer to listen to music that can match their heart rate, and music is made so this can happen. Almost every song falls into the same range as our heart rate, 60 BPM on the low side and up to a maximum of around 200 BPM [7]. The maximum is set both because few songs go about 200 BPM and because a heart rate that high would risk a heart attack. This is helpful for MusicBit because it can provide songs across the massive spectrum of sound and tailor the song choice for nearly any occasion. It seems that around 120 BPM is a sort of the ‘magic number’, just above the maximum resting rate and a great BPM for walking pace, clapping rhythm, etc. [7] [9]. This is a good thing to know because it can put the listeners into a productive mood that will get people up off the couch.

Most songwriters know this fact too, that's why the majority of songs fall at or near the 120 BPM range [7]. It makes sense that most songs are at this active BPM because in most cases when a person is active they're happy and would listen to the song again.

While there are good songs, there are also songs to get the user working out. Workout songs are incredible because they help listeners keep a good pace and push them through the workout. In contrast to the feel-good music, exercise music falls around 160 BPM or higher [9].

Not only does this raise the listener's heart rate, but listening to music at this BPM almost forces the listener to be productive because of how fast the song's beat is playing. This is beneficial to the listener since it will support their drive during the workout. In an article about the psychology of effective workout music, doing activity in sync with music can help keep a steady pace, increasing efficiency [9]. Clearly, allowing MusicBit to keep you at workout tempo is beneficial because it will make the workout flow smoother and keep you focused instead of making you want to scroll through Instagram for 10 minutes between sets.

The opposite would also be true of course, listening to music closer to the 60 or 80 BPM mark would encourage the listener's heart rate to decrease [8], which could provide a nice cool down post-workout, or just help to relax stress.

MusicBit can and will provide many benefits to the listener and just one of those amazing features is the ability to listen to music that fits the environment.

Topic 3: Track Recommendation Algorithms

Researchers: Carson Smith, Brian Ross

Software engineers, more than those in most other fields, are the most likely to ask themselves, "Should I reinvent the wheel?" It isn't the worst question. In software there already exists the solution to most problems you might encounter. An application might require the implementation of a hash table, for example. The programmer could sit down and design their own implementation, handling all of the logic and error-handling, deciding how to handle collisions, etc. This gives them ownership and fine control of their entire application. Alternatively, there already exist strong, stable libraries in the world that support hash tables out of the box. The latter is faster and easier to implement, but the process behind its implementation is a black box. Either solution can be valid to a given problem. Here, this project is faced with the same issue. In the process of pushing songs to the users whose BPMs match their heart rate, should it utilize existing algorithms, or design a custom algorithm for this?

The group wanted to find the best way to provide the user with input as for what songs would be chosen, given that the songs matched the user's heart rate. This could be done in multiple ways, however, the group had narrowed it down to two methods due primarily to familiarity with Spotify vs other music suggesting applications from a user perspective and due to the extremely well documented API that Spotify offers developers compared to other music applications. The group decided that our best options were either through API calls to spotify, if possible, or through our own algorithm. A decision therefore must be made of which is better for the application.

The first solution is utilizing Spotify API calls to get recommendations and add songs to the playback queue. Doing this, the application could get a list of recommended songs directly from Spotify, off-loading the heavy lifting to their existing resources. It could simply look through

the recommendations for the song that closest matches the desired BPM. It can then push the selected song to Spotify's playback queue.

Pros	Cons
Simple, existing API	Less control over suggestions
Seed recommendation algorithm accepts up to 5 seeds (artists, tracks, genres)	Trusting Spotify not to attempt recommending the same song every other time
Look through returned recommendations and easily examine metadata of tracks	More difficult to recommend less popular tracks to users with more niche tastes
API call allows restriction of min and max_tempo right out of the box	
Allow user to select their own seeds, or populate them from a single given track	
Reduced time and difficulty of implementation	

Essentially, implementation of Spotify's existing algorithm should be relatively trivial. Their existing API allows for a lot of flexibility in custom tailoring recommendations[11]. Control is given over seed values, tempo of returned songs, even the length of recommendations we would have to sort through. The algorithm accepts an argument for "max_energy", which highlights the number of options available. Spotify returns all these tracks to us as an easily-parsable array of JSON objects,[11] which we can process, decide on a track, and push that track's ID or URI to their playback queue[12]. In return for this ease-of-use, we have to trust

Spotify's recommendation algorithm, which focuses on more popular music over more niche music.

The second solution is using Spotify's tag-based metadata to allow the user to select a genre from which recommended songs will be chosen that are closest to the target BPM, then pushing the closest match to the playback queue.

Pros	**Cons**
Fine control over song recommendation; Allow for more niche music recommendations than Spotify typically supports	Would require construction of a massive database of songs, and populating this with metadata
Spotify allows for easy examination of track metadata to allow our application to improve its recommendations	Require development of a new heuristic scoring algorithm, based on criteria Spotify has already spent years studying
	Makes the app responsible (in both truth and the user's mind) for bad song suggestions
	Reinventing the wheel

Developing a new recommendation algorithm would allow for greater control over the types of songs given to users. Spotify's API allows easy access to the metadata of the tracks on their platform, allowing the application to perform the necessary database queries to find the songs we're looking for based on the criteria provided. However, to recommend a song, the algorithm has to know about the song. This works well on Spotify's end, they have, presumably, a large database with every song in their catalog. This project does not, and Spotify does not

offer access to their raw database[14]. The application can't suggest an appropriately-tempoed math rock song without having access to all (or at least, a large selection of) math rock songs on the platform. Further, the application can't catalog a song without knowing it exists. The group would need to design an algorithm designed to scrape Spotify's entire catalog, then store that data separately, either centrally, requiring the development of a *new* web service and API for our application to work with, or building a copy of this massive database on each user's machine, which is not an acceptable solution. Further, the group would then need to periodically check for updates, i.e. new songs, removed songs, new genres, etc. All of this in a way Spotify does *not* support. While it could be cobbled together through a massive number of Spotify API calls, attempting to crawl through *every* possible category, it would be a massive undertaking to essentially duplicate something that already exists on the Spotify servers. Updates would be even worse, as without access to Spotify's database directly, the group can only rerun the scraping algorithm, inefficiently looking back over countless objects that have already been cataloged.

Weighing the pros and cons of each solution and understanding the roadblocks inherent in developing a new recommendation algorithm, it seems supremely more prudent to utilize Spotify's own existing recommendation algorithms over the development of a new one. The data cost, computation time, number of web API calls, and everything else listed above make the generation of our own music database impractical for the development of this application. While the fine control offered by a custom algorithm is tempting, Spotify offers a significant amount of customizability in its own API that renders most of that benefit insubstantial. As aforementioned the group chose to use Spotify rather than another music application due to two main reasons being familiarity and extensive documentation that other 3rd party options lacked.

Conclusion

MusicBit aims to fulfill all music lovers' desires. As the conclusion of the research was reached, the idea of the project being possible was made clear and so was the ability to create this product on a low budget and provide numerous benefits to users. The group looked into other topics as well including the need for a database. This topic did not need a full section since the group has not collected any information that needs to be stored as all the current information will be deleted after the session. With this the team has decided that a database isn’t necessary because no information is being kept. All major information has been covered by our research and if another topic needs to be looked into then the team will add it to this document.

Development Strategy

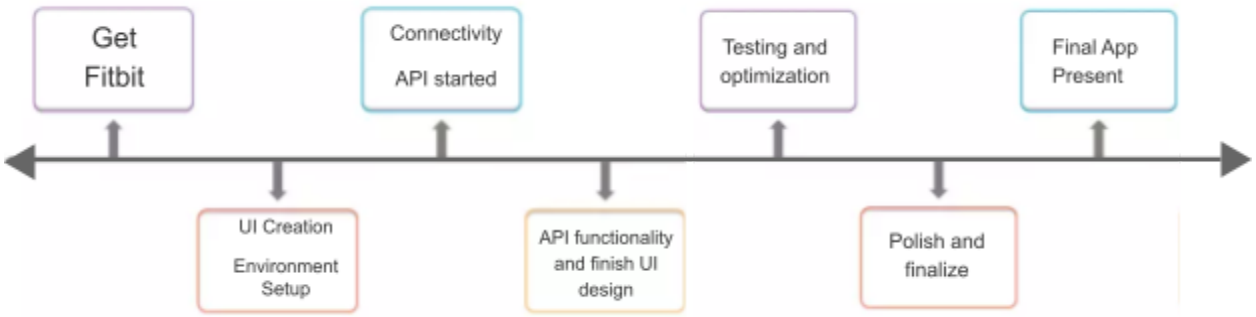
- **Waterfall:**
    - Advantages:
        - The cost and timeline of the project can be accurately estimated [15]
        - Easier to measure progress since milestones are defined
        - Keeps focus on end goal at all times and will eliminate risk of getting sidetrack or bogged down [16]
    - Disadvantages:
        - Not flexible, difficult to navigate around unforeseen roadblocks
        - Projects can take longer than Agile method since this is a chronological approach
- **Agile:**
    - Advantages [17]:
        - Helps to ensure each milestone works as intended
        - Reduces bugs
        - Encourages frequent inspection and adaptation of work
    - Disadvantages [18]:
        - Fragmented output delivery
        - Less clear big picture
        - No fixed endpoint
- **Scrum:**
    - Advantages: [19]
        - Fast moving and efficient use of resources

        - Sprint based development that allows for higher granularity of product development
        - Constant feedback from the consumer allows for making sure the product is what the consumer wants.
    - Disadvantages: [20]
        - Scope creep
        - High chance of failure if individuals aren't committed
        - Challenging for small teams
- **Spiral:**
    - Advantages [21]:
        - Risk Handling
        - Good for large projects
        - Flexibility in Requirements
        - End-user satisfaction
    - Disadvantages [21]:
        - Complex and expensive
        - Too much dependability on Risk Analysis
        - Difficulty in time management

- **Group's Choice: Agile**
    - Always have a deliverable
    - Easy to break down project into milestones for the team to work on
    - Waterfall is not flexible, if roadblocks occur it will throw off the entire schedule whilst Agile will give more flexibility and situations can be adapted at an easier rate

Agile seems an easy pick based upon the presumed structure of Capstone II. The group assumes that the grades will include multiple check-ins where the group must prove some amount of progress between milestones. Agile best allows the group to have distinct deliverables throughout the course at these various milestones. The group would also benefit from the reduction of bug fixing time that agile provides as this lack of unexpected increases in development time would allow members to keep on time more consistently.

Development Timeline/Schedule



Above is an estimated timeline of how the group is deciding to tackle this project. It has been broken down to mainly backend applications until later on in the semester when we would then want a website to display user actions.

Agile Timeline

For the agile approach, the best fit for the group would be to meet twice a week during a total of five three week long sprints. For sprint one, the main focus will be UI and setting up server, sprint two would be API focus. Sprint three would be implementation of API calls and tidying UI. Sprints four through seven would be testing and any fixes. Additionally, the last sprints will focus on user experience and overall optimization.

Sprint 1: (Aug 22 - Sep 12)

- This sprint will focus on setting up the environment needed to create and run this application. This sprint will include obtaining the FitBit device, setting up the AWS server, and working on the basic elements of the UI.

Sprint 2: (Sep 12 - Oct 3)

- This sprint focuses on the connectivity that the application relies on. By the end of this sprint, the API's that the application uses should be set up and some data should be returned to the backend.

Sprint 3: (Oct 3 - Oct 24)

- This sprint will finalize API calls and complete additional UI work. At the end of this sprint all connectivity to external API's should be complete. This includes the Spotify API and the FitBit API. The UI should be functionally complete.

Sprint 4: (Oct 24 - Nov 14)

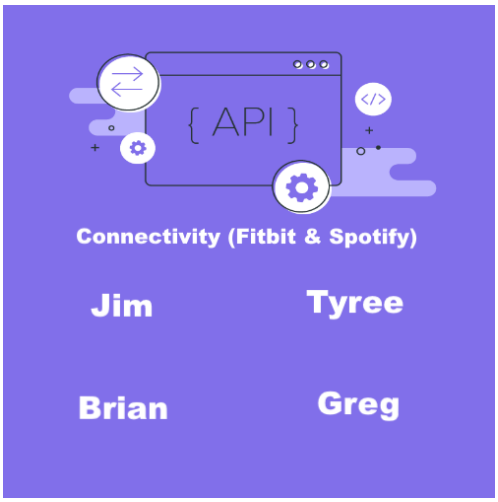
- This sprint will focus on testing and optimization. At the end of this sprint, the application should be running smoothly with complete functionality. This sprint will ensure product stability and reliability.

Sprint 5: (Nov 13 - Dec 9) (3.5 week sprint to include Reading Day)

- This sprint will focus on completing any backlogged work that the group has fallen behind on. At the end of this sprint, the application should be in its final form with no known errors and a good user experience. The application should be optimized for a fast and fluid user experience.

Work Delegation

Projects that need to be covered:



Specifics

- Greg: Design aspects of the frontend (Color ways and overall look of the document), helping create an open network to link the fitbit and spotify.
- Carson: Website creation and working on UI interactions. Helping incorporate Spotify's algorithms to the website.
- Kyler: Linking the website with the backend algorithms when features are used, managing AWS server.
- Tyree: Helping with website design making it look presentable and getting the features working. Also getting the Fitbit and Spotify to link with each other.
- Jim: Linking FitBit and spotify with the backend algorithms. Once connectivity has been properly established and tested, will help in whichever sub-group necessary.

- Brian: Helping all around with the backend and connectivity teams. Linking FitBit and Spotify with the backend algorithms.

Contingency Planning

- Spotify removes access to API
    - Find a different music player. Options include Apple Music, Youtube Music, etc.
- FitBit API fails to behave according to planned designs
    - The application asks the user for input of desired heart rate. This is less streamlined, however this would work if the team is unable to find a suitable work around.
- FitBit stops selling the device the group wants
    - Buy the lowest priced available FitBit device that supports the API
- AWS free tier gets downgraded or is insufficient for the project's goals
    - Use paid tier that satisfies the team's requirements
- One of the sub-teams gets behind schedule:
    - Additional help could be assigned to a subgroup if it is found to require more work than the others
- One of the 3rd party resources does not work as the group predicted.
    - The group has a strong understanding of each of the subsystems in the overall system well enough to find suitable replacement in the case that a 3rd party resource stops working.
- Learning the selected JavaScript Framework is too cumbersome for Agile development
    - Try a different framework
    - Find a mentor that specializes in the given framework

Testing Plan

Intentionally left blank in accordance with documentation instructions.

Works Cited

[1] Marius, Hucker. “Uncovering How the Spotify Algorithm Works.” *Medium*, Towards Data Science, 23 Nov. 2021, https://towardsdatascience.com/uncovering-how-the-spotify-algorithm-works-4d3c021ebc0

[2] “Get Heart Rate Intraday by Date.”*Fitbit.* https://dev.fitbit.com/build/reference/web-api/intraday/get-heartrate-intraday-by-date/

[3] “Get Track's Audio Analysis” *Spotify* *https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-analysis*

[4] “Get Recommendations” *Spotify* https://developer.spotify.com/documentation/web-api/reference/#/operations/get-recommendations

[5] “Get Heart Rate Intraday by Date.”*Fitbit.* https://dev.fitbit.com/build/reference/web-api/intraday/get-heartrate-intraday-by-date/

[6] “Get Heart Rate Time Series by Date” *Fitbit.* https://dev.fitbit.com/build/reference/web-api/heartrate-timeseries/get-heartrate-timeseries-by-date/

[7] Mitchum, Rob. “Groove Is in the Heart: Matching Beats per Minute to Heart Rate.” *Medium*, Spotify, 1 Nov. 2016, https://medium.com/@Spotify/groove-is-in-the-heart-matching-beats-per-minute-to-heart-rate-271a79b7f96a

[8] Agrawal, Ayush, et al. “The Effect of Music on Heart Rate - Emerging Investigators.” *Journal of Emerging Investigators*, Centerville High School, 25 Apr. 2013, https://emerginginvestigators.org/articles/the-effect-of-music-on-heart-rate/pdf

[9] Jabr, Ferris. “Let's Get Physical: The Psychology of Effective Workout Music.” *Scientific American*, Scientific American, 20 Mar. 2013, https://www.scientificamerican.com/article/psychology-workout-music/

[10] ”Recommedations for Harmonic Mixing” *Tunebat.* https://tunebat.com/Info/Die-Young-Kesha/6mnjcTmK8TewHfyOp3fC9C

[11] “Get Recommendations” *Spotify* https://developer.spotify.com/documentation/web-api/reference/#/operations/get-recommendations

[12] “Add Item to Playback Queue” *Spotify* https://developer.spotify.com/documentation/web-api/reference/#/operations/add-to-queue

[13] “Get Track's Audio Analysis” *Spotify* *https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-analysis*

[14] <Solbia01> "Database Access" Database access - The Spotify Community <Spotify Community Board><2019/24/03>

[15] “Waterfall Methodology.” *Waterfall Methodology - A Complete Guide | Adobe Workfront*, https://www.workfront.com/project-management/methodologies/waterfall#:~:text=The%20Waterfall%20methodology%E2%80%94also%20known,before%20the%20next%20phase%20begins.

[16] Lucid Content Team. “The Pros and Cons of Waterfall Methodology.” *Lucidchart Blog*, 10 Aug. 2018, https://www.lucidchart.com/blog/pros-and-cons-of-waterfall-methodology.

[17] “What Is Agile? - What Is Scrum? - Agile FAQ's.” *Cprime*, 17 Jan. 2022, https://www.cprime.com/resources/what-is-agile-what-is-scrum/#:~:text=Agile%20software%20development%20refers%20to%20a%20group%20of%20software%20development,%2Dorganizing%20cross%2Dfunctional%20teams.

[18] Lynn, Rachaelle. “Disadvantages of Agile.” *Planview*, Planview, 21 May 2021, https://www.planview.com/resources/articles/disadvantages-agile/.

[19] “Scrum (Software Development).” *GeeksforGeeks*, 7 June 2019, https://www.geeksforgeeks.org/scrum-software-development/.

[20] Chandana. “Scrum Project Management: Advantages and Disadvantages.” *Simplilearn.com*, Simplilearn, 12 Jan. 2022, https://www.simplilearn.com/scrum-project-management-article.

[21] “Software Engineering: Spiral Model.” *GeeksforGeeks*, 9 Feb. 2022, https://www.geeksforgeeks.org/software-engineering-spiral-model/.