Project P.E.P.

Spotify Playlist Enhancement Plan

Michelle Gordon
David Eccles School of Business
IS 4430-090

No part of this original project has been used for any other course or institution.

Table of Contents

Project Selection & Requirements Analysis

Project Evangelist

Michelle Gordon

BSIS student with intention to apply to co-terminal MSIS program at the University of Utah David Eccles School of Business.

Has worked in the private and public sector as a Jr. Business Analyst and Graphic Artist. Experience with Java, Python, HTML, CSS, RFP response building, Adobe InDesign and Adobe Dreamweaver.

Interests in software development, web development, Raspberry Pi, and UX design.

Executive Summary

Introduction:

Spotify is a pioneer of music streaming services with a large customer base. Customers have come to enjoy the ability to listen to any song, album, or playlist on demand and have the option to upgrade to Spotify's paid premium subscription for ad-free listening at a low monthly cost. Customers particularly enjoy the ability to create their own playlists, or follow playlists curated by Spotify.

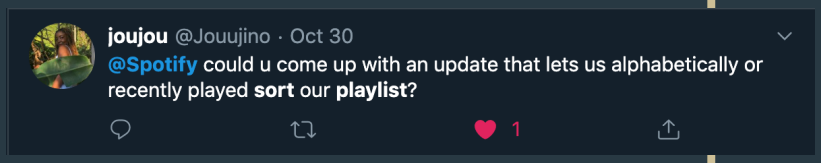
Playlists are incredibly valuable to our business. The number of playlists a user creates or follows has been shown to be positively correlated with their loyalty to Spotify. Without the ability to export playlists, moving to another streaming service becomes a cumbersome prospect for our customers. They would have to start over and create new playlists from scratch - one song at a time. Thus, playlists are a valuable asset to Spotify.

Problem

Currently, there is no functionality in our application that allows users to sort their playlists alphabetically when in their music library.

Pandora and other music streaming services provide the ability for their users to sort their lists in alphabetical order or by creation date. The fact that Spotify lacks these features gives users incentive to try other music platforms and erodes the loyalty of our customers.

As previously mentioned, an increase in number of playlists increases customer loyalty. However, it simultaneously increases the level of frustration for our customers as well. Once a user hits a certain number of playlists, trying to find the correct one to add a new song to quickly becomes an exercise in futility.



Solution

Project P.E.P. will build multiple features onto the playlist module of the Spotify app. Users subscribed to our premium service will have access to all of the new features. Free users will have access to a limited subset of features. Thus, this project will enhance Spotify's usability and utility for our users with the added bonus of providing additional incentives to subscribe to our premium service.

Project P.E.P. will be a module housed within the Spotify app that will interact with other modules of the app and the user music library database. The features provided by this solution would include the following modifications to users' playlists:

Premium:
-Add tags -Friend-based custom playlists -Sort by name or date
-Merge -Filter -Visual playlist identification
Free:
-Add tags -Friend-based custom playlists -Visual playlist identification

Outcomes

The outcome of implementing Project P.E.P. will appear as an increase in Spotify's overall value by increasing its competitiveness with other music streaming platforms while providing customers with a higher quality experience.

Spotify's competitors include Pandora, Google Play, and Slacker Radio. These companies all have free and paid customer tiers that have the following features:

- Ad-free listening
- Search and play any song
- Subscribe to podcasts

- Unlimited skips
- Offline listening
- Make and share playlists
- Custom radio stations for your mood or environment
- Ability to sort radio stations or playlists by name or date

Spotify already has all of these features built into its platform with the exception of being able to sort playlists by name or date. By adding competitive features along with new features not available in other music streaming platforms, Spotify will emerge as the most forward-thinking option. Increasing the usability of Spotify's music library as well as adding advanced features will produce more loyal customers, whether they are experienced or new to Spotify. It is known that advanced features can become a liability if they are frustrating and difficult to use. Therefore, Project P.E.P.'s focus on usability, quick results, and intuitive feel will augment the value of the new features.

The most tangible outcome will be an increase in paid subscriptions - projected at a 30% increase. Due to implementing most of the new features in the premium tier, users will be more enticed to subscribe to Spotify Premium. Furthermore, creating a custom Friend Discovery playlist for users adds a fun and social feel to the Spotify experience. In particular, it will create an incentive for users to convince their friends to join, so that they can all enjoy learning about each others' music tastes.

Summarized benefits from implementing Project P.E.P.:

- Projected 30% increase in subscribed users
- Increased customer satisfaction and loyalty
- Fewer delays in finding and playing music
- More advanced features than competitor apps
- Enriched customer experience
- Increased usability for both novice and experienced users

Primary Actors:

1 Subscribed Customers

These are customers that pay for our monthly premium membership. Subscribing to the premium membership provides the perk of unlimited ad-free listening in addition to the features already available to free users.

2 Free Customers

Free customers enjoy a wide range of features in exchange for ads being included in their listening experience. Features available include saving songs and playlists to their music library, curated playlist suggestions, and access to our podcasts.

Secondary Actors:

1 Spotify System

The rest of the Spotify system will be interacting with the playlist module to provide information about user music libraries, curated playlists, user history, user authentication, etc. as needed.

Requirements Gathering

Methods

I used a combination of two methods: discussion groups and surveys.

I decided to use Twitter to host the discussion group. I saw that a large subset of our users prefer to submit feedback to us via twitter, and so thought it would be a great opportunity to solicit requirements. Ultimately, I ended up collecting over five thousand tweets and extracted them from Twitter for analysis. The discussion group was setup through a prompt in the following tweet:

"Hey **#Spotify** Users! We want to hear from you! Tell us what kind of features you want to see integrated to your playlists using **#DearSpotify** You have 72 hours, go!"

The surveys were employed as a pop-up window when users had the Spotify app open. It was a short five-question survey designed to be filled out quickly. The survey is shown on the following page (Pg. 10) in its entirety.

Spotify Survey

Help us get a sense of opportunities improve our service!
This survey should only take 20 seconds to complete.

How long have you used Spotify?

- ○ 0 - I just signed up
- ○ 1-2 years
- ○ 3-4 years
- ○ 5-6 years
- ○ 7-8 years

What is your favorite thing about using Spotify?

Your answer

How many playlists do you currently have in your library?

Choose

What would you like to be able to do with your playlists?

Your answer

On a scale of 1-10, how much would adding that feature increase your enjoyment of the Spotify app?

	1	2	3	4	5	6	7	8	9	10	
Would barely make any difference	○	○	○	○	○	○	○	○	○	○	Would extremely increase enjoyment

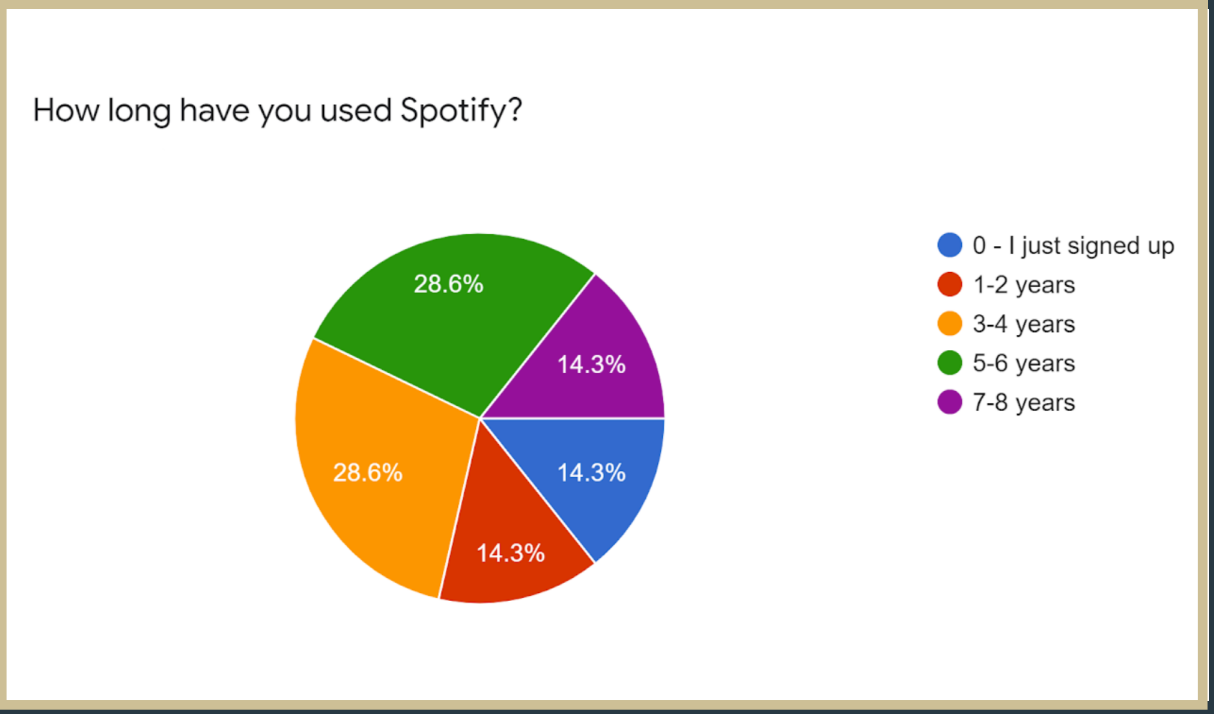
Submit

Findings & Priorities

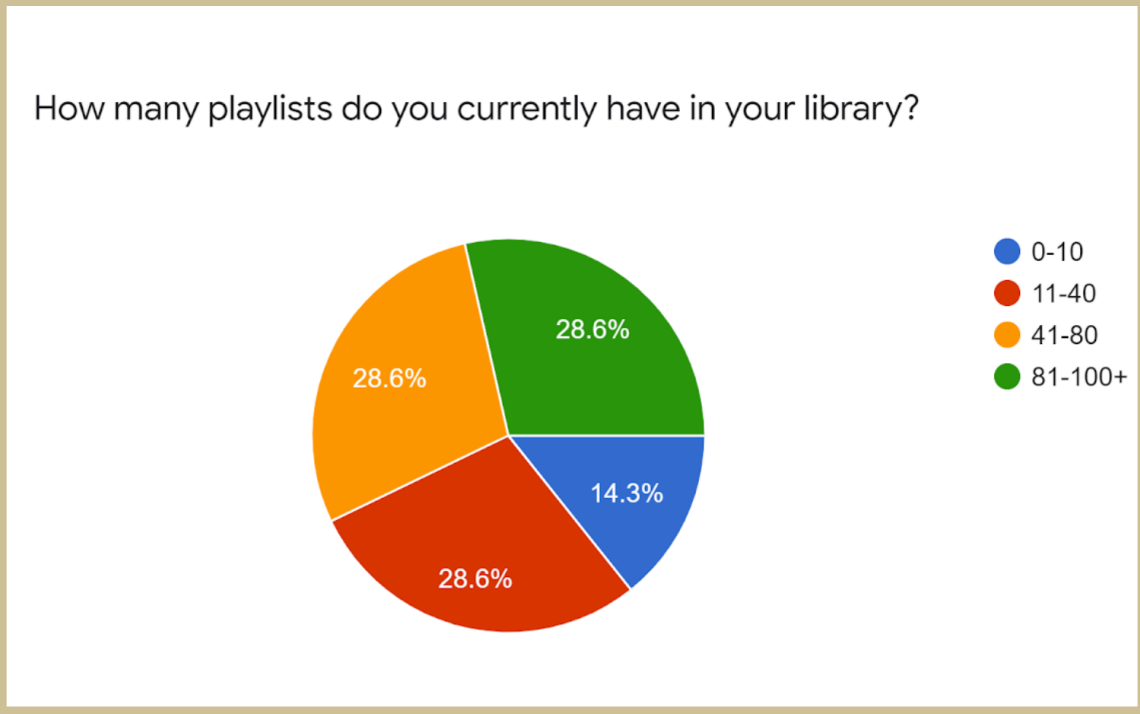
The findings from the survey were comparable to the info I received from the Twitter discussion group. In both cases, sorting the playlists alphabetically scored the highest with regard to what feature users wanted implemented most. From the surveys, I also learned that the length of time with Spotify seemed related to the number of playlists users have.

The survey information combined with the Twitter discussion information confirms that Project P.E.P. is focused in the right direction. The next step is to use the data collected to narrow down the requirements into a more focused set of deliverables. You will see later in this document, that the collected user stories have been pared down into a set of five "must-haves" that will be the emphasis of this project going forward. Additionally, these five items have been sorted into free or premium features.

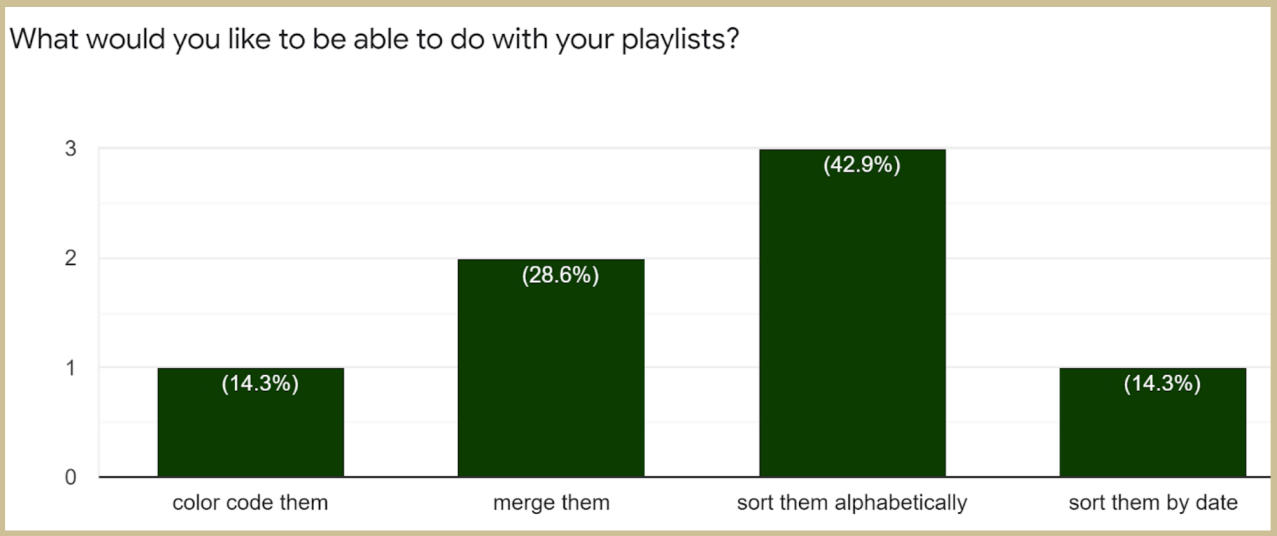
Length of Time With Spotify



Number of Playlists in Library



Most Wanted Feature

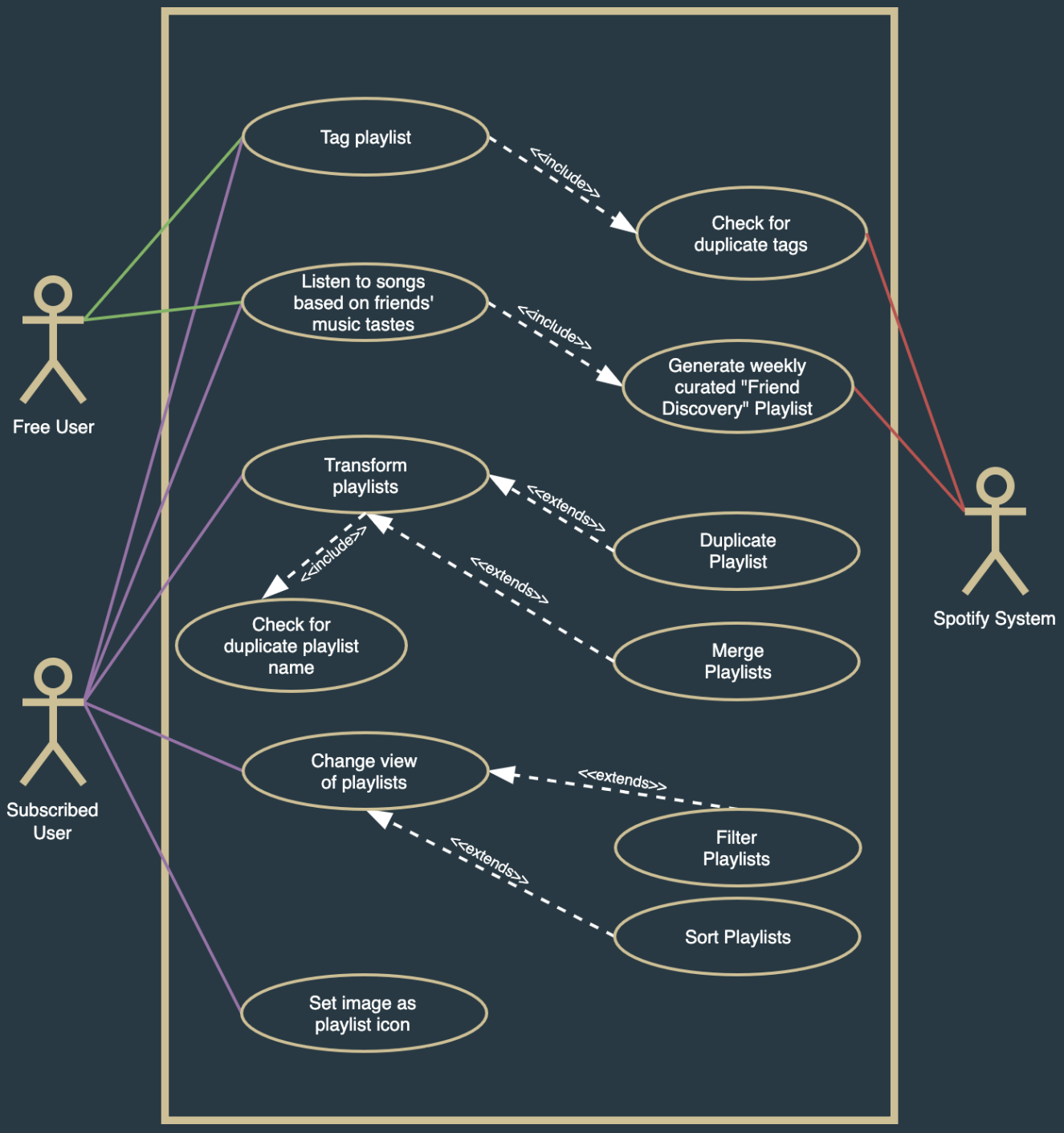


High-Level Scope Boundary

User Stories

As A/An	I want	So That	Acceptance Criteria	Functional or Non-Functional
User	to be able to change how my playlists are organized	it is easier for me to find the playlist I want to listen to	Add ability to sort playlists by name, date added, as well as ability to filter out spotify created playlists or user created playlists	F
User	the sorting to happen instantaneously	I don't have to wait to listen to my playlist	Sort must complete within 0.5 seconds	NF
User	Create another playlist based on the ones already in my library	I don't have to create one from scratch if I want to include songs I already have in my playlists	Add ability to duplicate and merge playlists.	F
User	To be able to visually identify my playlists by icon instead of text	I can easily find my playlists without having to use as much screen space	Add ability for user to upload an image for their playlist icon	F
User	It want it to be easy to create a good quality playlist icon	so that it doesn't look pixelated or distorted	add instructions in the upload screen that tells user of correct image size, resolution, and format	NF
User	get song suggestions based on my friends' music preferences	so that I can discover new songs to add to my playlists and learn what music my friends like	Analyze friends' music libraries and create weekly discover playlist for user based on friends' music tastes	F
User	to be able to categorize my playlists in multiple ways	so that I can easily search for them by moods, genres, etc	Box at the bottom of user playlist available for users to insert tags separated by commas	F

Use Case Diagram



Use Case Narratives

The following pages contain narratives that correspond to the Use Cases shown on the Use Case Diagram.

Use-Case name	Tag playlist
Last revised	11-11-19
Description	This use case describes the steps required to add category tags to their playlists for searching purposes.
Actors	Free user, Subscribed user
Pre-condition	The user must be logged in and have at least one playlist in their library.
Post-condition	The user's playlist now has category tags in its metadata to make searching easier.
Other business rules (if any)	

Success Path

1. The user clicks into the text box at the bottom of the screen to activate the cursor.
2. The user types tags for the playlist separated by commas, and in the following format: #tag_goes_here [Extends: Tag already exists.]
3. The tags are autosaved as they are typed.
4. Use case terminates.

Variations in success flows

user may also edit or delete any previously entered tags.

Alternate Paths (Extensions/Exceptions)

2a. Tag already exists

- 2a.1. System prompts user to re-enter new tag.
- 2a.2. If user enters unique tag, return to 3.
- 2a.3. If user chooses to cancel the operation, the use case terminates.

List related use-case names

Check for duplicate tags

Use-Case name	Check for Duplicate Tags
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to check for duplicate playlist tags.
Actors	Spotify System
Pre-condition	The user must have at least one playlist
Post-condition	The system will notify the user if the tag already exists in the playlist's metadata.
Other business rules (if any)	

Success Path

1. The Spotify system receives a playlist tag from the user.
2. The Spotify system check's the playlist's saved tags to see if the tag already exists.
3. The Spotify system notifies UCN1 (tag playlist) if the tag is unique or already exists.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Tag playlist

Use-Case name	Listen to songs based on friends' music tastes
Last revised	11-11-19
Description	This use case describes the steps required for the user to listen to a custom playlist based off of friends' music library.
Actors	Free user, Subscribed user, Spotify system
Pre-condition	The user must be logged in and have at least one friend.
Post-condition	The user has a weekly friends discovery playlist in their music library.
Other business rules (if any)	

Success Path

1. The Spotify system generates a weekly playlist comprised of songs listened to and collected by user's friends.
2. The user listens to the playlist.
3. Use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Generate weekly curated "Friend Discovery" playlist

Use-Case name	Generate weekly curated "Friend Discovery" playlist
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to create a weekly playlist inspired by the user's friends' musical tastes.
Actors	Spotify System
Pre-condition	The user must have at least one friend with at least one playlist.
Post-condition	The user will have the "Friend Discovery" playlist available in their library, which is updated weekly.
Other business rules (if any)	

Success Path

1. The Spotify system analyzes the songs in the user's friends' music libraries to determine songs that might match the user's interests based on rhythm, tonality, and vocal style. It also looks at the most replayed songs in the friends' music libraries.
2. The Spotify system generates a playlist of 50 songs to put into that week's "Friend Discovery" playlist which is found in the user's music library.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Listen to songs based on friends' music tastes

Use-Case name	Transform playlists
Last revised	11-11-19
Description	This use case describes the steps required for the user to transform one or more playlists into a new playlist.
Actors	Subscribed user
Pre-condition	The user must be logged in and have at least one playlist.
Post-condition	The user's library now has a new playlist based off of a parent playlist(s).
Other business rules (if any)	The user must be subscribed to Spotify's premium account

Success Path

1. The user chooses to duplicate or merge or duplicate playlists.
2. The user chooses which playlist(s) to include in the operation.
3. The system prompts the user for the new playlist name. [Extend: Playlist already exists with that name.]
4. New transformed playlist is created.
5. Use case terminates.

Variations in success flows

1. The duplicate option can also be accessed from the user library by clicking on the option button next to the playlist name.

Alternate Paths (Extensions/Exceptions)

3a. Playlist already exists with that name.

- 3a.1. System prompts user to re-enter new playlist name.
- 3a.2. If user enters unique name (no other playlist has same name), return to 4.
- 3a.3. If user chooses to cancel the operation, the use case terminates.

1a. Duplicate Playlist

1b. Merge Playlists

List related use-case names

Check for duplicate playlist name

Use-Case name	Check for duplicate playlist name
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to check for duplicate playlist names.
Actors	Spotify System
Pre-condition	The user must have at least one playlist.
Post-condition	The system will notify the user if another playlist exists with the same name.
Other business rules (if any)	

Success Path

1. The Spotify system receives a playlist name from the user.
2. The Spotify system check's the user's music library to see if the name is already in use with another playlist.
3. The Spotify system notifies UCN3 (Transform playlists) if the playlist name is unique or already in use.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Transform playlists

Use-Case name	Merge Playlists
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to merge the user's selected playlists
Actors	Spotify System
Pre-condition	The user must have at least two playlists.
Post-condition	The system will create a new merged playlist.
Other business rules (if any)	

Success Path

1. The user selects the merge option via UCN3 (Transform playlists)
2. The spotify system prompts the user to select which playlists to merge from a list of the user's playlists.
3. The spotify system combines the songs from the selected playlists into one new album.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Change view of playlists

Use-Case name	Duplicate Playlist
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to duplicate a user's playlist.
Actors	Spotify System
Pre-condition	The user must have at least one playlist.
Post-condition	The system will create a new duplicated playlist.
Other business rules (if any)	

Success Path

1. The user selects the duplicate option via UCN3 (Transform playlists)
2. The spotify system prompts the user to select which playlist to duplicate from a list of the user's playlists.
3. The spotify system creates a new playlist with songs copied from the selected playlist.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Change view of playlists

Use-Case name	Change view of playlists
Last revised	11-11-19
Description	This use case describes the steps required for the user to change their view of playlists in their music library through sorting or filtering.
Actors	Subscribed user
Pre-condition	The user must be logged in and have more than one playlist.
Post-condition	The user's playlists are now sorted or filtered in the manner chosen by the user.
Other business rules (if any)	The user must be subscribed to Spotify's premium account

Success Path

1. User clicks on the filters button at the top-right corner of the music library.
2. User chooses from Pop-up menu options: Sort by name, sort by date, filter user created, filter spotify created
3. System arranges view of playlists in manner chosen by user.
4. Use case terminates.

Variations in success flows

2. The user can click on either sort option multiple times to toggle the sort mode between ascending and descending.
2. The user can also apply these settings when choosing what playlist to add a song to.

Alternate Paths (Extensions/Exceptions)

2a. Filter Playlists
2b. Sort Playlists

List related use-case names

Use-Case name	Filter Playlists
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to filter the user's playlists
Actors	Spotify System
Pre-condition	The user must have at least two playlists.
Post-condition	The system will update the user's library to only show the playlists in the chosen filter method.
Other business rules (if any)	

Success Path

1. The user selects a filter option via UCN4 (Change view of playlists)
2. The Spotify system hides all playlists except for the type selected in the filter option. (i.e. filter user-created will only show playlists created by the user)
4. The spotify system stores the user's filter preference.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Change view of playlists

Use-Case name	Sort Playlists
Last revised	11-11-19
Description	This use case describes the steps required for the Spotify System to sort the user's playlists
Actors	Spotify System
Pre-condition	The user must have at least two playlists.
Post-condition	The system will update the user's library to show playlists sorted in the order chosen by the user.
Other business rules (if any)	

Success Path

1. The user selects a sort option via UCN4 (Change view of playlists)
2. The Spotify system sorts the playlists in the user's library in the method chosen (by name or by date).
4. The spotify system stores the user's sort preference.
4. The use case terminates.

Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Change view of playlists

Use-Case name	Set image as playlist icon
Last revised	11-11-19
Description	This use case describes the steps required for the user to set an image as their playlist icon.
Actors	Subscribed user
Pre-condition	The user must be logged in and have at least one playlist.
Post-condition	The user's playlist now has an icon associated with it.
Other business rules (if any)	The user must be subscribed to Spotify's premium account

Success Path

1. User clicks on the "edit details" option from the options drop-down menu.
2. User clicks on "add image" button.
3. User selects image from their computer's file explorer.
4. User clicks on "save" button.
5. Use case terminates.

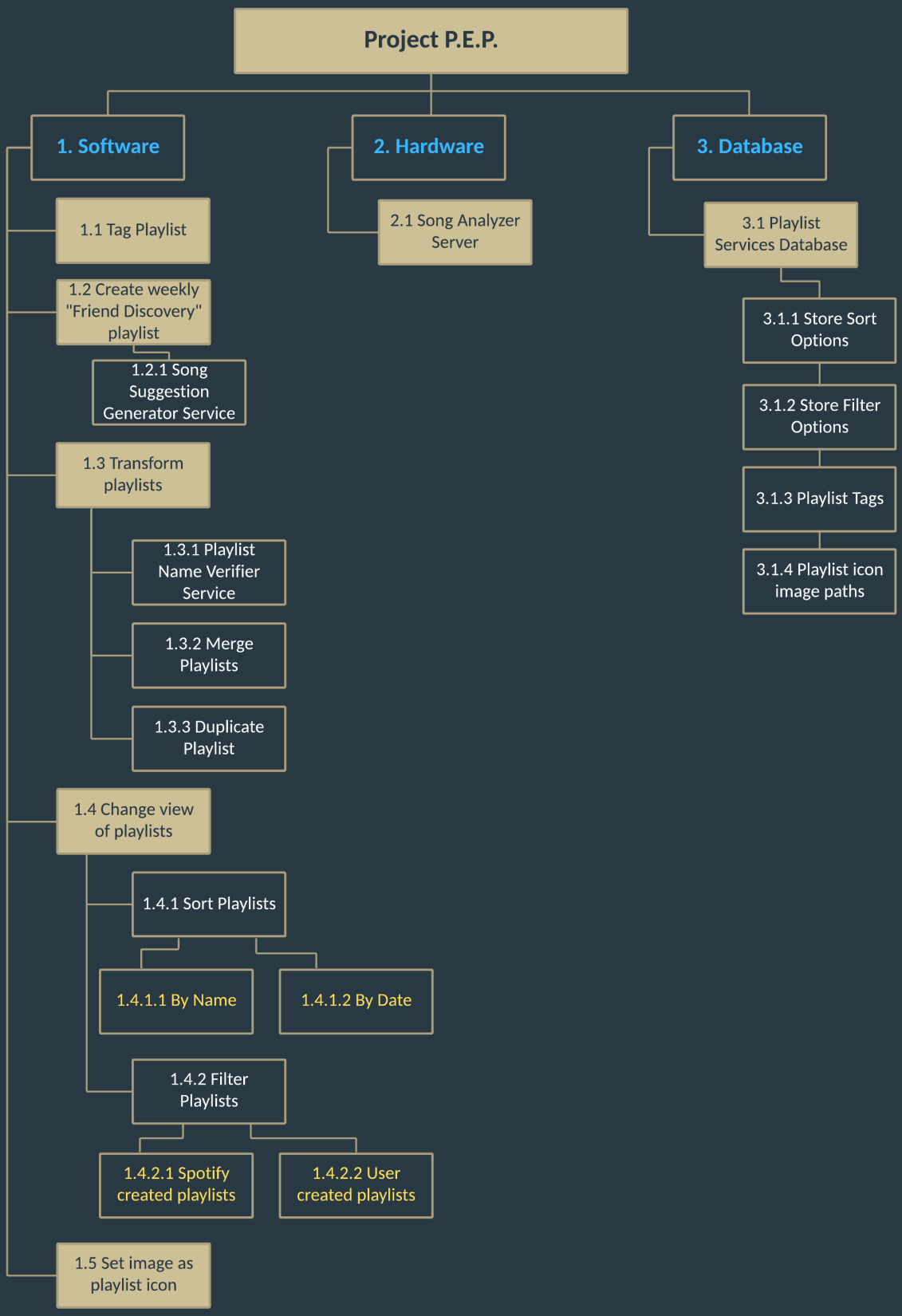
Variations in success flows

Alternate Paths (Extensions/Exceptions)

List related use-case names

Project Plan

Project-Oriented WBS



Estimation

Using Agile Methodology

A Planning Poker meeting was conducted with managers from Spotify's development teams to estimate the effort and labor needed for this project as well as its duration. We decided a 2-person team would be appropriate for the size of this project. Spotify uses the Agile methodology for all of its development projects and has standardized the length of sprints and maximum story points per developer at 2 weeks and 11 points, respectively. Shown below are the results of the Planning Poker meeting.

I want	So That	Acceptance Criteria	Points	Priority	Sprint
to be able to change how my playlists are organized	it is easier for me to find the playlist I want to listen to	Add ability to sort playlists by name, date added, as well as ability to filter out spotify created playlists or user created playlists	13	1	1
the sorting to happen instantaneously	I don't have to wait to listen to my playlist	Sort must complete within 0.5 seconds	5	2	1
Create another playlist based on the ones already in my library	I don't have to create one from scratch if I want to include songs I already have in my playlists	Add ability to duplicate and merge playlists.	8	4	3
To be able to visually identify my playlists by icon instead of text	I can easily find my playlists without having to use as much screen space	Add ability for user to upload an image for their playlist icon	5	6	3
It want it to be easy to create a good quality playlist icon	so that it doesn't look pixelated or distorted	add instructions in the upload screen that tells user of correct image size, resolution, and format	3	7	3
get song suggestions based on my friends' music preferences	so that I can discover new songs to add to my playlists and learn what music my friends like	Analyze friends' music libraries and create weekly discover playlist for user based on friends' music tastes	21	3	2
to be able to categorize my playlists in multiple ways	so that I can easily search for them by moods, genres, etc	Box at the bottom of user playlist available for users to insert tags separated by commas	5	5	3
		Total Points:	**60**		
		Team Velocity:	**22 Points**		
		Sprints Needed:	**2.7**		

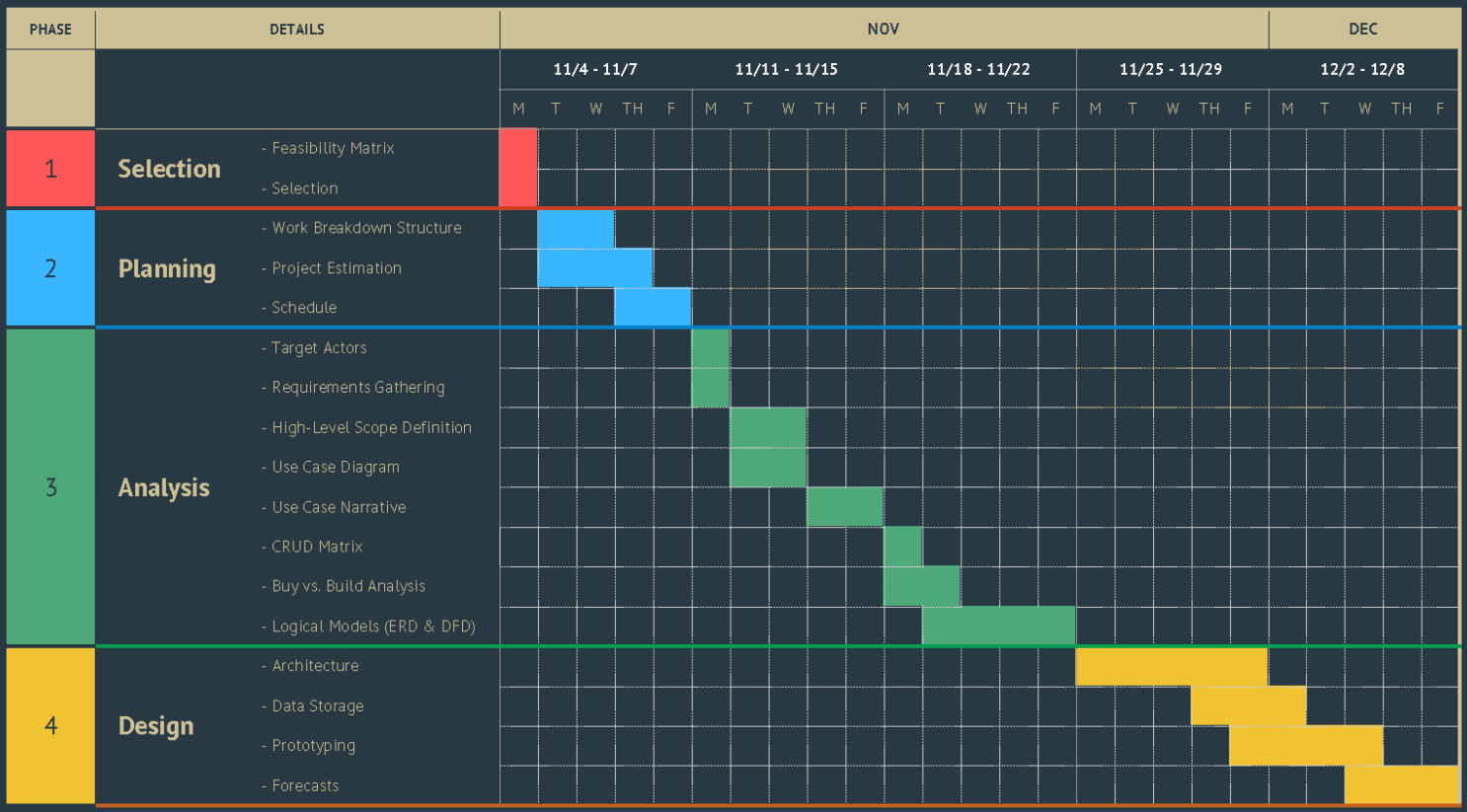
Schedule

The Project P.E.P. development team will consist of two members - one Sr. Developer and one Jr. Developer. In addition to the six weeks (3 sprints) of software development, they will need time for the following items: Song Analyzer server setup, Playlist Services database setup, implementation, and documentation. In total, we estimate our Project P.E.P. team will require nine weeks to complete their tasks.

Please note that all activities on the following charts are in weekly format except for the sprints which are two-week blocks.

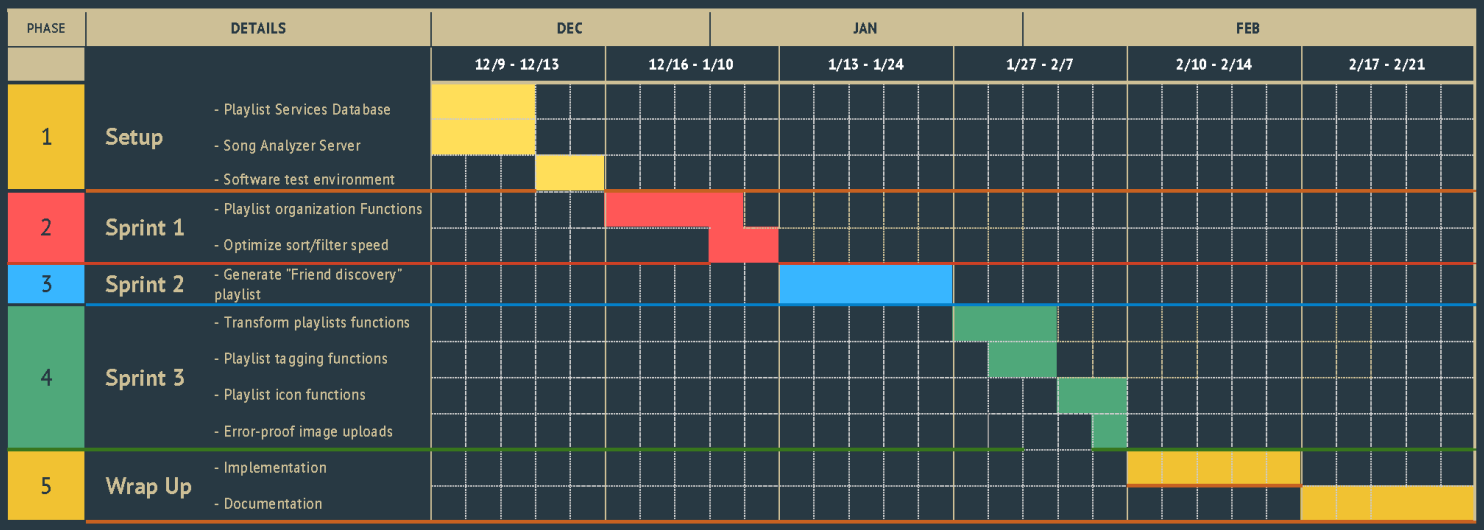
Semester Activities

PROJECT TITLE	Project P.E.P.	**COMPANY NAME**	Spotify
PROJECT MANAGER	Michelle Gordon	**DATE**	11/1/19



Post-Semester Activities

PROJECT TITLE	Project P.E.P.	COMPANY NAME	Spotify
PROJECT MANAGER	Michelle Gordon	DATE	11/1/19



Labor Costs

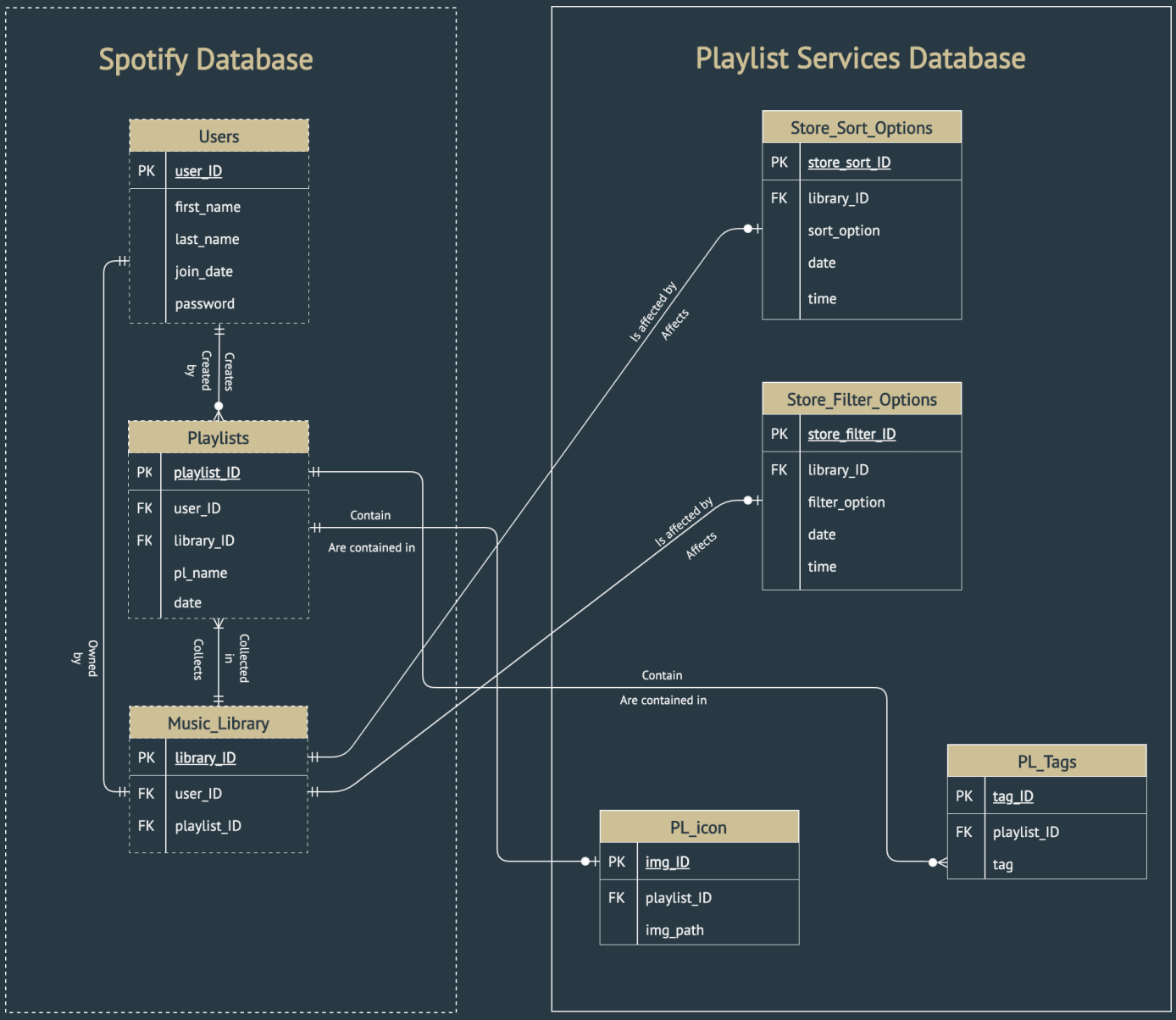
The Project P.E.P. development team will consist of two members - one Sr. Developer and one Jr. Developer. Due to the difference in salary between the two developers, we have calculated the total labor cost as follows:

Title	Hourly Rate	Weekly Salary	Project Cost (9 weeks of work)
Senior Developer	$60	$2,400	$21,600
Jr. Developer	$40	$1,600	$14,400
		Total:	$36,000

Analysis Documents

Logical ERD

The diagram below shows the data that will be stored in Project P.E.P.'s Playlist Services Database. The entities within the dashed line boundary are some of the entities located within the greater Spotify system that will interact with the Playlist Services DB entities.



- A user can have 1 and only 1 library, and a library can have 1 and only 1 user.
- A user can create 0 to many playlists, and a playlist can be created by 1 and only 1 user.
- A playlist can be stored in 1 and only 1 library, and a library can store 1 to many playlists.
- A playlist can contain 0 to many tags, and tags can be contained in 1 and only 1 playlist.
- A playlist can contain 0 to 1 icons, and an icon can be contained in 1 and only 1 playlist.
- A library is affected by 0 to 1 stored sort options, and stored sort options can affect 1 and only 1 library.
- A library is affected by 0 to 1 stored filter options, and stored filter options can affect 1 and only 1 library.

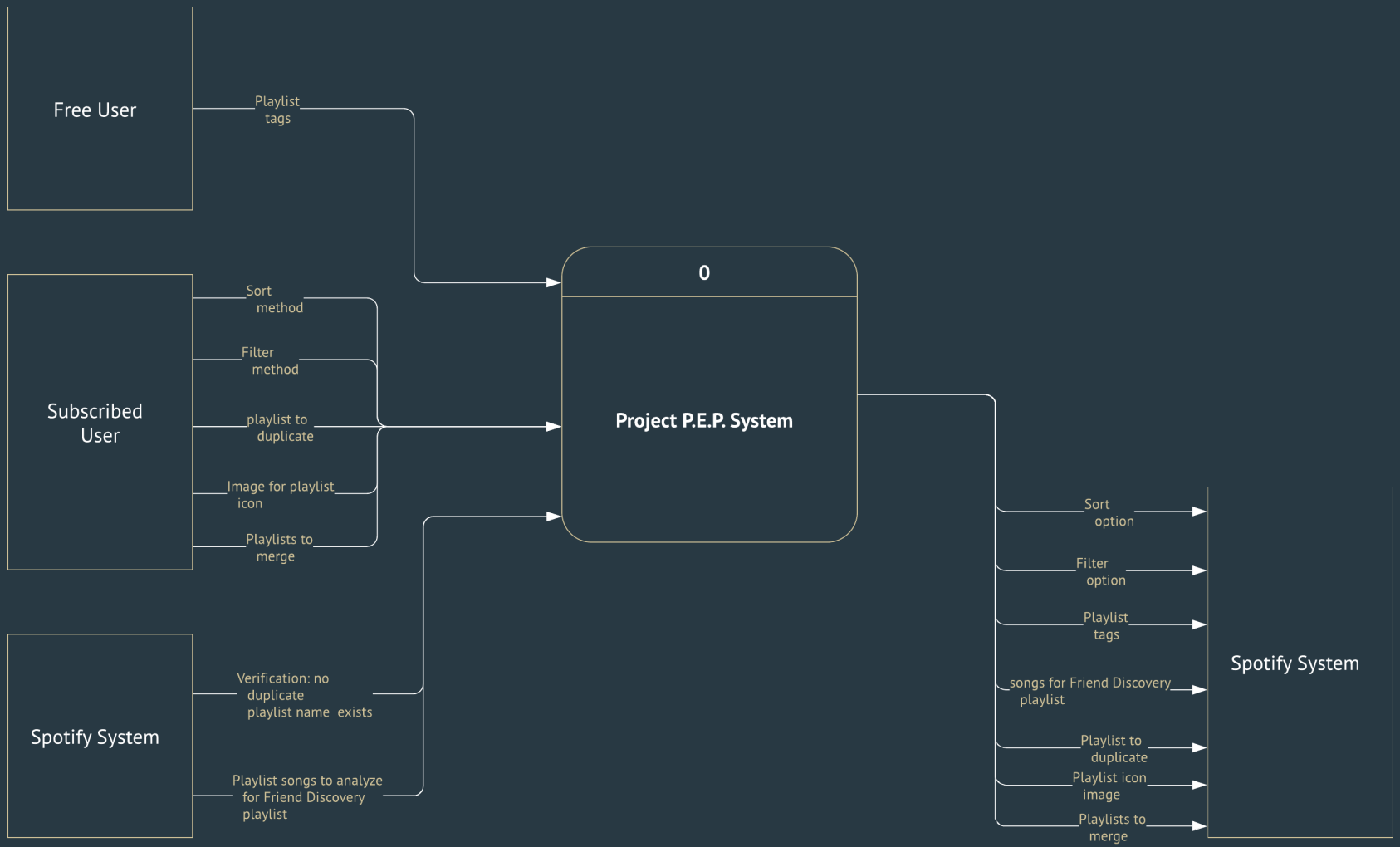
Data Dictionary

The data dictionary shows the entities and attributes in the ERD diagram and explains the function of each item. This includes all of the items within the scope of Project P.E.P.

Entity	Attribute	Description	Format
Store_Sort_Options	This entity stores the last sort option the user selected, so that when they come back to their music library, it is still sorted the same way.		
	store_sort_ID	Unique ID number for each user's sort activity	SERIAL
	library_ID	Unique ID for user's music library from Music_Library entity	INT(10)
	sort_option	Stores the type of sort the user selected (1=by name, 2=by date)	INT(1)
	date	Date of when sort method was selected	DATE
	time	Time of when sort method was selected	TIME
Store_Filter_Options	This entity stores the last filter option the user selected, so that when they come back to their music library, it is still sorted the same way.		
	store_filter_ID	Unique ID number for each user's filter activity	SERIAL
	library_ID	Unique ID for user's music library from Music_Library entity	INT(10)
	filter_option	Stores the type of filter the user selected (0=none, 1=only show user playlists, 2=only show Spotify created playlists)	INT(1)
	date	Date of when the sort method was selected	DATE
	time	Time of when the sort method was selected	TIME
PL_icon	Subscribed users will upload images to create playlist icons. This entity stores the path to where the image is stored on the server.		
	img_ID	Unique ID number for each image that is uploaded.	SERIAL
	playlist_ID	Playlist's unique ID from Playlists entity	INT(10)
	img_path	location of the image file on the server	VARCHAR(MAX)
PL_Tags	This entity stores tags that users have applied to their playlists.		
	tag_ID	Unique ID number for each tag entered by the user	SERIAL
	playlist_ID	Playlist's unique ID from Playlists entity	INT(10)
	tag	The actual tag entered by the user	VARCHAR(30)

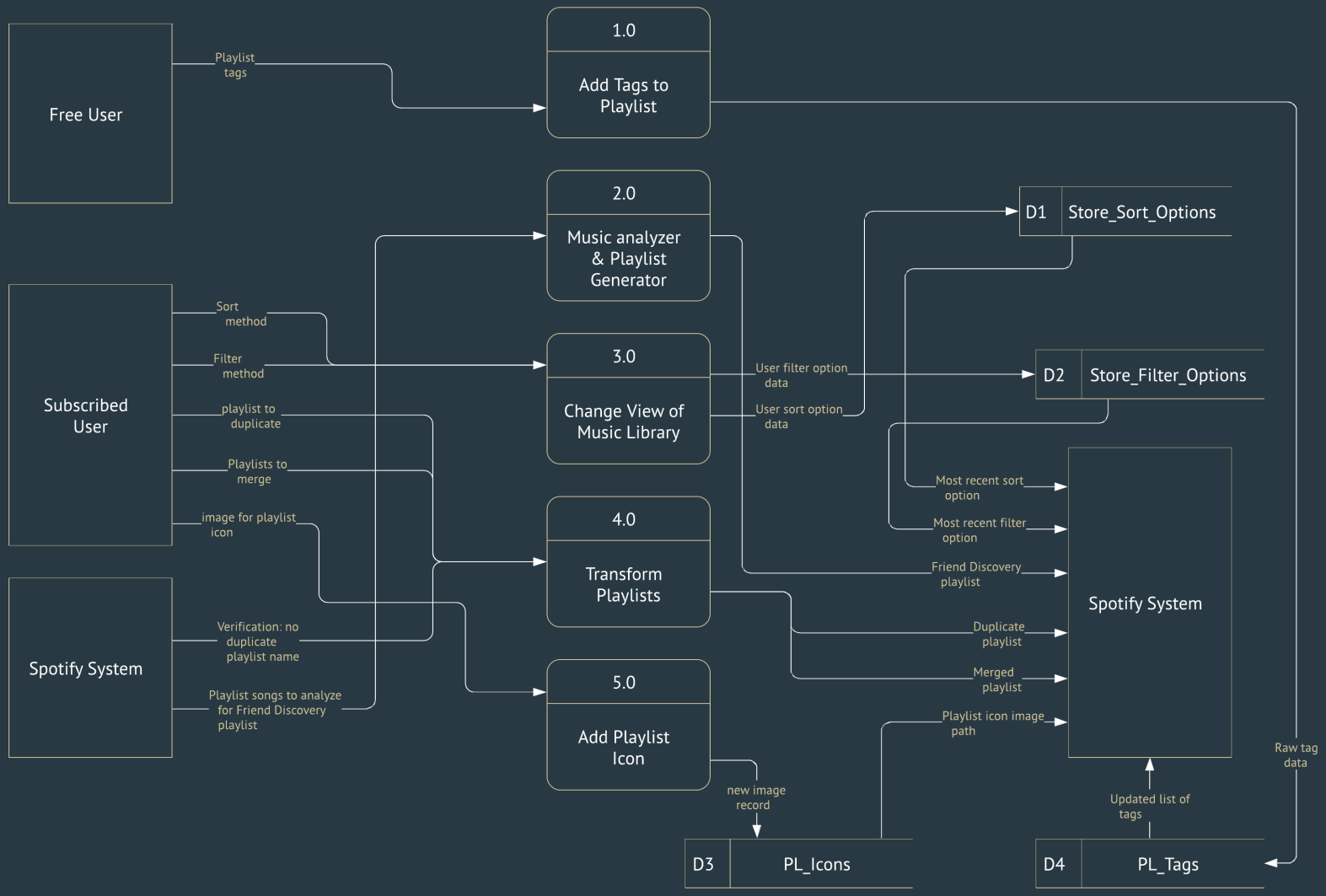
Logical DFD

Context Diagram



Logical DFD

Level 0



CRUD Matrix

The CRUD matrix is used to check that all data stores have a purpose and are being used by at least one process. The letters in the cells indicate how processes use the data stores: C = creates, R = reads, U = updates, and D = deletes.

Program/Table	D1: Store_Sort_Options	D2: Store_Filter_Options	D3: PL_Icons	D4: PL_Tags
Add Tags to Playlist				CRUD
Music analyzer & Playlist Generator				
Change View of Music Library	CRU	CRU		
Transform Playlists				
Add Playlist Icon			CU	

Buy vs. Build Analysis

Project P.E.P. is both a simple project and a complicated project in that we are not rebuilding Spotify as a whole, but adding a small yet intricate module to the system to improve user interactions with their playlists. In considering the best approach, the intricacies of Project P.E.P. and the importance of its successful integration and maintenance outweigh the incentives to cut costs or internal effort. I reviewed three options and considered their benefits and risks.

Outsourcing

There is no question that outsourcing development internationally can cut costs dramatically. A sampling of salaries for a Sr. Software Developer around the world provided the following salary range:

- India - $15/hr
- Vietnam - $30/hr
- Ukraine - $35/hr
- Mexico - $55/hr

In addition to saving on salary costs, outsourcing also cuts on HR costs for things like employee benefits and administrative effort. There are also a large number of risks involved with outsourcing. Communication and coordination becomes complicated when managers and other team members are in different time zones. There is increased time required due to on-boarding and end-of-contract debriefing.

Lastly, there are two risks that are important to consider. First, when rolling off contractors, it is very common to incur a loss of knowledge that then impacts maintenance of the system. Second, and most important, is the risk of proprietary information leaking out to competitors. Non-compete and non-disclosure agreements are commonly used to combat proprietary disclosures, but they are nearly impossible to enforce outside of the US. Considering the competitive environment of music streaming services, this is a risk that should be considered heavily.

Third Party Software

Integrating third-party software (or commercial packages) with customized software is notorious for being difficult. This would definitely be the case with Project P.E.P. because it is a feature-enhancement project for custom-built software. The black-box aspect of this type of solution also requires on-going maintenance provided by the software company while it is still in use. This means that significant effort would have to be placed on maintaining a good relationship with the vendor in order to secure quick response times if something were to go wrong.

The only part of Project P.E.P. that would be compatible with third-party software would be the song-suggestion generator service. It could run separately from the Spotify system, and supply song suggestion data as needed.

Build In-house

Building the system in-house appears to have the most benefits for our organization. In-house knowledge of processes and business rules contained by the development teams would cut out the need for on-boarding time or training.

One of the most important aspects about developing Project P.E.P. in-house is the security of Spotify's proprietary information. By keeping the development within Spotify, the chance of information being leaked is dramatically reduced. Keeping the development in-house also keeps the knowledge in-house. This becomes an important asset when it comes to maintaining and upgrading software over time.

In order to make sure that all options were considered fairly, I used an alternative matrix that would produce a weighted score for each option.

Evaluation Criteria	Importance (Weight)	Option 1: Outsourcing Internationally	Score (1-5)	Weighted Score	Option 2: Third-Party Software	Score (1-5)	Weighted Score	Option 3: In-House Development	Score (1-5)	Weighted Score
Technical:										
Complexity	10	Complex because it will be custom built to integrate with Spotify system, but a small project	2	20	Simple because would already be built and would just need customization (if possible)	4	40	Complex because it will be custom built to integrate with Spotify system, but a small project	2	20
Technical Risk	20	Medium risk - possible miscommunication of requirements due to language barriers, and possible loss of knowledge after project ends.	3	60	High risk - difficult to integrate 3rd party software with custom built enterprise system	1	20	Lowest - In-house expertise with current system, and keeping knowledge of new system in-house allows for highest level of control and customization	4	80
Economic:										
Cost	10	Half the cost of hiring within the US or using in-house developers	5	50	Lower cost up front but requires monthly payment and a contract	3	30	Higher cost up front with minimal maintenance costs in the future	3	30
Time Investment	15	more time invested in addition to development and integration for onboarding and debriefing	2	30	Only time for integration and testing	4	60	only need time for development and integration	3	45
Organizational										
Ease of Integration	20	Will be custom built module, so would integrate well if developed well	4	80	Very difficult to match pre-built software with business needs and rules	1	20	Very easy, especially since Spotify is an agile development environment that uses iterative approach	5	100
Security of Proprietary Info	25	Low control over developers' ethics or competing interests. Non-disclosure agreements hard to enforce.	1	25	Integration will happen in-house, so 3rd party company would have access to very little proprietary info, if at all	4	100	Most secure - all info stays within company and all employees have signed non-compete and non-disclosure agreements	5	125
Total	100			265			270			400

Design

Architecture & Data Storage

Spotify divides its development teams into what are called self-service "squads." Each squad handles a different feature of Spotify in its entirety. This includes developing, implementing, and maintaining those features on all versions of the app - cloud player, mobile app, tablet app, smart tv app, and desktop. Therefore, Project P.E.P. will be handled by the playlist squad and will be pushed out to all of Spotify's available platforms.

Additionally, consistency and scalability is important as Spotify's audience continues to grow. Thus, instead of setting up Project P.E.P. on different technology, it will be designed to align with our current information systems. Due to the variety of distributions of Spotify, Project P.E.P. will be developed in C++ and Java, with the backend being written in Python. It will then be integrated into the Spotify system that is currently hosted on Google Cloud to ensure dynamic scalability.

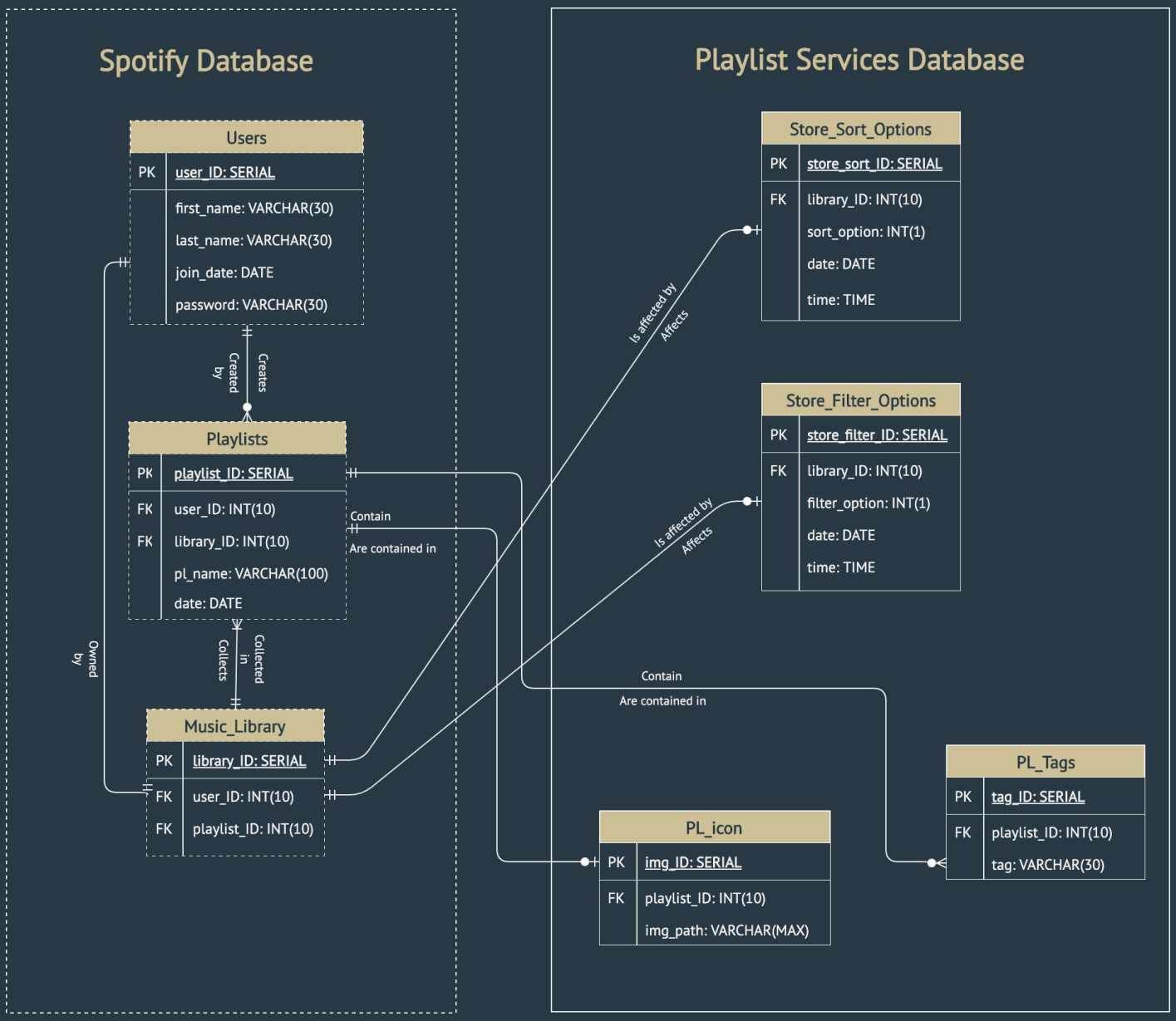
Spotify's data is stored in multiple secure data centers as a variety of Apache Cassandra or PostgreSQL multi-site clusters. This ensures its reliability and performance by having redundancy in place in case of overload or failure. Data from Project P.E.P. will be stored in an Apache Cassandra database that will be setup within this infrastructure.

Part of the decision to use Apache Cassandra was its ability to handle large amounts of data with high performance. Per the first non-functional requirement in the user stories, this is essential as our users want to be able to manipulate their playlists without lag-time or other performance issues.

Lastly, UX design principles will be focused on in order to address the second non-functional requirement from the user stories. In order to make sure that a user's playlist icon looks clear, instructions will be included on the upload screen. The instructions will indicate the correct file type and size to achieve a good looking playlist icon.

Physical ERD

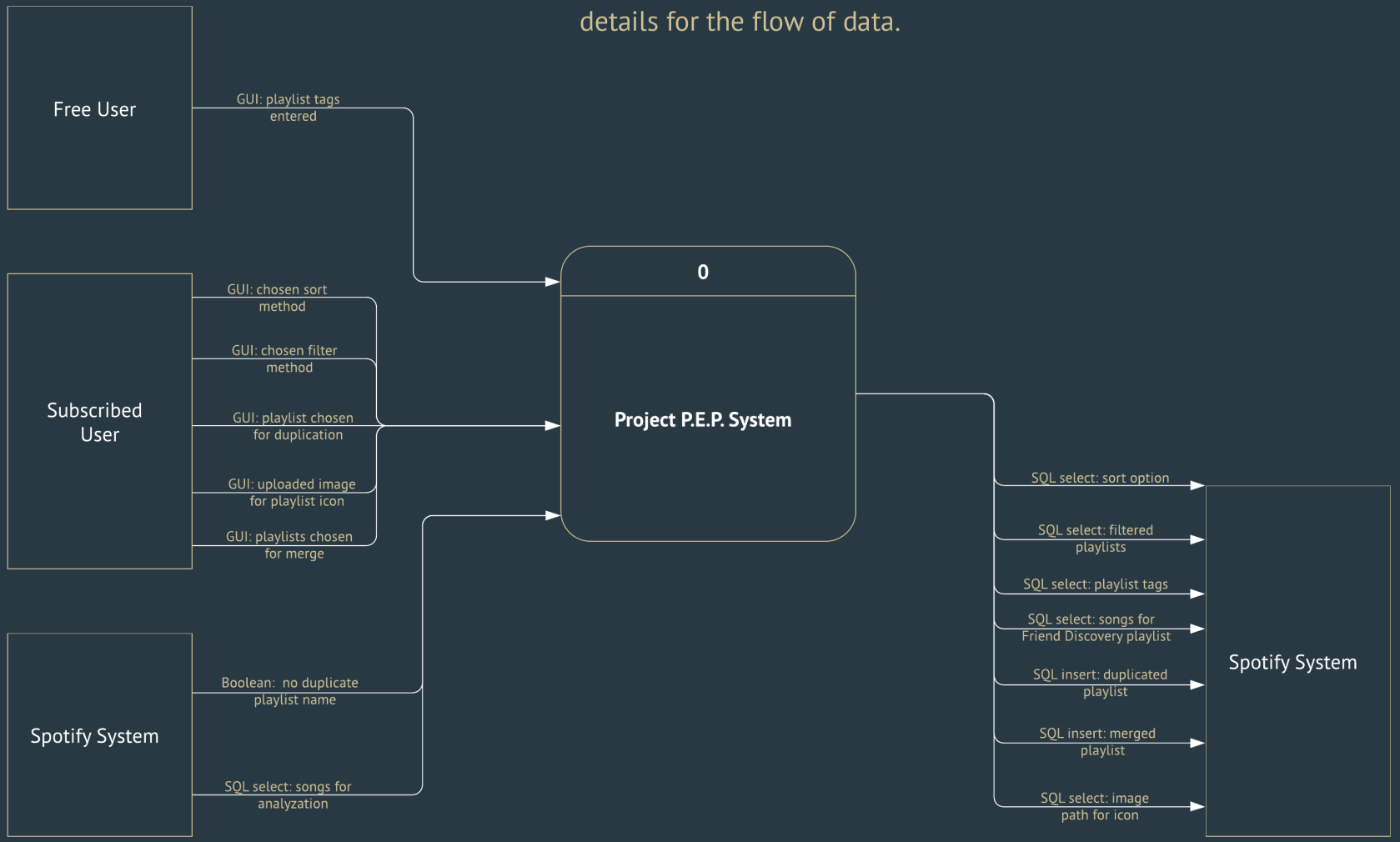
The physical ERD presented below augments the logical ERD by including the design of the entities in the Playlist Services Database.



Physical DFD

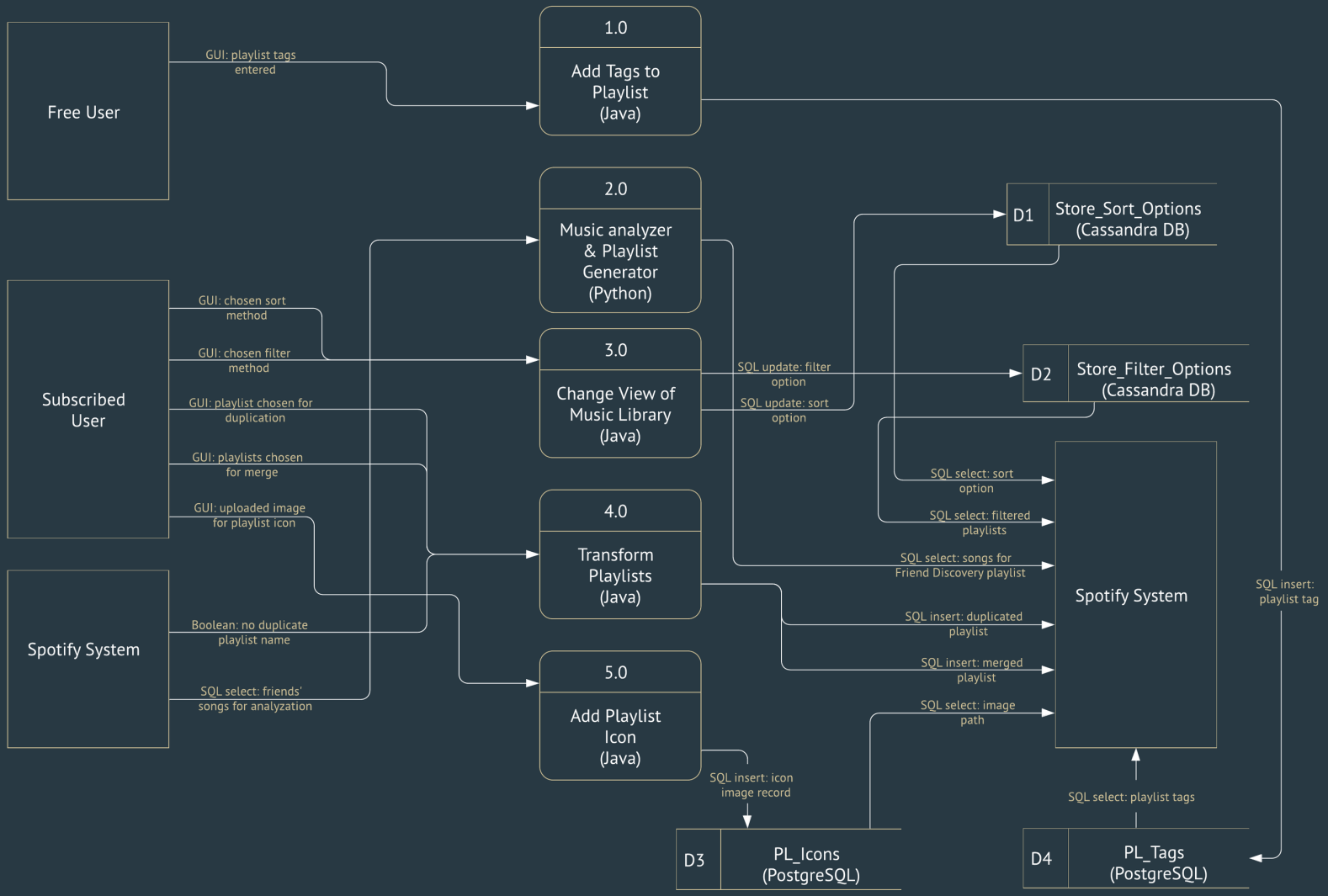
Context Diagram

Physical DFD's augment the logical DFD's by including implementation details for the flow of data.



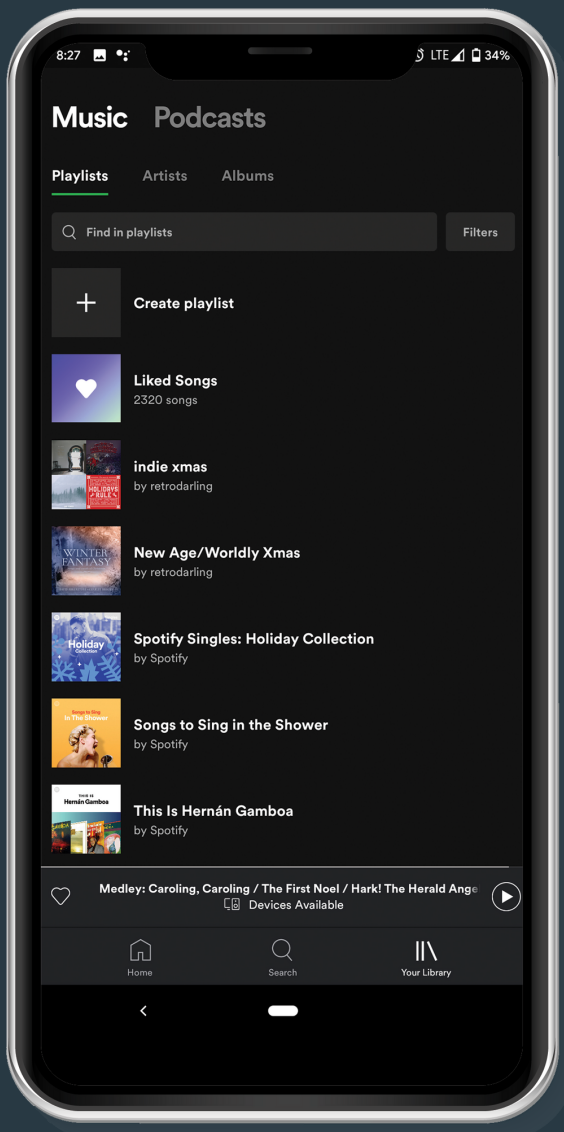
Physical DFD

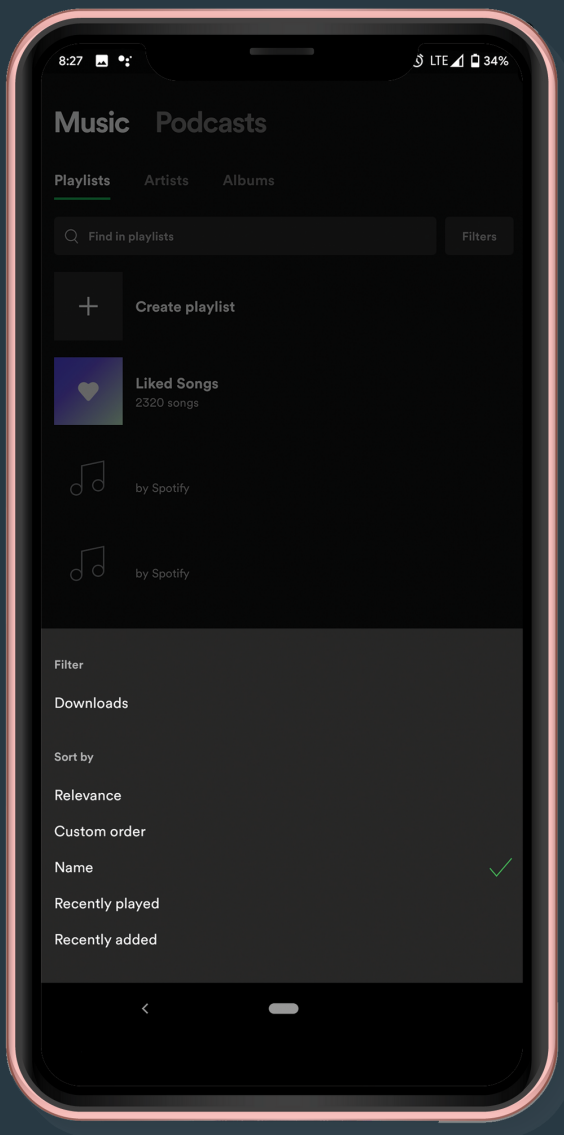
Level 0



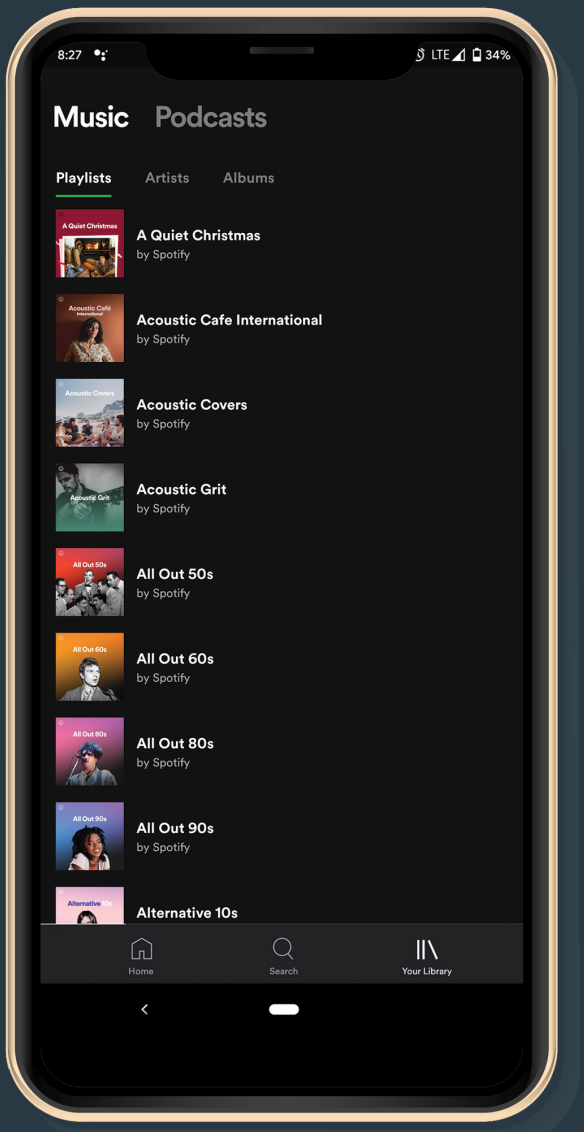
Prototypes

Below is a prototype of the sort feature. The user will click on the "Filters" button at the top-right corner, which brings up a menu of options. In this example, the customer has chosen to sort by name.

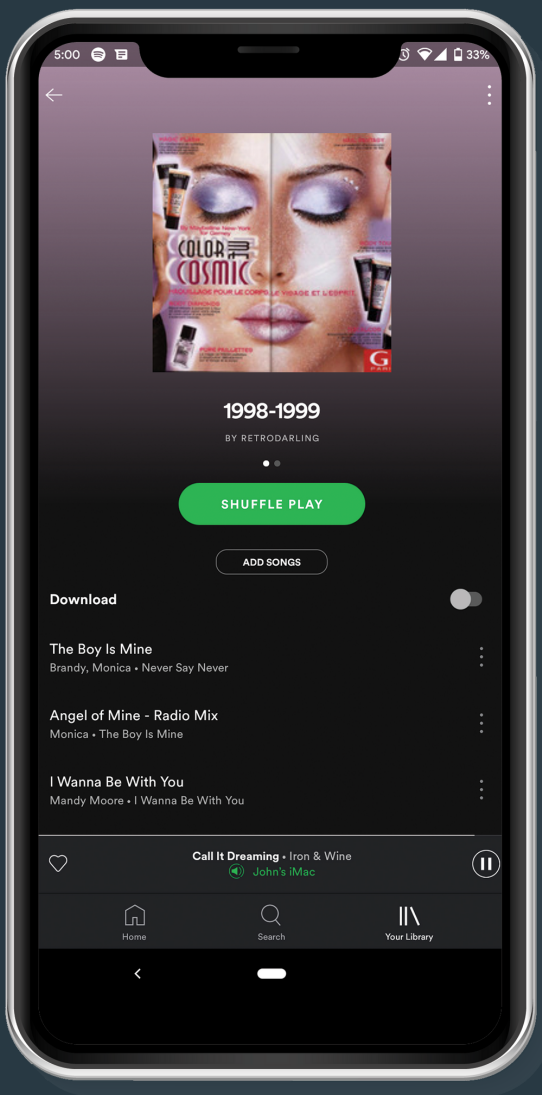


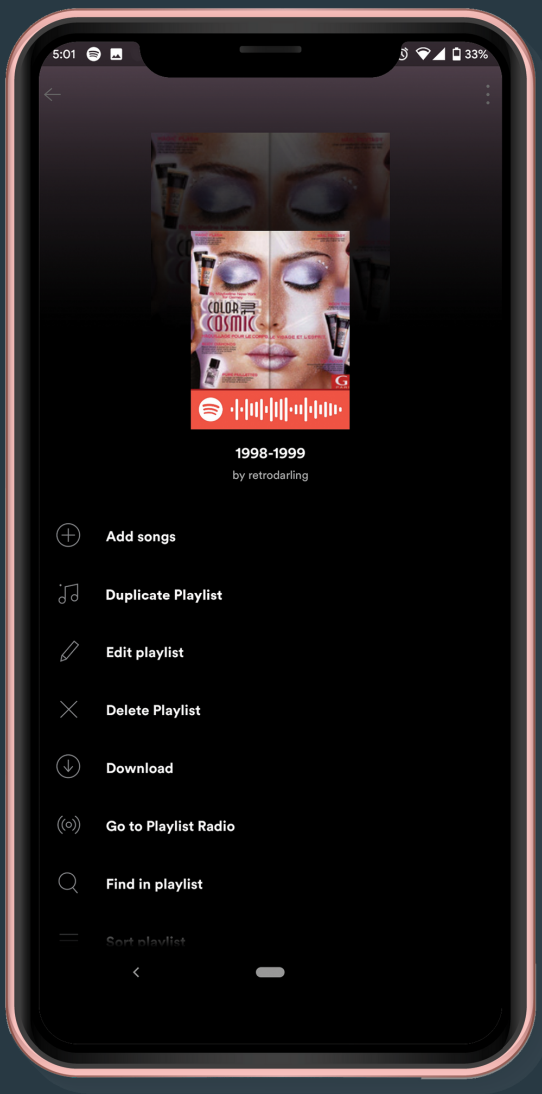


On the last screen, you will now see the playlists have been sorted by name.

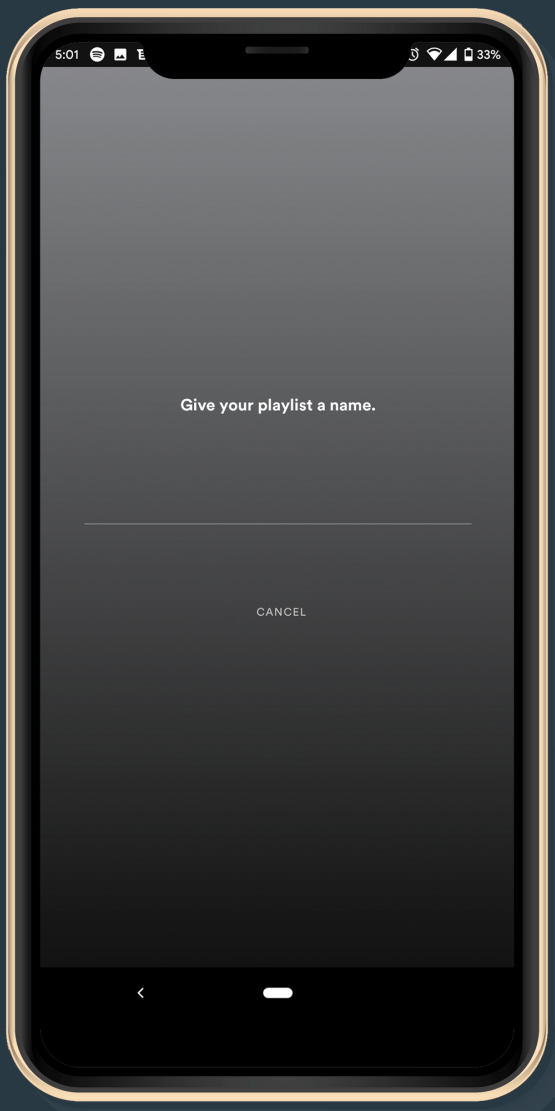


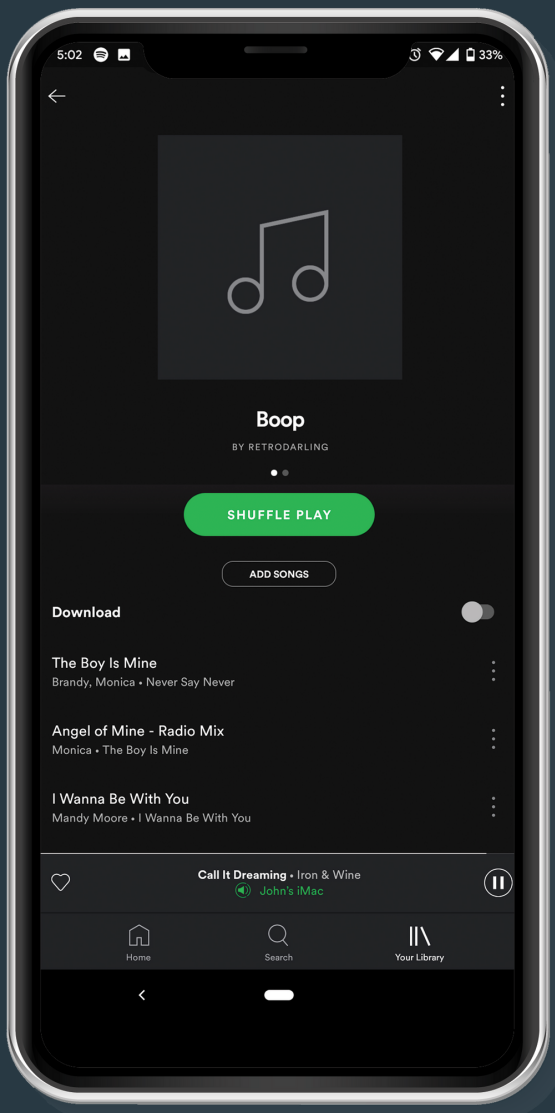
Next, is a prototype of the duplicate playlist feature. The user will access the playlist's options by pressing on the three dots at the top-right corner of the screen, and then select the "Duplicate Playlist" option.



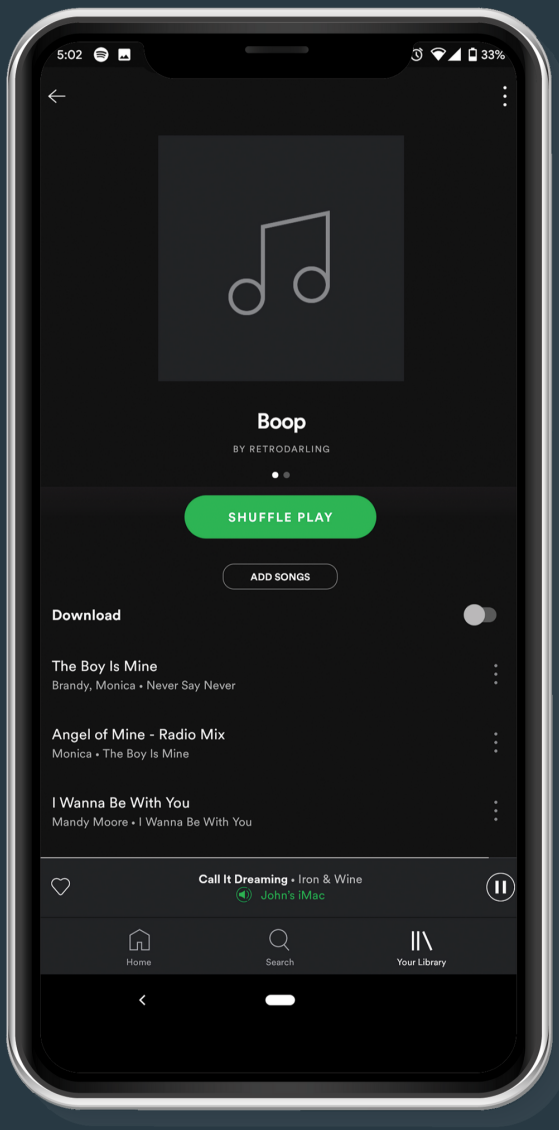


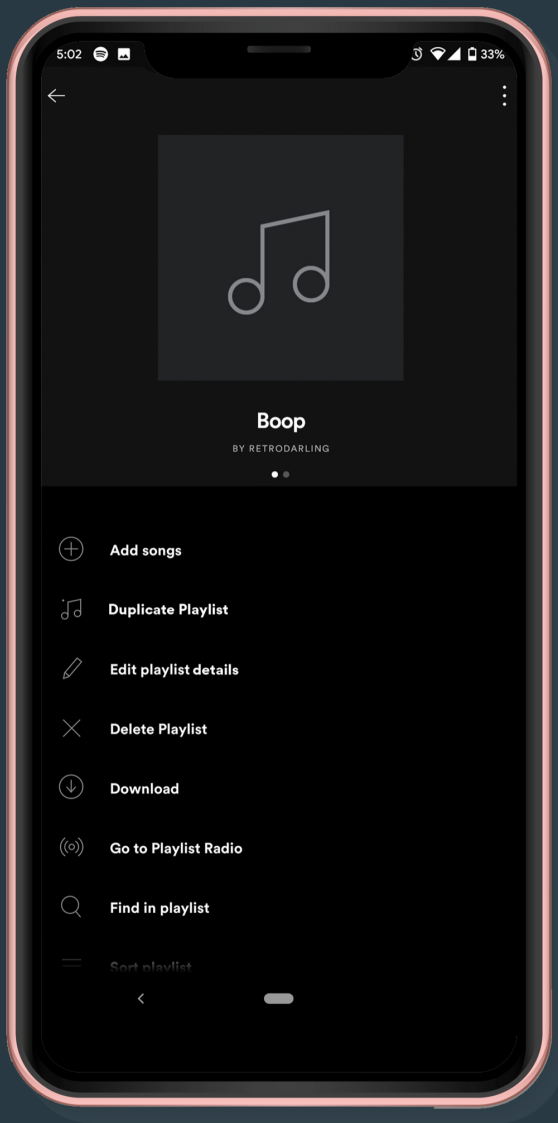
The user will be prompted to name the new playlist. Then, a new playlist with the new name will be created that will contain the same songs as the original playlist.



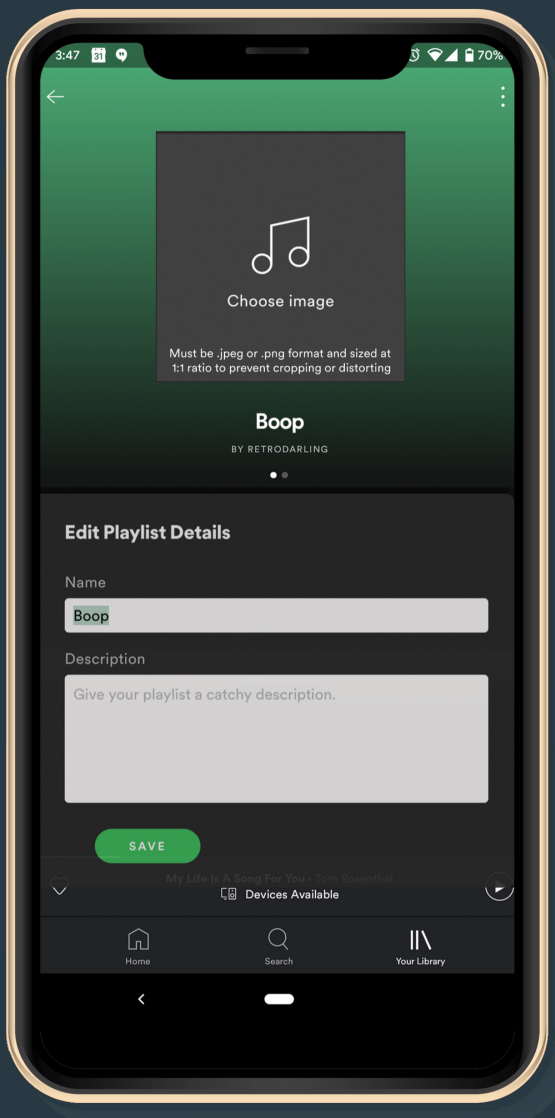


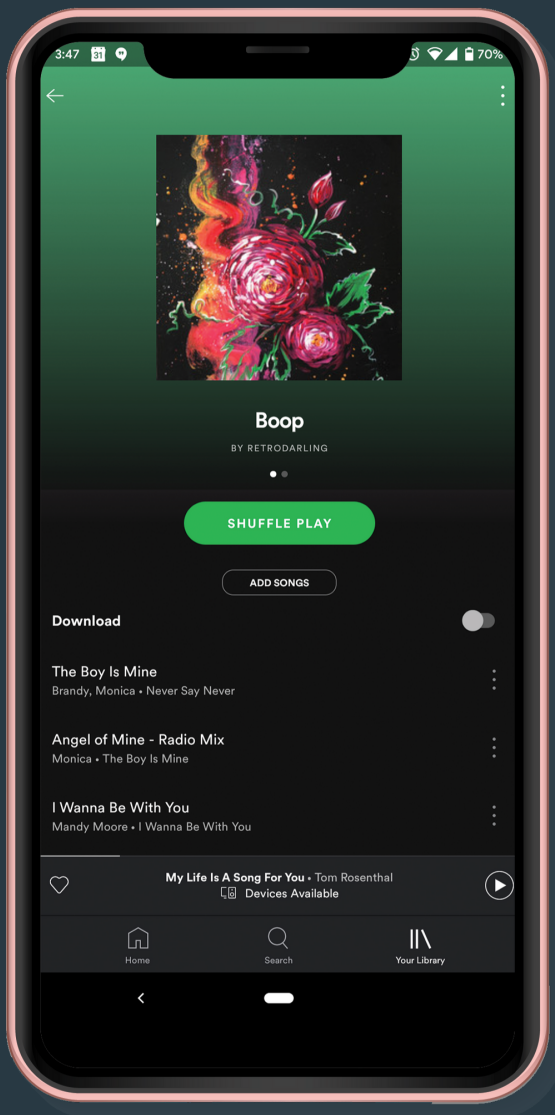
Shown below is a prototype of the process to upload an image to create a playlist icon. The process starts by pressing on the option button in the upper right-hand corner of the screen. The user will then select the "Edit playlist details" option.

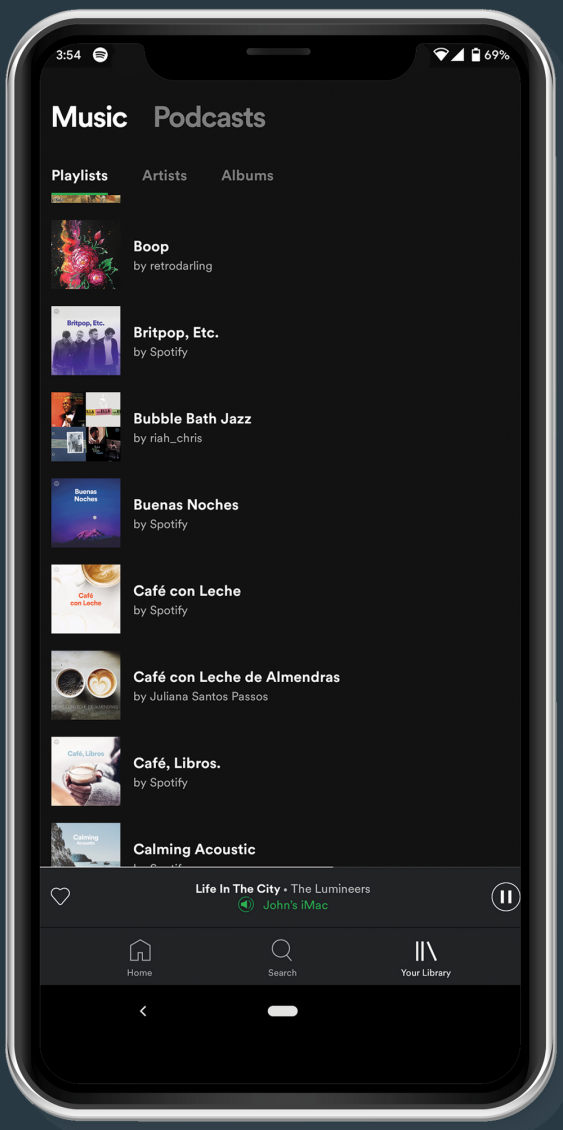




The system will pop up the "edit details" screen which includes the "choose image" button. Below the "choose image" button are instructions for the required file types and correct ratio of the image to ensure great image quality after upload. When the user presses that button, it will prompt them to choose a file from their device's storage to upload. Once the image has been uploaded, it will appear on the playlist's screen as well as in the music library.







- rest of page left intentionally blank -

References

Information:

- https://www.atlassian.com/agile/project-management/user-stories
- https://seilevel.com/requirements/the-anatomy-of-a-user-story
- https://labs.spotify.com/2013/03/15/backend-infrastructure-at-spotify/
- https://labs.spotify.com/2013/03/20/how-we-use-python-at-spotify/
- https://labs.spotify.com/2013/02/25/in-praise-of-boring-technology/
- https://qubit-labs.com/average-hourly-rates-offshore-development-services-software-development-costs-guide/
- https://play.google.com/music/listen?u=0#/sulp
- https://www.pandora.com/
- https://www.slacker.com/

Software:

- Teamweek
- Google Sheets
- Google Forms
- Google Drive
- Canva
- Draw.io
- Lucidchart
- Adobe Photoshop
- Adobe Acrobat
- Twitter
- Spotify