jTunes Project Proposal

Jonathan Ziesmer
School of Computing
Southern Adventist University
Collegedale, United States
jziesmer@southern.edu

***Abstract*—Today's music players are not making use of all the features computers have to offer. The ideal music player will auto-generate playlists that exactly fit the needs of the user and would provide the user with a more intuitive way of sorting their music. Sophisticated auto-generation of playlists is still a long way off from being realized, but this proposal explores how such an algorithm might function. The project itself aims to combine an intuitive music storage interface with a backend ready for a sophisticated playlist generation algorithm to create a prototype music player with an improved user experience.**

I. Introduction

Music is the ultimate pathway for human emotion. We write it to tell stories, play it to express emotion, and listen to it to connect with others. Computers have been integral in how we categorize and consume music ever since the release of WinAMP. Sadly, however, much of their potential to improve our music-consumption experience has gone untapped. None of today's music players have satisfactory playlist generation, and they limit themselves with clunky, inflexible data-handling functionality. By building a music player designed to overcome these flaws, I will provide users with a simple tool allowing them to interact with their music in an intuitive, logical, and flexible manner.

The ideal music player software would auto-generate quality playlists for any scenario described by a short list of parameters, relying on songs from the user's personal library, but reinforced by massive libraries like those of Pandora or Spotify. This would involve a personal database of the user's songs, the licensing and API for a Spotify/Pandora-like cloud-based library, sophisticated neural networks for playlist generation, and a UI to connect all the pieces together. Sadly, such a player is far beyond the scale of this project. The present goal of jTunes is to prototype a local application implementing a music database, very basic playlist generation functionality, and a simple, functional UI. Hopefully, a future me, or another unrealistically altruistic developer, will pick up the project and carry it closer to the ideal. For this basic implementation, jTunes will be targeting Windows 10 users who already have a library of music files on their local machine.

Section II includes background information on prior work, a brief survey of music emotion annotation literature, and music emotion annotation methodology. Section III discusses the proposed project, including a description of the chosen solution, a delineation of major tasks, the final deliverables, and a proposed timeline. Section IV discusses the testing and evaluation plan. Section V concludes the proposal and is followed by an appendix in Section VI, containing a detailed list of tasks, target results, and a draft user acceptance survey.

II. Background

A. Prior Work

Many platforms have implemented some form of playlist-generating functionality, including well-known platforms such as Pandora, Spotify, and Apple Music.

In 2018, Pandora launched a new feature called "Personalized Soundtracks" which automatically creates playlists of varying themes for Pandora Premium users [1]. The playlists are updated every week to represent the themes and songs preferred by the user. The algorithm used to create these playlists draws on data from Pandora's famous Music Genome Project (MGP), which includes 30 million tracks with over 400 hand-labelled features for each track [2]. The MGP's data allows the algorithm to use extremely well-trained neural networks to select tracks for individual playlists [3]. Pandora was also able to draw on their sophisticate "listener affinity" algorithms, by which songs are chosen that Pandora hopes the user will like. While most reviewers reacted positively to Pandora's Personalized Soundtracks, Pandora's other features are lacking. Due to its digital radio format, licensing restricts Pandora users from playing individual songs. Pandora also uses the low-quality, lossy AAC file format at only 192kbps. Perhaps most egregiously, the best of Pandora's features are hidden behind a $10/month paywall [4].

Spotify was originally designed to be a music player based on a large music library with a "Discover Weekly" playlist on the side. Spotify's users loved the Discover Weekly playlist feature so much that, in recent years, Spotify has readjusted its focus to algorithms identifying similar songs to be used in the playlist. Spotify's algorithm relies on three models: collaborative filtering models, natural language processing models, and audio models [5].

Collaborative filtering models takes into account the feedback of multiple users in selecting a song to recommend. This is the same concept as Pandora's Music Genome Project, but instead of using feedback from experts, Spotify uses its users' play history. Basically, Spotify finds two users A and B who have listened to most of the same songs. It then takes songs only user A has played and recommends them to user B, and vice versa [6].

Natural language processing involves the computer interpreting human language to identify some feature. Using a proprietary algorithm, Spotify comes up with a daily-updated list of thousands of terms for each song and artist. The terms are weighted by how much they describe the song. Spotify then uses the same sort of comparison as in a collaborative filtering model to recommend songs with similar term weights [6].

Spotify's third major model focuses on analyzing the audio of a song. This is especially important because it allows new, "undiscovered" songs to enter the mainstream. Spotify uses a convolutional neural network to estimate features like time signature, key, tempo, and loudness, allowing new songs to be compared to older songs [6].

Unlike Pandora, Spotify's playlist recommendations are available to users for free. But, like Pandora, Spotify hides many of its best features for those purchasing a premium membership, requiring users to pay $10/month to access the features giving the platform an acceptable user experience [7]. However, Spotify is widely considered to have the best music recommendation algorithms in the industry, so a lot can still be learned by studying their methods.

Although Apple Music has a music library 33% larger than Spotify's, most users agree that Spotify's music recommendation capabilities far outstrip those of Apple Music, allowing Spotify to actively compete with Apple Music for the largest market share. However, Apple Music is able to maintain their current lead by utilizing exclusive deals with several uber-popular artists [8]. Unlike Pandora and Spotify, Apple Music doesn't offer any free listening options [9].

TABLE I
COMPARISON OF SELECTED STREAMING SERVICES

Platform	Users	Library size	Premium rate	Free version
Pandora	31.5 M	30 M	$10/mo	Yes
Spotify	44.2 M	35 M	$10/mo	Yes
Apple Music	44.5 M	45 M	$10/mo	No

[a]Number of users taken from [10]

B. Survey of Music Emotion Annotation Literature

To research previous work in the field of music emotion annotation, we read the relevant sources cited in "A Survey of Audio-Based Music Classification and Annotation" by Fu, Lu, Ting, and Zhang from 2011 [11]. Some of the notable contributions are briefly described below.

Li and Ogihara [12] developed several classifiers of genre, emotion, style, and similarity. A three-axis binary classifier was used to identify song emotion. The three axes were from relaxing to exciting, comforting to disturbing, and cheerful to depressing. These axes are a modification of the well-known arousal-valence plane: the first corresponds to the arousal axis and the last two combined correspond to the valence axis. They identified emotional adjectives from audio clips and put the emotional adjectives into the three binary classes. They don't describe their classification method in much more detail, but report accuracy ranging between 69.6% and 83.3%. They attribute error to the number of classification classes to choose from. They also neglected to mention that their data set consisted only of jazz tracks labelled by just two individuals.

Han, Rho, Dannenberg, and Hwang [13] approached the emotion annotation problem using support vector regression. Their method consisted of three main steps: 1) extract seven quantitative features from the song, 2) map the features into eleven emotion categories on the arousal-valence plane, and 3) train two regression functions with SVR to predict arousal-valence values. Using this method, they were able to achieve 94.55% accuracy. They used a quality, diverse data set from the music database All Music Guide to train the model but did not specify what data set was used for testing [14].

Yang, Liu, and Chen [15] used a method similar to [13] but chose music categories for song segments instead of for the entire song.

Korhonen, Clausi, and Jernigan [16] again adapted the same approaches used by [13] and [15] to develop an algorithm that predicts the emotion of a piece over time, as the piece progresses. This is probably the ideal that music emotion annotation should aim for – once emotion can be reliably predicted as a function of time, software can begin to mix music just as well as making playlists.

More relevant literature is referenced below.

C. Music Emotion Annotation Methodology

As seen in the discussion of Spotify's playlist generation methodology, there are two main sets of features that can be used to decide the emotions of a track: quantitative features like key, tempo, and harmonics, which can all be measured fairly easily by neural networks, and qualitative features like emotional adjectives, which are chosen by human test subjects [17]. The emotions chosen by the algorithm to describe a song can depend on any combination of these features with varying degrees of success.

There are a series of problems to be considered when picking which features to use to optimize a playlist generation function, the greatest of which being that such a discussion begs for tests to prove hypotheses, and that no such tests have been found. First, there are over 4000 emotional adjectives in the English language, so it's just not practical to track all of them. In addition, humans tend to not assign similar adjectives to the same piece of music (e.g. a test subject might choose happy or cheerful but not both) [18]. This is an issue when a user asks for a cheerful song and doesn't get any songs labelled as happy. Both problems can be avoided by using adjective groups, like those in Fig. 1 below. By grouping together adjectives like happy, cheerful, and upbeat, the size of the database would decrease and user satisfaction would increase [20].

Second, words can be difficult to perform computations with. It's much easier if they can be represented as numbers. A popular solution is to use an arousal-valence (AV) coordinate plane to describe songs instead of emotional adjectives [21]. Using an AV plane like the one in Fig. 2 allows any song's

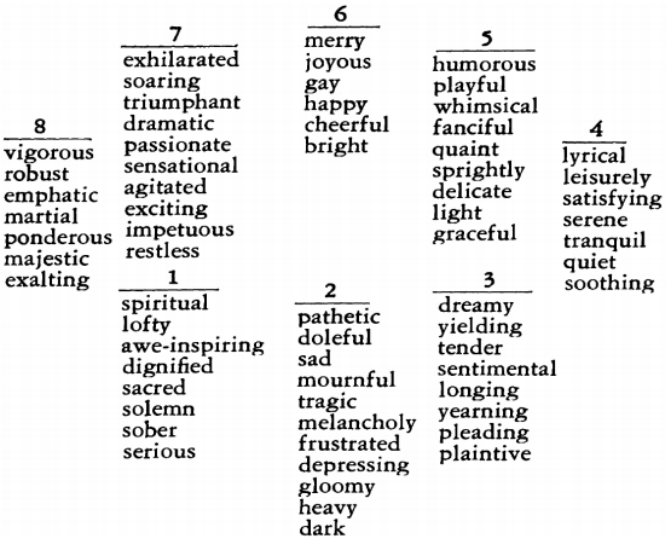

Fig. 1. Henver's adjective circle [19].

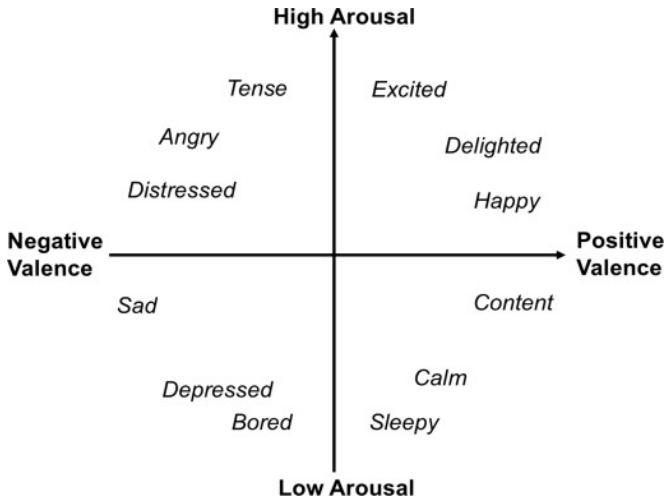

Fig. 2. Audio-valence plane.

overall emotion to be described by a single coordinate pair. This would allow for a database much smaller than one tracking individual emotions.

However, it is impractical to expect users to learn to think in terms of arousal and valence. Users will likely continue using traditional emotional adjectives. Thus it would probably be wise to train a neural network to translate emotional adjectives into the AV coordinate pair or region most statistically likely to correspond [22]. This would allow the user to input desired emotional adjectives and for the algorithm and database to reap the benefits of working only in AV coordinates.

An interesting problem is discovered when the difference in cultural music tastes is considered: every major streaming platform we have researched is focused on Western audiences. Songs that are cheerful and exciting to Westerners are likely very different from songs that elicit similar emotions in other cultures. As far as we can tell, only Spotify's algorithm would function just as well for other cultures, but Spotify's playlist generation is limited to suggested songs and doesn't attempt to pick songs from an input set of parameters.

This is an interesting problem that could potentially be solved by developing a neural network to warp the AV plane for Western music into an AV plane scaled to align more closely to another culture's taste. An alternative, possibly easier solution may be to have a new user assign adjectives to the songs in a predetermined playlist as part of the process of setting up their account. The user's adjective assignment could then be used to warp the original AV plane into a unique plane personalized for the user. This approach could allow the emotions in a song to be calculated in a method similar to Spotify's method for identifying similar songs.

III. Proposed Project

A. Description of Solution

As stated in the introduction, jTunes will be implementing 1) a personal music database, 2) basic playlist generation, and 3) a functional prototype UI. As established in the preceding Background section, the AV plane is probably the best model to use to describe a song's environmental effect. Therefore, the features used in the AV plane will also be used by the database to sort the songs, the generator to create playlists, and by the UI as parameters.

The generator will search the database for songs matching parameters input from the UI. It will arrange the songs so that the songs most closely matching the input parameters will be played first. The database storing the songs will track their generation feature values as well as generic metadata traditionally associated with music. The database relations seen in Fig. 3 will be programmatically described by Django 3.1 models, which will then be automatically built into a PostgreSQL 13 database. See Appendix A for a more detailed diagram of the database's design. A web API will be created with Python 3.8 to allow the user, via a web interface, to add music files, albums, and playlists to the database, search the database, and generate playlists from a set of chosen parameters. The user will be able to interact with the API through a UI designed in Django, using a mix of Python and the traditional HTML and CSS.

B. Delineation of Major Tasks

The following list enumerates the major tasks that make up the project. A more detailed list is included in Appendix B.

1) Design database
2) Create add song/album/playlist functions
3) Populate database
4) Create database search function
5) Create playlist generation function
6) Create retrieve music file function
7) Connect to online databases to retrieve feature values when adding a new song
8) Create basic frontend
9) Testing

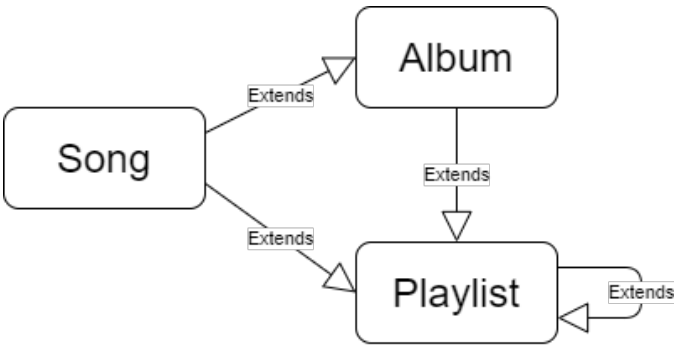

Fig. 3. Simple database schematic

10) Package deliverables

C. Final Deliverables

The end result of this project will be a zipped Python package including (1) the relevant Python helper functions, (2) Django models for generating the database, (3) the other Django components for the web frontend, (4) instructions on how to generate the database or an executable to accomplish the same end, and (5) a web page allowing the user to access the program.

D. Required Software

1) Python 3.8
2) Django 3.1
3) PostgreSQL 13
4) Windows Subsystem for Linux
5) cURL, Postman

E. Diagrams

Including use case (Fig. 4), architecture (Fig. 5), API (Fig. 6), and proposed timeline (Table II).

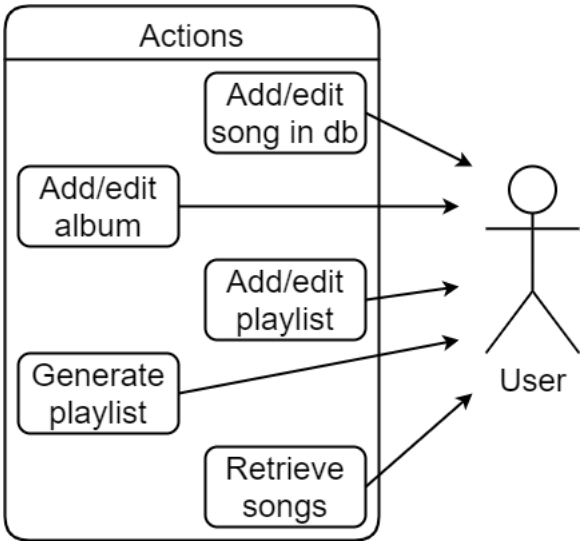

Fig. 4. Use cases diagram

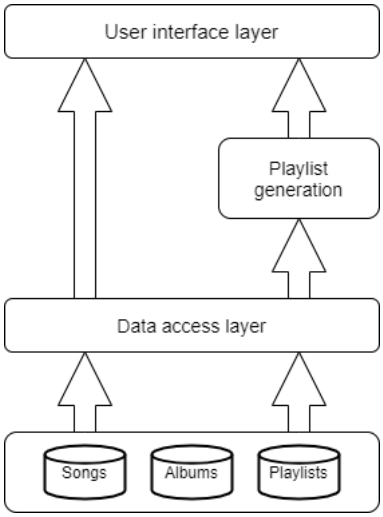

Fig. 5. Project architecture

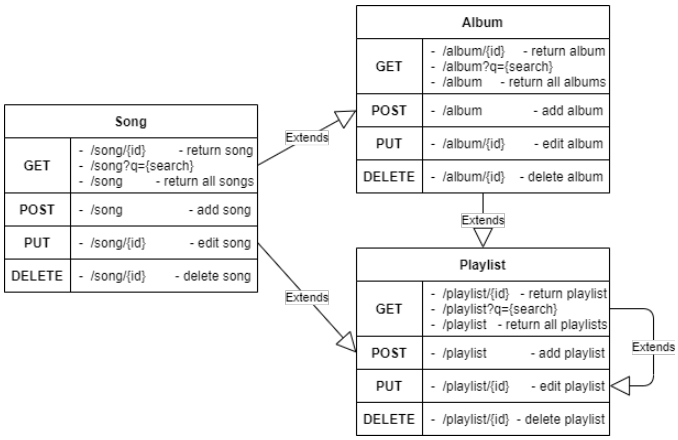

Fig. 6. API diagram

IV. Testing and Evaluation Plan

This project should enable the user to add, view, edit, and delete songs, albums, and playlists and to submit parameters to auto-generate a playlist. These tasks will be evaluated with a combination of automated unit tests and by manipulating our peers to donate their time to evaluate their ability to use the web interface. The project will be considered a success if automated test coverage is over 90% with a pass rate of 100% and if surveyed users give an average score greater than 4 on

TABLE II
Proposed Timeline

Milestone	Complete by	Estimated time (hrs)
Populated database	2/9/21	20
Functional API	3/2/21	20
Suggest feature values	3/23/21	30
Functional frontend	4/13/21	20
Testing and packaging	5/4/21	22
	Total time:	**112**

a 5-point acceptance scale.

The automated tests will involve unit tests for each piece of functionality and module tests for each target result listed in Appendix C.

All unit, module, integration, and system tests are to be written immediately after the section(s) of code they are meant to test. Acceptance testing will be performed as the final task before the project is packaged.

The project will be evaluated based on the ease with which the user is able to use the web interface to perform the actions listed in Target Results under Appendix C, specifically (1) add songs, albums, and playlists, (2) get music files from the database, and (3) auto-generate playlists, without guarantee of the aesthetic of the web interface or of the quality of auto-generated playlists. The degree to which success was achieved will be evaluated by a group of our peers who have volunteered to perform the evaluation. They will be given a survey that walks them through the tasks they should be able to complete and asks them to evaluate the ease with which they were able to do so. This survey is attached in Appendix D. The design of the web interface has not been completed, so the included survey is just a draft. The survey will be updated after the interface is completed with more complete navigational instructions.

V. Conclusion

While this project is small and does not aspire to accomplish much, it is a step in the direction of the ideal music player. It lays the groundwork for continued research on playlist generation algorithms and provides an intuitive music storage system for the user. The future of computer-assisted music consumption is very bright indeed.

Acknowledgment

The author is grateful for the invaluable assistance of Professor Robert Ordóñez in the completion of this paper.

References

[1] (2018) Pandora engineering medium blog. [Online]. Available: https://engineering.pandora.com/building-the-next-generation-of-personalized-themed-playlists-43f567b964f9

[2] (2020) Pandora review. [Online]. Available: https://www.reviews.com/entertainment/streaming/pandora-review/

[3] W. T. Glaser, T. B. Westergren, J. P. Stearns, and J. M. Kraft, "Consumer item matching method and system," U.S. Patent 7 003 515B1, Feb. 21, 2006.

[4] (2021) Upgrade to pandora premium. [Online]. Available: https://www.pandora.com/upgrade/premium

[5] (2021) Spotify vs pandora. [Online]. Available: https://www.soundguys.com/spotify-vs-pandora-36915/

[6] (2017) How does spotify know you so well? [Online]. Available: https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe

[7] (2021) Spotify premium. [Online]. Available: https://www.spotify.com/us/premium/

[8] (2021) Apple music vs spotify. [Online]. Available: https://www.soundguys.com/apple-music-vs-spotify-36833/

[9] (2021) Apple music. [Online]. Available: https://www.apple.com/apple-music/

[10] (2018) Most popular us music streaming services ranked by audience. [Online]. Available: https://www.statista.com/statistics/798125/most-popular-us-music-streaming-services-ranked-by-audience/

[11] Z. Fu, G. Lu, K. M. Ting, and D. Zhang, "A survey of audio-based music classification and annotation," *IEEE Transactions on Multimedia*, vol. 13, pp. 303–319, April 2011.

[12] T. Li and M. Ogihara, "Toward intelligent music information retrieval," *IEEE Transactions on Multimedia*, vol. 8, pp. 564–574, July 2006.

[13] B. Han, S. Rho, R. Dannenberg, and E. Hwang, "Smers: Music emotion recognition using support vector regression," in *10th International Society for Music Information Retrieval Conference*, 2009, pp. 651–656.

[14] (2021) Allmusic. [Online]. Available: https://www.allmusic.com/

[15] Y. Yang, C. Liu, and H. H. Chen, "Music emotion classification: a fuzzy approach," in *Proceedings of the 41th ACM international conference on Multimedia*, Oct. 2006, pp. 81–84.

[16] M. D. Korhonen, D. A. Clausi, and M. E. Jernigan, "Modeling emotional content of music using system identification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, pp. 588–599, June 2006.

[17] M. M. Misron, N. Rosli, N. A. Manaf, and H. A. Halim, "Music emotion classification (mec): Eploiting vocal and instrumental sound features," in *Recent Advances in Intelligent Systems and Computing*, vol. 287, 2014, pp. 539–549.

[18] L. Lu, D. Liu, and H. Zhang, "Automatic mood detection and tracking of music audio signals," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, pp. 5–18, January 2006.

[19] K. Henver, "Experimental studies of the elements of expression in music," *The American Journal of Psychology*, vol. 48, pp. 246–268, 1936.

[20] W. L. Cheung and G. Lu, "Music emotion annotation by machine learning," in *International Workshop on Multimedia Signal Processing*, Oct. 2008, pp. 580–585.

[21] R. E. Thayer, *The Biopsychology of Mood and Arousal*. Oxford, England: Oxford University Press, 1990.

[22] Y. Yang, Y. Lin, Y. Su, and H. Chen, "A regression approach to music emotion recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, pp. 448–457, February 2008.

VI. Appendix A

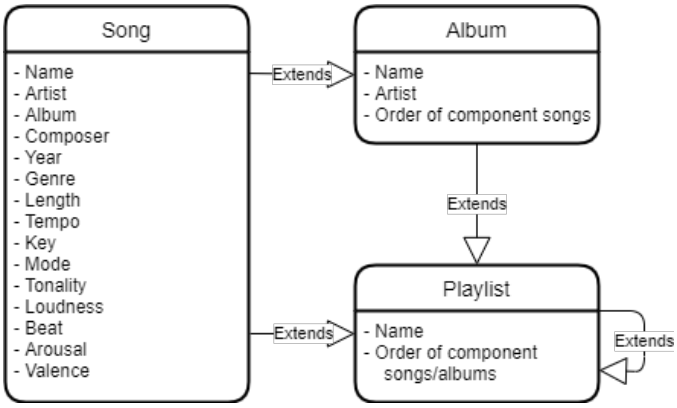

Fig. 7. Detailed database schematic

VII. Appendix B

Detailed List of Tasks with Time Estimates

1) Fully populated database (20 hours)
   a) Set up database (11 hours)
      i) Set up Django (2 hours)
         A) docs.djangoproject.com/en/3.1/intro/
      ii) Create Django models (5 hours)
         A) oodesign.com/composite-pattern.html
      iii) Set up PostgreSQL (3 hours)
         A) docs.djangoproject.com/en/3.1/ref/databases/

iv) Generate PostgreSQL database from models (1 hour)
b) Create add song/album/playlist functions (4 hours)
i) Add song function (2 hours)
ii) Add album function (1 hour)
iii) Add playlist function (1 hour)
c) Populate database (5 hours)
i) Select 100 diverse songs (0.5 hours)
ii) Auto-fill known features (1.5 hours)
iii) Manually enter unknown features (3 hours)
2) Functional API (20 hours)
a) Search database (7 hours)
i) Search songs, albums, and playlists by feature (2 hours)
ii) Search songs by similarity to audio sample (5 hours)
b) Add song/album/playlist (1 hour)
c) Playlist generation (10 hours)
i) Return string array of song names that fit within input parameters (2 hours)
ii) Experiment with an AI approach (8 hours)
d) Music file retrieval (2 hours)
3) Suggest feature values (30 hours)
a) Connect to online databases for standard features (10 hours)
i) JSTOR (4 hours)
ii) Discogs (3 hours)
iii) Spotify (3 hours)
iv) wikipedia.org/wiki/List_of_online_music_databases
b) Experiment with neural network for non-standard features (20 hours)
4) Create prototype frontend (20 hours)
a) Add song, album, playlist (6 hours)
b) View separate lists of songs, albums and playlists (2 hours)
c) View individual song, album, playlist (3 hours)
d) Edit song, album, playlist (4 hours)
e) Generate playlist (5 hours)
5) Testing (10 hours)
a) Final unit, module, integration, and system tests (6 hours)
b) Manual tests (4 hours)
6) Packaging (12 hours)
a) Create Python package of helper functions (2 hours)
b) Add Django components to Python package (4 hours)
c) Create and add database setup method (5 hours)
d) Add HTML pages and any remaining resources (1 hour)

VIII. Appendix C

Target Results

1) Add a song and its features
a) This should require uploading a music file
b) Accept, reject, or overwrite suggested feature values
i) Features including but not limited to song title, artist, genre, etc.
2) Add an album and its features
a) Accept, reject, or overwrite suggested feature values
i) Features including but not limited to playlist title, songs, albums, etc.
3) View the list of songs in the database
4) View the list of albums in the database
5) View the list of playlists in the database
6) View a song and its features
7) View an album and its features
8) View a playlist and its features
9) Edit a song and its features
10) Edit an album and its features
11) Edit a playlist and its features
12) Delete a song and its features
13) Delete an album and its features
14) Delete a playlist and its features
15) Retrieve a music file from the song view
16) Retrieve an album's component music files from the album view
17) Retrieve a playlist's component music files from the playlist view
18) Specify parameters to be used to auto-generate a playlist
19) Specify the value of parameters to be used to auto-generate a playlist

IX. Appendix D

Acceptance Testing Survey Draft

The answer scale for the following questions is 1-5.

1) Can you upload a music file?
2) Can you accept auto-filled song feature values?
3) Can you manually enter song feature values?
4) Can you create a new album?
5) Can you accept auto-filled album feature values?
6) Can you manually enter album feature values?
7) Can you create a new playlist?
8) Can you accept auto-filled playlist feature values?
9) Can you manually enter playlist feature values?
10) Can you view the features of a song?
11) Can you view the features of an album?
12) Can you view the features of a playlist?
13) If you edit the title of a song that is in an album, can you see the edit when viewing the album?
14) If you edit the title of a song that is in a playlist, can you see the edit when viewing the playlist?
15) If you edit the title of an album that is in a playlist, can you see the edit when viewing the playlist?
16) Can you view the list of songs in the database?
17) Can you view the list of albums in the database?

18) Can you view the list of playlists in the database?
19) Can you retrieve a music file when viewing the features of a song?
20) Can you retrieve the music files of the songs in an album when viewing the features of an album?
21) Can you retrieve the music files of the songs in an album when viewing the features of a playlist?