# Project 50 Music Playlist Manager

**Team with Github names:**

Badam-Osor Khaltar (badamosor)

Paul McCumber (paulmccumber)

Luke Meszar (lukemeszar)

**Vision:** Provide a platform for building and sharing music playlists.

## 1 Project Description

Sharing playlists of music between a large group of people can be difficult when everyone doesn't listen to music through the same service. This will allow people to create playlists in a generic format that can then be integrated into the users service of choice. This allows people to share their playlists with their friends who can then export it into the streaming service of their choice.

## 2 Completed Requirements

User Requirements

| ID | Description | Priority |
|---|---|---|
| UR-01 | Users can create account | High |
| UR-02 | Users can create a playlist | High |
| UR-03 | Users can search database for song title, album title, artist, and playlist name | High |
| UR-04 | Users can add content (songs, from albums or artists) to a playlist (only songs can be added) | High |
| UR-06 | Users can make playlists collaborative | Low |
| UR-07 | Users can share a playlist | High |
| UR-11 | Users can export playlists into Spotify (through a fake API backend) | High |
| UR-12 | Users can export playlists into Tidal (through a fake API backend) | High |
| UR-13 | Users can export playlists into Google Play Music (through a fake API backend) | High |

Non-Functional Requirements

| ID | Description | Priority |
|---|---|---|
| NFR-01 | Search has to be less than 20ms | High |
| NFR-03 | Should be able to expand to other music services easily | Medium |

Functional Requirements

| ID | Description | Priority |
|---|---|---|
| FR-01 | When a user creates an account, a user record has to be added to the database | High |
| FR-02 | When a user creates a playlist, a playlist record is added to the database which includes the name of the playlist and an optional description | High |
| FR-03 | There has to be a database of music for users to search that is categorized by song, album, artist, and playlist | High |
| FR-04 | When users search music database, it has to return any song, artist, or album, or playlist that matches the search query | High |
| FR-05 | When a user selects a song to add to a playlist, a list of all playlists that the user owns or can collaborate on is returned to the user | High |
| FR-06 | After selecting which playlist to add a song to, an association is created between the song and the playlist | High |
| FR-10 | When a user selects an artist, a list of their albums and songs is returned | High |
| FR-12 | When a user chooses to make a playlist collaborative, a flag is set allowing users other than the owner to edit the playlist | Low |

# 3 Incomplete Requirements

User Requirements

| ID | Description | Priority |
|---|---|---|
| UR-05 | Users can remove songs from a playlist | Medium |
| UR-08 | Users can register their Spotify account to our service | High |
| UR-09 | Users can register their Tidal account to our service | High |
| UR-10 | Users can register their Google Play Music account to our service | High |

Non-Functional Requirements

| ID | Description | Priority |
|---|---|---|
| NFR-02 | Service should support thousands of concurrent users | High |

Functional Requirements

| ID | Description | Priority |
|---|---|---|
| FR-07 | When a user selects an entire album to add to a playlist, a list of all playlists that the user owns or can collaborate on is returned to the user | High |
| FR-08 | After selecting which playlist to add an entire to, an association is created between all the songs in the album and the playlist | High |
| FR-09 | When a user chooses to view an album, all the songs on that album are listed | High |
| FR-11 | When a user removes a song from a playlist, the association between the song and playlist is removed | Medium |
| FR-13 | When a user chooses to share a playlist, a unique link has to be created, based on the playlist id. This link has to be able to direct back to the playlist | High |
| FR-14 | When a user connects their Spotify account with our service, our system will need to make a connection with Spotify to authorize the account | High |
| FR-15 | After authorizing the connection with Spotify, their Spotify credentials need to be stored with the user's record | High |
| FR-16 | When a user connects their Tidal account with our service, our system will need to make a connection with Tidal to authorize the account | High |
| FR-17 | After authorizing the connection with Tidal, their Tidal credentials need to be stored with the user's record | High |
| FR-18 | When a user connects their Google Play Music account with our service, our system will need to make a connection with Google Play Music to authorize the account | High |
| FR-19 | After authorizing the connection with Google Play Music, their Google Play Music credentials need to be stored with the user's record | High |
| FR-20 | When exporting a playlist to Spotify a new playlist has to be created for the Spotify account associated with our user | |
| FR-21 | When exporting a playlist to Spotify, each song has to be searched for using the Spotify API to get the song's id for Spotify. Then each song has to be added to the playlist that was created in FR-20 | |
| FR-22 | When exporting a playlist to Tidal a new playlist has to be created for the Tidal account associated with our user | |
| FR-23 | When exporting a playlist to Tidal, each song has to be searched for using the Tidal API to get the song's id for Tidal. Then each song has to be added to the playlist that was created in FR-22 | |
| FR-24 | When exporting a playlist to Google Play Music a new playlist has to be created for the Google Play Music account associated with our user | |

FR-25 When exporting a playlist to Google Play Music, each song has to be searched for using the Google Play Music API to get the song's id for Google Play Music. Then each song has to be added to the playlist that was created in FR-24

# 4 Class Diagrams

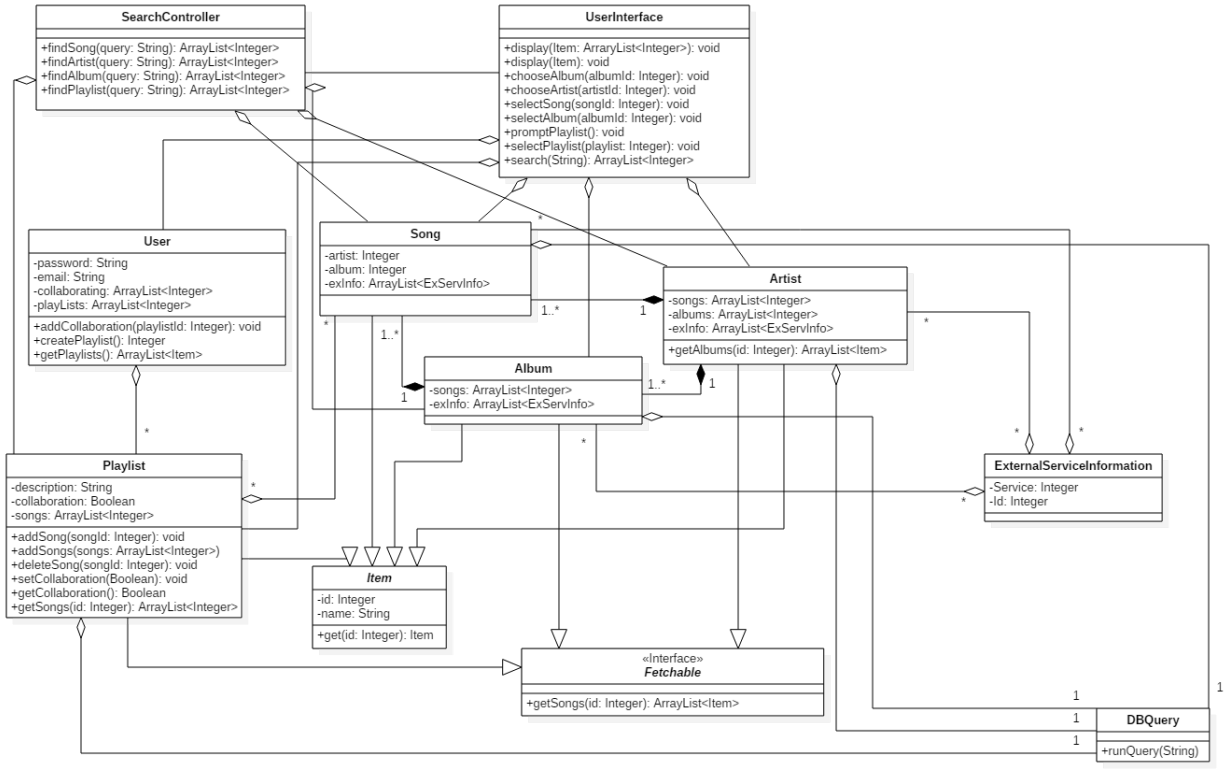


Figure 1: Class Diagram from Part 2

Our class diagram changed significantly between Part 2 and now. The fundamental reason for this is we started working with the Django framework after we finished the Part 2. Since Django had a lot of built of functionality for accessing data from a database and then displaying that data, we incorporated our ideas from Part 2 into this framework. We also updated the class diagram to reflect the design patterns we had implemented.

The class diagram in Figure 2 is entirely completed. The green classes are classes that are implemented by Django so there are no details included. The blue classes represent a larger set of classes that didn't fit in this diagram. They can be seen in detail in Figure 3 and Figure 4 respectively.

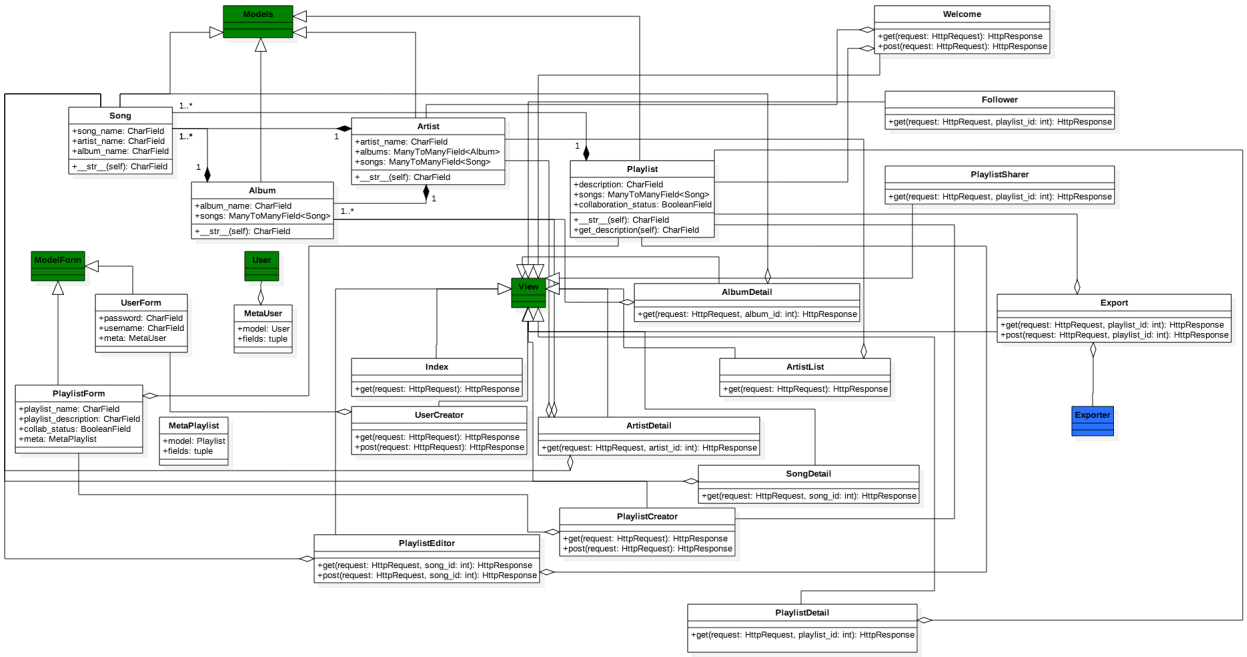


Figure 2: Class Diagram Completed

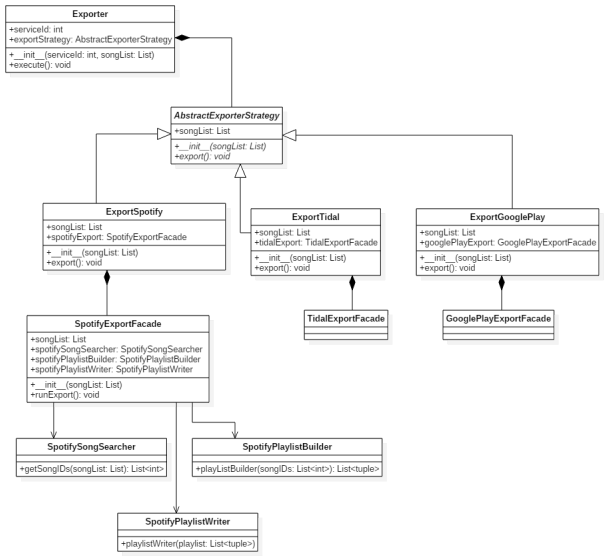


Figure 3: Class Diagram for Export

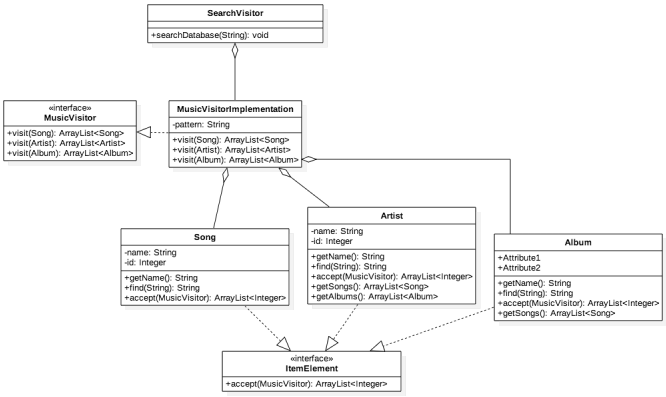


Figure 4: Class Diagram for Search

# 5 Design Patterns

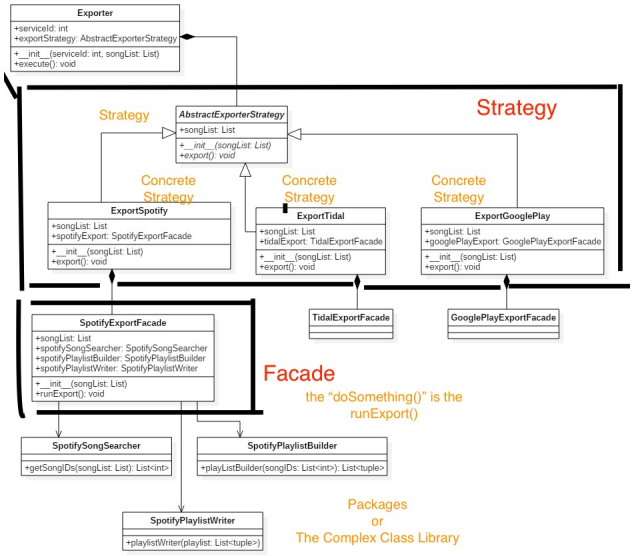


Figure 5: Class Diagram for Export with Strategy and Facade Patterns Highlighted

We used the Strategy pattern as a way to choose which external service we wanted to export to and use the appropriate algorithm for it. This was combined with the Facade pattern to handle the complexities of exporting a playlist. This allowed us to make a nice interface that didn't care about all the nitty-gritty details of exporting. This pattern would be even more important if we had worked with the actual music service API's where things would have been even more complicated.

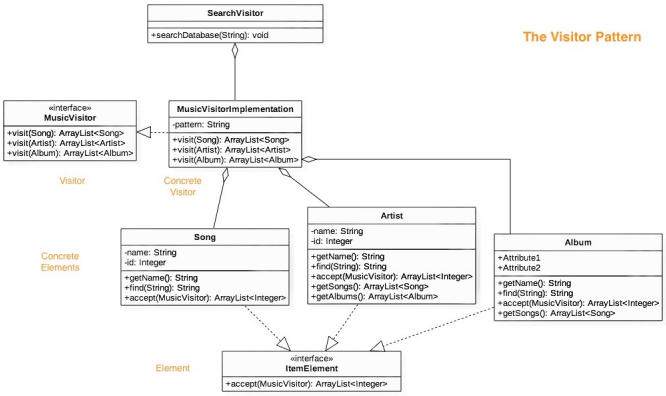


Figure 6: Class Diagram for Search with Visitor Pattern Highlighted

We decided to implement Visitor pattern for search. We created a visitor class that implements all the specialization of searching or Artist, Song, and Album data without having to modify the classes themselves (other than implementing the ItemElement interface with the accept()). The MusicVisitorImplementation instead did the searching and built the lists of matches that could be rendered to the search results page in a fashion that can be used in content editing.

# 6 What We Learned

Our choice to use the Django framework had a significant impact on our original design. Much of the work we did in Part 2 and even Part 3 ended up being thrown out. This comes from learning aspects of the MVC after we did Part 2. The framework make managing the data much different than we originally designed. We ended up having to not only change our class diagrams but much of the prototyping we had done was thrown out as well.

Even so, the work we did in Part 2 and Part 3 made the work in proceeding with Django much easier. Plus we had some tools for working with the database and ideas about the flow that had been worked out and prototyped that made implementation much easier. When converting the classes for Artist, Song, Album, and Playlist into Django Models (essentially Python representations for database objects), not a lot of thinking had to be done since we had already done it in Part 2.

Plus, everyone on the team enjoyed the opportunity of getting some exposure to an MVC. Throughout this project, we learned the importance of planning ahead.

Overall, having the design on paper before hand made planning and communicating as a team easier. It was easier to decide who was going to implement what when we had concrete ideas of what needed to be implemented based on the analysis. One of the most useful parts was the requirements analysis. Having a list of things we wanted to get done from the beginning made for a lot simpler plan going forward. It was very nice to be able to check items off a list as we completed them. Overall, doing the analysis and design made the who process a lot smoother than it would have been if you had just jumped straight in.