1 Project description

Team with Github names:

Badam-Osor Khaltar (badamosor)

Paul McCumber (paulmccumber)

Luke Meszar (lukemeszar)

Title: Music Playlist Manager

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 Class diagram

Figure 1 shows our class diagram. We fixed two things in our previous diagram based on the feedback. First, we corrected the missing data types for compositions. Second, we draw dashed line with arrow for implementing an interface.

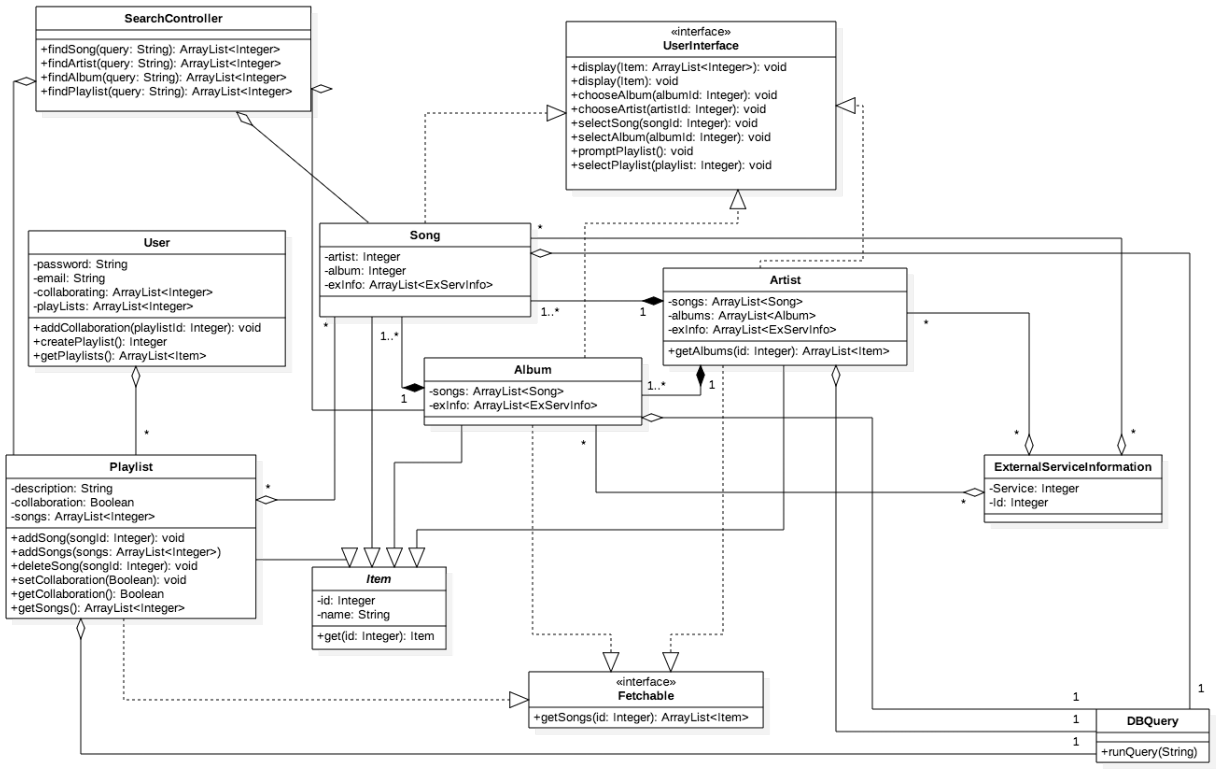

SearchController
+findSong(query: String): ArrayList<Integer>
+findArtist(query: String): ArrayList<Integer>
+findAlbum(query: String): ArrayList<Integer>
+findPlaylist(query: String): ArrayList<Integer>
«interface»
UserInterface
+display(Item: ArrayList<Integer>): void
+display(Item): void
+chooseAlbum(albumId: Integer): void
+chooseArtist(artistId: Integer): void
+selectSong(songId: Integer): void
+selectAlbum(albumId: Integer): void
+promptPlaylist(): void
+selectPlaylist(playlist: Integer): void
User
-password: String
-email: String
-collaborating: ArrayList<Integer>
-playLists: ArrayList<Integer>
+addCollaboration(playlistId: Integer): void
+createPlaylist(): Integer
+getPlaylists(): ArrayList<Item>
Song
-artist: Integer
-album: Integer
-exInfo: ArrayList<ExServInfo>
Artist
-songs: ArrayList<Song>
-albums: ArrayList<Album>
-exInfo: ArrayList<ExServInfo>
+getAlbums(id: Integer): ArrayList<Item>
Album
-songs: ArrayList<Song>
-exInfo: ArrayList<ExServInfo>
Playlist
-description: String
-collaboration: Boolean
-songs: ArrayList<Integer>
+addSong(songId: Integer): void
+addSongs(songs: ArrayList<Integer>)
+deleteSong(songId: Integer): void
+setCollaboration(Boolean): void
+getCollaboration(): Boolean
+getSongs(): ArrayList<Integer>
Item
-id: Integer
-name: String
+get(id: Integer): Item
ExternalServiceInformation
-Service: Integer
-Id: Integer
«interface»
Fetchable
+getSongs(id: Integer): ArrayList<Item>
DBQuery
+runQuery(String)
*
1..*
1
1..*
1
*
1..*
1
*
*
*
*
*
*
*
*
1
1
1
1

Figure 1: Class Diagram

3 Completed class diagram

Figure 2 shows our class diagram and in which the classes that have implemented so far are shown in blue color.

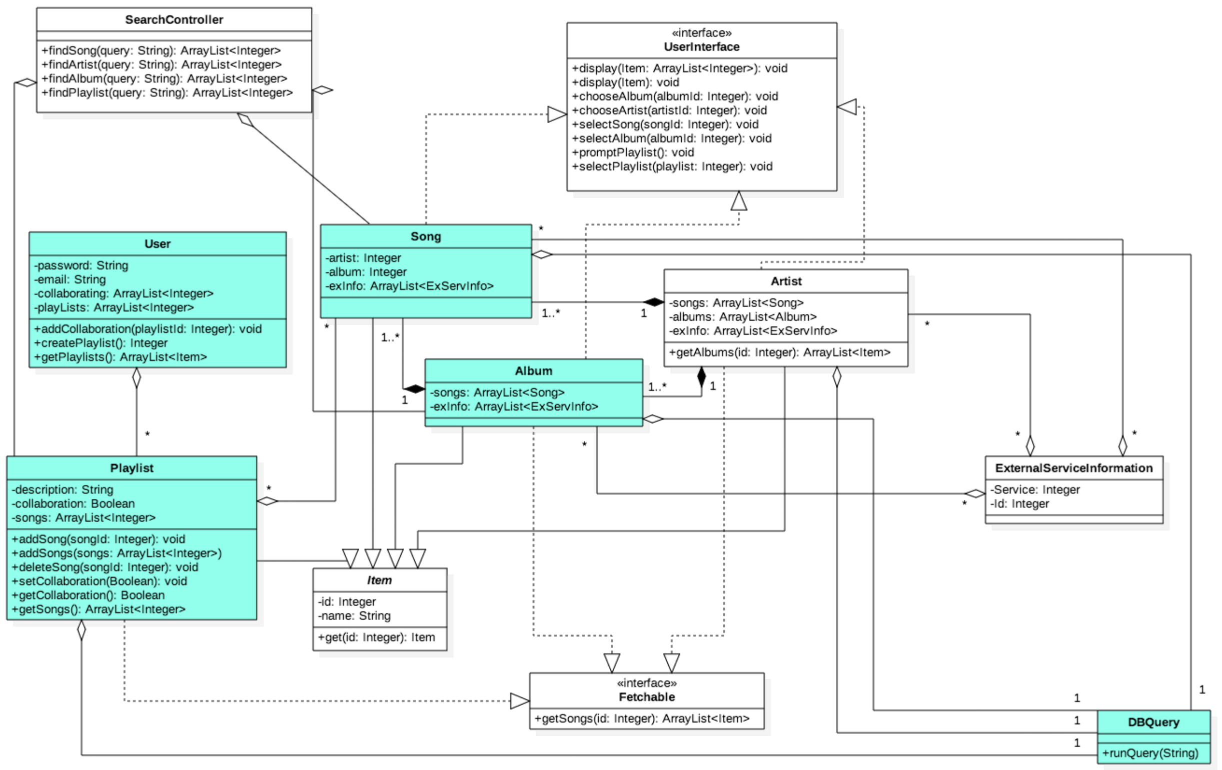

SearchController
+findSong(query: String): ArrayList<Integer>
+findArtist(query: String): ArrayList<Integer>
+findAlbum(query: String): ArrayList<Integer>
+findPlaylist(query: String): ArrayList<Integer>
«interface»
UserInterface
+display(Item: ArrayList<Integer>): void
+display(Item): void
+chooseAlbum(albumId: Integer): void
+chooseArtist(artistId: Integer): void
+selectSong(songId: Integer): void
+selectAlbum(albumId: Integer): void
+promptPlaylist(): void
+selectPlaylist(playlist: Integer): void
User
-password: String
-email: String
-collaborating: ArrayList<Integer>
-playLists: ArrayList<Integer>
+addCollaboration(playlistId: Integer): void
+createPlaylist(): Integer
+getPlaylists(): ArrayList<Item>
Song
-artist: Integer
-album: Integer
-exInfo: ArrayList<ExServInfo>
Artist
-songs: ArrayList<Song>
-albums: ArrayList<Album>
-exInfo: ArrayList<ExServInfo>
+getAlbums(id: Integer): ArrayList<Item>
Album
-songs: ArrayList<Song>
-exInfo: ArrayList<ExServInfo>
Playlist
-description: String
-collaboration: Boolean
-songs: ArrayList<Integer>
+addSong(songId: Integer): void
+addSongs(songs: ArrayList<Integer>)
+deleteSong(songId: Integer): void
+setCollaboration(Boolean): void
+getCollaboration(): Boolean
+getSongs(): ArrayList<Integer>
Item
-id: Integer
-name: String
+get(id: Integer): Item
ExternalServiceInformation
-Service: Integer
-Id: Integer
«interface»
Fetchable
+getSongs(id: Integer): ArrayList<Item>
DBQuery
+runQuery(String)
*
1..*
1
1..*
1
1..*
1
*
*
*
*
*
*
*
1
1
1
1

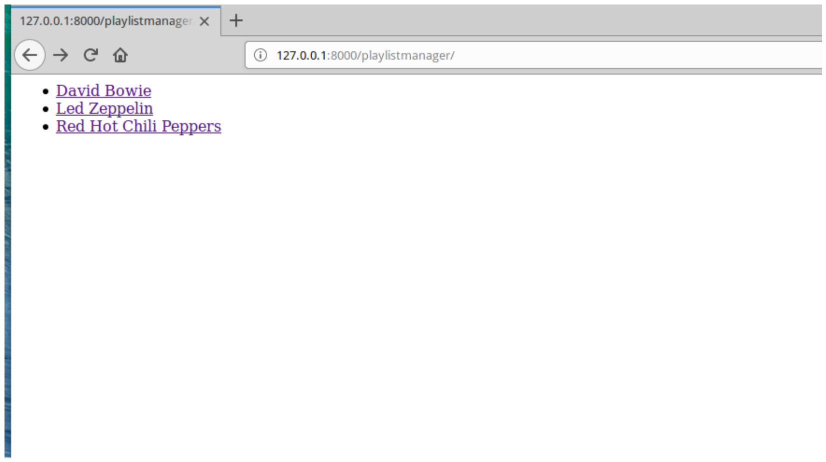
Figure 2: Class diagram

4 Summary

We accomplished following tasks since Part 2:

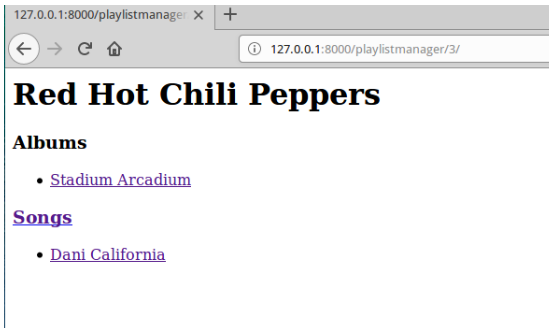
- Investigation into an MVC
  - Standardized our machines on Python 3.5.
  - Installed Django.
  - Some team members have completed the Django tutorial.
  - Work is being done working on the front end and investigating how it might integrate into our design.
  - Prototyping integration of the framework into our application.
  - Using the framework created models and views for followings:

* Album, song, and artist.
  Examples are shown in Figures 3 - 5.


127.0.0.1:8000/playlistmanager
127.0.0.1:8000/playlistmanager/

- David Bowie
- Led Zeppelin
- Red Hot Chili Peppers

Figure 3: Artists


127.0.0.1:8000/playlistmanager
127.0.0.1:8000/playlistmanager/3/

Red Hot Chili Peppers

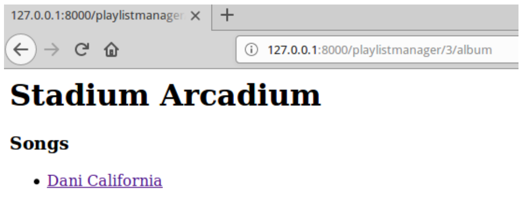
Albums

- Stadium Arcadium

Songs

- Dani California

Figure 4: Artist, Albums and Songs


127.0.0.1:8000/playlistmanager
127.0.0.1:8000/playlistmanager/3/album

Stadium Arcadium

Songs

- Dani California

Figure 5: Album and Songs

* Admin's login (Figure 6) and administration (Figure 7).

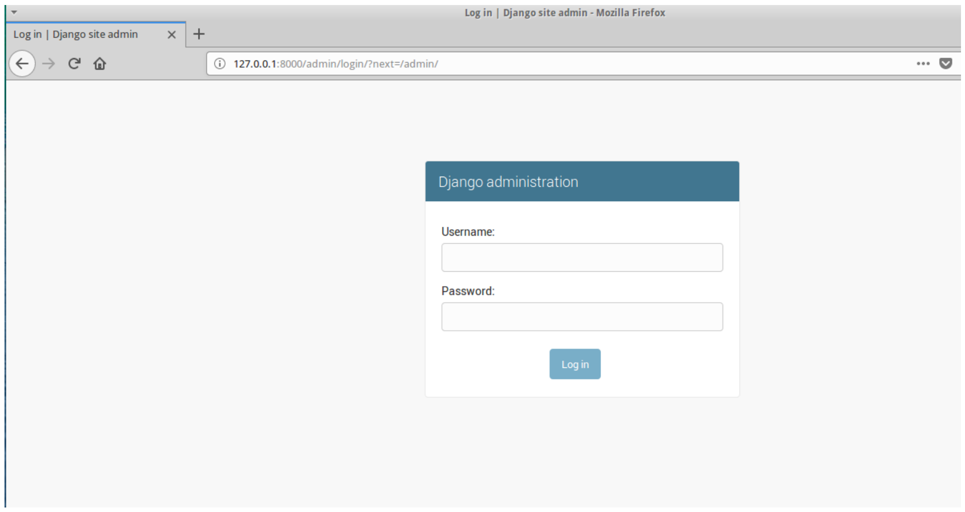
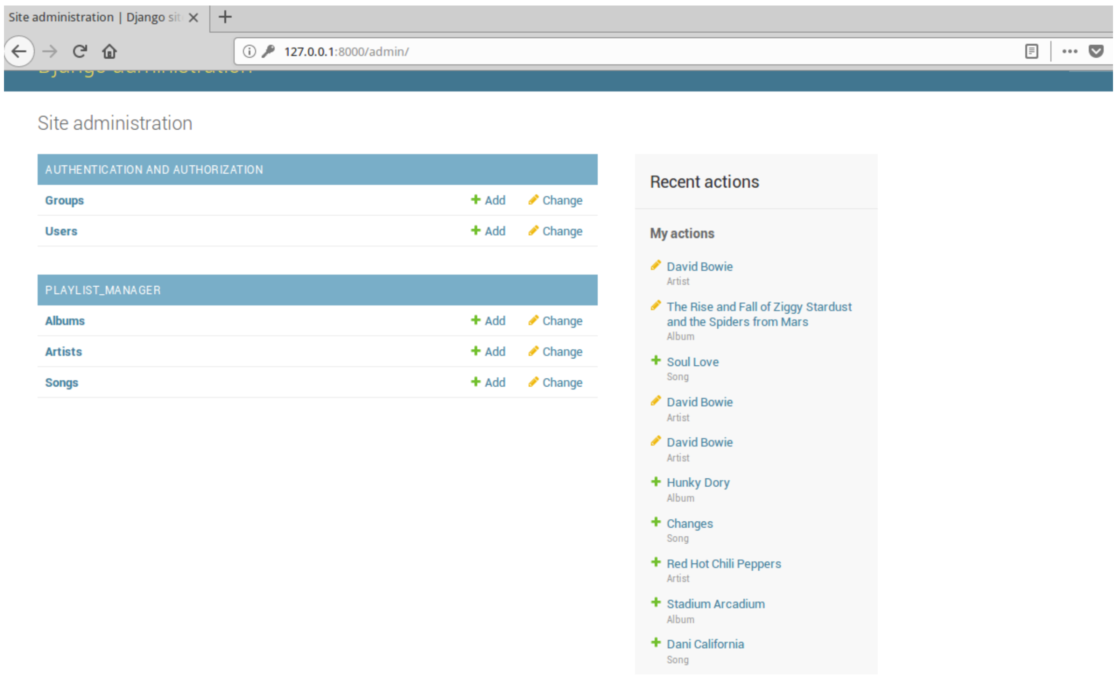

Log in | Django site admin - Mozilla Firefox
Log in | Django site admin
127.0.0.1:8000/admin/login/?next=/admin/
Django administration
Username:
Password:
Log in

Figure 6: Admin login


Site administration | Django sit

127.0.0.1:8000/admin/

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

PLAYLIST_MANAGER		
Albums	+ Add	Change
Artists	+ Add	Change
Songs	+ Add	Change

Recent actions

My actions

- David Bowie
  Artist
- The Rise and Fall of Ziggy Stardust and the Spiders from Mars
  Album
- Soul Love
  Song
- David Bowie
  Artist
- David Bowie
  Artist
- Hunky Dory
  Album
- Changes
  Song
- Red Hot Chili Peppers
  Artist
- Stadium Arcadium
  Album
- Dani California
  Song

Figure 7: Administration

    * User login (Figure 8) and password reset (Figure 9).

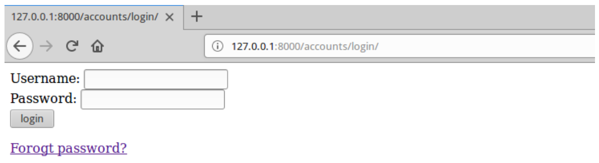

127.0.0.1:8000/accounts/login/
127.0.0.1:8000/accounts/login/
Username:
Password:
login
Forogt password?

Figure 8: User login

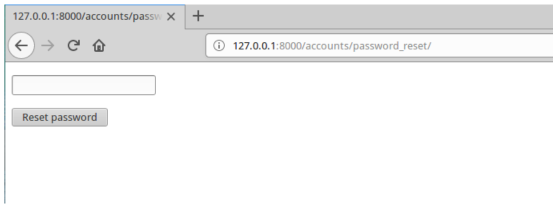

127.0.0.1:8000/accounts/passw
127.0.0.1:8000/accounts/password_reset/
Reset password

Figure 9: Reset password

- Software prototyped
  - Wrote a database interface which supports pulling items and lists of items from the database.
    * Wrote scripts to create the database and add content.
    * Wrote a handful of txt files that serve as inputs to create music content for test.
    * Using sqlite3 currently. We had intended to use MySQL but the Django tutorial was using sqlite3 out of the box and we decided that the capability was sufficient for the time being.
    * Prototyped a number of our classes (Song, Album, Artist, User, Playlist).
    * The prototypes implement getName(), getSongs() using the database definitions.
    * The prototypes implemented tests in their definitions of the classes.

        * This should serve as a framework for testing and supporting the rest of the prototyping.
        * We intend to introduce a couple design patterns and interfaces shortly.

- Updated Class Diagram
- Prepared content for Part 3

5 Breakdown

Luke:

- Took lead in Django investigation.
- Installed tools and completed the tutorial.
- Prototyping elements of the system and integration to our design.
- Completed artist, song, album class in Django with necessary views.
- Updated class diagram from part 2.
- Refactored team's GitHub repo.

Badam-Osor:

- Installed tools and completed tutorial for Django.
- Wrote the prototypes for Playlist and User.
- Working on prototyping Django front end aspects as well.
- Pulling the content and writing the Part 3 drop.

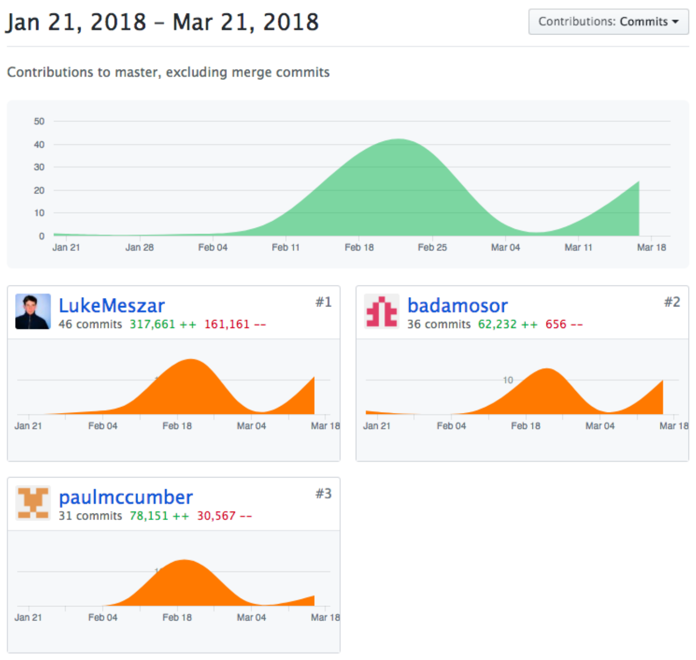
Paul:

- Installed the tools.
- Wrote scripts for creating and populating some of the database tables.
- Wrote Python utility DBQuery for interfacing to the database.

- Wrote Song, Album, and Artist prototypes.
- Wrote tests to validate current prototype status of the database and classes.

All: We had four meetings and have daily GroupMe conversations. We all consulted with our professor and TA for guidance.

6 GitHub Graph


Jan 21, 2018 – Mar 21, 2018

Contributions to master, excluding merge commits

LukeMeszar #1 46 commits 317,661 ++ 161,161 --

badamosor #2 36 commits 62,232 ++ 656 --

paulmccumber #3 31 commits 78,151 ++ 30,567 --

Figure 10: Graph of each member's contribution

7 Estimate of remaining effort

- Come to closure on how our MVC will integrate into our design. This is first priority and we need to close by the time we come back from break.

- Select a few design patterns to implement. These should be prototyped in the next 1.5 weeks as well.
  - In the near term, look at prototyping Vistor for our Search capability.
  - Adapter for interfacing to Spotify/Tidal/Google Play Music.
- Implement interfaces and relationships correctly now that nuts and bolts of our tools appear to work.
- Finish the classes. 12 hours.
- Integrate with framework. 16 hours.
- Continously test.
- Continue to update our Class Diagrams.

8 Next Iteration

We are planning following tasks to be done for the next iteration:

- Select design patterns and implement.
- Finish the classes.
- Integrate with framework.