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 Previous class diagram

Figure 1 shows our class diagram for Part 2.

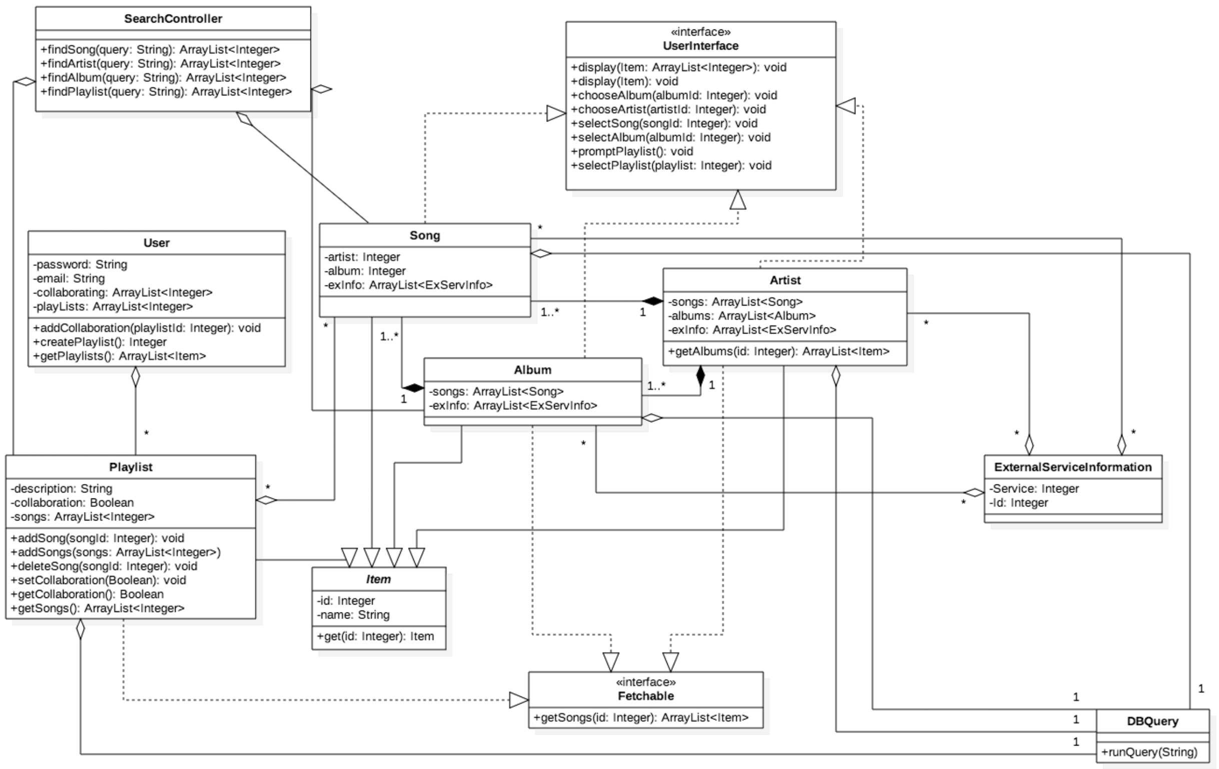

SearchController
+findSong(query: String): ArrayList<Integer>
+findArtist(query: String): ArrayList<Integer>
+findAlbum(query: String): ArrayList<Integer>
+findPlaylist(query: String): ArrayList<Integer>
«interface»
UserInterface
+display(Item: ArrayList<Integer>): void
+display(Item): void
+chooseAlbum(albumId: 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

Figure 1: Class Diagram

3 Completed class diagram

We integrated our design for Playlist manager to Django framework. Therefore our class diagram for Part 2 has been changed based on Django framework architecture and model. Figure 2 shows our new class diagram. The classes shown in green color are Django's built-in classes. The classes we implemented for our project extend these built-in classes.

The classes shown in blue color are implemented using design patterns which we're describe in detail in Section 8. Since the nature of the structure of the model was fairly fluid, we show the diagrams for these classes separately in Figures 13-12 of the Section 9. All the classes shown in Figure 2 have been implemented.

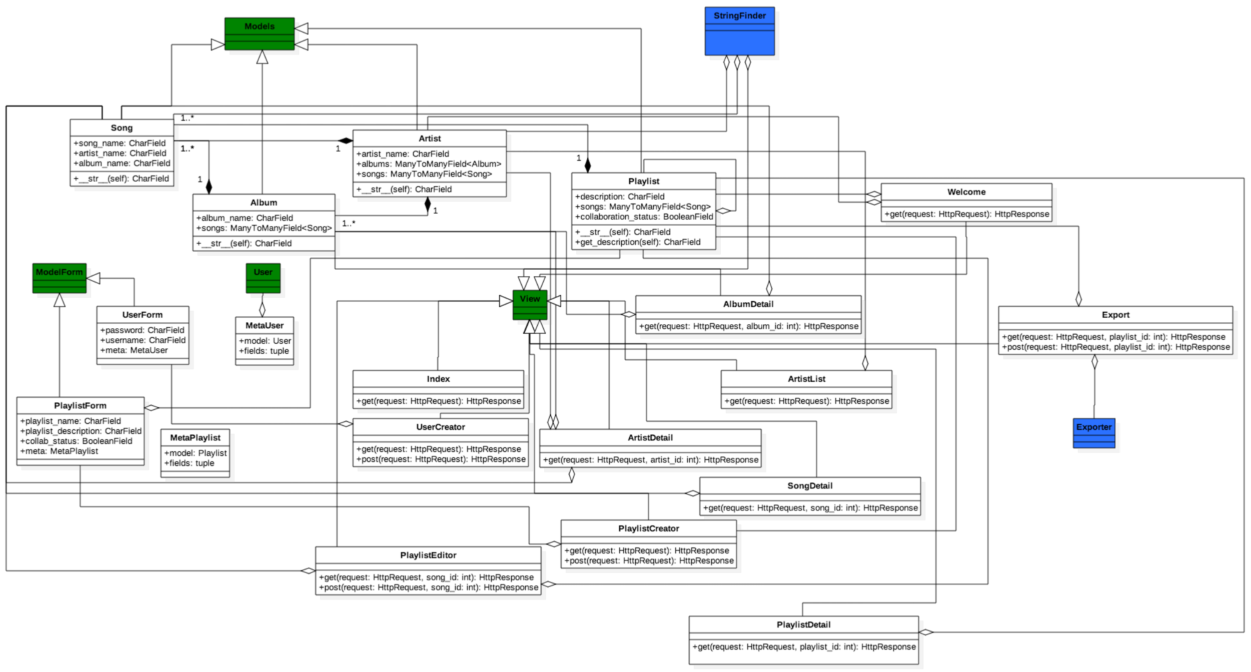
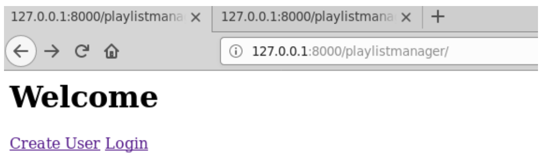

Models
StringFinder
Song
+song_name: CharField
+artist_name: CharField
+album_name: CharField
+__str__(self): CharField
1..*
1..*
1
Artist
+artist_name: CharField
+albums: ManyToManyField<Album>
+songs: ManyToManyField<Song>
+__str__(self): CharField
1
1
Album
+album_name: CharField
+songs: ManyToManyField<Song>
+__str__(self): CharField
1..*
1
Playlist
+description: CharField
+songs: ManyToManyField<Song>
+collaboration_status: BooleanField
+__str__(self): CharField
+get_description(self): CharField
Welcome
+get(request: HttpRequest): HttpResponse
ModelForm
User
UserForm
+password: CharField
+username: CharField
+meta: MetaUser
MetaUser
+model: User
+fields: tuple
View
AlbumDetail
+get(request: HttpRequest, album_id: int): HttpResponse
Export
+get(request: HttpRequest, playlist_id: int): HttpResponse
+post(request: HttpRequest, playlist_id: int): HttpResponse
Index
+get(request: HttpRequest): HttpResponse
ArtistList
+get(request: HttpRequest): HttpResponse
PlaylistForm
+playlist_name: CharField
+playlist_description: CharField
+collab_status: BooleanField
+meta: MetaPlaylist
UserCreator
+get(request: HttpRequest): HttpResponse
+post(request: HttpRequest): HttpResponse
MetaPlaylist
+model: Playlist
+fields: tuple
ArtistDetail
+get(request: HttpRequest, artist_id: int): HttpResponse
Exporter
SongDetail
+get(request: HttpRequest, song_id: int): HttpResponse
PlaylistCreator
+get(request: HttpRequest): HttpResponse
+post(request: HttpRequest): HttpResponse
PlaylistEditor
+get(request: HttpRequest, song_id: int): HttpResponse
+post(request: HttpRequest, song_id: int): HttpResponse
PlaylistDetail
+get(request: HttpRequest, playlist_id: int): HttpResponse

Figure 2: Class diagram

4 Summary

The work progressed nicely on two fronts:

- The MVC implementation took shape. A fair amount of the final functionality of the system were implemented addition to Part 2. The new functionalities include followings:
  - user registration

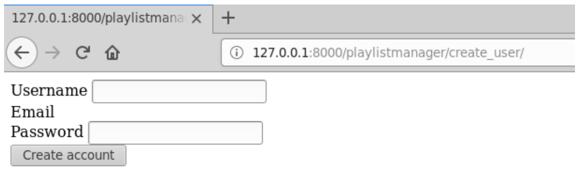


127.0.0.1:8000/playlistmana × 127.0.0.1:8000/playlistmana × +

127.0.0.1:8000/playlistmanager/

Welcome

Create User Login


127.0.0.1:8000/playlistmanager/create_user/

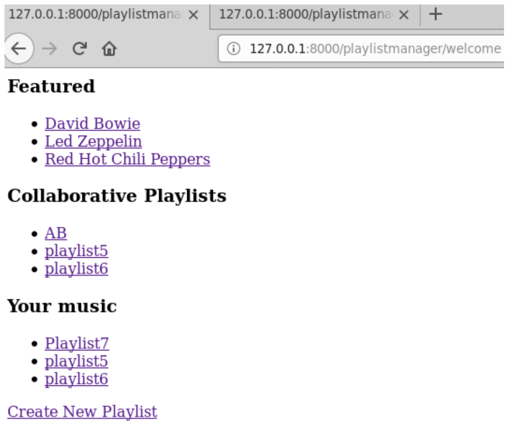
Username

Email

Password

Figure 3: Create user

– creating playlists


127.0.0.1:8000/playlistmanager/welcome

Featured

- David Bowie
- Led Zeppelin
- Red Hot Chili Peppers

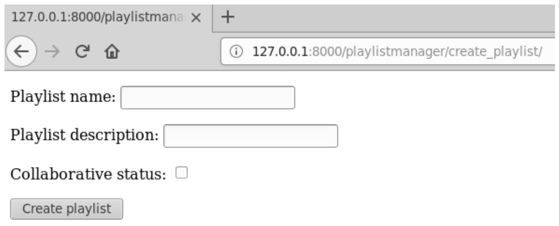
Collaborative Playlists

- AB
- playlist5
- playlist6

Your music

- Playlist7
- playlist5
- playlist6

Create New Playlist


127.0.0.1:8000/playlistmanager/create_playlist/

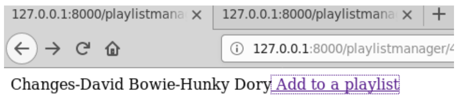
Playlist name:

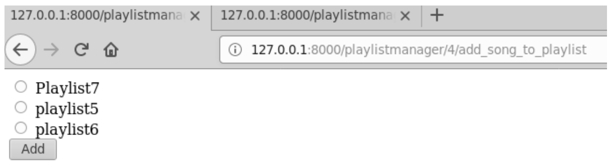
Playlist description:

Collaborative status:

Figure 4: Create playlist

– editing playlists

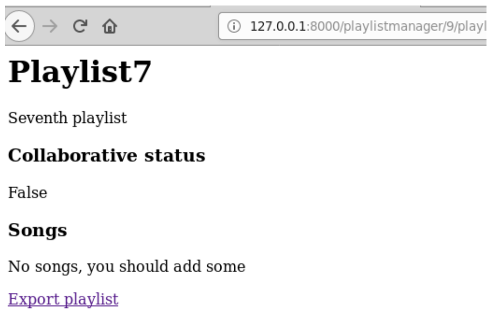

Changes-David Bowie-Hunky Dory Add to a playlist


127.0.0.1:8000/playlistmana
127.0.0.1:8000/playlistmana
127.0.0.1:8000/playlistmanager/4/add_song_to_playlist

○ Playlist7
○ playlist5
○ playlist6
Add

Figure 5: Add to playlist

- exporting to streaming service


127.0.0.1:8000/playlistmanager/9/playl

Playlist7

Seventh playlist

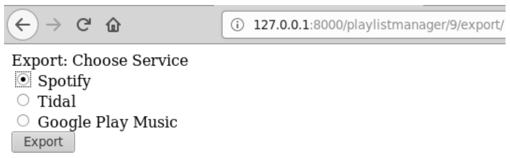
Collaborative status

False

Songs

No songs, you should add some

Export playlist


127.0.0.1:8000/playlistmanager/9/export/

Export: Choose Service
[✓ Spotify]
○ Tidal
○ Google Play Music
Export

Figure 6: Export to streaming service

- Several design patterns were prototyped including Strategy, Visitor, Observer, and Facade.

  A rough version of the Facade and Strategy were written for the Spotify interface in the model. The Strategy, Visitor, and Observer patterns were implemented and test drivers written outside of the model.

5 Breakdown

Luke:

- Much of the MVC implementation.
- Got the team rolling in the implementation.
- Redid the project class diagram based on the MVC implementation to date.
- Implemented Facade and Strategy design patterns for Exporter class.

Badam-Osor:

- Much of the MVC implementation.
- Project writeup.
- Implemented Observer pattern.

Paul:

- Implemented the Search prototype with Strategy and Visitor.
- Constructed the class diagram for Search.
- Produced writeups for Design Patterns with markup and examples of the implemented design patterns.

6 GitHub Graph

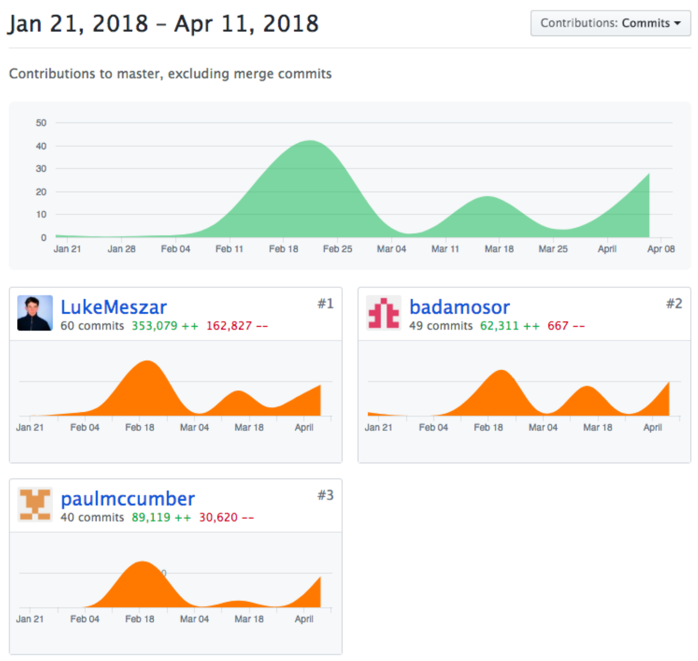

Jan 21, 2018 – Apr 11, 2018
Contributions: Commits
Contributions to master, excluding merge commits
50
40
30
20
10
0
Jan 21
Jan 28
Feb 04
Feb 11
Feb 18
Feb 25
Mar 04
Mar 11
Mar 18
Mar 25
April
Apr 08
LukeMeszar #1
60 commits 353,079 ++ 162,827 --
badamosor #2
49 commits 62,311 ++ 667 --
paulmccumber #3
40 commits 89,119 ++ 30,620 --

Figure 7: Graph of each member's contribution

7 Estimate of remaining effort

- 6-8 hours integrating the Search into the model.
- 10-20 hours integrating editing of playlists with search results.
- 3-5 hours integrating Observer design pattern into the model.
- 30 hours writing the rest of the functionalities.

8 Design Patterns

- Facade - was prototyped in the model to abstract the Spotify interface from the application.

- Strategy - was prototyped in the model as a nice way to export to one of the three streaming services dynamically.
- Visitor - We prototyped one of the search methods using the Visitor design pattern.
- Strategy - We prototyped StringFinder using Strategy and two implementations that can be dynamically switched.
- Observer - We prototyped a playlist that uses the Observer design pattern. Concretely, playlist can be collaboratively edited by its followers. If the collaborative playlist is changed, its followers get notified that a change has occurred.

9 Class diagrams for design patterns

- Figures 8-11 show the class diagram for Strategy, Visitor, Facade and Observer design patterns, respectively.

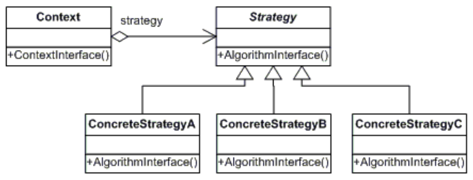


Figure 8: Class diagram of Strategy

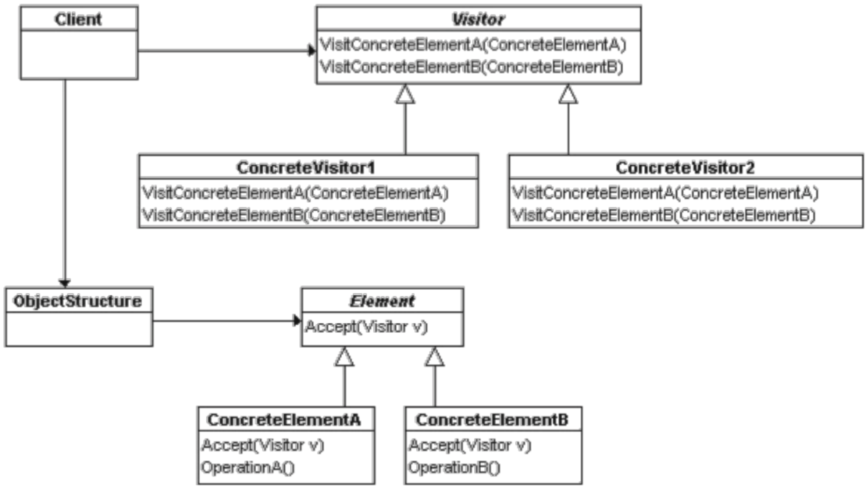
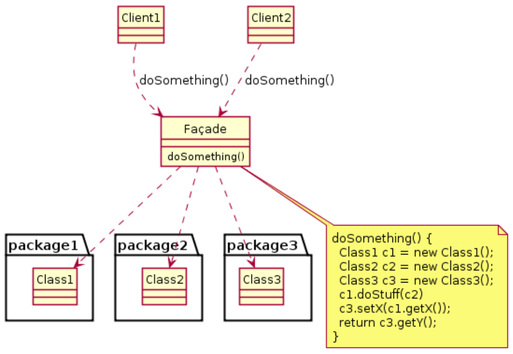

Client
Visitor
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor1
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor2
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ObjectStructure
Element
Accept(Visitor v)
ConcreteElementA
Accept(Visitor v)
OperationA()
ConcreteElementB
Accept(Visitor v)
OperationB()

Figure 9: Class diagram of Visitor



```
doSomething() {
  Class1 c1 = new Class1();
  Class2 c2 = new Class2();
  Class3 c3 = new Class3();
  c1.doStuff(c2)
  c3.setX(c1.getX());
  return c3.getY();
}
```

Figure 10: Class diagram of Facade

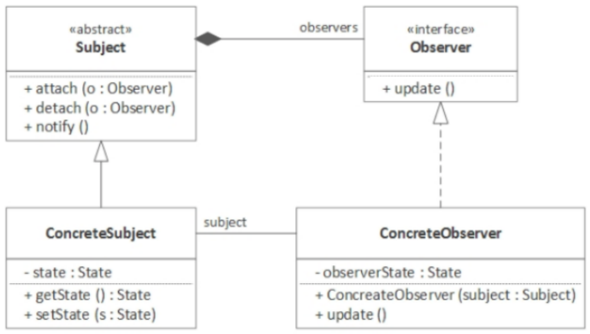


Figure 11: Class diagram of Observer

- Figures 12-14 show the class diagrams that implement above design patterns.

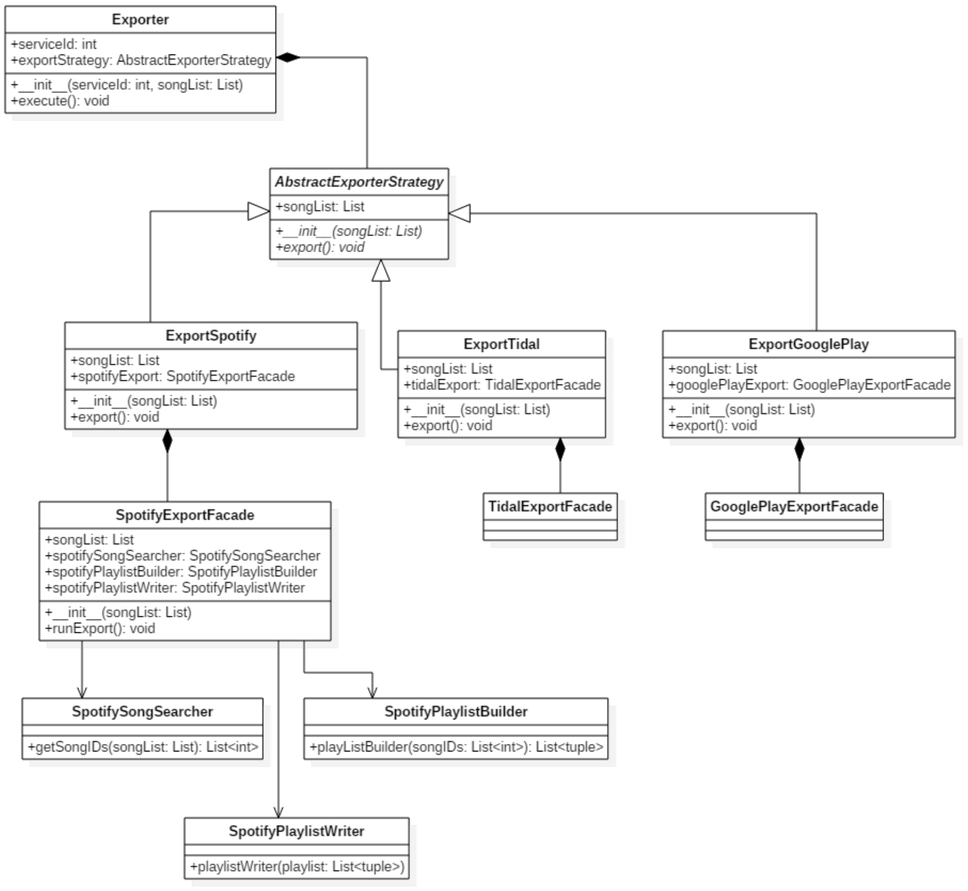

Exporter
+serviceId: int
+exportStrategy: AbstractExporterStrategy
+__init__(serviceId: int, songList: List)
+execute(): void
AbstractExporterStrategy
+songList: List
+__init__(songList: List)
+export(): void
ExportSpotify
+songList: List
+spotifyExport: SpotifyExportFacade
+__init__(songList: List)
+export(): void
ExportTidal
+songList: List
+tidalExport: TidalExportFacade
+__init__(songList: List)
+export(): void
ExportGooglePlay
+songList: List
+googlePlayExport: GooglePlayExportFacade
+__init__(songList: List)
+export(): void
SpotifyExportFacade
+songList: List
+spotifySongSearcher: SpotifySongSearcher
+spotifyPlaylistBuilder: SpotifyPlaylistBuilder
+spotifyPlaylistWriter: SpotifyPlaylistWriter
+__init__(songList: List)
+runExport(): void
TidalExportFacade
GooglePlayExportFacade
SpotifySongSearcher
+getSongIDs(songList: List): List<int>
SpotifyPlaylistBuilder
+playListBuilder(songIDs: List<int>): List<tuple>
SpotifyPlaylistWriter
+playlistWriter(playlist: List<tuple>)

Figure 12: Class diagram of Exporter

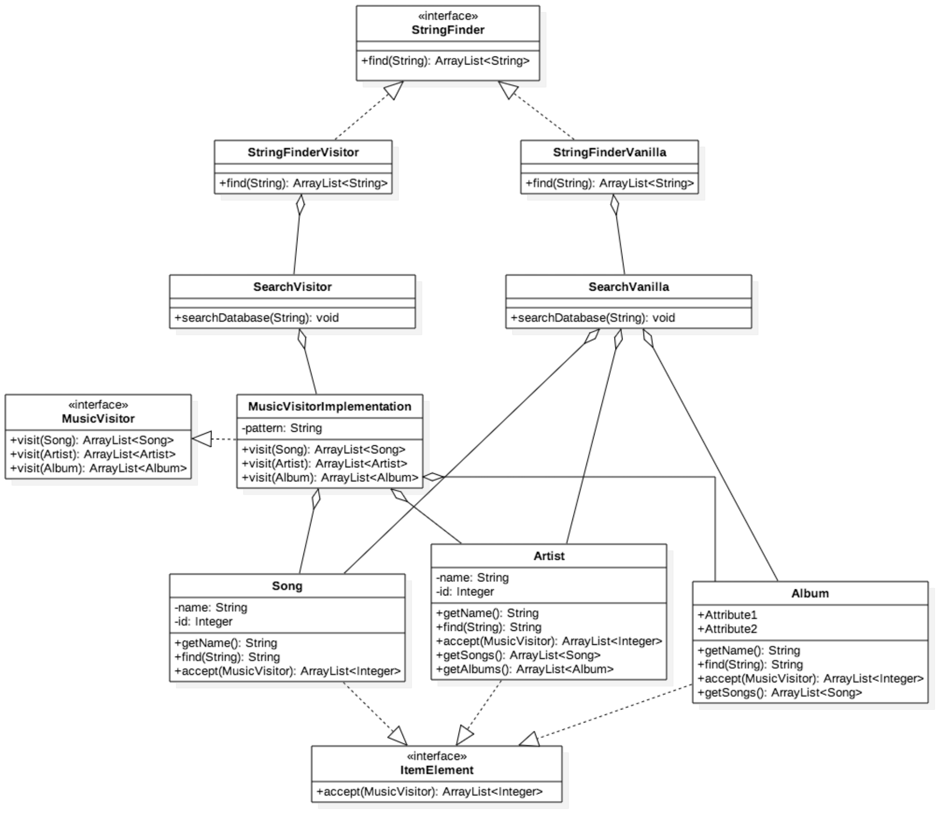

«interface»
StringFinder
+find(String): ArrayList<String>
StringFinderVisitor
+find(String): ArrayList<String>
StringFinderVanilla
+find(String): ArrayList<String>
SearchVisitor
+searchDatabase(String): void
SearchVanilla
+searchDatabase(String): void
«interface»
MusicVisitor
+visit(Song): ArrayList<Song>
+visit(Artist): ArrayList<Artist>
+visit(Album): ArrayList<Album>
MusicVisitorImplementation
-pattern: String
+visit(Song): ArrayList<Song>
+visit(Artist): ArrayList<Artist>
+visit(Album): ArrayList<Album>
Song
-name: String
-id: Integer
+getName(): String
+find(String): String
+accept(MusicVisitor): ArrayList<Integer>
Artist
-name: String
-id: Integer
+getName(): String
+find(String): String
+accept(MusicVisitor): ArrayList<Integer>
+getSongs(): ArrayList<Song>
+getAlbums(): ArrayList<Album>
Album
+Attribute1
+Attribute2
+getName(): String
+find(String): String
+accept(MusicVisitor): ArrayList<Integer>
+getSongs(): ArrayList<Song>
«interface»
ItemElement
+accept(MusicVisitor): ArrayList<Integer>

Figure 13: Class diagram of StringFinder

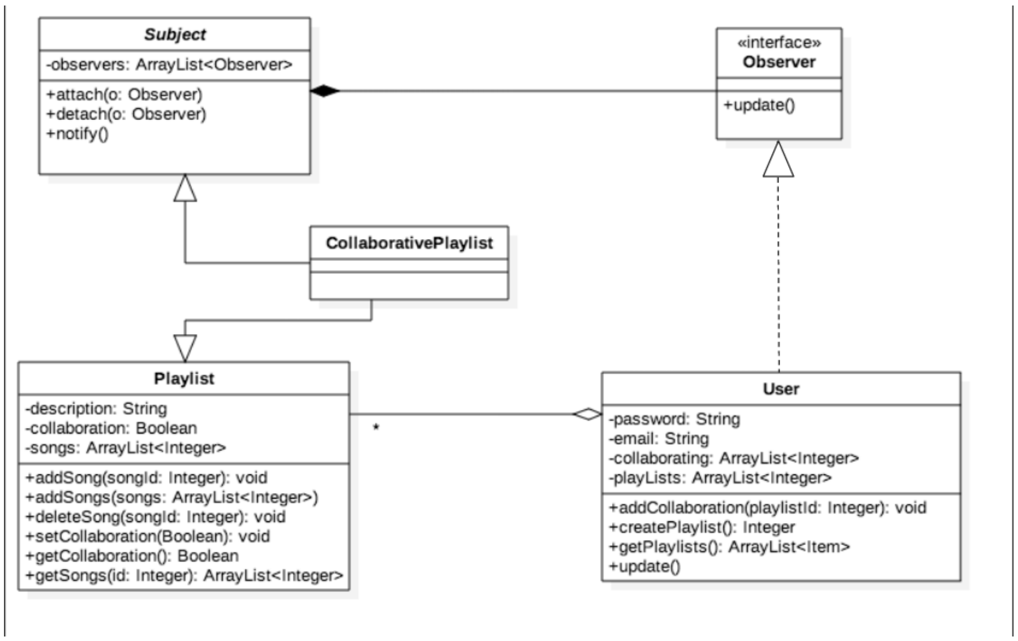

Subject
-observers: ArrayList<Observer>
+attach(o: Observer)
+detach(o: Observer)
+notify()
«interface»
Observer
+update()
CollaborativePlaylist
Playlist
-description: String
-collaboration: Boolean
-songs: ArrayList<Integer>
+addSong(songId: Integer): void
+addSongs(songs: ArrayList<Integer>)
+deleteSong(songId: Integer): void
+setCollaboration(Boolean): void
+getCollaboration(): Boolean
+getSongs(id: Integer): ArrayList<Integer>
*
User
-password: String
-email: String
-collaborating: ArrayList<Integer>
-playLists: ArrayList<Integer>
+addCollaboration(playlistId: Integer): void
+createPlaylist(): Integer
+getPlaylists(): ArrayList<Item>
+update()

Figure 14: Collaborative playlist implementation using Observer

- Figures 15-17 illustrate the participants of the diagrams shown in Figures 13-14, respectively.

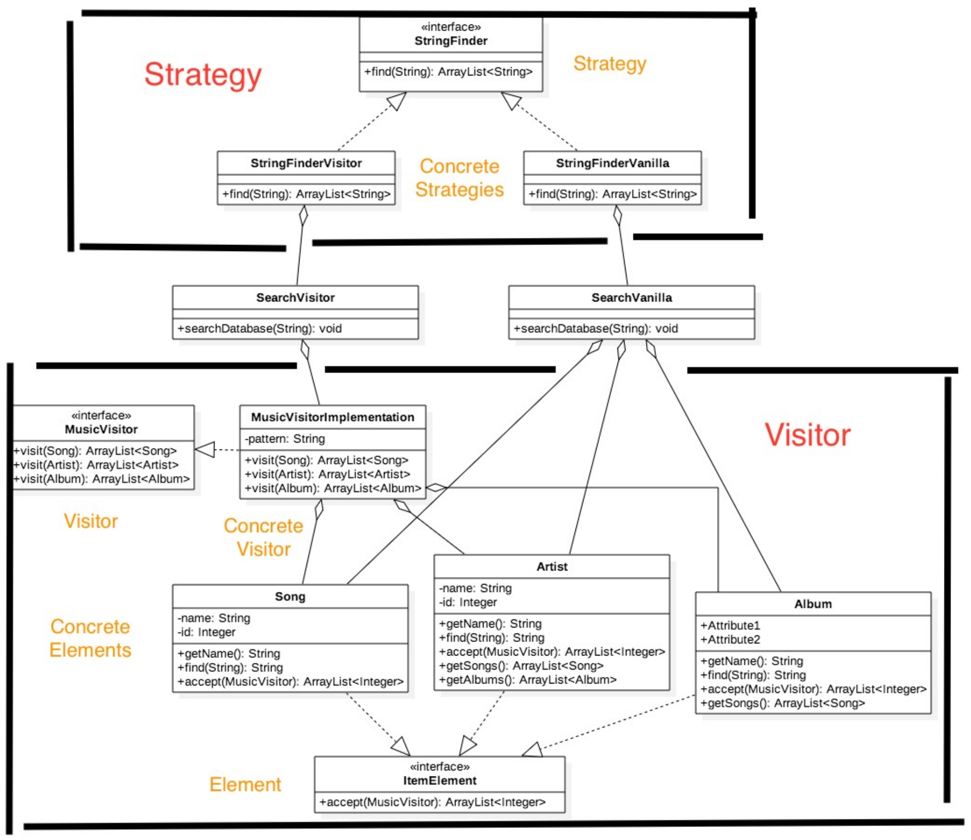

Strategy
«interface» StringFinder
+find(String): ArrayList<String>
Strategy
StringFinderVisitor
+find(String): ArrayList<String>
Concrete Strategies
StringFinderVanilla
+find(String): ArrayList<String>
SearchVisitor
+searchDatabase(String): void
SearchVanilla
+searchDatabase(String): void
«interface» MusicVisitor
+visit(Song): ArrayList<Song>
+visit(Artist): ArrayList<Artist>
+visit(Album): ArrayList<Album>
MusicVisitorImplementation
-pattern: String
+visit(Song): ArrayList<Song>
+visit(Artist): ArrayList<Artist>
+visit(Album): ArrayList<Album>
Visitor
Visitor
Concrete Visitor
Artist
-name: String
-id: Integer
+getName(): String
+find(String): String
+accept(MusicVisitor): ArrayList<Integer>
+getSongs(): ArrayList<Song>
+getAlbums(): ArrayList<Album>
Song
-name: String
-id: Integer
+getName(): String
+find(String): String
+accept(MusicVisitor): ArrayList<Integer>
Album
+Attribute1
+Attribute2
+getName(): String
+find(String): String
+accept(MusicVisitor): ArrayList<Integer>
+getSongs(): ArrayList<Song>
Concrete Elements
Element
«interface» ItemElement
+accept(MusicVisitor): ArrayList<Integer>

Figure 15: Participants of the class diagram for StringFinder

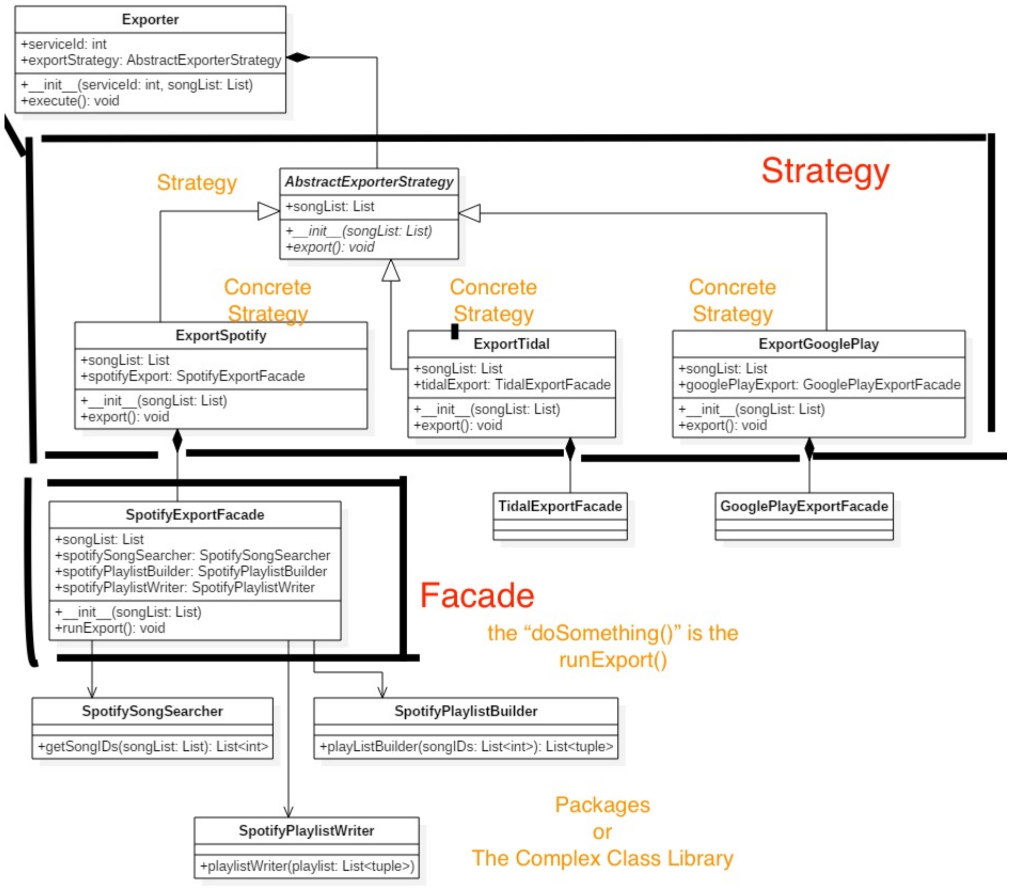

Exporter
+serviceId: int
+exportStrategy: AbstractExporterStrategy
+__init__(serviceId: int, songList: List)
+execute(): void
Strategy
AbstractExporterStrategy
+songList: List
+__init__(songList: List)
+export(): void
Strategy
Concrete Strategy
Concrete Strategy
Concrete Strategy
ExportSpotify
+songList: List
+spotifyExport: SpotifyExportFacade
+__init__(songList: List)
+export(): void
ExportTidal
+songList: List
+tidalExport: TidalExportFacade
+__init__(songList: List)
+export(): void
ExportGooglePlay
+songList: List
+googlePlayExport: GooglePlayExportFacade
+__init__(songList: List)
+export(): void
TidalExportFacade
GooglePlayExportFacade
SpotifyExportFacade
+songList: List
+spotifySongSearcher: SpotifySongSearcher
+spotifyPlaylistBuilder: SpotifyPlaylistBuilder
+spotifyPlaylistWriter: SpotifyPlaylistWriter
+__init__(songList: List)
+runExport(): void
Facade
the “doSomething()” is the runExport()
SpotifySongSearcher
+getSongIDs(songList: List): List<int>
SpotifyPlaylistBuilder
+playListBuilder(songIDs: List<int>): List<tuple>
SpotifyPlaylistWriter
+playlistWriter(playlist: List<tuple>)
Packages
or
The Complex Class Library

Figure 16: Participants of the class diagram for Exporter

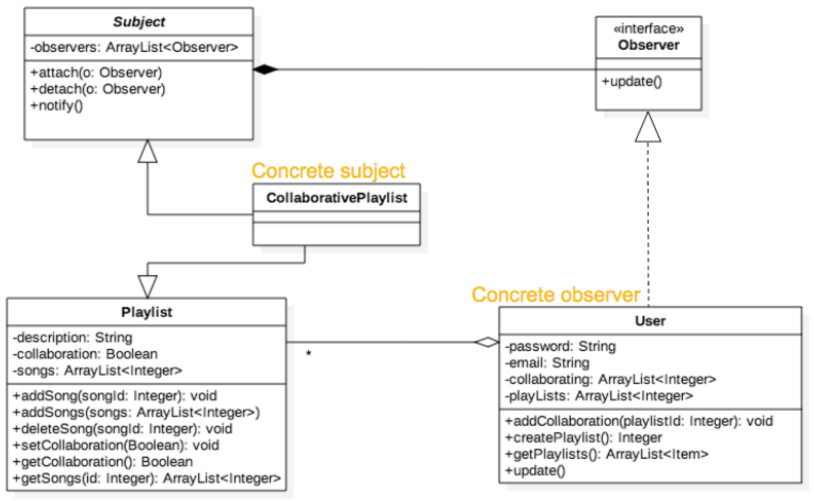

Subject
-observers: ArrayList<Observer>
+attach(o: Observer)
+detach(o: Observer)
+notify()
«interface»
Observer
+update()
Concrete subject
CollaborativePlaylist
Playlist
-description: String
-collaboration: Boolean
-songs: ArrayList<Integer>
+addSong(songId: Integer): void
+addSongs(songs: ArrayList<Integer>)
+deleteSong(songId: Integer): void
+setCollaboration(Boolean): void
+getCollaboration(): Boolean
+getSongs(id: Integer): ArrayList<Integer>
*
Concrete observer
User
-password: String
-email: String
-collaborating: ArrayList<Integer>
-playLists: ArrayList<Integer>
+addCollaboration(playlistId: Integer): void
+createPlaylist(): Integer
+getPlaylists(): ArrayList<Item>
+update()

Figure 17: Participants of the class diagram for Observer implementation

10 Final Iteration

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

- Integrating the StringFinder into the model.
- Integrating editing of playlists with search results.
- Integrating Observer design pattern into the model.
- Finish the rest of the functionalities.