Final Report

TinyTunes

Team Members: Sam Berger, Sam Busser, Jasmine Bascom

Final System State:

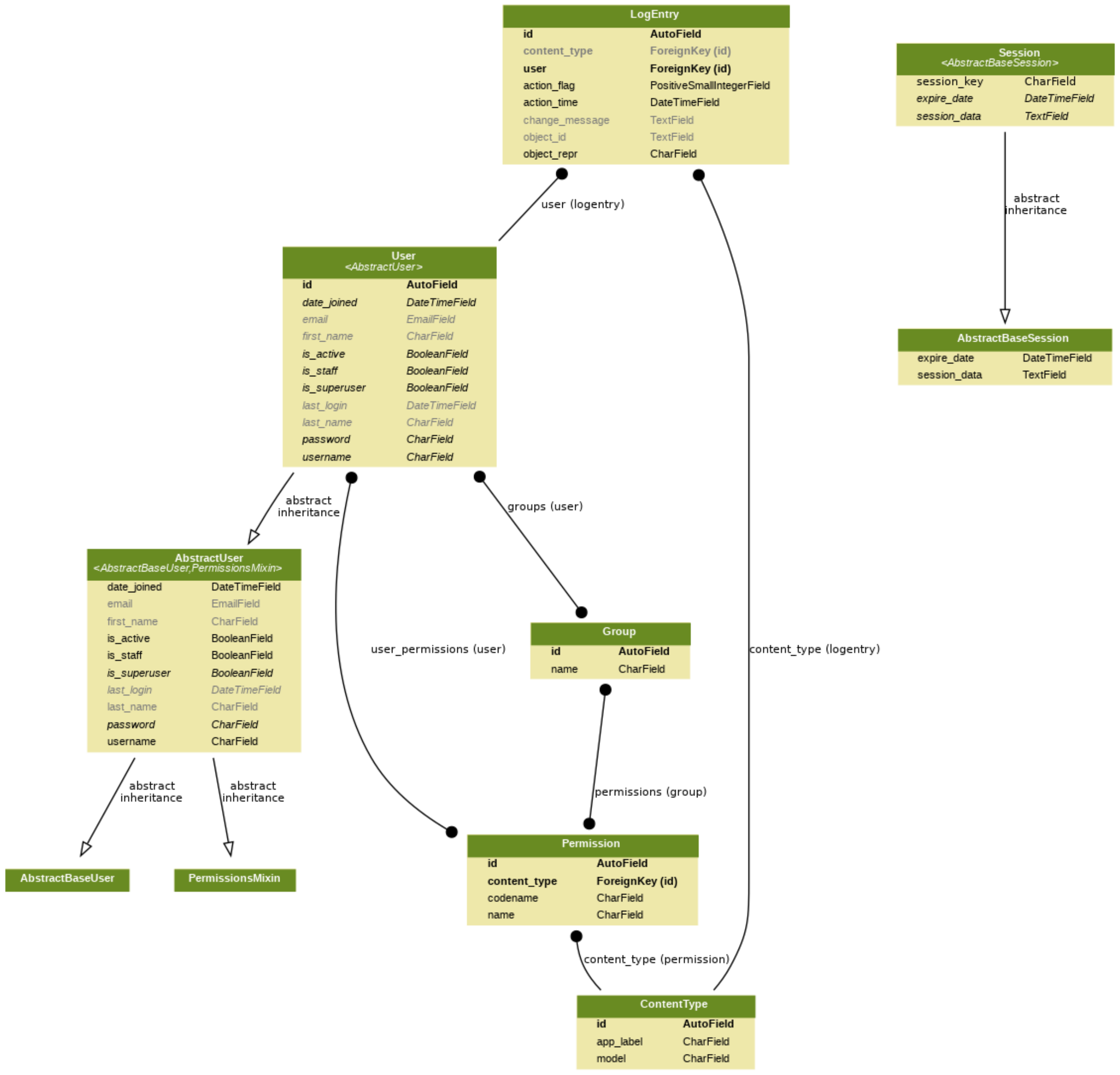
As our system stands now, it allows the user to sign in with their Spotify credentials, then give our app permission to modify their account and playlists. Then, the user is shown a list of their current playlists and is asked which playlist they would like to modify/prune. The app then takes the user to a new page showing what song has been selected to prune, shows the album cover of the song and allows the user to play a 30 second demo of it. The user is given a choice of deleting the song or not. If they choose to delete the song from the playlist, they are taken to a page confirming the song has been deleted, with a button to take them back to the home page. If they choose to not delete it, they are taken back to the list of their playlists.

List of features:

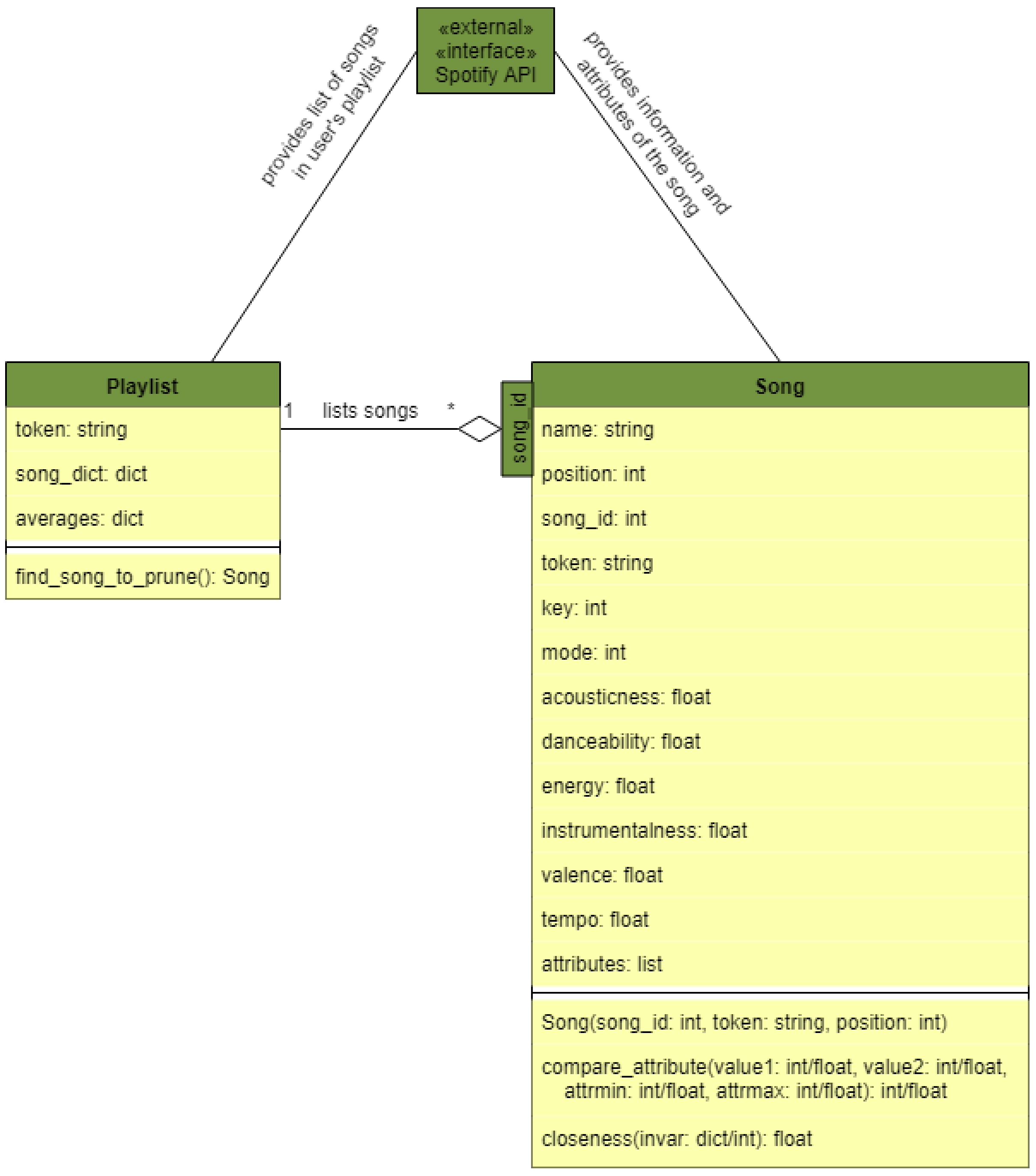
- Log in and authenticate with Spotify
- Display user's playlists
- Use algorithm to determine best song to remove from selected playlist
    - This is done by using Spotify's recommend a song feature. We call this API on the selected playlist, then use the Spotify API's in-depth song analysis to compare the recommended song to every song in the playlist. While doing this, we find the song from the playlist that has the highest numerical difference from the recommended song and choose that song as the one to prune.
- Display song that has been chosen along with album cover and 30 second sample
    - Also give user choice to remove or not remove song
- If remove button clicked, song is removed from user's playlist
- If don't remove button clicked, song is not removed from user's playlist and user taken back to playlists page.

Class Diagram:

Django class diagram:



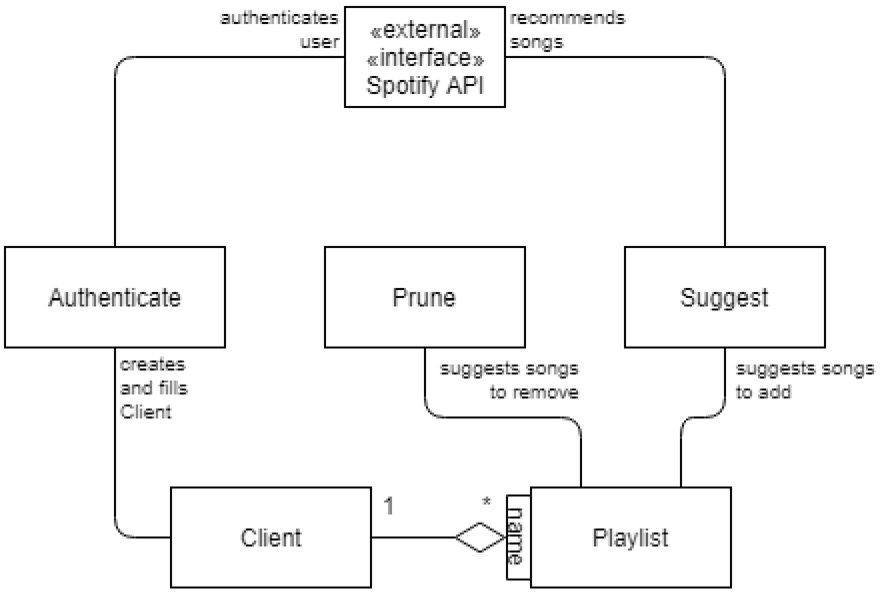
Our class diagram:



«external»
«interface»
Spotify API
provides list of songs in user's playlist
provides information and attributes of the song
Playlist
token: string
song_dict: dict
averages: dict
find_song_to_prune(): Song
1
lists songs
*
song_id
Song
name: string
position: int
song_id: int
token: string
key: int
mode: int
acousticness: float
danceability: float
energy: float
instrumentalness: float
valence: float
tempo: float
attributes: list
Song(song_id: int, token: string, position: int)
compare_attribute(value1: int/float, value2: int/float, attrmin: int/float, attrmax: int/float): int/float
closeness(invar: dict/int): float

Final Project vs. Initial Design

Homework 4 Class Diagram:



Discussion of changes:

As can be seen from our two class diagrams, our project scope has changed quite a bit. In our original design, we did not consider the challenges that would be faced with hosting our live web application. This is why our original class diagram was pretty simple. The majority of our class diagram now is taken up by simply hosting a web application and dealing with users and permissions. The Authenticate class and the Prune class as seen in our original class diagram ended up becoming Django applications as a part of our web site. These applications are required to use our HTML code along with our Python logic for the site. However, these applications are quite similar to classes, they just also include logic for how the website works. We did create the playlist class seen in our original diagram, and we created a new class, Song, as well. One other major difference between our initial and final design is the fact that we do not have the suggest feature in our final design. This is simply because of a lack of time. We are very close and have most of the pieces in place for it, but we ran out of time to do a good job on it. This is something we plan to add in the future, as we intend to work on this project outside of this class.

Third-Party Code vs. Original Code:

One major library we used in our project was the Python library Spotipy. This library is quite similar to the Spotify API and allowed us to get access to the user's playlists and list them on our select page. (https://spotipy.readthedocs.io/en/latest/). We also made use of the Spotify API and we accessed all the different functions of the API with the requests library in Python. We used this for tasks such as getting the username, getting in depth analysis of songs, getting the songs in a playlist, and deleting songs from a playlist. Requests library: https://2.python-requests.org//en/master/.

Spotify API: https://developer.spotify.com/documentation/web-api/

While we made use of these external libraries, all of the code we wrote was our own. We took what the libraries provided us and turned that into a functional application that allows users to trim down their playlists.

Design Patterns:

Due to how Django applications are laid out, we were not presented with a lot of opportunity to use design patterns. When creating Django applications, the layout of the application is generated right when you create the application, leaving little room to implement design patterns in the site. However, we were able to somewhat implement an adapter pattern in our Playlist and Song classes. In both of these classes, we are getting responses from the Spotify API, then converting them to JSON format, something that Python can understand. Once we make this conversion, we can manipulate the data and get exactly what we are looking for in each case. We are not altering any of the data in the responses, just how it is represented.

Learnings on OOAD

One of the biggest things we learned about OOAD from this project was the fact that requirements and designs are going to change as development goes on in a project. We discussed this earlier in the semester, but this project solidified this idea. We started out with a well-designed set of classes to carry out our goal for the application. But as we got started and began looking into frameworks to create and host our live site, we realized our original design was going to be tough to create. We also learned that the entire Django format is object oriented in its views and templates. Because of this, it was very easy to make changes to our site and to link different sections of our site.