Spot Dat New Playlist Final Report

Marlo Terr | Marissa Montano | Jorge Moreno

Final State of System Statement:
We were able to implement all the features we initially set out to implement. Our system has a user interface where you can login to your Spotify account. You can then see all of your playlists and choose one. If you click the picture, you can then use the sliding scales of features to customize a new playlist. Once you submit your new playlist feature preferences, you're redirected to your new playlist and you can see what tracks were added. If you select the playlist name, you are redirected to another page that shows the select playlists and the songs as well as the artists. Hovering over the playlist name, a song, or an artist name and selecting it will result in a redirect to spotify and the chosen item.

If we could add a feature, we would like to have shown the user a song's feature ratings when hovered over. This would help the user get a sense of what type of range to choose their filter criteria.

Final Class Diagram and Comparison Statement:

Comparison Statement
For our final design, we overhauled our class diagrams from project 4 and 5. At the time, we included only a class diagram of our MVC Pattern and were still unsure on what attributes and patterns we were going to be using. We did decide on using MVC, Decorator, and template pattern early on (Project 4) and fortunately kept them with our project. Below are some of the major changes for each project.

From project 4 to 5, we started working on our project and highlighted methods that still needed to be implemented. With that said, we did not include any of the other classes for our other patterns.

From project 5 to 6, we did a lot of work to add to our project. Below are the list of patterns we used and their classes. A quick description is also provided on how they work.

MVC Pattern:
- Model_Class: Grabs information from user input and callback/authentication checks
- Control_class: Grabs and makes a playlist using the spotify api. It also grabs other info like songs and artist information all from spotify api.
- Viewer (our flask server) : From the info grabbed from both control and model classes, this loads appropriate html templates
- Viewer: Also uses the decorator (form from html) to load appropriate template

Decorator Pattern:
- This uses a combination of html, javascript, jqeury, and the viewer class to work.
- Using the form, the user sets the filter they want to use and the range of their ideal song for a defined feature.
- Form gets passed to viewer

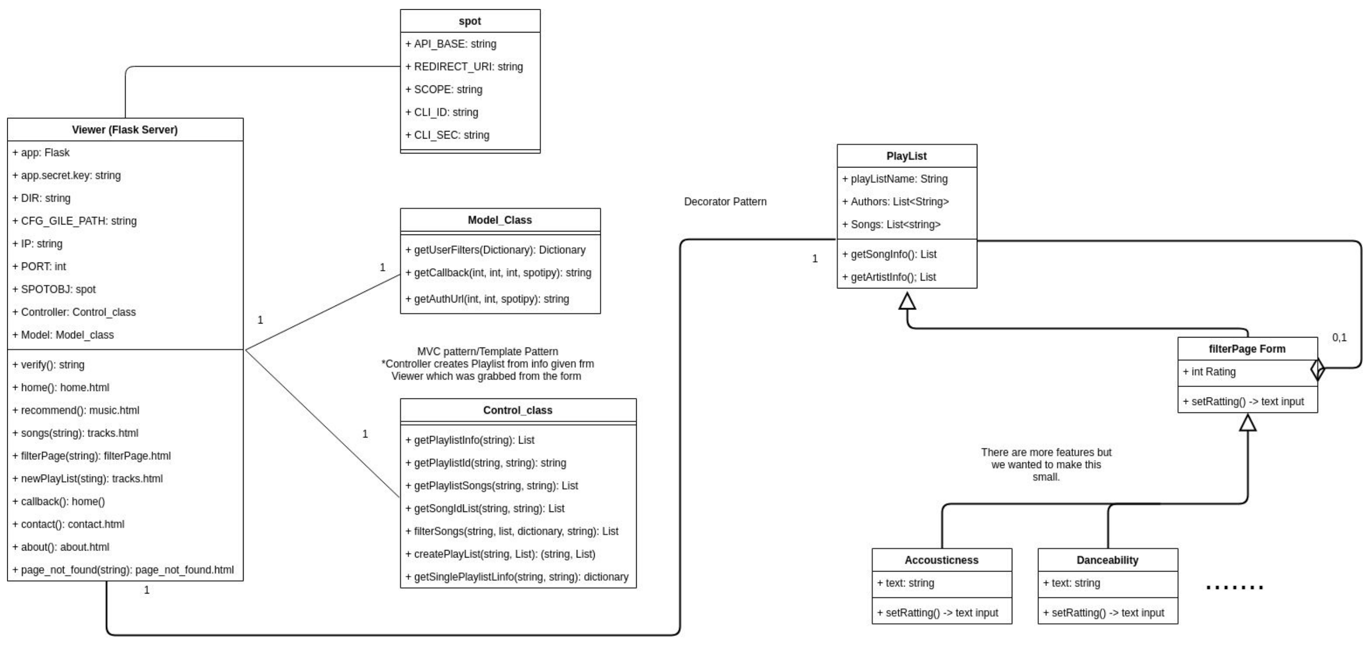
- Viewer calls Control_class with given info to make new playlist
- the controller then calls the Viewer with new info and loads the appropriate page.
- List of Features are : ["Accousticness", "Danceability", "Energy", "Instrumentalness", "Liveness", "Loudness", "speechiness", "Valence", "Tempo"]

Template pattern:

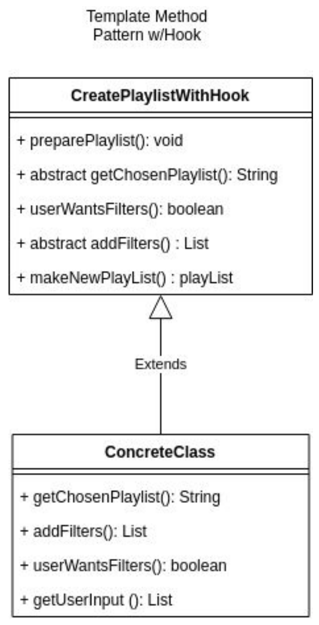
- Mixed with MVC and Decorator patterns
- As filters are selected, control creates a new playlist with new songs and passes it back to viewer
- Viewer then creates a "new" template to show new playlists from a skeleton html template provided
- User then sees new playlist with info on screen

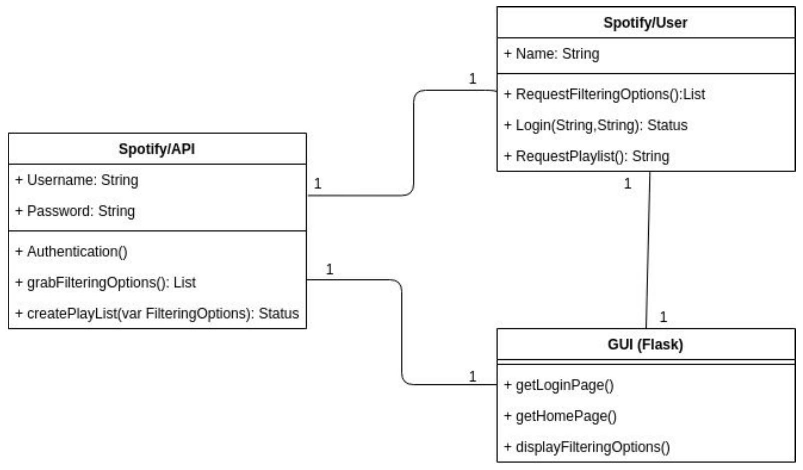
Spot Class: This class is used to store information in order to verify a user through spotify.

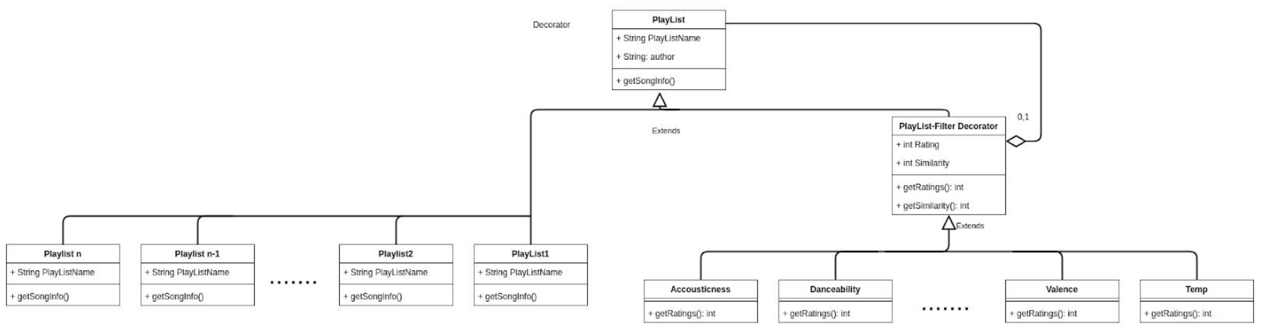
Final Class Diagram (Note attached are png for each diagram for better quality)


spot
+ API_BASE: string
+ REDIRECT_URI: string
+ SCOPE: string
+ CLI_ID: string
+ CLI_SEC: string
Viewer (Flask Server)
+ app: Flask
+ app.secret.key: string
+ DIR: string
+ CFG_GILE_PATH: string
+ IP: string
+ PORT: int
+ SPOTOBJ: spot
+ Controller: Control_class
+ Model: Model_class
+ verify(): string
+ home(): home.html
+ recommend(): music.html
+ songs(string): tracks.html
+ filterPage(string): filterPage.html
+ newPlayList(sting): tracks.html
+ callback(): home()
+ contact(): contact.html
+ about(): about.html
+ page_not_found(string): page_not_found.html
Model_Class
+ getUserFilters(Dictionary): Dictionary
+ getCallback(int, int, int, spotipy): string
+ getAuthUrl(int, int, spotipy): string
MVC pattern/Template Pattern
*Controller creates Playlist from info given frm Viewer which was grabbed from the form
Control_class
+ getPlaylistInfo(string): List
+ getPlaylistId(string, string): string
+ getPlaylistSongs(string, string): List
+ getSongIdList(string, string): List
+ filterSongs(string, list, dictionary, string): List
+ createPlayList(string, List): (string, List)
+ getSinglePlaylistLinfo(string, string): dictionary
1
1
1
1
1
Decorator Pattern
PlayList
+ playListName: String
+ Authors: List<String>
+ Songs: List<string>
+ getSongInfo(): List
+ getArtistInfo(); List
filterPage Form
+ int Rating
+ setRatting() -> text input
0,1
There are more features but we wanted to make this small.
Accousticness
+ text: string
+ setRatting() -> text input
Danceability
+ text: string
+ setRatting() -> text input
....

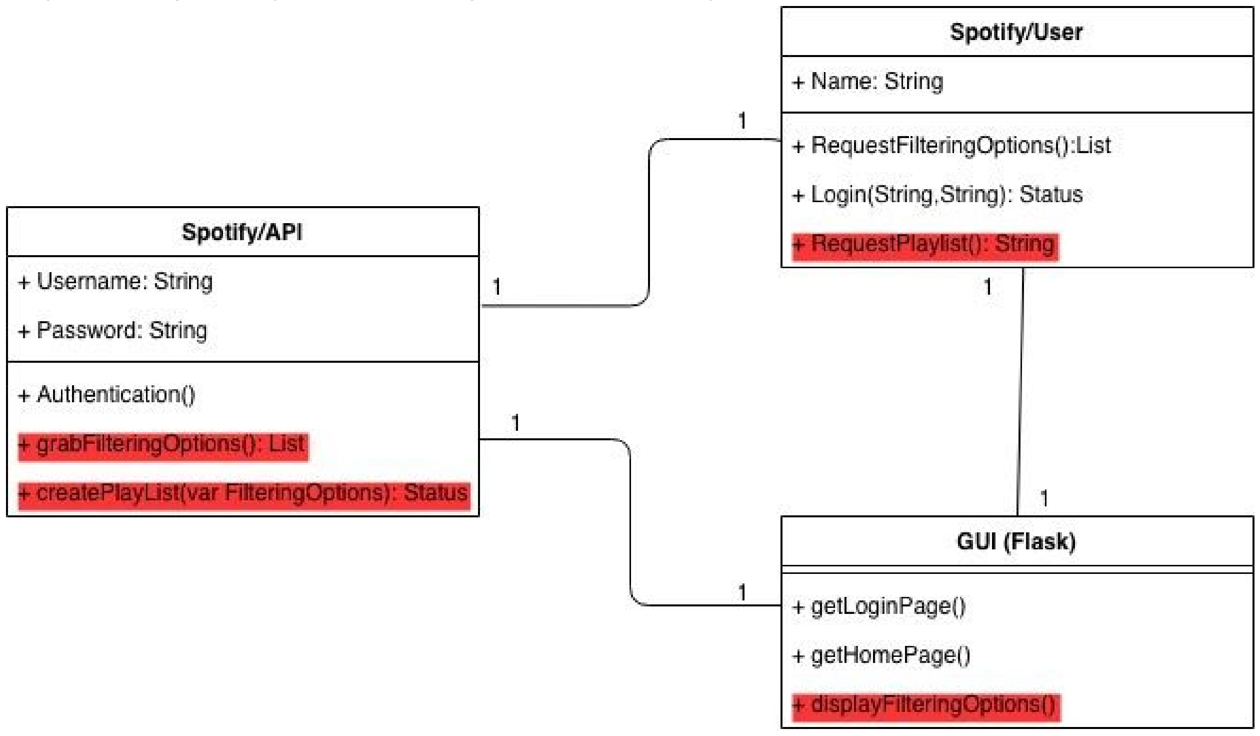
Project 4 Diagrams (left to right: Template Skeleton, MVC, Decorator Skeleton)


Template Method
Pattern w/Hook
CreatePlaylistWithHook
+ preparePlaylist(): void
+ abstract getChosenPlaylist(): String
+ userWantsFilters(): boolean
+ abstract addFilters() : List
+ makeNewPlayList() : playList
Extends
ConcreteClass
+ getChosenPlaylist(): String
+ addFilters(): List
+ userWantsFilters(): boolean
+ getUserInput (): List


Spotify/User
+ Name: String
+ RequestFilteringOptions():List
+ Login(String,String): Status
+ RequestPlaylist(): String
Spotify/API
+ Username: String
+ Password: String
+ Authentication()
+ grabFilteringOptions(): List
+ createPlayList(var FilteringOptions): Status
GUI (Flask)
+ getLoginPage()
+ getHomePage()
+ displayFilteringOptions()
1
1
1
1
1
1
1
1



Project 5 Diagrams: (MVC class diagram, red = TODO)


Spotify/User
+ Name: String
+ RequestFilteringOptions():List
+ Login(String,String): Status
+ RequestPlaylist(): String
Spotify/API
+ Username: String
+ Password: String
+ Authentication()
+ grabFilteringOptions(): List
+ createPlayList(var FilteringOptions): Status
GUI (Flask)
+ getLoginPage()
+ getHomePage()
+ displayFilteringOptions()
1
1
1
1
1
1

Third-Party code vs. Original code Statement:

- Html slider help: https://jqueryui.com/slider/#range
- Spotify Authentication help: https://stackoverflow.com/questions/57580411/storing-spotify-token-in-flask-session-using-spotipy

Statement on the OOAD process for your overall Semester Project:

1. (Issue/Obstacle) We chose a project that was hard to design in an object-oriented fashion. Our main goal was to make a functioning project and then alter it so that it was well written and had object oriented design patterns. This approach worked surprisingly well
2. (Positive) We came up with a good skeleton design for our project early on. By coming up with our 3 patterns we wanted to use, a sequence diagram, and having some templates to visualize how we wanted our project to look, we had a good foundation to work off of . With that said, implementation of our original ideas (specifically methods and interactions) were drastically changed once we started coding. We were able to overcome this by breaking our project into manageable subtasks to get done over a course of a couple of weeks each.
3. (Both positive and negative) We are all still early on in our careers and had some initial trouble with version control. With that said, we used git with separate branches to overcome this. We also communicated very well with each other when it came to bugs we found, additions that we added, or anything related to the project that was major.