Music streaming platform “Spotify”

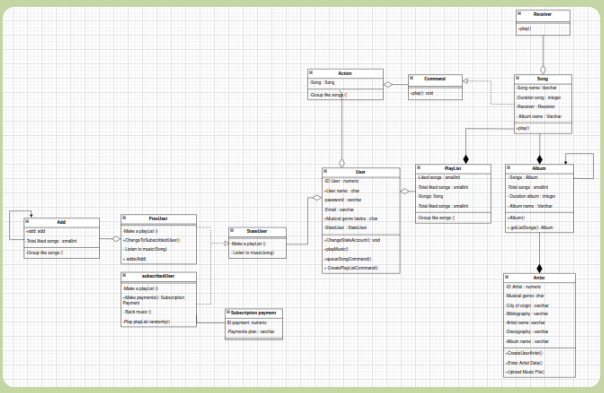
Daniela Muñoz, Juan Alvárez
Distrital University
Software modeling

Introduction

We developed a music player website with certain similarities to the Spotify music platform, which had a focus on the logic part (backEnd), considering the use of design patterns and avoiding the use of antipatterns and code smells

Metodology

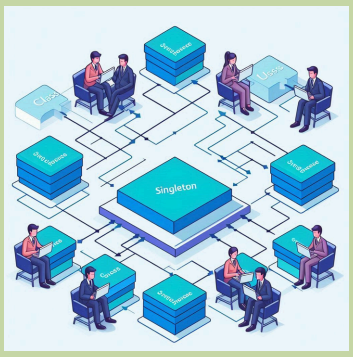
Design patterns are models that guide the search for solutions to common problems in software development and other areas of interaction or interface design.
The following were used for this project:



Creational patterns

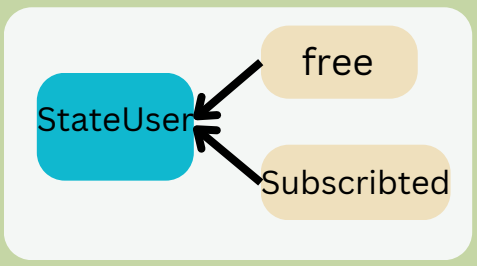
They solve problems related to the creation of object instances

the singleton pattern allows us to instantiate only once a class which does not change its state and is accessed globally. We use this pattern for the class song, ads and album, which allows several clients to access only one instance of these classes, which helps the memory usage.



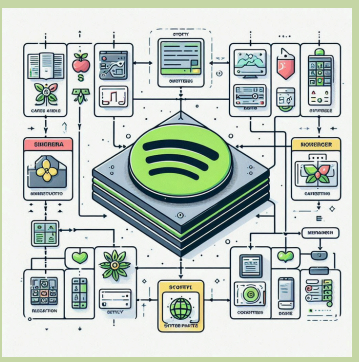
Behavioral patterns

We use the State pattern so that a user can change their status to be a free user or a subscriber. Changing their status changes their internal methods
We also use command, which allows us to encapsulate a request for some operation under a method, this is very useful for when you want to queue songs.



Structural patterns

They deal with the composition of classes and objects to form larger and more flexible structures
for our project it was not feasible to use them. That's where the idea of antipatterns came from.



Antipatterns

Antipatterns are design patterns that lead to a bad solution to a problem, that is, instead of solving a problem with a quick solution what they do is to add complexity to the code.
An example of antipattern for our project were the structure patterns, which when implemented gave us more complexity and therefore showed that it was not a viable way to implement them (Smoke and Mirrors).

code smells

A code smell is an indication that the code may have deeper problems. Its name indicates that the code smells suspicious and is not on the right track. We solve these problems by refactoring the code without changing the functionality of the code, so we have a clean program and avoid “spaghetti code”.

SOLID principles

SOLID principles and design patterns are closely related and complement each other.
Applying the State pattern respects the Single Responsibility Principle, which suggests that a class should have only one reason to change.
the singleton respects the Liskov Substitution Principle by ensuring that there is only one instance of a class in the entire system.

Results

As a result, in order to guarantee the quality of the software and the implemented logic, unit tests were performed on the system, being able to analyze the internal behavior and how the classes work. All the logic was connected to a database with a Docker.

conclusions

In the realization of this website good software quality metrics were handled, respecting aspects such as code functionality, security between classes, maintainability and performance of the code.

references

- Software Modeling. (2021). GitHub.https://github.com/EngAndres/ud-Public/tree/main/courses/software-modeling
- [Design Patterns - SourceMaking] (https://sourcemaking.com/design_patterns)