**RESPOSITORY PATTERN**

One of the design patterns used for this bot is the Repository pattern, which is utilized in the database interface. The pattern uses EntityFrameworksCore to create a way to take a class and have it act as a model for repositories. Then it uses Generic Type Parameter to handle database context builds and the actual simplifies quarying mechanism by forcing it through a template and having the compiler decide how to implement the SQL demands, with context acting as the deciding factor. Context Build maps to SQL commands such as DELETE, SELECT, ADD, and UPDATE, among others. Both synchronous and asynchronous methods were implemented to properly handle syncing, and the data layers are isolated to reduce redundant columns. The models provide entities that are strongly typed, and thus behaviours can be debugged easily when it comes to data type errors. Another one of the benefits of this pattern is that it is easy to map models to columns in the database; It is also able to build tables when not available using the EntityFrameworks code and utilizing the models built using that repository pattern.

**WRAPPER PATTERN**

The other design pattern used is the C# wrapper, which is used as a base model for the whole entire project. It provides access to the Discord API using C#, allowing for code written by the developers to use the returned data from the API. This is used to communicate between the Discord client and the bot. DSharpPlus is the wrapper used so it provides the interfaces and a bit more to help the developer just go right into implementing without worrying about