Design

Here I will perform a general review and explanation of how and why my program works in the way it does. It will be broken into two main sections, the Web API section and the Discord+Twitch Bot section.   
The majority of objectives have been met, however some weren’t done in the most effective manner. But I have tried my best to make code as efficient and effective as possible.

Web-API

The web-api is necessary to allow for bots, other than our bot, to manipulate and view currency data. It will also allow us to create a web site, where currency creator can configure their currency and the viewing of leader boards, for viewers to see their ranking inside of the currency in terms of balance and watch time.  
Without the web-api we wouldn’t be able to serve data to these web pages and it would also prevent the introduction of custom bots.  
It would’ve been possible to merge the web-api and the bot into one program, but my experience told me that this would’ve caused long hangs on both the bot and the web-api whenever the other was performing a larger operation. So by splitting them it will prevent said hangs.

The webapi composes of 2 key sections:

**The database emulation and interaction**, which composes of a set of objects designed to replicate the data stored in the database’s tables; along with the functions required to read and manipulate data in said tables. Which will allow for far simpler manipulation of the database, along with removing the need for duplicate code whenever we need to perform read/manipulate actions.

And the **web request handling**, which composes of two handler objects , one for Post requests and one for Get requests, which will take in a set of headers and/or parameters. Which will correspond to a set action; this usually entails the reading of 1 or more of the objects discussed above and/or the manipulation and saving of said objects.

Due to the web handling managing the manipulation of data in the database, the database interaction was created first, so I will start there.

Database Emulation

The emulation of the database into a set of objects allows for far more elegant integration with the data. And will also allow for us to place the sql functions inside of the objects in 2 different types; functions that fetch data will be static so they can be used without an instance of the object, And data manipulation functions, that will only be available inside of an object instance as data in the objects will be used in the sql commands.

Base Object

public class BaseObject {

//All objects will have an ID value

public int ID;

//All objects will need to be convertable into json format for transmission

public Newtonsoft.Json.Linq.JToken ToJson()

{

return Newtonsoft.Json.Linq.JToken.FromObject(this);

}

}

Due to all objects sharing an ID variable and a ToJson Function, we created a BaseObject class. Which all database objects will inherit from.

Example Object Functions

The following functions are examples of sql commands that follow a very similar structure across the different objects

The FromJSON function allows for the object to be created from a given json. Which will allow us to reconstruct the objects after receipt from the client machine.

public static Bot FromJson(Newtonsoft.Json.Linq.JToken Json)//Convert a json into a Bot object

{

return Json.ToObject<Bot>();

}