Twitch-Discord-Reward-API/Backend/Data/Objects/BaseObject.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend.Data.Objects{ public class BaseObject { public int ID;//All objects will have an ID value public Newtonsoft.Json.Linq.JToken ToJson()//All objects will need to be convertable into json format for transmission { return Newtonsoft.Json.Linq.JToken.FromObject(this); } }}

Twitch-Discord-Reward-API/Backend/Data/Objects/Bot.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Data.OleDb;namespace Twitch\_Discord\_Reward\_API.Backend.Data.Objects{ public class Bot : BaseObject//This object will be commented as an example, the other objects follow a similar structure { public Currency Currency; //Define variables to replicate the Bot table public string AccessToken, RefreshToken, BotName; public DateTime TokenRefreshDateTime; public Login OwnerLogin; public bool IsSuperBot=false; public static Bot FromJson(Newtonsoft.Json.Linq.JToken Json)//Convert a json into a Bot object { return Json.ToObject<Bot>(); } public static Bot FromID(int ID,bool WithSecretData=false)//All Single item From functions follow a similar structure { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID",ID) };//Create a set of paramaters for the SQL query List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Bots.BotID, Bots.CurrencyID, Bots.AccessToken, Bots.TokenRefreshDateTime, Bots.RefreshToken, Bots.LoginID, Bots.IsSuperBot, Bots.BotNameFROM BotsWHERE (((Bots.BotID)=@ID));", Params);//Select table data from the table, where the BotsID matches the ID paramater if (RData.Count == 0) { return null; }//Check we have at least 1 item in the returned sql results Bot Bot = new Bot();//Create a new bot object Bot.ID = int.Parse(RData[0][0]);//Set the bots variables using the sql results if (RData[0][1] != "") { Bot.Currency = Currency.FromID(int.Parse(RData[0][1])); } if (WithSecretData)//Only add this information if WithSecretData is set to true { Bot.AccessToken = RData[0][2]; Bot.TokenRefreshDateTime = DateTime.Parse(RData[0][3]); Bot.RefreshToken = RData[0][4]; } Bot.BotName = RData[0][7]; Bot.IsSuperBot = RData[0][6] == "True"; Bot.OwnerLogin = Login.FromID(int.Parse(RData[0][5])); return Bot;//Return the bot } public static List<Bot> FromLogin(int LoginID, bool WithSecretData = false)//All List item from functions follow a similar structure too the single item functions { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("LoginID",LoginID) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Bots.BotID, Bots.CurrencyID, Bots.AccessToken, Bots.TokenRefreshDateTime, Bots.RefreshToken, Bots.LoginID, Bots.IsSuperBot, Bots.BotNameFROM BotsWHERE (((Bots.LoginID)=@LoginID));", Params); List<Bot> Bots = new List<Bot> { };//By not returning null and instead returning an empty list, we remove the necesity to check for a null object, in place of an empty list foreach (String[] Item in RData)//Instead of only creating a single object, we loop through all items in the sql results { Bot Bot = new Bot(); Bot.ID = int.Parse(Item[0]); if (Item[1] != "") { Bot.Currency = Currency.FromID(int.Parse(Item[1])); } if (WithSecretData) { Bot.AccessToken = Item[2]; Bot.TokenRefreshDateTime = DateTime.Parse(Item[3]); Bot.RefreshToken = Item[4]; Bot.IsSuperBot = Item[6] == "True"; } Bot.BotName = Item[7]; Bots.Add(Bot);//And we add each object into our list of objects } return Bots;//return the list of objects } public static List<Bot> FromCurrency(int CurrencyID,bool WithSecretData = false) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("CurrencyID",CurrencyID) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Bots.BotID, Bots.CurrencyID, Bots.AccessToken, Bots.TokenRefreshDateTime, Bots.RefreshToken, Bots.LoginID, Bots.IsSuperBot, Bots.BotNameFROM BotsWHERE (((Bots.CurrencyID)=@CurrencyID));", Params); List<Bot> Bots = new List<Bot> { }; foreach (String[] Item in RData) { Bot Bot = new Bot(); Bot.ID = int.Parse(Item[0]); Bot.OwnerLogin = Login.FromID(int.Parse(Item[5])); Bot.IsSuperBot = Item[6] == "True"; Bot.BotName = Item[7]; Bots.Add(Bot); } return Bots; } public bool Save() { this.AccessToken = Networking.TokenSystem.CreateToken(64); this.RefreshToken = Networking.TokenSystem.CreateToken(128); this.TokenRefreshDateTime = DateTime.Now; List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("LoginID",this.OwnerLogin.ID), new OleDbParameter("AccessToken",Init.ScryptEncoder.Encode(this.AccessToken)), new OleDbParameter("RefreshToken",Init.ScryptEncoder.Encode(this.RefreshToken)), new OleDbParameter("TokenRefreshDateTime",this.TokenRefreshDateTime.ToString()), new OleDbParameter("BotName",this.BotName) };//Set the sql paramaters Init.SQLi.Execute(@"INSERT INTO Bots (CurrencyID, LoginID, AccessToken, RefreshToken, TokenRefreshDateTime, BotName) VALUES (NULL, @LoginID, @AccessToken, @RefreshToken, @TokenRefreshDateTime, @BotName)", Params); //Insert the bot into the table return true; } public bool UpdateCurrency()//Change the Bots associtated currency id { if (FromID(this.ID) != null)//Check if the Bot appears in the database { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("CurrencyID",this.Currency.ID), new OleDbParameter("ID",this.ID) };//Set the sql paramaters Init.SQLi.Execute(@"UPDATE Bots SET Bots.CurrencyID = @CurrencyIDWHERE (((Bots.BotID) = @ID));", Params);//Change the CurrencyID for the BotID return true; }//Report if the currency was updated, or if it failed return false; } public bool PerformRefresh()//Refresh the Access and Refresh Tokens { if (FromID(this.ID) != null)//Check if the Bot appears in the database { this.AccessToken = Networking.TokenSystem.CreateToken(64);//Change the Access and Refresh Tokens along with the RefreshDateTime this.TokenRefreshDateTime = DateTime.Now; this.RefreshToken = Networking.TokenSystem.CreateToken(128); List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("AccessToken",Init.ScryptEncoder.Encode(this.AccessToken)), new OleDbParameter("TokenRefreshDateTime",this.TokenRefreshDateTime.ToString()), new OleDbParameter("RefreshToken",Init.ScryptEncoder.Encode(this.RefreshToken)), new OleDbParameter("ID",this.ID) };//Set the sql paramaters Init.SQLi.Execute(@"UPDATE Bots SET Bots.AccessToken = @AccessToken, Bots.TokenRefreshDateTime = @TokenRefreshDateTime, Bots.RefreshToken = @RefreshTokenWHERE (((Bots.BotID) = @ID));", Params);//Update the Access+Refresh Token and TokenRefreshDateTime for the BotID return true; }//Report if the refresh was completed successfully return false; } public void Delete() { if (FromID(this.ID) != null)//Check if the Bot appears in the database { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID", this.ID) }; Init.SQLi.Execute(@"DELETE FROM BotsWHERE (((Bots.BotID)=@ID));", Params); //Delete entry where the BotID matches } } }}

Twitch-Discord-Reward-API/Backend/Data/Objects/Currency.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Data.OleDb;namespace Twitch\_Discord\_Reward\_API.Backend.Data.Objects{ public class Currency : BaseObject { public Login OwnerLogin; public Newtonsoft.Json.Linq.JToken LoginConfig, CommandConfig; public void LoadConfigs(bool WithLogin = false)//Load the confuartion files into the bot object { //Only load the login config if WithLogin is true if (WithLogin) { LoginConfig = FileManager.ReadFile("./Data/CurrencyConfigs/" + ID + "/Login.config.json"); } CommandConfig = FileManager.ReadFile("./Data/CurrencyConfigs/" + ID + "/Command.config.json"); } public static Currency FromJson(Newtonsoft.Json.Linq.JToken Json) { return Json.ToObject<Currency>(); } public static Currency FromID(int ID) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID",ID) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Currency.CurrencyID, Currency.LoginIDFROM [Currency]WHERE (((Currency.CurrencyID)=@ID));", Params); if (RData.Count == 0) { return null; } Currency Currency = new Currency(); Currency.ID = ID; Currency.LoadConfigs(); Currency.OwnerLogin = Login.FromID(int.Parse(RData[0][1])); return Currency; } public static List<Currency> FromLogin(int UserID) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("UserID",UserID) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Currency.CurrencyID, Currency.LoginIDFROM [Currency]WHERE (((Currency.LoginID)=@UserID));", Params); List<Currency> Currencies = new List<Currency> { }; foreach (String[] Item in RData) { Currency Currency = new Currency(); Currency.ID = int.Parse(Item[0]); Currency.LoadConfigs(); Currencies.Add(Currency); } return Currencies; } public static List<Currency> All(bool WithSecretData = false) { List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Currency.CurrencyID, Currency.LoginIDFROM [Currency];"); List<Currency> Currencies = new List<Currency> { }; foreach (String[] Item in RData) { Currency Currency = new Currency(); Currency.ID = int.Parse(Item[0]); Currency.OwnerLogin = Login.FromID(int.Parse(RData[0][1])); Currency.LoadConfigs(WithSecretData); Currencies.Add(Currency); } return Currencies; } public bool Save() { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("LoginID",this.OwnerLogin.ID) }; Init.SQLi.Execute(@"INSERT INTO [Currency] (LoginID) VALUES (@LoginID)", Params); Currency C = FromLogin(this.OwnerLogin.ID).Last(); //Create a directory for the configuration files System.IO.Directory.CreateDirectory("./Data/CurrencyConfigs/" + C.ID); //Copy the example config files into the directory System.IO.File.Copy("./Data/DefaultConfigs/Command.config.json", "./Data/CurrencyConfigs/" + C.ID+ "/Command.config.json"); System.IO.File.Copy("./Data/DefaultConfigs/Login.config.json", "./Data/CurrencyConfigs/" + C.ID + "/Login.config.json"); return true; } public void UpdateConfigs() { //Overwrite the current contents of the configuration files with the new config data FileManager.WriteFile("./Data/CurrencyConfigs/" + this.ID + "/Command.config.json",this.CommandConfig.ToString()); FileManager.WriteFile("./Data/CurrencyConfigs/" + this.ID + "/Login.config.json", this.LoginConfig.ToString()); } public void Delete() { if (FromID(this.ID) != null) { //Delete the folder with the currency configuration files in System.IO.Directory.Delete("./Data/CurrencyConfigs/" + this.ID,true); List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID", this.ID) }; Init.SQLi.Execute(@"DELETE FROM [Currency]WHERE (((Currency.CurrencyID)=@ID));", Params); } } }}

Twitch-Discord-Reward-API/Backend/Data/Objects/Login.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Data.OleDb;namespace Twitch\_Discord\_Reward\_API.Backend.Data.Objects{ public class Login : BaseObject { public string UserName, HashedPassword, AccessToken,Email; public DateTime LastLoginDateTime; public static Login FromJson(Newtonsoft.Json.Linq.JToken Json) { return Json.ToObject<Login>(); } public static Login FromID(int ID,bool WithSecretData=false) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID",ID) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Logins.LoginID, Logins.UserName, Logins.HashedPassword, Logins.AccessToken, Logins.LastLoginDateTime, Logins.EmailFROM LoginsWHERE (((Logins.LoginID)=@ID));", Params); return FromRData(RData, WithSecretData); } public static Login FromUserName(string UserName,bool WithSecretData=false) { if (UserName == null) { return null; } List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("UserName", UserName) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Logins.LoginID, Logins.UserName, Logins.HashedPassword, Logins.AccessToken, Logins.LastLoginDateTime, Logins.EmailFROM LoginsWHERE (((Logins.UserName)=@UserName));", Params); return FromRData(RData, WithSecretData); } public static Login FromEmail(string Email,bool WithSecretData = false) { if (Email == null) { return null; } List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("Email",Email) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Logins.LoginID, Logins.UserName, Logins.HashedPassword, Logins.AccessToken, Logins.LastLoginDateTime, Logins.EmailFROM LoginsWHERE (((Logins.Email)=@Email));", Params); return FromRData(RData, WithSecretData); } static Login FromRData(List<string[]> RData, bool WithSecretData) { if (RData.Count == 0) { return null; } Login Login = new Login(); Login.ID = int.Parse(RData[0][0]); Login.UserName = RData[0][1]; if (WithSecretData) { Login.HashedPassword = RData[0][2]; Login.AccessToken = RData[0][3]; Login.Email = RData[0][5]; } Login.LastLoginDateTime = DateTime.Parse(RData[0][4]); return Login; } public bool Save() { if (FromEmail(this.Email) == null && FromUserName(this.UserName) == null) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("HashedPassword",this.HashedPassword), new OleDbParameter("AccessToken",Init.ScryptEncoder.Encode(Networking.TokenSystem.CreateToken(64))), new OleDbParameter("LastLoginDateTime",DateTime.Now.ToString()) }; string PreValue = ""; string PostValue = ""; if (this.Email != null) { Params.Add(new OleDbParameter("Email", this.Email)); PreValue += "Email"; PostValue += "@Email"; } if (this.UserName != null) { Params.Add(new OleDbParameter("UserName", this.UserName)); if (PreValue != "") { PreValue += ","; PostValue += ","; } PreValue += "UserName"; PostValue += "@UserName"; } Init.SQLi.Execute(@"INSERT INTO Logins (HashedPassword, AccessToken, LastLoginDateTime, "+PreValue+@") VALUES (@HashedPassword, @AccessToken, @LastLoginDateTime, "+PostValue+@")", Params); return true; } return false; } public bool UpdateToken() { if (FromID(this.ID)!=null) { this.AccessToken = Networking.TokenSystem.CreateToken(64); this.LastLoginDateTime = DateTime.Now; List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("AccessToken",Init.ScryptEncoder.Encode(this.AccessToken)), new OleDbParameter("LastLoginDateTime",this.LastLoginDateTime.ToString()), new OleDbParameter("ID",this.ID) }; Init.SQLi.Execute(@"UPDATE Logins SET Logins.AccessToken = @AccessToken, Logins.LastLoginDateTime = @LastLoginDateTimeWHERE(((Logins.LoginID) = @ID));", Params); return true; } return false; } public bool UpdateUserNameEmailPassword() { bool IsTaken = false; if (FromUserName(this.UserName) != null) { if (FromUserName(this.UserName).ID != this.ID) { IsTaken = true; } } if (FromEmail(this.Email) != null) { if (FromEmail(this.Email).ID != this.ID) { IsTaken = true; } } if (FromID(this.ID) != null) { if (!IsTaken) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("HashedPassword",this.HashedPassword) }; string UpdateString = ""; if (this.UserName != null) { UpdateString += ", Logins.UserName = @UserName"; Params.Add(new OleDbParameter("UserName", this.UserName)); } if (this.Email != null) { UpdateString += ", Logins.Email = @Email"; Params.Add(new OleDbParameter("Email", this.Email)); } Params.Add(new OleDbParameter("ID", this.ID)); Init.SQLi.Execute(@"UPDATE Logins SET Logins.HashedPassword = @HashedPassword"+UpdateString+@"WHERE(((Logins.LoginID) = @ID));", Params); return true; } } return false; } public void Delete() { if (FromID(this.ID) != null) { foreach (Currency C in Currency.FromLogin(this.ID)) { C.Delete(); }//Delete all currencies tied to this login List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID", this.ID) }; Init.SQLi.Execute(@"DELETE FROM LoginsWHERE (((Logins.LoginID)=@ID));", Params); } } }}

Twitch-Discord-Reward-API/Backend/Data/Objects/Viewer.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Data.OleDb;namespace Twitch\_Discord\_Reward\_API.Backend.Data.Objects{ public class Viewer:BaseObject { public int Balance,WatchTime; public string TwitchID, DiscordID; public Currency Currency; public bool LiveNotifcations,DontReward; public static Viewer FromJson(Newtonsoft.Json.Linq.JToken Json) { return Json.ToObject<Viewer>(); } public static Viewer FromID(int ID) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID",ID) }; List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Viewer.ViewerID, Viewer.DiscordID, Viewer.TwitchID, Viewer.Balance, Viewer.CurrencyID, Viewer.WatchTime, Viewer.LiveNotifications, Viewer.DontRewardFROM ViewerWHERE(((Viewer.ViewerID) = @ID));", Params); if (RData.Count == 0) { return null; } Viewer Viewer = new Viewer(); Viewer.ID = ID; Viewer.Balance = int.Parse(RData[0][3]); Viewer.DiscordID = RData[0][1]; Viewer.TwitchID = RData[0][2]; Viewer.Currency = Currency.FromID(int.Parse(RData[0][4])); Viewer.WatchTime = int.Parse(RData[0][5]); Viewer.LiveNotifcations = RData[0][6] == "True"; Viewer.DontReward = RData[0][7] == "True"; return Viewer; } public static List<Viewer> FromCurrency(int CurrencyID,string OrderBy = "None") { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("CurrencyID", CurrencyID) }; string Command = @"SELECT Viewer.ViewerID, Viewer.DiscordID, Viewer.TwitchID, Viewer.Balance, Viewer.CurrencyID, Viewer.WatchTime, Viewer.LiveNotifications, Viewer.DontRewardFROM ViewerWHERE (((Viewer.CurrencyID)=@CurrencyID))"; if (OrderBy == "Balance") { Command += "ORDER BY Viewer.Balance DESC"; } if (OrderBy == "WatchTime") { Command += "ORDER BY Viewer.WatchTime DESC"; } Command += ";"; List<String[]> RData = Init.SQLi.ExecuteReader(Command, Params); List<Viewer> CurrencyBanks = new List<Viewer> { }; foreach (String[] Item in RData) { Viewer Viewer = new Viewer(); Viewer.ID = int.Parse(Item[0]); Viewer.DiscordID = Item[1]; Viewer.TwitchID = Item[2]; Viewer.Balance = int.Parse(Item[3]); Viewer.WatchTime = int.Parse(Item[5]); Viewer.LiveNotifcations = Item[6] == "True"; Viewer.DontReward = Item[7] == "True"; CurrencyBanks.Add(Viewer); } return CurrencyBanks; } public static List<Viewer> FromTwitchDiscord(string DiscordID=null,string TwitchID=null) { List<OleDbParameter> Params = new List<OleDbParameter> { }; string WhereStatment = ""; if (DiscordID != null) { Params.Add(new OleDbParameter("DiscordID", DiscordID)); WhereStatment += "((Viewer.DiscordID)=@DiscordID)"; }//Add the DiscordID paramater if DiscordID isnt null if (TwitchID != null){//If TwitchID isnt null if (WhereStatment != "") { WhereStatment += " AND "; }//If weve already added DiscordID we add AND into the statment Params.Add(new OleDbParameter("TwitchID", TwitchID)); WhereStatment += "((Viewer.TwitchID)=@TwitchID)";//Add the TwitchID paramater } List<String[]> RData = Init.SQLi.ExecuteReader(@"SELECT Viewer.ViewerID, Viewer.DiscordID, Viewer.TwitchID, Viewer.Balance, Viewer.CurrencyID, Viewer.WatchTime, Viewer.LiveNotifications, Viewer.DontRewardFROM ViewerWHERE " + WhereStatment+@";", Params); List<Viewer> UserBanks = new List<Viewer> { }; foreach (String[] Item in RData) { Viewer Viewer = new Viewer(); Viewer.ID = int.Parse(Item[0]); Viewer.DiscordID = Item[1]; Viewer.TwitchID = Item[2]; Viewer.Balance = int.Parse(Item[3]); Viewer.Currency = Currency.FromID(int.Parse(Item[4])); Viewer.WatchTime = int.Parse(Item[5]); Viewer.LiveNotifcations = Item[6] == "True"; Viewer.DontReward = Item[7] == "True"; UserBanks.Add(Viewer); } return UserBanks; } public static Viewer FromTwitchDiscord(string DiscordID = null, string TwitchID = null,int CurrencyID=-1) { if (CurrencyID != -1) { List<Viewer> B = FromTwitchDiscord(DiscordID, TwitchID); return B.Find(x => x.Currency.ID == CurrencyID); } return null; } //Increment the balance and watchtime by the gioven amount for all accounts with the given ids public static bool Increment(List<string> DiscordIDs = null, List<string> TwitchIDs=null,int BalanceIncrementBy=0,int WatchTimeIncrementBy=0,int CurrencyID=0) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("BalanceIncrement", BalanceIncrementBy),new OleDbParameter("WatchTimeIncrement",WatchTimeIncrementBy) }; string WhereStatement = ""; int i = 0; foreach(string DID in DiscordIDs)//Cycle through every ID in the discord id set { Params.Add(new OleDbParameter("DiscordID" + i, DID));//Add a paramater containing the discord id to the set if (WhereStatement != "") { WhereStatement += " OR "; }//and an OR between each statement WhereStatement += "Viewer.DiscordID=@DiscordID" + i;//Add on the conditional statement i++; } i = 0; foreach (string TID in TwitchIDs)//Does the same as above just for twitch ids { Params.Add(new OleDbParameter("TwitchID" + i, TID)); if (WhereStatement != "") { WhereStatement += " OR "; } WhereStatement += "Viewer.TwitchID=@TwitchID" + i; i++; } string ExtraStatement = ""; if (CurrencyID != 0) { ExtraStatement = " AND (Viewer.CurrencyID=@CurrencyID)"; Params.Add(new OleDbParameter("CurrencyID", CurrencyID)); } Init.SQLi.Execute(@"UPDATE Viewer SET Viewer.Balance = Viewer.Balance + @BalanceIncrement, Viewer.WatchTime = Viewer.WatchTime + @WatchTimeIncrementWHERE (((Viewer.DontReward)=False) AND (" + WhereStatement+@")"+ExtraStatement+@");", Params); //Increment all matching ids balances and watchtime by the given amount return true; } public bool Save() { //Check if DiscordID or TwitchID is already in the database if (FromTwitchDiscord(this.DiscordID,this.TwitchID,this.Currency.ID) == null) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("Balance",this.Balance), new OleDbParameter("CurrencyID",this.Currency.ID) }; //Set the sql paramaters string PostStatment = "",PreStatment=""; //If DiscorID isnt null, we add it to our params and value statments if (DiscordID != null) { Params.Add(new OleDbParameter("DiscordID", DiscordID)); PreStatment += "DiscordID"; PostStatment += "@DiscordID"; } //If TwitchID isnt null, we add it to our params and value statments if (TwitchID != null) { //If we have already added to our statments we will need a comma to seperate the values if (PostStatment != "") { PreStatment += ","; PostStatment += ","; } Params.Add(new OleDbParameter("TwitchID", TwitchID)); PreStatment += "TwitchID"; PostStatment += "@TwitchID"; } Init.SQLi.Execute(@"INSERT INTO Viewer (Balance, CurrencyID, " + PreStatment+ @") VALUES (@Balance, @CurrencyID, " + PostStatment+@")", Params); //insert the viewer into the table return true; } return false; } public bool Update() { if (FromID(this.ID) != null) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("DiscordID",this.DiscordID), new OleDbParameter("TwitchID",this.TwitchID), new OleDbParameter("Balance",this.Balance), new OleDbParameter("Notifcations",this.LiveNotifcations), new OleDbParameter("DontReward",this.DontReward), new OleDbParameter("WatchTime",this.WatchTime), new OleDbParameter("ID",this.ID) }; Init.SQLi.Execute(@"UPDATE Viewer SET Viewer.DiscordID = @DiscordID, Viewer.TwitchID = @TwitchID, Viewer.Balance = @Balance, Viewer.LiveNotifications = @Notifications, Viewer.DontReward = @DontReward, Viewer.WatchTime = @WatchTimeWHERE(((Viewer.ViewerID) = @ID));", Params); return true; } else { return false; } } public void Delete() { if (FromID(this.ID) != null) { List<OleDbParameter> Params = new List<OleDbParameter> { new OleDbParameter("ID",this.ID) }; Init.SQLi.Execute(@"DELETE FROM ViewerWHERE (((Viewer.ViewerID)=@ID));",Params); } } }}

Twitch-Discord-Reward-API/Backend/Data/FileManager.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend.Data{ public static class FileManager { public static Newtonsoft.Json.Linq.JToken ReadFile(string FilePath) { if (System.IO.File.Exists(FilePath))//Check if the file exists { string Raw = System.IO.File.ReadAllText(FilePath);//Read the file try { return Newtonsoft.Json.Linq.JToken.Parse(Raw); }//Try to convert the file contents to json form and pass it back catch { return null; }//If it cant be converted return null } return null; } public static void WriteFile(string FilePath,Newtonsoft.Json.Linq.JToken Json) { System.IO.File.WriteAllText(FilePath, Json.ToString());//Write the json into the given file } }}

Twitch-Discord-Reward-API/Backend/Data/SQL.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Data.OleDb;namespace Twitch\_Discord\_Reward\_API.Backend.Data{ public class SQL { private OleDbConnection Conn;//Stores The Active Database Coneection private string DBase = "";//Stores The Databases File Path public SQL(string DataBase) { DBase = DataBase; RestartConn(); } private void RestartConn() { if (Conn != null) { if (Conn.State == System.Data.ConnectionState.Open) { Conn.Close(); } /\* If connection is open, close it\*/ } Conn = new OleDbConnection("Provider = Microsoft.ACE.OLEDB.12.0; Data Source = " + DBase + ".accdb"); // Open a new database connection Conn.Open(); } public List<String[]> ExecuteReader(String sCommand, List<OleDbParameter> ParamCollection = null) { OleDbCommand Command = new OleDbCommand(sCommand, Conn); // Create the command, using the opened connection and the sql string if (ParamCollection != null) { for (int i = 0; i < ParamCollection.Count; i++) { Command.Parameters.Add(ParamCollection[i]); } } // Add the paramaters OleDbDataReader Results = Command.ExecuteReader(); // Execute the reader and store the result List<String[]> LResults = new List<string[]> { }; // Create a list of String[] too store the rows and collumns of the results while (Results.Read()) // Keep reading untill all is read { string[] Data = new string[Results.FieldCount]; // Create a temporary String[] for (int i = 0; i < Results.FieldCount; i++) { Data[i] = Results.GetValue(i).ToString(); } // Place each collumn in the row into the array LResults.Add(Data); // Add the row to the list } Results.Close(); // Terminate read and pass the formatted results back return LResults; } public void Execute(String sCommand, List<OleDbParameter> ParamCollection = null) { OleDbCommand Command = new OleDbCommand(sCommand, Conn); // Create the command, using the opened connection ad the sql string parameter if (ParamCollection != null) { for (int i = 0; i < ParamCollection.Count; i++) { Command.Parameters.Add(ParamCollection[i]); } } // Add the paramaters try { Command.ExecuteNonQuery(); /\*RestartConn();\*/ } catch (OleDbException E) { Console.WriteLine(E); } // Execute the command } }}

Twitch-Discord-Reward-API/Backend/Networking/HTTPServer/Get.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Net;using System.IO;namespace Twitch\_Discord\_Reward\_API.Backend.Networking.HTTPServer{ public static class Get { public static ResponseObject Handle(StandardisedRequestObject Context) { bool ErrorOccured = false; // Check if TwitchID and DiscordID only compose of numbers if (Context.Headers.AllKeys.Contains("TwitchID")) { if (!Checks.IsValidID(Context.Headers["TwitchID"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, TwitchID contains invalid characters"; return Context.ResponseObject; } } if (Context.Headers.AllKeys.Contains("DiscordID")) { if (!Checks.IsValidID(Context.Headers["DiscordID"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, DiscordID contains invalid characters"; return Context.ResponseObject; } } if (Context.URLSegments[1] == "viewer")//Check the url path for viewer { if (Context.Headers.AllKeys.Contains("ID")) // Get the viewer where header ID matches { try { int.Parse(Context.Headers["ID"]); }//Check if the ID Header can be converted to an integer catch {//If it cant be converted, set the contents of the Response Object to reflect this Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Viewer B = Data.Objects.Viewer.FromID(int.Parse(Context.Headers["ID"]));//Fetch the Viewer Object with the given ID if (B != null) { Context.ResponseObject.Data = B.ToJson(); }//If We get a Viewer back, set the Response Objects data to the JSON format of the Viewer else {//If we didnt get a viewer back, set the contents of the Response Object to reflect that a viewer doesnt exist with the given ID Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not match an existing object"; ErrorOccured = true; } } else if ((Context.Headers.AllKeys.Contains("TwitchID") || Context.Headers.AllKeys.Contains("DiscordID")) && Context.Headers.AllKeys.Contains("CurrencyID")) // Get the viewer where header (TwitchID and/or DiscordID) and CurrencyID matches { try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Viewer B = Data.Objects.Viewer.FromTwitchDiscord(Context.Headers["DiscordID"], Context.Headers["TwitchID"], int.Parse(Context.Headers["CurrencyID"])); if (B != null) { Context.ResponseObject.Data = B.ToJson(); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, TwitchID and/or DiscordID does not match an existing object"; ErrorOccured = true; } } else if (Context.Headers.AllKeys.Contains("CurrencyID")) // Get all viewers for the CurrencyID { string OrderBy = null; if (Context.Headers["Order"] == "WatchTime" || Context.Headers["Order"] == "Balance") { OrderBy = Context.Headers["Order"]; } try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } List<Data.Objects.Viewer> B = Data.Objects.Viewer.FromCurrency(int.Parse(Context.Headers["CurrencyID"]), OrderBy); if (B.Count != 0) { Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(B); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, CurrencyID does not match an existing object"; ErrorOccured = true; } } else if (Context.Headers.AllKeys.Contains("TwitchID") || Context.Headers.AllKeys.Contains("DiscordID")) // Get all viewers for any currency where TwitchID and/or DiscordID matches { List<Data.Objects.Viewer> B = Data.Objects.Viewer.FromTwitchDiscord(Context.Headers["DiscordID"], Context.Headers["TwitchID"]); if (B.Count != 0) { Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(B); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, TwitchID and/or DiscordID does not match an existing object"; ErrorOccured = true; } } else//Inform requestor that we dont have any infomation to work with { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "currency") { if (Context.Headers.AllKeys.Contains("ID"))//Get Currency where ID matches { try { int.Parse(Context.Headers["ID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Currency C = Data.Objects.Currency.FromID(int.Parse(Context.Headers["ID"])); if (Context.Headers.AllKeys.Contains("AccessToken") && Context.Headers.AllKeys.Contains("LoginID")) { // If a valid accesstoken is provided, get private information try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["LoginID"]), true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { if (Data.Objects.Currency.FromLogin(L.ID).Find(x => x.ID == C.ID) != null) { C.LoadConfigs(true); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, LoginID does not correspond to an existing user"; } } if (C != null) { Context.ResponseObject.Data = C.ToJson(); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not match an existing object"; ErrorOccured = true; } } else if (Context.Headers.AllKeys.Contains("LoginID"))// Get all Currencies of the LoginID { try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed LoginID"; return Context.ResponseObject; } List<Data.Objects.Currency> C = Data.Objects.Currency.FromLogin(int.Parse(Context.Headers["LoginID"])); Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(C); Context.ResponseObject.Code = 200; Context.ResponseObject.Message = "Unknown Outcome, It is not known if the LoginID matches an object"; ErrorOccured = true; } else//Inform requestor that we dont have any infomation to work with { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "login") { if (Context.Headers.AllKeys.Contains("ID"))//Get Login where ID matches { try { int.Parse(Context.Headers["ID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["ID"])); if (L != null) { Context.ResponseObject.Data = L.ToJson(); if (Context.Headers.AllKeys.Contains("AccessToken")) { if (Context.Headers["AccessToken"] != "") { L = Data.Objects.Login.FromID(int.Parse(Context.Headers["ID"]), true); if (!Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken doesnt match"; ErrorOccured = true; } else { L.AccessToken = null; L.HashedPassword = null; Context.ResponseObject.Data = L.ToJson(); } } } } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not match an existing object"; ErrorOccured = true; } } else if (Context.Headers.AllKeys.Contains("UserName"))//Get Login where UserName matches { Data.Objects.Login L = Data.Objects.Login.FromUserName(Context.Headers["UserName"]); if (L != null) { Context.ResponseObject.Data = L.ToJson(); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, UserName does not match an existing object"; ErrorOccured = true; } } else if (Context.Headers.AllKeys.Contains("Email"))//Get Login where Email matches { Data.Objects.Login L = Data.Objects.Login.FromEmail(Context.Headers["Email"]); if (L != null) { Context.ResponseObject.Data = L.ToJson(); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Email does not match an existing object"; ErrorOccured = true; } } else//Inform requestor that we dont have any infomation to work with { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "bot") { if (Context.Headers.AllKeys.Contains("ID") && Context.Headers.AllKeys.Contains("LoginID"))//Get Bot where ID matches { bool WithSecretData = false; try { int.Parse(Context.Headers["ID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } if (Context.Headers.AllKeys.Contains("AccessToken"))// If a valid accesstoken is provided, get private information { try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["LoginID"]), true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { if (Data.Objects.Bot.FromLogin(L.ID).Find(x => x.ID == int.Parse(Context.Headers["ID"])) != null) { WithSecretData = true; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, LoginID does not correspond to an existing user"; } } Data.Objects.Bot B = Data.Objects.Bot.FromID(int.Parse(Context.Headers["ID"]), WithSecretData); if (B != null) { Context.ResponseObject.Data = B.ToJson(); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not match an existing object"; ErrorOccured = true; } } else if (Context.Headers.AllKeys.Contains("LoginID"))//Get all Bots of LoginID { try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed LoginID"; return Context.ResponseObject; } List<Data.Objects.Bot> B = Data.Objects.Bot.FromLogin(int.Parse(Context.Headers["LoginID"])); Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(B); Context.ResponseObject.Code = 200; Context.ResponseObject.Message = "Unknown Outcome, It is not known if the LoginID matches an object"; ErrorOccured = true; } else if (Context.Headers.AllKeys.Contains("CurrencyID"))//Get all Bots of CurrencyID { try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } List<Data.Objects.Bot> B = Data.Objects.Bot.FromCurrency(int.Parse(Context.Headers["CurrencyID"])); if (B.Count != 0) { Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(B); } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, CurrencyID does not match an existing object"; ErrorOccured = true; } } else//Inform requestor that we dont have any infomation to work with { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "nightbot") { Context.GetStateParams(); if (Context.URLParamaters.ContainsKey("code") && Context.URLParamaters.ContainsKey("state") && Context.StateParamaters.ContainsKey("currencyid") && Context.StateParamaters.ContainsKey("accesstoken")) { string Code = Context.URLParamaters["code"]; try { int.Parse(Context.StateParamaters["currencyid"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Currency C = Data.Objects.Currency.FromID(int.Parse(Context.StateParamaters["currencyid"])); if (C == null) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, CurrencyID does not match an existing object"; ErrorOccured = true; } else { Data.Objects.Login L = Data.Objects.Login.FromID(C.OwnerLogin.ID, true); if (Backend.Init.ScryptEncoder.Compare(Context.StateParamaters["accesstoken"], L.AccessToken)) { C.LoadConfigs(true); WebRequest Req = WebRequest.Create("https://api.nightbot.tv/oauth2/token"); Req.Method = "POST"; byte[] PostData = Encoding.UTF8.GetBytes("client\_id=" + C.LoginConfig["NightBot"]["ClientId"] + "&client\_secret=" + C.LoginConfig["NightBot"]["ClientSecret"] + "&grant\_type=authorization\_code&redirect\_uri=" + Backend.Init.APIConfig["WebURL"] + "/nightbot/&code=" + Code); Req.Method = "POST"; Req.ContentType = "application/x-www-form-urlencoded"; Req.ContentLength = PostData.Length; Stream PostStream = Req.GetRequestStream(); PostStream.Write(PostData, 0, PostData.Length); PostStream.Flush(); PostStream.Close(); try { WebResponse Res = Req.GetResponse(); string D = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JObject JD = Newtonsoft.Json.Linq.JObject.Parse(D); C.LoginConfig["NightBot"]["RefreshToken"] = JD["refresh\_token"]; C.UpdateConfigs(); } catch (WebException E) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Something went wrong"; Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "AccessToken is not allowed to modify that currency"; } } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Code and/or currencyid and/or accesstoken is missing"; } } else if (Context.URLSegments[1] == "streamlabs") { Context.GetStateParams(); if (Context.URLParamaters.ContainsKey("code") && Context.URLParamaters.ContainsKey("state") && Context.StateParamaters.ContainsKey("currencyid")) { string Code = Context.URLParamaters["code"]; try { int.Parse(Context.StateParamaters["currencyid"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Currency C = Data.Objects.Currency.FromID(int.Parse(Context.StateParamaters["currencyid"])); if (C == null) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, CurrencyID does not match an existing object"; ErrorOccured = true; } else { Data.Objects.Login L = Data.Objects.Login.FromID(C.OwnerLogin.ID, true); if (Backend.Init.ScryptEncoder.Compare(Context.StateParamaters["accesstoken"], L.AccessToken)) { C.LoadConfigs(true); WebRequest Req = WebRequest.Create("https://streamlabs.com/api/v1.0/token"); Req.Method = "POST"; Req.ContentType = "application/x-www-form-urlencoded"; byte[] PostData = Encoding.UTF8.GetBytes("grant\_type=authorization\_code&client\_id=" + C.LoginConfig["StreamLabs"]["ClientId"] + "&client\_secret=" + C.LoginConfig["StreamLabs"]["ClientSecret"] + "&redirect\_uri=" + Backend.Init.APIConfig["WebURL"] + "/streamlabs/&code=" + Code); Req.ContentLength = PostData.Length; Stream PostStream = Req.GetRequestStream(); PostStream.Write(PostData, 0, PostData.Length); PostStream.Flush(); PostStream.Close(); WebResponse Res; try { Res = Req.GetResponse(); Newtonsoft.Json.Linq.JObject D = Newtonsoft.Json.Linq.JObject.Parse(new StreamReader(Res.GetResponseStream()).ReadToEnd()); C.LoginConfig["StreamLabs"]["RefreshToken"] = D["refresh\_token"]; C.UpdateConfigs(); } catch (WebException E) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Something went wrong"; Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "AccessToken is not allowed to modify that currency"; } } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Code and/or currencyid and/or accesstoken is missing"; } } else if (Context.URLSegments[1] == "twitch") { Context.GetStateParams(); if (Context.URLParamaters.ContainsKey("code") && Context.URLParamaters.ContainsKey("state") && Context.StateParamaters.ContainsKey("currencyid")) { string Code = Context.URLParamaters["code"]; try { int.Parse(Context.StateParamaters["currencyid"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Currency C = Data.Objects.Currency.FromID(int.Parse(Context.StateParamaters["currencyid"])); if (C == null) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, CurrencyID does not match an existing object"; ErrorOccured = true; } else { Data.Objects.Login L = Data.Objects.Login.FromID(C.OwnerLogin.ID, true); if (Backend.Init.ScryptEncoder.Compare(Context.StateParamaters["accesstoken"], L.AccessToken)) { C.LoadConfigs(true); WebRequest Req = WebRequest.Create("https://id.twitch.tv/oauth2/token"); Req.Method = "POST"; Req.ContentType = "application/x-www-form-urlencoded"; byte[] PostData = Encoding.UTF8.GetBytes("grant\_type=authorization\_code&client\_id=" + C.LoginConfig["Twitch"]["API"]["ClientId"] + "&client\_secret=" + C.LoginConfig["Twitch"]["API"]["ClientSecret"] + "&redirect\_uri=" + Backend.Init.APIConfig["WebURL"] + "/twitch/&code=" + Code); Req.ContentLength = PostData.Length; Stream PostStream = Req.GetRequestStream(); PostStream.Write(PostData, 0, PostData.Length); PostStream.Flush(); PostStream.Close(); WebResponse Res; try { Res = Req.GetResponse(); Newtonsoft.Json.Linq.JObject D = Newtonsoft.Json.Linq.JObject.Parse(new StreamReader(Res.GetResponseStream()).ReadToEnd()); C.LoginConfig["Twitch"]["API"]["RefreshToken"] = D["refresh\_token"]; C.UpdateConfigs(); } catch (WebException E) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Something went wrong"; Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "AccessToken is not allowed to modify that currency"; } } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Code and/or currencyid and/or accesstoken is missing"; } } else//Inform requestor that the url does not got anywhere { Context.ResponseObject.Code = 404; Context.ResponseObject.Message = "Not Found"; ErrorOccured = true; } if (ErrorOccured == false) { Context.ResponseObject.Code = 200; Context.ResponseObject.Message = "The requested task was performed successfully"; } return Context.ResponseObject; } }}

Twitch-Discord-Reward-API/Backend/Networking/HTTPServer/Init.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading;using System.Net;namespace Twitch\_Discord\_Reward\_API.Backend.Networking.HTTPServer{ public static class Init { static HttpListener Listener; public static void Start() { Listener = new HttpListener(); // Initalise the Listener and configure it Listener.Prefixes.Add("http://+:"+Backend.Init.APIConfig["Port"]+"/"); Listener.Start(); Listener.BeginGetContext(HandleRequest, null);//When we recive a request send to the the HandleRequest procdeure if (Listener.IsListening) { Console.WriteLine("Web API is now running!"); } // Report that the listener is running } static void HandleRequest(IAsyncResult Request) { new Thread(() => RequestThread(Listener.EndGetContext(Request))).Start();//Create a thread of RequestThread, in order to prevent delay in handling new requests Listener.BeginGetContext(HandleRequest, null); // Restart listener } public static int Size=0; static void RequestThread(HttpListenerContext Context) { string Event = Context.Request.RemoteEndPoint + " Visited " + Context.Request.RawUrl + " Using " + Context.Request.HttpMethod; Console.WriteLine(Event); Size=(Size+1)%100; if (Size == 0) { Console.Clear(); } HttpListenerResponse Resp = Context.Response; // Create the Listener Response and set response parameters Resp.StatusCode = 200; Resp.ContentType = "application/json"; ResponseObject ResponseObject = new ResponseObject(); // Create a reponse object and assign default values ResponseObject.Code = 400; ResponseObject.Message = "Non-Specific Bad Request"; try { // Create a StandardisedRequestObject and provide it to the Get or Post function based on the method used by the request StandardisedRequestObject Req = new StandardisedRequestObject(Context, ResponseObject); if (Req.Method == "get") { Get.Handle(Req); } if (Req.Method == "post") { Post.Handle(Req); } } catch (Exception E) { Console.WriteLine(E); ResponseObject.Code = 500; ResponseObject.Message = "Internal Server Error"; } // If an unhandled error occurs set fallback values byte[] ByteResponseData = Encoding.UTF8.GetBytes(ResponseObject.ToJson().ToString()); // Convert the response object into its json equivalent and then into its byte values try { // Send the byte response data to the requestor Resp.OutputStream.Write(ByteResponseData, 0, ByteResponseData.Length); Resp.OutputStream.Close(); } catch { Console.WriteLine("Unable to send response too " + Context.Request.RemoteEndPoint); } // If we cant send the response report the error to console } }}

Twitch-Discord-Reward-API/Backend/Networking/HTTPServer/Post.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend.Networking.HTTPServer{ public static class Post { public static ResponseObject Handle(StandardisedRequestObject Context) { bool ErrorOccured = false; Backend.Data.Objects.Bot CorrespondingBot = AuthCheck(Context); if (Context.Headers.AllKeys.Contains("TwitchID")) { if (!Checks.IsValidID(Context.Headers["TwitchID"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, TwitchID contains invalid characters"; return Context.ResponseObject; } } if (Context.Headers.AllKeys.Contains("DiscordID")) { if (!Checks.IsValidID(Context.Headers["DiscordID"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, DiscordID contains invalid characters"; return Context.ResponseObject; } } if (Context.URLSegments[1] == "viewer") { if ((Context.Headers.AllKeys.Contains("TwitchID") || Context.Headers.AllKeys.Contains("DiscordID") || Context.Headers.AllKeys.Contains("Notifications") || Context.Headers.AllKeys.Contains("WatchTime") || Context.Headers.AllKeys.Contains("DontReward")) && Context.Headers.AllKeys.Contains("ID")) { if (CorrespondingBot != null) { Data.Objects.Viewer B = Data.Objects.Viewer.FromID(int.Parse(Context.Headers["ID"])); if (B == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing viewer"; return Context.ResponseObject; } if (B.Currency.ID == CorrespondingBot.Currency.ID || CorrespondingBot.IsSuperBot) { if (Context.Headers["DiscordID"] != null) { B.DiscordID = Context.Headers["DiscordID"]; } if (Context.Headers["TwitchID"] != null) { B.TwitchID = Context.Headers["TwitchID"]; } if (Context.Headers["Notifications"] != null) { B.LiveNotifcations = Context.Headers["Notifications"] == "True"; } if (Context.Headers["WatchTime"] != null) { B.WatchTime = int.Parse(Context.Headers["WatchTime"]); } if (Context.Headers["DontReward"] != null) { B.DontReward = Context.Headers["DontReward"] == "True"; } B.Update(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This bot does not have permission to edit that Bank"; } } else { ErrorOccured = true; //Context.ResponseObject.Code = 403; Context.ResponseObject.Message = "Invalid AuthToken"; } } else if (Context.Headers.AllKeys.Contains("TwitchID") || Context.Headers.AllKeys.Contains("DiscordID")) { if (CorrespondingBot != null) { Data.Objects.Viewer B = new Data.Objects.Viewer(); if (B == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing viewer"; return Context.ResponseObject; } B.DiscordID = Context.Headers["DiscordID"]; B.TwitchID = Context.Headers["TwitchID"]; if (Context.Headers.AllKeys.Contains("CurrencyID")) { try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } if (int.Parse(Context.Headers["CurrencyID"]) == CorrespondingBot.Currency.ID || CorrespondingBot.IsSuperBot) { B.Currency = Data.Objects.Currency.FromID(int.Parse(Context.Headers["CurrencyID"])); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This bot does not have permission to edit that Currency"; return Context.ResponseObject; } } else { B.Currency = CorrespondingBot.Currency; } B.Balance = int.Parse(CorrespondingBot.Currency.CommandConfig["InititalBalance"].ToString()); if (B.Currency != null) { if (!B.Save()) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, a Viewer already exists in this currency witht that Discord and/or Twitch ID"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, was unable to set Currency, try explicitly setting Currency with CurrencyID header"; } } else { ErrorOccured = true; //Context.ResponseObject.Code = 403; Context.ResponseObject.Message = "Invalid AuthToken"; } } else if (Context.Headers.AllKeys.Contains("ID") && Context.Headers.AllKeys.Contains("Operator") && Context.Headers.AllKeys.Contains("Value")) { if (CorrespondingBot != null) { try { int.Parse(Context.Headers["ID"]); int.Parse(Context.Headers["Value"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID and/or Value"; return Context.ResponseObject; } Data.Objects.Viewer B = Data.Objects.Viewer.FromID(int.Parse(Context.Headers["ID"])); if (B == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing viewer"; return Context.ResponseObject; } if (B.Currency.ID == CorrespondingBot.Currency.ID || CorrespondingBot.IsSuperBot) { if (Context.Headers["Operator"].ToString() == "+") { B.Balance += int.Parse(Context.Headers["Value"]); if (B.Balance >= 0) { B.Update(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Cannot set balance as negative"; } } else if (Context.Headers["Operator"].ToString() == "-") { B.Balance -= int.Parse(Context.Headers["Value"]); if (B.Balance >= 0) { B.Update(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Cannot set balance as negative"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Operator must be + or -"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This bot does not have permission to edit that Bank"; } } else { ErrorOccured = true; //Context.ResponseObject.Code = 403; Context.ResponseObject.Message = "Invalid AuthToken"; } } else if ((Context.Headers.AllKeys.Contains("BalanceIncrement") || Context.Headers.AllKeys.Contains("WatchTimeIncrement")) && Context.Headers.AllKeys.Contains("CurrencyID") && Context.RequestData != null) { if (CorrespondingBot != null) { int BalanceIncrement = 0, WatchTimeIncrement = 0; if (Context.Headers.AllKeys.Contains("BalanceIncrement")) { try { BalanceIncrement = int.Parse(Context.Headers["BalanceIncrement"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed BalanceIncrement"; return Context.ResponseObject; } } if (Context.Headers.AllKeys.Contains("WatchTimeIncrement")) { try { WatchTimeIncrement = int.Parse(Context.Headers["WatchTimeIncrement"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed WatchTimeIncrement"; return Context.ResponseObject; } } try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Currency C = Data.Objects.Currency.FromID(int.Parse(Context.Headers["CurrencyID"])); if (C != null) { if (C.ID == CorrespondingBot.Currency.ID || CorrespondingBot.IsSuperBot) { List<string> DiscordIDs = new List<string> { }, TwitchIDs = new List<string> { }; if (Context.RequestData["DiscordIDs"] != null) { DiscordIDs = Context.RequestData["DiscordIDs"].ToObject<List<string>>(); } if (Context.RequestData["TwitchIDs"] != null) { TwitchIDs = Context.RequestData["TwitchIDs"].ToObject<List<string>>(); } Data.Objects.Viewer.Increment(DiscordIDs, TwitchIDs, BalanceIncrement, WatchTimeIncrement, C.ID); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This bot does not have permission to edit that Bank"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, CurrencyID does not correspond to an existing Currency"; } } else { ErrorOccured = true; //Context.ResponseObject.Code = 403; Context.ResponseObject.Message = "Invalid AuthToken"; } } else if (Context.Headers.AllKeys.Contains("ID")) { if (CorrespondingBot != null) { try { int.Parse(Context.Headers["ID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Viewer B = Data.Objects.Viewer.FromID(int.Parse(Context.Headers["ID"])); if (B == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing viewer"; return Context.ResponseObject; } if (B.Currency.ID == CorrespondingBot.Currency.ID || CorrespondingBot.IsSuperBot) { B.Delete(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This bot does not have permission to edit that Bank"; } } else { ErrorOccured = true; //Context.ResponseObject.Code = 403; Context.ResponseObject.Message = "Invalid AuthToken"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "login") { if ((Context.Headers.AllKeys.Contains("UserName") || Context.Headers.AllKeys.Contains("Email") || Context.Headers.AllKeys.Contains("Password")) && Context.Headers.AllKeys.Contains("AccessToken") && Context.Headers.AllKeys.Contains("ID")) { try { int.Parse(Context.Headers["ID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["ID"]),true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { if (Context.Headers["Email"] != null) { if (!Checks.IsValidEmail(Context.Headers["Email"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Email is not valid"; return Context.ResponseObject; } L.Email = Context.Headers["Email"]; } if (Context.Headers["UserName"] != null) { if (!Checks.IsAlphaNumericString(Context.Headers["UserName"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Username is not AlphaNumeric"; return Context.ResponseObject; } L.UserName = Context.Headers["UserName"]; } if (Context.Headers["Password"] != null) { if (Context.Headers["Password"].Length < 8) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password too short"; return Context.ResponseObject; } if (!Checks.IsValidPassword(Context.Headers["Password"])) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password requires at least 1 Capital, 1 Number, 1 Special"; return Context.ResponseObject; } L.HashedPassword = new Scrypt.ScryptEncoder().Encode(Context.Headers["Password"]); } if (!L.UpdateUserNameEmailPassword()) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, That UserName or Email may be in use by another account"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing user"; } } else if (Context.Headers.AllKeys.Contains("Password")) { if (Context.Headers.AllKeys.Contains("UserName")) { Data.Objects.Login L = Data.Objects.Login.FromUserName(Context.Headers["UserName"], true); if (L != null) { if (Context.Headers["Password"] == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password is null"; } else { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["Password"], L.HashedPassword)) { L.UpdateToken(); L.HashedPassword = null; Context.ResponseObject.Data = L.ToJson(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password does not match"; } } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, UserName does not correspond to an existing user"; } } else if (Context.Headers.AllKeys.Contains("Email")) { Data.Objects.Login L = Data.Objects.Login.FromEmail(Context.Headers["Email"], true); if (L != null) { if (Context.Headers["Password"] == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password is null"; } else { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["Password"], L.HashedPassword)) { L.UpdateToken(); L.HashedPassword = null; Context.ResponseObject.Data = L.ToJson(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password does not match"; } } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Email does not correspond to an existing user"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Email or UserName header is required"; } } else if (Context.URLSegments.Length == 3) { try { int.Parse(Context.Headers["ID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } if (Context.Headers.AllKeys.Contains("AccessToken") && Context.Headers.AllKeys.Contains("ID") && Context.URLSegments[2] == "delete") { Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["ID"]),true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { L.Delete(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing user"; } } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "signup") { if ((Context.Headers.AllKeys.Contains("UserName") || Context.Headers.AllKeys.Contains("Email")) && Context.Headers.AllKeys.Contains("Password")) { Backend.Data.Objects.Login L = new Data.Objects.Login(); L.Email = Context.Headers["Email"]; L.UserName = Context.Headers["UserName"]; if (L.UserName != null) { if (!Checks.IsAlphaNumericString(L.UserName)) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Username is not AlphaNumeric"; return Context.ResponseObject; } } if (L.Email != null) { if (!Checks.IsValidEmail(L.Email)) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Email is not valid"; return Context.ResponseObject; } } if (Data.Objects.Login.FromEmail(L.Email) == null && Data.Objects.Login.FromUserName(L.UserName) == null) { string RawPassword = Context.Headers["Password"]; if (RawPassword.Length < 8) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password too short"; return Context.ResponseObject; } if (!Checks.IsValidPassword(RawPassword)) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Password requires at least 1 Capital, 1 Number, 1 Special"; return Context.ResponseObject; } L.HashedPassword = Backend.Init.ScryptEncoder.Encode(RawPassword); L.Save(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, User already exists"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "bot") { if (Context.Headers.AllKeys.Contains("RefreshToken") && Context.Headers.AllKeys.Contains("BotID")) { try { int.Parse(Context.Headers["BotID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed ID"; return Context.ResponseObject; } Data.Objects.Bot B = Data.Objects.Bot.FromID(int.Parse(Context.Headers["BotID"]),true); if (B != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["RefreshToken"], B.RefreshToken)) { B.PerformRefresh(); Context.ResponseObject.Data = B.ToJson(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Refresh Token is not valid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, BotID does not correspond to a bot"; } } else if (Context.Headers.AllKeys.Contains("AccessToken") && Context.Headers.AllKeys.Contains("CurrencyID") && Context.Headers.AllKeys.Contains("BotID") && Context.Headers.AllKeys.Contains("LoginID")) { try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } try { int.Parse(Context.Headers["BotID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed BotID"; return Context.ResponseObject; } try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed LoginID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["LoginID"]), true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { Data.Objects.Bot B = Data.Objects.Bot.FromID(int.Parse(Context.Headers["BotID"])); if (B != null) { if (B.Currency == null) { B.Currency = Data.Objects.Currency.FromLogin(L.ID).Find(x => x.ID == int.Parse(Context.Headers["CurrencyID"])); if (B.Currency == null) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is not allowed to edit that currency"; } else { B.UpdateCurrency(); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Bot is already bound to a currency"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, BotID doesnt match any bot"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing user"; } } else if (Context.Headers.AllKeys.Contains("AccessToken") && Context.Headers.AllKeys.Contains("LoginID")) { try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed LoginID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["LoginID"]), true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { if (Data.Objects.Bot.FromLogin(L.ID).Count >= 5) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, You are already at the max Bot count"; } else { Data.Objects.Bot B = new Data.Objects.Bot(); if (Context.Headers.AllKeys.Contains("BotName")) { B.BotName = Context.Headers["BotName"]; if (!Checks.IsAlphaNumericString(B.BotName)) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, BotName is not AlphaNumeric"; return Context.ResponseObject; } } else { B.BotName = "No Name Given"; } B.OwnerLogin = Data.Objects.Login.FromID(L.ID); B.Save(); Data.Objects.Bot NewB = Data.Objects.Bot.FromLogin(L.ID, true).Last(); NewB.RefreshToken = B.RefreshToken; NewB.AccessToken = B.AccessToken; Context.ResponseObject.Data = NewB.ToJson(); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing user"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else if (Context.URLSegments[1] == "currency") { if (Context.URLSegments.Length == 3) { if (Context.URLSegments[2] == "all") { if (CorrespondingBot != null && CorrespondingBot.IsSuperBot) { Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(Data.Objects.Currency.All(true)); } else { Context.ResponseObject.Data = Newtonsoft.Json.Linq.JToken.FromObject(Data.Objects.Currency.All()); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Bot is not SuperBot"; } } else if (((Context.Headers.AllKeys.Contains("AccessToken") && Context.Headers.AllKeys.Contains("LoginID")) || CorrespondingBot != null) && Context.RequestData != null && Context.Headers.AllKeys.Contains("CurrencyID") ) { try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Login L = null; if (Context.Headers.AllKeys.Contains("LoginID")) { try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed LoginID"; return Context.ResponseObject; } L = Data.Objects.Login.FromID(int.Parse(Context.Headers["LoginID"]), true); if (!Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { L = null; } } if (L != null||CorrespondingBot!=null) { Data.Objects.Currency B = Data.Objects.Currency.FromID(int.Parse(Context.Headers["CurrencyID"])); B.LoadConfigs(true); bool LoginGood = false, BotGood = false; if (L != null) { LoginGood = B.OwnerLogin.ID == L.ID; } if (CorrespondingBot != null) { BotGood = /\*CorrespondingBot.Currency.ID == B.ID ||\*/ CorrespondingBot.IsSuperBot; } if (LoginGood||BotGood) { if (Context.RequestData["LoginConfig"] != null) { if (CorrespondingBot == null || CorrespondingBot.IsSuperBot) { if (Checks.JSONLayoutCompare( Newtonsoft.Json.Linq.JToken.Parse(System.IO.File.ReadAllText("./Data/DefaultConfigs/Login.config.json")), Context.RequestData["LoginConfig"])) { B.LoginConfig = Context.RequestData["LoginConfig"]; } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, LoginConfig does not follow the required structure"; } } } if (Context.RequestData["CommandConfig"] != null) { if (Checks.JSONLayoutCompare( Newtonsoft.Json.Linq.JToken.Parse(System.IO.File.ReadAllText("./Data/DefaultConfigs/Command.config.json")), Context.RequestData["CommandConfig"])) { B.CommandConfig = Context.RequestData["CommandConfig"]; } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ComamndConfig does not follow the required structure"; } } if (ErrorOccured == false) { B.UpdateConfigs(); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This login does not have permission to edit that Currency"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else if (Context.Headers.AllKeys.Contains("CurrencyID") && CorrespondingBot != null) { try { int.Parse(Context.Headers["CurrencyID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed CurrencyID"; return Context.ResponseObject; } Data.Objects.Currency C = Data.Objects.Currency.FromID(int.Parse(Context.Headers["CurrencyID"])); if (/\*CorrespondingBot.Currency.ID == C.ID ||\*/ CorrespondingBot.IsSuperBot) { C.LoadConfigs(true); Context.ResponseObject.Data = C.ToJson(); } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, This bot does not have permission to read that Currency"; } } else if (Context.Headers.AllKeys.Contains("AccessToken")&& Context.Headers.AllKeys.Contains("LoginID")) { try { int.Parse(Context.Headers["LoginID"]); } catch { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed LoginID"; return Context.ResponseObject; } Data.Objects.Login L = Data.Objects.Login.FromID(int.Parse(Context.Headers["LoginID"]), true); if (L != null) { if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AccessToken"], L.AccessToken)) { if (Data.Objects.Currency.FromLogin(L.ID).Count >= 5) { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, You are already at the max currency count"; } else { Data.Objects.Currency B = new Data.Objects.Currency(); B.OwnerLogin = Data.Objects.Login.FromID(L.ID); B.Save(); B = Data.Objects.Currency.FromLogin(L.ID).Last(); Context.ResponseObject.Data = B.ToJson(); } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AccessToken is invalid"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, ID does not correspond to an existing user"; } } else { ErrorOccured = true; Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, No operable Headers provided"; } } else { Context.ResponseObject.Code = 404; Context.ResponseObject.Message = "Not Found"; ErrorOccured = true; } if (ErrorOccured == false) { Context.ResponseObject.Code = 200; Context.ResponseObject.Message = "The requested task was performed successfully"; } return Context.ResponseObject; } static Data.Objects.Bot AuthCheck(StandardisedRequestObject Context) { //Check if the required Headers are present if (Context.Headers.AllKeys.Contains("AuthToken") && Context.Headers.AllKeys.Contains("BotID")) { //Check if the ID can be converted into an Integer try { int.Parse(Context.Headers["BotID"]); } catch { //If it cant be converted, set the contents of the Response Object to reflect this Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Malformed BotID"; return null; } //Fetch the Bot Object with the given ID Data.Objects.Bot Bot = Data.Objects.Bot.FromID(int.Parse(Context.Headers["BotID"]),true); if (Bot == null) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, BotID does not correspond to an object"; return null; } //Check if the provided AuthToken matches the hash in the Bot Object //And return the bot object if it is valid if (Backend.Init.ScryptEncoder.Compare(Context.Headers["AuthToken"], Bot.AccessToken)) { if (!Bot.IsSuperBot && Bot.Currency == null) { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, Bot is not authorised for any currency"; return null; } return Bot; } else { Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AuthToken is invalid for that Bot"; return null; } } else { //If a Header is missing, set the contents of the Response Object to relfect it Context.ResponseObject.Code = 400; Context.ResponseObject.Message = "Bad Request, AuthToken or BotID is missing"; return null; } } }}

Twitch-Discord-Reward-API/Backend/Networking/Checks.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend.Networking{ public static class Checks { static Char[] NumberSet = "0123456789".ToCharArray(), LowerSet = "abcdefghijklmnopqrstuvwxyz ".ToCharArray(), UpperSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ ".ToCharArray(), SpecialSet = "!\"\n£$%^&\*()-\_=+{}[]@'#~;:,./`¬? ".ToCharArray(); public static bool IsValidID(string ID)//Check if all characters in the ID string are numbers { foreach (Char C in ID) { if (!NumberSet.Contains(C)) { return false; } } return true; } public static bool IsAlphaNumericString(string Str)//Check if all characters in the String are either numbers or letters { foreach (Char C in Str) { if (!NumberSet.Contains(C) && !LowerSet.Contains(C) && !UpperSet.Contains(C)) { return false; } } return true; } public static bool IsValidPassword(string Password)//Check if the string contains at least 1 capital,number and special { bool HasNumeric = false, HasCapital = false, HasSpecial = false; foreach (Char C in Password) { if (UpperSet.Contains(C)) { HasCapital = true; } else if (NumberSet.Contains(C)) { HasNumeric = true; } else if (SpecialSet.Contains(C)) { HasSpecial = true; } } return HasCapital && HasNumeric && HasSpecial; } public static bool IsValidValueInJsonConfig(string JsonValue)//Check if the value inside the json conforms to our valid charcter set { Char PrevC = Char.MinValue; int ClosableBrackets = 0; foreach (Char C in JsonValue) { if (!LowerSet.Contains(C) && !UpperSet.Contains(C) && !NumberSet.Contains(C) && !SpecialSet.Contains(C)) {//if the character isnt Lower,Upper,Number or special if (C.ToString() == ">" && ClosableBrackets > 0) { ClosableBrackets--; }//where we have the end of a paramater decreas the closable bracket count else if (C.ToString() != "<") {//if it isnt the start or end of a bracket return false to indicate that it is invalid return false; } } else if (PrevC.ToString() == "<" && C.ToString() == "@") { ClosableBrackets++; }//Where we have a start of a paramater increase the closable bracket count PrevC = C;//Set the last character } return ClosableBrackets == 0;//If we have closed all paramater brackets } public static bool IsValidEmail(string Email)//check if the string follows an email structure { int AtCount = 0; foreach (Char C in Email) { if (C.ToString() == "@") { AtCount++; }//Increment the amount of @s in the string else if (!NumberSet.Contains(C) && !LowerSet.Contains(C) && !UpperSet.Contains(C) && C.ToString() != ".") { return false; }//if the character isnt upper,lower or number } if (AtCount != 1) { return false; }//If we have more than one @ return false to indicate it is invalid if (!Email.Split("@".ToCharArray())[1].Contains(".")) { return false; }//If the string after the @ doesnt contain a . return false to induicate it is invalid return true; } public static bool JSONLayoutCompare(Newtonsoft.Json.Linq.JToken Layout, Newtonsoft.Json.Linq.JToken Data) { bool MissingItem = false, LayoutValuesAreAlphaNumeric = true, DataValuesAreAlphaNumeric = true;//Stores data related to the conformity of the json Data List<string> LayoutPaths = new List<string> { }, DataPaths = new List<string> { }; //Perform the search of the Layout and Data jsons PerformSearch(Layout, ref LayoutPaths, ref LayoutValuesAreAlphaNumeric); PerformSearch(Data, ref DataPaths, ref DataValuesAreAlphaNumeric); foreach (string Path in LayoutPaths)//Checks if a path in the layout json does not exist in the data json { if (!DataPaths.Contains(Path)) { MissingItem = true; break; }//if a path is missing indicate there is a non-conformity } foreach (string Path in DataPaths.Where(x => x.Contains(":::")))//checks all list/array paths to ensure all conform { if (!LayoutPaths.Contains(Path)) { MissingItem = true; break; }//if a path is missing in the list/array indicate there is a non-conformity } return !MissingItem && DataValuesAreAlphaNumeric;//returns true if the values all conform and the paths all exist } //Perform a recursive search of the given json, and check if the values conform to our valid character set public static void PerformSearch(Newtonsoft.Json.Linq.JToken Item, ref List<string> Paths, ref bool ValueIsAlphaNumeric, string CurrentPath = "") { try//Try to convert the json object to a jarray { Newtonsoft.Json.Linq.JArray J = Newtonsoft.Json.Linq.JArray.FromObject(Item); for (int i = 0; i < J.Count; i++)//Perform a search of all items in the array { PerformSearch(J[i], ref Paths, ref ValueIsAlphaNumeric, CurrentPath + "::"); } } catch { try//Try to convert the json object to a jobject { Newtonsoft.Json.Linq.JObject J = Newtonsoft.Json.Linq.JObject.FromObject(Item); foreach (Newtonsoft.Json.Linq.JProperty Key in J.Properties())//Look at all properties in the jobject { if (Key.Value.HasValues)//If the property has further values { if (!Paths.Contains(CurrentPath + Key.Name + ":"))//Check if we have all ready entered the current path into the path set and adds the path if we havent { Paths.Add(CurrentPath + Key.Name + ":"); } PerformSearch(Key.Value, ref Paths, ref ValueIsAlphaNumeric, CurrentPath + Key.Name + ":");//Perform search of items inside of the property } else { if (!Paths.Contains(CurrentPath + Key.Name + ":"))//Check if we have all ready entered the current path into the path set and adds the path if we havent { Paths.Add(CurrentPath + Key.Name + ":"); } if (!IsValidValueInJsonConfig(Key.Value.ToString()))//Check if the value conforms { if (!Key.Value.ToString().StartsWith("<:") && !Key.Value.ToString().StartsWith("<a:"))//ignore the non-conformity for these cases { ValueIsAlphaNumeric = false;//Indicates a value does not conform } } } } } catch//Treat the json object as a terminating value in the json { if (!IsValidValueInJsonConfig(Item.ToString()))//check if the value conforms { if (!Item.ToString().StartsWith("<:") && !Item.ToString().StartsWith("<a:")) {//if it doesnt start with discord emote indicators ValueIsAlphaNumeric = false; //Indicate that a value does not conform } } } } } } }}

Twitch-Discord-Reward-API/Backend/Networking/ResponseObject.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend.Networking{ public class ResponseObject//This object stores the data that will be returned to the requestor { public Newtonsoft.Json.Linq.JToken Data;//This will store the json, for the data that will be returned to the requestor public int Code;//These are used in place of a code and error message in the response, to seperate errors from the backend data handling and errors with the networking public string Message; public Newtonsoft.Json.Linq.JToken ToJson()//Allows us to convert this object to json form, for transmission { return Newtonsoft.Json.Linq.JToken.FromObject(this); } }}

Twitch-Discord-Reward-API/Backend/Networking/StandardisedRequestObject.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Net;namespace Twitch\_Discord\_Reward\_API.Backend.Networking{ public class StandardisedRequestObject { /\* This Object places usefull and frequently used data in an easy to access set of variables inside of the object \* This will allow for shorter code, and by placing it in an object, the data can be kept together in a very elegant manner.\*/ public string URL,Method; public string[] URLSegments; public Dictionary<string, string> URLParamaters,StateParamaters; public System.Collections.Specialized.NameValueCollection Headers; public ResponseObject ResponseObject;//By keeping the response object and request data here, we wont need to pass it seperatly to functions public Newtonsoft.Json.Linq.JToken RequestData; public HttpListenerContext Context;//We store the original data for circumstances where the data is not stored seperatly in this object public StandardisedRequestObject(HttpListenerContext Context,ResponseObject ResponseObject) // When creating the object we will require the ListenerContext and the ResponseObject that are being used { Headers = Context.Request.Headers;//Set the objects data URL = Context.Request.RawUrl.ToLower(); Method = Context.Request.HttpMethod.ToLower(); URLSegments = URL.Split("/".ToCharArray()); URLParamaters = GetParamaters(Context.Request.RawUrl); if (Method == "post")//If the method is post, read the posted data into json format and store it { string StreamString = new System.IO.StreamReader(Context.Request.InputStream).ReadToEnd(); if (StreamString != "") { RequestData = Newtonsoft.Json.Linq.JToken.Parse(StreamString); } } this.Context = Context;//Set the objects object references this.ResponseObject = ResponseObject; } //Convert all url paramaters into a dictionary for ease of use Dictionary<string, string> GetParamaters(string URL) { Dictionary<string, string> Params = new Dictionary<string, string> { }; if (URL.Contains("?"))//Only attempt if the url does contain a ? { string[] ParamSet = URL.Split("?".ToCharArray())[1].Split("&".ToCharArray());//split the parameter string into its individual variables foreach (string Param in ParamSet)//Go through each variable and add the key and value into the dictionary { string[] SplitParam = Param.Split("=".ToCharArray()); if (SplitParam.Length == 2) { Params.Add(SplitParam[0].ToLower(), SplitParam[1]); } } } return Params; } //Converts vriables part of the url state paramater into a dictionary public void GetStateParams() { Dictionary<string, string> Params = new Dictionary<string, string> { }; string[] ParamSet = this.URLParamaters["state"].Split(new string[] { "%20","+" },StringSplitOptions.None);//split the state paramater into its sub-variables foreach (string Param in ParamSet)//Go through each sub-variable and add the key and value into the dictionary { string[] SplitParam = Param.Split(new string[] { "%3D" },StringSplitOptions.None); if (SplitParam.Length == 2) { Params.Add(SplitParam[0].ToLower(), SplitParam[1]); } } StateParamaters = Params; } }}

Twitch-Discord-Reward-API/Backend/Networking/TokenSystem.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend.Networking{ public static class TokenSystem { static Char[] TokenChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray(); public static string CreateToken(int Length) { string S = ""; for (int i = 0; i < Length; i++)//While we havent reached the given length, add a random character from alphabet to the return string { S += TokenChars[Init.Rnd.Next(0, TokenChars.Length)]; } return S; } }}

Twitch-Discord-Reward-API/Backend/Init.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API.Backend{ public static class Init { public static Random Rnd = new Random(); public static Data.SQL SQLi = new Data.SQL("./Data/Database"); // Create an instance of the sql object, that will be used everywhere public static Newtonsoft.Json.Linq.JToken APIConfig = Data.FileManager.ReadFile("./Data/Api.config.json"); // Read the API's master config from storage public static Scrypt.ScryptEncoder ScryptEncoder = new Scrypt.ScryptEncoder(); // Create an instance of the ScryptEncoder public static void Start() { Networking.HTTPServer.Init.Start(); // Start the HTTPServer while (true) { Console.ReadLine(); } } }}

Twitch-Discord-Reward-API/Program.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_API{ class Program { static void Main(string[] args) { Backend.Init.Start(); } }}