Twitch-Discord-Reward-Bot/Backend/Bots/Commands/CommandHandler.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading;using System.Threading.Tasks;using TwitchLib.Client;using TwitchLib.Client.Models;using TwitchLib.Client.Events;using Discord.WebSocket;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots.Commands{ using Objects = Data.APIIntergrations.RewardCurrencyAPI.Objects; public class CommandHandler : BaseObject { public CommandHandler(BotInstance BotInstance) : base(BotInstance) { } public void Handle(object sender, OnMessageReceivedArgs e) { if (e.ChatMessage.Username != BotInstance.TwitchBot.Client.TwitchUsername) { Handle(sender, StandardisedMessageRequest.FromTwitch(e, BotInstance)); } } public async Task Handle(SocketMessage e) { if (e.Author.Id != BotInstance.DiscordBot.Client.CurrentUser.Id) { Handle(null, StandardisedMessageRequest.FromDiscord(e, BotInstance)); } } public void Handle(object sender, StandardisedMessageRequest e) { new Thread(async () => await HandleThread(e)).Start(); } public Dictionary<string, string> SongRequestHistory = new Dictionary<string, string> { }; async Task HandleThread(StandardisedMessageRequest e) { if (BotInstance.TimeEvents == null) { return; } //var C1 = BotInstance.DiscordBot.Client.GetChannel(546382361151930388); //var C2 = (ISocketMessageChannel)(C1); //var M = await C2.GetMessageAsync(548057440898514945); //await C2.DeleteMessageAsync(M); try { RewardForChatting(e); #region "Welcome" if (e.IsNewUser) { if (CommandEnabled(BotInstance.CommandConfig["WelcomeMessage"],e.MessageType)) { if (LiveCheck(BotInstance.CommandConfig["WelcomeMessage"])) { if (e.MessageType == MessageType.Twitch) { await SendMessage(BotInstance.CommandConfig["WelcomeMessage"]["TwitchWelcome"].ToString(), e); } if (e.MessageType == MessageType.Discord) { await SendMessage(BotInstance.CommandConfig["WelcomeMessage"]["DiscordWelcome"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), e.MessageType,e.User); } } } } #endregion #region "Commands" if (e.SenderID != BotInstance.DiscordBot.Client.CurrentUser.Id.ToString()) { string Prefix = BotInstance.CommandConfig["Prefix"].ToString(), Command = e.SegmentedBody[0].Replace(Prefix, "").ToLower(); bool StopAfterNotifcations = false; if (e.MessageType == MessageType.Discord && BotInstance.CommandConfig["Discord"]["Channels"].Where(x => x.ToString() == e.ChannelID).Count() == 0) { if (e.DiscordRaw.Channel.GetType() == typeof(SocketDMChannel)) { StopAfterNotifcations = true; } else { return; } } if (e.SegmentedBody[0].StartsWith(Prefix) && !e.SegmentedBody[0].StartsWith(Prefix + Prefix)) { bool CommandWasValid = true; if (CheckRateLimit(e)) { return; } else { UpdateRateLimit(e); } Objects.Viewer.MergeAccounts(e, BotInstance, e.SenderID); #region "Viewer" #region "Notifications" if (CommandEnabled(BotInstance.CommandConfig["LiveNotifications"], e) && JArrayContainsString(BotInstance.CommandConfig["LiveNotifications"]["Commands"], Command)) { if (e.SegmentedBody.Length == 2) { Objects.Viewer V = e.Viewer; if (e.SegmentedBody[1].ToLower() == "on") { V.LiveNotifcations = true; await SendMessage(BotInstance.CommandConfig["LiveNotifications"]["Responses"]["On"].ToString(), e); } else if (e.SegmentedBody[1].ToLower() == "off") { V.LiveNotifcations = false; await SendMessage(BotInstance.CommandConfig["LiveNotifications"]["Responses"]["Off"].ToString(), e); } List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("ID",V.ID.ToString()), new KeyValuePair<string, string>("Notifications",V.LiveNotifcations.ToString()) }; Data.APIIntergrations.RewardCurrencyAPI.WebRequests.PostRequest("viewer", Headers, true); } } #endregion else if (StopAfterNotifcations) { return; } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Balance"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Balance"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Balance"])) { if (e.SegmentedBody.Length == 1) { if (e.Viewer != null) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Balance"]["Responses"]["OwnBalance"].ToString(), e, null, e.Viewer.Balance); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else if (e.SegmentedBody.Length == 2) { StandardisedUser U = IDFromMessageSegment(e.SegmentedBody[1], e); if (U != null) { Objects.Viewer B = Objects.Viewer.FromTwitchDiscord(e, BotInstance, U.ID); if (B != null) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Balance"]["Responses"]["OtherBalance"].ToString(), e, U, B.Balance); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["CannotFindUser"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["WatchTime"],e)&& JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["WatchTime"]["Commands"], Command)) { if (e.SegmentedBody.Length == 1) { if (e.Viewer != null) { string Duration = AgeString(TimeSpan.FromMinutes(e.Viewer.WatchTime)); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["WatchTime"]["Responses"]["Self"].ToString(), e, OtherString:Duration); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else if (e.SegmentedBody.Length == 2) { StandardisedUser U = IDFromMessageSegment(e.SegmentedBody[1], e); if (U != null) { Objects.Viewer B = Objects.Viewer.FromTwitchDiscord(e, BotInstance, U.ID); if (B != null) { string Duration = AgeString(TimeSpan.FromMinutes(B.WatchTime)); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["WatchTime"]["Responses"]["Other"].ToString(), e, U, OtherString: Duration); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Pay"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Pay"])) { if (e.SegmentedBody.Length == 3) { StandardisedUser U = IDFromMessageSegment(e.SegmentedBody[1], e); if (U.ID != null) { Objects.Viewer Self = e.Viewer, Other = Objects.Viewer.FromTwitchDiscord(e, BotInstance, U.ID); int ChangeBy = ValueFromMessageSegment(e.SegmentedBody[2], Self), MinPayment = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Pay"]["MinimumPayment"].ToString()); if (ChangeBy == -1) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); return; } if (Self != null && Other != null) { if (ChangeBy >= 0) { if (ChangeBy >= MinPayment) { if (Self.Balance - ChangeBy >= 0) { if (Objects.Viewer.AdjustBalance(Self, ChangeBy, "-")) { if (Objects.Viewer.AdjustBalance(Other, ChangeBy, "+")) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["Paid"].ToString(), e, U, ChangeBy); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["NotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["TooSmall"].ToString(), e, null, MinPayment); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterNegative"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["CannotFindUser"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Gamble"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Gamble"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Gamble"])) { if (e.SegmentedBody.Length == 2) { Objects.Viewer Self = e.Viewer; int ChangeBy = ValueFromMessageSegment(e.SegmentedBody[1], Self), MinPayment = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Gamble"]["MinimumPayment"].ToString()), WinChance = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Gamble"]["WinChance"].ToString()), WinMultiplyer = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Gamble"]["WinMultiplyer"].ToString()); if (ChangeBy == -1) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); return; } if (Self != null) { if (ChangeBy >= 0) { if (ChangeBy >= MinPayment) { if (Self.Balance - ChangeBy >= 0) { string Operator; if (Init.Rnd.Next(0, 100) <= WinChance) { Operator = "+"; ChangeBy \*= WinMultiplyer; } else { Operator = "-"; } if (Objects.Viewer.AdjustBalance(Self, ChangeBy, Operator)) { if (Operator == "+") { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Gamble"]["Responses"]["Win"].ToString(), e, null, ChangeBy); } else if (Operator == "-") { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Gamble"]["Responses"]["Lose"].ToString(), e, null, ChangeBy); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["NotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["TooSmall"].ToString(), e, null, MinPayment); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterNegative"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Slots"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Slots"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Slots"])) { if (e.SegmentedBody.Length == 2) { Objects.Viewer Self = e.Viewer; int ChangeBy = ValueFromMessageSegment(e.SegmentedBody[1], Self), MinPayment = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Slots"]["MinimumPayment"].ToString()), WinChance = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Slots"]["WinChance"].ToString()), WinMultiplyer = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Slots"]["WinMultiplyer"].ToString()); if (ChangeBy == -1) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); return; } if (Self != null) { if (ChangeBy >= 0) { if (ChangeBy >= MinPayment) { if (Self.Balance - ChangeBy >= 0) { string Operator; if (Init.Rnd.Next(0, 100) <= WinChance) { Operator = "+"; ChangeBy \*= WinMultiplyer; } else { Operator = "-"; } if (Objects.Viewer.AdjustBalance(Self, ChangeBy, Operator)) { Newtonsoft.Json.Linq.JToken EmoteSet = null; if (e.MessageType == MessageType.Discord) { EmoteSet = BotInstance.CommandConfig["CommandSetup"]["Slots"]["Emotes"]["Discord"]; } if (e.MessageType == MessageType.Twitch) { EmoteSet = BotInstance.CommandConfig["CommandSetup"]["Slots"]["Emotes"]["Twitch"]; } if (Operator == "+") { int i = Init.Rnd.Next(0, EmoteSet.Count()); string PanelString = "[ " + EmoteSet[i].ToString() + " | " + EmoteSet[i].ToString() + " | " + EmoteSet[i].ToString() + " ]"; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Slots"]["Responses"]["Win"].ToString(), e, null, ChangeBy, -1, PanelString); } else if (Operator == "-") { string[] PanelArray = new string[] { "", "", "" }; while (PanelArray[0] == PanelArray[1] && PanelArray[1] == PanelArray[2]) { PanelArray = new string[] { EmoteSet[Init.Rnd.Next(0, EmoteSet.Count())].ToString(), EmoteSet[Init.Rnd.Next(0, EmoteSet.Count())].ToString(), EmoteSet[Init.Rnd.Next(0, EmoteSet.Count())].ToString() }; } await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Slots"]["Responses"]["Lose"].ToString(), e, null, ChangeBy, -1, "[ " + PanelArray[0] + " | " + PanelArray[1] + " | " + PanelArray[2] + " ]"); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["NotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["TooSmall"].ToString(), e, null, MinPayment); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterNegative"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Fish"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Fish"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Fish"])) { if (e.SegmentedBody.Length == 1) { Objects.Viewer Self = e.Viewer; int ViewerCost = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Fish"]["Cost"]["Viewer"].ToString()), SubscriberCost = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Fish"]["Cost"]["Subscriber"].ToString()); int Cost = ViewerCost; if (Self != null) { if (IsSubscriber(e)) { Cost = SubscriberCost; } if (Self.Balance - Cost >= 0) { if (BotInstance.TimeEvents.Fishermen.Where(x => x.Value.e.SenderID == e.SenderID).Count() == 0) { if (Objects.Viewer.AdjustBalance(Self, Cost, "-")) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Fish"]["Responses"]["Started"].ToString(), e); BotInstance.TimeEvents.Fishermen.Add(DateTime.Now, new Fisherman(e, BotInstance)); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Fish"]["Responses"]["AlreadyFishing"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Pay"]["Responses"]["NotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Duel"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Duel"])) { if (e.SegmentedBody.Length == 3) { int MinimumPayment = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Duel"]["MinimumPayment"].ToString()); StandardisedUser Target = IDFromMessageSegment(e.SegmentedBody[1], e); if (Target != null) { Objects.Viewer Self = e.Viewer, TargetBank = Objects.Viewer.FromTwitchDiscord(e, BotInstance, Target.ID); if (Self != null && TargetBank != null) { int ChangeBy = ValueFromMessageSegment(e.SegmentedBody[2], Self), TargetChangeBy = ValueFromMessageSegment(e.SegmentedBody[2], TargetBank); if (ChangeBy != -1 && TargetChangeBy != -1) { if (TargetChangeBy < ChangeBy) { ChangeBy = TargetChangeBy; } if (ChangeBy >= 0) { if (ChangeBy >= MinimumPayment) { if (ChangeBy <= Self.Balance) { if (ChangeBy <= TargetBank.Balance) { Duel Duel = new Duel(); Duel.BotInstance = BotInstance; StandardisedUser S = new StandardisedUser(); S.ID = e.SenderID; S.UserName = e.SenderUserName; Duel.Creator = S; Duel.Acceptor = Target; if (!BotInstance.TimeEvents.UserDueling(S)) { if (!BotInstance.TimeEvents.UserDueling(Target)) { Duel.e = e; Duel.ChangeBy = ChangeBy; BotInstance.TimeEvents.UserDueling(S); BotInstance.TimeEvents.Duels.Add(DateTime.Now, Duel); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Responses"]["Started"].ToString(), e, Target, ChangeBy); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Responses"]["OtherDueling"].ToString(), e, Target); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Responses"]["SelfDueling"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Responses"]["OtherNotEnough"].ToString(), e, Target); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Responses"]["SelfNotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Responses"]["TooSmall"].ToString(), e, null, MinimumPayment); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterNegative"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["CannotFindUser"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Duel"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Commands"], Command)) { if (BotInstance.TimeEvents.UserDueling(e.User)) { KeyValuePair<DateTime, Duel> TDuel = BotInstance.TimeEvents.Duels.Where(x => x.Value.Acceptor.ID == e.User.ID || x.Value.Creator.ID == e.User.ID).First(); if (TDuel.Value.Acceptor.ID == e.User.ID) { BotInstance.TimeEvents.Duels.Remove(TDuel.Key); Objects.Viewer Acceptor = e.Viewer, Creator = Objects.Viewer.FromTwitchDiscord(e, BotInstance, TDuel.Value.Creator.ID); if (Acceptor != null && Creator != null) { if (TDuel.Value.ChangeBy <= Acceptor.Balance) { if (TDuel.Value.ChangeBy <= Creator.Balance) { int Winner = Init.Rnd.Next(0, 2); if (Winner == 0) { Objects.Viewer.AdjustBalance(Acceptor, TDuel.Value.ChangeBy, "+"); Objects.Viewer.AdjustBalance(Creator, TDuel.Value.ChangeBy, "-"); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Responses"]["Win"].ToString(), e, TDuel.Value.Creator, TDuel.Value.ChangeBy); } if (Winner == 1) { Objects.Viewer.AdjustBalance(Acceptor, TDuel.Value.ChangeBy, "-"); Objects.Viewer.AdjustBalance(Creator, TDuel.Value.ChangeBy, "+"); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Responses"]["Lose"].ToString(), e, TDuel.Value.Creator, TDuel.Value.ChangeBy); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Responses"]["OtherNotEnough"].ToString(), e, TDuel.Value.Creator); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Responses"]["SelfNotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Responses"]["SelfAccept"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Accepting"]["Responses"]["NotDueling"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Duel"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Denying"]["Commands"], Command)) { if (BotInstance.TimeEvents.UserDueling(e.User)) { KeyValuePair<DateTime, Duel> TDuel = BotInstance.TimeEvents.Duels.Where(x => x.Value.Acceptor.ID == e.User.ID || x.Value.Creator.ID == e.User.ID).First(); BotInstance.TimeEvents.Duels.Remove(TDuel.Key); if (e.SenderID == TDuel.Value.Acceptor.ID) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Denying"]["Responses"]["Canceled"].ToString(), e, TDuel.Value.Creator); } if (e.SenderID == TDuel.Value.Creator.ID) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Denying"]["Responses"]["Canceled"].ToString(), e, TDuel.Value.Acceptor); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Duel"]["Denying"]["Responses"]["NotDueling"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["Raffle"], e) && JArrayContainsString(BotInstance.CommandConfig["Raffle"]["Joining"]["Commands"], Command)) { if (!BotInstance.TimeEvents.UserRaffleing(e.User)) { Raffler R = new Raffler(); R.User = e.User; R.RequestedFrom = e.MessageType; BotInstance.TimeEvents.RaffleParticipants.Add(R); if (BotInstance.CommandConfig["Raffle"]["Joining"]["Responses"]["Joined"].ToString() != "") { await SendMessage(BotInstance.CommandConfig["Raffle"]["Joining"]["Responses"]["Joined"].ToString(), e); } } else if (BotInstance.CommandConfig["Raffle"]["Joining"]["Responses"]["AlreadyRaffling"].ToString() != "") { await SendMessage(BotInstance.CommandConfig["Raffle"]["Joining"]["Responses"]["AlreadyRaffling"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Alert"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Alert"]["Commands"], Command)) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["Alert"])) { if (e.SegmentedBody.Length > 1) { if (BotInstance.TimeEvents.AlertTimeOutExpired(e.User)) { string AlertName = e.MessageBody.Replace(e.SegmentedBody[0] + " ", ""); Dictionary<int, int> MostSuitableAlert = new Dictionary<int, int> { }; for (int iAlert = 0; iAlert < BotInstance.CommandConfig["CommandSetup"]["Alert"]["Alerts"].Count(); iAlert++) { MostSuitableAlert.Add(iAlert, AlertName.Split(" ".ToCharArray()).Count( x => BotInstance.CommandConfig["CommandSetup"]["Alert"]["Alerts"][iAlert]["Name"].ToString().Split(" ".ToCharArray()).Contains(x) || BotInstance.CommandConfig["CommandSetup"]["Alert"]["Alerts"][iAlert]["Name"].ToString().Split(" ".ToCharArray()).Count(y => y.Contains(x)) != 0 )); } if (MostSuitableAlert.Count == 0) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Alert"]["Responses"]["InvalidAlert"].ToString(), e); return; } KeyValuePair<int, int> ChosenAlert = new KeyValuePair<int, int>(0, MostSuitableAlert.First().Value); foreach (int Key in MostSuitableAlert.Keys) { if (ChosenAlert.Value < MostSuitableAlert[Key]) { ChosenAlert = new KeyValuePair<int, int>(Key, MostSuitableAlert[Key]); } } Newtonsoft.Json.Linq.JToken Alert = BotInstance.CommandConfig["CommandSetup"]["Alert"]["Alerts"][ChosenAlert.Key]; Objects.Viewer V = e.Viewer; if (V != null) { int Cost = int.Parse(Alert["Cost"].ToString()); if (V.Balance >= Cost) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Streamlabs.PlayAlert(BotInstance, Alert["URL"].ToString()); if (JData["success"] != null) { Objects.Viewer.AdjustBalance(V, Cost, "-"); Alerter A = new Alerter(); A.User = e.User; A.LastAlert = DateTime.Now; BotInstance.TimeEvents.AlertRequests.Add(A); BotInstance.TimeEvents.LastAlert = DateTime.Now; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Alert"]["Responses"]["Requested"].ToString(), e, OtherString: Alert["Name"].ToString()); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Alert"]["Respomses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Alert"]["Responses"]["NotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Alert"]["Responses"]["TimeOut"].ToString(), e, Amount: BotInstance.TimeEvents.GetRemainingCooldown(e.User)); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["UpTime"], e)&& JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["UpTime"]["Commands"],Command)) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Twitch.GetStream(BotInstance); if (JData != null) { if (JData["stream"].HasValues) { string Duration = AgeString(DateTime.Now-DateTime.Parse(JData["stream"]["created\_at"].ToString())); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["UpTime"]["Responses"]["LiveFor"].ToString(), e,OtherString:Duration); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["UpTime"]["Responses"]["NotLive"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["UpTime"]["Respomses"]["APIError"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["LatestVid"],e)&& JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["LatestVid"]["Commands"], Command)) { string Video = Data.APIIntergrations.Youtube.LatestVid(BotInstance); if (Video != null) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["LatestVid"]["Responses"]["LatestVid"].ToString(), e, OtherString: Video); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["LatestVid"]["Responses"]["APIError"].ToString(), e); } } #endregion #region "NightBot" else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Playlist"]["Commands"], Command)) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Playlist"]["Response"].ToString(), e); } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Queue"]["Commands"], Command)) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Queue"]["Response"].ToString(), e); } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Commands"], Command)) { if (e.SegmentedBody.Length >= 2) { string Request = e.MessageBody.Replace(e.SegmentedBody[0] + " ", ""); int Cost = int.Parse(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Cost"]["Viewer"].ToString()), SubscriberCost = int.Parse(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Cost"]["Subscriber"].ToString()); if (IsSubscriber(e)) { Cost = SubscriberCost; } Objects.Viewer B = e.Viewer; if (B.Balance >= Cost) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.RequestSong(BotInstance, Request); if (JData["status"].ToString() == "200") { if (!SongRequestHistory.ContainsKey(e.SenderID)) { SongRequestHistory.Add(e.SenderID, JData["item"]["\_id"].ToString()); } else { SongRequestHistory[e.SenderID] = JData["item"]["\_id"].ToString(); } Objects.Viewer.AdjustBalance(B, Cost, "-"); string MessageContent = JData["item"]["track"]["title"] + " by " + JData["item"]["track"]["artist"] + " -- " + JData["item"]["track"]["url"]; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Responses"]["Requested"].ToString(), e, Amount: int.Parse(JData["item"]["\_position"].ToString()), OtherString: MessageContent); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Responses"]["NotEnough"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Cancel"]["Commands"], Command)) { if (SongRequestHistory.ContainsKey(e.SenderID)) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.GetQueue(BotInstance); if (JData["status"].ToString() == "200") { if (JData["queue"].Where(x => x["\_id"].ToString() == SongRequestHistory[e.SenderID]).Count() != 0) { JData = Data.APIIntergrations.Nightbot.RemoveID(BotInstance, SongRequestHistory[e.SenderID]); if (JData["status"].ToString() == "200") { int Cost = int.Parse(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Cost"]["Viewer"].ToString()), SubscriberCost = int.Parse(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Request"]["Cost"]["Subscriber"].ToString()); if (IsSubscriber(e)) { Cost = SubscriberCost; } Objects.Viewer.AdjustBalance(e.Viewer, Cost, "-"); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Cancel"]["Responses"]["CanceledSong"].ToString(), e); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Cancel"]["Responses"]["SongDoesntExist"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Cancel"]["Responses"]["NoSong"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Current"]["Commands"], Command)) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.GetQueue(BotInstance); if (JData["status"].ToString() == "200") { if (JData["\_currentSong"].HasValues) { string MessageContent = JData["\_currentSong"]["track"]["title"] + " by " + JData["\_currentSong"]["track"]["artist"] + " -- " + JData["\_currentSong"]["track"]["url"]; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Current"]["Responses"]["CurrentlyPlaying"].ToString(), e, OtherString: MessageContent); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Current"]["Responses"]["NotPlaying"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } #endregion #region "Moderator" else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["SetTitle"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["SetTitle"]["Commands"], Command)) { if (IsModerator(e)) { string Title = e.MessageBody.Replace(e.SegmentedBody[0] + " ", "").Replace(e.SegmentedBody[0],""); Data.APIIntergrations.Twitch.UpdateChannelTitle(BotInstance, Title); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["SetTitle"]["Responses"]["SetTitle"].ToString(), e, null, -1, -1, Title); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["SetGame"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["SetGame"]["Commands"], Command)) { if (IsModerator(e)) { string Game = e.MessageBody.Replace(e.SegmentedBody[0] + " ", "").Replace(e.SegmentedBody[0], ""); Data.APIIntergrations.Twitch.UpdateChannelGame(BotInstance, Game); await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["SetGame"]["Responses"]["SetGame"].ToString(), e, null, -1, -1, Game); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["GiveCoin"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["GiveCoin"]["Commands"], Command)) { if (IsModerator(e)) { if (e.SegmentedBody.Length == 3) { Objects.Viewer B = e.Viewer; int ChangeBy = ValueFromMessageSegment(e.SegmentedBody[2], B); if (ChangeBy != -1) { if (ChangeBy >= 0) { StandardisedUser S = null; if (e.MessageType == MessageType.Twitch) { S = StandardisedUser.FromTwitchUsername(e.SegmentedBody[1], BotInstance); } if (e.MessageType == MessageType.Discord) { S = StandardisedUser.FromDiscordMention(e.SegmentedBody[1], BotInstance); } if (S != null) { B = Objects.Viewer.FromTwitchDiscord(e.MessageType, BotInstance, S.ID); if (B != null) { if (Objects.Viewer.AdjustBalance(B, ChangeBy, "+")) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["GiveCoin"]["Responses"]["Gave"].ToString(), e, S, ChangeBy); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["LatestVid"]["Responses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["LatestVid"]["Responses"]["CannotFindUser"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterNegative"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Pause"]["Commands"], Command)) { if (IsModerator(e)) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.PauseSong(BotInstance); if (JData["status"].ToString() == "200") { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Pause"]["Response"].ToString(), e); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Play"]["Commands"], Command)) { if (IsModerator(e)) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.PlaySong(BotInstance); if (JData["status"].ToString() == "200") { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Play"]["Response"].ToString(), e); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Skip"]["Commands"], Command)) { if (IsModerator(e)) { Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.SkipSong(BotInstance); if (JData["status"].ToString() == "200") { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Skip"]["Response"].ToString(), e); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Remove"]["Commands"], Command)) { if (IsModerator(e)) { if (e.SegmentedBody.Length == 2) { try { int.Parse(e.SegmentedBody[1]); } catch { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); return; } Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.RemoveItem(BotInstance, int.Parse(e.SegmentedBody[1])); if (JData["status"].ToString() == "200") { string MessageContent = JData["item"]["track"]["title"] + " by " + JData["item"]["track"]["artist"] + " -- " + JData["item"]["track"]["url"]; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Remove"]["Response"].ToString(), e, OtherString: MessageContent); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["NightBot"], e) && JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Volume"]["Commands"], Command)) { if (IsModerator(e)) { if (e.SegmentedBody.Length == 2) { try { int.Parse(e.SegmentedBody[1]); } catch { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["NumberParamaterInvalid"].ToString(), e); return; } int Volume = int.Parse(e.SegmentedBody[1]); Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Nightbot.SetVolume(BotInstance, Volume); if (JData["status"].ToString() == "200") { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Moderator"]["Volume"]["Response"].ToString(), e, Amount: Volume); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["NightBot"]["Responses"]["APIError"].ToString(), e, OtherString: JData["message"].ToString()); } } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["DontReward"], e)&& JArrayContainsString(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["DontReward"]["Commands"],Command)) { if (IsModerator(e)) { if (e.SegmentedBody.Length == 3) { StandardisedUser Target = IDFromMessageSegment(e.SegmentedBody[1], e); if (Target != null) { Objects.Viewer Viewer = Objects.Viewer.FromTwitchDiscord(e, BotInstance, Target.ID); if (Viewer != null) { if (e.SegmentedBody[2].ToLower() == "on") { Viewer.DontReward = false; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["DontReward"]["Responses"]["On"].ToString(), e,Target); } else if (e.SegmentedBody[2].ToLower() == "off") { Viewer.DontReward = true; await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["DontReward"]["Responses"]["Off"].ToString(), e,Target); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["DontReward"]["Responses"]["InvalidState"].ToString(), e); return; } List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("ID",Viewer.ID.ToString()), new KeyValuePair<string, string>("DontReward",Viewer.DontReward.ToString()) }; Data.APIIntergrations.RewardCurrencyAPI.WebRequests.PostRequest("viewer", Headers, true); } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["APIError"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["CannotFindUser"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["ErrorResponses"]["ParamaterCount"].ToString(), e); } } else { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["Moderator"]["Responses"]["NotMod"].ToString(), e); } } #endregion else if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["SimpleResponses"], e) && BotInstance.CommandConfig["CommandSetup"]["SimpleResponses"]["Commands"][Command.ToLower()] != null) { if (LiveCheck(BotInstance.CommandConfig["CommandSetup"]["SimpleResponses"])) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["SimpleResponses"]["Commands"][Command.ToLower()].ToString(), e); } } else { CommandWasValid = false; if (CommandEnabled(BotInstance.CommandConfig["CommandSetup"]["FallbackMessage"], e)) { await SendMessage(BotInstance.CommandConfig["CommandSetup"]["FallbackMessage"]["Response"].ToString(), e); } } if (!CommandWasValid) { RevertRateLimit(e); } } } #endregion } catch (Exception E) { Console.WriteLine(E); } } Dictionary<string, DateTime> MessageHistory = new Dictionary<string, DateTime> { }; Dictionary<string, DateTime> PreviousMessageHistory = new Dictionary<string, DateTime> { }; public bool CheckRateLimit(StandardisedMessageRequest e) { int Delay = int.Parse(BotInstance.CommandConfig["MessageDelay"].ToString()); if (Delay < 3) { Delay = 3; } return MessageHistory.Where(x => x.Key == e.SenderID && (((TimeSpan)(DateTime.Now - x.Value)).TotalSeconds <= Delay)).Count() != 0; } public void UpdateRateLimit(StandardisedMessageRequest e) { if (MessageHistory.Where(x => x.Key == e.SenderID).Count() == 0) { MessageHistory.Add(e.SenderID, DateTime.Now); } else { if (PreviousMessageHistory.Where(x => x.Key == e.SenderID).Count() != 0) { PreviousMessageHistory[e.SenderID] = MessageHistory[e.SenderID]; } else { PreviousMessageHistory.Add(e.SenderID, MessageHistory[e.SenderID]); } MessageHistory[e.SenderID] = DateTime.Now; } } public void RevertRateLimit(StandardisedMessageRequest e) { if (PreviousMessageHistory.Where(x => x.Key == e.SenderID).Count() == 0) { MessageHistory[e.SenderID] = PreviousMessageHistory[e.SenderID]; } else { MessageHistory[e.SenderID] = DateTime.MinValue; } } public void RewardForChatting(StandardisedMessageRequest e) { if (BotInstance.TimeEvents != null) { if (!e.Viewer.DontReward) { IEnumerable<Viewer> Vs = BotInstance.TimeEvents.ViewerRewardTracking.Where(x => x.User.ID == e.SenderID); if (Vs.Count() != 0) { Viewer V = Vs.First(); int TwitchDelay = int.Parse(BotInstance.CommandConfig["AutoRewards"]["Chatting"]["Twitch"]["Interval"].ToString()), TwitchReward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["Chatting"]["Twitch"]["Reward"].ToString()), DiscordDelay = int.Parse(BotInstance.CommandConfig["AutoRewards"]["Chatting"]["Discord"]["Interval"].ToString()), DiscordReward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["Chatting"]["Discord"]["Reward"].ToString()); if (e.MessageType == MessageType.Twitch) { if (((TimeSpan)(DateTime.Now - V.LastTwitchMessage)).TotalSeconds >= TwitchDelay) { V.LastTwitchMessage = DateTime.Now; if (e.Viewer != null) { Objects.Viewer.AdjustBalance(e.Viewer, TwitchReward, "+"); } } } else if (e.MessageType == MessageType.Discord) { if (((TimeSpan)(DateTime.Now - V.LastTwitchMessage)).TotalSeconds >= DiscordDelay) { V.LastTwitchMessage = DateTime.Now; if (e.Viewer != null) { Objects.Viewer.AdjustBalance(e.Viewer, DiscordReward, "+"); } } } } else { Viewer V = new Viewer(); StandardisedUser U = new StandardisedUser(); V.User = e.User; BotInstance.TimeEvents.ViewerRewardTracking.Add(V); RewardForChatting(e); } } } } public StandardisedUser IDFromMessageSegment(string MessageSegment, StandardisedMessageRequest e) { if (e.MessageType == MessageType.Discord) { return StandardisedUser.FromDiscordMention(MessageSegment, BotInstance); } else if (e.MessageType == MessageType.Twitch) { StandardisedUser S = StandardisedUser.FromTwitchUsername(MessageSegment,BotInstance); return S; } return null; } public int ValueFromMessageSegment(string MessageSegment,Objects.Viewer Bank) { try { return int.Parse(MessageSegment); } catch { } if (MessageSegment.ToLower() == "all") { return Bank.Balance; } if (MessageSegment.ToLower().EndsWith("k")) { try { return int.Parse(MessageSegment.ToLower().Replace("k", "")) \* 1000; } catch { } } if (MessageSegment.ToLower().EndsWith("m")) { try { return int.Parse(MessageSegment.ToLower().Replace("m", "")) \* 1000000; } catch { } } return -1; } public bool JArrayContainsString(Newtonsoft.Json.Linq.JToken Array,string S) { foreach (Newtonsoft.Json.Linq.JToken Item in Array) { if (Item.ToString() == S) { return true; } } return false; } public bool CommandEnabled(Newtonsoft.Json.Linq.JToken Command,StandardisedMessageRequest e) { return CommandEnabled(Command, e.MessageType); } public bool CommandEnabled(Newtonsoft.Json.Linq.JToken Command, MessageType e) { if (e == MessageType.Discord) { if (Command["DiscordEnabled"].ToString().ToLower() == "true") { return true; } } if (e == MessageType.Twitch) { if (Command["TwitchEnabled"].ToString().ToLower() == "true") { return true; } } return false; } public string AgeString(TimeSpan Span) { string Age = ""; int Years = (int)Math.Floor((decimal)Span.Days / 365); int Months = (int)Math.Floor((decimal)(Span.Days - (Years \* 365)) / 30); int Days = Span.Days - ((Years \* 365) + (Months \* 30)); if (Years != 0) { if (Years == 1) { Age += Years + " Year"; } else { Age += Years + " Years"; } } if (Months != 0 && Days == 0 && Span.Hours == 0&&Span.Minutes== 0 && Age != "") { Age += " and "; } if (Months != 0) { if (Age != "") { Age += " "; } if (Months == 1) { Age += Months + " Month"; } else { Age += Months + " Months"; } } if (Days != 0 && Span.Hours == 0&&Span.Minutes== 0 && Age != "") { Age += " and "; } if (Days != 0) { if (Age != "") { Age += " "; } if (Days == 1) { Age += Days + " Day"; } else { Age += Days + " Days"; } } if (Span.Hours != 0 && Span.Minutes == 0 && Age != "") { Age += " and "; } if (Span.Hours != 0) { if (Age != "") { Age += " "; } if (Span.Hours == 1) { Age += Span.Hours + " Hour"; } else { Age += Span.Hours + " Hours"; } } if (Span.Minutes != 0 && Age!="") { Age += " and "; } if (Span.Minutes != 0) { if (Age != "") { Age += " "; } if (Span.Minutes == 1) { Age += Span.Minutes + " Minute"; } else { Age += Span.Minutes + " Minutes"; } } if (Age == "") { Age = "0 Minutes"; } return Age; } public bool IsModerator(StandardisedMessageRequest e) { if (e.MessageType == MessageType.Twitch) { return e.TwitchRaw.ChatMessage.IsModerator || e.TwitchRaw.ChatMessage.IsBroadcaster; } else if (e.MessageType == MessageType.Discord) { return ((SocketGuildUser)e.DiscordRaw.Author).Roles.Where(x => x.Id.ToString() == BotInstance.CommandConfig["Discord"]["ModeratorRoleID"].ToString()).Count() != 0; } return false; } public bool IsSubscriber(StandardisedMessageRequest e) { if (e.MessageType == MessageType.Twitch) { if (e.TwitchRaw.ChatMessage.IsSubscriber) { return true; } } if (e.MessageType == MessageType.Discord) { if (((SocketGuildUser)e.DiscordRaw.Author).Roles.Where(x => x.Id.ToString() == BotInstance.CommandConfig["Discord"]["SubscriberRoleID"].ToString()).Count() != 0) { return true; } } return false; } public bool LiveCheck(Newtonsoft.Json.Linq.JToken Object) { if (Object["RequireLive"].ToString().ToLower() == "true") { return Data.APIIntergrations.Twitch.IsLive(BotInstance); } return true; } public async Task SendMessage(string ParamaterisedMessage, StandardisedMessageRequest e, StandardisedUser TargetUser = null, int Amount = -1, int NewBal = -1, string OtherString = "", string SenderUsername = null) { ParamaterisedMessage = MessageParser(ParamaterisedMessage, e, e.MessageType,TargetUser, Amount, NewBal, OtherString, SenderUsername); if (e.MessageType == MessageType.Twitch) { BotInstance.TwitchBot.Client.SendMessage(e.ChannelName, ParamaterisedMessage); } else { await e.DiscordRaw.Channel.SendMessageAsync(ParamaterisedMessage); } } public async Task SendMessage(string ParamaterisedMessage, string Channel, MessageType MessageType, StandardisedUser TargetUser = null, int Amount = -1, int NewBal = -1, string OtherString = "", string SenderUsername = null) { string ServerID = null; if (MessageType == MessageType.Discord) { ServerID = ((SocketGuildChannel)BotInstance.DiscordBot.Client.GetChannel(ulong.Parse(Channel))).Guild.Id.ToString(); } ParamaterisedMessage = MessageParser(ParamaterisedMessage, null, MessageType, TargetUser, Amount, NewBal, OtherString, SenderUsername,ServerID); if (MessageType == MessageType.Twitch) { BotInstance.TwitchBot.Client.SendMessage(Channel, ParamaterisedMessage); } else { await ((ISocketMessageChannel)BotInstance.DiscordBot.Client.GetChannel(ulong.Parse(Channel))).SendMessageAsync(ParamaterisedMessage); } } public string MessageParser(string ParamaterisedMessage, StandardisedMessageRequest e, MessageType MessageType, StandardisedUser TargetUser = null, int Amount = -1, int NewBal = -1, string OtherString = "", string SenderUsername = null,string ServerID=null) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@CurrencyName>", BotInstance.CommandConfig["CurrencyName"].ToString()); ParamaterisedMessage = ParamaterisedMessage.Replace("<@ChannelName>", BotInstance.CommandConfig["ChannelName"].ToString()); ParamaterisedMessage = ParamaterisedMessage.Replace("<@Amount>", Amount.ToString("N0")); ParamaterisedMessage = ParamaterisedMessage.Replace("<@NewBalance>", NewBal.ToString("N0")); ParamaterisedMessage = ParamaterisedMessage.Replace("<@CurrencyAcronym>", BotInstance.CommandConfig["CurrencyAcronym"].ToString()); ParamaterisedMessage = ParamaterisedMessage.Replace("<@Prefix>", BotInstance.CommandConfig["Prefix"].ToString()); ParamaterisedMessage = ParamaterisedMessage.Replace("<@LatestTweet>", Data.APIIntergrations.Twitter.GetLatestTweet(BotInstance)); if (MessageType == MessageType.Twitch) { if (TargetUser != null) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@TargetUser>", "@" + TargetUser.UserName); } if (e!=null) if (e.SenderUserName != null) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@SenderUser>", "@" + e.SenderUserName); } else { ParamaterisedMessage = ParamaterisedMessage.Replace("<@SenderUser>", "@" + SenderUsername); } } else { if (ServerID != null) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@everyone>", "@everyone"); } if (TargetUser != null) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@TargetUser>", "<@" + TargetUser.ID + ">"); } ParamaterisedMessage = ParamaterisedMessage.Replace("/me", ""); if (e != null) ParamaterisedMessage = ParamaterisedMessage.Replace("<@SenderUser>", "<@" + e.SenderID + ">"); } ParamaterisedMessage = ParamaterisedMessage.Replace("<@everyone>", ""); ParamaterisedMessage = ParamaterisedMessage.Replace("<@OtherString>", OtherString); foreach (Newtonsoft.Json.Linq.JToken Emote in BotInstance.CommandConfig["Emotes"]) { if (MessageType == MessageType.Discord) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@" + Emote["Name"].ToString() + ">", Emote["Discord"].ToString()); } if (MessageType == MessageType.Twitch) { ParamaterisedMessage = ParamaterisedMessage.Replace("<@" + Emote["Name"].ToString() + ">", Emote["Twitch"].ToString()); } } return ParamaterisedMessage; } }}

Twitch-Discord-Reward-Bot/Backend/Bots/Commands/TimeEvents.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading;using System.Threading.Tasks;using Discord;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots.Commands{ public class TimeEvents { BotInstance BotInstance; Thread T; public void Start(BotInstance BotInstance) { this.BotInstance = BotInstance; T=new Thread(async () => await TimeThread()); T.Start(); } public void Stop() { if (T != null) { if (T.IsAlive) { T.Abort(); } } } async Task TimeThread() { while (true){ try { await Fish(); await AutoMessage(); RemoveDuels(); PerformRaffle(); RewardForViewing(); await LiveNotifications(); await CheckForDonations(); } catch (Exception E) { Console.WriteLine(E); } System.Threading.Thread.Sleep(10000); } } public DateTime LastAlert = DateTime.MinValue; public List<Alerter> AlertRequests = new List<Alerter> { }; void CullAlerts() { int AlertTimeout = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Alert"]["CoolDown"]["Individual"].ToString()); foreach (Alerter V in AlertRequests.Where(x => ((TimeSpan)(DateTime.Now - x.LastAlert)).TotalSeconds >= AlertTimeout)) { AlertRequests.Remove(V); } } public bool AlertTimeOutExpired(StandardisedUser U) { int GlobalAlertTimeout= int.Parse(BotInstance.CommandConfig["CommandSetup"]["Alert"]["CoolDown"]["Global"].ToString()); CullAlerts(); return (AlertRequests.Where(x => x.User.ID == U.ID).Count() == 0) && (((TimeSpan)(DateTime.Now-LastAlert)).TotalSeconds>=GlobalAlertTimeout); } public int GetRemainingCooldown(StandardisedUser U) { int GlobalAlertTimeout = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Alert"]["CoolDown"]["Global"].ToString()), GlobalTimeoutRemaining = GlobalAlertTimeout - (int)((TimeSpan)(DateTime.Now - LastAlert)).TotalSeconds, IndividualAlertTimeout = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Alert"]["CoolDown"]["Individual"].ToString()), IndividualTimeoutRemaining = 0; if (AlertRequests.Where(x => x.User.ID == U.ID).Count() != 0) { IndividualTimeoutRemaining = IndividualAlertTimeout - (int)((TimeSpan)(DateTime.Now - AlertRequests.Where(x => x.User.ID == U.ID).First().LastAlert)).TotalSeconds; } if (GlobalTimeoutRemaining > IndividualTimeoutRemaining) { return GlobalTimeoutRemaining; } else { return IndividualTimeoutRemaining; } } bool IsLive = false; async Task LiveNotifications() { if (BotInstance.CommandConfig["LiveNotifications"]["Enabled"].ToString() == "True") { bool NewIsLive = Data.APIIntergrations.Twitch.IsLive(BotInstance); if (IsLive != NewIsLive) { IsLive = NewIsLive; if (NewIsLive) { Newtonsoft.Json.Linq.JToken StreamLocal = Data.FileHandler.ReadJSON("./Data/Streams.json"); string StreamCurrent = Data.APIIntergrations.Twitch.GetStreamHelix(BotInstance)["data"][0]["id"].ToString(); if (BotInstance.CommandHandler.JArrayContainsString(StreamLocal, StreamCurrent)) { return; } else { List<String> StreamList = StreamLocal.ToObject<List<string>>(); StreamList.Add(StreamCurrent); Data.FileHandler.SaveJSON("./Data/Streams.json", Newtonsoft.Json.Linq.JToken.FromObject(StreamList)); } foreach (Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer Viewer in Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromCurrency(BotInstance).Where(x => x.LiveNotifcations)) { if (Viewer.DiscordID != "") { await BotInstance.DiscordBot.Client.GetUser(ulong.Parse(Viewer.DiscordID)).SendMessageAsync(BotInstance.CommandHandler.MessageParser(BotInstance.CommandConfig["LiveNotifications"]["Responses"]["LiveDM"].ToString(), null, MessageType.Discord)); } } if (BotInstance.CommandConfig["LiveNotifications"]["SendToDiscordNotificationChannel"].ToString() == "True") { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["LiveNotifications"]["Responses"]["LiveNotification"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord); } } } } } DateTime LastDonationCheck = DateTime.MinValue; async Task CheckForDonations() { if (((TimeSpan)(DateTime.Now - LastDonationCheck)).TotalSeconds < 60) { return; } if (!Data.APIIntergrations.Twitch.IsLive(BotInstance) && BotInstance.CommandConfig["AutoRewards"]["Donating"]["RequireLive"].ToString() == "True") { return; } LastDonationCheck = DateTime.Now; Newtonsoft.Json.Linq.JToken NetData = Data.APIIntergrations.Streamlabs.GetDonations(BotInstance), LocalData=Data.FileHandler.ReadJSON("./Data/DonationCache/"+BotInstance.Currency.ID+".json"); int DonationReward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["Donating"]["RewardPerWhole"].ToString()); if (LocalData != null) { if (NetData == null) { return; } if (NetData["data"][0]["donation\_id"].ToString() != LocalData["data"][0]["donation\_id"].ToString()) { for (int i = 0; i < LocalData["data"].Count(); i++) { if (LocalData["data"][0]["donation\_id"].ToString() != NetData["data"][i]["donation\_id"].ToString()) { Newtonsoft.Json.Linq.JToken Donation = NetData["data"][i]; await RewardDonator(Donation,DonationReward); } else { break; } } Data.FileHandler.SaveJSON("./Data/DonationCache/" + BotInstance.Currency.ID + ".json", NetData); } } else { foreach (Newtonsoft.Json.Linq.JToken Donation in NetData["data"]) { await RewardDonator(Donation,DonationReward); } Data.FileHandler.SaveJSON("./Data/DonationCache/" + BotInstance.Currency.ID + ".json",NetData); } } async Task RewardDonator(Newtonsoft.Json.Linq.JToken Donation,int DonationReward) { int DonationAmount = (int)Math.Round(double.Parse(Donation["amount"].ToString()), 2), AdjustedReward= (int)Math.Ceiling((double)DonationAmount \* DonationReward); StandardisedUser S = StandardisedUser.FromTwitchUsername(Donation["name"].ToString(), BotInstance); if (S != null) { Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer B = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(MessageType.Twitch, BotInstance, S.ID); Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(B, DonationAmount, "+"); await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["Donating"]["Response"].ToString(), BotInstance.CommandConfig["ChannelName"].ToString(), MessageType.Twitch, S, AdjustedReward,-1, DonationAmount+" "+Donation["currency"].ToString()); } } public List<Viewer> ViewerRewardTracking = new List<Viewer> { }; public DateTime LastViewerRewardCheck = DateTime.MinValue; void RewardForViewing() { if (((TimeSpan)(DateTime.Now - LastViewerRewardCheck)).TotalSeconds < 60) { return; } LastViewerRewardCheck = DateTime.Now; if (!Data.APIIntergrations.Twitch.IsLive(BotInstance)) { return; } Newtonsoft.Json.Linq.JToken JData = Data.APIIntergrations.Twitch.GetViewers(BotInstance); int Reward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["Viewing"]["RewardPerMinute"].ToString()); IEnumerable<Newtonsoft.Json.Linq.JToken> Merged = JData["chatters"]["vips"]. Union(JData["chatters"]["moderators"]). Union(JData["chatters"]["staff"]). Union(JData["chatters"]["admins"]). Union(JData["chatters"]["global\_mods"]). Union(JData["chatters"]["viewers"]); List<KeyValuePair<string, string>> Headers; Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("BalanceIncrement",Reward.ToString()), new KeyValuePair<string, string>("WatchTimeIncrement","1"), new KeyValuePair<string, string>("CurrencyID",this.BotInstance.Currency.ID.ToString()) }; JData = Newtonsoft.Json.Linq.JToken.Parse("{'TwitchIDs':[]}"); List<string> TwitchIDs = new List<string> { }; foreach (Newtonsoft.Json.Linq.JToken StreamViewer in Merged) { StandardisedUser U = StandardisedUser.FromTwitchUsername(StreamViewer.ToString(), BotInstance); if (U != null) { TwitchIDs.Add(U.ID); } } JData["TwitchIDs"] = Newtonsoft.Json.Linq.JToken.FromObject(TwitchIDs); Data.APIIntergrations.RewardCurrencyAPI.WebRequests.PostRequest("viewer", Headers, true,JData); } void RewardUser(int Reward,StandardisedUser U,MessageType MessageType) { Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer B = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(MessageType, BotInstance, U.ID); Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(B, Reward, "+"); } int RaffleNumber=0; public List<Raffler> RaffleParticipants = new List<Raffler> { }; DateTime LastRaffle = DateTime.MinValue; public bool UserRaffleing(StandardisedUser User) { return RaffleParticipants.Where(x=>x.User.ID==User.ID).Count()!=0; } void PerformRaffle() { if (BotInstance.CommandHandler.LiveCheck(BotInstance.CommandConfig["Raffle"])) { bool DoRaffle = false; int MinDelay = int.Parse(BotInstance.CommandConfig["Raffle"]["Triggers"]["Delay"].ToString()); if (BotInstance.CommandConfig["Raffle"]["Triggers"]["OnMinuteOfHour"].Count() != 0) { if (BotInstance.CommandHandler.JArrayContainsString(BotInstance.CommandConfig["Raffle"]["Triggers"]["OnMinuteOfHour"], DateTime.Now.Minute.ToString())) { DoRaffle = true; } } else if (((TimeSpan)(DateTime.Now - LastRaffle)).TotalSeconds >= MinDelay) { DoRaffle = true; } if (!RaffleRunning&&DoRaffle) { new Thread(async () => await RaffleThread()).Start(); } } } bool RaffleRunning = false; async Task RaffleThread() { RaffleRunning = true; int RaffleSize = 0; Newtonsoft.Json.Linq.JToken ChosenRaffle = null; foreach (Newtonsoft.Json.Linq.JToken RaffleType in BotInstance.CommandConfig["Raffle"]["Sizes"]) { RaffleSize += int.Parse(RaffleType["Frequency"].ToString()); if (RaffleSize > RaffleNumber && ChosenRaffle==null) { ChosenRaffle = RaffleType; } } int RaffleReward = int.Parse(ChosenRaffle["Size"].ToString()), CountDownMessages = int.Parse(BotInstance.CommandConfig["Raffle"]["CountDownMessages"].ToString()), MinuteSegmentSize=(int)Math.Ceiling((decimal)60/CountDownMessages); RaffleParticipants = new List<Raffler> { }; for (int i = 0; i < CountDownMessages; i++) { String TimeLeft = (CountDownMessages-i)\*MinuteSegmentSize+" seconds"; if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["Raffle"], MessageType.Twitch)) { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["LeadUp"].ToString(), BotInstance.CommandConfig["ChannelName"].ToString(), MessageType.Twitch,null, RaffleReward, -1,TimeLeft); } if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["Raffle"], MessageType.Discord)) { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["LeadUp"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, null, RaffleReward, -1, TimeLeft); } Thread.Sleep(MinuteSegmentSize\*1000); } if (RaffleParticipants.Count != 0) { int WinnerCount = int.Parse(ChosenRaffle["Winners"].ToString()); if (WinnerCount > RaffleParticipants.Count) { WinnerCount = RaffleParticipants.Count; } RaffleReward = (int)Math.Ceiling((decimal)RaffleReward / WinnerCount); for (int i=WinnerCount; WinnerCount > 0; WinnerCount--) { int WinnerN = Init.Rnd.Next(0, RaffleParticipants.Count); Raffler Winner = RaffleParticipants[WinnerN]; RaffleParticipants.RemoveAt(WinnerN); if (Winner.RequestedFrom == MessageType.Twitch) { if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["Raffle"], MessageType.Discord)) { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["OtherWinner"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, null, RaffleReward, -1, Winner.User.UserName); } await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["Winner"].ToString(), BotInstance.CommandConfig["ChannelName"].ToString(), MessageType.Twitch, Winner.User, RaffleReward); } else if (Winner.RequestedFrom == MessageType.Discord) { if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["Raffle"], MessageType.Twitch)) { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["OtherWinner"].ToString(), BotInstance.CommandConfig["ChannelName"].ToString(), MessageType.Twitch, null, RaffleReward, -1, Winner.User.UserName); } await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["Winner"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, Winner.User, RaffleReward); } Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer B = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(Winner.RequestedFrom, BotInstance, Winner.User.ID); Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(B, RaffleReward, "+"); } } else { if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["Raffle"], MessageType.Twitch)) { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["NoOne"].ToString(), BotInstance.CommandConfig["ChannelName"].ToString(), MessageType.Twitch,null, RaffleReward); } if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["Raffle"], MessageType.Discord)) { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["Raffle"]["Responses"]["NoOne"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, null, RaffleReward); } } RaffleNumber = (RaffleNumber + 1) % RaffleSize; RaffleRunning = false; LastRaffle = DateTime.Now; } public Dictionary<DateTime,Duel> Duels = new Dictionary<DateTime, Duel> { }; public void RemoveDuels() { int RemoveAfter = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Duel"]["CancelAfter"].ToString()); List<DateTime> KeysToRemove = new List<DateTime> { }; foreach (KeyValuePair<DateTime,Duel> Duel in Duels.Where(x => ((TimeSpan)(DateTime.Now - x.Key)).TotalSeconds > RemoveAfter)) { KeysToRemove.Add(Duel.Key); } foreach (DateTime Key in KeysToRemove) { Duels.Remove(Key); } } public bool UserDueling(StandardisedUser User) { return Duels.Where(x => x.Value.Creator.ID == User.ID || x.Value.Acceptor.ID == User.ID).Count() != 0; } Dictionary<int, DateTime> MessageHistory = new Dictionary<int, DateTime> { }; DateTime MessageLast = DateTime.MinValue; public async Task AutoMessage() { if (BotInstance.CommandHandler.LiveCheck(BotInstance.CommandConfig["AutoMessage"])) { if (BotInstance.TwitchBot.Client.IsConnected) { int MinDelay = int.Parse(BotInstance.CommandConfig["AutoMessage"]["MinimumDelay"].ToString()); if (((TimeSpan)(DateTime.Now - MessageLast)).TotalSeconds >= MinDelay) { Newtonsoft.Json.Linq.JToken Items = BotInstance.CommandConfig["AutoMessage"]["Messages"]; for (int i = 0; i < Items.Count(); i++) { bool ShouldSend = false; if (MessageHistory.ContainsKey(i)) { if (((TimeSpan)(DateTime.Now - MessageHistory[i])).TotalSeconds >= int.Parse(Items[i]["Delay"].ToString())) { ShouldSend = true; } } else { ShouldSend = true; } if (ShouldSend) { if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["AutoMessage"], MessageType.Twitch)) { await BotInstance.CommandHandler.SendMessage(Items[i]["Body"].ToString(), BotInstance.CommandConfig["ChannelName"].ToString(), MessageType.Twitch); } if (BotInstance.CommandHandler.CommandEnabled(BotInstance.CommandConfig["AutoMessage"], MessageType.Discord)) { await BotInstance.CommandHandler.SendMessage(Items[i]["Body"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord); } MessageLast = DateTime.Now; if (!MessageHistory.ContainsKey(i)) { MessageHistory.Add(i, DateTime.Now); } else { MessageHistory[i] = DateTime.Now; } return; } } } } } } public Dictionary<DateTime, Fisherman> Fishermen = new Dictionary<DateTime, Fisherman> { }; public async Task Fish() { List<DateTime> FishToRemove=new List<DateTime> { }; IEnumerable<KeyValuePair<DateTime, Fisherman>> FinishedFishermen = Fishermen.Where(x => ((TimeSpan)(DateTime.Now - x.Key)).TotalSeconds >= x.Value.SecondsToFish); foreach (KeyValuePair<DateTime,Fisherman> Fisher in FinishedFishermen) { Newtonsoft.Json.Linq.JToken Items = Fisher.Value.BotInstance.CommandConfig["CommandSetup"]["Fish"]["Items"], ChosenItem = Items[Init.Rnd.Next(0,Items.Count())]; int TotalChance = 0,ChosenChance=-1; foreach (Newtonsoft.Json.Linq.JToken Item in Items) { TotalChance += int.Parse(Item["Chance"].ToString()); } ChosenChance = Init.Rnd.Next(0, TotalChance); TotalChance = 0; foreach (Newtonsoft.Json.Linq.JToken Item in Items) { TotalChance += int.Parse(Item["Chance"].ToString()); if (TotalChance >= ChosenChance) { ChosenItem = Item; break; } } Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer B = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(Fisher.Value.e, Fisher.Value.BotInstance, Fisher.Value.e.SenderID); if (Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(B, int.Parse(ChosenItem["Reward"].ToString()), "+")) { await Fisher.Value.BotInstance.CommandHandler.SendMessage(Fisher.Value.BotInstance.CommandConfig["CommandSetup"]["Fish"]["Responses"]["Finished"].ToString(), Fisher.Value.e,null,int.Parse(ChosenItem["Reward"].ToString()),-1, ChosenItem["Name"].ToString()); FishToRemove.Add(Fisher.Key); } } foreach (DateTime FishKey in FishToRemove) { Fishermen.Remove(FishKey); } } } public class Viewer { public StandardisedUser User; public DateTime LastDiscordMessage = DateTime.MinValue, LastTwitchMessage = DateTime.MinValue; } public class Alerter { public StandardisedUser User; public DateTime LastAlert; } public class Raffler { public StandardisedUser User; public MessageType RequestedFrom; } public class Duel { public StandardisedMessageRequest e; public BotInstance BotInstance; public StandardisedUser Creator, Acceptor; public int ChangeBy; } public class Fisherman { public StandardisedMessageRequest e; public BotInstance BotInstance; public int SecondsToFish; public Fisherman(StandardisedMessageRequest e,BotInstance BotInstance) { this.e = e; this.BotInstance = BotInstance; int MinTime = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Fish"]["MinTime"].ToString()), MaxTime = int.Parse(BotInstance.CommandConfig["CommandSetup"]["Fish"]["MaxTime"].ToString()); SecondsToFish = Init.Rnd.Next(MinTime, MaxTime); } }}

Twitch-Discord-Reward-Bot/Backend/Bots/DiscordBot/Instance.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using Discord.WebSocket;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots.DiscordBot{ public class Instance : BaseObject { public Instance(BotInstance BotInstance) : base(BotInstance) { StartBot(); } public DiscordSocketClient Client; public async void StartBot() { DiscordSocketConfig SocketConfig = new DiscordSocketConfig(); SocketConfig.AlwaysDownloadUsers = true; Client = new DiscordSocketClient(SocketConfig); Client.MessageReceived += BotInstance.CommandHandler.Handle; await Client.LoginAsync(Discord.TokenType.Bot, BotInstance.LoginConfig["Discord"]["Bot"]["AuthToken"].ToString()); await Client.StartAsync(); await Client.SetGameAsync(BotInstance.CommandConfig["Discord"]["StatusMessage"].ToString()); Console.WriteLine("Started DiscordBot for Currency: " + BotInstance.Currency.ID); } }}

Twitch-Discord-Reward-Bot/Backend/Bots/TwitchBot/Events.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using TwitchLib.Client;using TwitchLib.Client.Models;using TwitchLib.Client.Events;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots.TwitchBot{ public class Events : BaseObject { public Events(BotInstance BotInstance) : base(BotInstance) { } public async void SubGifted(object sender, OnGiftedSubscriptionArgs e) { StandardisedUser Gifter = new StandardisedUser(), Giftee = new StandardisedUser(); ; Gifter.ID = e.GiftedSubscription.Id; Gifter.UserName = e.GiftedSubscription.DisplayName; Giftee.ID = e.GiftedSubscription.MsgParamRecipientId; Giftee.UserName = e.GiftedSubscription.MsgParamRecipientDisplayName; int Reward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["GiftSub"]["Reward"].ToString()); Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer V = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(MessageType.Twitch, BotInstance, Gifter.ID); if (V != null) { Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(V, Reward, "+"); } await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["GiftSub"]["Response"].ToString(), e.Channel.ToString(),MessageType.Twitch,Gifter,Reward,OtherString:"@"+Giftee.UserName); if (BotInstance.CommandConfig["AutoRewards"]["DiscordSubNotifications"].ToString() == "True") { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["GiftSub"]["Response"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, Gifter, Reward, OtherString: "<@" + Giftee.UserName +">"); } } public async void Subbed(object sender, OnNewSubscriberArgs e) { StandardisedUser Subber = new StandardisedUser(); Subber.ID = e.Subscriber.UserId; Subber.UserName = e.Subscriber.DisplayName; int Reward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["NewSub"]["Reward"].ToString()); Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer V = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(MessageType.Twitch, BotInstance, Subber.ID); if (V != null) { Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(V, Reward, "+"); } await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["NewSub"]["Response"].ToString(), e.Channel.ToString(), MessageType.Twitch, Subber, Reward); if (BotInstance.CommandConfig["AutoRewards"]["DiscordSubNotifications"].ToString() == "True") { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["NewSub"]["Response"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, Subber, Reward); } } public async void ReSubbed(object sender, OnReSubscriberArgs e) { StandardisedUser Subber = new StandardisedUser(); Subber.ID = e.ReSubscriber.UserId; Subber.UserName = e.ReSubscriber.DisplayName; int Reward = int.Parse(BotInstance.CommandConfig["AutoRewards"]["ReSub"]["Reward"].ToString()); Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer V = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(MessageType.Twitch, BotInstance, Subber.ID); if (V != null) { Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.AdjustBalance(V, Reward, "+"); } await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["ReSub"]["Response"].ToString(), e.Channel.ToString(), MessageType.Twitch, Subber, Reward); if (BotInstance.CommandConfig["AutoRewards"]["DiscordSubNotifications"].ToString() == "True") { await BotInstance.CommandHandler.SendMessage(BotInstance.CommandConfig["AutoRewards"]["ReSub"]["Response"].ToString(), BotInstance.CommandConfig["Discord"]["NotificationChannel"].ToString(), MessageType.Discord, Subber, Reward); } } }}

Twitch-Discord-Reward-Bot/Backend/Bots/TwitchBot/Instance.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using TwitchLib.Client;using TwitchLib.Client.Models;//using TwitchLib.Communication.Events;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots.TwitchBot{ public class Instance:BaseObject { public Events Events; public Instance(BotInstance BotInstance):base(BotInstance) { Events = new Events(BotInstance); StartBot(); } public TwitchClient Client; void StartBot() { ConnectionCredentials BotDetails = new ConnectionCredentials( BotInstance.LoginConfig["Twitch"]["Bot"]["Username"].ToString(), BotInstance.LoginConfig["Twitch"]["Bot"]["AuthToken"].ToString() ); Client = new TwitchClient(); Client.Initialize(BotDetails,BotInstance.CommandConfig["ChannelName"].ToString()); Client.OnMessageReceived += BotInstance.CommandHandler.Handle; Client.OnNewSubscriber += Events.Subbed; Client.OnReSubscriber += Events.ReSubbed; Client.OnGiftedSubscription += Events.SubGifted; //Client.OnDisconnected += BotDisconnected; Client.OnConnectionError += BotConnectionError; //Client.OnError += BotError; Client.Connect(); Console.WriteLine("Started Twitch Bot for Currency: " + BotInstance.Currency.ID); } //public void BotDisconnected(object Sender, OnDisconnectedEventArgs e) //{ // if (BotInstance.Isrunning) // { // StartBot(); // } //} public void BotConnectionError(object Sender, TwitchLib.Client.Events.OnConnectionErrorArgs e) { Console.WriteLine(e); if (BotInstance.Isrunning) { StartBot(); } } // public void BotError(object Sender, OnErrorEventArgs e) // { // Console.WriteLine(e); // if (BotInstance.Isrunning) // { // StartBot(); // } // } }}

Twitch-Discord-Reward-Bot/Backend/Bots/BaseObject.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots{ public class BaseObject { protected BotInstance BotInstance; public BaseObject(BotInstance BotInstance) { this.BotInstance = BotInstance; } }}

Twitch-Discord-Reward-Bot/Backend/Bots/Objects.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using TwitchLib.Client.Events;using Discord.WebSocket;using System.Net;using System.IO;namespace Twitch\_Discord\_Reward\_Bot.Backend.Bots{ public class StandardisedMessageRequest { public string MessageBody,ChannelID,ChannelName,SenderID,SenderUserName; public string[] SegmentedBody; public MessageType MessageType; public OnMessageReceivedArgs TwitchRaw; public SocketMessage DiscordRaw; public bool IsNewUser; public StandardisedUser User; public Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer Viewer; public static StandardisedMessageRequest FromTwitch(OnMessageReceivedArgs e,BotInstance BotInstance) { StandardisedMessageRequest S = new StandardisedMessageRequest(); S.MessageBody = e.ChatMessage.Message; S.SegmentedBody = S.MessageBody.Split(" ".ToCharArray()); S.MessageType = MessageType.Twitch; S.SenderID = e.ChatMessage.UserId; S.SenderUserName = e.ChatMessage.Username; S.TwitchRaw = e; S.ChannelName = e.ChatMessage.Channel; S.User = new StandardisedUser(); S.User.ID = S.SenderID; S.User.UserName = S.SenderUserName; S.Viewer = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(S,BotInstance,S.User.ID,ref S.IsNewUser); return S; } public static StandardisedMessageRequest FromDiscord(SocketMessage e, BotInstance BotInstance) { StandardisedMessageRequest S = new StandardisedMessageRequest(); S.MessageBody = e.Content; S.SegmentedBody = S.MessageBody.Split(" ".ToCharArray()); S.MessageType = MessageType.Discord; S.SenderID = e.Author.Id.ToString(); S.SenderUserName = e.Author.Username; S.DiscordRaw = e; S.ChannelID = e.Channel.Id.ToString(); S.ChannelName = e.Channel.Name; S.User = new StandardisedUser(); S.User.ID = S.SenderID; S.User.UserName = S.SenderUserName; S.Viewer = Data.APIIntergrations.RewardCurrencyAPI.Objects.Viewer.FromTwitchDiscord(S, BotInstance, S.User.ID, ref S.IsNewUser); return S; } } public class StandardisedUser { public string UserName; public string ID; public static StandardisedUser FromTwitchUsername(string MessageSegment, BotInstance BotInstance,int Depth=0) { if (Depth == 5) { return null; } string UserName = MessageSegment.Replace("@", ""); try { WebRequest Req = WebRequest.Create("https://api.twitch.tv/helix/users?login=" + UserName); Req.Method = "GET"; Req.Headers.Add("Authorization", BotInstance.LoginConfig["Twitch"]["API"]["AuthToken"].ToString()); WebResponse Res = Req.GetResponse(); string StreamString = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JToken JData = Newtonsoft.Json.Linq.JToken.Parse(StreamString); StandardisedUser U = new StandardisedUser(); U.ID = JData["data"][0]["id"].ToString(); U.UserName = UserName; return U; } catch { return null; FromTwitchUsername(MessageSegment, BotInstance, Depth+1); } return null; } public static StandardisedUser FromDiscordMention(string MessageSegment, BotInstance BotInstance) { StandardisedUser U = new StandardisedUser(); U.ID = MessageSegment.Replace("<@", "").Replace(">", "").Replace("!",""); return U; } } public enum MessageType { Discord,Twitch }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/Objects/BaseObject.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI.Objects{ public class BaseObject // The baseobject is inherited by the other objects, so we can have generic functions be shared, reducing code repeats. { public int ID; public Newtonsoft.Json.Linq.JToken ToJson() { return Newtonsoft.Json.Linq.JToken.FromObject(this); } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/Objects/Bot.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI.Objects{ public class Bot : BaseObject { public string InviteCode; public Currency Currency; public string AccessToken, RefreshToken; public DateTime TokenRefreshDateTime; public Login OwnerLogin; public static Bot FromJson(Newtonsoft.Json.Linq.JToken Json) { return Json.ToObject<Bot>(); } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/Objects/Currency.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI.Objects{ public class Currency : BaseObject { public Login OwnerLogin; public Newtonsoft.Json.Linq.JToken LoginConfig, CommandConfig; public static Currency FromJson(Newtonsoft.Json.Linq.JToken Json) { return Json.ToObject<Currency>(); } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/Objects/Login.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI.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>(); } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/Objects/User.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\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI.Objects{ public class User : BaseObject { public uint UserId; public string TwitchId, DiscordId; public Account Account; }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/Objects/Viewer.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Net;using System.IO;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI.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 FromTwitchDiscord(Bots.StandardisedMessageRequest e, BotInstance BotInstance,string ID) { return FromTwitchDiscord(e.MessageType, BotInstance, ID); } public static Viewer FromTwitchDiscord(Bots.StandardisedMessageRequest e, BotInstance BotInstance, string ID,ref bool CreatedViewer) { return FromTwitchDiscord(e.MessageType, BotInstance, ID,ref CreatedViewer); } public static Viewer FromTwitchDiscord(Bots.MessageType e, BotInstance BotInstance, string ID) { bool Temp=false; return FromTwitchDiscord(e, BotInstance, ID, ref Temp); } public static Viewer FromTwitchDiscord(Bots.MessageType e, BotInstance BotInstance, string ID, ref bool CreatedViewer) { List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("CurrencyID", BotInstance.Currency.ID.ToString()) }; if (e == Bots.MessageType.Twitch) { Headers.Add(new KeyValuePair<string, string>("TwitchID", ID)); } if (e == Bots.MessageType.Discord) { Headers.Add(new KeyValuePair<string, string>("DiscordID", ID)); } ResponseObject RObj = WebRequests.PostRequest("viewer", Headers, true); CreatedViewer = RObj.Code == 200; RObj = WebRequests.GetRequest("viewer", Headers); if (RObj.Code == 200) { Viewer B = FromJson(RObj.Data); return B; } return null; } public static List<Viewer> FromCurrency(BotInstance BotInstance) { List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("CurrencyID", BotInstance.Currency.ID.ToString()) }; ResponseObject RObj = WebRequests.GetRequest("viewer", Headers); if (RObj.Code == 200) { List<Viewer> B = new List<Viewer> { }; foreach (Newtonsoft.Json.Linq.JToken Item in RObj.Data) { B.Add(Item.ToObject<Viewer>()); } return B; } return null; } public static bool AdjustBalance(Viewer Bank, int Value, string Operator) { List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("ID", Bank.ID.ToString()), new KeyValuePair<string, string>("Value",Value.ToString()), new KeyValuePair<string, string>("Operator",Operator) }; ResponseObject RObj = WebRequests.PostRequest("viewer", Headers, true); if (Operator == "+") { Bank.Balance += Value; } else if (Operator == "-") { Bank.Balance -= Value; } if (RObj.Code == 200) { return true; } return false; } public static bool MergeAccounts(Bots.StandardisedMessageRequest e,BotInstance BotInstance,string ID) { if (BotInstance.CommandConfig["Discord"]["TwitchMerging"].ToString().ToLower() == "true") { if (e.MessageType == Bots.MessageType.Discord) { if (e.Viewer.TwitchID != "") { return false; } try { WebRequest Req = WebRequest.Create("https://discordapp.com/api/v6/users/" + ID + "/profile"); Req.Headers.Add("authorization", Init.MasterConfig["Discord"]["User"]["AuthToken"].ToString()); Req.Method = "GET"; WebResponse Res = Req.GetResponse(); string D = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JObject ProfileData = Newtonsoft.Json.Linq.JObject.Parse(D); foreach (Newtonsoft.Json.Linq.JObject Connection in ProfileData["connected\_accounts"]) { if (Connection["type"].ToString() == "twitch") { Viewer Twitch = FromTwitchDiscord(Bots.MessageType.Twitch, BotInstance, Connection["id"].ToString()); Viewer Discord = e.Viewer; if (Twitch.DiscordID == "" && Discord.TwitchID == "") { AdjustBalance(Twitch, Twitch.Balance, "-"); AdjustBalance(Discord, Twitch.Balance, "+"); List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("TwitchID", Connection["id"].ToString()), new KeyValuePair<string, string>("DiscordID",ID), new KeyValuePair<string, string>("ID",Discord.ID.ToString()) }; ResponseObject RObj = WebRequests.PostRequest("viewer", Headers, true); Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("ID", Twitch.ID.ToString()) }; RObj = WebRequests.PostRequest("viewer", Headers, true); } } } } catch (WebException E) { } } } return false; } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/ResponseObject.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI{ public class ResponseObject:Objects.BaseObject { public Newtonsoft.Json.Linq.JToken Data; public int Code; public string Message; public static ResponseObject FromJson(Newtonsoft.Json.Linq.JToken Json) { return Json.ToObject<ResponseObject>(); } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/RewardCurrencyAPI/WebRequests.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Net;using System.IO;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations.RewardCurrencyAPI{ public static class WebRequests { public static ResponseObject GetRequest(string URL, List<KeyValuePair<string, string>> Headers = null) { string URLStart = Init.MasterConfig["API"]["WebAddress"] + ":" + Init.MasterConfig["API"]["Port"] + "/"; if (Init.MasterConfig["API"]["AddressPath"].ToString() != "") { URLStart += Init.MasterConfig["API"]["AddressPath"] + "/"; } WebRequest Req = WebRequest.Create( URLStart + URL); Req.Method = "GET"; if (Headers != null) { foreach (KeyValuePair<string, string> HeaderPair in Headers) { Req.Headers.Add(HeaderPair.Key, HeaderPair.Value); } } try { WebResponse Res = Req.GetResponse(); string StreamString = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JToken JData = Newtonsoft.Json.Linq.JToken.Parse(StreamString); ResponseObject RObj = ResponseObject.FromJson(JData); return RObj; } catch (WebException E) { Console.WriteLine(E); return null; } } public static ResponseObject PostRequest(string URL, List<KeyValuePair<string, string>> Headers = null,bool Auth=false,Newtonsoft.Json.Linq.JToken Data = null) { string URLStart = Init.MasterConfig["API"]["WebAddress"] + ":" + Init.MasterConfig["API"]["Port"] + "/"; if (Init.MasterConfig["API"]["AddressPath"].ToString() != "") { URLStart += Init.MasterConfig["API"]["AddressPath"] + "/"; } WebRequest Req = WebRequest.Create( URLStart + URL); Req.Method = "POST"; if (Headers != null) { foreach (KeyValuePair<string, string> HeaderPair in Headers) { Req.Headers.Add(HeaderPair.Key, HeaderPair.Value); } } if (Auth) { Req.Headers.Add("AuthToken",GetAuthToken()); Req.Headers.Add("BotID", Init.MasterConfig["API"]["ID"].ToString()); } Byte[] PostData = new byte[] { }; if (Data != null) { PostData = Encoding.UTF8.GetBytes(Data.ToString()); } Req.ContentLength = PostData.Length; Stream PostStream = Req.GetRequestStream(); PostStream.Write(PostData, 0, PostData.Length); PostStream.Flush(); PostStream.Close(); try { WebResponse Res = Req.GetResponse(); string StreamString = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JToken JData = Newtonsoft.Json.Linq.JToken.Parse(StreamString); ResponseObject RObj = ResponseObject.FromJson(JData); return RObj; } catch (WebException E) { Console.WriteLine(E); return null; } } static string LastAuthToken = null; static DateTime LastRefreshed = DateTime.MinValue; public static string GetAuthToken() { if (((TimeSpan)(DateTime.Now - LastRefreshed)).TotalSeconds > 600) { string URLStart = Init.MasterConfig["API"]["WebAddress"] + ":" + Init.MasterConfig["API"]["Port"] + "/"; if (Init.MasterConfig["API"]["AddressPath"].ToString() != "") { URLStart += Init.MasterConfig["API"]["AddressPath"] + "/"; } WebRequest Req = WebRequest.Create(URLStart + "bot"); Req.Headers.Add("RefreshToken", Init.MasterConfig["API"]["RefreshToken"].ToString()); Req.Headers.Add("BotID", Init.MasterConfig["API"]["ID"].ToString()); Req.Method = "POST"; Stream PostStream = Req.GetRequestStream(); PostStream.Write(new byte[] { }, 0, new byte[] { }.Length); PostStream.Flush(); PostStream.Close(); try { WebResponse Res = Req.GetResponse(); string StreamString = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JToken JData = Newtonsoft.Json.Linq.JToken.Parse(StreamString); ResponseObject RObj = ResponseObject.FromJson(JData); if (RObj.Data.Count()!=0) { Objects.Bot B = Objects.Bot.FromJson(RObj.Data); Init.MasterConfig["API"]["RefreshToken"] = B.RefreshToken; LastRefreshed = B.TokenRefreshDateTime; LastAuthToken = B.AccessToken; FileHandler.SaveJSON("./Data/Master.config.json", Init.MasterConfig); } else { return null; } } catch (WebException E) { Console.WriteLine(E); return null; } } return LastAuthToken; } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/Nightbot.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.IO;using System.Net;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations{ public static class Nightbot { public static AccessToken GetAuthToken(BotInstance BotInstance) { if (BotInstance.AccessTokens.ContainsKey("Nightbot")) { if (((TimeSpan)(BotInstance.AccessTokens["Nightbot"].ExpiresAt - DateTime.Now)).TotalMinutes > 1) { return BotInstance.AccessTokens["Nightbot"]; } } WebRequest Req = WebRequest.Create("https://api.nightbot.tv/oauth2/token"); byte[] PostData = Encoding.UTF8.GetBytes("client\_id=" + BotInstance.LoginConfig["NightBot"]["ClientId"] + "&client\_secret=" + BotInstance.LoginConfig["NightBot"]["ClientSecret"] + "&grant\_type=refresh\_token&redirect\_uri=" + Init.MasterConfig["Redirect"]["WebAddress"] + "/" + Init.MasterConfig["Redirect"]["AddressPath"] + "/nightbot/" + "&refresh\_token=" + BotInstance.LoginConfig["NightBot"]["RefreshToken"]); 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); BotInstance.LoginConfig["NightBot"]["RefreshToken"] = JD["refresh\_token"]; List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("CurrencyID", BotInstance.Currency.ID.ToString()) }; var R = RewardCurrencyAPI.WebRequests.PostRequest("currency", Headers, true, Newtonsoft.Json.Linq.JToken.Parse("{'LoginConfig':" + BotInstance.LoginConfig.ToString() + @"}")); AccessToken Tk = new AccessToken(JD["access\_token"].ToString(), int.Parse(JD["expires\_in"].ToString())); if (BotInstance.AccessTokens.ContainsKey("Nightbot")) { BotInstance.AccessTokens["Nightbot"] = Tk; } else { BotInstance.AccessTokens.Add("Nightbot", Tk); } return BotInstance.AccessTokens["Nightbot"]; } catch (WebException E) { Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); return null; } } public static Newtonsoft.Json.Linq.JToken GenericExecute(BotInstance BotInstance, string URL, string Method) { return GenericExecute(BotInstance, URL, "", Method); } public static Newtonsoft.Json.Linq.JToken GenericExecute(BotInstance BotInstance, string URL, string Data, string Method) { WebRequest Req = WebRequest.Create(URL); Req.Method = Method; Req.Headers.Add("Authorization", "Bearer " + GetAuthToken(BotInstance).Token); Req.ContentType = "application/x-www-form-urlencoded"; if (Data != "") { byte[] PostData = Encoding.UTF8.GetBytes(Data); 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); return JD; } catch (WebException E) { return Newtonsoft.Json.Linq.JToken.Parse(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); } } static int PrevVolume = 10; public static Newtonsoft.Json.Linq.JToken GetQueue(BotInstance BotInstance) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests/queue", "GET"); } public static Newtonsoft.Json.Linq.JToken PauseSong(BotInstance BotInstance) { PrevVolume = int.Parse(GetQueue(BotInstance)["settings"]["volume"].ToString()); return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests", "volume=0", "PUT"); } public static Newtonsoft.Json.Linq.JToken PlaySong(BotInstance BotInstance) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests", "volume=" + PrevVolume, "PUT"); } public static Newtonsoft.Json.Linq.JToken SkipSong(BotInstance BotInstance) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests/queue/skip", "POST"); } public static Newtonsoft.Json.Linq.JToken SetVolume(BotInstance BotInstance, int Volume) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests", "volume=" + Volume, "PUT"); } public static Newtonsoft.Json.Linq.JToken RequestSong(BotInstance BotInstance, string Url) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests/queue", "q=" + Url, "POST"); } public static Newtonsoft.Json.Linq.JToken RemoveItem(BotInstance BotInstance, int i) { Newtonsoft.Json.Linq.JToken Song = GetSongFromPos(BotInstance, i); return RemoveID(BotInstance, Song["\_id"].ToString()); } public static Newtonsoft.Json.Linq.JToken RemoveID(BotInstance BotInstance, string ID) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests/queue/" + ID, "DELETE"); } public static Newtonsoft.Json.Linq.JToken PromoteItem(BotInstance BotInstance, int i) { Newtonsoft.Json.Linq.JToken Song = GetSongFromPos(BotInstance, i); return RemoveID(BotInstance, Song["\_id"].ToString()); } public static Newtonsoft.Json.Linq.JToken PromoteID(BotInstance BotInstance, string ID) { return GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests/queue/" + ID + "/promote", "POST"); } public static Newtonsoft.Json.Linq.JToken GetSongFromPos(BotInstance BotInstance, int i) { Newtonsoft.Json.Linq.JToken CurrentQueue = GenericExecute(BotInstance, "https://api.nightbot.tv/1/song\_requests/queue", "GET"); if (CurrentQueue["status"].ToString() != "200") { return Newtonsoft.Json.Linq.JToken.Parse("{\"message\":\"Error occured\",\"status\":400}"); } if (CurrentQueue["queue"].Count() < i || i <= 0) { return Newtonsoft.Json.Linq.JToken.Parse("{\"message\":\"Out of range!\",\"status\":400}"); } return CurrentQueue["queue"][i - 1]; } }}

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

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations{ public class AccessToken { public string Token; public DateTime ExpiresAt; public AccessToken(string Token,int ExpiresIn) { ExpiresAt = DateTime.Now; ExpiresAt = ExpiresAt.AddSeconds((double)ExpiresIn); this.Token = Token; } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/Streamlabs.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.IO;using System.Net;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations{ public static class Streamlabs { public static AccessToken GetAuthToken(BotInstance BotInstance) { if (BotInstance.AccessTokens.ContainsKey("Streamlabs")) { if (((TimeSpan)(BotInstance.AccessTokens["Streamlabs"].ExpiresAt - DateTime.Now)).TotalMinutes > 1) { return BotInstance.AccessTokens["Streamlabs"]; } } WebRequest Req = WebRequest.Create("https://streamlabs.com/api/v1.0/token"); byte[] PostData = Encoding.UTF8.GetBytes("client\_id=" + BotInstance.LoginConfig["StreamLabs"]["ClientId"] + "&client\_secret=" + BotInstance.LoginConfig["StreamLabs"]["ClientSecret"] + "&grant\_type=refresh\_token&redirect\_uri=" + Init.MasterConfig["Redirect"]["WebAddress"] + "/" + Init.MasterConfig["Redirect"]["AddressPath"] + "/streamlabs/" + "&refresh\_token=" + BotInstance.LoginConfig["StreamLabs"]["RefreshToken"]); 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); BotInstance.LoginConfig["StreamLabs"]["RefreshToken"] = JD["refresh\_token"]; List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("CurrencyID", BotInstance.Currency.ID.ToString()) }; var R = RewardCurrencyAPI.WebRequests.PostRequest("currency", Headers, true, Newtonsoft.Json.Linq.JToken.Parse("{'LoginConfig':" + BotInstance.LoginConfig.ToString() + @"}")); AccessToken Tk = new AccessToken(JD["access\_token"].ToString(), int.Parse(JD["expires\_in"].ToString())); if (BotInstance.AccessTokens.ContainsKey("Streamlabs")) { BotInstance.AccessTokens["Streamlabs"] = Tk; } else { BotInstance.AccessTokens.Add("Streamlabs", Tk); } return BotInstance.AccessTokens["Streamlabs"]; } catch (WebException E) { Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); return null; } } public static Newtonsoft.Json.Linq.JToken GenericExecute(BotInstance BotInstance, string URL, string Data="", string Method="GET",bool URLAuth=false) { if (URLAuth) { if (URL.Contains("?")) { URL += "&access\_token=" + GetAuthToken(BotInstance).Token; } else { URL += "?access\_token=" + GetAuthToken(BotInstance).Token; } } WebRequest Req = WebRequest.Create(URL); Req.Method = Method; Req.ContentType = "application/x-www-form-urlencoded"; //Req.Timeout = 2000; if (Data != "") { byte[] PostData = Encoding.UTF8.GetBytes(Data); 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); return JD; } catch (WebException E) { return Newtonsoft.Json.Linq.JToken.Parse(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); } } public static Newtonsoft.Json.Linq.JToken GetDonations(BotInstance BotInstance) { return GenericExecute(BotInstance, "https://streamlabs.com/api/v1.0/donations?limit=100",URLAuth:true); } public static Newtonsoft.Json.Linq.JToken PlayAlert(BotInstance BotInstance, string SoundURL) { return GenericExecute(BotInstance, "https://streamlabs.com/api/v1.0/alerts", "access\_token="+ GetAuthToken(BotInstance).Token + "&type=donation&message= &user\_message=This Is An Alert Not A Donation&image\_href=https://upload.wikimedia.org/wikipedia/commons/thumb/0/0b/TransparentPlaceholder.svg/240px-TransparentPlaceholder.svg.png&sound\_href=" + SoundURL, Method:"POST"); } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/Twitch.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Net;using System.IO;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations{ public static class Twitch { static bool Islive = false; static DateTime LastLiveCheck; public static bool IsLive(BotInstance BotInstance) { if (((TimeSpan)(DateTime.Now - LastLiveCheck)).TotalSeconds < 15) { return Islive; } try { Newtonsoft.Json.Linq.JToken JD = GetStreamHelix(BotInstance); if (JD["data"].Count() != 0) { if (JD["data"][0]["type"].ToString() == "live") { Islive = true; return true; } } LastLiveCheck = DateTime.Now; Islive = false; return false; } catch (Exception E) { Console.WriteLine(E); return false; } } public static AccessToken GetAuthToken(BotInstance BotInstance) { if (BotInstance.AccessTokens.ContainsKey("Twitch")) { if (((TimeSpan)(BotInstance.AccessTokens["Twitch"].ExpiresAt - DateTime.Now)).TotalMinutes > 1) { return BotInstance.AccessTokens["Twitch"]; } } WebRequest Req = WebRequest.Create("https://id.twitch.tv/oauth2/token"); byte[] PostData = Encoding.UTF8.GetBytes("client\_id=" + BotInstance.LoginConfig["Twitch"]["API"]["ClientId"] + "&client\_secret=" + BotInstance.LoginConfig["Twitch"]["API"]["ClientSecret"] + "&grant\_type=refresh\_token&redirect\_uri=" + Init.MasterConfig["Redirect"]["WebAddress"] + "/" + Init.MasterConfig["Redirect"]["AddressPath"] + "/twitch/" + "&refresh\_token=" + BotInstance.LoginConfig["Twitch"]["API"]["RefreshToken"]); 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); BotInstance.LoginConfig["Twitch"]["API"]["RefreshToken"] = JD["refresh\_token"]; List<KeyValuePair<string, string>> Headers = new List<KeyValuePair<string, string>> { new KeyValuePair<string, string>("CurrencyID", BotInstance.Currency.ID.ToString()) }; var R = RewardCurrencyAPI.WebRequests.PostRequest("currency", Headers, true, Newtonsoft.Json.Linq.JToken.Parse("{'LoginConfig':" + BotInstance.LoginConfig.ToString() + @"}")); AccessToken Tk = new AccessToken(JD["access\_token"].ToString(), int.Parse(JD["expires\_in"].ToString())); if (BotInstance.AccessTokens.ContainsKey("Twitch")) { BotInstance.AccessTokens["Twitch"] = Tk; } else { BotInstance.AccessTokens.Add("Twitch", Tk); } return BotInstance.AccessTokens["Twitch"]; } catch (WebException E) { Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); return null; } } public static Newtonsoft.Json.Linq.JToken Request(BotInstance BotInstance,string URL,string Method="GET", string PostData=null) { try { WebRequest Req = WebRequest.Create(URL); Req.Method = Method; Req.Headers.Add("Client-ID", BotInstance.LoginConfig["Twitch"]["API"]["ClientId"].ToString()); Req.Headers.Add("Authorization", "OAuth " + GetAuthToken(BotInstance).Token); if (PostData != null) { Byte[] BytePostData = Encoding.UTF8.GetBytes(PostData); Req.ContentLength = BytePostData.Length; Req.ContentType = "application/x-www-form-urlencoded"; Stream PostStream = Req.GetRequestStream(); PostStream.Write(BytePostData, 0, BytePostData.Length); PostStream.Flush(); PostStream.Close(); } WebResponse Res = Req.GetResponse(); string D = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JObject JD = Newtonsoft.Json.Linq.JObject.Parse(D); return JD; } catch (WebException E) { Console.WriteLine(new StreamReader(E.Response.GetResponseStream()).ReadToEnd()); return null; } } public static Newtonsoft.Json.Linq.JToken GetStreamHelix(BotInstance BotInstance) { return Request(BotInstance, "https://api.twitch.tv/helix/streams?user\_login=" + BotInstance.CommandConfig["ChannelName"]); } public static Newtonsoft.Json.Linq.JToken GetChannel(BotInstance BotInstance) { return Request(BotInstance, "https://api.twitch.tv/kraken/channels/" + BotInstance.CommandConfig["ChannelName"].ToString()); } public static Newtonsoft.Json.Linq.JToken UpdateChannelTitle(BotInstance BotInstance,string NewTitle) { return Request(BotInstance, "https://api.twitch.tv/kraken/channels/" + BotInstance.CommandConfig["ChannelName"].ToString(), "PUT", "channel[status]="+NewTitle.Replace(" ","+")); } public static Newtonsoft.Json.Linq.JToken UpdateChannelGame(BotInstance BotInstance, string NewGame) { return Request(BotInstance, "https://api.twitch.tv/kraken/channels/" + BotInstance.CommandConfig["ChannelName"].ToString(), "PUT", "channel[game]=" + NewGame.Replace(" ", "+")); } public static Newtonsoft.Json.Linq.JToken GetStream(BotInstance BotInstance) { return Request(BotInstance, "https://api.twitch.tv/kraken/streams/" + BotInstance.CommandConfig["ChannelName"].ToString().ToString()); } public static Newtonsoft.Json.Linq.JToken GetViewers(BotInstance BotInstance) { WebRequest Req = WebRequest.Create("https://tmi.twitch.tv/group/user/" +BotInstance.CommandConfig["ChannelName"]+"/chatters"); WebResponse Res = Req.GetResponse(); string D = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JObject JD = Newtonsoft.Json.Linq.JObject.Parse(D); return JD; } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/Twitter.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using Tweetinvi;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations{ public static class Twitter { public static string GetLatestTweet(BotInstance BotInstance) { try { Auth.SetUserCredentials( BotInstance.LoginConfig["Twitter"]["ConsumerKey"].ToString(), BotInstance.LoginConfig["Twitter"]["ConsumerSecret"].ToString(), BotInstance.LoginConfig["Twitter"]["AccessToken"].ToString(), BotInstance.LoginConfig["Twitter"]["AccessSecret"].ToString() ); Tweetinvi.Models.IUser TwitterUser = User.GetUserFromScreenName("TheHarbonator"); Tweetinvi.Models.ITweet UsersLatestTweet = TwitterUser.GetUserTimeline(1).Last(); return UsersLatestTweet.Url; } catch { return "!Was Unable To Find URL!"; } } }}

Twitch-Discord-Reward-Bot/Backend/Data/APIIntergrations/Youtube.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;using System.Net;using System.IO;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data.APIIntergrations{ public static class Youtube { public static string LatestVid(BotInstance BotInstance) { WebRequest Req = WebRequest.Create("https://www.googleapis.com/youtube/v3/search?key=" + BotInstance.LoginConfig["Youtube"]["AuthToken"].ToString() + "&channelId=" + BotInstance.LoginConfig["Youtube"]["ChannelID"].ToString() + "&part=snippet,id&order=date&maxResults=1"); Req.Method = "GET"; try { WebResponse Res = Req.GetResponse(); string SData = new StreamReader(Res.GetResponseStream()).ReadToEnd(); Newtonsoft.Json.Linq.JToken Resp = Newtonsoft.Json.Linq.JToken.Parse(SData); return "https://youtu.be/" + Resp["items"][0]["id"]["videoId"].ToString(); } catch { return null; } } }}

Twitch-Discord-Reward-Bot/Backend/Data/FileHandler.cs

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;namespace Twitch\_Discord\_Reward\_Bot.Backend.Data{ public static class FileHandler { public static Newtonsoft.Json.Linq.JToken ReadJSON(string FilePath) { try { return Newtonsoft.Json.Linq.JToken.Parse(System.IO.File.ReadAllText(FilePath)); } catch { return null; } } public static void SaveJSON(string FilePath,Newtonsoft.Json.Linq.JToken Data) { try { System.IO.File.WriteAllText(FilePath, Data.ToString()); } catch { SaveJSON(FilePath, Data); } } }}

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

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading;namespace Twitch\_Discord\_Reward\_Bot.Backend{ public static class Init { public static Random Rnd = new Random(); public static Newtonsoft.Json.Linq.JToken MasterConfig = Data.FileHandler.ReadJSON("./Data/Master.config.json"); static Dictionary<int,BotInstance> Instances = new Dictionary<int, BotInstance> { }; public static void Start() { string S = Data.APIIntergrations.RewardCurrencyAPI.WebRequests.GetAuthToken(); if (S == null) { Console.WriteLine("API Tokens misconfigured!"); } else { while (true) { foreach (Newtonsoft.Json.Linq.JToken Currency in Data.APIIntergrations.RewardCurrencyAPI.WebRequests.PostRequest("currency/all", null, true).Data) { Data.APIIntergrations.RewardCurrencyAPI.Objects.Currency C = Data.APIIntergrations.RewardCurrencyAPI.Objects.Currency.FromJson(Currency); if (C.CommandConfig.Count() != 0 && C.LoginConfig.Count() != 0) { if (C.CommandConfig["BotsEnabled"].ToString() == "True") { if (!Instances.Keys.Contains(C.ID)) { Instances.Add(C.ID, new BotInstance(C)); } else { Instances[C.ID].CommandConfig = C.CommandConfig; Instances[C.ID].LoginConfig = C.LoginConfig; Instances[C.ID].Currency = C; Instances[C.ID].Start(); } } else if (Instances.Keys.Contains(C.ID)) { Instances[C.ID].Stop(); } } else { Console.WriteLine("There was a error relating to currency configs"); } } System.Threading.Thread.Sleep(300000); } } } } public class BotInstance { public Data.APIIntergrations.RewardCurrencyAPI.Objects.Currency Currency; public Backend.Bots.DiscordBot.Instance DiscordBot; public Backend.Bots.TwitchBot.Instance TwitchBot; public Backend.Bots.Commands.CommandHandler CommandHandler; public Bots.Commands.TimeEvents TimeEvents; public Newtonsoft.Json.Linq.JToken CommandConfig, LoginConfig; public Dictionary<string, Data.APIIntergrations.AccessToken> AccessTokens = new Dictionary<string, Data.APIIntergrations.AccessToken> { }; public bool Isrunning = false; public BotInstance(Data.APIIntergrations.RewardCurrencyAPI.Objects.Currency Currency) { this.Currency = Currency; this.CommandConfig = this.Currency.CommandConfig; this.LoginConfig = this.Currency.LoginConfig; //new Thread(() => CheckBotsAlive()).Start(); Start(); } public void Start() { if (Isrunning) { CheckBotsAlive(); return; } Isrunning = true; this.CommandHandler = new Bots.Commands.CommandHandler(this); try { DiscordBot = new Backend.Bots.DiscordBot.Instance(this); } catch { } try { TwitchBot = new Backend.Bots.TwitchBot.Instance(this); } catch { } System.Threading.Thread.Sleep(5000); if (this.TimeEvents != null) { this.TimeEvents.Stop(); } this.TimeEvents = new Bots.Commands.TimeEvents(); this.TimeEvents.Start(this); } public void CheckBotsAlive() { while (true) { Thread.Sleep(10000); if (Isrunning) { if (!TwitchBot.Client.IsConnected) { try { TwitchBot.Client.Connect(); } catch { } } if (DiscordBot.Client.ConnectionState == Discord.ConnectionState.Disconnected) { try { DiscordBot.Client.StartAsync(); } catch { } } } } } public void Stop() { if (!Isrunning) { return; } Isrunning = false; DiscordBot.Client.StopAsync(); TwitchBot.Client.Disconnect(); TimeEvents.Stop(); this.CommandHandler = null; TimeEvents = null; DiscordBot = null; TwitchBot = null; Console.WriteLine("Stopped " + Currency.ID + " Bots"); } }}

Twitch-Discord-Reward-Bot/Program.cs

using System;namespace Twitch\_Discord\_Reward\_Bot{ class Program { static void Main(string[] args) { Backend.Init.Start(); while (true) { Console.ReadLine(); } } }}