**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехника»

Кафедра «Информатика и вычислительная техника»

Курс «Парадигмы и конструкции языков программирования»

Отчет по домашнему заданию

Тема «Менеджер – дискорд бот»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-31Б: |  | преподаватель каф. ИУ5 |
| Ергалиев А.Г. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2023 г.

**ОПИСАНИЕ ЗАДАНИЯ**

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

Я создавал дискорд бота, который будет выполнять несколько функций:

1. Создавать голосовые каналы по велению любого пользователя на сервере
2. Ограничение доступа к созданным каналам для определенных ролей и/или пользователей
3. Определение категории, в которой будут создаваться новые каналы

Бота писали на C#.

**ОПИСАНИЕ, ТЕКСТЫ ПРОГРАММ И ЭКРАННЫЕ ФОРМА**

**Основные Файлы Конфигурации и Сборки**

1. **DiscordChannelsBot.sln**: Файл решения Visual Studio, организующий все проекты и файлы в рамках вашего приложения Discord бота.

Microsoft Visual Studio Solution File, Format Version 12.00

# Visual Studio Version 16

VisualStudioVersion = 16.0.30406.217

MinimumVisualStudioVersion = 10.0.40219.1

Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "DiscordChannelsBot", "DiscordChannelsBot.csproj", "{1C465032-281B-4CD1-B15C-FEA090C84F23}"

EndProject

Global

GlobalSection(SolutionConfigurationPlatforms) = preSolution

Debug|Any CPU = Debug|Any CPU

Release|Any CPU = Release|Any CPU

EndGlobalSection

GlobalSection(ProjectConfigurationPlatforms) = postSolution

{1C465032-281B-4CD1-B15C-FEA090C84F23}.Debug|Any CPU.ActiveCfg = Debug|Any CPU

{1C465032-281B-4CD1-B15C-FEA090C84F23}.Debug|Any CPU.Build.0 = Debug|Any CPU

{1C465032-281B-4CD1-B15C-FEA090C84F23}.Release|Any CPU.ActiveCfg = Release|Any CPU

{1C465032-281B-4CD1-B15C-FEA090C84F23}.Release|Any CPU.Build.0 = Release|Any CPU

EndGlobalSection

GlobalSection(SolutionProperties) = preSolution

HideSolutionNode = FALSE

EndGlobalSection

GlobalSection(ExtensibilityGlobals) = postSolution

SolutionGuid = {3FD27C42-2BDE-4AFC-AE98-83EA35ACAD5D}

EndGlobalSection

EndGlobal

1. **DiscordChannelsBot.csproj**: Файл проекта Visual Studio, содержащий детали конфигурации, ссылки на библиотеки, файлы исходного кода и настройки сборки.

<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>

        <OutputType>Exe</OutputType>

        <TargetFramework>net7.0</TargetFramework>

        <ImplicitUsings>enable</ImplicitUsings>

        <Nullable>disable</Nullable>

        <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>

    </PropertyGroup>

    <ItemGroup>

        <PackageReference Include="Discord.Net" Version="3.10.0"/>

        <PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.5"/>

        <PackageReference Include="Microsoft.Extensions.DependencyInjection" Version="7.0.0"/>

        <PackageReference Include="Microsoft.Extensions.Hosting" Version="7.0.1"/>

        <PackageReference Include="Newtonsoft.Json" Version="13.0.3"/>

        <PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL" Version="7.0.4"/>

    </ItemGroup>

    <ItemGroup>

        <None Update="appsettings.json">

            <CopyToOutputDirectory>Always</CopyToOutputDirectory>

        </None>

    </ItemGroup>

</Project>

1. **global.json**: Определяет используемые версии инструментов .NET для обеспечения совместимости и стабильности проекта.

{

  "sdk": {

    "version": "7.0.0",

    "rollForward": "latestMajor",

    "allowPrerelease": false

  }

}

1. **appsettings.json**: Хранит конфигурационные настройки приложения, такие как строки подключения и ключи API.

{

  "Logging": {

    "LogLevel": {

      "Default": "Warning"

    }

  },

  "DiscordBot": {

    "Token": ""

  }

}

1. **docker-compose.yml**: Файл конфигурации Docker Compose, определяющий связи между контейнерами Docker.

version: "3.4"

networks:

  default-network:

    driver: bridge

services:

  discord-bot:

    container\_name: discord-bot

    build:

      context: .

      dockerfile: Dockerfile

    depends\_on: [ postgres\_image ]

    ports:

      - "8080:80"

    environment:

      DB\_CONNECTION\_STRING: host=discord-bot-postgres;port=5432;database=disdb;username=discord;password=discord

    #      REDIS\_CONNECTION\_STRING: redis\_image

    networks:

      - default-network

  #  redis\_image:

  #    image: redis:latest

  #    container\_name: redis.cache

  postgres\_image:

    container\_name: discord-bot-postgres

    image: postgres:latest

    ports:

      - "5432:5432"

    restart: always

    volumes:

      - db\_volume:/var/lib/postgresql/data

    environment:

      POSTGRES\_USER: "discord"

      POSTGRES\_PASSWORD: "discord"

      POSTGRES\_DB: "disdb"

    networks:

      - default-network

volumes:

  db\_volume:

1. **Dockerfile**: Описывает создание Docker-образа для приложения, включая инструкции по установке и конфигурации.

FROM mcr.microsoft.com/dotnet/runtime:7.0 AS base

WORKDIR /app

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build

WORKDIR /src

COPY ["DiscordChannelsBot.csproj", "./"]

RUN dotnet restore "DiscordChannelsBot.csproj"

COPY . .

WORKDIR "/src/"

RUN dotnet build "DiscordChannelsBot.csproj" -c Release -o /app/build

FROM build AS publish

RUN dotnet publish "DiscordChannelsBot.csproj" -c Release -o /app/publish

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "DiscordChannelsBot.dll"]

**Код Исходного Приложения**

1. **Program.cs**: Выполняет несколько ключевых функций:
2. **Инициализация и Запуск Приложения**: Основная точка входа в приложение (**Main** метод), который вызывает асинхронный метод **MainAsync** для инициализации и запуска бота.
3. **Создание и Конфигурация Хоста**: Метод **CreateHostBuilder** используется для создания и настройки хоста приложения, включая логирование и конфигурацию сервисов.
4. **Настройка Логирования**: Метод **ConfigureLogging** настраивает логирование, очищая существующие провайдеры и добавляя консольное логирование.
5. **Конфигурация Сервисов**: Метод **ConfigureServices** добавляет и конфигурирует различные зависимости и сервисы, такие как база данных (**ApplicationDbContext**), Discord клиент, сервисы для обработки команд и управления голосовыми каналами.
6. **Запуск Бота**: В методе **StartBot** происходит запуск бота и установка бесконечной задержки, чтобы приложение оставалось активным.

using Discord;

using Discord.Interactions;

using Discord.WebSocket;

using DiscordChannelsBot.CommandManagement.ChannelManagement;

using DiscordChannelsBot.CommandManagement.CommandHandling;

using DiscordChannelsBot.Configuration;

using DiscordChannelsBot.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

namespace DiscordChannelsBot;

internal class Program

{

    public static void Main(string[] args)

    {

        MainAsync(args).GetAwaiter().GetResult();

    }

    private static async Task MainAsync(string[] args)

    {

        using var host = CreateHostBuilder(args).Build();

        await using var serviceScope = host.Services.CreateAsyncScope();

        var serviceProvider = serviceScope.ServiceProvider;

        var bot = serviceProvider.GetRequiredService<DiscordBot>();

        await StartBot(bot);

    }

    private static async Task StartBot(DiscordBot bot)

    {

        await bot.StartAsync();

        await Task.Delay(Timeout.Infinite);

    }

    private static IHostBuilder CreateHostBuilder(string[] args)

    {

        return Host.CreateDefaultBuilder(args)

            .ConfigureLogging(ConfigureLogging)

            .ConfigureServices(ConfigureServices);

    }

    private static void ConfigureLogging(HostBuilderContext context, ILoggingBuilder loggingBuilder)

    {

        loggingBuilder.ClearProviders();

        loggingBuilder.AddConsole();

    }

    private static void ConfigureServices(HostBuilderContext context, IServiceCollection services)

    {

        AddDbContext(services);

        services.Configure<DiscordBotConfiguration>(context.Configuration.GetSection("DiscordBot"))

            .AddSingleton(\_ => new DiscordSocketClient(new DiscordSocketConfig

            {

                GatewayIntents = GatewayIntents.All

            }))

            // .AddSingleton<CommandService>()

            .AddSingleton<InteractionService>()

            .AddSingleton<IDiscordBotConfigurationService, DiscordBotFileBasedConfigurationService>()

            .AddSingleton<ICommandHandlingService, CommandHandlingService>()

            .AddSingleton<IVoiceChannelManagementService, VoiceChannelManagementService>()

            .AddSingleton<DiscordBot>();

    }

    private static void AddDbContext(IServiceCollection services)

    {

        var connectionString = Environment.GetEnvironmentVariable("DB\_CONNECTION\_STRING");

        services.AddDbContext<ApplicationDbContext>(options =>

            options.UseNpgsql(connectionString!)

        );

    }

}

Этот файл является критически важным для функционирования приложения, поскольку он управляет загрузкой, конфигурацией и запуском всех основных компонентов бота.

1. **DiscordBot.cs**: выполняет следующие основные функции:.
2. **Инициализация Компонентов**: Конструктор класса инициализирует основные компоненты, такие как `DiscordSocketClient`, конфигурация бота, сервисы для обработки команд и интеракций.
3. **Обработка Событий**: Зарегистрированы обработчики событий для логирования (`Log`), обработки команд и интеракций.
4. **Добавление Модулей**: Добавляет модуль управления каналами в `InteractionService`.
5. **Асинхронный Запуск**: Метод `StartAsync` асинхронно выполняет вход в систему бота и его запуск. После готовности бота регистрируются глобальные команды.
6. **Логирование**: Реализована функция для преобразования уровня серьезности лога в уровень логирования .NET.

using Discord;

using Discord.Interactions;

using Discord.WebSocket;

using DiscordChannelsBot.CommandManagement.CommandHandling;

using DiscordChannelsBot.Models;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

namespace DiscordChannelsBot;

public class DiscordBot : IAsyncExecutor

{

    private readonly DiscordSocketClient \_bot;

    private readonly DiscordBotConfiguration \_configuration;

    private readonly InteractionService \_interactionService;

    private readonly ILogger<DiscordBot> \_logger;

    public DiscordBot(IServiceProvider serviceProvider,

        IOptions<DiscordBotConfiguration> configuration,

        DiscordSocketClient bot,

        ILogger<DiscordBot> logger,

        ICommandHandlingService commandHandlingService,

        InteractionService interactionService)

    {

        \_configuration = configuration.Value;

        \_bot = bot;

        \_logger = logger;

        \_interactionService = interactionService;

        \_bot.Log += Log;

        \_bot.InteractionCreated += commandHandlingService.HandleMessageReceivedAsync;

        \_interactionService.AddModuleAsync<ChannelsManagementModule>(serviceProvider).ConfigureAwait(false).GetAwaiter()

            .GetResult();

        \_interactionService.InteractionExecuted += commandHandlingService.HandleCommandExecutedAsync;

    }

    public async Task StartAsync()

    {

        await \_bot.LoginAsync(TokenType.Bot, \_configuration.Token);

        await \_bot.StartAsync();

        \_bot.Ready += async () => await \_interactionService.RegisterCommandsGloballyAsync();

    }

    private Task Log(LogMessage log)

    {

        \_logger.Log(ConvertLogLevel(log.Severity), log.Exception, log.Message);

        return Task.CompletedTask;

    }

    private LogLevel ConvertLogLevel(LogSeverity logSeverity)

    {

        return logSeverity switch

        {

            LogSeverity.Critical => LogLevel.Critical,

            LogSeverity.Error => LogLevel.Error,

            LogSeverity.Warning => LogLevel.Warning,

            LogSeverity.Info => LogLevel.Information,

            LogSeverity.Verbose => LogLevel.Trace,

            LogSeverity.Debug => LogLevel.Debug,

            \_ => throw new ArgumentOutOfRangeException(nameof(logSeverity), logSeverity, null)

        };

    }

}

Этот файл критически важен для работы бота, так как он управляет его основными функциями, включая взаимодействие с Discord API, обработку событий и команд.

**Сервисы и Интерфейсы**

1. **VoiceChannelManagementService.cs** и **IVoiceChannelManagementService.cs**: Реализация и интерфейс для управления голосовыми каналами Discord.

**VoiceChannelManagementService.cs:**

1. **Создание Голосового Канал:** Метод `**CreateVoiceChannelAsync**` создаёт голосовой канал в Discord, используя конфигурацию гильдии и контекст групп. Он также устанавливает разрешения для доступа к каналу.
2. **Управление Разрешениями**: Метод `**AllowOnlyRolesAsync**` настраивает разрешения для ролей и пользователей в голосовом канале, определяя кто может видеть и подключаться к каналу.
3. **Обработка Изменений Состояния Пользователя**: Обработчик события `**UserVoiceStateUpdatedHandleAsync**` реагирует на изменения состояния пользователя в голосовом канале, например, когда пользователь подключается или отключается.
4. **Автоматическое Удаление Пустых Каналов**: Методы `**RunDeletionCheckAsync**` и `**CancelDeletionCheckIfExists**` управляют проверкой на удаление пустых голосовых каналов после определённого времени без активности.

using Discord;

using Discord.WebSocket;

using DiscordChannelsBot.Common;

using DiscordChannelsBot.Configuration;

using DiscordChannelsBot.Models;

using Microsoft.Extensions.DependencyInjection;

namespace DiscordChannelsBot.CommandManagement.ChannelManagement;

public class VoiceChannelManagementService : IVoiceChannelManagementService

{

    private readonly IDiscordBotConfigurationService \_discordBotConfigurationService;

    private readonly DiscordSocketClient \_discordClient;

    private readonly Dictionary<ulong, CancellationTokenSource>

        \_voiceChannelDeletionCheckCancellationTokenSourcesDictionary;

    public VoiceChannelManagementService(IServiceProvider serviceProvider)

    {

        \_voiceChannelDeletionCheckCancellationTokenSourcesDictionary =

            new Dictionary<ulong, CancellationTokenSource>();

        \_discordBotConfigurationService = serviceProvider.GetRequiredService<IDiscordBotConfigurationService>();

        \_discordClient = serviceProvider.GetRequiredService<DiscordSocketClient>();

        \_discordClient.UserVoiceStateUpdated += UserVoiceStateUpdatedHandleAsync;

    }

    public async Task CreateVoiceChannelAsync(IGuild guild, string name, GuildGroupsContext guildGroupsContext)

    {

        var guildConfiguration = await \_discordBotConfigurationService.GetGuildConfigurationAsync(guild.Id);

        if (guildConfiguration == null)

        {

            throw new ArgumentException("Не указана категория для создания голосовых каналов!");

        }

        var categoryChannel =

            await DiscordBotUtils.GetCategoryAsync(guild, guildConfiguration.VoiceChannelCreationCategory);

        Action<VoiceChannelProperties> voiceChannelProperties = \_ => { };

        if (categoryChannel != null)

        {

            voiceChannelProperties += channel => channel.CategoryId = categoryChannel.Id;

        }

        var voiceChannel = await guild.CreateVoiceChannelAsync(name, voiceChannelProperties);

        if (guildGroupsContext != null)

        {

            if (guildGroupsContext.Roles != null && guildGroupsContext.Roles.Any() ||

                guildGroupsContext.Users != null && guildGroupsContext.Users.Any())

            {

                await AllowOnlyRolesAsync(guild, voiceChannel, guildGroupsContext);

            }

        }

        RunDeletionCheckAsync(voiceChannel.Id);

    }

    private async Task AllowOnlyRolesAsync(IGuild guild, IVoiceChannel voiceChannel,

        GuildGroupsContext guildGroupsContext)

    {

        var rolePermissions = new OverwritePermissions().Modify(viewChannel: PermValue.Allow,

            connect: PermValue.Allow,

            speak: PermValue.Allow);

        var denyPermissions = new OverwritePermissions().Modify(viewChannel: PermValue.Deny,

            connect: PermValue.Deny,

            speak: PermValue.Deny);

        await voiceChannel.AddPermissionOverwriteAsync(guild.EveryoneRole, denyPermissions);

        await voiceChannel.AddPermissionOverwriteAsync(guildGroupsContext.CurrentUser, rolePermissions);

        if (guildGroupsContext.Roles != null)

        {

            foreach (var role in guildGroupsContext.Roles)

            {

                await voiceChannel.AddPermissionOverwriteAsync(role, rolePermissions);

            }

        }

        if (guildGroupsContext.Users != null)

        {

            foreach (var user in guildGroupsContext.Users)

            {

                await voiceChannel.AddPermissionOverwriteAsync(user, rolePermissions);

            }

        }

    }

    private Task UserVoiceStateUpdatedHandleAsync(SocketUser user, SocketVoiceState originalState,

        SocketVoiceState updatedState)

    {

        if (originalState.VoiceChannel != null)

        {

            var voiceChannel = originalState.VoiceChannel;

            if (voiceChannel.ConnectedUsers.Count == 0)

            {

                RunDeletionCheckAsync(voiceChannel.Id);

            }

        }

        if (updatedState.VoiceChannel != null)

        {

            var voiceChannel = updatedState.VoiceChannel;

            CancelDeletionCheckIfExists(voiceChannel.Id);

        }

        return Task.CompletedTask;

    }

    private async void RunDeletionCheckAsync(ulong voiceChannelId)

    {

        var cancellationTokenSource = new CancellationTokenSource();

        var cancellationToken = cancellationTokenSource.Token;

        \_voiceChannelDeletionCheckCancellationTokenSourcesDictionary.Add(voiceChannelId, cancellationTokenSource);

        try

        {

            await Task.Delay(TimeSpan.FromMinutes(1), cancellationToken);

            var voiceChannel = (SocketVoiceChannel) \_discordClient.GetChannel(voiceChannelId);

            if (voiceChannel != null && voiceChannel.ConnectedUsers.Count == 0)

            {

                await voiceChannel.DeleteAsync();

            }

        }

        catch (TaskCanceledException)

        {

        }

    }

    private void CancelDeletionCheckIfExists(ulong voiceChannelId)

    {

        if (\_voiceChannelDeletionCheckCancellationTokenSourcesDictionary.ContainsKey(voiceChannelId))

        {

            \_voiceChannelDeletionCheckCancellationTokenSourcesDictionary[voiceChannelId].Cancel();

            \_voiceChannelDeletionCheckCancellationTokenSourcesDictionary.Remove(voiceChannelId);

        }

    }

}

Этот файл важен для управления голосовыми каналами в вашем Discord боте, предоставляя функциональность для создания, настройки разрешений и автоматического удаления каналов.

**IVoiceChannelManagementService.cs:**

using Discord;

using DiscordChannelsBot.Models;

namespace DiscordChannelsBot.CommandManagement.ChannelManagement;

public interface IVoiceChannelManagementService

{

    Task CreateVoiceChannelAsync(IGuild guild, string name, GuildGroupsContext guildGroupsContext);

}

1. **CommandHandlingService.cs** и **ICommandHandlingService.cs**: Обработка и интерфейс для обработки команд, отправленных боту.

**CommandHandlingService.cs:**

Файл `CommandHandlingService.cs` вашего проекта Discord бота реализует следующие ключевые аспекты:

1. **Инициализация Сервисов**: Конструктор класса инициализирует необходимые сервисы, включая `**DiscordSocketClient**`, `**InteractionService**`, и сервисы для обработки команд.
2. **Обработка Команд**: Осуществляет обработку команд, поступающих от пользователей, используя `**InteractionService**`.
3. **Обработка Результата Команды**: Метод `**HandleCommandExecutedAsync**` управляет логикой ответов на результаты выполнения команд, включая обработку ошибок и неправильно сформулированных команд.
4. **Обработка Входящих Сообщений**: Метод `**HandleMessageReceivedAsync**` обрабатывает входящие взаимодействия и передает их на выполнение в `**InteractionService**`.

using Discord;

using Discord.Interactions;

using Discord.WebSocket;

using DiscordChannelsBot.Models;

using Microsoft.Extensions.Options;

namespace DiscordChannelsBot.CommandManagement.CommandHandling;

internal class CommandHandlingService : ICommandHandlingService

{

    private readonly DiscordSocketClient \_bot;

    private readonly InteractionService \_interactionService;

    private readonly IServiceProvider \_serviceProvider;

    public CommandHandlingService(IServiceProvider serviceProvider,

        IOptions<DiscordBotConfiguration> discordBotConfiguration, InteractionService interactionService,

        DiscordSocketClient bot)

    {

        \_serviceProvider = serviceProvider;

        \_interactionService = interactionService;

        \_bot = bot;

    }

    public async Task HandleCommandExecutedAsync(ICommandInfo command, IInteractionContext context, IResult result)

    {

        if (result.IsSuccess)

        {

            return;

        }

        switch (result.Error)

        {

            case InteractionCommandError.BadArgs:

                await context.Interaction.RespondAsync(

                    "Произошла ошибка. Слишком мало аргументов в команде.", ephemeral: true);

                return;

            case InteractionCommandError.Exception:

                await context.Interaction.RespondAsync($"Произошла ошибка. {result.ErrorReason}", ephemeral: true);

                return;

            case InteractionCommandError.Unsuccessful:

                await context.Interaction.RespondAsync("Произошла ошибка. Команда не выполнена.", ephemeral: true);

                return;

            case InteractionCommandError.ParseFailed:

                await context.Interaction.RespondAsync("Произошла ошибка. Команда не распознана.", ephemeral: true);

                return;

            default:

                await context.Interaction.RespondAsync("Произошла ошибка.", ephemeral: true);

                return;

        }

    }

    public async Task HandleMessageReceivedAsync(SocketInteraction interaction)

    {

        await \_interactionService.ExecuteCommandAsync(new SocketInteractionContext(\_bot, interaction),

            \_serviceProvider);

    }

}

Этот файл критически важен для функционирования бота, так как он обеспечивает обработку и выполнение команд, получаемых от пользователей Discord.

**ICommandHandlingService.cs**, определяет интерфейс для сервиса обработки команд:

using Discord;

using Discord.Interactions;

using Discord.WebSocket;

namespace DiscordChannelsBot.CommandManagement.CommandHandling;

public interface ICommandHandlingService

{

    Task HandleCommandExecutedAsync(ICommandInfo command, IInteractionContext context,

        IResult result);

    Task HandleMessageReceivedAsync(SocketInteraction interaction);

}

1. **DiscordBotConfiguration.cs** и **IDiscordBotConfigurationService.cs**: Конфигурация бота и интерфейс для ее управления.

**DiscordBotConfiguration.cs:**

namespace DiscordChannelsBot.Models;

public class DiscordBotConfiguration

{

    public string Token { get; init; }

}

**IDiscordBotConfigurationService.cs**:

using DiscordChannelsBot.Models;

namespace DiscordChannelsBot.Configuration;

public interface IDiscordBotConfigurationService

{

    ValueTask<DiscordGuildConfiguration> GetGuildConfigurationAsync(ulong guildId);

    Task UpdateAsync(DiscordGuildConfiguration discordGuildConfiguration);

    Task SaveAsync(DiscordGuildConfiguration discordGuildConfiguration);

}

1. **DiscordBotFileBasedConfigurationService.cs**: Реализация сервиса конфигурации, основанная на файлах.

using DiscordChannelsBot.Models;

namespace DiscordChannelsBot.Configuration;

public class DiscordBotFileBasedConfigurationService : IDiscordBotConfigurationService

{

    private readonly ApplicationDbContext \_applicationDbContext;

    public DiscordBotFileBasedConfigurationService(ApplicationDbContext applicationDbContext)

    {

        \_applicationDbContext = applicationDbContext;

    }

    public async ValueTask<DiscordGuildConfiguration> GetGuildConfigurationAsync(ulong guildId)

    {

        return await \_applicationDbContext.GuildConfigurations

            .FindAsync(guildId);

    }

    public async Task UpdateAsync(DiscordGuildConfiguration discordGuildConfiguration)

    {

        \_applicationDbContext.GuildConfigurations.Update(discordGuildConfiguration);

        await \_applicationDbContext.SaveChangesAsync();

    }

    public async Task SaveAsync(DiscordGuildConfiguration discordGuildConfiguration)

    {

        await \_applicationDbContext.GuildConfigurations.AddAsync(discordGuildConfiguration);

        await \_applicationDbContext.SaveChangesAsync();

    }

}

1. **IAsyncExecutor.cs**: Интерфейс для асинхронных операций в проекте.

namespace DiscordChannelsBot;

public interface IAsyncExecutor

{

    Task StartAsync();

}

**Модули и Утилиты**

1. **ChannelsManagementModule.cs**: Модуль для управления различными типами каналов в Discord.

1. **Создание Голосового Канала**: Реализована команда **`/voice**`, позволяющая создавать голосовые каналы с настраиваемыми разрешениями для выбранных пользователей или ролей.

2. **Установка Категории Голосовых Каналов**: Команда `**/voice-category**` позволяет устанавливать категорию, в которой будут создаваться голосовые каналы.

3. **Интеграция с Сервисами**: Класс использует `**IVoiceChannelManagementService**` и `**IDiscordBotConfigurationService**` для управления каналами и конфигурацией.

using Discord;

using Discord.Interactions;

using DiscordChannelsBot.CommandManagement.ChannelManagement;

using DiscordChannelsBot.Configuration;

using DiscordChannelsBot.Models;

namespace DiscordChannelsBot.CommandManagement.CommandHandling;

public class ChannelsManagementModule : InteractionModuleBase<SocketInteractionContext>

{

    public IDiscordBotConfigurationService DiscordBotConfigurationService { get; init; }

    public IVoiceChannelManagementService VoiceChannelManagementService { get; init; }

    [SlashCommand("voice", "Create a voice channel")]

    [RequireContext(ContextType.Guild)]

    public async Task CreateVoiceChannel([Summary("Channel")] string channel,

        [Summary("Permission1", "A person or role that can will able to join the channel")]

        IMentionable firstPermission = null,

        [Summary("Permission2", "A person or role that can will able to join the channel")]

        IMentionable secondPermission = null,

        [Summary("Permission3", "A person or role that can will able to join the channel")]

        IMentionable thirdPermission = null,

        [Summary("Permission4", "A person or role that can will able to join the channel")]

        IMentionable fourthPermission = null,

        [Summary("Permission5", "A person or role that can will able to join the channel")]

        IMentionable fifthPermission = null)

    {

        await CreateVoiceChannel(channel,

            new List<IMentionable>

                    {firstPermission, secondPermission, thirdPermission, fourthPermission, fifthPermission}

                .Where(e => e != null).ToList());

    }

    private async Task CreateVoiceChannel(string channel, List<IMentionable> permissions)

    {

        await Context.Guild.DownloadUsersAsync();

        var guildRolesAndUsers = new GuildGroupsContext

        {

            CurrentUser = Context.User,

            Roles = permissions.OfType<IRole>(),

            Users = permissions.OfType<IUser>()

        };

        await VoiceChannelManagementService.CreateVoiceChannelAsync(Context.Guild, channel, guildRolesAndUsers);

        await Context.Interaction.RespondAsync($"{Context.User.Mention} добавил канал под названием \*\*{channel}\*\*.");

    }

    [RequireUserPermission(ChannelPermission.ManageChannels)]

    [SlashCommand("voice-category", "Set a guild category where voice channels will be created")]

    [RequireContext(ContextType.Guild)]

    public async Task SetVoiceCategory([Summary("Category")] string category)

    {

        var guildConfiguration = await DiscordBotConfigurationService.GetGuildConfigurationAsync(Context.Guild.Id);

        if (guildConfiguration == null)

        {

            guildConfiguration = new DiscordGuildConfiguration

            {

                Id = Context.Guild.Id,

                VoiceChannelCreationCategory = category

            };

            await DiscordBotConfigurationService.SaveAsync(guildConfiguration);

        }

        else

        {

            guildConfiguration.VoiceChannelCreationCategory = category;

            await DiscordBotConfigurationService.UpdateAsync(guildConfiguration);

        }

        await Context.Interaction.RespondAsync(

            $"Теперь голосовые каналы будут создаваться в категории \*\*{category}\*\*.", ephemeral: true);

    }

}

1. **DiscordBotUtils.cs**: Утилитный файл с общими функциями и помощниками.

using Discord;

namespace DiscordChannelsBot.Common;

public static class DiscordBotUtils

{

    public static async Task<ICategoryChannel> GetCategoryAsync(IGuild guild, string categoryName)

    {

        var categoryChannels = await guild.GetCategoriesAsync();

        return categoryChannels.FirstOrDefault(category => category.Name.Equals(categoryName));

    }

}

**База Данных и Контексты**

1. **ApplicationDbContext.cs**: Контекст базы данных для управления взаимодействием с базой данных.

using Microsoft.EntityFrameworkCore;

namespace DiscordChannelsBot.Models;

public sealed class ApplicationDbContext : DbContext

{

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

    {

        Database.EnsureCreated();

        Database.Migrate();

    }

    public DbSet<DiscordGuildConfiguration> GuildConfigurations { get; set; }

}

1. **GuildGroupsContext.cs**: Обработка данных о группах в сервер Discord.

using Discord;

namespace DiscordChannelsBot.Models;

public class GuildGroupsContext

{

    public IUser CurrentUser { get; init; }

    public IEnumerable<IRole> Roles { get; init; }

    public IEnumerable<IUser> Users { get; init; }

}

1. **DiscordGuildConfiguration.cs**: Конфигурация для отдельных серверов в Discord.

namespace DiscordChannelsBot.Models;

public class DiscordGuildConfiguration

{

    public ulong Id { get; init; }

    public string VoiceChannelCreationCategory { get; set; }

}

**ЭКРАННЫЕ ФОРМЫ РАБОТЫ БОТА:**





