МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

РАЗРАБОТКА DISCORD-БОТА

Работу выполнил _____ Астежев А.О.
(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Технология программирования

Научный руководитель
д-р тех. наук, зав. каф. _____ Коваленко А.В.
(подпись)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ Калайдина Г.В.
(подпись)

Краснодар
2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

РАЗРАБОТКА DISCORD-БОТА

Работу выполнил _____ Астежев А.О.
(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Технология программирования

Научный руководитель
д-р тех. наук, зав. каф. _____ Коваленко А.В.
(подпись)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ Калайдина Г.В.
(подпись)

Краснодар
2022

РЕФЕРАТ

Курсовая работа 41 с., 20 рис., 10 источников.

DISCORD, МУЛЬТИМЕДИА, АДМИНИСТРИРОВАНИЕ СЕРВЕРА, НАБОР ФУНКЦИЙ, LAVA PLAYER, РОЛИ, ПРАВА, БОТ, РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ, JAVA

Цель курсовой работы – разработать Discord-бота, который призван расширить инструментарий мессенджера путем внедрения недостающих функций в Discord. Задача бота обеспечивать поддержку отсутствующих команд в мессенджере, а также добавить возможность вспроизведения мультимедиа файлов в формате аудио с последующей трансляцией в голосовые чаты.

Курсовая работа состоит из 3 глав. В первой главе будет расписана необходимость разработки данного типа продукта с приведением аргументов, основанных на статистике и пожеланиях сообщества. Во второй главе подробно расписаны принципы работы бота с примерами кода и его описанием. В третьей главе представлены результаты работы бота с последующими пояснениями. В заключении подведены итоги работы и рассказывется про полученный опыт после работы с сервисом Discord и фреймворком JDA в частности.

Также в результате работы были приобретен опыт работы с некоторыми фреймворками на языке программирования Java, понимание общих принципов ООП, взаимодействия с внешними интернет-ссылками и получено более широкое представление о работе мессенджера.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы мессенджеры набирают огромную популярность в связи с надобностью возможности быстрого обмена информации. Хотя текстовые мессенджеры не теряют популярности, существует альтернатива в виде голосовых мессенджеров, например таких как Skype, Microsoft Teams, Chanty и т.д.. Они обеспечивают возможность обмена информации более оперативным способом и могут передовать информацию в более развёрнутом виде, за счет технологии потокового вещания голоса в чат.

Со временем начало появляться всё больше приложений, обеспечивающих потоковое вещание голоса пользователя. Это связано с распостранением IT-технологий по всему миру, где теперь практически у каждого есть возможность выхода в интернет. Голосовые мессенджеры перестали быть просто средством офисных работников для организации онлайн-встреч или обмена файлами, и сейчас никто не может представить жизнь без мессенджеров.

Одним из таких приложений является Discord. Изначально созданный для поддержания связи игроков в онлайн-игры, он быстро заимел популярность среди них. В Discord можно создавать сервера и приглашать туда участников для дальнейшего общения и обмена информации. Свой сервер может создать абсолютно любой зарегистрированнный пользователь и настроить его как ему угодно – установить любое лого, добавить описание, а с недавних пор и задать тематику сервера. Также мессенджер обеспечивает быструю и без задержек передачу голоса благодаря аудио-кодеку “Opus”. “Opus” создаёт возможность передачи голоса в высоком качестве, хотя и можно выбирать качество передачи голоса для понижения потребления трафика. Discord предоставляет широкие возможности структурирования сервера, т.е. имеется возможность создавать каналы – как голосовые, так и текстовые для разграничивания участников и информации в каналах. Обо всех особенностях мессенджера пойдёт речь в главе 1.

Возможности мессенджера можно расширить с помощью ботов, которые можно добавить на любой сервер, где они выполняют всю работу без дополнительного вмешательства пользователей. Целью работы является создание бота, содержащего базовый функционал, дополняющий возможности Discord’a. А именно возможность воспроизведения некоторых мультмедиа файлов, а также организация более гибкого модерирования сервера.

Для реализации поставленной цели сформулированы следующие задачи:

– Выбор фреймворка для создания функционала бота.

– Структурирование команд бота.

– Обеспечение поддержки плагина воспроизведения мультимедиа файлов и обеспечение возможности их поиска в интернете.

Discord-боты сейчас, как никогда пользуются популярностью не только среди онлайн-игроков, но и среди обычных пользователей персональных компьютеров [1]. Из-за причины всемирной изоляции многие учебные учреждения были вынуждены найти альтернативные способы обучения учащихся. Такие сервисы как Microsoft Teams, Zoom и Discord оказались как нельза кстати. Но Discord обладает рядом приемуществ перед своими конкурентами, в частности более интуитивно понятный интерфейс, бесплатность и надёжность. Всё это достигнуто путем многолетней поддержки приложения [2].

Все недоступные функции добавляются, как было сказано ранее, с помощью ботов. Discord имеет лояльное сообщество пользователей, в связи с чем существет множество фреймворков на совершенно различных языках, из-за чего бота может создать любой, кто хоть немного знаком с основами программирования.

1 Возможности современных голосовых мессенджеров

С каждым годом потребность в быстрой передаче информации возрастает и классические текстовые мессенджеры не позволяют вести диалог с такой же скоростью и взаимопониманием, как живое общение. В связи с этим в некоторые текстовые мессенджеры, например такие как Telegram [3], начали внедрять голосовые чаты. Но до всех этих нововведений существовали и приложения, которые уже имели такую возможность. Первым популярным приложением, поддерживающим голосовые чаты был Skype, но со временем он начал терять популярность в силу моральной устарелости и “сырости” приложения [4]. Разработка аналога была лишь вопросом времени.

Как говорилось ранее, обычные пользователи не пользовались голосовыми мессенджерами, они были нужны и изначально разрабатывались для офисных работников, чтобы предоставить им возможность проводить совещания или обмениваться файлами онлайн. С развитием интернета, голосовыми мессенджерами начали интересоваться и игроки в компьютерные игры. К сожалению, по вышеуказанным причинам, Skype не обеспечивал должного комфорта, а аналоги по типу TeamSpeak быстро устарели. Тогда в 2015 году разработчик Hammer & Chisel взялись за разработку приложения Bonfire, который позже будет переименован в Discord. Финансирование производилось на средства полученные со стартапа, но позже Hammer & Chisel заручились поддержкой таких финансовых гигантов как Benchmark и Tencent [5]. Разработчики стремились создать сервис, включающий в себя все плюсы конкурентов в лице вышеупомянутых Skype и TeamSpeak. Сервис быстро набрал популярность, благодаря популяризации киберспортсменами и сообществом сервиса Twitch. На 2016 год число зарегестрированных пользователей составило более 11 млн. человек. Компания разработчик начала внедрять новые технологии в сервис. Так, например, в октябре 2017 года был добавлен видеочат, а 11 марта 2020 года максимальное количество пользователей в чате было увеличено с 10 до 50 человек, в связи с объявлением

режима всемирной самоизоляции. В 2017 году была добавлена возможность приобретения подписки “Discord-Nitro”, которая снимала некоторые ограничения на пользователей. С увеличением количества человек в сообществе сервиса, подписка стала одним из основных источников дохода [6]. Также были внедрены новые технологии для обеспечения более высокого качества передачи аудио и комфорта пользователей – Noise Suppression by Krisp. Технология позволяет подавлять лишние шумы, захватываемые устройством ввода и делать передаваемый голос чище [7].

Также Discord превнёс новые возможности, которые упрощали администрирование сервера. Была создана возможность создавать роли – набор прав пользователя. На больших серверах стало возможно организовывать свой собственный порядок путём ограничения прав некоторых пользователей и привелерегирования других. Благодаря гибкой настройке ролей, можно отключать возможность нахождения в голосовых чатах, ограничивать количество сообщений в текстовых чатах или отключать возможность передачи файлов. Можно даже убирать из видимости голосовые и текстовые чаты.

С изменением курса развития сервиса в 2020 году [6] с сервиса для общения онлайн-игроков на сервис для общения любых пользователей, востребованность ролей и прав возросла. Плюс ко всему этому сообщество на 2019 год составляло более 250 млн. человек, а это означало, что нужно создать универсальный инструмент для администрирования больших серверов и удовлетворения потребностей пользователей, например таких как транслирование мультимедиа файлов в голосовые чаты. Благодаря открытому исходному коду и дружелюбности разработчиков к сообществу, на данный момент существует богатый набор фреймворков для написания собственного бота, разработанного для нужд собственного сервера [8]. Все эти фреймворки также имеют открытый исходный код и обновляются на регулярной основе. Каждый из них имеет свои плюсы и недостатки, но в целом они выполняют надлежащие задачи.

Также для раскрытия функционала бота используются плагины. Они не имеют привязки к какому-либо фреймворку и их можно подключить к практически любой программе, написанной на любом фреймворке. Плагины упрощают разработку функционала бота. Не нужны разрабатывать функционал самостоятельно, достаточно просто подключить нужный плагин.

В главе 2 будет рассмотрен бот под названием “Bot Marvin” созданный на фреймворке JDA (Java Discord API) [9], написанный на языке программирования Java, а также реализация плагина LavaPlayer [10].

2 Описание разработанного бота

2.1 Описание средств разработки

Для начала разработки бота нужно выбрать язык программроания на котором он будет базироваться. Было решено писать код на языке Java, т.к. он отлично подходит для программ, нуждающихся в четкой структуризации кода и реализует в себе все принципы ООП. ООП является ключевой технологией для создания бота. Было создано достаточно большое количество классов и было реализовано несколько интерфейсов. Возможно, проще было бы реализовать весь функционал бота на языке Python или JavaScript, но они не предоставляют нужной скорости исполнения кода и её должной структуризации.

Фреймворк JDA предоставляет доступ ко всем нужным встроенным методам Discord’a. Можно получить данные об авторе сообщения в текстовом канале, состояние отдельного пользователя (находится в голосовом канале или нет) и т.д. . Всё это возможно благодаря классификации так называемых событий или же “event”, как именуется в самом фреймворке. Каждое событие имеет свои поля и методы, которые можно использовать посредством классических get- set- методов. Так например, если участник (в фреймворке member) отправит сообщение в текстовый канал, сработает ListenerAdapter и он зафиксирует event, который будет классифицирован как событие получения сообщения (в фреймфорке onMessageReceived выступает как метод фиксирования события, а событие имеет название MessageReceivedEvent). Благодаря таким несложным обработчикам событий, возможна реализация собственных команд, которые могут выполнять всё, что угодно, их реализация зависит только от способностей и задумки разработчика.

С помощью событий в основном и строится логика работы бота. Для того, чтобы отследить определённую команду, достатоно отследить событие

получения сообщения и с помощью условных операторов языка программирования создать обработчик, активирующийся при наличии кодового слова. Но просто создать обработчик для кодового слова недостаточно. Существуют префиксы, которые служат разделителем сообщений на простые сообщения, оставляемые пользователями и на команды. Префиксы можно задать любые, чаще всего это символы “~”, “-”, “&” и прочие их вариации. Можно сделать и так, что бот может исполнять команды в зависимости от префикса, но в сообществе разработчиков ботов принято применять только один префикс, чтобы не создавать проблем ботам, например, в ситуации, когда на сервере присутствуют несколько ботов. Также Discord предлагает использовать префикс “/” для расширения набора команд, ведь этот префикс используется для встроенных в сам мессенджер команд. В любом случае принято придумывать свои префиксы, если бот создан для коммерческого использования. Не обязательно префиксом должен быть символ, префиксом может быть и слово, как это реализовано, например, в Ear Tensifier мультимедиа боте, где префиксом служит слово “ear”. В этой работе префиксом послужит символ “~”, но его можно заменить вслучае надобности. Префикс хранится как поле основного класса App и поменять его не составит труда, ведь остальные классы обращаются к нему по ссылке, а не хранят префикс в условии условного оператора.

Для организации учета входа и выхода участников на/с сервера был создан класс Events, где отслеживаются все соответствующие события. Их можно отслеживать с помощью GuildMemberJoinEvent и GuildMemberRemoveEvent. При входе, бот составляет Embed, т.е. формирует сообщение с расширенным функционалом оформления. В него можно добавить описание, название, задать цвет. Всё это выглядит на порядок лучше встроенных приветствий в сам мессенджер, а текст приветствия можно задавать самому. Также и с сообщением об уходе с сервера. В данной работе также реализована функция выдачи новому участнику роли, которая также реализована через GuildMemberJoinEvent.

Функционал фреймворка не ограничивается этими тремя событиями и возможно расширение функционала во много раз. Был рассмотрен только основной функционал, требуемый для классического сервера, а именно вышерассмотренные функции и ещё пару функций, не входящих в основной набор библиотеки, но реализованных на его методах.

Одна из таких функций это воспроизведение мультимедиа файлов. Реализация проводилась на основе JDA, но в себя включает только привязка плагина LavaPlayer, также написанном на Java. LavaPlayer является одним из самых популярных плееров для воспроизведения медиа файлов. Причиной выбора является возможность поиска файлов в интернете. Поиск может производиться на одном из ресурсов, а именно:

- Видеохостинг YouTube
- SoundCloud
- Bandcamp
- Vimeo
- Потоки Twitch
- Поддержка локальных файлов и HTTP URL-запросов

Плеер поддерживает множество современных форматов мультимедиа файлов. Вот их список[10]:

- MP3
- FLAC
- WAV
- WebM
- MP4/M4A
- OGG
- AAC
- M3U/PLS

Реализация поддержки осуществлена посредством написания вспомогательных классов, рассмотрение которых будет далее в главе 2.

Рассмотрим функционал Bot'а Marvin. Всего в реализации используется 9 классов, где 5 из них являются вспомогательными классами для поддержки мультимедиа плеера, три класса реализуют отдельные системы, а именно: команды, события и систему накопленного опыта, и главный класс, который с помошью встроенного в JDA класса JDA связывает все остальные классы.

Для начала рассмотрим класс App, который является главным классом. Именно он исполняется при запуске программы (Рисунок 1).

```
public class App {
    5 usages
    public static JDA jda;
    6 usages
    public static String prefix = "~";

    //Main method
    public static void main(String[] args) throws LoginException {
        JDACommands jdaCommands = new JDACommands( prefix: "~");
        jdaCommands.registerCommand(new PlayCommand());

        jda = JDABuilder.createDefault(
                token: "OTU2MjI2MjkwODMzMzIyMDg0.YjtJIQ.8buaF7ZbC3zmxZK7X_LIq1lb_bA",
                GatewayIntent.GUILD_MEMBERS,
                GatewayIntent.GUILD_MESSAGES,
                GatewayIntent.GUILD_VOICE_STATES
        ).enableCache(CacheFlag.VOICE_STATE).build();
        jda.addEventListener(new Commands());
        jda.addEventListener(new Events());
        jda.addEventListener(new ExpSystem());
        jda.addEventListener(jdaCommands);
    }
}
```

Рисунок 1 – представление класса App

В начале создаётся экземпляр класса JDA, который, собственно, и инициализирует бота и является основным объектом, над которым уже

наращивается остальной функционал. Далее задаётся префикс, который является объектом класса String.

Все взаимосвязи и определения объектов выполняются в главном и единственном методе класса App – main. Объект класса JDACommands выступает в роли вспомогательного объекта и привязывает класс PlayCommand, требуемый для возможности воспроизведения мультимедиа файлов и являющийся основным для всех остальных вспомогательных классов. Теперь необходимо создать бота и привязать его токен к программе. Токен является уникальным для каждого бота и получается на специальном сайте при его создании на официальном сайте Discord. Также в конструктор передаются интенты, требуемые для корректной работы классов. Если не передавать конкретные интенты, то будет передан интент DEFAULT, который в теории включает в себя все стартовые интенты, но на практике лучше передавать нужные интенты вручную. CacheFlag.VOICE_STATE нужен для воспроизведения мультимедиа файлов, без него ничего не будет работать в силу особенностей плагина LavaPlayer. Бот попросту не будет иметь возможность вещать аудио в голосовой канал, а соответственно плагин выдаст ощибку. В конце метода выполняется добавление Listener’ов, которыми являются созданные классы.

Класс Commands является довольно большим классом, поэтому рассмотрим по отдельности его методы, а не весь класс в целом. Методы Commands ничем по сути не отличаются – все их условные операторы это просто проверка на эквивалентность полученного сообщения с нужной командой, а активация команд происходит с помощью Listener’a, который ожидает получения события MessageReceivedEvent, после чего уже проводится обработка полученного сообщения.

На **Error! Reference source not found.** видно, что полученное сообщение разделяется символом пробела (в Java лучше писать “\\s+”, который является символьным представлением пробела, чтобы избежать ошибок) для дальнейшей обработки. Благодаря разделению, можно получить любой

аргумент команды и составлять команды имеющие более одного аргумента, тем самым не ограничивая себя в функционале комнад.

Также на Рисунок 2 представлена одна из команд “hi”, которая имеется в каждом боте просто для проверки работоспособности бота. Она не несёт особого функционала, просто отвечает пользователю в ответ одним из заранее заданных сообщений.

```
public class Commands extends ListenerAdapter {
    public void onMessageReceived(MessageReceivedEvent event) {
        String[] args = event.getMessage().getContentRaw().split( regex: "\\s+");

        //Command that's just send "hi" message
        if (args[0].equalsIgnoreCase( anotherString: App.prefix + "hi")) {
            String[] hiMessages = {
                    "Hi there, [member] :fire:",
                    "Oh hello there :sunglasses:",
                    "...ummm so... Ahh, hi [member]!"
            };
            Random rand = new Random();
            int number = rand.nextInt(hiMessages.length);

            event.getChannel().sendTyping().queue();
            event.getChannel().sendMessage(hiMessages[number].replace( target: "[member]", event.getMember().getAsMention())).queue();
        }
```

Рисунок 2 – Метод-обработчик команды "hi"

Следующий метод в классе отвечает за вывод информации о боте и работает по тому же принципу, что и “hi”. На представлена реализация метода.

```
//Information about bot
if (args[0].equalsIgnoreCase( anotherString: App.prefix + "info")) {
    EmbedBuilder info = new EmbedBuilder();
    info.setTitle(":mortar_board: Marvin bot :mortar_board:");
    info.setDescription("Server administration and multimedia bot");
    info.setColor(0xffffff);
    info.setFooter( text: "Created by Alan Pugachev", event.getMember().getUser().getAvatarUrl());
    event.getChannel().sendMessageEmbeds(info.build()).queue();
    info.clear();
}
```

Рисунок 3 – Метод-обработчик команды "info"

Следующая команда удаляет определённое заданное количество сообщений, но из-за ограничений на ботов самим Discord, необходимо

использовать блок try-catch для предотвращения ошибок. Также в случае нарушения синтаксиса команды был создан условный оператор, который предотвращает ошибку и информирует пользователя о неправильном синтаксисе. Ниже представлена метод-обработчик команды “clear” (рисунки Рисунок *4* и Рисунок *5*).

```
//Command that's clear chat
if (args[0].equalsIgnoreCase( anotherString: App.prefix + "clear")) {
    if (args.length != 2) {
        // Wrong usage
        EmbedBuilder usage = new EmbedBuilder();
        usage.setColor(0xff3923);
        usage.setTitle(":warning: Specify amount to delete");
        usage.setDescription("Usage: '" + App.prefix + "clear [amount of messages]'");
        event.getChannel().sendMessageEmbeds(usage.build()).queue();
        usage.clear();
    }
    else {
        try {
            List<Message> messages = event.getChannel().getHistory().retrievePast( amount: Integer.parseInt(args[1]) + 1).complete();
            event.getTextChannel().deleteMessages(messages).queue();

            // Success
            EmbedBuilder success=new EmbedBuilder();
            success.setColor(0x22ff2a);
            success.setTitle(":white_check_mark: Successfully deleted " + args[1] + ".");
            event.getChannel().sendMessageEmbeds(success.build()).queue();
        }
```

Рисунок 4 – Метод-обработчик команды "clear" (без блока catch)

```
catch (IllegalArgumentException e) {
    //e.printStackTrace();

    if (e.toString().startsWith("java.lang.IllegalArgumentException: Message retrieval")) {
        //Too many messages
        EmbedBuilder error = new EmbedBuilder();
        error.setColor(0xff3923);
        error.setTitle(":warning: Too many messages selected");
        error.setDescription("Between 1-100 messages can be deleted at one time");
        event.getChannel().sendMessageEmbeds(error.build()).queue();
        error.clear();
    }
    else {
        //Messages too old
        EmbedBuilder error = new EmbedBuilder();
        error.setColor(0xff3923);
        error.setTitle(":warning: Selected messages are older than 2 weeks");
        error.setDescription("Messages older than 2 weeks cannot be deleted.");
        event.getChannel().sendMessageEmbeds(error.build()).queue();
        error.clear();
    }
}
```

Рисунок 5 – Метод-обработчик команды "clear" (блок catch)

На видно, что блок catch срабатывает при двух случаях: больше 100 сообщений и сообщения существуют дольше 2 недель. Это и есть те самые ограничения на ботов

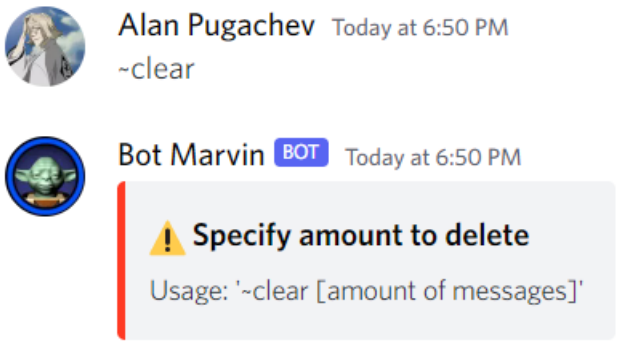

Рисунок 6 – Ошибка в синтаксисе

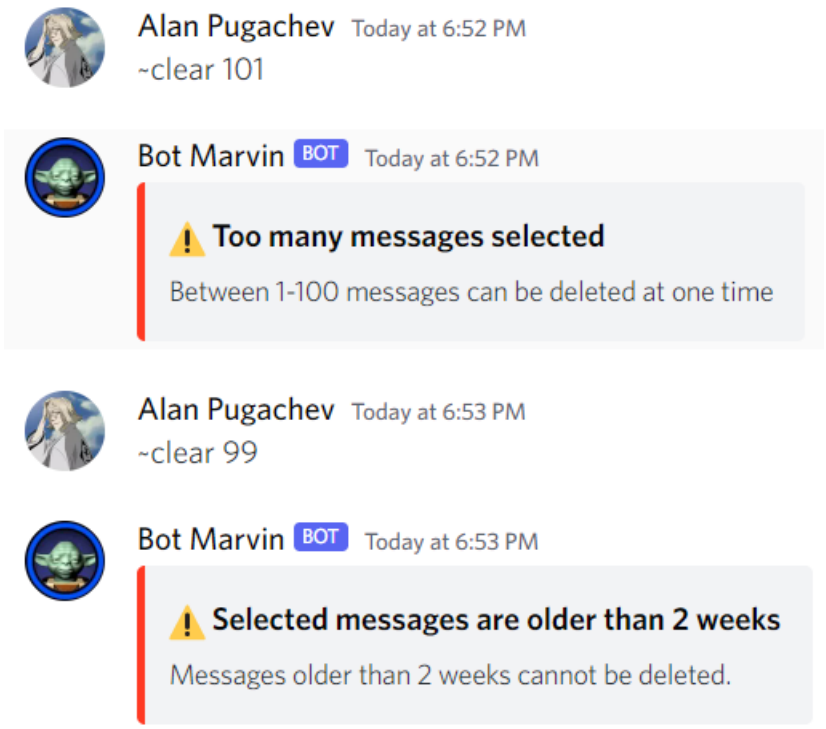

Рисунок 7 – Ошибки вызванные ограничениями бота

Также JDA предоставляет возможность выдавать роли отдельным участникам. Выдача ролей через команды является одним из основных назначений любого бота для администрирования сервера. На Рисунок 8 представлен метод-обработкчик, который по команде “mute [member]” выдаёт участнику роль (для условности была выбрана роль участника, который не может писать в текстовые чаты и говорить в голосовых). Также если в команде передать два аргумента, а именно участника и время в секнудах, то можно выдать роль участнику на определённое время.

```
//Mute command
if (args[0].equalsIgnoreCase( anotherString: App.prefix + "mute")) {
    if (args.length > 1 && args.length < 3) {
        /*Not working----------------------------------------------------------------------------------*/
        /*Member member = event.getGuild().getMemberById(args[1].replace("<@","").replace(">", ""));*/
        /*-------------------------------------------------------------------------------Not working*/

        Member member = event.getMessage().getMentionedMembers().get(0);
        Role role = event.getGuild().getRoleById("973265318036258897");

        if (!member.getRoles().contains(role)) {
            //Mute user
            event.getChannel().sendMessage( text: "Muted " + args[1] + ".").queue();
            event.getGuild().modifyMemberRoles(member, role).complete();
        }
        else {
            //Unmute user
            event.getChannel().sendMessage( text: "Unmuted " + args[1] + ".").queue();
            event.getGuild().removeRoleFromMember(member, role).complete();
        }
    }
    else if (args.length == 3) {
        Member member = event.getMessage().getMentionedMembers().get(0);
        Role role = event.getGuild().getRoleById("973265318036258897");

        event.getChannel().sendMessage( text: "Muted " + args[1] + " for " + args[2] + " seconds.").queue();
        event.getGuild().modifyMemberRoles(member, role).complete();

        //Unmute after few seconds
        new java.util.Timer().schedule(
             new java.util.TimerTask() {
                @Override
                public void run() {
                    event.getChannel().sendMessage( text: "Unmuted " + args[1] + ".").queue();
                    event.getGuild().removeRoleFromMember(member, role).complete();
                }
             },
                delay: Integer.parseInt(args[2]) * 1000
        );

    }
    else{
        event.getChannel().sendMessage( text: "Incorrect syntax. Use `~mute [member] [time (optional)]`").queue();
    }
}
```

Рисунок 8 – Метод-обработчик команды "mute"

Реализована также система оповещений о входе/выходе на сервер. Система всё также работает на обработчике событий. Метод-обработчик представлен на Рисунок 9.

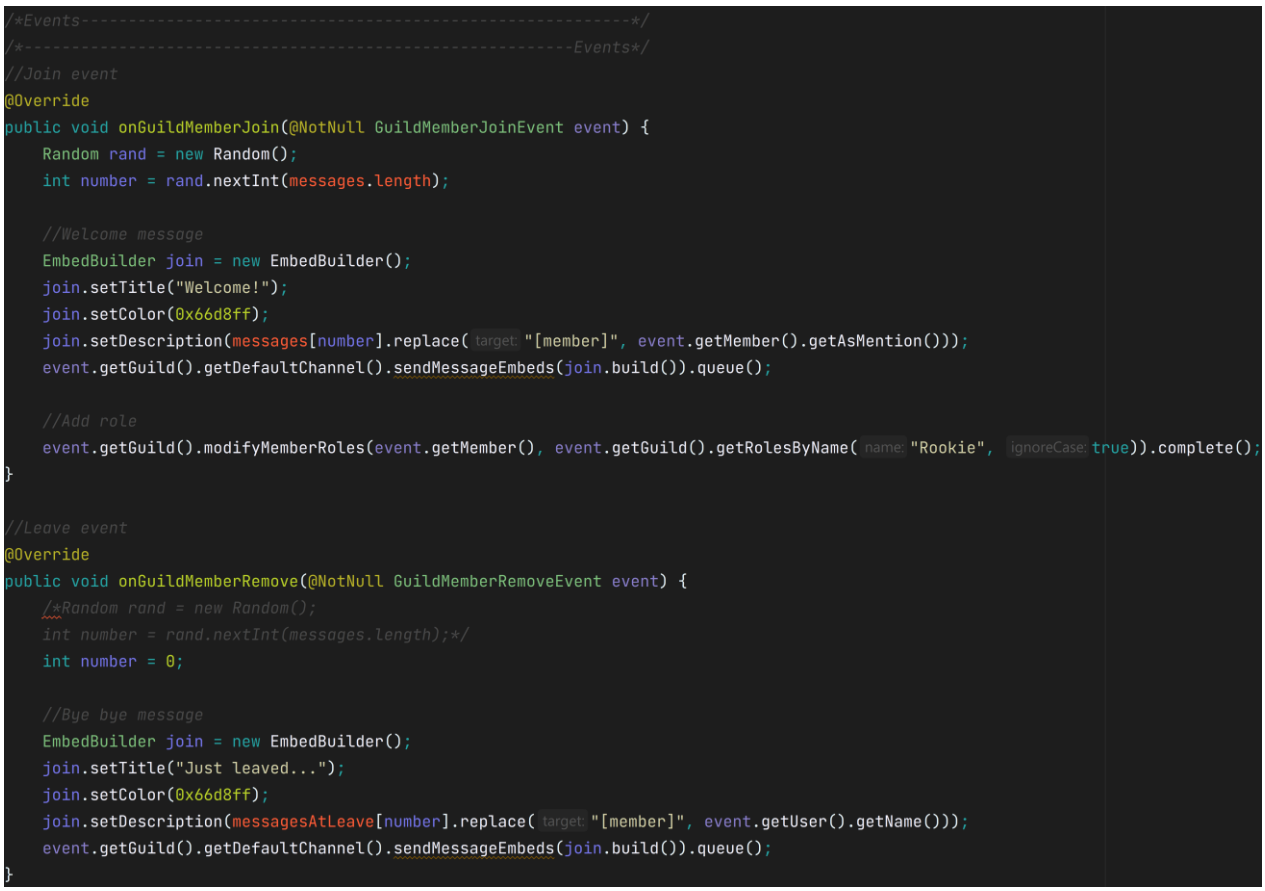

Рисунок 9 – Метод-обработчик событий входа/выхода на сервер

Была реализована система опыта пользоваетелей. Это одна из последних новинок, которая появилась в ботах. Эта система предоставляет возможность учета сообщений пользователя и увеличения уровня на сервере. С увеличением уровня, например, можно выдавать роли. Реализация системы показана на Рисунок 10.

```java
public class ExpSystem extends ListenerAdapter {

    3 usages
    HashMap<User, Integer> playerXp = new HashMap<>();
    2 usages
    HashMap<User, Integer> playerTimer = new HashMap<>();
    1 usage
    int counter = 0;

    public void onMessageReceived(MessageReceivedEvent event) {
        String[] args = event.getMessage().getContentRaw().split( regex: " ");
        String command = args[0];
        randXp(event.getMember().getUser());

        if (command.equalsIgnoreCase( anotherString: App.prefix + "xp")) {
            if (canGetXp(event.getMember().getUser())) {
                if (event.getMember().getUser() == event.getMessage().getMember().getUser()) {
                    counter++;
                    //setPlayerMessages(event.getMember().getUser(), counter);
                    randXp(event.getMember().getUser());
                    //setPlayerTime(event.getMember().getUser(), 3);
                    event.getChannel().sendMessage( text: "You have " + getPlayerXp(event.getMember().getUser()) + " xp").queue();
                }

            } else {
                event.getChannel().sendMessage( text: "You have " + getPlayerXp(event.getMember().getUser()) + " xp").queue();
            }

        }
    }

    3 usages
    private int getPlayerXp(User member) { return playerXp.get(member); }

    2 usages
    private void setPlayerXp(User member, int number) { playerXp.put(member, number); }

    private int getPlayerTime(User member) { return playerTimer.get(member); }

    2 usages
    private void randXp(User member){
        Random rand = new Random();
        if (!playerXp.containsKey(member)){
            setPlayerXp(member,  number: 0);
        }
        setPlayerXp(member,  number: getPlayerXp(member) + rand.nextInt( bound: 25));
    }

    1 usage
    private boolean canGetXp(User member){
        if(!playerTimer.containsKey(member)){
            return true;
        }
        return false;
    }
}
```

Рисунок 10 – Реализация класса ExpSystem

Опыт выдаётся за отправленные сообщения, а объём полученного опыта выбирается случайно от 1 до 25.

Результаты тестирования команд будут представленны в главе 3.

3 Discord-бот для администрирования сервера и поддержки мультимедиа файлов

Каждая из команд была протестирована на наличие ошибок и обработку ошибок. При написании программы были учтены как возможные синтаксические ошибки, так и ошибки, которые могут возникнуть из-за ограничений Discord’а.

Как описывалось в главе 2, команда “hi” отвечает пользователю приветствием. Результат представлен на Рисунок 11.

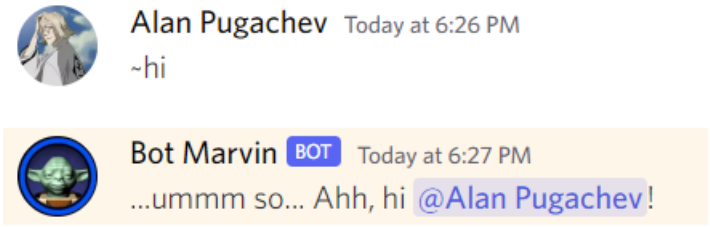

Рисунок 11 – Результат выполнения команды "hi"

Можно увидеть, что пользователь ввёл команду и бот ответил на неё пользователю (в Discord реализована система упоминаний – при упоминании пользователя, сообщение с его упоминанием подсвечивается).

Команда инфо была написана таким образом, чтобы выводить информацию про бота кратко, но не обычным сообщением, а embed’ом. Embed’ы это вид сообщений, отображение которых можно менять в коде программы – менять описание, название, добавлять любое количество полей. Далее для вывода сообщений будут использоваться именно embed’ы. На Рисунок 12 предтавлен результат работы команды “info”.

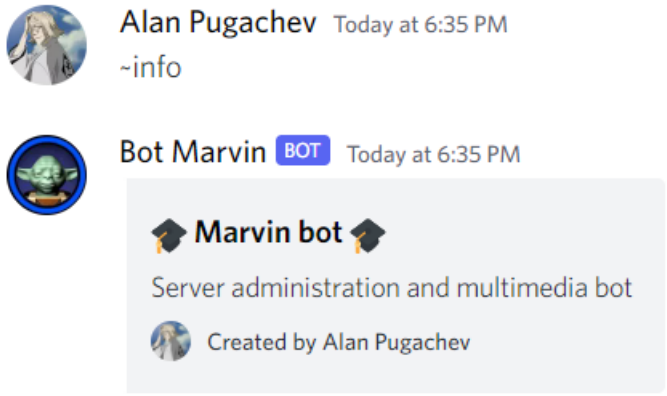

Рисунок 12 – Результат выполнения команды "info"

Видно, что сообщение структурировано, т.е. содержит название “Marvin bot”, основной текст – “Server administration and multimedia bot”, а также описание.

Основной функцией, встраиваемой в ботов является функция удаления сообщений пользователей. Ошибки выполнения команды “clear” были рассмотренны в главе 2, а результат успешного выполнения представлен на Рисунок 13.

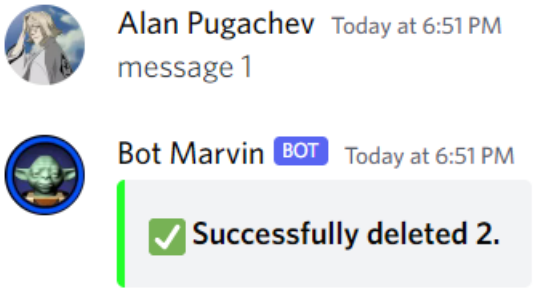

Рисунок 13 – Результат выполнения команды "clear"

При успешном выполнении также выводится сообщение о завершении операции.

Реализация команды “mute” была рассмотрена в главе 2, а речь о необходимости этой, на первый взгляд, ненужной команды, пойдёт далее. Эту команду часто встраивают в Discord ботов, т.к. с помощбю неё можно

реализовать возможность выдачи специальных ролей на более интуитивном уровне. Например, каждая онлайн-игра имеет свой собственный сервер, на котором находится огромное количество людей и соответственно серверу для модерации требуется большое количество ролей. Эти роли можно выдавать вручную через интерфейс Discord, но если их, например, больше ста, то это становится не очень удобно и приятно, проще написать команду, которая выдаст определённую роль определённому пользователю. Роль, которая будет выдаваться, назначается в коде самой программы. Программа понимает, какую роль выдать из-за Id роли, заданной в специальном методе.

Для показания результата работы бота, была взята и создана на сервере самая распостранённая роль “Muted”. При первом вызове команды, выдаётся роль выбранному пользователю, а при повторном вызове эта роль изымается. Все процессы сопровождаются соответствующими сообщениями в чате. Результат первого вызова команды представлен на Рисунок 14, а результат повторного вызова представлен на Рисунок 15.

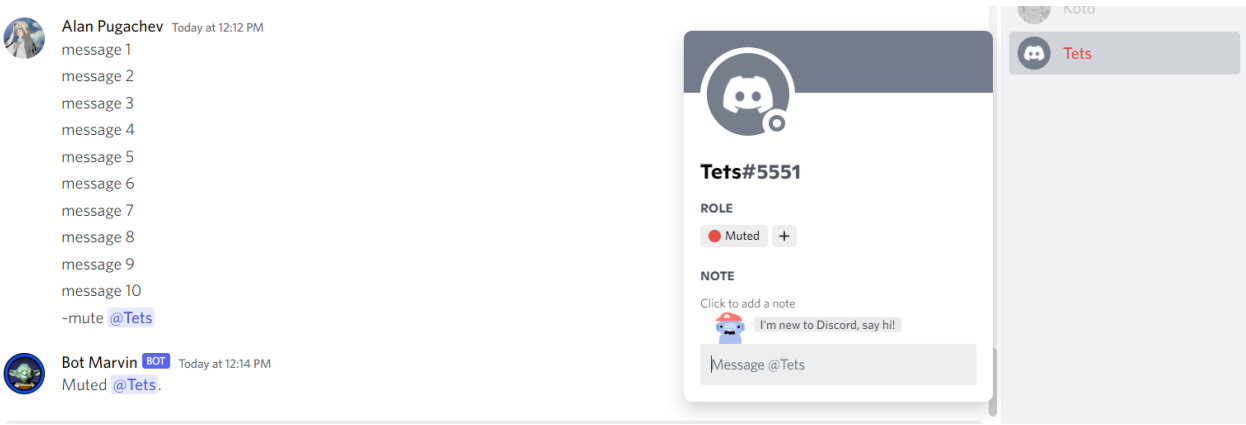

Рисунок 14 – Результат выполнения комады "mute"

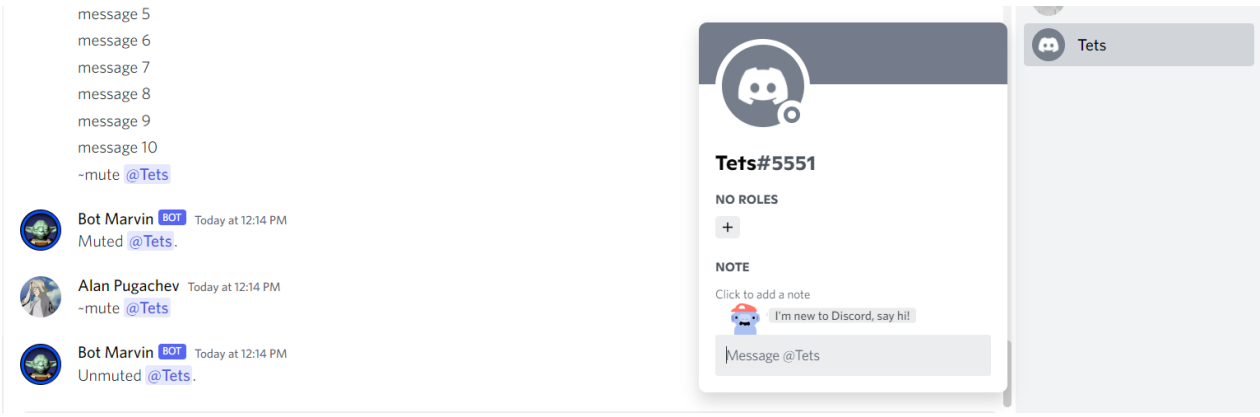

Рисунок 15 – Результат повторного вызова команды "mute"

Также частым необходимым требованием для команд такого типа является возможность выдачи роли на время. Такая возможность также была реализована и протестирована. Результаты можно увидеть на и .

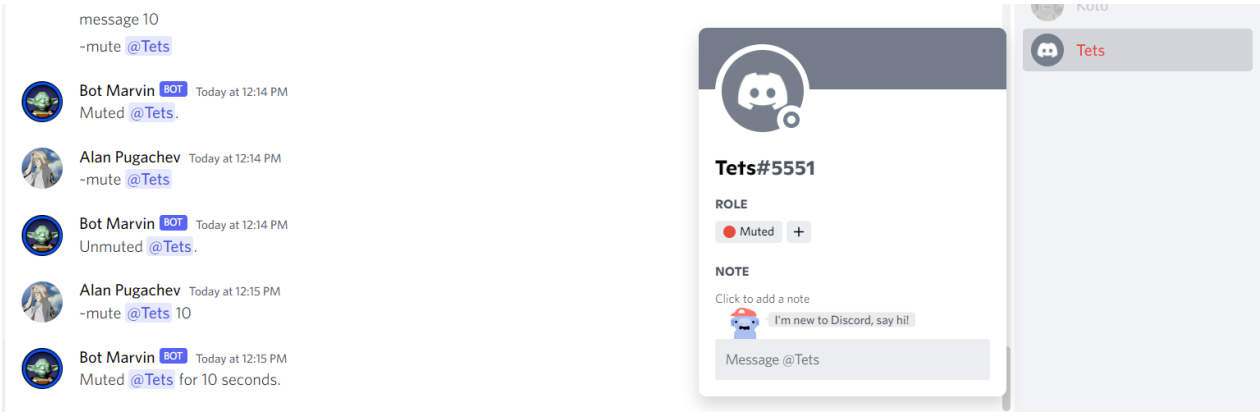

Рисунок 16 – Результат выполнения команды "mute" с секундами

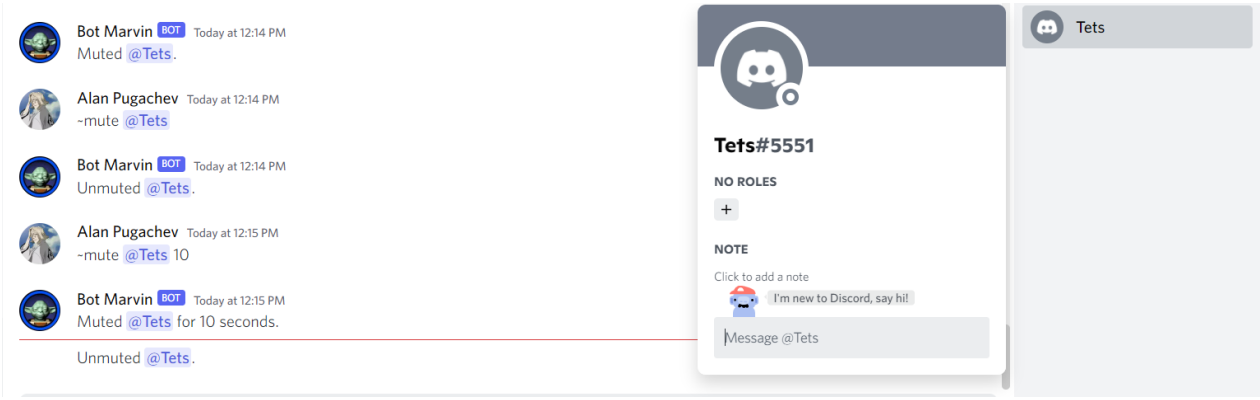

Рисунок 17 – По прошествию 10 секунд, роль была удалена

Реализация системы оповещений о входе/выходе с сервера была представлена в главе 2. Ниже представлен результат действия этой системы.

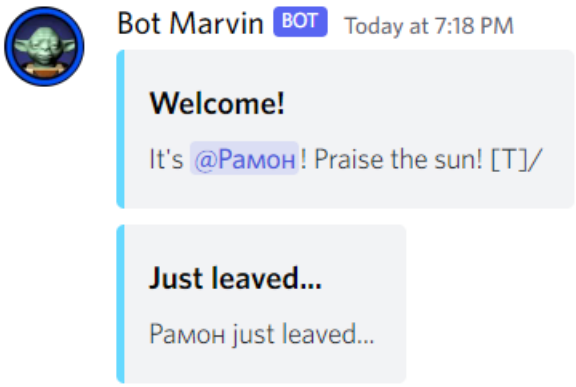

Рисунок 18 – Результат работы системы оповещения

На Рисунок 18 видно, что при первом входе на сервер, пользователю пишется приветствие, а при выходе с сервера в чат пишется имя пользователя, который покинул сервер.

Система опыта, как упоминалось в главе 2, тоже является частью базового функционала современного бота. Ниже представлен результат работы этой системы в боте “Marvin”.

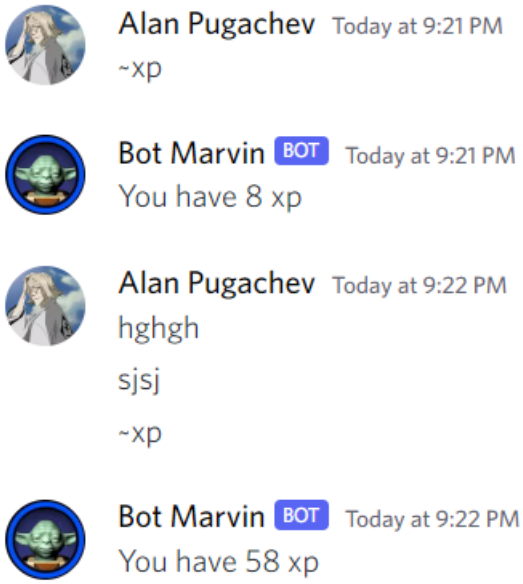

Рисунок 19 – Результат работы системы опыта

На Рисунок 19 видно, что при первом запросе пользователя его опыт составлял 8 единиц, а после отправки нескольких сообщений его опыт увеличился до 58 единиц. Опыт, в большинстве случаев, начисляется за количество сообщений на сервере, которые отправил пользователь, т.к. невозможно отследить, например, время нахождения пользователя в голосовом чате, а расчёт времени нахождения пользователя на сервере занимается очень много памяти на хосте, котором запущен бот.

Также ниже представлен результат работы LavaPlayer, который включен в функционал Marvin’a, но является плагином.

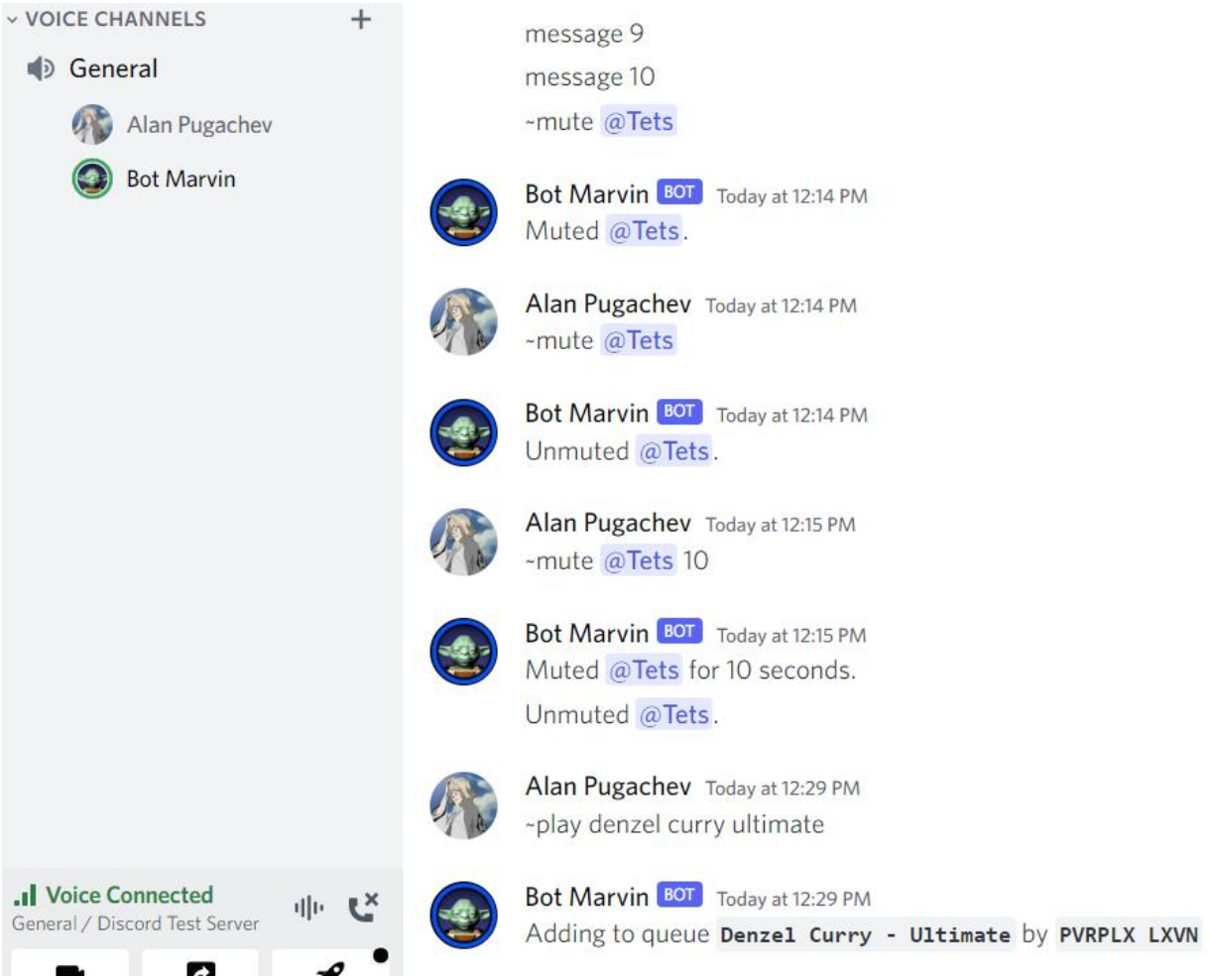

Рисунок 20 – Результат работы плагина LavaPlayer

На Рисунок 20 видно, что пользователь сделал запрос с помощью команды, а бот присоеденился к голосовому чату, в котором находится

пользователь, чей запрос обработал бот, и начал воспроизведение мультимедиа файла, который был найден по запросу пользователя. Воспроизведение можно отследить по подсветке иконки бота в голосовом чате – Discord подсвечивает пользователей, звук от которых транслируется в голосовой чат.

ЗАКЛЮЧЕНИЕ

В современном мире сложно представить жизнь без мессенджеров. Каждый разработчик пытается добавить как можно больше функций в свой мессенджер, но всем удовлетворить нельзя. Отсюда возникает потребность в расширении интрументария. Для этого разработчики публикуют свой код в открытый доступ, чтобы пользователи сами расширяли и добавляли нужные им возможности.

Цель курсовой работы, а именно разработка Discord-бота достигнута. Для этого был изучен фреймворк JDA и реализован функционал на основе этого фреймворка. В боте исключена возможность возникновения ошибки и бота можно использовать на больших серверах, не переживая за его стабильность.

Главная задача, а именно расширение функционала бота, была реализована и протестирована на примерах, представленных в главе 3. Бот “Marvin” воплащает в себе весь базовый функционал имеющийся в большинстве современных ботов, без которых существование Discord-серверов сложно себе представить. Бот имеет большой потенциал расширения функционала, благодаря фреймворку JDA, который позволяет получить доступ к любой информации, но наличие функционала бота уже покрывает потребности большинства серверов. В Marvine’е не реализованы специфичные функции, например, поиск в интернете какой-то информации, которая задаётся непосредственно в Discord’е, т.к. они бесполезны, вряд ли пользователи будут ими пользоваться, а веб-браузеры справляются с этим намного лучше.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Discord Revenue and Usage Statistics (2022). URL: https://www.businessofapps.com/data/discord-statistics/ (дата обращения 29.04.2022)

2. Discord Blog. URL: https://discord.com/blog/ (дата обращения 29.04.2022)

3. Voice Chats Done Right. URL: https://telegram.org/blog/voice-chats/ (дата обращения 30.04.2022)

4. Why Is Skype Going Away?| Future of Skype. URL: https://www.eltima.com/is-skype-going-away/ (дата посещения 02.05.2022)

5. Tencent – Official Site. URL: https://www.tencent.com/en-us/ (дата обращения 02.05.2022)

6. Discord – Official Site. URL: https://discord.com/ (дата посещения 30.04.2022)

7. Krisp – Noise Cancellation and Echo Removal. URL: https://krisp.ai/ (дата обращения 30.04.2022)

8. Discord Developer Portal. URL: https://discord.com/developers/docs/topics/community-resources/ (дата посещения 05.05.2022)

9. DV8FromTheWorld/JDA. URL: https://github.com/DV8FromTheWorld/JDA/ (дата посещения 05.05.2022)

10. Sedmelluq/lavaplayer. URL: https://github.com/sedmelluq/lavaplayer/ (дата посещения 06.05.2022)

ПРИЛОЖЕНИЕ А

Код классов приложения

```
import MusicPlayer.PlayCommand;
import ca.tristan.jdacommands.JDACommands;
import ca.tristan.jdacommands.ICommand;
import net.dv8tion.jda.api.JDA;
import net.dv8tion.jda.api.JDABuilder;
import net.dv8tion.jda.api.requests.GatewayIntent;
import net.dv8tion.jda.api.utils.cache.CacheFlag;

import javax.security.auth.login.LoginException;

public class App {
    public static JDA jda;
    public static String prefix = "~";

    //Main method
    public static void main(String[] args) throws LoginException {
        JDACommands jdaCommands = new JDACommands("~");
        jdaCommands.registerCommand(new PlayCommand());

        jda = JDABuilder.createDefault(

"OTU2MjI2MjkwODMzMzIyMDg0.YjtJIQ.8buaF7ZbC3zmxZK7X_LIq1lb_bA",
                GatewayIntent.GUILD_MEMBERS,
                GatewayIntent.GUILD_MESSAGES,
                GatewayIntent.GUILD_VOICE_STATES
        ).enableCache(CacheFlag.VOICE_STATE).build();
        jda.addEventListener(new Commands());
        jda.addEventListener(new Events());
        jda.addEventListener(new ExpSystem());
        jda.addEventListener(jdaCommands);
    }
}

import net.dv8tion.jda.api.EmbedBuilder;
import net.dv8tion.jda.api.entities.Member;
import net.dv8tion.jda.api.entities.Message;
import net.dv8tion.jda.api.entities.Role;
import net.dv8tion.jda.api.events.message.MessageReceivedEvent;
import net.dv8tion.jda.api.hooks.ListenerAdapter;

import java.util.List;
import java.util.Random;

public class Commands extends ListenerAdapter {
    public void onMessageReceived(MessageReceivedEvent event) {
        String[] args =
event.getMessage().getContentRaw().split("\\s+");

        //Command that's just send "hi" message
        if (args[0].equalsIgnoreCase(App.prefix + "hi")) {
            String[] hiMessages = {
                    "Hi there, [member] :fire:",
```

```
                    "Oh hello there :sunglasses:",
                    "...ummm so... Ahh, hi [member]!"
            };
            Random rand = new Random();
            int number = rand.nextInt(hiMessages.length);

            event.getChannel().sendTyping().queue();

event.getChannel().sendMessage(hiMessages[number].replace("[member]",
event.getMember().getAsMention())).queue();
        }

        //Information about bot
        if (args[0].equalsIgnoreCase(App.prefix + "info")) {
            EmbedBuilder info = new EmbedBuilder();
            info.setTitle(":mortar_board: Marvin bot :mortar_board:");
            info.setDescription("Server administration and multimedia
bot");
            info.setColor(0xffffff);
            info.setFooter("Created by Alan Pugachev",
event.getMember().getUser().getAvatarUrl());
            event.getChannel().sendMessageEmbeds(info.build()).queue();
            info.clear();
        }

        //Command that's clear chat
        if (args[0].equalsIgnoreCase(App.prefix + "clear")) {
            if (args.length != 2) {
                // Wrong usage
                EmbedBuilder usage = new EmbedBuilder();
                usage.setColor(0xff3923);
                usage.setTitle(":warning: Specify amount to delete");
                usage.setDescription("Usage: '" + App.prefix + "clear
[amount of messages]'");

event.getChannel().sendMessageEmbeds(usage.build()).queue();
                usage.clear();
            }
            else {
                try {
                    List<Message> messages =
event.getChannel().getHistory().retrievePast(Integer.parseInt(args[1])
+ 1).complete();

event.getTextChannel().deleteMessages(messages).queue();

                    // Success
                    EmbedBuilder success=new EmbedBuilder();
                    success.setColor(0x22ff2a);
                    success.setTitle(":white_check_mark: Successfully
deleted " + args[1] + ".");

event.getChannel().sendMessageEmbeds(success.build()).queue();
                }
                catch (IllegalArgumentException e) {
                    //e.printStackTrace();
```

```
                    if
(e.toString().startsWith("java.lang.IllegalArgumentException: Message
retrieval")) {
                        //Too many messages
                        EmbedBuilder error = new EmbedBuilder();
                        error.setColor(0xff3923);
                        error.setTitle(":warning: Too many messages
selected");
                        error.setDescription("Between 1-100 messages
can be deleted at one time");

event.getChannel().sendMessageEmbeds(error.build()).queue();
                        error.clear();
                    }
                    else {
                        //Messages too old
                        EmbedBuilder error = new EmbedBuilder();
                        error.setColor(0xff3923);
                        error.setTitle(":warning: Selected messages are
older than 2 weeks");
                        error.setDescription("Messages older than 2
weeks cannot be deleted.");

event.getChannel().sendMessageEmbeds(error.build()).queue();
                        error.clear();
                    }
                }
            }
        }

        //Mute command
        if (args[0].equalsIgnoreCase(App.prefix + "mute")) {
            if (args.length > 1 && args.length < 3) {
                /*Not working----------------------------------------
-----------------------------------*/
                /*Member member =
event.getGuild().getMemberById(args[1].replace("<@","").replace(">",
""));*/
                /*-------------------------------------------------------
-------------------------Not working*/

                Member member =
event.getMessage().getMentionedMembers().get(0);
                Role role =
event.getGuild().getRoleById("973265318036258897");

                if (!member.getRoles().contains(role)) {
                    //Mute user
                    event.getChannel().sendMessage("Muted " + args[1] +
".").queue();
                    event.getGuild().modifyMemberRoles(member,
role).complete();
                }
                else {
                    //Unmute user
                    event.getChannel().sendMessage("Unmuted " + args[1]
+ ".").queue();
```

```
                    event.getGuild().removeRoleFromMember(member,
role).complete();
                }
            }
            else if (args.length == 3) {
                Member member =
event.getMessage().getMentionedMembers().get(0);
                Role role =
event.getGuild().getRoleById("973265318036258897");

                event.getChannel().sendMessage("Muted " + args[1] + "
for " + args[2] + " seconds.").queue();
                event.getGuild().modifyMemberRoles(member,
role).complete();

                //Unmute after few seconds
                new java.util.Timer().schedule(
                     new java.util.TimerTask() {
                         @Override
                         public void run() {
                             event.getChannel().sendMessage("Unmuted "
+ args[1] + ".").queue();

event.getGuild().removeRoleFromMember(member, role).complete();
                         }
                     },
                        Integer.parseInt(args[2]) * 1000
                );

            }
            else{
                event.getChannel().sendMessage("Incorrect syntax. Use
`~mute [member] [time (optional)]`").queue();
            }
        }
    }
}

import net.dv8tion.jda.api.EmbedBuilder;
import net.dv8tion.jda.api.events.guild.member.GuildMemberJoinEvent;
import net.dv8tion.jda.api.events.guild.member.GuildMemberRemoveEvent;
import net.dv8tion.jda.api.events.message.MessageReceivedEvent;
import net.dv8tion.jda.api.hooks.ListenerAdapter;
import net.dv8tion.jda.api.entities.Member;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.*;

public class Events extends ListenerAdapter {
    //List of join messages
    String[] messages = {
            "[member] joined. You must construct addition pylons!",
            "Never gonna give [member] up! Never gonna let [member]
down!",
            "Hey! Listen! [member] has joined!",
            "Ha! [member] has joined! You activated my trap card!",
            "We've been expecting you, [member].",
```

```
            "It's dangerous to go alone, take [member]!",
            "Swoooosh. [member] just landed.",
            "Brace yourselves. [member] just joined the server.",
            "A wild [member] appeared.",
            "[member] just slid into the server!",
            "Ermagherd. [member] is here.",
            "[member] joined your party.",
            "[member] just joined the server. - glhf",
            "[member] just joined. Everyone, look busy!",
            "[member]  just joined. Can I get a heal?",
            "Welcome, [member]. Stay awhile and listen",
            "Welcome, [member]. Leave your weapons by the door.",
            "Welcome, [member]. We hope you brought pizza.",
            "Brace yourselves. [member] just joined the server.",
            "[member] just joined. Hide your bananas.",
            "[member] just arrived. Seems OP - please nerf.",
            "A [member] has spawned in the server.",
            "Big [member] showed up!",
            "Where’s [member]? In the server!",
            "[member] hopped into the server. Kangaroo!!",
            "[member] just showed up. Hold my beer.",
            "Challenger approaching - [member] has appeared!",
            "It's a bird! It's a plane! Nevermind, it's just
[member].",
            "It's [member]! Praise the sun! [T]/",
            "Roses are red, violets are blue, [member] joined this
server with you",
            "Hello. Is it [member] you're looking for?",
            "[member] is here to kick butt and chew bubblegum. And
[member] is all out of gum.",
            "[member] has arrived. Party's over.",
            "Ready player [member]",
            "Look! [member] has OP items! Oh no RUN!",
            "Look! It's my mom! oh wait its my dad! oh nvm its just
[member] chillin",
            "[member] have fun ok bye bye",
            "Hey! We have a new pogger in town his name is called
[member]",
            "[member] is finally here lets get this party started!",
            "[member] is here! Play minecraft NOW!"
    };

    //List of leave messages
    String[] messagesAtLeave = {
            "[member] just leaved..."
    };



    /*Events------------------------------------------------------------
*/
    /*----------------------------------------------------------
Events*/
    //Join event
    @Override
    public void onGuildMemberJoin(@NotNull GuildMemberJoinEvent event)
{
        Random rand = new Random();
```

```
        int number = rand.nextInt(messages.length);

        //Welcome message
        EmbedBuilder join = new EmbedBuilder();
        join.setTitle("Welcome!");
        join.setColor(0x66d8ff);
        join.setDescription(messages[number].replace("[member]",
event.getMember().getAsMention()));

event.getGuild().getDefaultChannel().sendMessageEmbeds(join.build()).qu
eue();

        //Add role
        event.getGuild().modifyMemberRoles(event.getMember(),
event.getGuild().getRolesByName("Rookie", true)).complete();
    }

    //Leave event
    @Override
    public void onGuildMemberRemove(@NotNull GuildMemberRemoveEvent
event) {
        /*Random rand = new Random();
        int number = rand.nextInt(messages.length);*/
        int number = 0;

        //Bye bye message
        EmbedBuilder join = new EmbedBuilder();
        join.setTitle("Just leaved...");
        join.setColor(0x66d8ff);
        join.setDescription(messagesAtLeave[number].replace("[member]",
event.getUser().getName()));

event.getGuild().getDefaultChannel().sendMessageEmbeds(join.build()).qu
eue();
    }
}

import net.dv8tion.jda.api.entities.User;
import net.dv8tion.jda.api.events.message.MessageReceivedEvent;
import net.dv8tion.jda.api.hooks.ListenerAdapter;

import java.util.HashMap;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;

public class ExpSystem extends ListenerAdapter {

    HashMap<User, Integer> playerXp = new HashMap<>();
    HashMap<User, Integer> playerTimer = new HashMap<>();
    int counter = 0;

    public void onMessageReceived(MessageReceivedEvent event) {
        String[] args = event.getMessage().getContentRaw().split(" ");
        String command = args[0];
        randXp(event.getMember().getUser());

        if (command.equalsIgnoreCase(App.prefix + "xp")) {
```

```
            if (canGetXp(event.getMember().getUser())) {
                if (event.getMember().getUser() ==
event.getMessage().getMember().getUser()) {
                    counter++;
                    //setPlayerMessages(event.getMember().getUser(),
counter);
                    randXp(event.getMember().getUser());
                    //setPlayerTime(event.getMember().getUser(), 3);
                    event.getChannel().sendMessage("You have " +
getPlayerXp(event.getMember().getUser()) + " xp").queue();
                }

            } else {
                event.getChannel().sendMessage("You have " +
getPlayerXp(event.getMember().getUser()) + " xp").queue();
            }

        }
    }

    private int getPlayerXp(User member){
        return playerXp.get(member);
    }

    private void setPlayerXp(User member, int number){
        playerXp.put(member, number);
    }

    private int getPlayerTime(User member){
        return playerTimer.get(member);
    }

    private void randXp(User member){
        Random rand = new Random();
        if (!playerXp.containsKey(member)){
            setPlayerXp(member, 0);
        }
        setPlayerXp(member, getPlayerXp(member) + rand.nextInt(25));
    }

    private boolean canGetXp(User member){
        if(!playerTimer.containsKey(member)){
            return true;
        }
        return false;
    }
}

package MusicPlayer;

import com.sedmelluq.discord.lavaplayer.player.AudioPlayer;
import
com.sedmelluq.discord.lavaplayer.track.playback.MutableAudioFrame;
import net.dv8tion.jda.api.audio.AudioSendHandler;
import org.jetbrains.annotations.Nullable;

import java.nio.Buffer;
import java.nio.ByteBuffer;
```

```
public class AudioPlayerSendHandler implements AudioSendHandler {
    private final AudioPlayer audioPlayer;
    private final ByteBuffer buffer;
    private final MutableAudioFrame frame;

    public AudioPlayerSendHandler (AudioPlayer audioPlayer) {
        this.audioPlayer = audioPlayer;
        this.buffer = ByteBuffer.allocate(1024);
        this.frame = new MutableAudioFrame();
        this.frame.setBuffer(buffer);
    }

    @Override
    public boolean canProvide() {
        return this.audioPlayer.provide(this.frame);
    }

    @Nullable
    @Override
    public ByteBuffer provide20MsAudio() {
        final Buffer buffer = ((Buffer) this.buffer).flip();
        return (ByteBuffer) buffer;
    }

    @Override
    public boolean isOpus() {
        return true;
    }
}
```

```
package MusicPlayer;

import com.sedmelluq.discord.lavaplayer.player.AudioPlayer;
import com.sedmelluq.discord.lavaplayer.player.AudioPlayerManager;

public class GuildMusicManager {
    public final AudioPlayer audioPlayer;
    public final TrackScheduler scheduler;
    private final AudioPlayerSendHandler sendHandler;

    public GuildMusicManager(AudioPlayerManager manager) {
        this.audioPlayer = manager.createPlayer ();
        this.scheduler = new TrackScheduler(this.audioPlayer);
        this.audioPlayer.addListener(this.scheduler);
        this. sendHandler = new
AudioPlayerSendHandler(this.audioPlayer);
    }

    public AudioPlayerSendHandler getSendHandler() {
        return this.sendHandler;
    }
}
```

```
package MusicPlayer;

import ca.tristan.jdacommands.ExecuteArgs;
import ca.tristan.jdacommands.ICommand;
```

```
import net.dv8tion.jda.api.entities.VoiceChannel;
import net.dv8tion.jda.api.managers.AudioManager;

import java.net.URI;
import java.net.URISyntaxException;

public class PlayCommand implements ICommand {
    @Override
    public void execute(ExecuteArgs event) {
        if(!event.getMemberVoiceState().inAudioChannel()) {
            event.getTextChannel().sendMessage("You need to be in a
voice channel for this command to work.").queue();
            return;
        }

        if(!event.getSelfVoiceState().inAudioChannel()) {
            final AudioManager audioManager =
event.getGuild().getAudioManager();
            final VoiceChannel memberChannel = (VoiceChannel)
event.getMemberVoiceState().getChannel();

            audioManager.openAudioConnection(memberChannel);
        }

        String link = String.join(" ", event.getArgs());

        if(!isUrl(link)) {
            link = "ytsearch:" + link + " audio";
        }

        PlayerManager.getINSTANCE().loadAndPlay(event.getTextChannel(),
link);
    }

    public boolean isUrl(String url) {
        try {
            new URI(url);
            return true;
        }
        catch (URISyntaxException e) {
            return false;
        }
    }

    @Override
    public String getName() {
        return "play";
    }

    @Override
    public String helpMessage() {
        return null;
    }

    @Override
    public boolean needOwner() {
        return false;
    }
```

```
}
```

```
package MusicPlayer;

import com.sedmelluq.discord.lavaplayer.player.AudioLoadResultHandler;
import com.sedmelluq.discord.lavaplayer.player.AudioPlayerManager;
import
com.sedmelluq.discord.lavaplayer.player.DefaultAudioPlayerManager;
import com.sedmelluq.discord.lavaplayer.source.AudioSourceManagers;
import com.sedmelluq.discord.lavaplayer.tools.FriendlyException;
import com.sedmelluq.discord.lavaplayer.track.AudioPlaylist;
import com.sedmelluq.discord.lavaplayer.track.AudioTrack;
import net.dv8tion.jda.api.entities.Guild;
import net.dv8tion.jda.api.entities.TextChannel;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class PlayerManager {
    private static PlayerManager INSTANCE;
    private final Map<Long, GuildMusicManager> musicManagers;
    private final AudioPlayerManager audioPlayerManager;

    public PlayerManager() {
        this.musicManagers = new HashMap();
        this.audioPlayerManager=new DefaultAudioPlayerManager();
        AudioSourceManagers.registerRemoteSources
(this.audioPlayerManager);

AudioSourceManagers.registerLocalSource(this.audioPlayerManager);
    }

    public GuildMusicManager getMusicManager(Guild guild) {
        return this.musicManagers.computeIfAbsent(guild.getIdLong(),
(guildId) -> {
            final GuildMusicManager guildMusicManager = new
GuildMusicManager(this.audioPlayerManager);

guild.getAudioManager().setSendingHandler(guildMusicManager.getSendHand
ler());
            return guildMusicManager;
        });
    }

    public void loadAndPlay(TextChannel textChannel, String trackURL) {
        final GuildMusicManager musicManager =
this.getMusicManager(textChannel.getGuild());

        this.audioPlayerManager.loadItemOrdered(musicManager, trackURL,
new AudioLoadResultHandler() {
            @Override
            public void trackLoaded(AudioTrack audioTrack) {
                musicManager.scheduler.queue(audioTrack);

                textChannel.sendMessage("Adding to queue **`")
                        .append(audioTrack.getInfo().title)
                        .append("`** by **`")
```

```
                        .append(audioTrack.getInfo().author)
                        .append("`**")
                        .queue();
            }

            @Override
            public void playlistLoaded(AudioPlaylist audioPlaylist) {
                final List<AudioTrack> tracks =
audioPlaylist.getTracks();
                if (!tracks.isEmpty()) {
                    musicManager.scheduler.queue(tracks.get(0));
                    textChannel.sendMessage("Adding to queue **`")
                            .append(tracks.get(0).getInfo().title)
                            .append("`** by **`")
                            .append(tracks.get(0).getInfo().author)
                            .append("`**")
                            .queue();
                }
            }

            @Override
            public void noMatches() {

            }

            @Override
            public void loadFailed(FriendlyException e) {

            }
        });
    }

    public static PlayerManager getINSTANCE() {
        if(INSTANCE == null) {
            INSTANCE = new PlayerManager();
        }
        return INSTANCE;
    }
}
```

```
package MusicPlayer;

import com.sedmelluq.discord.lavaplayer.player.AudioPlayer;
import com.sedmelluq.discord.lavaplayer.player.event.AudioEventAdapter;
import com.sedmelluq.discord.lavaplayer.track.AudioTrack;
import com.sedmelluq.discord.lavaplayer.track.AudioTrackEndReason;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class TrackScheduler extends AudioEventAdapter {
    public final AudioPlayer audioPlayer;
    public final BlockingQueue<AudioTrack> queue;

    public TrackScheduler(AudioPlayer audioPlayer) {
        this.audioPlayer = audioPlayer;
        this.queue = new LinkedBlockingQueue();
    }
```

```
    public void queue(AudioTrack track) {
        if (!this.audioPlayer.startTrack(track, true)) {
            this.queue.offer(track);
        }
    }

    public void nextTrack() {
        this.audioPlayer.startTrack(this.queue.poll(), false);
    }

    @Override
    public void onTrackEnd(AudioPlayer player, AudioTrack track,
AudioTrackEndReason endReason) {
        if (endReason.mayStartNext) {
            nextTrack();
        }
    }
}
```