СОДЕРЖАНИЕ

1. ОБЩЕЕ ЗНАКОМСТВО С БАЗОЙ ПРАКТИКИ

Производственная практика проходила в ГКУ РК "Государственный Архив Республики Крым". Государственный архив располагается в г. Симферополь на улице Кечкеметская, 3 либо на улице Павленко, 1-а.

Архив создан в соответствии распоряжением Совета министров Республики Крым от 21 октября 2014 года №1064-р «О создании Государственного казенного учреждения Республики Крым «Государственный архив Республики Крым по личному составу» Юридическую самостоятельность архив получил с 16 февраля 2015 г.

Основным направлением деятельности государственного казенного учреждения Республики Крым «Государственный архив Республики Крым по личному составу» является: комплектование, обеспечение сохранности, государственного учета и использования документов по личному составу и других архивных документов ликвидированных органов государственной власти Республики Крым, республиканских учреждений, организаций и предприятий, а также исполнение запросов граждан, учреждений, организаций, предприятий социально-правового характера, связанных с обеспечением законных прав и интересов заявителей.

Архив хранит документы по личному составу (приказы по личному составу, лицевые счета и расчетные ведомости по начислению заработной платы работникам, личные дела, трудовые книжки, и др.) ликвидированных организаций всех форм собственности, расположенных на территории Республики Крым.

Доступ и использование документов осуществляется в соответствии с требованиями законодательства.

По вопросам приема и упорядочения документов по личному составу, подлежащих передаче на государственное хранение, сотрудниками архива оказывается методическая помощь ответственным за архивы организаций.[1]

Рисунок 1 – Государственный архив РК

2. ОЗНАКОМЛЕНИЕ С ЗАКОНОДАТЕЛЬНЫМИ И ИНЫМИ НОРМАТИВНЫМИ ПРАВОВЫМИ АКТАМИ, РЕГУЛИРУЮЩИМИ ДЕЯТЕЛЬНОСТЬ ХОЗЯЙСТВУЮЩЕГО СУБЪЕКТА

База производственной практики относится к категории разработки программных продуктов и предоставляет коммерческие услуги, вследствие чего к ней относятся следующие нормативно-правовые акты: [2]

Уголовный кодекс Российской Федерации от 13 июня 1996 г. № 63-ФЗ (Глава 28 «Преступления в сфере компьютерной информации»)

Федеральный закон от 23 августа 1996 г. № 127-ФЗ «О науке и государственной научно-технической политике»

Федеральный закон от 6 апреля 2011 г. № 63-ФЗ «Об электронной подписи»

Федеральный закон от 7 июля 2003 г. № 126-ФЗ «О связи»

Федеральный закон от 29 июля 2004 г. № 98-ФЗ «О коммерческой тайне»

Федеральный закон от 27 июля 2006 г. № 149-ФЗ «Об информации, информационных технологиях и о защите информации»

Федеральный закон от 9 февраля 2009 № 8-ФЗ «Об обеспечении доступа к информации о деятельности государственных органов и органов местного самоуправления»

Федеральный закон от 28 декабря 2010 г. № 390-ФЗ «О безопасности»

Указ Президента РФ от 20 января 1994 г. № 170 «Об основах государственной политики в сфере информатизации»

Указ Президента РФ от 31 декабря 2015 г. № 683 «О Стратегии национальной безопасности Российской Федерации»

Указ Президента РФ от 17 марта 2008 г. № 351 «О мерах по обеспечению информационной безопасности Российской Федерации при использовании информационно-телекоммуникационных сетей международного информационного обмена»;

Постановление Правительства РФ от 26 июня 1995 г. № 608 «О сертификации средств защиты информации»

Постановление Правительства Российской Федерации от 22 октября 2007 г. № 689 «Об утверждении Положения о лицензировании деятельности по выявлению электронных устройств, предназначенных для негласного получения информации, в помещениях и технических средствах (за исключением случая, если указанная деятельность осуществляется для обеспечения собственных нужд юридического лица или индивидуального предпринимателя)»

Постановление Правительства РФ от 6 ноября 2007 года № 758 «О государственной аккредитации организаций, осуществляющих деятельность в области информационных технологий»

Постановление Правительства Российской Федерации от 25 декабря 2007 г. № 931 «О некоторых мерах по обеспечению информационного взаимодействия государственных органов и органов местного самоуправления при оказании государственных услуг гражданам и организациям»

Постановление Правительства Российской Федерации от 29 декабря 2007 г. № 947 «Об утверждении Правил разработки, апробации, доработки и реализации типовых программно-технических решений в сфере региональной информатизации»

3. РАЗРАБОТКА СПЕЦИФИКАЦИЙ ОТДЕЛЬНЫХ КОМПОНЕНТОВ

При разработке программного обеспечения каждой организации, или отдельным специалистам необходимо принять один или несколько шаблонов спецификации требований к ПО, относящихся к стандартным, для использования в своих работах. Данных спецификаций существует достаточно много, при этом они достаточно различны по своей сути. Каждый шаблон спецификации предназначен для отдельного типа и размера программного обеспечения, будь то глобальное обновление на существующую систему, меняющее множество корневых механик, или же создание небольшого приложения для облегчения работы с конкретной областью деятельности.

В данной работе используется ГОСТ 19.202-78.

Приведенный шаблон подходит для многих проектов. Состоит он из следующих разделов:

1. Введение
1.1 Назначение
1.2 Соглашения, принятые в документах
1.3 Границы проекта
1.4 Ссылки
2. Общее описание
2.1 Общий взгляд на продукт
2.2 Классы и характеристики пользователей
2.3 Операционная среда
2.4 Ограничения дизайна и реализации
2.5 Предположения и зависимости
3. Функции системы
3.x Функция системы X
3.1.1 Описание
3.1.2 Функциональные требования
4. Требования к данным
4.1 Логическая модель данных
4.2 Словарь данных
4.3 Отчеты
4.4 Получение, целостность, хранение и утилизация данных
5. Требования к внешним интерфейсам
5.1 Пользовательские интерфейсы
5.2 Интерфейсы ПО
5.3 Интерфейсы оборудования
5.4 Коммуникационные интерфейсы
6. Атрибуты качества
6.1 Удобство использования
6.2 Производительность
6.3 Безопасность
6.4 Техника безопасности
6.x [Другие]
7. Требования по интернационализации и локализации
8. Остальные требования

Приложение А. Словарь терминов Приложение
Б. Модели анализа [3].

Данное приложение проектировалось в среде разработке PyCharm, сам процесс состоял из установки необходимого ПО и прописания самих команд на языке Python.

Здесь изображена команда help. При ее вызове будет появляться список команд, которые можно задать боту.

```python
@client.command(pass_context = True)
async def help(ctx):
    prefix = '!'
    emb = discord.Embed(title = '**Навигация по командам**')

    emb.add_field (name = '{}join'.format(prefix), value = 'Команда, чтобы бот зашел в ваш голосовой канал')
    emb.add_field (name = '{}leave'.format(prefix), value = 'Команда, чтобы бот покинул ваш голосовой канал')

    emb.add_field (name = '{}play'.format(prefix), value = 'Эта команда воспроизводит музыку')
    emb.add_field (name = '{}stop'.format(prefix), value = 'Эта команда пропускает песню, которую ты сейчас слушаешь')
    emb.add_field (name = '{}pause'.format(prefix), value = 'Команда для остановки воспроизведния песни')
    emb.add_field (name = '{}resume'.format(prefix), value = 'Команда для продолжения воспроизводения песни')
    emb.add_field (name = '{}queue'.format(prefix), value = 'Команда для добавлении песни в очередь')
    emb.add_field (name = '{}loopmode'.format(prefix), value = 'Команда зацикливает песни в очереди')
    emb.add_field (name = '{}view'.format(prefix), value = 'Команда для просмотра списка очереди')
    emb.add_field (name = '{}remove'.format(prefix), value = 'Эта команда удаляет песню из очереди')

    emb.add_field (name = '{}hello'.format(prefix), value = 'Команда для общения')
    emb.add_field (name = '{}clear'.format(prefix), value = 'Убирает нужное количество сообщений')
    emb.add_field (name = '{}ping'.format(prefix), value = 'Эта команда показывает задержку')
    emb.add_field (name = '{}credits'.format(prefix), value = 'Эта команда показывает информацию о разработчике')

    await ctx.send (embed = emb)
```

Рисунок 2 – Команда help

4. РАЗРАБОТКА КОДА ПРОГРАММНОГО ПРОДУКТА НА ОСНОВЕ ГОТОВЫХ СПЕЦИФИКАЦИЙ НА УРОВНЕ МОДУЛЯ

Данный Discord бот был разработан для «Государственного архива Республики Крым» для удобства воспроизведения документов, которые присутствуют только в аудио и видео формате. Благодаря этому появляется возможность перенести необходимую информацию из аудио формата в текстовый.

С помощью Discord бота можно также прослушивать аудио и видео материалы, которые находятся в общедоступном пользовании в Интернет-ресурсе на официальном сайте «Государственного архива Республики Крым».

Созданный бот предоставляет возможность упорядочивать документы в аудио формате и представлять их в виде списка.

Python – это язык программирования общего назначения, который широко применяется в различных областях. Язык скриптовый, он универсален и является самым популярным языком программирования в мире [4]. Интерпретируемость — одно из главных преимуществ Python. Интерпретируемым называется тот язык, код на котором не нужно компилировать, а можно сразу запускать.

Кроме этого, можно выделить следующие преимущества данного языка программирования:

- сходство с английским языком;
- простота изучения;
- бесплатная доступность и открытый исходный код; • независим от ОС.

Рисунок 3 - Логотип Python

При этом PyCharm делает разработку максимально продуктивной благодаря функциям автодополнения и анализа кода, мгновенной подсветке ошибок и быстрым исправлениям. Автоматические рефакторинги помогают эффективно редактировать код, а удобная навигация позволяет мгновенно перемещаться по проекту [5].

У данной программы можно выделить следующие преимущества:

- понятный git;
- простая организация проектов;
- удобный автокомплит; • приятный интерфейс; • очень быстрый.

Рисунок 4 - Логотип PyCharm

Основной целью работы является создание Discord бота для прослушивания музыки. Сам процесс будет делиться на две части – установка необходимого ПО и добавления команд.

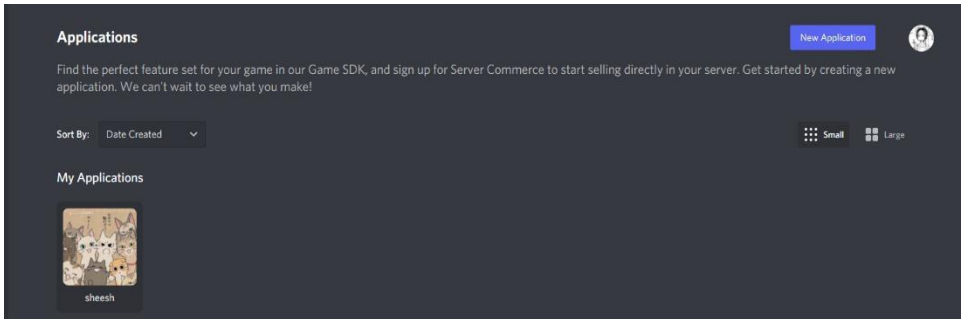

Рисунок 5 – Создание бота

На рисунке выше представлен официальный сайт Discord, который предназначен для разработки ботов – Discord Developers Portal. Для создания нового приложения необходимо нажать на кнопку «My Application», которая позволит создать само приложение, и дать ему имя, добавить описание и картинку.

Для добавления бота на сервер нужна ссылка-приглашение, которая будет находиться в разделе «URL Generator».

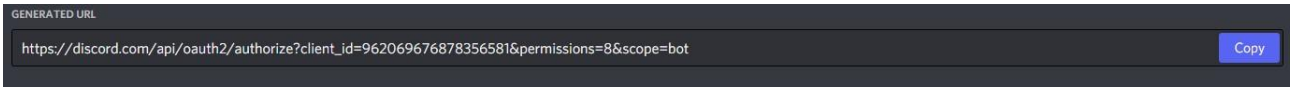

Рисунок 6 – Сгенерированная ссылка

Выполнив эти действия, сайт выдает сгенерированную ссылку. После чего бот появляется на сервере с соответствующим значком «БОТ».

Теперь можно перейти к написанию кода, где необходимо создать переменную client, и дать ей командный префикс. С помощью него бот будет понимать, что это обращение к нему.

```
client = commands.Bot(command_prefix='!')
```

Рисунок 7 - Префикс бота

После можно перейти к созданию команд. С помощью них соответственно и можно будет управлять ботом.

```
@client.command()
async def join(ctx):
    if not ctx.message.author.voice:
        await ctx.send("Ты не подключен к голосову каналу")
        return
    else:
        channel = ctx.message.author.voice.channel

    await channel.connect()
```

Рисунок 8 - Создание команды join

```
@client.command()
async def leave(ctx):
    voice_client = ctx.message.guild.voice_client
    await voice_client.disconnect()
```

Рисунок 9 - Создание команды leave

На двух рисунках выше представлены команды join и leave. С помощью первой команды можно присоединить бота к голосовому каналу, а благодаря второй его можно отключить от этого же канала.

Следующая команда, которую будет создана - команда queue. Она будет выступать в роли плейлиста, благодаря ей, можно добавить сразу несколько песен, и прослушивать треки по очередности.

```
@client.command()
async def queue(ctx, *, url):
    global que

    que.append(url)
    await ctx.send(f'`{url}` добавлена в очередь!')
```

Рисунок 10 - Создание команды queue

На двух рисунках ниже показаны команды view и remove. View будет показывать все песни, которые находятся в плейлисте, а с помощью команды remove можно удалить какой-либо трек из этого списка.

```
@client.command(pass_context = True)
async def view(ctx):
    await ctx.send(f'Сейчас очередь состоит состоит из `{que}` !')
```

Рисунок 11 - Создание команды view

```
@client.command()
async def remove(ctx, number):
    global que

    try:
        del(que[int(number)])
        await ctx.send(f'Сейчас ваша очередь `{que}!`')

    except:
        await ctx.send('Ваша очередь пуста или такого номера нет в очереди :(')
```

Рисунок 12 - Создание команды remove

Дальше можно создать команду loopmode. При использовании этой команды песня, которая сейчас играет, будет зацикливаться.

```
    @client.command()
    async def loopmode(ctx):
        global loop

        if loop:
            await ctx.send('Мод зацикливания сейчас `Выключен`')
            loop = False;

        else:
            await ctx.send('Мод зацикливания сейчас `Включен`')
            loop = True
```

Рисунок 13 - Создание команды loopmode

Время приступить к команде play, которая будет соответственно начинать воспроизводить песни в очереди.

```
@client.command()
async def play(ctx):
    global que

    if not ctx.message.author.voice:
        await ctx.send("Ты не подключен к голосовому каналу")
        return

    elif len(que) == 0:
        await ctx.send('В очереди пусто! Используй `!queue`, чтобы добавить песню!')

    else:
        try:
            channel = ctx.message.author.voice.channel
            await channel.connect()
        except: pass
```

Рисунок 14 - Создание команды play

И для удобства стоит добавить еще команды для остановки, продолжения и прекращения проигрывания песни.

Команда pause будет останавливать проигрываемую песню.

```
@client.command()
async def pause(ctx):
    server = ctx.message.guild
    voice_channel = server.voice_client

    voice_channel.pause()
```

Рисунок 15 - Создание команды pause

Команда resume будет продолжать проигрывать песню после остановки.

```
@client.command()
async def resume(ctx):
    server = ctx.message.guild
    voice_channel = server.voice_client

    voice_channel.resume()
```

Рисунок 16 - Создание команды resume

Команда stop будет прекращать проигрывать песню, и будет переходить к следующей.

```
@client.command()
async def stop(ctx):
    server = ctx.message.guild
    voice_channel = server.voice_client

    voice_channel.stop()
```

Рисунок 17 - Создание команды stop

Теперь создаем команду, которая при запуске кода будет нам высвечивать, что бот онлайн. Так мы будем знать, что бот запустился, и готов выполнять команды.

```
@client.event
async def on_ready():
    change_status.start()
    print('Бот онлайн!')
```

Рисунок 18 - Создание команды on ready

Далее можем перейти к команде, которая будет менять статус активности бота. Для этого необходимо создать переменную status, где будут прописаны все статусы.

```
status = ['я сплю', 'отдыхаю', 'танцую']
```

Рисунок 19 – Переменная status

```
@tasks.loop(seconds=180)
async def change_status():
    await client.change_presence(activity=discord.Game(choice(status)))
```

Рисунок 20 - Создание команды change status

Ниже представлена команды credits. Она при использовании просто будет выводить прописанный в ней текст.

```
@client.command()
async def credits(ctx):
    await ctx.send(`Поставьте 5, пожалуйста`)
```

Рисунок 21 - Создание команды credits

Для общения с ботом была создана команда hello. Вызывая ее, бот будет в ответ выдавать случайно выбранную фразу из тех, что прописаны в коде.

```
@client.command()
async def hello(ctx):
    responses = ['я сплю', 'привет', 'пока', 'спокойной ночи']
    await ctx.send(choice(responses))
```

Рисунок 22 - Создание команды hello

И последняя команда, которая будет добавлена – команда clear. Она будет соответственно очищать отправленные сообщения. Лимит 100 сообщений.

```
@client.command()
async def clear(ctx, amount = 100):
    await ctx.channel.purge(limit = amount)
```

Рисунок 22 - Создание команды clear

5. ОТЛАДКА ПРОГРАММНЫХ МОДУЛЕЙ С ИСПОЛЬЗОВАНИЕМ СПЕЦИАЛИЗИРОВАННЫХ ПРОГРАММНЫХ СРЕДСТВ

Отладка — это поиск, анализ и устранение ошибок в программном обеспечении, которые были найдены во время тестирования.

Отладчик позволяет разработчику контролировать выполнение и проверять состояние программ. Сравнение фактических и ожидаемых значений переменных или наблюдение за ходом выполнения кода может помочь в отслеживании логических ошибок [6].

Рисунок 23 - Работа отладчика в PyCharm

На рисунке выше представлен отладчик PyCharm, где он нашел ошибку.

Если код запускается без ошибок, то в отладчике будет показано сообщение, что бот вышел в онлайн.

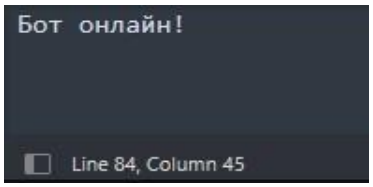

Рисунок 24 – Запуск кода

6. ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

Тестирование программного кода - процесс выполнения программного кода, направленный на выявление существующих в нем дефектов. Под дефектом здесь понимается участок программного кода, выполнение которого при определенных условиях приводит к неожиданному поведению системы [7].

Использовав ссылку-приглашения, можно добавить бота на сервер, где и можно его проверить.

Для начала посмотрим, как работают старые созданные команды. Применив команду join бот должен присоединиться к отправителю в голосовой канал, иначе будет высвечено соответствующее сообщение.

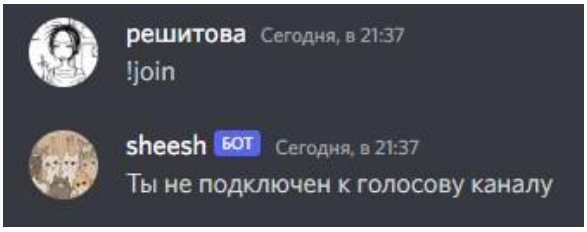

Рисунок 25 – Использование команды join, если мы не находимся в голосовом канале

Проверяя команду queue, мы должны отправить боту ссылку на песню, либо написать ее название.

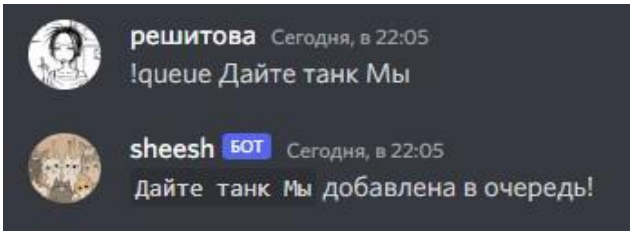

Рисунок 26 – Проверка команды queue

Проверка команды view для просмотра очереди.

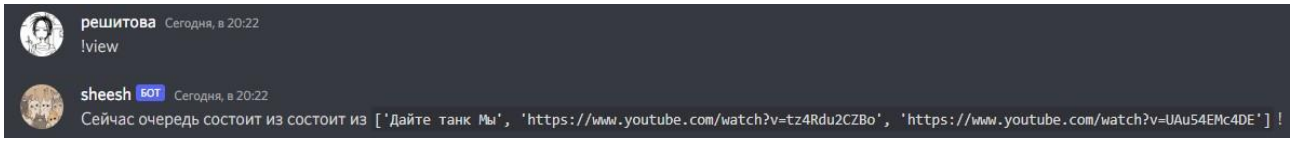

Рисунок 27 - Проверка команды view

Проверка команды remove для удаление какой-либо песни из очереди.

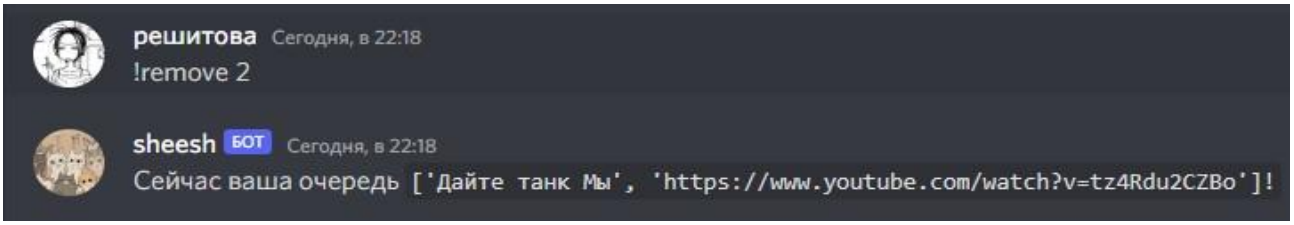

Рисунок 28 – Проверка команды remove

Проверка команды loopmode, которая зацикливает песню.

Рисунок 29 – Проверка команды loopmode

Проверка команды play, при ее использовании бот пишет в чат название песни, которая сейчас играет.

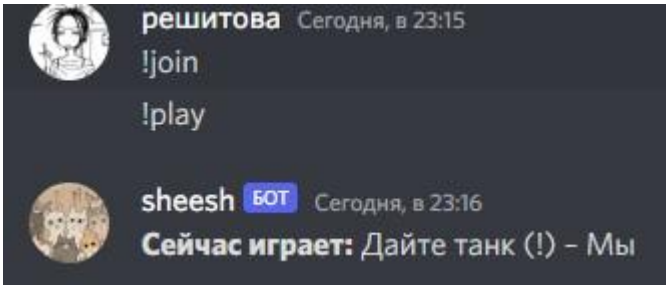

Рисунок 30 – Проверка команды play

При запуске бота мы видим, что работает команда on ready.

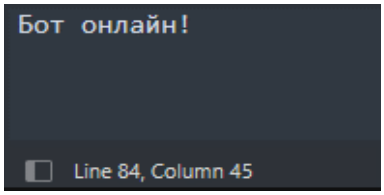

Рисунок 31 – Проверка команды ready on

Зайдя на сервер, где находиться бот, можно заметить, что вторая команда status change также работает.

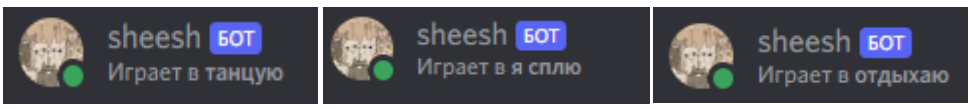

Рисунок 32 – Проверка команды change status

Далее проверяем команду credits.

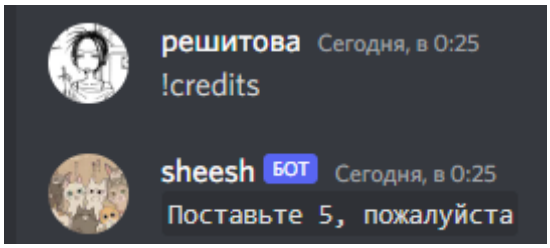

Рисунок 33 – Проверка команды credits

И остается команда hello, она отправляет случайную фразу.

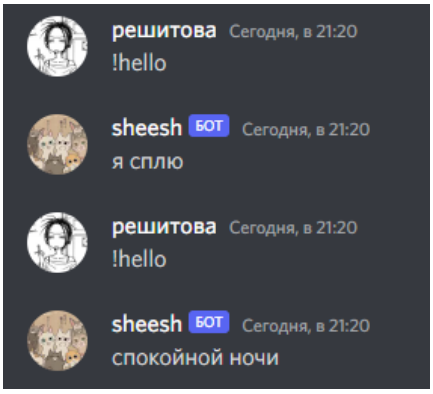

Рисунок 34 – Проверка команды hello

7. ОПТИМИЗАЦИЯ ПРОГРАММНОГО КОДА МОДУЛЯ

Оптимизация кода - различные методы преобразования кода ради улучшения его характеристик и повышения эффективности. Среди целей оптимизации можно указать уменьшения объема кода, объема используемой программой оперативной памяти, ускорение работы программы, уменьшение количества операций ввода-вывода [8].

При выполнении работы я использовала несколько сторонних библиотек, который упростили процесс. Это discord.py и youtube_dl.

```
import asyncio
import discord
from discord.ext import commands, tasks
from discord.voice_client import VoiceClient
import youtube_dl
from random import choice
```

Рисунок 35 – Импортируемые библиотеки

А также было установлено стороннее ПО – FFmpeg, чтобы бот проигрывал музыку с сайта YouTube. Он конвертирует видео в mp3 файл для воспроизведения аудио в Discord.

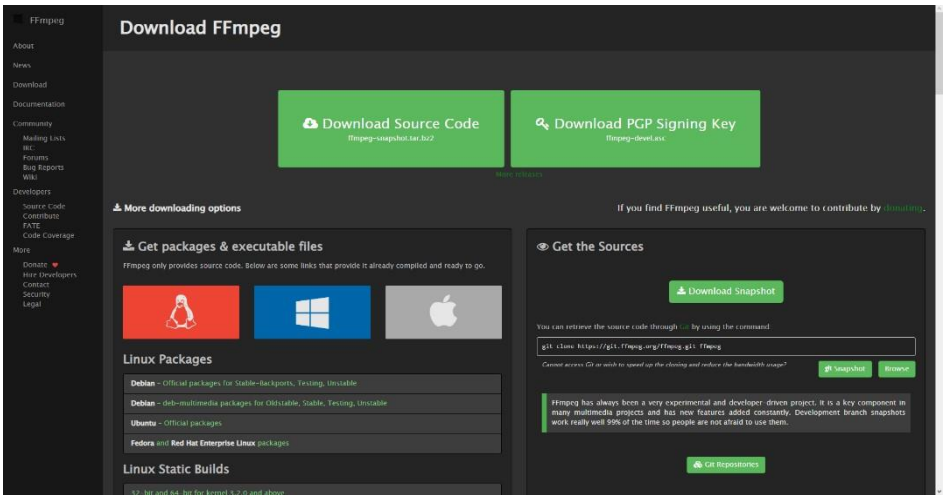

Рисунок 36 – сайт FFmpeg

8. РАЗРАБОТКА КОМПОНЕНТОВ ПРОЕКТНОЙ И ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ ЯЗЫКОВ СПЕЦИФИКАЦИЙ

Целью работы было разработать музыкального Discord бота. Приложение было разработано в среде PyCharm на языке программирования Python.

Были использованы библиотеки discord.py и youtube_dl. Они были просты в использовании, и понятны для любого пользователя. Также была установлена сторонняя программа FFmpeg, благодаря которой можно не прибегать к употреблению сложных функций конвертирования и скачивания.

Разработанный музыкальный бот представляет собой программу, которая дает проигрывать любые треки на пользовательском сервере в Discord. Так как сама платформа не имеет возможности транслировать треки, они воспроизводятся по принципу имитации запуска музыки, но выводят ее не с помощью динамиков, а микрофона. Благодаря такому принципу как раз и передаются все треки.

Для его подключения требуется всего лишь перейти по ссылке-приглашению, где высветиться окно с доступными серверами.

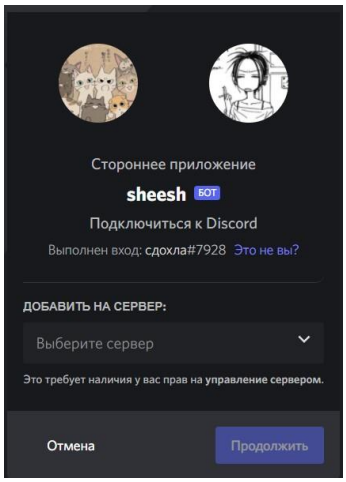

Рисунок 37 – Добавление на сервер

Далее бот будет добавлен на выбранный сервер.

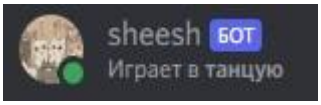

Рисунок 38 – Бот на сервере

Чтобы узнать какая команда необходима пользователю нужно использовать команду help, которая покажет отправителю что может делать бот.

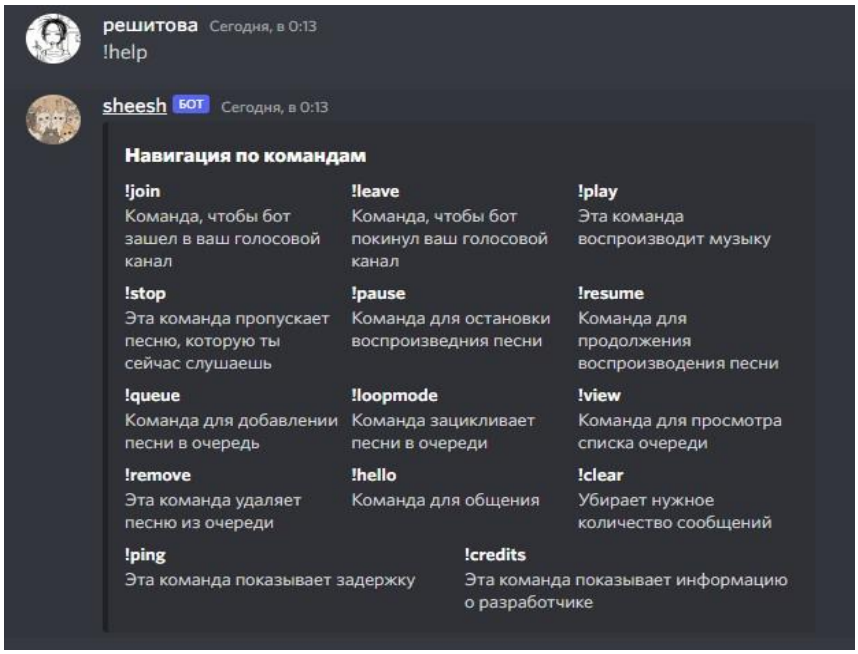

Рисунок 39 – Команда help

По итогу работы можно отметить, что бот является отличным решением для людей, которые при общении с друзьями желают послушать музыку вместе. Так как приложение выполняет все свои функции. При необходимости можно добавить еще команды, которые могут разнообразить функционал бота.

9. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГКУ РК "Государственный Архив Республики Крым" [электронный ресурс] // URL: https://krymgosarchiv.ru – (дата обращения 25.06.2022)
2. РОСКОМНАДЗОР. Правовая информация в сфере информационных технологий" [электронный ресурс] // URL: https://78.rkn.gov.ru/law/p4641/ – (дата обращения 25.06.2022)
3. Ольховский Д.Д. Методические указания “Разработка спецификаций Программному продукту” [электронный ресурс] https://infourok.ru/ - URL: https://infourok.ru/metodicheskie-ukazaniya-razrabotka-specifikacij-k-programmnomuproduktu-5025705.html. – (дата обращения 25.06.2022)
4. Язык программирования Python: применение, особенности и перспективы [электронный ресурс] // URL: https://timeweb.com/ru/community/articles/chto-takoepython/ – (дата обращения 25.06.2022)
5. Хабр. PyCharm [электронный ресурс] // URL: https://habr.com/ru/post/122018/ – (дата обращения 25.06.2022)
6. Skillfactory. Отладка [электронный ресурс] // URL: https://blog.skillfactory.ru/glossary/otladka-debugging/ - (дата обращения: 06.05.2022)
7. Тестирование программного кода (методы+окружение) [электронный ресурс] // URL: https://intuit.ru/studies/courses/1040/209/lecture/5385/ – (дата обращения 25.06.2022)
8. Pvs-studio. Оптимизация кода [электронный ресурс] // URL: https://pvsstudio.com/ru/blog/terms/0084/ – (дата обращения 25.06.2022)