

**«Санкт-Петербургский государственный электротехнический университет  
«ЛЭТИ» им. В.И.Ульянова (Ленина)»  
(СПбГЭТУ «ЛЭТИ»)**

<b>Направление</b>	01.03.02 – Прикладная математика и информатика
<b>Профиль</b>	Математическое обеспечение программно-информационных систем
<b>Факультет</b>	КТИ
<b>Кафедра</b>	МО ЭВМ

*К защите допустить*

И. о. зав. кафедрой

А.А. Лисс

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
БАКАЛАВРА**

**Тема: Разработка Discord-бота для автоматизации учебного процесса**

Студент		<hr/>	Р.В. Рыжих
		<i>подпись</i>	
Руководитель	К.Т.Н., доцент	<hr/>	М.М. Заславский
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	
Консультанты		<hr/>	С.А. Глазунов
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	
	К.Э.Н.	<hr/>	Артамонова О.С.
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	

Санкт-Петербург

2023

## ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю

И.о. зав. кафедрой МО ЭВМ

\_\_\_\_\_ А.А. Лисс

«\_\_» \_\_\_\_\_ 2023 г.

Студент          Рыжих Р.В.

Группа 9382

Тема работы: Разработка Discord-бота для автоматизации учебного процесса

Место выполнения ВКР: Санкт-Петербургский государственный электро-  
технический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

Исходные данные (технические требования):

Технологии программирования выбираются исполнителем

Содержание ВКР:

Обзор предметной области, постановка задачи и формулировка требований,  
архитектура программной реализации, обеспечение качества разработки,  
продукции, программного продукта.

Перечень отчетных материалов: пояснительная записка, иллюстративный  
материал.

Дополнительные разделы: обеспечение качества разработки, продукции,  
программного продукта

Дата выдачи задания

Дата представления ВКР к защите

«4» \_\_\_\_\_ февраля \_\_\_\_\_ 2023 г.

«8» \_\_\_\_\_ июня \_\_\_\_\_ 2023 г.

Студент

\_\_\_\_\_

Р.В. Рыжих

Руководитель к.т.н., доцент

(Уч. степень, уч. звание)

\_\_\_\_\_

М.М. Заславский

Консультант

(Уч. степень, уч. звание)

\_\_\_\_\_

С.А. Глазунов

# КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю

И. о. зав. кафедрой МО ЭВМ

\_\_\_\_\_ А.А. Лисс

« \_\_\_\_ » \_\_\_\_\_ 2023 г.

Студент Рыжих Р.В.

Группа 9382

Тема работы: Разработка Discord-бота для автоматизации учебного процесса

№ п/п	Наименование работ	Срок вы- полнения
1	Обзор литературы по теме работы	04.02 – 01.05
2	Обзор предметной области	01.01 – 05.01
3	Постановка задачи и формулировка требований	06.01 – 10.01
4	Реализация	11.01 – 01.05
5	Обеспечение качества разработки, продукции, про- граммного продукта	02.05 – 05.05
6	Оформление пояснительной записки	06.05 – 13.05
7	Оформление иллюстративного материала	14.05 – 15.05

Студент

\_\_\_\_\_ Р.В. Рыжих

Руководитель к.т.н., доцент  
(Уч. степень, уч. звание)

\_\_\_\_\_ М.М. Заславский

Консультант

(Уч. степень, уч. звание)

\_\_\_\_\_ С.А. Глазунов

## РЕФЕРАТ

Пояснительная записка 51 стр., 22 рис., 3 табл., 19 ист.

DISCORD, БОТ, АВТОМАТИЗАЦИЯ.

**Объектом исследования** является автоматизация дистанционного учебного процесса в приложении Discord

**Предметом исследования** является Discord-бот для администрирования учебного Discord-сервера и автоматизации учебного процесса.

**Цель работы:** создание Discord-бота для автоматизированного управления Discord-сервером и ведения учебного процесса.

В рамках работы проведён анализ существующих Discord-ботов на возможность выполнять перечень задач для автоматизации дистанционного процесса обучения через приложение Discord. По результатам сравнения было решено создать собственного Discord-бота для автоматизации дистанционного обучения. В работе представлен метод работы Discord-бота, а также взаимодействие с ним через команды внутри приложения Discord и через созданный веб-сервис.

## **ABSTRACT**

As part of the work, an analysis of existing Discord bots was carried out for the ability to perform a list of tasks to automate the distance learning process through the Discord application. Based on the results of the comparison, it was decided to create our own Discord bot to automate distance learning. The paper presents the method of operation of the Discord bot, as well as interaction with it through commands inside the Discord application and through the created web service.

## СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....	7
ВВЕДЕНИЕ.....	8
1 Обзор предметной области.....	10
1.1 Обзор существующих Discord-ботов .....	10
1.1.1 Принцип отбора аналогов .....	10
1.1.2 Определение критериев оценки аналогов .....	11
1.1.3 Сравнительный анализ аналогов .....	12
2 Постановка задачи и формулировка требований.....	16
2.1 Постановка задачи.....	16
2.2 Требования к решению .....	16
3 Архитектура программной реализации .....	18
3.1 Используемые технологии .....	18
3.2 Клиент Discord-бота.....	19
3.2.1 Основной класс .....	19
3.2.2 Вспомогательные классы.....	20
3.3 Слэш-команды .....	26
3.4 Веб-сервис.....	38
4 Обеспечение качества разработки, продукции, программного продукта ....	42
4.1 Определение потребителей .....	42
4.2 Функция продукции и услуги .....	42
4.3 Качество и характеристики .....	42
4.4 Измерение характеристик качества.....	44
4.5 Выводы.....	48
ЗАКЛЮЧЕНИЕ .....	49
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	50

## **ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

ЯП – язык программирования;

IDE – Integrated Development Environment;

PR – PullRequest.

## ВВЕДЕНИЕ

Приложение Discord позволяет работать с большой группой студентов в одном канале. В самом приложении предусмотрено создание нескольких голосовых и текстовых каналов для разделения участников по группам, предметам, преподавателям. Данный сервис предлагает следующий набор возможностей модерации: настройки безопасности (установка параметров доступа по уровню верификации аккаунта Discord), «автомод» (блокировка спама, фильтры по недопустимым словам и фразам), «журнал аудита» (просмотр действий участников Discord-сервера). Таким образом, с помощью Discord можно удобно организовать процесс дистанционного обучения.

Однако, чтобы администрировать Discord-сервер, необходимо выполнять множество постоянных повторяющихся действий: выдавать студентам и преподавателям роли, создавать отдельные голосовые и текстовые каналы, генерировать сообщения (приветствующие, напоминающие и др.) для предстоящих мероприятий (начало занятия, приближение срока сдачи контрольных и лабораторных работ).

Для автоматизации выполнения множества повторяющихся действий на данный момент существуют различные по возможностям и функциональности Discord-боты, выполняющие неполный перечень необходимых действий. Кроме того, многие из них являются платными. Одновременное использование нескольких ботов может привести к сложностям в управлении Discord-сервером у администраторов (например, использование различных префиксов команд управления). То есть решения существуют, но реализуют неполный перечень необходимых действий, что является недостатком. Поэтому необходимо разработать своё решение.

**Цель данной работы** состоит в создании Discord-бота для автоматизированного управления Discord-сервером и ведения учебного процесса.

### **Задачи данной работы:**

1. Выявить необходимые требования к разрабатываемому Discord-боту.



2. Проанализировать существующих Discord-ботов для автоматизированного управления Discord-сервером и ведения учебного процесса.
3. Найти технологии, пригодные для создания Discord-бота и выполнения поставленных задач.
4. Создать Discord-бота, удовлетворяющего поставленным требованиям.

**Объектом исследования** является автоматизация дистанционного учебного процесса.

**Предметом исследования** является Discord-бот для администрирования Discord-сервера и автоматизации учебного процесса.

**Практическая ценность работы:** внедрение разработанного приложения позволит автоматизировать проведение дистанционного учебного процесса на Discord-сервере.

## **1 Обзор предметной области**

### **1.1 Обзор существующих Discord-ботов**

#### **1.1.1 Принцип отбора аналогов**

Для поиска аналогов Discord-бота по администрированию сервера использовались запросы по следующим словам и словосочетаниям:

Запросы Google:

Русский язык:

- Дискорд боты для сервера
- Боты для администрирования дискорд
- Боты дискорд для обучения

Английский язык:

- Discord bots administration
- Discord bots for server administration
- Discord bots for moderation
- Discord bots for education

Запросы в каталог приложений (App Directory) в Discord:

- Moderation
- Education

Для подбора аналогов учитывались только Discord-боты, продолжающие свою работу на момент мая 2023 года. Также, ввиду огромного количества аналогов (Само приложение Discord насчитывает 3240 ботов, из которых 1565 - боты для модерации Discord-серверов), в качестве аналогов выступают либо наиболее популярные боты, либо те, которые реализуют конкретные задачи.

### **1.1.2 Определение критериев оценки аналогов**

Discord-бот будет использоваться преподавателями и администраторами для ведения дистанционного обучения. Автоматизация процесса подразумевает собой администрирование сервера, уведомления пользователей о различных событиях. Для преподавателей и администраторов предусматривается графический интерфейс (приложение или веб-сервис) для удобного доступа к функциональности Discord-бота.

В связи с тем, что разрабатываемый Discord-бот должен автоматизировать процесс дистанционного обучения, он обязан уметь решать следующие задачи: отправка истории сообщений, выдача ролей участникам, генерация настраиваемых сообщений, генерация новых категорий с голосовыми и текстовыми каналами, уведомление пользователей о приближающемся сроке сдачи лабораторных и курсовых работ, заполнение таблиц для распределения вариантов по лабораторным и курсовым работам.

Дополнительно ко всему, у Discord-бота должен быть открытый исходный код для исключения случаев: индивидуального запрета, неожиданного выключения и изменения функциональности разработчиком.

Из выше сказанного, к Discord-боту предъявляются следующие требования:

1. Администрирование Discord-сервера (установка ролей участникам, удаление пользователей по роли, генерация новых категорий)
2. Уведомление участников о событиях (генерация первого сообщения на защиту, уведомление о приближающихся сроках сдачи работ, добавление мероприятий с началом занятия в календарь)
3. Наличие приложения или веб-сервиса для работы с Discord-ботом
4. Распределение вариантов по лабораторным и курсовым работам
5. Наличие открытого исходного кода

### 1.1.3 Сравнительный анализ аналогов

Для анализа существующих аналогов были выбраны следующие Discord-боты:

#### *MEE6*

MEE6 — это бот, использующийся для модерации Discord-серверов. Данный бот позволяет создавать определённые настраиваемые команды, а также устанавливать участникам роли посредством реакций на определённое сообщение. [13]

#### *JuniperBot*

JuniperBot — это бот, использующийся для модерации Discord-серверов. Данный бот позволяет настраивать префикс команд, права доступа к командам, создавать определённые настраиваемые команды, а также настраивать дополнительное логирование событий, происходящих на Discord-сервере. [14]

#### *Dyno*

Dyno — это бот, использующийся для модерации Discord-серверов. Данный бот позволяет устанавливать роли участникам, создавать определённые настраиваемые команды, создавать напоминания. Также, присутствует веб-сервис, позволяющий настраивать бота через него. [15]

#### *Maki*

Maki — это бот, использующийся для модерации Discord-серверов. Данный бот позволяет удалять спам, фильтровать лишние упоминания, оскорбления, выдавать пользователям роли. Также присутствует веб-сервис, позволяющий настраивать бота через него. [16]

#### *Bosley*

Bosley — это бот-планировщик, который позволяет пользователям устанавливать напоминания о событиях, трансляциях и т. д. и делиться ими.

Bosley может отправлять уведомления определенным каналам и ролям, пользователи могут соглашаться (или отказываться) от получения уведомлений, а администраторы могут применять уведомления. [17]

### *ProBot*

ProBot — это бот, использующийся для модерации Discord-серверов. Данный бот позволяет настраивать дополнительное логирование событий на Discord-сервере, удалять спам и устанавливать фильтры на сообщения, а также автоматически выдавать роли участникам, присоединившимся по определённой ссылке. Также присутствует веб-сервис, позволяющий настраивать бота через него. [18]

Результаты анализа представлены в таблице 1.

Таблица 1 -- Анализ существующих Discord-ботов по выделенным критериям

Название Discord-бота	Администрирова- ние	Уведомления участников	Веб-сервис или приложение	Распределение вариан- тов	Наличие откры- того исходного кода
MEE6	+	-	+	-	-
JuniperBot	+	+	-	-	+
Dyno	+	+	+	-	+
Maki	+	-	+	-	+
Bosley	-	+	-	-	-
ProBot	+	-	+	-	-

По итогам проведённого анализа можно установить, что ни один из ботов не удовлетворяет всем существующим критериям.

Настройка пользовательских команд в ботах MEE6, JuniperBot, Dyno и ProBot не позволяет реализовать недостаточную функциональность путём создания определённых команд, а веб-сервис у ботов MEE6, Dyno, Maki и ProBot не позволяет отправлять команды ботам для удалённого управления Discord-сервером.

## **2 Постановка задачи и формулировка требований**

### **2.1 Постановка задачи**

Необходимо разработать Discord-бота для администрирования учебного Discord-сервера с дополнительными функциями, помогающими облегчить организацию учебного процесса.

У проекта должны быть реализованы следующие функции:

1. Установка ролей участникам
2. Удаление пользователей по роли
3. Генерация новых категорий с голосовыми и текстовыми каналами
4. Генерация первого настраиваемого сообщения на защиту
5. Уведомление о приближающихся сроках сдачи работ
6. Добавление мероприятий, таких как: сдача лабораторных и курсовых работ, защита лабораторных и курсовых работ, в календарь
7. Распределение вариантов по лабораторным и курсовым работам
8. Наличие приложения или веб-сервиса для работы с Discord-ботом

### **2.2 Требования к решению**

Результат должен обладать следующим набором свойств:

1. Приложение должно развёртываться на различных операционных системах или сервере
2. Пользователь может отдавать команды Discord-боту как через слэш-команды, так и через веб-сервис
3. Приложение должно уметь работать с GitHub репозиторием
4. Все необходимые константы должны находиться в файле
5. После перезапуска приложение должно сохранять своё предыдущее состояние



### **Обоснование требований к решению.**

1. Приложение можно будет развернуть локально на ПК на любой системе или сервере для дальнейшего использования в случае, если прошлый хостинг прекратит или приостановит свою работу
2. Пользователь сможет отдавать команды боту через текстовый канал в Discord с помощью специального символа '/', а также сможет использовать некоторые команды через веб-сервис с ПК или телефона.
3. Для генерации сообщения на защиту необходимо проверить, какие студенты выгрузили нужную работу на GitHub к определённому сроку
4. Размещение констант в файл позволит настраивать определённый шаблон, с которым будет работать приложение
5. При повторном хостинге приложения не надо будет заново настраивать бота для его корректной работы

### 3 Архитектура программной реализации

#### 3.1 Используемые технологии

Discord-бот разрабатывается на высокоуровневом ЯП Python с помощью библиотеки `nextcord.py` [1]. Для разработки веб-сервиса использован `React.js`, а для обработки запросов с веб-сервиса используется фреймворк `Quart` [2]. Дополнительно при разработке использовались следующие библиотеки:

- `github` – библиотека, используемая для получения информации с платформы GitHub [3].
- `os` – библиотека, используемая для получения переменных среды из операционной системы [4].
- `json` – библиотека, используемая для обработки файлов формата `json` [5].
- `nextcord.ext` – расширение к библиотеке `nextcord.py`, позволяющая создавать клиент межпроцессного взаимодействия и облегчающее работу с сообщениями на Discord-канале [6].
- `smtplib` – библиотека SMTP [7].
- `mimetypes` [8].
- `email` [9].
- `re` – библиотека, используемая для определения шаблонов регулярных выражений, предоставляющая методы для работы с ними: поиск первого вхождения, поиск всех вхождений, замена всех вхождений, проверка соответствия строки шаблону [10].
- `ics` [11].

## 3.2 Клиент Discord-бота

### 3.2.1 Основной класс

#### DiscordBot

Класс, наследуемый от библиотечного класса *commands.Bot*. Этот класс служит для создания и запуска Discord-бота. Через этот класс происходит взаимодействие пользователя с DiscordAPI [12] посредством вызова слэш-команд, действий на Discord-сервере, и отправкой команд с веб-сервиса. Для создания объекта класса необходимо передать разрешения для бота. Все разрешения берутся из класса *nextcord.Intents*.

В конструкторе класса устанавливаются следующие поля:

- *events* – события (если бот запущен впервые, то событий не будет; если нет – события считываются из файла конфигурации бота);
- *message\_id* – идентификатор сообщения, по которому пользователям будут выдаваться роли;
- *roles* – список ролей, которые будут выдаваться пользователям;
- *ipc* – клиент межпроцессного взаимодействия для обмена данными между сервером и ботом.

В классе присутствуют следующие методы:

- *track\_message()* – функция, которая устанавливает поля *message\_id* и *roles*. Вызывается слэш-командой боту при его первоначальной настройке;
- *on\_raw\_reaction\_add()* – асинхронная функция, которая вызывается при любом добавлении реакции на сообщение на Discord-сервере. При вызове получает класс *payload*, содержащий в себе всю полученную информацию от *DiscordAPI*. Вызывает метод вспомогательного класса *DiscordManager add\_role\_by\_reaction()* с аргументами *client* (объект класса *DiscordBot*), *payload*, *message\_id*, *roles*;

- *on\_member\_join()* – асинхронная функция, которая вызывается при присоединении пользователя на Discord-сервер. При вызове получает класс *member*, содержащий в себе информацию о присоединившемся пользователе, полученную от *DiscordAPI*. Добавляет присоединившемуся пользователю роль с названием текущего года.

В файле с классом созданы следующие функции:

- *start()* – функция, которая загружает в объект класса *DiscordBot* слэш-команды, запускает клиент межпроцессного взаимодействия и самого Discord-бота;
- *generate\_message()* – функция, созданная с помощью декоратора *ipc.route()*. Служит для отправки сообщения на защиту в текстовый Discord-канал;
- *chat\_history()* – функция, созданная с помощью декоратора *ipc.route()*. Служит для отправки истории сообщений Discord-канала на почту.

### 3.2.2 Вспомогательные классы

Для того, чтобы класс *DiscordBot* не обладал слишком большой ответственностью за все задачи, были созданы два вспомогательных класса: *DiscordManager* и *EventManager*

#### **DiscordManager**

*DiscordManager* отвечает за выдачу пользователю роли на Discord-сервере по поставленной реакции. При изменении структуры сообщения, которое отслеживает бот (изменение значений реакций, создание новых реакций) в классе можно создать новую функцию обработки и в основном классе *DiscordBot* вызывать именно её.

В классе реализован метод *add\_role\_by\_reaction()*, который при добавлении реакции пользователем на определённое сообщение выдаёт ему соответствующую роль.

При добавлении пользователем новой реакции с него снимается прошлая роль и выдаётся новая. Такой способ реализации поможет студенту выбрать новую роль, если первый его выбор был неправильный (пользователь случайно поставил реакцию, которая не соответствует его группе)

Пример использования функции:

1. Создаётся сообщение для выбора реакции. Пример сообщения представлен на рис. 1.

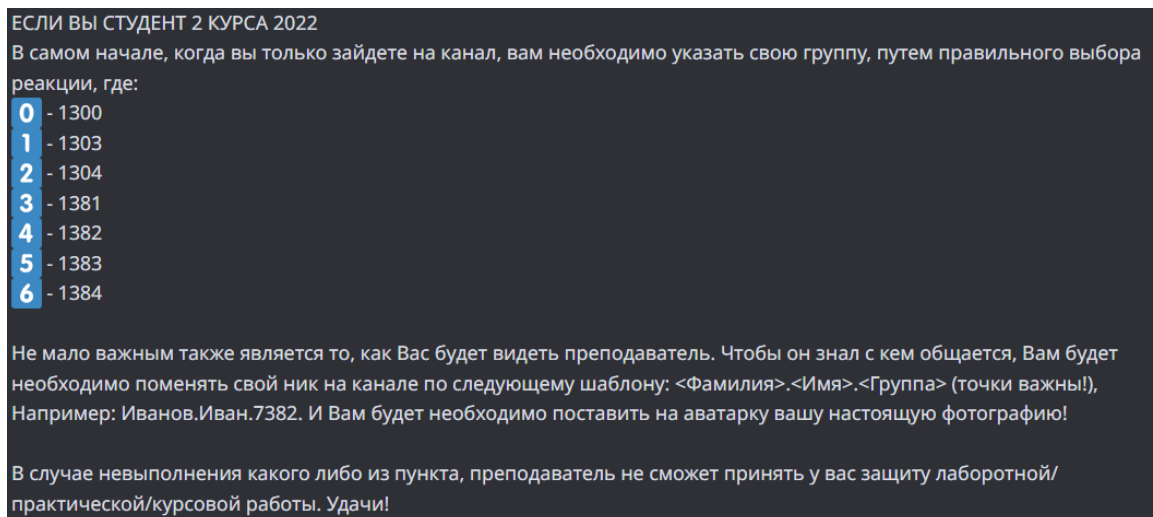


Рисунок 1 – Пример сообщения с реакциями.

2. Используется слэш-команда бота `/track_message` с параметрами `message_id` и `roles`. Пример использования слэш-команды представлен на рис. 2.

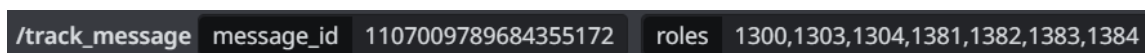


Рисунок 2 – Пример использования слэш-команды `track_message`

3. Пользователь оставляет реакцию на сообщение и у него появляется новая роль. Пример профиля пользователя до использования реакции и после представлены на рис. 3 и рис. 4.

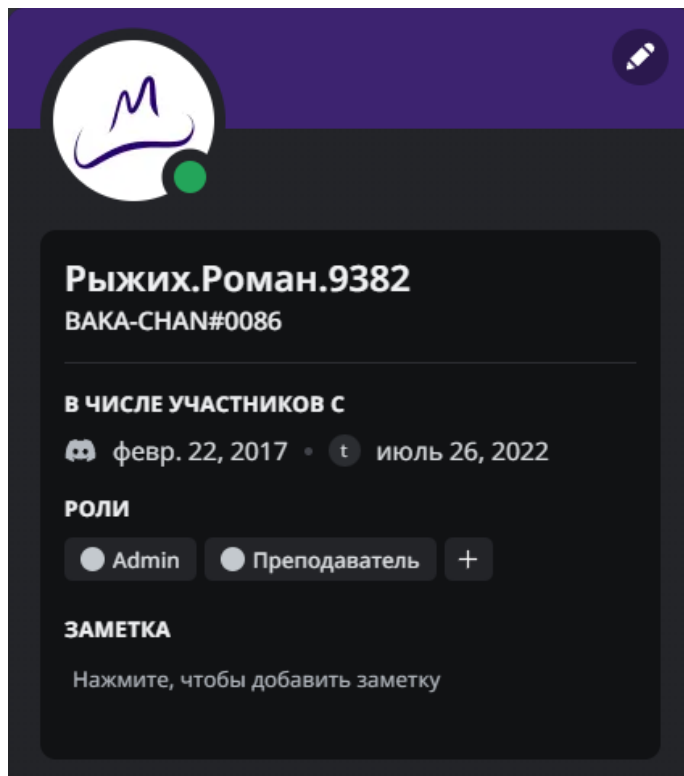


Рисунок 3 – Профиль пользователя до использования реакции.

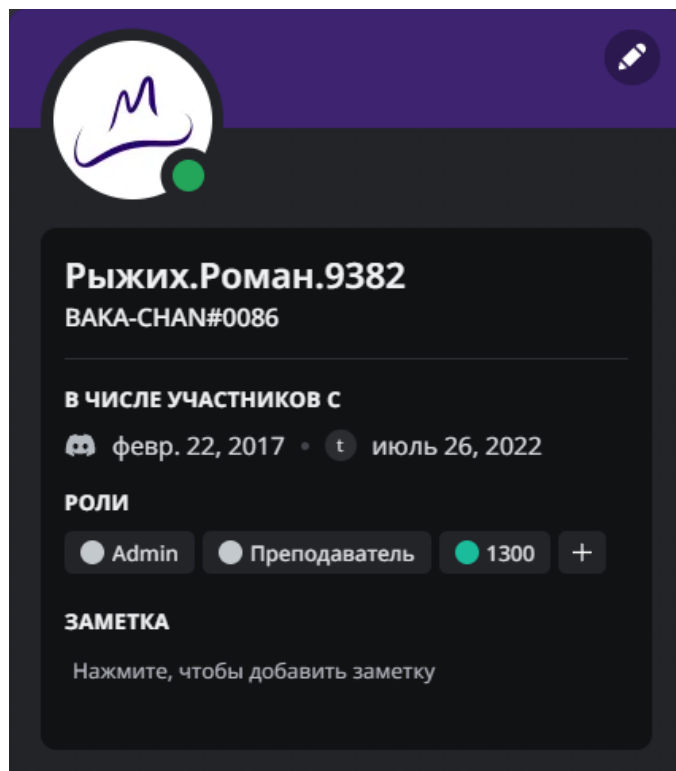


Рисунок 4 – Профиль пользователя после использования реакции.

## Event

Класс события. Под событиями подразумевается начало занятий, крайний срок сдачи лабораторных/курсовых работ и др. В конструкторе принимает следующие параметры:

- *index* – уникальный номер события, по которому можно удалять и изменять события;
- *title* – заголовок события;
- *time* – время проведения события;
- *data* – дата проведения события;
- *repeat* – повторение события (без повторений, ежедневно, еженедельно, каждые 2 недели);
- *description* – описание события;
- *roles* – роли, которые упоминаются в событии.

В классе реализованы следующие методы:

- *show()* – функция, которая возвращает строковое представление всех полей класса.
- *get\_embed()* – функция, которая создаёт объект Embed (особый формат сообщения в Discord), добавляет все поля класса Event и возвращает созданный объект. Пример Embed-сообщения представлен на рис. 5.

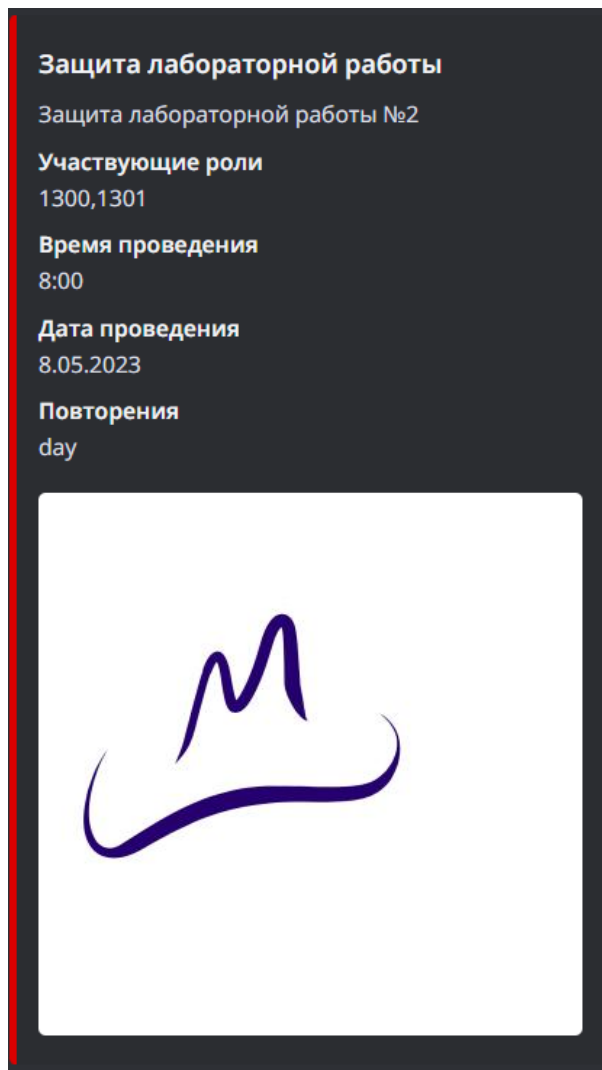


Рисунок 5 – Пример Embed-сообщения.

### EventManager

Класс EventManager отвечает за обработку событий (Event) и работу с ними. Любое событие устанавливается посредством слэш-команды боту.

Имеет только одно поле – *events*, которое представляет собой список объектов класса Event.

В классе реализованы следующие методы:

- *add\_event()* – функция, которая принимает в себя необходимые для класса Event параметры, создаёт объект класса Event и добавляет его в список *events*;
- *delete\_event()* – функция, которая отвечает за удаление событий из списка



- *show()* – функция, которая выводит в Discord-канал все события.
- *save()* – функция, которая сохраняет все существующие события в календарь. На вход принимает название файла календаря. В функции происходит создание объекта класса *ics.Calendar*, куда затем добавляются все события: создаются объекты класса *ics.Event*, в каждый из которых добавляются все поля событий; затем, в объект класса *ics.Calendar* добавляется созданный и настроенный объект *ics.Event*, после чего объект *ics.Calendar* записывается в файл. Возвращаемым значением данной функции является файл формата ics, который можно открыть через стандартные приложения календаря на ПК.

### 3.3 Слэш-команды

Слэш-команды – особый тип сообщения, адресованный Discord-боту (если Discord-бот поддерживает такие команды). Такие сообщения можно отправлять с любого текстового канала на Discord-сервере; все команды начинаются с слэш-символа ‘/’. При вводе в текстовый канал такого символа Discord выведет особое диалоговое окно, в котором будут отображаться стандартные слэш-команды Discord и доступные слэш-команды Discord-ботов, добавленных на Discord-сервер, которые поддерживают такие команды (для выбора, какому Discord-боту адресовывать команду). Пример диалогового окна представлен на рис. 6.

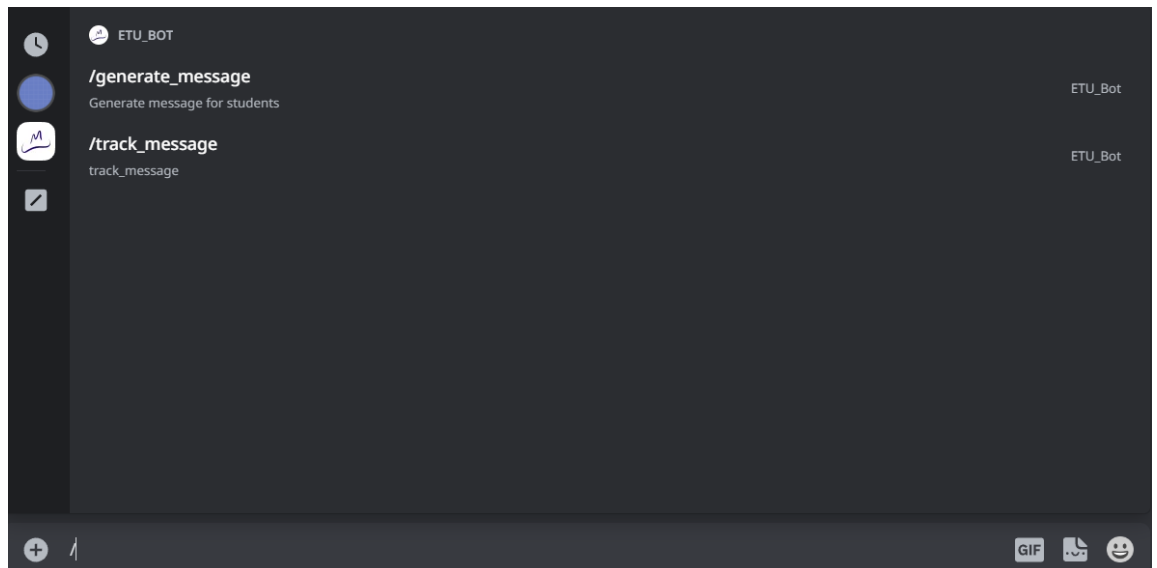


Рисунок 6 – Диалоговое окно выбора слэш-команд.

Все слэш-команды – отдельные классы, наследующиеся от класса *commands.Cog*. *commands.Cog* – класс, содержащий в себе коллекцию команд и опциональных состояний, чтобы группа команд могла работать вместе.

Все команды разделены по файлам, названия которых соответствует классу, реализованному в нём для облегчения отладки, настройки, добавления и удаления команд. В каждом файле слэш-команд присутствует функция *setup()*, которая вызывается классом *DiscordBot* при добавлении всех существующих слэш-команд. *setup()* добавляет объект класса команды к объекту класса *DiscordBot* при помощи унаследованного метода *add\_cog()*.

В каждом классе содержится один метод *self()*, реализованный через декоратор *nextcord.slash\_command*, который принимает в себя следующие параметры:

- *name* – название слэш-команды (написание для её вызова)
- *description* – описание слэш-команды, которое будет выводиться в диалоговом окне Discord
- *guild\_ids* – список, состоящий из идентификаторов Discord-серверов, на которых эта команда будет работать

При создании Discord-бота были реализованы следующие классы слэш-команд: *AddCategory*, *ChatHistory*, *Clear*, *CreateEvent*, *DeleteMembersByRole*, *GenerateMessage*, *GiveRole*, *ShowEvents*, *TrackMessage*.

### **AddCategory**

Этот класс служит для автоматизации добавления Discord-категорий с текстовыми Discord-каналами. В конструкторе принимает объект класса *DiscordBot*, к которому будет добавляться.

В классе присутствуют следующие поля:

- *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.
- *groups* – параметр, представляющий собой список, состоящий из номеров групп.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове слэш-команды. Необходим для создания категории и текстовых каналов в ней.
- *name* – строка, представляющая собой название категории, которое будет отображаться пользователям.

- *prefix* – строка, представляющая собой префикс для названий текстовых каналов.

В методе создаётся категория с переданным ей названием. Затем, по всем группам из списка *groups* в созданной категории создаются текстовые каналы, название которых соответствует шаблону *<prefix>-<номер группы>*.

Пример использования слэш-команды:

1. Пользователь вводит команду */add\_category* с параметрами *name* и *prefix*. Пример слэш-команды представлен на рис. 7.

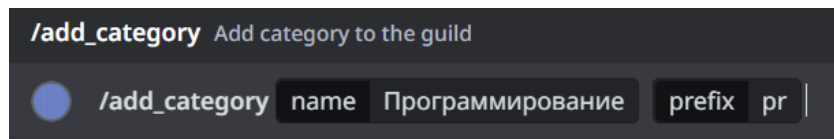


Рисунок 7 – Пример использования команды */add\_category*.

2. После отправки команды Discord-боту на Discord-сервере появится созданная категория со всеми текстовыми каналами для групп. Пример созданной категории с текстовыми каналами представлен на рис. 8.

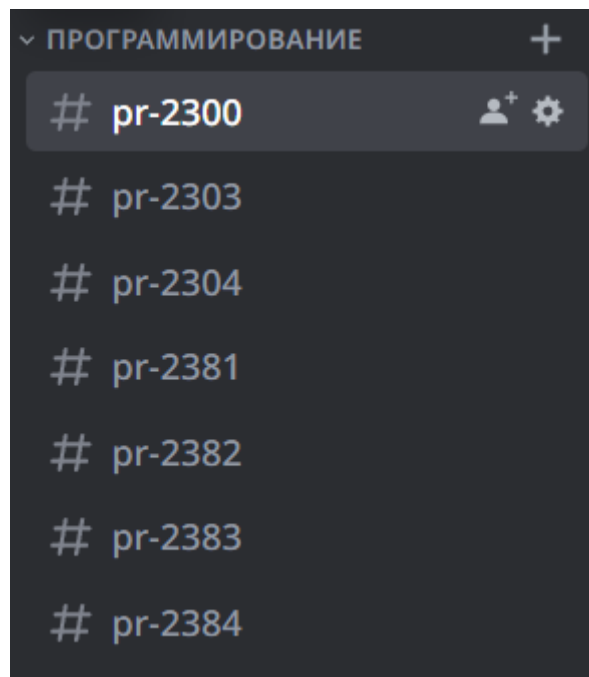


Рисунок 8 – Категория «Программирование» с текстовыми каналами групп.

## ChatHistory

Этот класс служит для отправки истории сообщений текстового канала по почте в виде файла формата txt.

В классе присутствует одно поле *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове слэш-команды. Необходим для просмотра истории сообщений текстового канала, в котором используется слэш-команда.
- *email* – адрес электронной почты, на который необходимо отправить историю сообщений.
- *limit* – опциональный параметр, представляющий собой количество сообщений для отправки. Если не указан, принимает значение *None*.

В методе создаётся список всех сообщений на Discord-канале, с которого была отправлена слэш-команда. Если параметр *limit* не указан, то в списке сообщений будут присутствовать все сообщения с момента создания текстового канала; если параметр *limit* указан, то в списке сообщений будет присутствовать то количество сообщений, которое передал пользователь. Затем, все сообщения из списка записываются в файл под тем же названием, что и Discord-канал, с которого была отправлена слэш-команда, в формате «год-месяц-день часы-минуты-секунды <имя пользователя>: <содержимое сообщения>» и файл отправляется на указанный в *email* адрес электронной почты.

Пример использования слэш-команды без параметра *limit*:

1. Пользователь вводит команду */chat\_history* с параметром *email* без указания ограничения количества сообщений. Пример слэш-команды представлен на рис. 9.

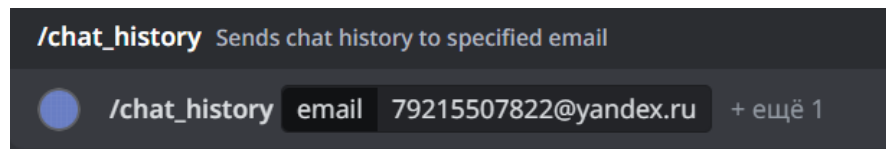


Рисунок 9 – Пример команды `/chat_history` без параметра *limit*.

2. На указанную почту приходит сообщение с прикреплённым файлом формата txt. Сообщения в Discord-канале представлены на рис. 10. Содержимое файла представлены на рис. 11.

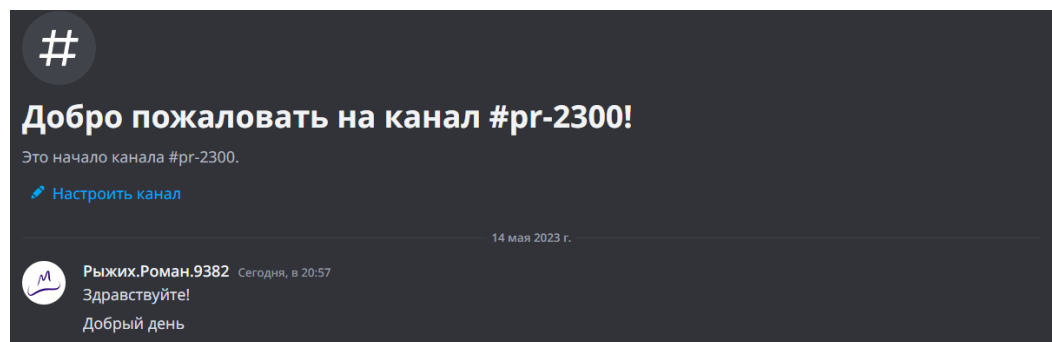


Рисунок 10 – Сообщения в Discord-канале.

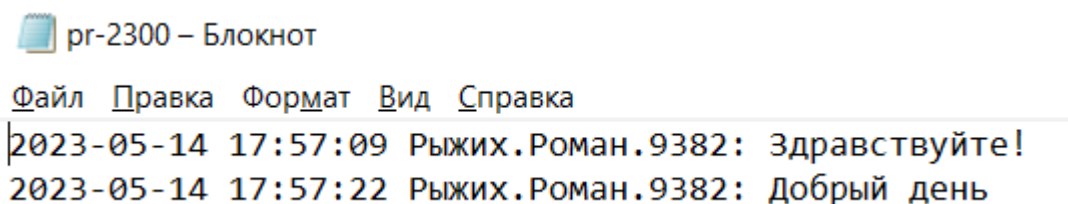


Рисунок 11 – Содержимое полученного файла при использовании команды без параметра *limit*.

Пример использования слэш-команды с параметром *limit*:

1. Пользователь вводит команду `/chat_history` с параметром *email* с указанием ограничения количества сообщений. Пример слэш-команды представлен на рис. 12.

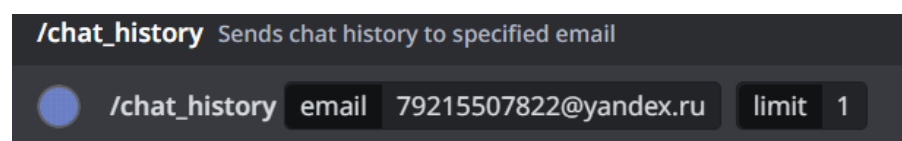



Рисунок 12 – Пример команды `/chat_history` с параметром *limit*.

2. На указанную почту приходит сообщение с прикрепленным файлом формата txt. Сообщения в Discord-канале представлены на рис.10. Содержимое файла представлено на рис. 13.

 pg-2300 – Блокнот

Файл Правка Формат Вид Справка

2023-05-14 17:57:09 Рыжих.Роман.9382: Здравствуйте!

Рисунок 13 – Содержимое полученного файла при использовании команды с параметром *limit*.

## Clear

Этот класс служит для удаления сообщений текстового канала.

В классе присутствует одно поле *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове слэш-команды. Необходим для удаления сообщений из текстового канала.
- *amount* – количество сообщений для удаления.

В методе класса производится удаление определённого количества последних сообщений, переданных через параметр *amount*.

## CreateEvent

Этот класс служит для создания событий класса Event.

В классе присутствует одно поле *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове слэш-команды.
- *title* – заголовок события.
- *time* – время проведения события.
- *data* – дата проведения события.
- *repeat* – повторение события (без повторений, ежедневно, еженедельно, каждые 2 недели).
- *description* – описание события.
- *roles* – роли, которые упоминаются в событии.

В методе к объекту класса *DiscordBot* добавляется событие при помощи метода *add\_event()* с всеми параметрами кроме *interaction*. Принцип работы метода *add\_event()* описан выше в классе *EventManager*.

Пример использования слэш-команды:

1. Пользователь вводит команду */create\_event* с всеми необходимыми параметрами. Пример слэш-команды представлен на рис. 14.

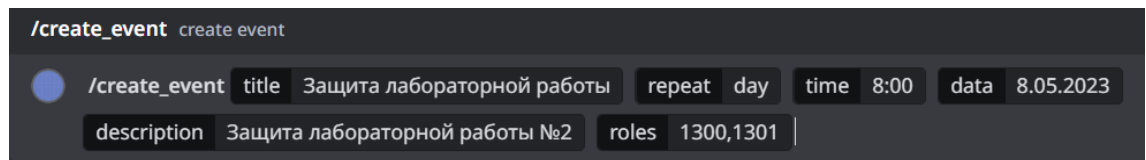


Рисунок 14 – Пример команды */create\_event*.

2. Создаётся событие, которое можно будет вывести впоследствии.

## DeleteMembersByRole

Этот класс служит для удаления пользователей с определённой ролью.

В классе присутствует одно поле *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове



слэш-команды. Необходим для получения информации о всех участниках Discord-сервера и их ролях.

- *role* – название роли для удаления.

В методе класса происходит удаление пользователей, у которых есть роль, соответствующая названию, переданному в *role*.

Пример использования слэш-команды:

1. Пользователь вводит команду `/delete_members_by_role` с указанием названия роли, по которой будут удаляться участники. Пример слэш-команды представлен на рис. 15.

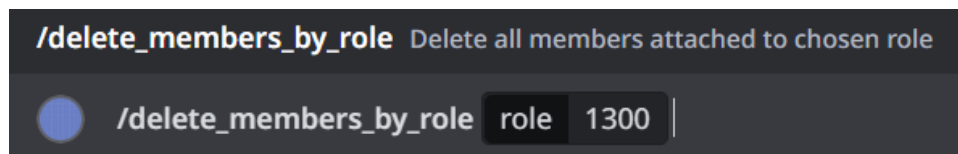


Рисунок 15 – Пример команды `/delete_members_by_role`

2. После отправки команды с Discord-сервера удаляются все пользователи с указанной ролью.

## GenerateMessage

Этот класс служит для генерации сообщения на защиту.

В классе присутствует одно поле *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове слэш-команды. Необходим для отправки сообщения в текстовый канал, получения названия текстового канала, получения имён участников Discord-сервера.
- *message\_type* – тип отправляемого сообщения.

- *repo\_name* – название репозитория на GitHub, с которого будет считываться информация о участниках, которые отправили лабораторную или курсовую работу.
- *work* – опциональный параметр, представляющий собой сокращённое название работы (lb1 – лабораторная работа №1; sw – курсовая работа). Если не указан, выводятся все работы, которые отправлены на GitHub.

В методе класса считывается заготовленное сообщение, находящееся в файле *settings.json*. Затем, с указанного в *repo\_name* репозитория считываются все PR. По полученным PR определяются фамилия, имя и тип работы студента через регулярное выражение. Фамилия, имя и номер группы студента объединяются в одну строку и по ней ищется данный студент на Discord-сервере. Если студент на сервере не найден (не присоединён к нему или подписан по-другому), в итоговом сообщении выводится полученная строка и лабораторная или курсовая работа; если студент есть на сервере (присоединён к нему и подписан правильно), в итоговом сообщении данный студент будет выведен с упоминанием.

Если никто из студентов не сделал выбранную работу, в Discord-канал выведется сообщение, что никто не сделал эту работу.

Пример использования слэш-команды:

1. Пользователь вводит команду */generate\_message* с указанием типа сообщения и названием репозитория. Пример слэш-команды представлен на рис. 16.

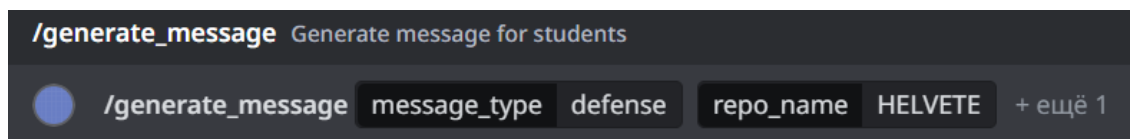


Рисунок 16 – Пример команды */generate\_message*

2. После отправки команды в Discord-канале выводится заданное сообщение с упоминанием студентов, которые сделали работу. Пример выводимого сообщения представлен на рис. 17.

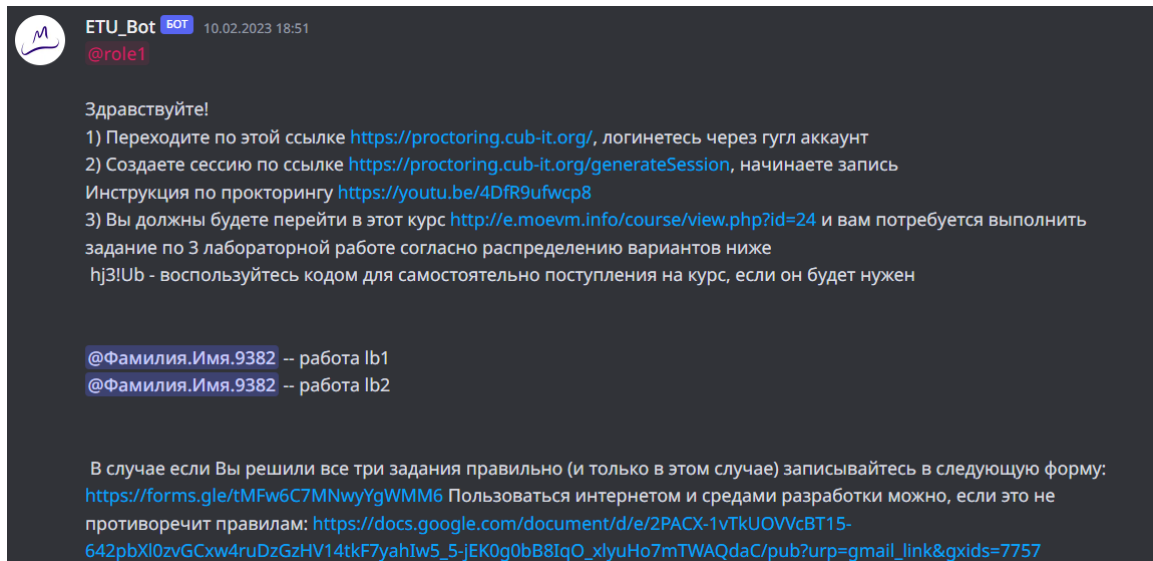


Рисунок 17 – Пример сообщения на защиту.

## GiveRole

Этот класс служит для выдачи определённой роли всем участникам Discord-сервера.

В классе присутствует одно поле *server\_id* – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод *self()*, принимающий следующие параметры:

- *interaction* – класс *nextcord.Interaction*, который передаётся от декоратора *nextcord.slash\_command* автоматически при вызове слэш-команды. Необходим для получения имён участников Discord-сервера.
- *role* – название роли, которую надо выдать всем пользователям.

В методе класса выдаётся выбранная роль всем участникам на Discord-сервере.

Пример использования слэш-команды:

1. Пользователь вводит команду `/give_role` с указанием роли, которую необходимо выдать. Пример слэш-команды представлен на рис. 18.

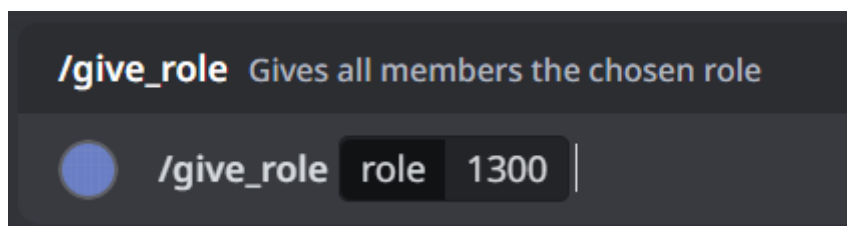


Рисунок 18 – команды `/give_role`

2. После отправки команды всем пользователям Discord-сервера будет выдана указанная роль.

## ShowEvents

Этот класс служит для вывода существующих событий всем участникам Discord-сервера.

В классе присутствует одно поле `server_id` – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод `self()`, принимающий следующие параметры:

- `interaction` – класс `nextcord.Interaction`, который передаётся от декоратора `nextcord.slash_command` автоматически при вызове слэш-команды.

В методе класса получают все события, сохранённые в объекте класса `DiscordBot`. Затем, в текстовый канал отправляются Embed сообщения с информацией о событиях.

Пример использования слэш-команды:

1. Пользователь вводит команду `/show_events`. Пример использования команды представлен на рис. 19.

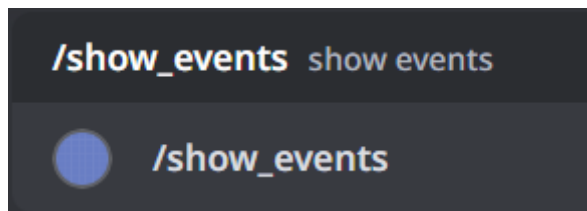


Рисунок 19 – Пример использования команды `/show_events`.

2. После отправки команды в Discord-канал выводятся Embed сообщения события. Пример Embed сообщения представлен на рис. 5.

### TrackMessage

Этот класс служит для отслеживания определённого сообщения, по реакции на которое пользователям будут выставляться роли.

В классе присутствует одно поле `server_id` – параметр, который содержит в себе идентификационный номер Discord-сервера, на котором будет работать эта команда.

В классе реализован метод `self()`, принимающий следующие параметры:

- `interaction` – класс `nextcord.Interaction`, который передаётся от декоратора `nextcord.slash_command` автоматически при вызове слэш-команды.
- `message_id` – идентификатор сообщения, которое нужно отслеживать.
- `roles` – список названий ролей.

В методе класса вызывается метод объекта класса `DiscordBot` `track_message` с параметрами `message_id` и `roles`.

Пример использования слэш-команды представлен выше в описании класса **DiscordManager**.

### 3.4 Веб-сервис

Одной из основных задач дипломного проекта было создание веб-сервиса для работы с Discord-ботом через него. Веб-сервис подразумевает собой создание сайта и сервера – программы, которая будет обрабатывать запросы с сайта, а также вызывать команды у Discord-бота.

Сайт написан на ЯП React.js.

Внешний вид сайта представлен на рис. 20.

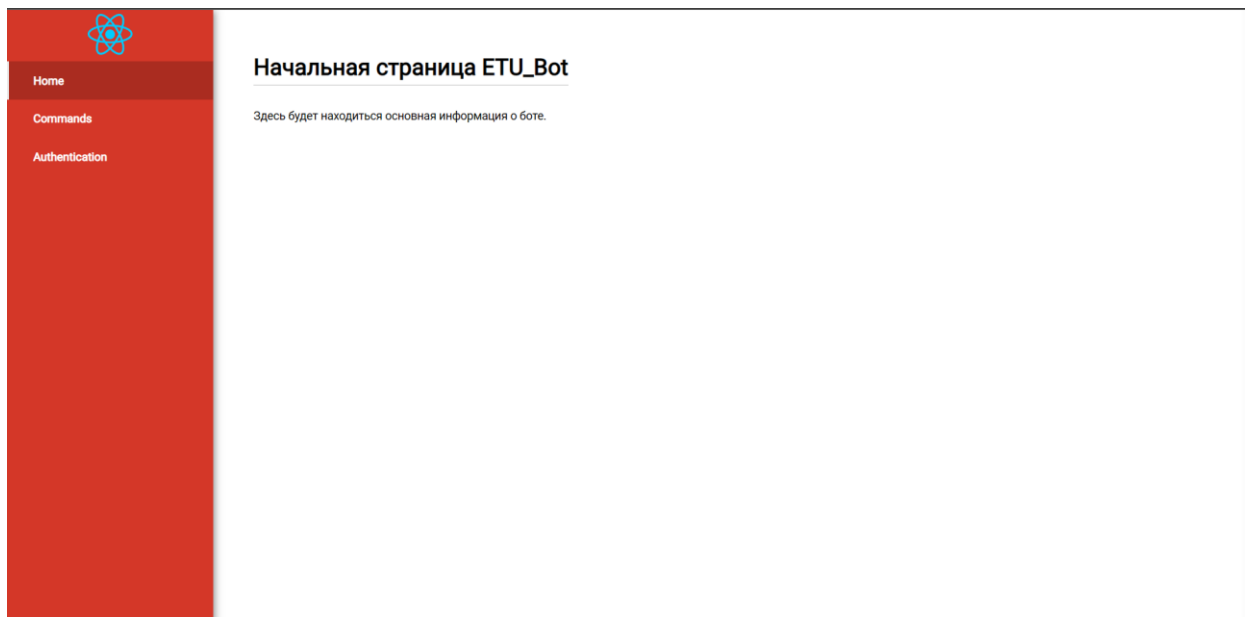


Рисунок 20 – Внешний вид сайта.

На сайте доступны следующие вкладки: «Информация о боте», «Команды» и «Аутентификация».

Внешний вид вкладки «Информация о боте» представлен на рис. 20.

Внешний вид вкладки «Аутентификации» представлен на рис. 21.

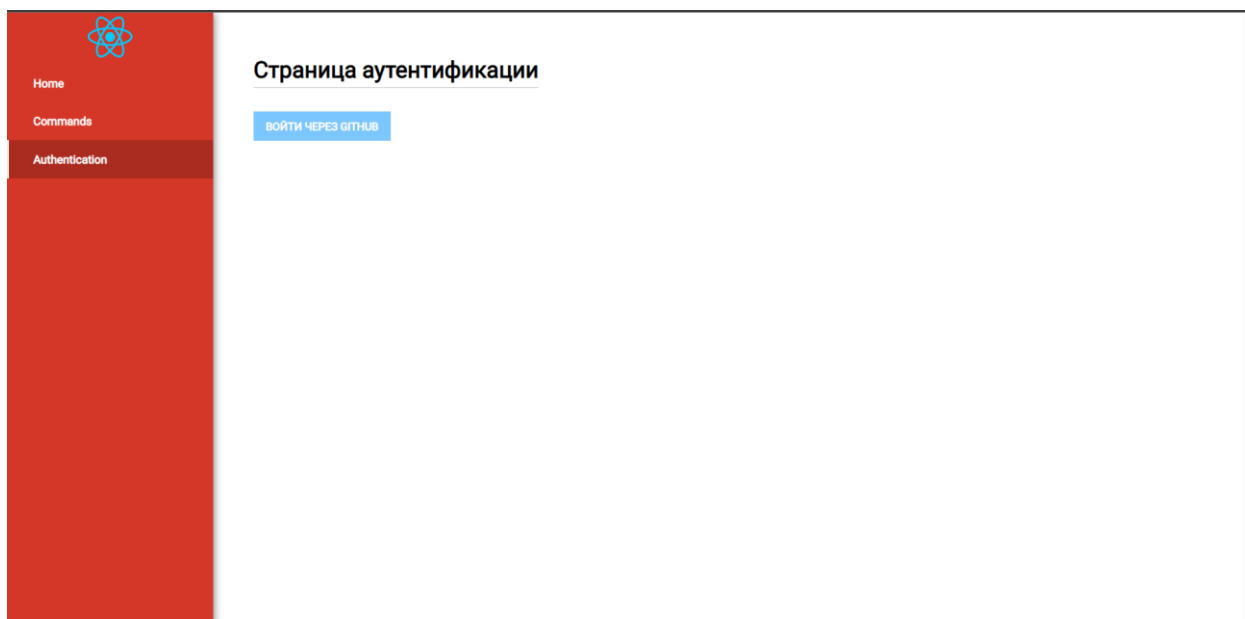


Рисунок 21 – внешний вид вкладки «Аутентификация»

Внешний вид вкладки «Команды» представлен на рис. 22.

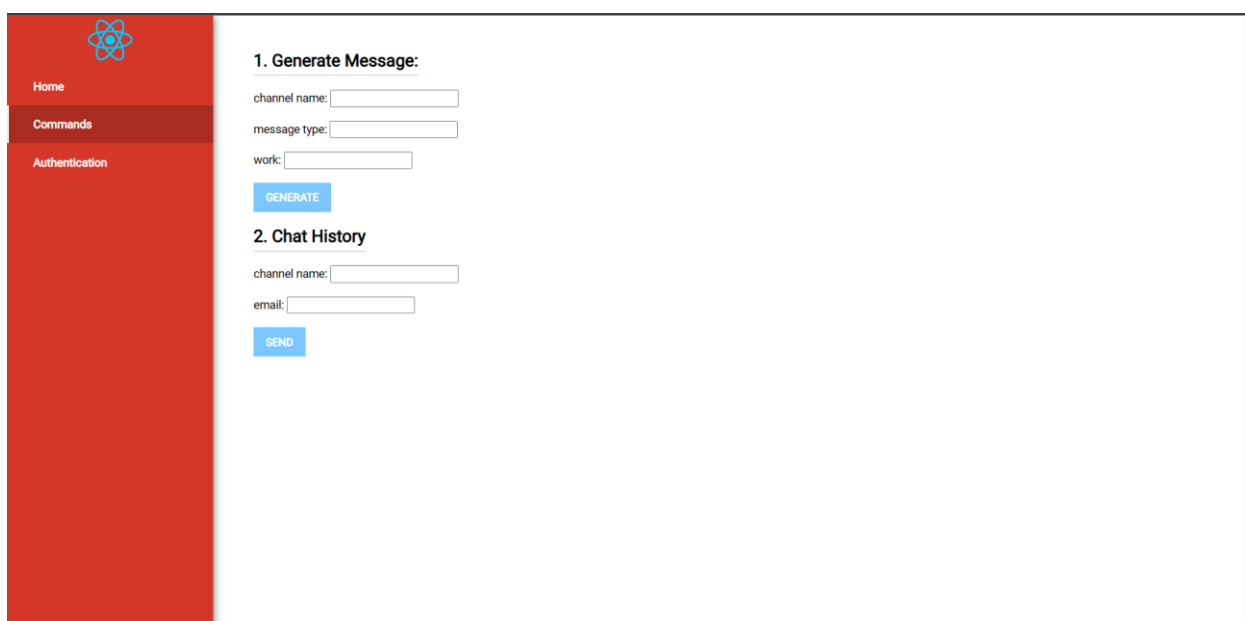


Рисунок 22 – внешний вид вкладки «Команды»

При первом посещении сайта пользователя автоматически переносит на страницу аутентификации, где он обязан авторизоваться через GitHub. Если пользователь состоит в группе преподавателей на GitHub, ему даётся возможность перейти к командам, которые он может использовать.

Вкладка «Команды» содержит в себе две функции: `Generate Message` и `Chat History`, что является отправкой сообщения на защиту студентам и истории сообщений соответственно.

При отправке сообщения на защиту есть три поля, которые необходимо заполнить:

1. *channel name* – название Discord-канала, куда отправится сообщение.
2. *message type* – тип сообщения.
3. *work* – сокращённое название работы (lb1 – лабораторная работа №1; sw – курсовая работа). Если не указан, выводятся все работы

После введения всех необходимых полей и нажатия кнопки «Generate» в указанный Discord-канал будет отправлено необходимое сообщение.

При отправке истории сообщений на почту есть два поля, которые необходимо заполнить:

1. *channel name* – название Discord-канала, куда отправится сообщение.
2. *email* – адрес электронной почты, на который необходимо отправить историю сообщений.

После введения всех необходимых полей и нажатия кнопки «Send» на указанную почту будет отправлена история сообщений выбранного Discord-канала.

Сервер написан на ЯП Python.

Для обработки запросов с сайта использовался фреймворк Quart. В файле сервера также создаётся клиент межпроцессного взаимодействия с тем же секретным ключом, что использовался в Discord-боте. Необходим для вызова команд Discord-бота через определённые запросы.

На сервере написано две функции: `generate_message()` и `chat_history()`.

`generate_message()`



Функция реализована через декоратор *Quart.route()*, принимающий на вход url, на котором находится переданная с сайта информация, а также методы передачи, которые поддерживает функция (POST, GET, OPTIONS)

Функция отправляет запрос Discord-боту через клиент межпроцессного взаимодействия с переданными ей параметрами *channel name*, *message type*, *work*, после чего Discord-бот отправляет сообщение на защиту на необходимый Discord-канал.

### **chat\_history()**

Функция реализована через декоратор *Quart.route()*, принимающий на вход url, на котором находится переданная с сайта информация, а также методы передачи, которые поддерживает функция (POST, GET, OPTIONS)

Функция отправляет запрос Discord-боту через клиент межпроцессного взаимодействия с переданными ей параметрами *channel name*, *email*, после чего Discord-бот отправляет электронное сообщение на переданный электронный адрес с файлом, содержащим историю сообщений указанного Discord-канала.

## **4 Обеспечение качества разработки, продукции, программного продукта**

### **4.1 Определение потребителей**

Потребителями данной разработки являются преподаватели ЛЭТИ, которые осуществляют свою деятельность на Discord-сервере «MOEVM».

### **4.2 Функция продукции и услуги**

При приобретении продукта или услуги потребителя интересуют в первую очередь полезные свойства продукции, т.е. функции, которые получает потребитель.

Функции изделия или услуги – это требования и ожидания потребителя, которые могут быть установлены, предполагаются или являются обязательными.

Функции Discord-бота:

- Администрирование Discord-сервера (установка ролей участникам, удаление пользователей по роли, генерация новых категорий);
- Уведомление участников о событиях (генерация первого сообщения на защиту, уведомление о приближающихся сроках сдачи работ, добавление мероприятий с началом занятия в календарь);
- Наличие приложения или веб-сервиса для работы с Discord-ботом;
- Распределение вариантов по лабораторным и курсовым работам;
- Наличие открытого исходного кода.

### **4.3 Качество и характеристики**

В соответствии с ГОСТ Р ИСО 9000-2015 качество – степень соответствия совокупности присущих характеристик объекта требованиям.

Для сопоставления функций с соответствующими требованиями для разработанной модели были рассмотрены стандарты, отраслевые требования и лучшие практики.

Анализ функциональных требований к разработке представлен в таблице 2.

Таблица 2 – Анализ функциональных требований к разработке

Функции по группам	Источник	Требования
Функциональная пригодность	ГОСТ Р ИСО/МЭК 25010-2015	Корректная работа функций с переданными параметрами
Уровень производительности	ГОСТ Р ИСО/МЭК 25010-2015	Discord-бот должен реагировать на команды в пределах 400 мс
Удобство использования	ГОСТ Р ИСО/МЭК 25010-2015	Web-сервис для бота должен быть интуитивно понятен и лёгок в эксплуатации; консольные команды для бота должны быть понятны по названию и минималистичны: включать в себя минимальное количество параметров для корректной работы

Продолжение таблицы 2

Функции по группам	Источник	Требования
Защищённость	ГОСТ Р ИСО/МЭК 25010-2015	Доступ к Web-сервису должен быть только у преподавателей; консольные команды могут использовать только преподаватели и администраторы Discord-сервера.
Переносимость	ГОСТ Р ИСО/МЭК 25010-2015	Docker-образ должен запускаться на сервере или может быть запущен на ПК на любой системе

#### 4.4 Измерение характеристик качества

Менеджмент качества возможен только при наличии информации о важнейших для организации объектах: процессах, продукции и услугах, которые они выпускают и др. Измерение – важнейший источник этой информации. Невозможно улучшать процессы и их результаты без измерений.

Операциональное определение (ОО) – это уточнение значения того или иного термина применительно к данной системе, находящейся в конкретных условиях и для людей, в ней задействованных.

ОО должно содержать как минимум три компоненты:

1. требования или стандарт, относительно которого оценивается результат измерения или испытания (критерий);
2. метод испытания или процедура измерения свойства объекта (тест),

3. процедура принятия решения (анализ), которое показывает, соответствует ли результат испытания стандарту.

Операциональные определения представлены в таблице 3.

Таблица 3 – операциональные определения

Требование	Тест		Решение (правило)		
	Характеристика	Метод измерения	Соответствие	Частичное соответствие	Несоответствие
Администрирование Discord-сервера: установка ролей участникам	Соответствие ожидаемого результата выполнения функции с действительным	Использование функции на больших объёмах данных (студентов)	Всем участникам роли установлены в соответствии с выбранными	Роли устанавливаются только тем участникам, у которых нет ролей; Роли участника заменяются на выбранную	Роли участникам не устанавливаются
Администрирование Discord-сервера: удаление пользователей по выбранной роли	Соответствие ожидаемого результата выполнения функции с действительным	Использование функции на больших объёмах данных (студентов)	Удаляются все пользователи, у которых установлена выбранная роль	Участники не удаляются, если у них присутствует несколько ролей	Участники не удаляются

Требование	Тест		Решение (правило)		
	Характеристика	Метод измерения	Соответствие	Частичное соответствие	Несоответствие
Уведомление участников о событиях	Соответствие ожидаемого результата выполнения функции с действительным	Использование функции на больших объёмах данных (студентов)	Все необходимые участники упоминаются с звуковым уведомлением	Упоминаются только те участники, у которых правильно указаны имя, фамилия и группа.	Никто из участников не упоминается
Наличие приложения или веб-сервиса для работы с Discord-ботом	Время обработки запросов с сервера к Discord-боту	Анализ времени обработки запросов	Все запросы обрабатываются с задержкой не более 400 мс	Запросы обрабатываются с задержкой более 400 мс	Запросы не обрабатываются
Распределение вариантов по лабораторным и курсовым работам	Соответствие ожидаемого результата выполнения функции с действительным	Использование функции на больших объёмах данных (студентов)	Всем студентам выданы правильные варианты лабораторной и курсовой работы	Студентам выдан хотя бы один правильный вариант работы	Студентам не выданы варианты лабораторной и курсовой работы

#### **4.5 Выводы**

Были определены потребители разработки и ее функции. Также составлены операциональные определения, по которым можно определить соответствие разработки требованиям и оценить качество разработанного приложения.



## ЗАКЛЮЧЕНИЕ

В результате выполнения работы было разработано приложение Discord-бота, которое включает в себя самого бота, сайт для работы с ним и сервер, обрабатывающий запросы с сайта. Код разработанного приложения представлен на сервисе GitHub [19]

В ходе работы были выполнены следующие задачи:

- Выявлены необходимые требования к разрабатываемому Discord-боту.
- Проанализированы существующие Discord-ботов для автоматизированного управления Discord-сервером и ведения учебного процесса.
- Найдены технологии, пригодные для создания Discord-бота и выполнения поставленных задач.
- Создание Discord-бота, удовлетворяющего поставленным требованиям.

Процесс создания Discord-бота был разделён на три части:

1. Разработка основного класса Discord-бота.
2. Создание вспомогательных классов для корректной работы Discord-бота и уменьшения ответственности класса за выполнение задач.
3. Создание слэш-команд для работы с Discord-ботом через приложение Discord.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Welcome to nextcord [Электронный ресурс]. URL: <https://docs.nextcord.dev/en/stable/> (дата обращения: 01.05.2023).
2. Quart [Электронный ресурс]. URL: <https://github.com/pallets/quart> (дата обращения: 01.05.2023).
3. PyGithub [Электронный ресурс]. URL: <https://github.com/PyGithub/PyGithub> (дата обращения: 14.02.2023).
4. Модуль os [Электронный ресурс]. URL: <https://pythonworld.ru/moduli/modul-os.html> (дата обращения: 10.02.2023).
5. Модуль json [Электронный ресурс]. URL: <https://pythonworld.ru/moduli/modul-json.html> (дата обращения: 14.02.2023).
6. Nextcord Extensions [Электронный ресурс]. URL: <https://docs.nextcord.dev/en/stable/ext/commands/extensions.html> (дата обращения: 02.05.2023).
7. SMTP protocol client [Электронный ресурс]. URL: <https://docs.python.org/3/library/smtplib.html> (дата обращения: 04.02.2023).
8. Map filenames to MIME types [Электронный ресурс]. URL: <https://docs.python.org/3/library/mimetypes.html> (дата обращения: 04.02.2023).
9. Работа с почтой – модули email/smtplib в Python [Электронный ресурс]. URL: <https://python-scripts.com/send-email-smtp-python> (дата обращения: 04.02.2023).
10. Regular expression operations [Электронный ресурс]. URL: <https://docs.python.org/3/library/re.html> (дата обращения: 14.02.2023).
11. Ics.py : iCalendar for Humans [Электронный ресурс]. URL: <https://icspy.readthedocs.io/en/stable/> (дата обращения: 01.05.2023).

12. DiscordAPI [Электронный ресурс]. URL: <https://discord.com/developers/docs/intro> (дата обращения: 01.05.2023).
13. MEE6 Discord Bot [Электронный ресурс]. URL: <https://mee6.xyz/en> (дата обращения: 01.12.2022).
14. Juniper Discord Bot [Электронный ресурс]. URL: <https://juniper.bot/> (дата обращения: 01.12.2022).
15. Dyno Discord Bot [Электронный ресурс]. URL: <https://dyno.gg/bot> (дата обращения: 01.12.2022).
16. Maki Discord Bot [Электронный ресурс]. URL: <https://maki.gg/> (дата обращения: 01.12.2022).
17. Bosley Discord Bot [Электронный ресурс]. URL: <https://top.gg/bot/541834279156711434> (дата обращения: 01.12.2022).
18. ProBot Discord Bot [Электронный ресурс]. URL: <https://probot.io/ru> (дата обращения: 01.12.2022).
19. Разработанное приложение [Электронный ресурс]. URL: [https://github.com/moevm/bsc\\_ryzhih](https://github.com/moevm/bsc_ryzhih)