**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інженерії програмного забезпечення**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

*підпис*

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

# Програмне забезпечення керування ботом Discord

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.2011022

***Здобувач*** _____А. М. Ткач

*підпис*

«__» _____ 2024 р.

***Керівник*** PhD, ст викладач _____ І. О. Кандиба

*підпис*

«__» _____ 2024 р.

**Консультант** канд. техн. наук, доцент _____ А. О. Алексєєва

*підпис*

«__» _____ 2024 р.

**Миколаїв – 2024**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інженерії програмного забезпечення**

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«_____» _____ 2024 р.

**ЗАВДАННЯ**

**на виконання кваліфікаційної роботи бакалавра**

Видано здобувачу групи 408 факультету комп'ютерних наук

_____Ткач Артем Максимович_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

Програмне забезпечення керування ботом Discord_____

_____

Затверджена наказом по ЧНУ від «22» _____грудня_____ 2023 р. № ___269_____

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є програмне забезпечення керування ботом Discord_____

_____

4. Перелік питань, що підлягають розробці:

- дослідження предметної області та аналіз існуючих аналогів;
- формування специфікації вимог до програмного забезпечення;
- визначення архітектури для проектування програмного забезпечення;
- моделювання та проектування програмного забезпечення;
- розробка програмного забезпечення;
- здійснення тестування роботи програмного забезпечення;

5. Перелік графічних матеріалів

__Презентація_____

6. Завдання до спеціальної частини

__Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій__

7. Консультанти:

| Консультант | Кафедра (організація) | Частина роботи |
|---|---|---|
| Алєксєєва А. О. | Кафедра екології | Спеціальна частина з охорони праці |

Керівник роботи _____ PhD, ст викладач Кандиба І. О. _____
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

_____Ткач А. М. _____
(прізвище, ім'я, по батькові)

_____
(підпис)

Дата видачі завдання «____» _____ 2024р.

**КАЛЕНДАРНИЙ ПЛАН**
**виконання бакалаврської кваліфікаційної роботи**

Тема: «Програмне забезпечення керування ботом Discord»

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1 | Розробка та затвердження завдання на виконання КРБ | 20.02.2024 | 23.02.2024 | Виконано |
| 2 | Огляд літератури за темою роботи | 24.02.2024 | 26.02.2024 | Виконано |
| 3 | Складання календарного плану КРБ | 26.02.2024 | 27.02.2024 | Виконано |
| 4 | Аналіз предметної області | 28.02.2024 | 01.03.2024 | Виконано |
| 5 | Розробка проектних рішень | 04.03.2024 | 10.03.2024 | Виконано |
| 6 | Моделювання та конструювання ПЗ | 22.03.2024 | 26.03.2024 | Виконано |
| 7 | Кодування ПЗ, тестування та апробація розробленого ПЗ, аналіз результатів тестування | 28.03.2024 | 22.05.2024 | Виконано |
| 8 | Розробка спеціальної частини з охорони праці | 22.05.2024 | 24.05.2024 | Виконано |
| 9 | Оформлення КРБ та презентації | 25.05.2024 | 03.06.2024 | Виконано |
| 10 | Відгук керівника КРБ | 03.06.2024 | 04.06.2024 | Виконано |
| 11 | Попередній захист | 03.06.2024 | 03.06.2024 | Виконано |
| 12 | Рецензування | 13.06.2024 | 14.06.2024 | Виконано |
| 13 | Захист кваліфікаційної роботи | 25.06.2024 | 26.06.2024 | Виконано |

Розробив здобувач _____Ткач А. М._____ _____
*(прізвище, ім'я, по батькові)* *(підпис)*
«24» січня 2024 р.

Керівник роботи PhD, ст викладач Кандиба І. О. _____
*(посада, прізвище, ім'я, по батькові)* *(підпис)*
«24»січня 2024 р.

# АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Програмне забезпечення керування ботом Discord»

Здобувач 408 гр,: Ткач Артем Максимович

Керівник: PhD, ст викладач Кандиба Ігор Олександрович.

Кваліфікаційна робота присвячена розробці програмного забезпечення керування ботом Discord, що відповідає сучасним вимогам інтернет-технологій та потребам користувачів на серверах у Discord.

Об'єкт кваліфікаційної роботи: процес розробки боту для месенджеру Discord.

Предмет кваліфікаційної роботи: інструментарій розробки боту для месенджеру Discord.

Метою кваліфікаційної роботи є розробка програмного забезпечення керування ботом Discord для спрощення управління серверами зазначеного месенджеру.

У першому розділі кваліфікаційної роботи представлено аналіз предметної області, аналіз аналогів та специфікацію вимог.

У другому розділі представлено створення діаграм варіантів використання та послідовностей

У третьому розділі представлено розробку діаграм класів, моделі бази даних та вибір засобів розробки.

У четвертому розділі представлено реалізацію бота та ручне тестування його функцій.

КРБ викладена на 64 сторінки, вона містить 4 розділи, 33 ілюстрацій, 11 таблиці,  20 джерел в переліку посилань

Ключові слова: бот, Discord Api, Python, Disnake.py, база даних, SQLite3

**ABSTRACT**

to the qualification work of the bachelor

"Discord Bot Management Software"

Student 408 gr,: Tkach Artem Maksymovich

Supervisor: PhD, Senior Lecturer Kandyba I.O.This thesis is devoted to the development of Discord bot management software that meets the modern requirements of Internet technologies and the needs of users on Discord servers.

The object of the qualifying work: the process of developing a bot for the Discord messenger.

The subject of the qualification work: a toolkit for developing a bot for the Discord messenger.

The purpose of the qualification work is the development of Discord bot management software to simplify the management of the servers of the specified messenger.

The first section of the qualification work presents the analysis of the subject area, the analysis of analogues and the specification of requirements.

The second section introduces the creation of use case and sequence diagrams

The third chapter presents the development of class diagrams, database models, and the selection of development tools.

The fourth chapter presents the implementation of the bot and manual testing of its functions.

QWB is laid out on 64 pages, it contains 4 sections, 33 illustrations, 11 tables, 20 sources in the list of references

Keywords: bot, Discord Api, Python, Disnake.py, database, SQLite3

# ЗМІСТ

# ВСТУП

На сьогоднішній день спостерігається тенденція зростання кількості молодих людей у віці від 14 до 25 років, які витрачають свій вільний час в соціальних мережах та месенджерах. Серед популярних платформ можна виділити Discord, Telegram, Instagram та Facebook. У цьому дослідженні особлива увага приділяється Discord.

Discord-сервер є засобом знаходження друзів, товаришів по грі або однодумців на різні теми. Тут можна намагатися знайти собі партнера для гри у відеоігри. Кожен сервер має свою ієрархію ролей та посад, які встановлюють правила поведінки на сервері. Зазвичай кількість правил не велика і вони можуть відрізнятися на різних серверах. Серед найпопулярніших ролей можна виділити адміністратора, модератора, редактора чату та підтримку. Також існують поняття економіки та подій, що дозволяють учасникам серверу цікаво проводити час.

Об'єкт кваліфікаційної роботи: процес розробки боту для месенджеру Discord.

Предмет кваліфікаційної роботи: інструментарій розробки боту для месенджеру Discord.

Метою кваліфікаційної роботи є розробка програмного забезпечення керування ботом Discord для спрощення управління серверами зазначеного месенджеру.

Для досягнення поставленої мети треба виконати наступні завдання:

- дослідження предметної галузі;
- аналіз аналогічних ботів для Discord;
- моделювання програмного забезпечення керування ботом Discord;
- проєктування зазначеного ПЗ;
- розробка програмного забезпечення керування ботом Discord;
- тестування розробленого ПЗ.

# 1 АНАЛІЗ СУЧАСНИХ ЧАТБОТІВ

## 1.1 Дослідження чатботів

Система адміністрування, що є головною в Discord ботах для публічних серверів, розподілена на кілька частин:

1. куратори відповідають за керування користувачами та їхніми можливостями;
2. модератори проводять час у голосових каналах, де забезпечують порядок та адекватність спілкування, а також реагують на конфлікти;
3. редактори чату контролюють дотримання правил серверу та платформи Discord у текстових каналах;
4. охоронці відповідають за верифікацію користувачів на сервері та їхнє допущення, а також ведуть базу даних користувачів.

Кожна частина модуля має власний функціонал:

– модуль охоронців дозволяє проводити верифікацію користувачів, зміну гендерної ролі та блокування користувачів за допомогою команд;

– модуль редакторів чату забезпечує можливість надавати застереження, накладати обмеження на можливість писати повідомлення у чаті, а також дозволяє переглядати історію обмежень;

– модуль модераторів надає функції зміни гендерної ролі, накладання застережень, обмежень на повідомлення, обмежень на використання мікрофону та блокування користувачів;

– у модулі кураторів можна додавати та видаляти користувачів з гілок адміністрування, а також переглядати список користувачів.

Ця система дозволяє ефективно керувати сервером та забезпечувати порядок і безпеку на ньому.

Система управління економікою є ключовою складовою кожного сервера. Цей модуль надає можливість учасникам отримувати невелике повернення грошей

у формі валюти сервера, яка є його основою. Валюту можна отримати різними способами:

- участь в онлайн-активностях на сервері;
- виконання різних завдань та посадових обов'язків на сервері;
- гра в ігри, доступні на сервері;
- участь у заходах та подіях, організованих на сервері;
- отримання щоденного подарунка.

Коли учасник накопичує достатню кількість валюти, він може витратити її на:

- створення власної ролі на сервері;
- створення власної голосової кімнати на сервері;
- участь у іграх разом з іншими учасниками сервера;
- передачу валюти іншим учасникам за їхнім бажанням.

Ця система має декілька основних функцій:

- видача та списання валюти учасників;
- передача валюти між учасниками;
- управління участю у подіях та винагородами за них;
- можливість перегляду профілю учасників;
- створення та видалення ролей на сервері;
- доступ до магазину ролей.

Система управління подіями передбачає організацію різноманітних заходів для учасників сервера. Для зручного керування цими подіями створюється спеціальна категорія каналів на сервері, а також канал для керування цією категорією. Керівники подій можуть створювати, редагувати та завершувати події, обмежувати доступ до них та видає винагороди за участь чи перемогу.

Кожен керівник подіями може мати лише одну активну подію одночасно, що дозволяє йому керувати подіями ефективно.

Ця система надає можливість ефективно управляти економікою та організовувати заходи на сервері, що забезпечує підвищення активності та зацікавленість учасників.

Цей модуль застосунку буде користуватися як ведучі подій (блок подій), так і звичайні користувачі (блок економіки).

Роль адміністратора наділяє людину правами адміністрування серверу. Вона може редагувати гілки посад на сервері та користуватися усіма командами на ньому.

Ведучий подій відповідає за проведення заходів на сервері і має доступ до наступних команд: створення події, редагування її налаштувань, видача нагород за участь у ній та закінчення.

Учасниками серверу є всі користувачі, які допущені до нього через охоронців, та мають доступ до участі у подіях.

Планується розробити такі команди для модуля подій:

- створення подій;
- зміна кількості учасників події;
- зміна режиму доступу;
- видалення події;
- видача винагород за участь чи перемогу;
- закінчення події ведучим.

А для модуля економіки планується такі команди:

- відкриття профілю користувача;
- перегляд балансу;
- передача серверної валюти;
- видалення всієї валюти;
- видача серверної валюти;
- створення користувацької ролі;
- видалення користувацької ролі;
- відображення магазину ролей;
- запуск гри "орел-решка".

## 1.2 Аналіз аналогів

Однією з основних проблем української discord спільноти є недостатня кількість українських серверів.

Ця проблема стала особливо актуальною після подій 2022 року. Після цих подій українська мова та спільнота почали відчувати ще більший тиск з боку адміністрацій російськомовних серверів. Багато з відомих серверів навіть накладали покарання за використання української мови у голосових каналах.

Кожен discord сервер має свій набір функцій. Для зручного керування цими функціями створено різні бібліотеки.

На відміну від сайтів чи застосунків з відкритим кодом, код бота у discord не є відкритим, тому аналізувати його можна лише через доступні команди.

Сервер у discord можна розглядати як свого роду веб-інтерфейс або консоль. Для розуміння всього його функціоналу потрібно детально проаналізувати сам сервер, на якому працює бот.

Більшість сайтів з моніторингу discord серверів є російськомовними.

Таблиця 1.1 – Основні характеристики існуючого ПЗ

| Назва серверу | SquadUkraine | UFamily | Mikone |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| Країна | Україна | Україна | Україна |
| Виробник | Приватна особа | Приватна особа | Приватна особа |
| Архітектура | Мікросервісна | Мікросервісна | Мікросервісна |
| Мова реалізації | Python, JavaScript | Python, JavaScript | Python, JavaScript |
| Основні функції | Модуль адміністрації<br>Модуль сторінки учасника<br>Модуль власних кімнат | Модуль адміністрації<br>Система економіки<br>Модуль подій<br>Модуль профілю<br>Модуль любовних кімнат | Модуль адміністрації<br>Система економіки<br>Модуль ігор<br>Модуль подій<br>Модуль сторінки учасника |

Продовження таблиці 1.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| Переваги | Мікросервісна архітектура, що сприяє масштабованості та гнучкості.<br>Наявність модулів адміністрації, сторінки учасника та власних кімнат. | Різноманітність посад на сервері, включаючи кураторів, редакторів та інші.<br>Можливість використання російської мови у голосових каналах.<br>Відсутність реклами. | Мікросервісна архітектура.<br>Наявність різноманітних модулів, включаючи ігри та події.<br>Наявність резервного збереження даних на хостингу та за допомогою Google Drive. |
| Недоліки | Присутність реклами.<br>Відсутність конвертації реальної валюти у серверну.<br>Відсутність резервного збереження даних. | Нижча кількість користувачів порівняно з іншими серверами (2700).<br>Відсутність резервного збереження даних. | Нижча кількість користувачів (1330).<br>Відсутність конвертації реальної валюти у серверну. |
| Джерело інформації | https://discord.com/invite/squa | https://discord.gg/rNkzGVZj | https://discord.gg/DTEdNuQh |

У результаті аналізу трьох серверів Discord, створених приватними особами в Україні, було визначено, що кожен сервер використовує бота, який має мікросервісну архітектуру та реалізований на мовах програмування Python і JavaScript. Ці боти забезпечують різноманітні функції та мають певні переваги і недоліки, які можна врахувати під час розробки нового боту або вдосконалення існуючого.

### 1.3 Специфікація вимог

Програмне забезпечення керування ботом Discord призначене для створення та налаштування ботів на платформі Discord з метою автоматизації та полегшення управління серверами.

**1) Функціональні вимоги:**

**а) Реєстрація та авторизація**

– кистема повинна забезпечувати можливість реєстрації нових користувачів;

– користувачі повинні мати змогу авторизуватися в системі за допомогою облікового запису Discord.

**б) Створення бота**

– користувачі повинні мати можливість створювати нових ботів для використання на своїх серверах Discord;

– створення бота повинно включати в себе встановлення основних параметрів, таких як ім'я, аватар, права доступу тощо.

**в) Налаштування бота**

– система повинна надавати можливість користувачам налаштовувати функціональність свого бота шляхом встановлення параметрів та обмежень;

– налаштування може включати в себе призначення команд, автоматизацію дій, реакції на певні події та інші опції.

**г) Управління серверами**

– користувачі повинні мати можливість призначати свого бота для управління конкретними серверами Discord;

– система повинна надавати інтерфейс для керування налаштуваннями бота на рівні окремого сервера.

**д) Моніторинг та журналювання**

– система повинна забезпечувати можливість моніторингу активності ботів, включаючи статистику виконаних команд та подій;

– повинна бути реалізована можливість журналювання дій ботів для налагодження та аналізу їхньої роботи.

**2) Нефункціональні вимоги:**

**а) Безпека**

– система повинна забезпечувати безпечний доступ до функціональності, використовуючи механізми аутентифікації та авторизації;

– мають бути реалізовані заходи для захисту від зловживання та недозволених дій з боку користувачів.

**б) Продуктивність**

– система повинна забезпечувати швидкий доступ до функціональності та мінімальний час відгуку на запити користувачів;

– навантаження на сервер повинно бути оптимізоване для ефективної роботи під час одночасної обробки багатьох запитів.

Програмне забезпечення керування ботом Discord має за мету надати користувачам зручний інструмент для створення, налаштування та управління ботами на платформі Discord. Виконання всіх вимог, вказаних у цій специфікації, дозволить забезпечити стабільну та безпечну роботу системи з високою продуктивністю та швидкодією.

**Висновки до розділу 1**

В першому розділі кваліфікаційної роботи бакалавра розглянуто дискорд бота, який пропонує зручний підхід до адміністрування дискорд серверів.

Описано функціонал дискорд боту спрямованого на адміністрування серверів.

В аналізі існуючих систем також виявлено ключові аспекти та вдалі рішення, які можна врахувати під час розробки власної платформи. Проаналізовані сервери SquadUkraine, UFamily та Mikone як аналоги на ринку програмного забезпечення для адміністрування серверів.

# 2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ БОТУ DISCORD

## 2.1 Створення діаграми варіантів використання

Діаграма варіантів використання – діаграма, на якій зображено відношення між акторами та прецедентами в системі.

Розглянемо кожний модуль бота окремо, створивши діаграми варіантів використання.

Діаграма варіантів використання для бота керуючого економікою представлено на рисунку 2.1.

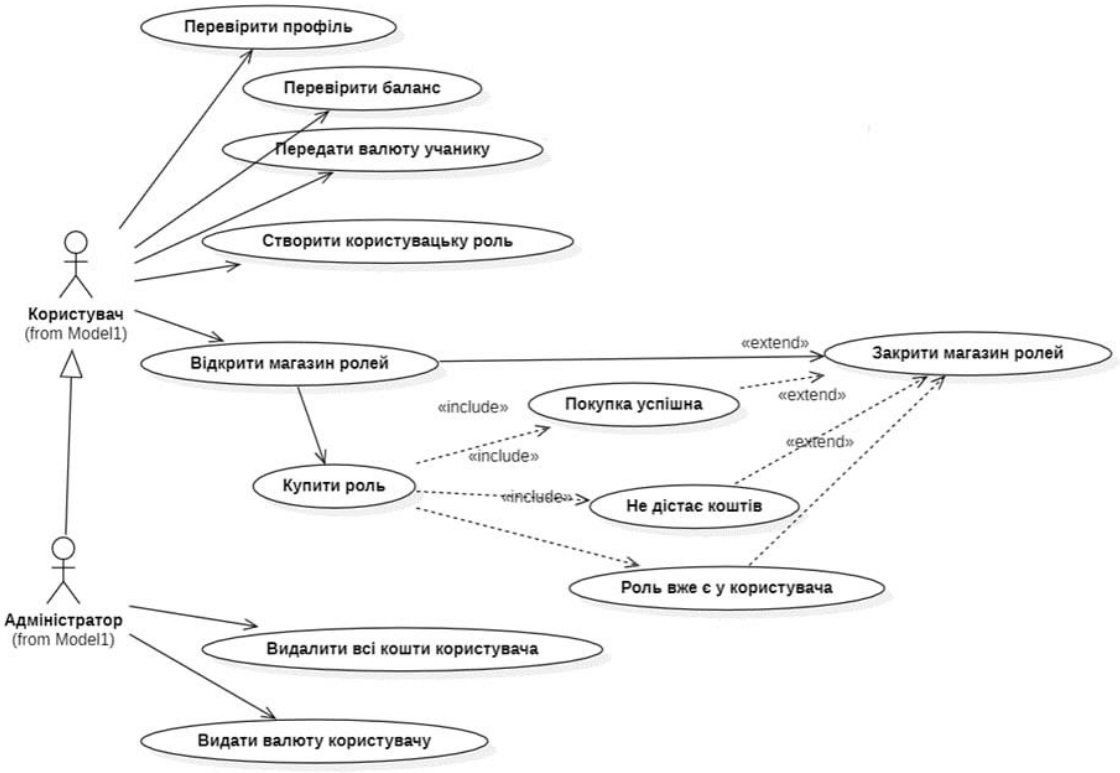


Рисунок 2.1 – Діаграма варіантів використання бота керуючого економікою

*Сценарій 1: Перевірити профіль*

Короткий опис: Даний варіант використання дозволяє користувачу перевірити свій профіль чи профіль іншого користувача.

Передумови: Відсутні.

Основний потік подій:

1) користувач заходить до чату;
2) користувач вводить команду /profile;

3) система виводить дані профілю користувача.

Альтернативний потік подій: Не визначений.

Постумови: Не визначені.

*Сценарій 2: Передати валюту користувачу*

Короткий опис: Даний варіант використання дозволяє користувачу передати серверну валюту іншому користувачу.

Передумови: Відсутні.

Основний потік подій:

1) користувач заходить до чату;
2) користувач вводить команду /give;
3) користувач вводить дані іншого користувача;
4) користувач вводить кількість монет, які хоче передати;
5) система передає валюту іншому користувачу.

Альтернативний потік подій:

1) якщо на рахунку користувача недостатньо коштів, система видає помилку та не проводить операцію передачі валюти.

Постумови: Не визначені.

*Сценарій 3: Створити користувацьку роль*

Короткий опис: Даний варіант використання дозволяє користувачу створити особисту роль на сервері.

Передумови: Відсутні.

Основний потік подій:

1) користувач заходить до чату;
2) користувач вводить команду /create_role;
3) користувач вводить назву ролі;
4) користувач вводить ціну ролі;
5) користувач вводить колір ролі;
6) система створює роль.

Альтернативний потік подій:

1) якщо роль з такою назвою вже існує, користувач не може її створити.

Постумови: Не визначені.

*Сценарій 4: Перевірити баланс*

Короткий опис: Даний варіант використання дозволяє користувачу переглянути свій баланс серверної валюти чи баланс іншого користувача на сервері.

Передумови: Відсутні.

Основний потік подій:

1) користувач заходить до чату;
2) користувач вводить команду /balance;
3) система відправляє користувачу повідомлення з його балансом або балансом зазначеного користувача.

Альтернативний потік подій: Не визначений.

Постумови: Не визначені.

*Сценарій 5: Відкрити магазин ролей*

Короткий опис: Даний варіант використання дозволяє користувачу відкрити магазин та купити роль на сервері за серверну валюту.

Передумови: Якщо у користувача вже існує певна роль, вона у магазині буде не активна.

Основний потік подій:

1) користувач заходить до чату;
2) користувач вводить команду /shop;
3) користувач переглядає сторінки магазину;
4) користувач купує вибрану роль.

Альтернативний потік подій:

1) якщо у користувача немає коштів для купівлі ролі, система виводить повідомлення про недостатність коштів.

Постумови: Не визначені.

*Сценарій 6: Закрити магазин ролей*

Короткий опис: Даний варіант використання дозволяє користувачу закрити магазин.

Передумови: Користувач повинен відкрити магазин ролей.

Основний потік подій:

1) користувач заходить до чату;
2) користувач вводить команду /shop;
3) користувач переглядає сторінки магазину;
4) якщо користувач не знаходить відповідну роль, він закриває магазин.

Альтернативний потік подій:

1) якщо користувач знаходить роль, яка йому подобається, він її купує за наявності коштів.

Постумови: Не визначені.

*Сценарій 7: Видалити всі кошти користувача*

Короткий опис: Даний варіант використання дозволяє адміністратору очистити весь баланс серверних коштів певного користувача.

Передумови: Відсутні.

Основний потік подій:

1) адміністратор заходить до чату;
2) адміністратор вводить команду /clear_balance;
3) адміністратор вводить ідентифікатор користувача;
4) система очищає баланс користувача.

Альтернативний потік подій: Не визначений.

Постумови: Не визначені.

*Сценарій 8: Видати валюту користувачу*

Короткий опис: Даний варіант використання дозволяє адміністратору видати серверну валюту користувачу.

Передумови: Відсутні.

Основний потік подій:

1) адміністратор заходить до чату;
2) адміністратор вводить команду /give_currency;
3) адміністратор вводить ідентифікатор користувача;
4) адміністратор вводить кількість валюти для видачі;

5) система додає зазначену кількість валюти на баланс користувача.

Альтернативний потік подій: Не визначений.

Постумови: Не визначені.

Діаграма варіантів використання для бота керуючого подіями представлено на рисунку 2.2.

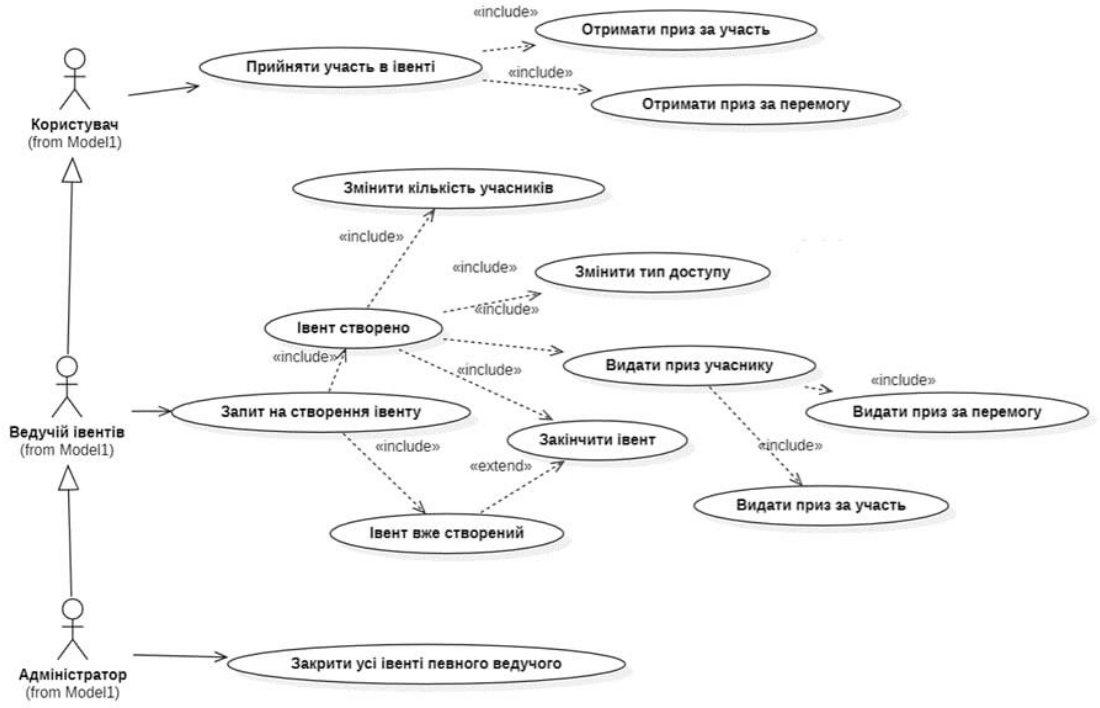


Рисунок 2.2 – Діаграма варіантів використання бота керуючого подіями

*Сценарій 1: Прийняти участь у події*

Короткий опис: Даний варіант використання дозволяє користувачу прийняти участь у події.

Передумови: Відсутні.

Основний потік подій:

1) користувач заходить до голосового каналу у категорії події до його початку;

2) користувач приймає участь у заході.

Альтернативний потік подій: Не визначені.

Постумови: Не визначені.

*Сценарій 2: Отримати приз за участь*

Короткий опис: Даний варіант використання дозволяє користувачу отримати приз за участь у події.

Передумови: Користувач повинен приймати участь у події.

Основний потік подій:

1) наприкінці заходу кожен учасник отримує приз за участь у події;

Альтернативний потік подій:

1) користувач може отримати приз за перемогу у події.

Постумови: Не визначені.

*Сценарій 3: Отримати приз за перемогу*

Короткий опис: Даний варіант використання дозволяє користувачу отримати приз за перемогу у події.

Передумови: Користувач повинен приймати участь у події.

Основний потік подій:

1) наприкінці заходу певний учасник отримує приз за перемогу у події.

Альтернативний потік подій:

1) користувач може отримати приз за участь у події.

Постумови: Не визначені.

*Сценарій 4: Запит на створення події*

Короткий опис: Даний варіант використання дозволяє ведучому створити запит на створення події.

Передумови: Відсутні.

Основний потік подій:

1) ведучій подій вводить команду /event_create;
2) ведучій обирає подію зі списку та створює його.

Альтернативний потік подій: Не визначені.

Постумови: Не визначені.

*Сценарій 5: Подію створено*

Короткий опис: Даний варіант використання дозволяє ведучому створити подію.

Передумови: Ведучій повинен створити запит на створення події.

Основний потік подій:

1) ведучій створює подію;

2) система перевіряє, чи є в ведучого вже створені події;

3) якщо створених подій немає, система створює нову подію.

Альтернативний потік подій:

1) якщо у ведучого є створені події, система надсилає повідомлення з проханням закінчити попередні події перед створенням нового.

Постумови: Не визначені.

*Сценарій 6:Подію вже створено*

Короткий опис: Даний варіант використання дозволяє ведучому створити подію.

Передумови: Ведучій повинен створити запит на створення подію.

Основний потік подій:

1) ведучій створює подію;

2) система перевіряє, чи є у ведучого вже створені події;

3) якщо у ведучого є створені події, система надсилає повідомлення з проханням закінчити попередні події.

Альтернативний потік подій:

1) якщо створених подій немає, система створює нову подію.

Постумови: Ведучій закінчує всі створені події.

*Сценарій 7: Змінити кількість учасників*

Короткий опис: Даний варіант використання дозволяє ведучому змінити кількість учасників події.

Передумови:Подія повинна бути створена.

Основний потік подій:

1) при створенні події ведучий обирає «змінити кількість учасників» на панелі налаштувань;

2) ведучий змінює кількість учасників відповідно до потреб.

Альтернативний потік подій: Не визначені.

Постумови: Не визначені.

*Сценарій 8: Змінити тип доступу*

Короткий опис: Даний варіант використання дозволяє ведучому змінити тип доступу з відкритого на закритий і навпаки.

Передумови: Ведучій повинен створити запит на створення події.

Основний потік подій:

1) тип доступу автоматично встановлюється як відкритий;
2) ведучий може закрити доступ учасникам при потребі.

Альтернативний потік подій:

1) якщо після закриття доступу хтось приєднується, ведучий може відкрити доступ і запросити нового учасника.

Постумови: Не визначено.

*Сценарій 9: Видати приз учаснику*

Короткий опис: Даний варіант використання дозволяє ведучому видати приз учаснику події.

Передумови: Ведучій повинен створити подію.

Основний потік подій:

1) після проведення події ведучий ініціює нагородження учасників.

Альтернативний потік подій: Не визначено.

Постумови: Не визначено.

*Сценарій 10: Видати приз за участь*

Короткий опис: Даний варіант використання дозволяє ведучому видати приз за участь учаснику події.

Передумови: Ведучій повинен створити подію.

Основний потік подій:

1) після проведення події ведучий ініціює нагородження учасників;
2) всі учасники, окрім переможця, отримують нагороду за участь.

Альтернативний потік подій:

1) переможець отримує нагороду за перемогу.

Постумови: Не визначено.

*Сценарій 11: Видати приз за перемогу*

Короткий опис: Даний варіант використання дозволяє ведучому видати приз за перемогу учаснику події.

Передумови: Ведучій повинен створити подію.

Основний потік подій:

1) після проведення події ведучий ініціює нагородження учасників;
2) переможець отримує нагороду за перемогу.

Альтернативний потік подій:

1) всі учасники, окрім переможця, отримують нагороду за участь.

Постумови: Не визначено.

*Сценарій 12: Закінчити подію*

Короткий опис: Даний варіант використання дозволяє ведучому закінчити подію.

Передумови: Ведучій повинен створити подію.

Основний потік подій:

1) після проведення події ведучий ініціює нагородження учасників;
2) переможець отримує нагороду за перемогу.

Альтернативний потік подій:

1) всі учасники, окрім переможця, отримують нагороду за участь.

Постумови: Не визначено.

Діаграми варіантів використання для бота адміністратора представлено на рисунку 2.3-2.5.


Рисунок 2.3 – Діаграма варіантів використання модулю модераторів


Рисунок 2.4 – Діаграма варіантів використання модулю редакторів чату


Рисунок 2.5 – Діаграма варіантів використання модулю охоронців

## 2.2 Створення діаграми послідовностей дій системи

Діаграма послідовності відображає взаємодію об'єктів впорядкованих за часом, зокрема, відображає задіяні об'єкти та послідовність відправлених повідомлень.

Діаграма послідовності для основного потоку подій прецеденту «Створення події» представлена на рисунку 2.6.

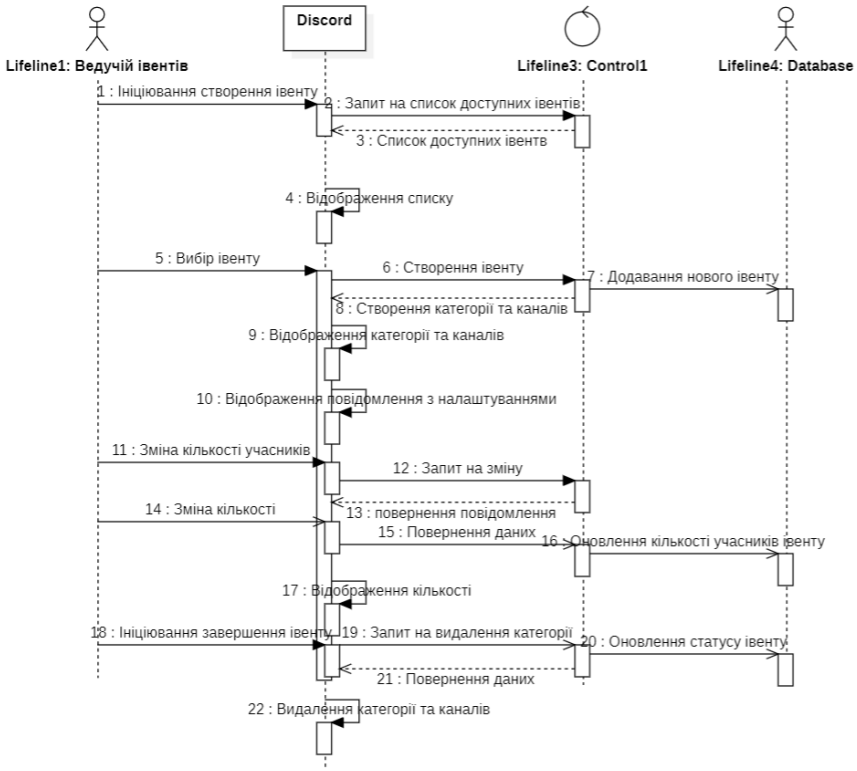


Рисунок 2.6 – Діаграма послідовності для створення та керування подією

Діаграма послідовності для основного потоку подій прецеденту «Вимкнення мікрофону користувачу» представлена на рисунку 2.7.

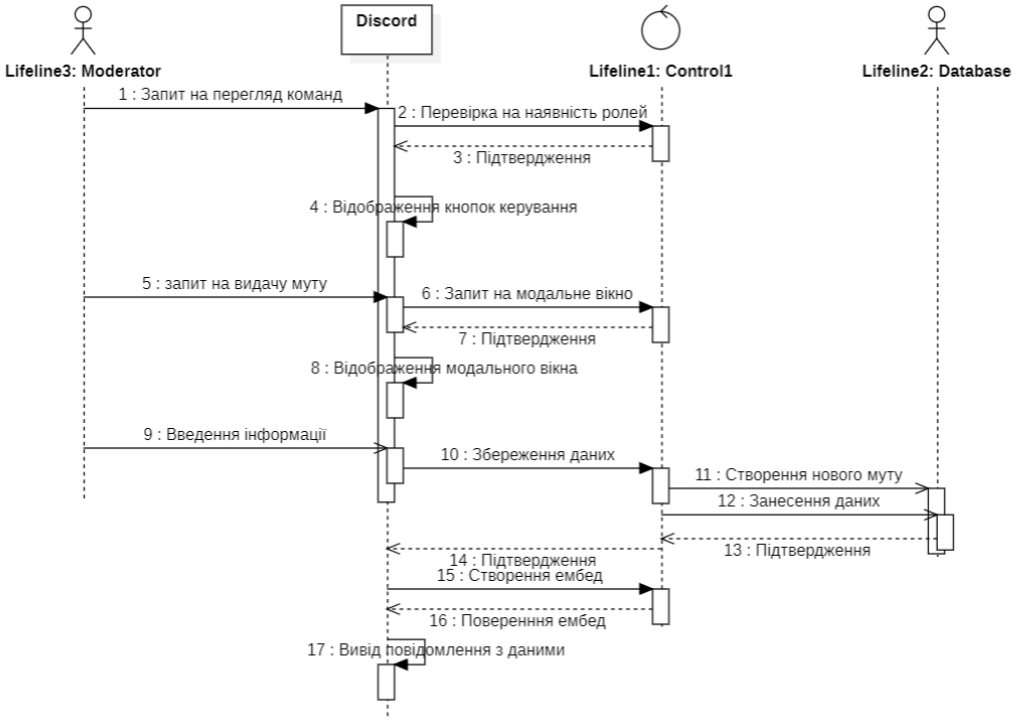


Рисунок 2.7 – Діаграма послідовності для вимкнення мікрофону користувачу

Діаграма послідовності для основного потоку подій покупці ролі у магазині представлена на рисунку 2.8.

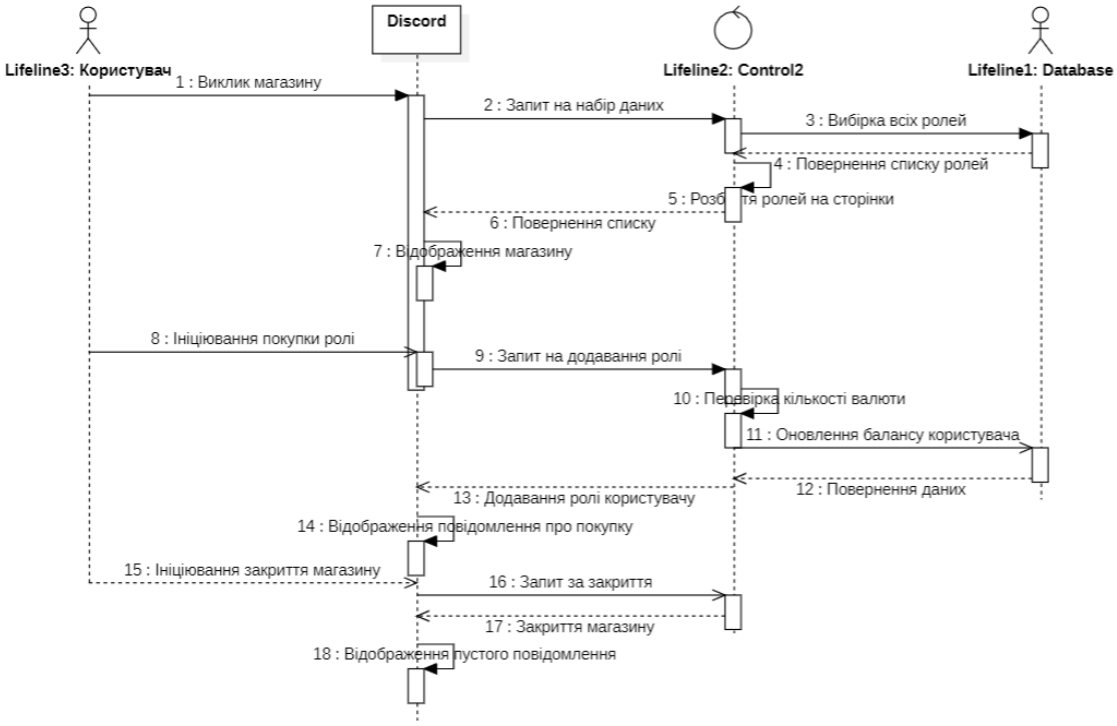


Рисунок 2.8 – Діаграма послідовності для купівлі ролі у магазині

**Висновки до розділу 2**

У другому розділі кваліфікаційної роботи бакалавра спроєктовано програмне забезпечення управління ботом.

Розроблено наступні моделі та діаграми:

– діаграми варіантів використання системи;

– діаграми послідовностей системи;

# 3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Розробка діаграми класів

Діаграма класів – діаграма, що демонструє класи система, їх атрибути, методи і взаємозв'язок між ними. Класи-сутності з атрибутами та методами представлені на рисунку 3.1-3.2.

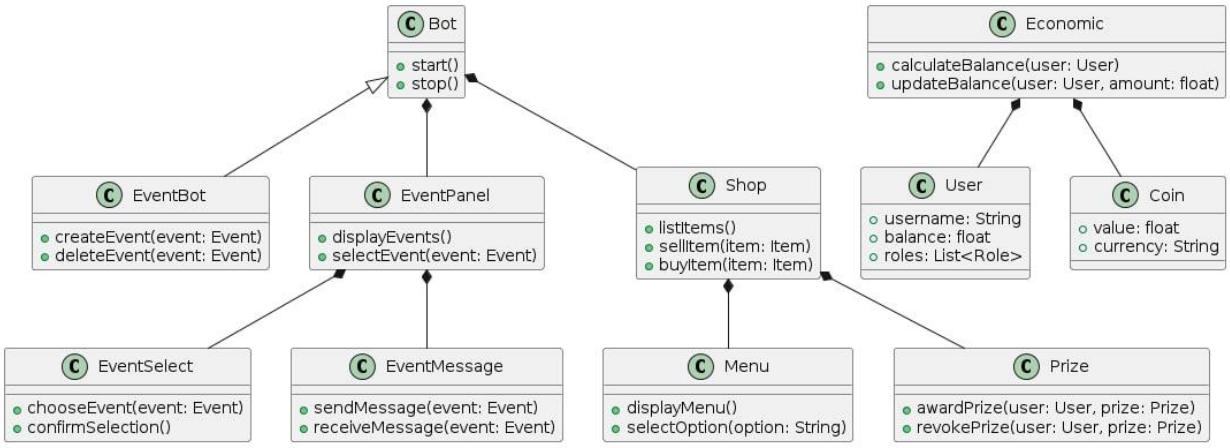


Рисунок 3.1 – Діаграма класів боту подій та економіки

Ось короткий опис кожного класу та їх зв’язків:

Bot:

– Має методи start() та stop().

– Зв’язаний з класом EventBot через відношення спадкування (inheritance), що означає, що EventBot є підкласом Bot.

EventBot:

– Має методи createEvent(event: Event), deleteEvent(event: Event), chooseSelectEvent() та confirmSelection().

– Зв’язаний з класом EventPanel двостороннім асоціативним відношенням.

EventPanel:

– Має методи displayEvents(), selectElement(event: Event), sendMessage(event: Event) та receiveMessage(event: Event).

Shop:

– Частина модуля Economic.

– Має методи listItems(), sellItem(item: Item) та buyItem(item: Item).

– Зв’язаний з класом User двостороннім асоціативним відношенням.

User:

– Частина модуля Economic.

– Має атрибути balance: float, username: String та roles: List<Role>.

– Має методи calculateBalance(user: User) та updateBalance(user: User, amount: float).

Coin:

– Має атрибути value: float та currency: String.

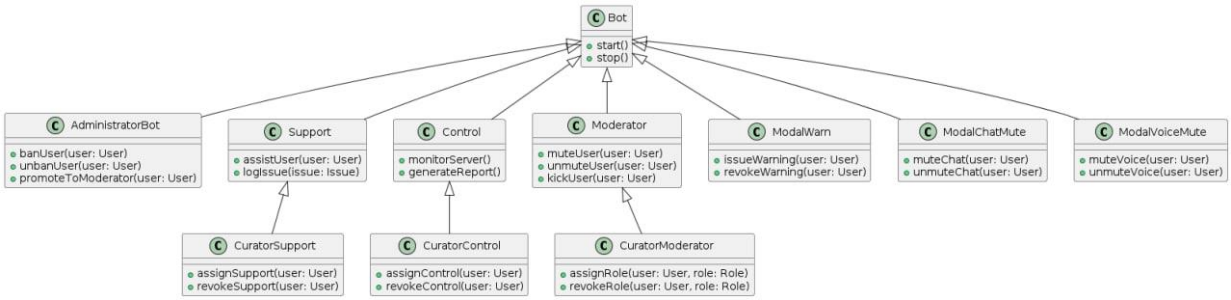


Рисунок 3.2 – Діаграма класів боту адміністрації

Ось десять класів, зображених на діаграмі:

– **User** - з атрибутами: userId, userName.

– **AdministratorBot** - успадковує від класу User з додатковим методом: promoteModerator(userId).

– **Support** - успадковує від класу User з додатковим методом: assignToSupportCase(userId).

– **Control** - успадковує від класу User з додатковим методом: monitorServer(serverId), closeIssue(issueId).

– **Moderator** - успадковує від класу User.

– **ModMailBot** - успадковує від класу Moderator з методами: receiveModMail(user), sendModMail(user), logAction(user).

– **ModMuteBot** - успадковує від класу Moderator з методами: muteUser(userId), unmuteUser(userId).

– **ModKickBot** - успадковує від класу Moderator з методами: kickUser(userId).

– **ModBanBot** - успадковує від класу Moderator з методами: banUser(userId), unbanUser(userId).

### 3.2 Розробка бази даних

Фізична модель бази даних (БД) – це конкретне реалізаційне представлення логічної моделі бази даних, адаптоване для певної системи управління базами даних (СУБД). Вона описує, як дані фізично зберігаються та організовані в системі, включаючи деталі зберігання, індексації, розподілу даних та інші аспекти, що впливають на продуктивність.

Фізична модель бази даних для бота представлено на рисунку 3.3.

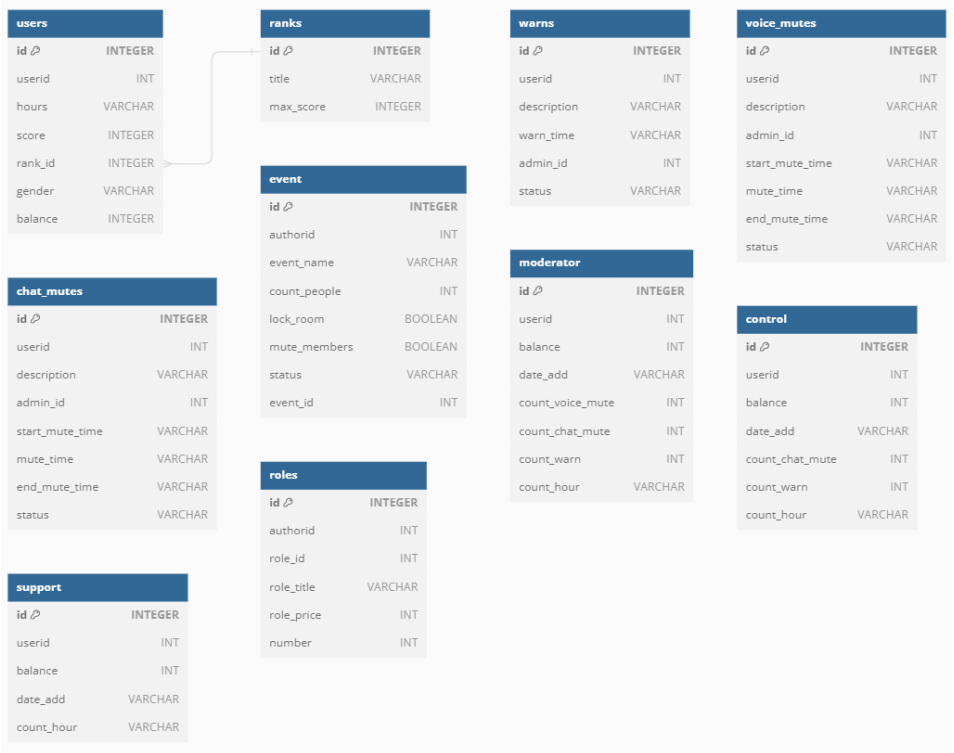


Рисунок 3.3 – Фізична модель бази даних

Давайте розглянемо кожну таблицю окремо.

Таблиця Users зберігає дані про користувачів які знаходяться на сервері, задля того щоб облегшити процес зберігання даних про користувача, які не зберігає сам Discord. Такими даними є дані про стать, баланс, рахунок, години проведені користувачем у голосових каналах.

Таблиця 3.1 – Структура таблиці Users

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| hours | VARCHAR | Години |
| score | INTEGER | Очки |
| rank_id | INTEGER | Код рангу, зовнішній ключ до таблиці ranks |
| gender | VARCHAR | Стать |
| balance | INTEGER | Баланс |

Таблиця 3.2 – Структура таблиці Ranks

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| title | VARCHAR | Назва рангу |
| max_score | INTEGER | Максимальна кількість очок |

Таблиця 3.3 – Структура таблиці Warns

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| description | VARCHAR | Опис |
| warn_time | VARCHAR | Час попередження |
| admin_id | INT | Ідентифікатор адміністратора |
| status | VARCHAR | Статус |

Таблиця 3.4 – Структура таблиці Voice mutes

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| description | VARCHAR | Опис |
| admin_id | INT | Ідентифікатор адміністратора |
| start_mute_time | VARCHAR | Час початку блокування голосу |
| mute_time | VARCHAR | Час блокування |
| end_mute_time | VARCHAR | Час завершення блокування |
| status | VARCHAR | Статус |

Таблиця 3.5 – Структура таблиці Chat mutes

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| description | VARCHAR | Опис |
| admin_id | INT | Ідентифікатор адміністратора |
| start_mute_time | VARCHAR | Час початку блокування чату |
| mute_time | VARCHAR | Час блокування |
| end_mute_time | VARCHAR | Час завершення блокування |

Таблиця 3.6 – Структура таблиці Event

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| authorid | INT | Ідентифікатор автора |
| event_name | VARCHAR | Назва події |
| count_people | INT | Кількість людей |
| lock_room | BOOLEAN | Заблокувати кімнату |
| mute_members | BOOLEAN | Вимкнути звук у учасників |
| status | VARCHAR | Статус |
| event_id | INT | Ідентифікатор події |

Таблиця 3.7 – Структура таблиці Moderator

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| balance | INT | Баланс |
| date_add | VARCHAR | Дата додавання |
| count_voice_mute | INT | Кількість блокувань голосу |
| count_chat_mute | INT | Кількість блокувань чату |
| count_warn | INT | Кількість попереджень |
| count_hour | VARCHAR | Кількість годин |

Таблиця 3.8 – Структура таблиці Control

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| balance | INT | Баланс |
| date_add | VARCHAR | Дата додавання |
| count_chat_mute | INT | Кількість блокувань чату |
| count_warn | INT | Кількість попереджень |
| count_hour | VARCHAR | Кількість годин |

Таблиця 3.9 – Структура таблиці Support

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| userid | INT | Ідентифікатор користувача |
| balance | INT | Баланс |
| date_add | VARCHAR | Дата додавання |
| count_hour | VARCHAR | Кількість годин |

Таблиця 3.10 – Структура таблиці Roles

| Поле | Тип даних | Опис |
|---|---|---|
| id | INTEGER | Первинний ключ, автоінкремент |
| authorid | INT | Ідентифікатор автора |
| role_id | INT | Ідентифікатор ролі |
| role_title | VARCHAR | Назва ролі |
| role_price | INT | Ціна ролі |
| number | INT | Кількість |

### 3.3 Вибір засобів розробки

Python - це високорівнева мова програмування загального призначення, яка відома своєю простотою і зручністю для користувачів. Вона була створена Гвідо ван Россумом і вперше випущена в 1991 році. Головна мета Python – зробити програмування легким і інтуїтивно зрозумілим. Це досягається завдяки читабельному синтаксису, який нагадує звичайну англійську мову, та значній

кількості стандартних бібліотек, що дозволяють швидко створювати функціональний код.

Однією з ключових переваг Python є його універсальність. Він підходить для розробки веб-додатків, аналізу даних, штучного інтелекту, автоматизації завдань, наукових досліджень і багато іншого. Багато популярних бібліотек і фреймворків розширюють можливості Python, зокрема, Django для веб-розробки, NumPy і Pandas для аналізу даних, TensorFlow і PyTorch для машинного навчання.

Python також славиться своєю підтримкою об'єктно-орієнтованого програмування (ООП), що дозволяє програмістам створювати добре структуровані і підтримувані програми. Крім того, мова підтримує інші парадигми програмування, такі як процедурне і функціональне програмування.

Інтерпретована природа Python означає, що код виконується рядок за рядком, що дозволяє швидко тестувати і відлагоджувати програми. Це робить Python ідеальним для швидкого прототипування і розробки мінімальних життєздатних продуктів (MVP). Однак, це ж може бути недоліком у випадках, коли потрібна максимальна продуктивність, оскільки інтерпретовані мови зазвичай працюють повільніше за компільовані.

Спільнота Python відіграє важливу роль у його розвитку і популярності. Величезна кількість відкритих проектів і бібліотек доступні на платформі GitHub і інших ресурсах, що дозволяє розробникам легко знаходити необхідні інструменти і ділитися своїми власними напрацюваннями. Крім того, існує багато форумів і груп підтримки, де новачки можуть отримати допомогу від більш досвідчених колег.

Python також активно використовується в освіті. Його простота і читабельність роблять його ідеальним для навчання основам програмування. Багато університетів і шкіл включають Python у свої навчальні програми, оскільки він дозволяє студентам зосередитися на концепціях програмування, а не на складнощах синтаксису.

Зростання популярності Python продовжується, і багато компаній, включаючи такі гіганти як Google, Facebook і Netflix, активно використовують

його у своїх проектах. Це підтверджує той факт, що Python є не лише мовою для початківців, але й потужним інструментом для професійної розробки.

Disnake.py – це асинхронна бібліотека для створення ботів та інтерактивних додатків для Discord. Вона написана на Python і є форком оригінальної бібліотеки discord.py, що була припинена у 2021 році. Disnake.py пропонує розширені можливості і нові функції, спрямовані на полегшення розробки ботів для Discord з використанням сучасних стандартів та технологій.

Головна особливість disnake.py полягає в її асинхронній природі, що дозволяє обробляти численні запити та події без блокування основного потоку виконання програми. Це особливо важливо для ботів, які мають реагувати на велику кількість одночасних повідомлень та подій у реальному часі. Асинхронність досягається завдяки використанню бібліотеки asyncio, яка є частиною стандартної бібліотеки Python.

Disnake.py пропонує широкий набір інструментів для роботи з API Discord. Вона дозволяє створювати та налаштовувати боти, обробляти повідомлення, реакції, команди і події. Бібліотека підтримує інтерактивні елементи, такі як кнопки та меню, що дозволяє створювати більш інтерактивні та зручні для користувачів боти. Крім того, disnake.py надає інструменти для роботи з мультимедіа, такими як завантаження та відтворення аудіофайлів.

Однією з ключових переваг disnake.py є її гнучкість. Вона дозволяє розробникам створювати боти з використанням як простих, так і складних логік. Наприклад, можна створити простого бота, який відповідає на команди користувачів, або ж розробити складну систему з обробкою даних, інтеграцією з іншими сервісами та управлінням різними аспектами серверів Discord.

Документація disnake.py детально описує всі доступні функції та методи, надаючи приклади використання і пояснення. Це значно спрощує процес розробки, дозволяючи навіть новачкам швидко освоїти основні принципи роботи з бібліотекою та створювати власні проекти.

Спільнота розробників disnake.py активно підтримує бібліотеку, випускаючи оновлення та нові функції. Форум і Discord-сервери спільноти

надають платформу для обговорення, обміну досвідом та допомоги у вирішенні проблем. Це робить процес розробки більш інтерактивним і підтримуваним.

Завдяки своїй потужності та гнучкості, disnake.py широко використовується для створення різноманітних ботів, від простих ігрових ботів до складних систем управління спільнотами та інтеграції з іншими сервісами. Її можливості дозволяють реалізувати майже будь-яку задумку, що робить її одним з найпопулярніших інструментів для розробки Discord-ботів.

SQLite3 - це легка, вбудована система управління базами даних (DBMS), яка надає реляційний SQL-інтерфейс для зберігання даних. Вона була створена у 2000 році Річардом Гіппом і відтоді стала однією з найбільш популярних баз даних у світі, завдяки своїй простоті, надійності та ефективності.

Однією з головних переваг SQLite3 є її вбудованість. Це означає, що база даних зберігається в одному файлі на диску і не потребує окремого серверного процесу для управління даними. Такий підхід робить SQLite3 ідеальним для використання в мобільних додатках, десктопних програмах, веб-браузерах і вбудованих системах. Вона дозволяє зберігати дані локально без необхідності встановлення і налаштування окремого серверного програмного забезпечення.

SQLite3 підтримує стандартний SQL, що дозволяє виконувати складні запити, створювати таблиці, індекси, тригери і представлення. Вона також підтримує транзакції, що гарантує цілісність даних навіть у випадку збою системи. Завдяки механізму журналювання змін, SQLite3 забезпечує надійне збереження даних і відновлення після збоїв.

Одним з ключових аспектів SQLite3 є її продуктивність. Вона оптимізована для швидкого доступу до даних і ефективного використання ресурсів. Це досягається завдяки використанню ефективних алгоритмів і структур даних, таких як B-дерева для індексації. Крім того, SQLite3 підтримує стиснення даних, що дозволяє зменшити розмір бази даних і зберігати більше інформації у меншому обсязі.

Документація SQLite3 детально описує всі аспекти роботи з базою даних, надаючи приклади і пояснення. Вона охоплює як базові поняття, так і складніші

теми, такі як оптимізація запитів і робота з транзакціями. Це дозволяє розробникам швидко освоїти SQLite3 і ефективно використовувати її у своїх проектах.

SQLite3 активно використовується у багатьох відомих продуктах і компаніях. Наприклад, браузери Mozilla Firefox і Google Chrome використовують SQLite3 для зберігання налаштувань та історії переглядів. Також вона широко використовується в мобільних додатках на платформах Android і iOS для зберігання даних локально.

Однією з цікавих особливостей SQLite3 є її здатність працювати в режимі без постійного зберігання даних. Це дозволяє використовувати базу даних у тимчасовому режимі, коли дані зберігаються лише у оперативній пам'яті і зникають після закриття програми. Такий режим корисний для тестування та тимчасового зберігання даних.

Завдяки своїй простоті, надійності і ефективності, SQLite3 є невід'ємною частиною багатьох сучасних додатків. Вона дозволяє розробникам легко інтегрувати базу даних у свої проекти, забезпечуючи надійне зберігання і швидкий доступ до даних.

PyCharm - це інтегроване середовище розробки (IDE) для Python, розроблене компанією JetBrains. Воно відоме своєю потужною функціональністю, зручністю використання та великою кількістю інструментів, що спрощують процес програмування. PyCharm пропонує розробникам все необхідне для створення, тестування і відлагодження Python-додатків.

Однією з головних особливостей PyCharm є його підтримка багатьох технологій і фреймворків. IDE має вбудовану підтримку Django, Flask, Pyramid та інших популярних веб-фреймворків, що дозволяє розробникам легко створювати веб-додатки. Крім того, PyCharm підтримує наукові бібліотеки, такі як NumPy, SciPy, Pandas і Matplotlib, що робить його ідеальним інструментом для наукових досліджень і аналізу даних.

PyCharm пропонує потужний редактор коду з підтримкою автодоповнення, підсвічування синтаксису та перевірки помилок у реальному часі. Це значно

підвищує продуктивність розробників, дозволяючи їм зосередитися на логіці програми, а не на дрібних помилках. Крім того, PyCharm підтримує рефакторинг коду, що дозволяє легко змінювати структуру програми без ризику виникнення помилок.

Інструменти для відлагодження і тестування коду є ще однією важливою частиною PyCharm. IDE надає потужний відлагоджувач, що дозволяє розробникам покроково виконувати код, встановлювати точки зупинки і аналізувати змінні. Це робить процес відлагодження значно простішим і ефективнішим. Крім того, PyCharm підтримує інтеграцію з системами керування версіями, такими як Git, що дозволяє легко відслідковувати зміни в коді і працювати в команді.

PyCharm також пропонує інструменти для роботи з базами даних. IDE підтримує широкий спектр баз даних, включаючи MySQL, PostgreSQL, SQLite, Oracle та інші. Це дозволяє розробникам легко виконувати запити, переглядати і редагувати дані безпосередньо з середовища розробки.

Налаштовуваність PyCharm - ще одна його сильна сторона. Розробники можуть налаштовувати IDE відповідно до своїх потреб, змінюючи тему оформлення, гарячі клавіші і багато іншого. Крім того, PyCharm підтримує встановлення плагінів, що дозволяє розширити його функціональність і додати нові інструменти.

Спільнота розробників PyCharm активно підтримує IDE, випускаючи регулярні оновлення і нові функції. Форум і офіційний сайт PyCharm надають багато корисної інформації, включаючи документацію, відеоуроки і приклади коду. Це дозволяє новачкам швидко освоїти IDE і почати використовувати її у своїх проектах.

Завдяки своїй потужності і гнучкості, PyCharm є одним з найпопулярніших інструментів для розробки на Python. Вона дозволяє розробникам зосередитися на створенні якісного коду, надаючи всі необхідні інструменти для ефективної роботи.

Discord Developer Portal - це платформа, яка надає розробникам інструменти і ресурси для створення ботів і інтеграцій для Discord. Вона є центральним місцем

для доступу до документації, API та інших ресурсів, що допомагають розробникам створювати потужні та інтерактивні додатки для Discord-серверів.

Однією з головних функцій Discord Developer Portal є можливість створення і налаштування ботів. Розробники можуть легко створити новий бот, зареєструвавши його на порталі і отримавши унікальний токен, який використовується для аутентифікації бота в Discord. Портал надає зручний інтерфейс для налаштування прав доступу бота, визначення команд та подій, на які бот має реагувати.

API Discord є ще одним важливим елементом порталу. Воно дозволяє розробникам взаємодіяти з серверами Discord, отримувати інформацію про користувачів, канали, повідомлення і багато іншого. API підтримує різноманітні методи запитів, такі як GET, POST, PUT і DELETE, що дозволяє виконувати різні операції з даними. Крім того, API підтримує веб-хуки, що дозволяє отримувати сповіщення про події в реальному часі.

Документація Discord Developer Portal детально описує всі аспекти використання API і створення ботів. Вона включає приклади коду, пояснення і рекомендації, що значно спрощує процес розробки. Крім того, на порталі є розділ з часто задаваними питаннями (FAQ) і форум, де розробники можуть отримати допомогу від спільноти та розробників Discord.

Discord Developer Portal також підтримує створення інтеграцій з іншими сервісами. Це дозволяє розробникам створювати боти, які можуть взаємодіяти з зовнішніми API, базами даних та іншими сервісами. Така функціональність відкриває безліч можливостей для автоматизації завдань, управління спільнотами та створення інтерактивних додатків.

Портал також надає інструменти для тестування і відлагодження ботів. Розробники можуть використовувати тестові сервери, щоб перевірити роботу своїх ботів, налаштувати логування і отримувати детальну інформацію про помилки. Це дозволяє швидко виявляти і виправляти проблеми, забезпечуючи стабільну роботу ботів.

Оновлення та нові функції регулярно додаються до Discord Developer Portal, що дозволяє розробникам використовувати найновіші можливості платформи. Команда Discord активно працює над покращенням документації, додаванням нових прикладів і інструментів, що робить процес розробки ще більш зручним і ефективним.

Завдяки своїй потужності і гнучкості, Discord Developer Portal є незамінним інструментом для розробників, які хочуть створювати інтерактивні та потужні боти для Discord. Він надає всі необхідні ресурси і інструменти, що дозволяють реалізувати майже будь-яку задумку, від простих команд до складних інтеграцій з іншими сервісами.

**Висновки до розділу 3**

У третьому розділі кваліфікаційної роботи розроблено діаграми класів і модель бази даних. Діаграми класів детально відображають структуру та взаємодію основних компонентів системи. Вони допомагають візуалізувати об'єктно-орієнтовану модель проекту, показуючи зв'язки між класами, їх атрибути та методи. Це є важливим інструментом для розуміння логіки та архітектури програмного забезпечення, яке розробляється.

Модель бази даних визначає структуру зберігання даних та їх взаємозв'язок. Вона включає детальний опис таблиць бази даних, їх полів, типів даних, а також ключів та індексів. Це забезпечує ефективне зберігання та доступ до даних, що є критично важливим для функціонування системи.

Також у цьому розділі детально описано вибір засобів розробки. Мову програмування Python обрано завдяки її простоті, читабельності та широким можливостям для розробки різноманітних програмних рішень. Python має велику кількість бібліотек та фреймворків, що значно спрощує процес розробки.

Використання бібліотеки Disnake.py дозволяє легко інтегрувати додаток з платформою Discord, забезпечуючи розширені можливості для взаємодії з користувачами через ботів. Disnake.py надає зручні інтерфейси для роботи з Discord API, що спрощує створення функціоналу бота.

# 4 ПРОГРАМНА РЕАЛІЗАЦІЯ БОТУ

## 4.1 Загальна структура боту

Результатом розробки стало створення комплексної системи адміністрування та управління на Discord сервері, яка складається з наступних модулів:

- модуль охоронців;
- модуль редакторів чату;
- модуль модераторів;
- модуль кураторів;
- модуль подій;
- модуль економіки.

На рисунку 4.1 представлено список всіх команд доступних у бота.

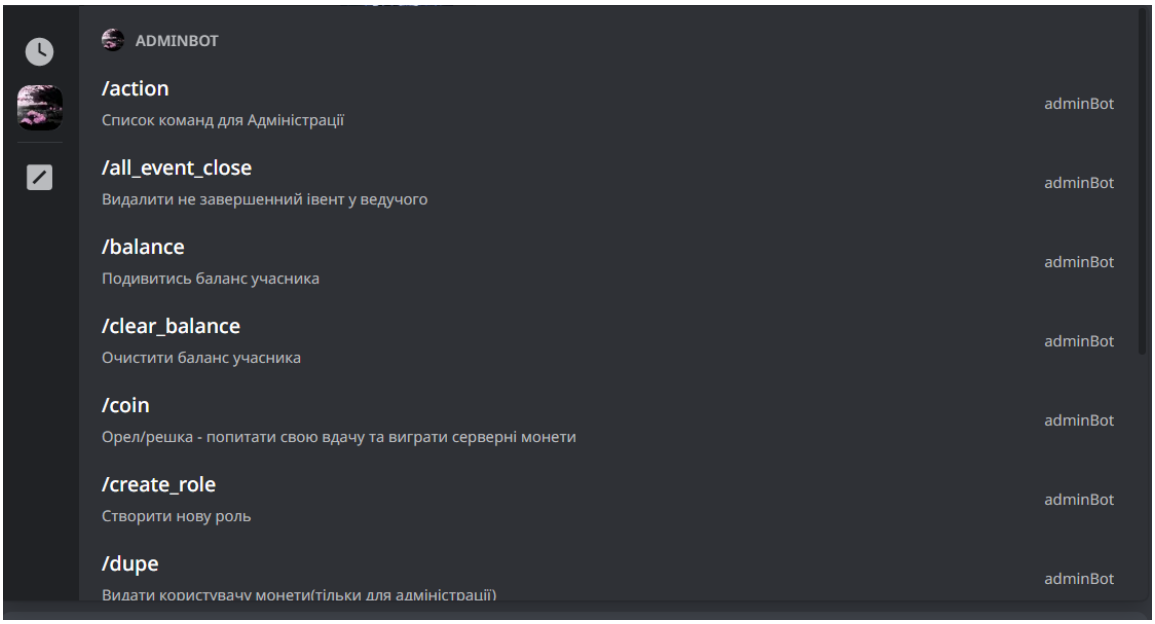


Рисунок 4.1 – Список всіх команд бота на сервері

На рисунку 4.2 представлено другу частинку списку всіх команд доступних у бота.

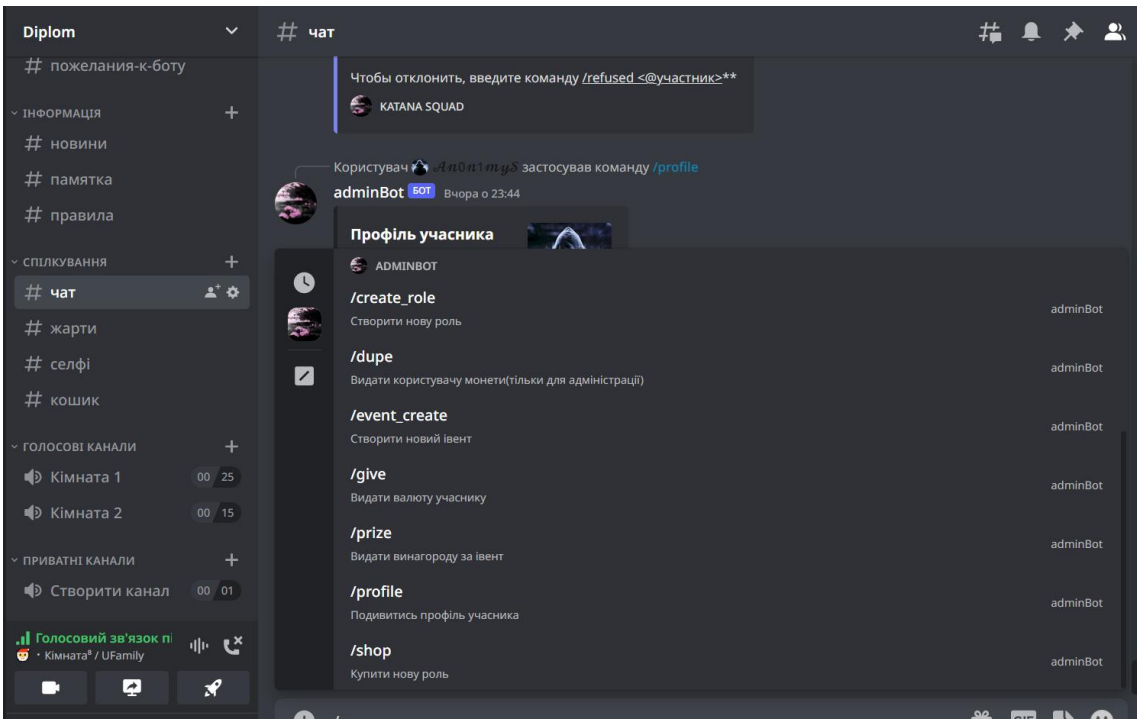


Рисунок 4.2 – Список всіх команд бота на сервері

## 4.2 Модуль адміністрації

### Модуль охоронців

Цей модуль надає можливість виконувати такі команди:

- верифікація користувача;
- зміна статусу користувача;
- блокування користувача на сервері.

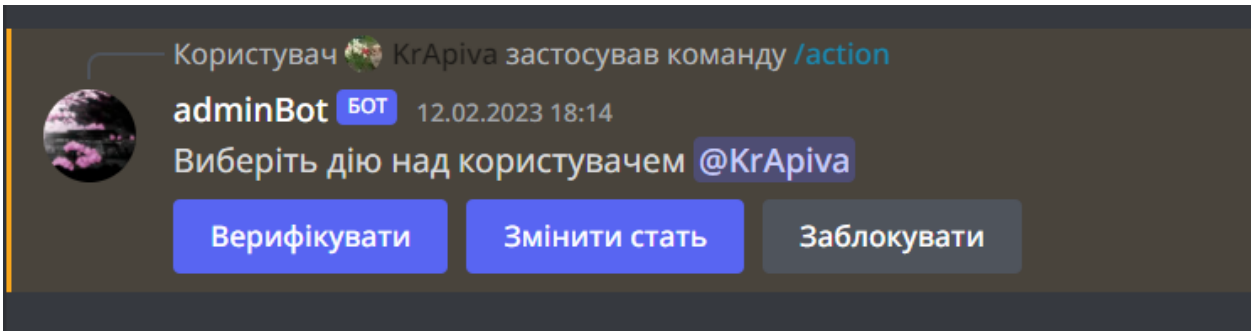


Рисунок 4.3 – Панель керування користувачем для охоронців

Лістинг функцій керування користувачем для охоронців:

```
@disnake.ui.button(label='Верифікувати',
style=disnake.ButtonStyle.blurple)
```

```
        async def button_verify(self, button: disnake.ui.Button,
interaction: disnake.MessageInteraction):
            self.stop()
            await interaction.response.send_message(f"Верифікація
{self.member.mention} успішна")
            unverify_role =
interaction.guild.get_role(dysium.UNVERIFY_ID)
            girl_role =
interaction.guild.get_role(dysium.GIRL_ROLE_ID)
            man_role =
interaction.guild.get_role(dysium.MAN_ROLE_ID)
            await self.member.remove_roles(unverify_role)
            await self.member.add_roles(man_role)
            conn = sqlite3.connect('database/users.db')
            curs = conn.cursor()
            curs.execute("SELECT * FROM users WHERE userid = ?",
self.member.id)
            data_s = curs.fetchone()
            conn.commit()
            conn.close()
            if data_s == None:
                if girl_role in self.member.roles:

                    conn = sqlite3.connect('database/users.db')
                    curs = conn.cursor()
                    curs.execute("UPDATE users SET gender = ? WHERE
userid = ?", ("♀", self.member.id,))
                    curs.execute("INSERT INTO users
VALUES(NULL,?,?, ?, ?, ?)",
                                 (self.member.id, self.member,
"Вільна", "♀", 0))
                    conn.commit()
                    conn.close()
                elif man_role in self.member.roles:

                    conn = sqlite3.connect('database/users.db')
```

```
                    curs = conn.cursor()
                    curs.execute("UPDATE users SET gender = ? WHERE
userid = ?", ("♂", self.member.id,))
                    curs.execute("INSERT INTO users
VALUES(NULL,?,?, ?, ?, ?)", (self.member.id, self.member.mention,
"Вільний", "♂", 0,))
                    conn.commit()
                    conn.close()
```

**Модуль редакторів чату**

Модуль включає наступні команди:

- видача попередження користувачу;
- зняття попередження з користувача;
- накладення користувачу обмеження писати повідомлення;
- зняття обмежень на повідомлення з користувача;
- перегляд історії обмежень користувача.

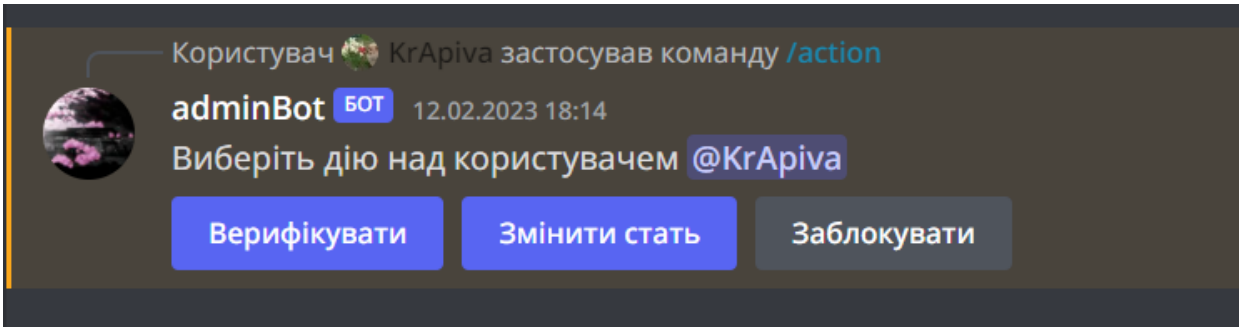


Рисунок 4.4 – Панель керування користувачем для редакторів чату

Лістинг функцій керування користувачем для редактора чату:

```
class Control(disnake.ui.View):
    message: disnake.Message

    def __init__(self, bot, author, member):
        super().__init__(timeout=30.0)
        self.bot = bot
        self.value = None
        self.author = author
        self.member = member
```

```
        async def interaction_check(self, interaction:
disnake.ApplicationCommandInteraction) -> bool:
            if interaction.author.id != self.author.id:
                return await interaction.response.send_message(
                    "Вы не можете управлять этими кнопками!",
ephemeral=True
                )
            return True

        async def on_timeout(self):
            for button in self.children:
                button.disabled = True
            await self.message.edit(view=self)

        @disnake.ui.button(label='Застереження',
style=disnake.ButtonStyle.blurple)
        async def button_warn(self, button: disnake.ui.Button,
inter: disnake.AppCmdInter):

            await
inter.response.send_modal(modal=ModalWarn(self.member, self.author))

        @disnake.ui.button(label="Зняти застереження",
style=disnake.ButtonStyle.grey)
        async def button_unwarn(self, button: disnake.ui.Button,
interaction: disnake.MessageInteraction):
            warn_role =
self.member.guild.get_role(dysium.WARN_ROLE_ID)
            await self.member.remove_roles(warn_role)
            await interaction.response.send_message(f"Користувач
{self.member.mention} позбавлений застереження")

        @disnake.ui.button(label='Чат мут',
style=disnake.ButtonStyle.blurple)
        async def button_chat_mute(self, button: disnake.ui.Button,
inter: disnake.AppCmdInter):
```

```
            await
inter.response.send_modal(modal=ModalChatMute(self.member,
self.author))
```

**Модуль модераторів**

Цей модуль містить команди для:

- зміни статусу користувача;
- блокування користувача;
- розблокування користувача;
- видачі попередження користувачу;
- зняття попередження з користувача;
- накладення користувачу на використання мікрофону;
- зняття з користувача обмеження на використання мікрофону;
- перегляду історії обмежень мікрофону укористувача;
- накладення користувачу обмежень на повідомлення;
- зняття з користувача обмежень на повідомлення;
- перегляду історії обмежень повідомлень у користувача.

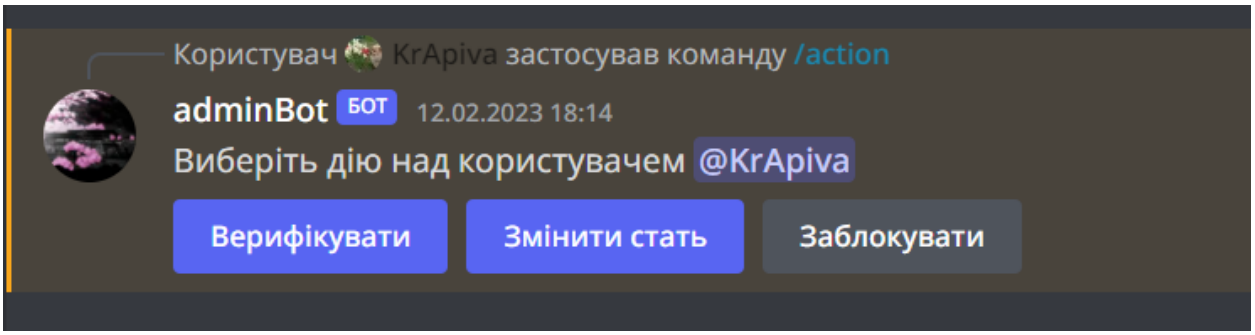


Рисунок 4.5 – Панель керування користувачем для модераторів

Лістинг функцій керування користувачем для охоронців:

```
class Moderator(disnake.ui.View):
    message: disnake.Message

    def __init__(self,bot, author, member):
        super().__init__(timeout=60.0)
        self.value = None
        self.bot = bot
```

```
            self.author = author
            self.member = member

        async def interaction_check(self, interaction:
disnake.ApplicationCommandInteraction) -> bool:
            if interaction.author.id != self.author.id:
                return await interaction.response.send_message(
                    "Ви не можете керувати даними кнопками!",
ephemeral=True
                )
            return True

        async def on_timeout(self):
            for button in self.children:
                button.disabled = True
            await self.message.edit(view=self)
```

Панель керування користувачем для кураторів охоронців представлена на рисунку 4.6.

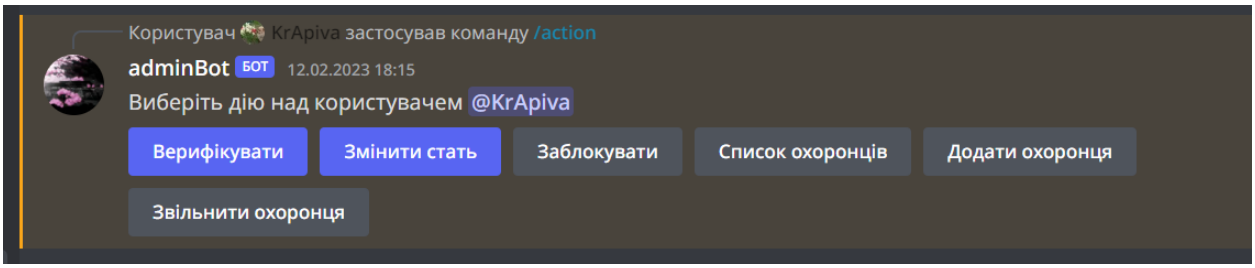


Рисунок 4.6 – Панель керування користувачем для кураторів охоронців

Лістинг функцій керування користувачем для кураторів охоронців:

```
class CuratorSupport(disnake.ui.View):
    message: disnake.Message

    def __init__(self, bot, author, member):
        super().__init__(timeout=30.0)
        self.value = None
        self.bot = bot
        self.author = author
        self.member = member
```

```
        async def interaction_check(self, interaction:
disnake.ApplicationCommandInteraction) -> bool:
            if interaction.author.id != self.author.id:
                return await interaction.response.send_message(
                    "Ви не можете керувати даними кнопками!",
ephemeral=True
                )
            return True

        async def on_timeout(self):
            for button in self.children:
                button.disabled = True
            await self.message.edit(view=self)
```

Панель керування користувачем для кураторів редакторів чату представлена на рисунку 4.7.

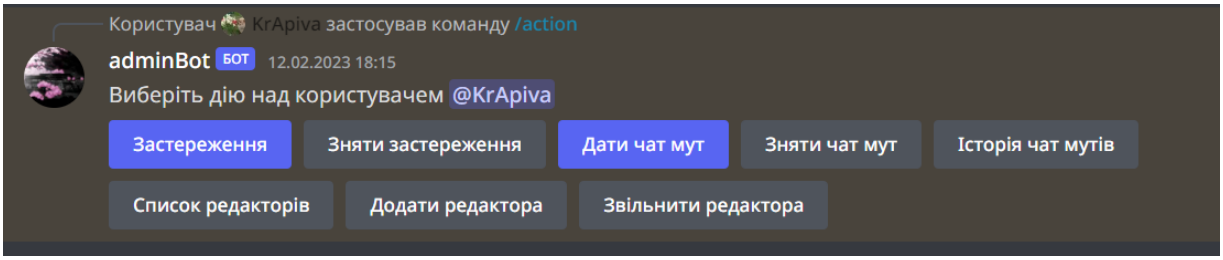


Рисунок 4.7 – Панель керування користувачем для кураторів редакторів чату

Лістинг функцій керування користувачем для кураторів редакторів чату:

```
class CuratorControl(disnake.ui.View):
    message: disnake.Message

    def __init__(self, bot, author, member):
        super().__init__(timeout=30.0)
        self.bot = bot
        self.value = None
        self.author = author
        self.member = member
```

```
        async def interaction_check(self, interaction:
disnake.ApplicationCommandInteraction) -> bool:
            if interaction.author.id != self.author.id:
                return await interaction.response.send_message(
                    "Ви не можете керувати даними кнопками!",
ephemeral=True
                )
            return True

        async def on_timeout(self):
            for button in self.children:
                button.disabled = True
            await self.message.edit(view=self)

        @disnake.ui.button(label='Застереження',
style=disnake.ButtonStyle.blurple)
        async def button_warn(self, button: disnake.ui.Button,
inter: disnake.AppCmdInter):

            await
inter.response.send_modal(modal=ModalWarn(self.member, self.author))
```

Панель керування користувачем для кураторів модераторів представлена на рисунку 4.8.

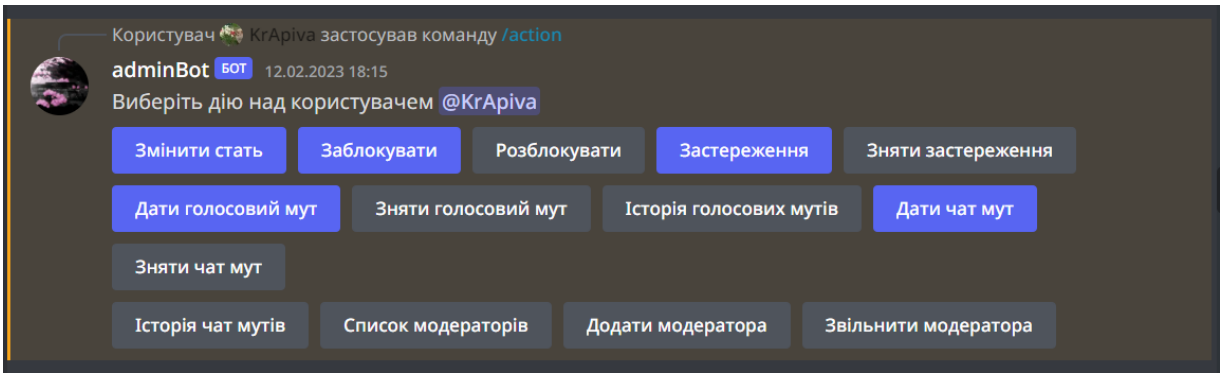


Рисунок 4.8 – Панель керування користувачем для кураторів модераторів

Лістинг функцій керування користувачем для кураторів модераторів:

```
class CuratorModerator(disnake.ui.View):
    message: disnake.Message
```

```
        def __init__(self,bot, author, member):
            super().__init__(timeout=60.0)
            self.value = None
            self.bot = bot
            self.author = author
            self.member = member

        async def interaction_check(self, interaction:
disnake.ApplicationCommandInteraction) -> bool:
            if interaction.author.id != self.author.id:
                return await interaction.response.send_message(
                    "Ви не можете керувати даними кнопками!",
ephemeral=True
                )
            return True

        async def on_timeout(self):
            for button in self.children:
                button.disabled = True
            await self.message.edit(view=self)
```

### 4.3 Модуль економіки та подій

**Модуль економіки**

Для публічного українського серверу розроблено Discord бота з модулем управління економікою, який включає наступні команди:

- перевірка профілю учасників на сервері;
- перегляд балансу учасників на сервері;
- передавання серверної валюти іншому учаснику;
- очистка балансу обраного учасника серверу;
- видача серверної валюти адміністратором серверу учасникам;
- створення користувацької ролі на сервері;
- видалення користувацької ролі на сервері;

- виклик магазину користувацьких ролей;
- запуск одиночної міні гри "Орел чи решка".

**Модуль подій**

Для управління подіями розроблено модуль, що містить такі команди:

- створення подій;
- зміна кількості учасників події;
- зміна режиму доступу до події;
- видача винагороди за участь чи перемогу на події;
- завершення події.

Першим етапом є створення подій на сервері за допомогою команди /event_create. Після виклику цієї команди, бот виводить повідомлення з вибором події зі списку. Роботу команди представлено на рисунку 4.9.

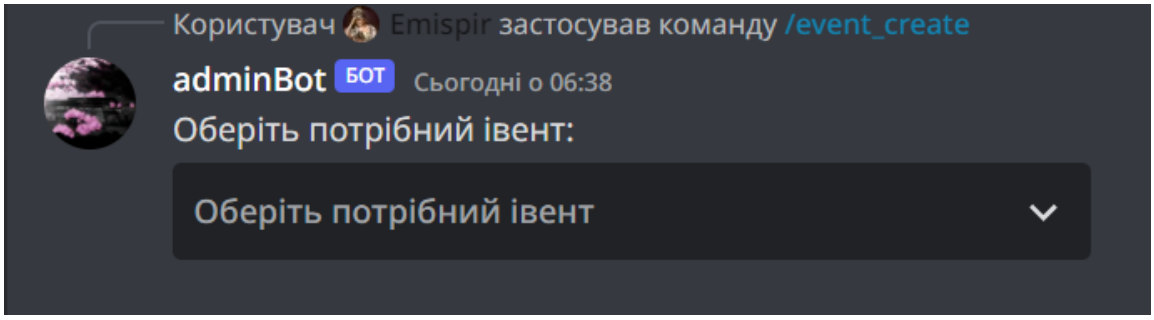


Рисунок 4.9 – Створення події

На рисунку 4.10 представлено вибір категорії кодії

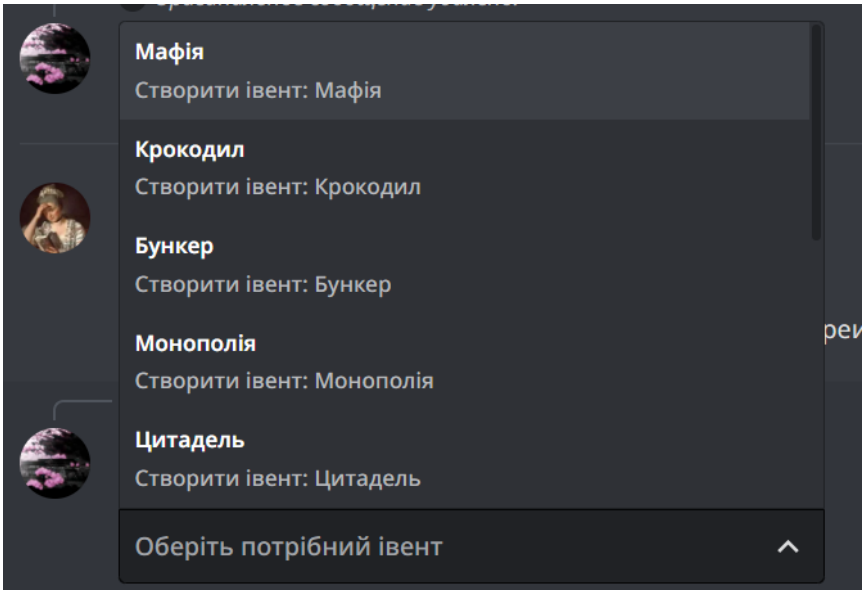


Рисунок 4.10 – Створення події

Лістинг функцій для створення події:

```
    @commands.slash_command(name="event_create",
description="Створити новий івент")
        async def event_create(self, ctx):
            print(f"member:{ctx.author} func: /event_create")
            # Create the view containing our dropdown
            status = "Active"
            conn = sqlite3.connect('database/event.db')
            curs = conn.cursor()
            event = curs.execute("SELECT * FROM event WHERE authorid
= ? AND status = ?",
                                                    (ctx.author.id,
status)).fetchone()
            conn.commit()
            conn.close()
            if event == None or event[6] =="Deactive":
                view = EventsPanel(self.bot,ctx.author)
                message = await ctx.send("Оберіть потрібний івент:",
view=view)
                view.message = message
            elif event[6] == "Active":
                await ctx.send("Завершіть старий івент, щоб почати
новий")
            # Sending a message containing our view
```

Далі після створення нової події, бот надсилає підтвердження про створення нового заходу. Роботу боту представлено на рисунку 4.11.

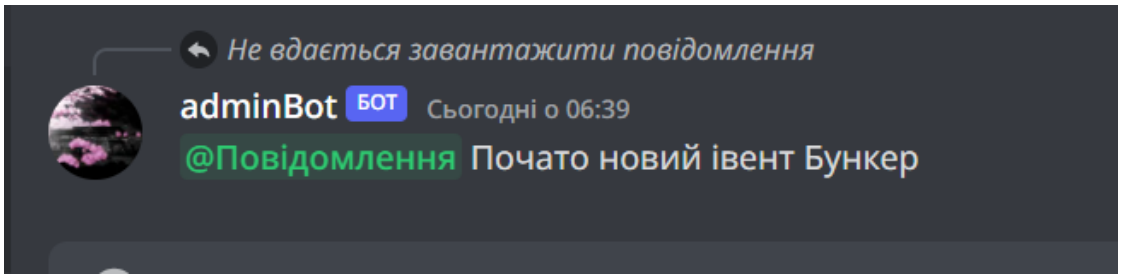


Рисунок 4.11 – Повідомлення про створення заходу

Після цього, система створює нову категорію каналів для цієї події. В цій категорії створюється три канали: для управління подією, для спілкування учасників події, та голосовий канал для проведення події.

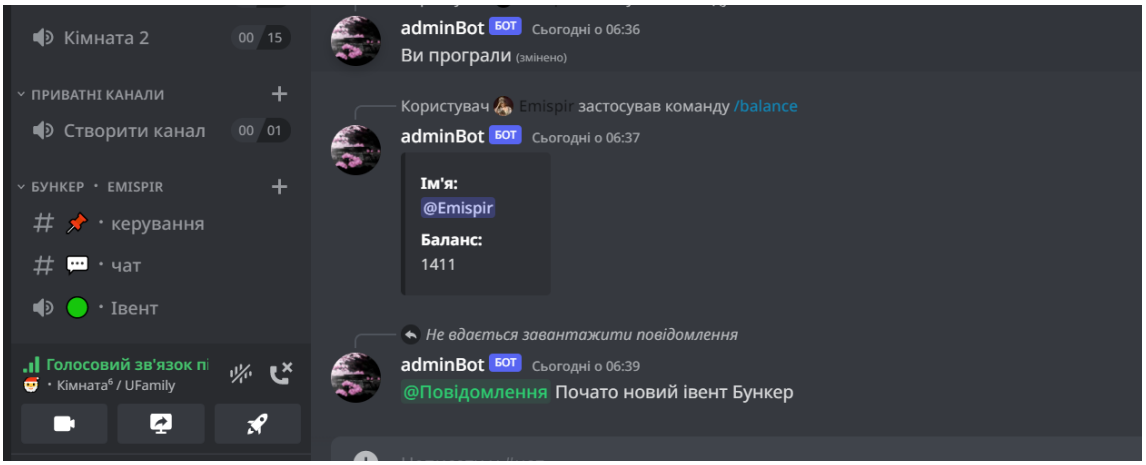


Рисунок 4.12 – Створено нові категорії каналів

Лістинг коду для створення нової категорії каналів:

```
async def callback(self, inter: disnake.MessageInteraction):
        await inter.message.delete()
        await inter.guild.create_category(f"{self.values[0]} · {inter.author.display_name}", position= 8)

        category = disnake.utils.get(inter.guild.categories, name=f"{self.values[0]} · {inter.author.display_name}")
        category_id = category.id
        conn = sqlite3.connect('database/event.db')
        curs = conn.cursor()
        curs.execute("INSERT INTO event VALUES(NULL,?,?,?,?,?,?,?)",
                                        (inter.author.id, self.values[0], 8, True, True,"Active", category_id))
        conn.commit()
        conn.close()

        text_channel = await inter.guild.create_text_channel(f"📌 · Керування", category=category,
)
```

```
            chat_channel = await
inter.guild.create_text_channel(f"💬 ・Чат", category=category)
            voice_channel = await
inter.guild.create_voice_channel(f"🟢 ・Івент", category=category)
            await inter.send(
                f" {disnake.utils.get(inter.guild.roles,
id=731099776585826355).mention} Почато новий івент
{self.values[0]}")
```

Після створення події, ведучий може увійти в канал "керування", де він може змінити кількість учасників, відкрити або закрити голосовий канал, а також завершити подію, представлено на рисунку 4.13.

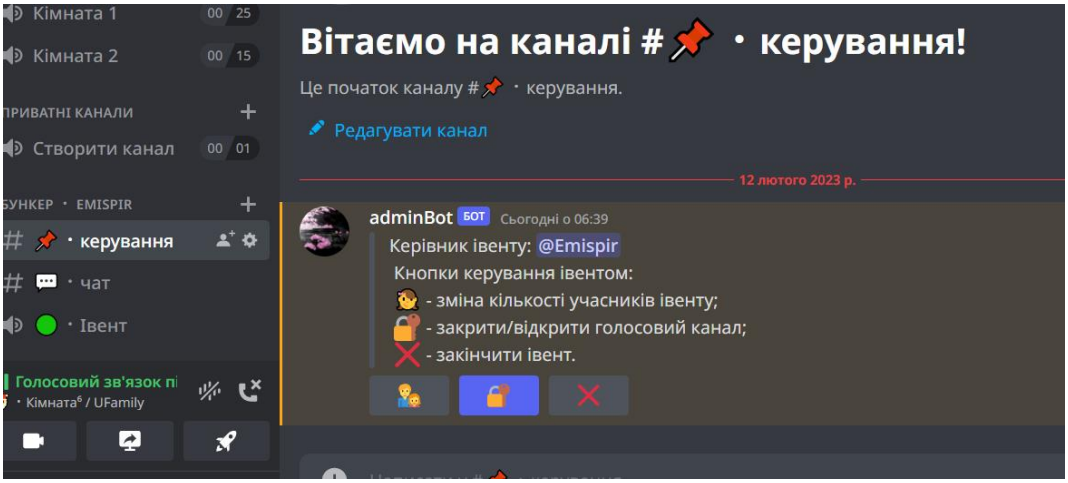


Рисунок 4.13 – Відображення каналу для керування

При зміні кількості учасників події, створюється обмеження на кількість учасників голосового каналу, представлено на рисунку 4.14.

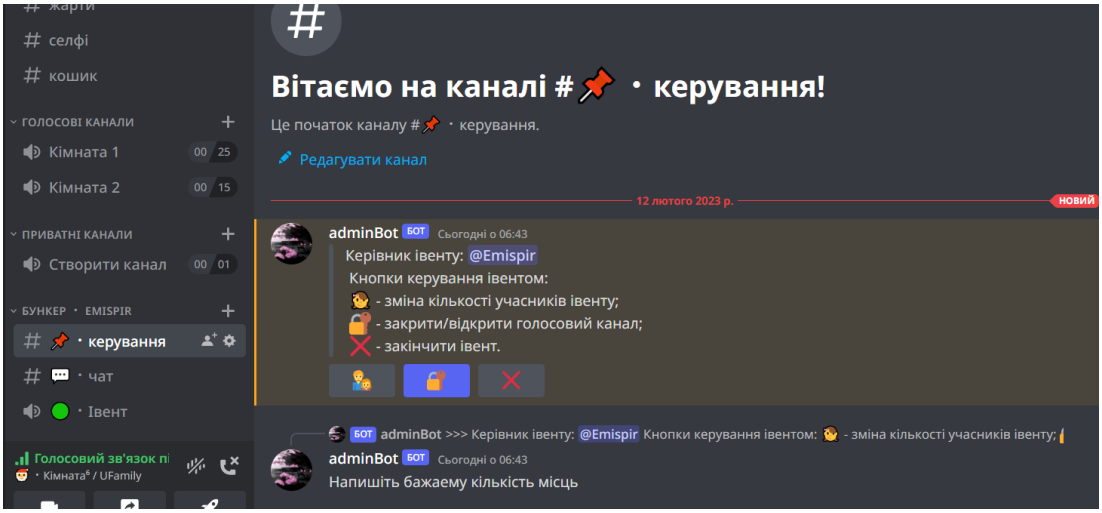


Рисунок 4.14 – Застосування команди

На рисунку 4.15 представлено написання бажаної кількості місць.

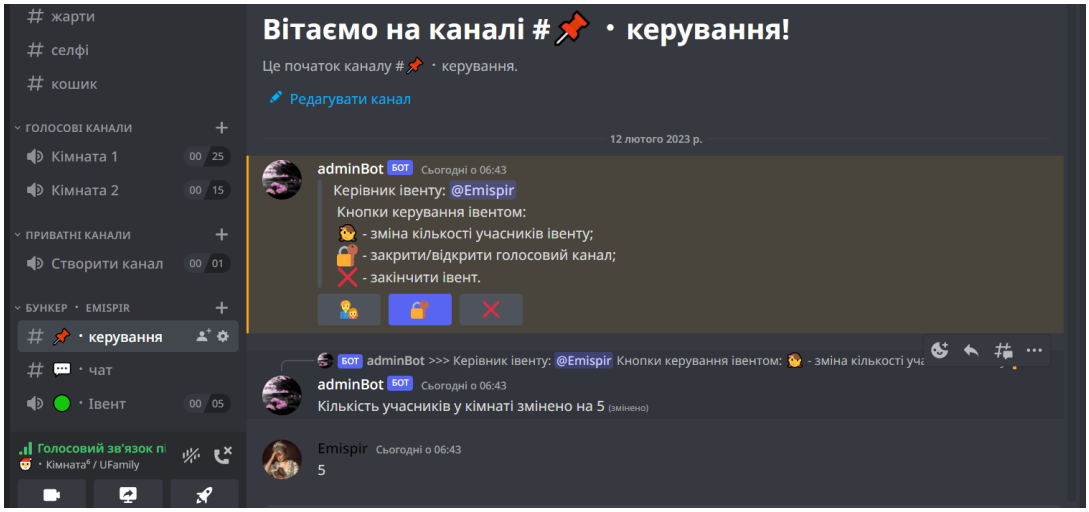


Рисунок 4.15 – Написання бажаної кількості місць

Якщо голосовий канал буде закритий, ніхто не зможе увійти в нього. Якщо ж канал буде відкритий без обмежень, учасники зможуть приєднатися до нього, представлено на рисунку 4.16.

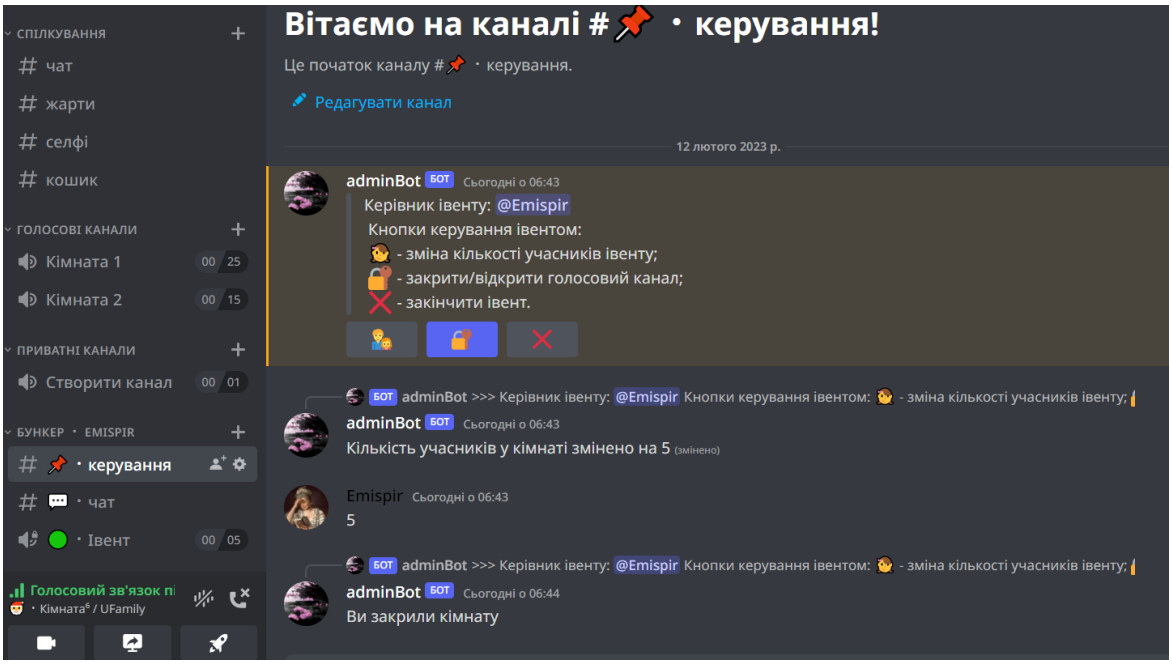


Рисунок 4.16 – Відображення закриття кімнати

На рисунку 4.17 представлено повідомлення про відкриття кімнати.

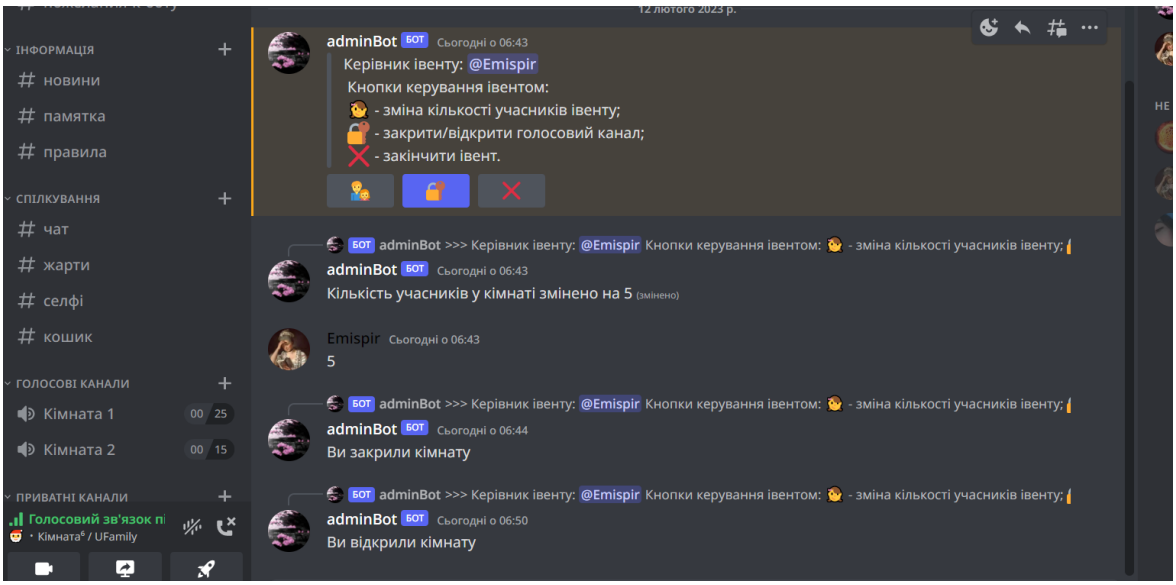


Рисунок 4.17 – Відображення відкриття кімнати

Лістинг коду керування подією:

```
class EventManage(disnake.ui.View):
    message: disnake.Message

    def __init__(self, bot, author,text_channel,chat_channel,voice_channel,category):
        super().__init__(timeout=999999)
        self.bot = bot
        self.value = None
        self.author = author
        self.text_channel = text_channel
        self.chat_channel = chat_channel
        self.voice_channel = voice_channel
        self.category = category

    async def on_timeout(self):
        for button in self.children:
            button.disabled = False
        await self.message.edit(view=self)

    async def interaction_check(self, interaction: disnake.ApplicationCommandInteraction) -> bool:
        conn = sqlite3.connect('database/event.db')
```

```
            curs = conn.cursor()
            event = curs.execute("SELECT * FROM event WHERE event_id
= ?",
                                 (self.category.id,)).fetchone()
            conn.commit()
            conn.close()
            member = disnake.utils.get(interaction.guild.members,
id=event[1])
            if interaction.author.id != member.id:
                return await interaction.response.send_message(
                    "Ви не можете керувати даними кнопками!",
ephemeral=True
                )
            return True
```

Після закінчення події, система надсилає повідомлення про завершення події та видаляє категорію події разом з усіма створеними каналами, представлено на рисунку 4.18.

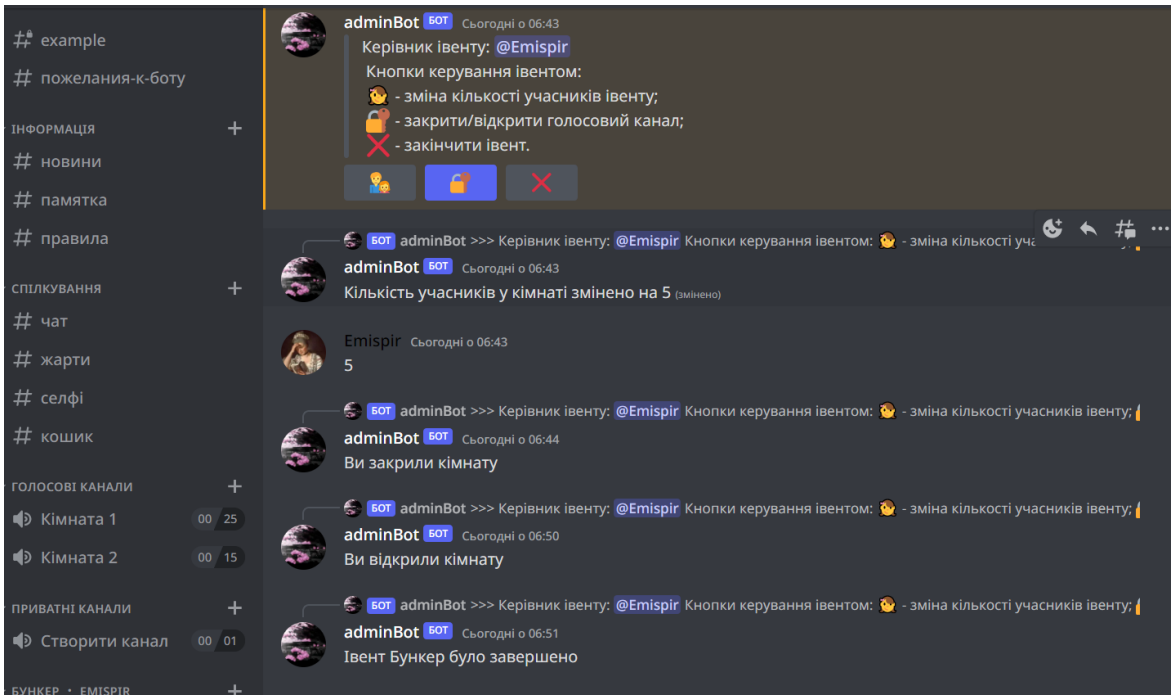


Рисунок 4.18 – Повідомлення про закриття події

Відображення видалення категорії каналів представлено на рисунку 4.19.

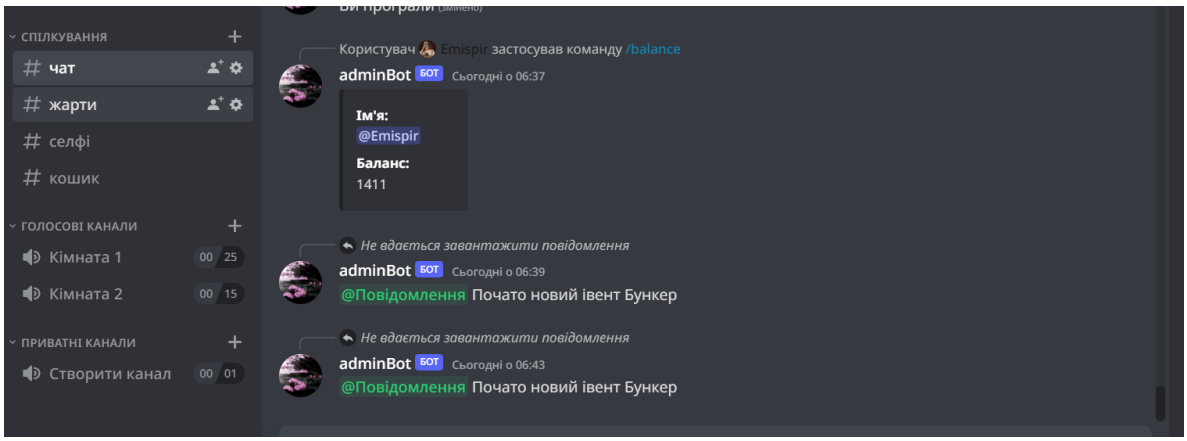


Рисунок 4.19 – Відображення видалення категорії

По закінченню події кожен учасник отримує нагороду, а переможець події отримує збільшену суму серверної валюти. Для цього ведучий використовує команду /prize та обирає статус учасника або переможця, представлено на рисунку 4.20.

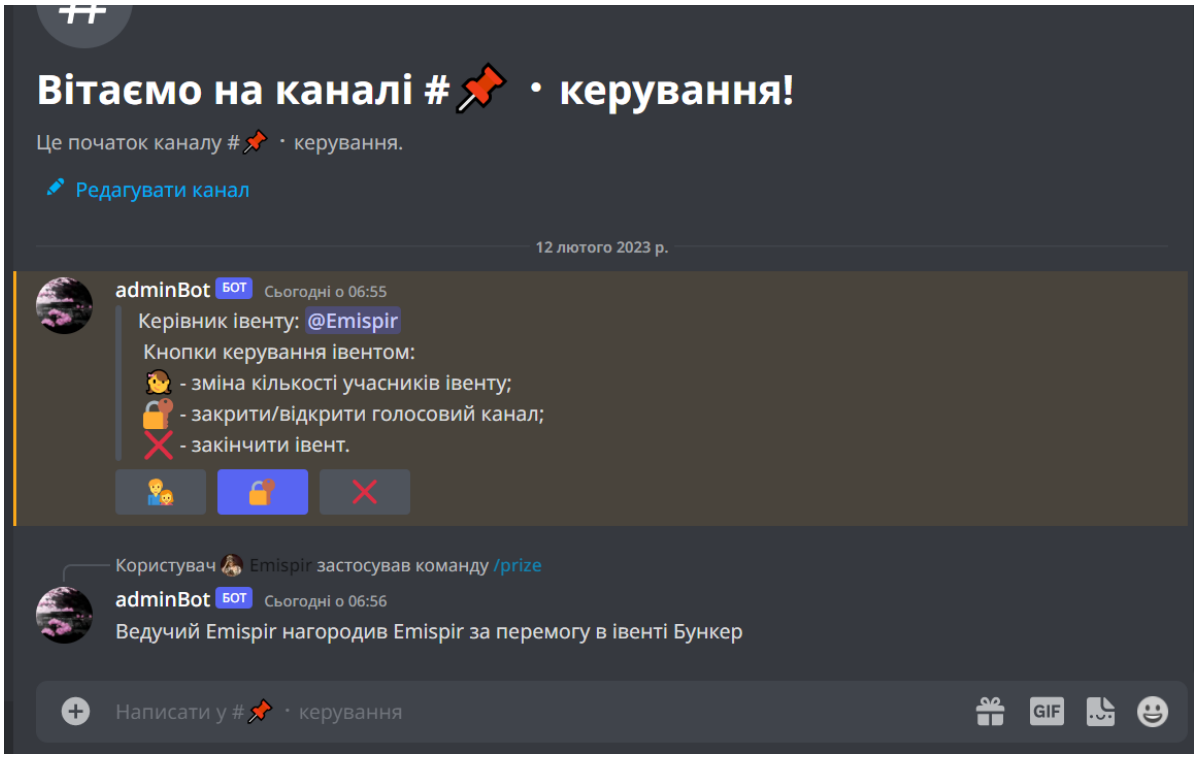


Рисунок 4.20 – Отримання переможцем винагороди за подію

Отримання винагороди учасником події, представлено на рисунку 4.21.

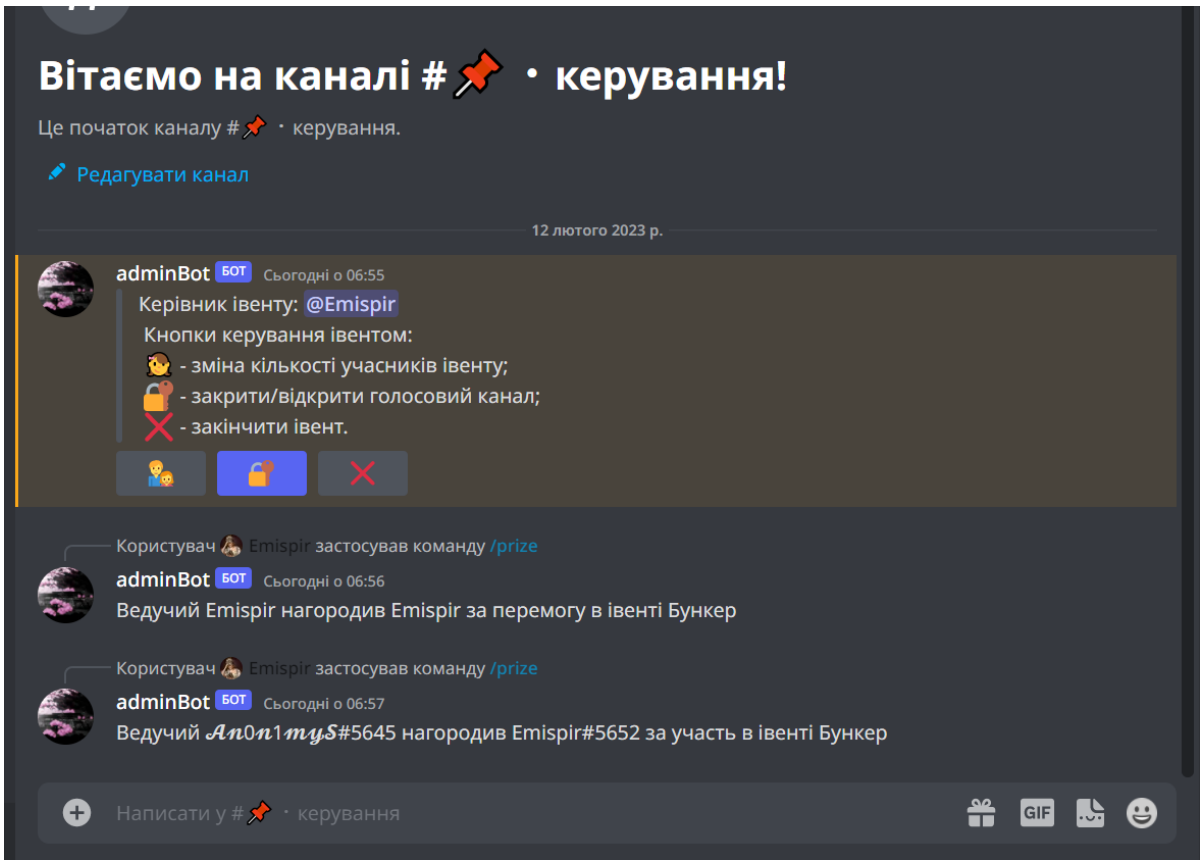


Рисунок 4.21 – Отримання учасником винагороди за подію

Після видачі нагород, бот надсилає повідомлення про успішну видачу винагороди, представлено на рисунку 4.22.

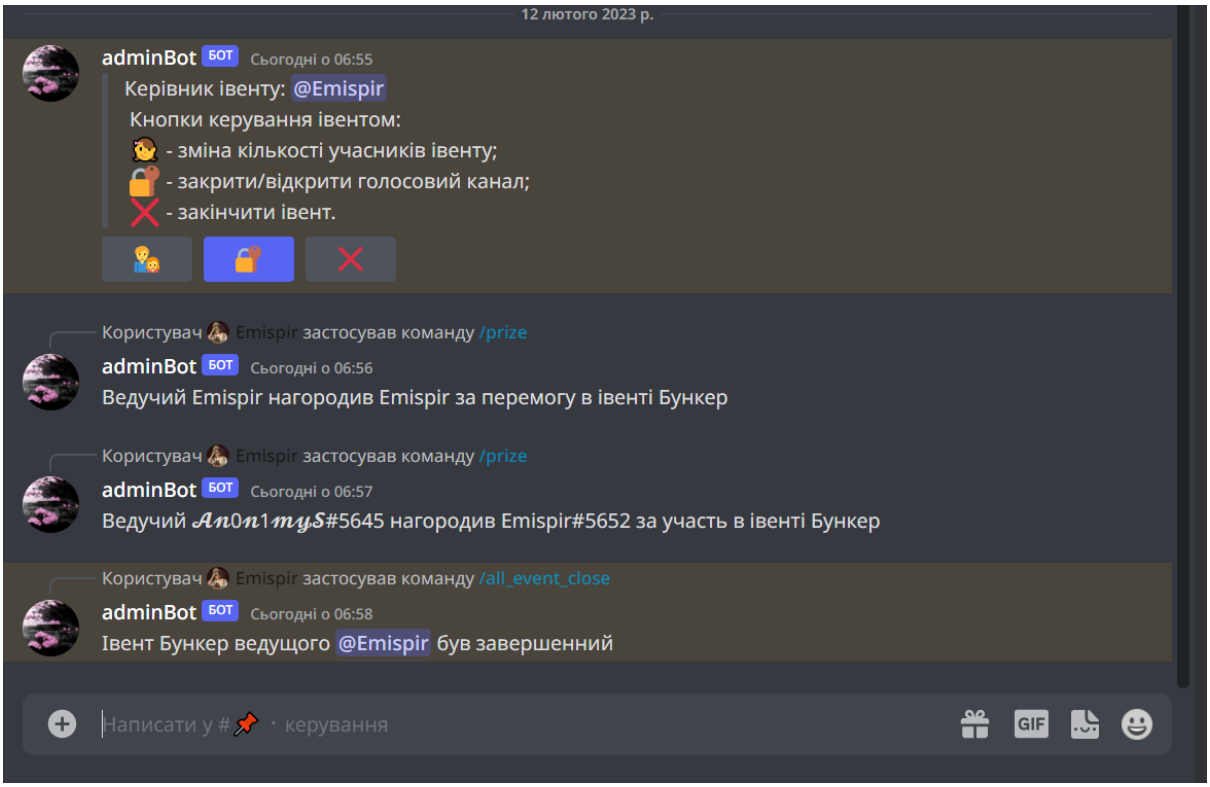


Рисунок 4.22 – Повідомлення про успішну видачу винагороди

Лістинг коду видачі призу:

```
    @commands.slash_command(name="prize", description="Видати
винагороду за івент")
        async def prize(self, ctx, target: Prize, member:
disnake.Member):
            print(f"member:{ctx.author} func: /prize")
            conn = sqlite3.connect('database/event.db')
            curs = conn.cursor()
            event = curs.execute("SELECT * FROM event WHERE authorid
= ? AND status = ?",
                                 (ctx.author.id,
"Active")).fetchone()

            if event == None:
                await ctx.send("Неможливо видати винагороду за
завершенний івент")
                conn.commit()
                conn.close()
            else:
                conn.commit()
                conn.close()
                conn = sqlite3.connect('database/users.db')
                curs = conn.cursor()
                user_data = curs.execute("SELECT * FROM users WHERE
userid = ?",
                                         (member.id,)).fetchone()

                if target == "Учасник":
                    prize = 75
                    new_balance = user_data[5] + prize

                    curs.execute("UPDATE users SET balance = ? WHERE
userid = ?", (new_balance, member.id,))
```

```
                    await ctx.send(f" Ведучий {ctx.author} нагородив {member} за участь в івенті {event[2]}")
                    conn.commit()
                    conn.close()
                elif target == "Переможець":
                    prize = 125
                    new_balance = user_data[5] + prize

                    curs.execute("UPDATE users SET balance = ? WHERE userid = ?", (new_balance, member.id,))

                    await ctx.send(f" Ведучий {ctx.author.display_name} нагородив {member.display_name} за перемогу в івенті {event[2]}")
                    conn.commit()
                    conn.close()
```

**Висновки до розділу 4**

В результаті проведеної роботи створено комплексну систему адміністрування для публічного українського сервера в Discord, що охоплює різноманітні аспекти керування та взаємодії з користувачами. Система складається з кількох модулів, кожен з яких відповідає за окремий напрямок адміністрування та забезпечує ефективне і зручне управління сервером.

Запровадження цих модулів значно покращило функціональність та гнучкість управління сервером, зробило процес адміністрування більш структурованим та ефективним. Система забезпечує чіткий розподіл обов'язків між різними ролями адміністрації та дозволяє швидко реагувати на потреби спільноти.

# ВИСНОВКИ

В результаті проведеної роботи створено комплексну систему адміністрування для публічного українського сервера в Discord, що охоплює різноманітні аспекти керування та взаємодії з користувачами. Система складається з кількох модулів, кожен з яких відповідає за окремий напрямок адміністрування та забезпечує ефективне і зручне управління сервером.

Під час аналізу предметної області виявлено ключові вимоги боту Discord, що включають його основні функціональні можливості та інтерфейси взаємодії з користувачами.

Розроблено діаграми варіантів використання допомогла структурувати можливі сценарії взаємодії користувачів з ботом, визначити ключові функції та акторів системи.

Діаграма класів була створена для визначення структури даних і взаємозв'язків між компонентами системи, що сприяє її модульності та масштабованості.

Розроблено діаграми послідовностей дій системи дала змогу детально проробити логіку обробки запитів і реакції системи, забезпечивши надійність та ефективність реагування на дії користувача.

У третьому розділі кваліфікаційної роботи розроблено діаграми класів і модель бази даних. Діаграми класів детально відображають структуру та взаємодію основних компонентів системи. Вони допомагають візуалізувати об'єктно-орієнтовану модель проекту, показуючи зв'язки між класами, їх атрибути та методи. Це є важливим інструментом для розуміння логіки та архітектури програмного забезпечення, яке розробляється.

Модель бази даних визначає структуру зберігання даних та їх взаємозв'язок. Вона включає детальний опис таблиць бази даних, їх полів, типів даних, а також ключів та індексів. Це забезпечує ефективне зберігання та доступ до даних, що є критично важливим для функціонування системи.

Також у цьому розділі детально описано вибір засобів розробки. Мову програмування Python обрано завдяки її простоті, читабельності та широким можливостям для розробки різноманітних програмних рішень. Python має велику кількість бібліотек та фреймворків, що значно спрощує процес розробки.

Використання бібліотеки Disnake.py дозволяє легко інтегрувати додаток з платформою Discord, забезпечуючи розширені можливості для взаємодії з користувачами через ботів. Disnake.py надає зручні інтерфейси для роботи з Discord API, що спрощує створення функціоналу бота.

Запровадження цих модулів значно покращило функціональність та гнучкість управління сервером, зробило процес адміністрування більш структурованим та ефективним. Система забезпечує чіткий розподіл обов'язків між різними ролями адміністрації та дозволяє швидко реагувати на потреби спільноти.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Discord Developer Portal: URL: https://discord.com/developers/applications (Дата звернення: 03.04.2024)
2. Official site PyCharm: URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата звернення: 04.04.2024)
3. Discord servers monitoring site: URL: https://myserver.gg/?language=ua (дата звернення: 05.04.2024)
4. Official documentation SqLite: URL: https://www.sqlite.org/index.html (дата звернення: 05.04.2024)
5. CASE: URL: https://uk.wikipedia.org/wiki/CASE (дата звернення: 06.04.2024)
6. Discord server Mikone: URL: https://discord.gg/squa (дата звернення: 06.04.2024)
7. Discord server UFamily: URL: https://discord.gg/ufamily (дата звернення: 07.04.2024)
8. Official site of the disnake.py library: URL: https://docs.disnake.dev/en/stable/ (Дата звернення: 08.04.2024)
9. Official site of the discord.py library: URL: https://discordpy.readthedocs.io/en/stable/api.html (дата звернення: 10.04.2024)
10. Discord server Squad Ukraine: URL: https://discord.gg/squa (дата звернення: 12.04.2024)
11. Official GitHub page of the Disnake.py library developers: URL: https://github.com/DisnakeDev/disnake (дата звернення: 15.04.2024)
12. Tutorial about installing disnake: URL: https://pypi.org/project/disnake/ (дата звернення: 15.04.2024)
13. Tutorial videos about bot development with Python and Disnake.py: URL: https://www.youtube.com/watch?v=cCiqcu2NP8I&list=PL-7Dfw57ZZVRB4N7VWPjmT0Q-2FIMNBMP (дата звернення: 16.04.2024)

14. GitHub project using disnake.py, ui.buttons: URL: https://github.com/DisnakeDev/disnake/blob/master/disnake/ui/button.py (дата звернення: 01.05.2024)

15. Official site SqLiteStudio: URL: https://sqlitestudio.pl/ (дата звернення: 02.05.2024)

16. The Python Programming Language : підручник. Ramalho L. Fluent python. "O'Reilly Media, Inc.", 2022.

17. Official site StartUml: URL: https://staruml.io/ (дата звернення: 02.05.2024)

18. Official site with documentation ErWin Data Modeler: URL: https://www.erwin.com/products/erwin-data-modeler/ (дата звернення: 02.05.2024)

19. The Python Programming Language : підручник. McKinney W. Python for data analysis. " O'Reilly Media, Inc.", 2022.

20. The Python Programming Language : підручник. Subasi A. Practical machine learning for data analysis using python. Academic Press, 2020.