Міністерство освіти і науки України

Університет митної справи та фінансів

Факультет інноваційних технологій

Кафедра комп’ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка кросплатформового Discord-бота з використанням штучного інтелекту»

Виконав: студент групи К-20-2

Спеціальність 122 «Комп’ютерні науки»

Доманський Ф. Р.

Керівник: к.ф.-м.н., доц. Рудянова Т. М.

Рецензент: Університет митної справи та фінансів

(місце роботи)

доцент кафедри кібербезпеки та інформаційних технологій

(посада)

к.т.н., доц. Клим В. Ю.

(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Доманський Ф.Р. «Дослідження особливостей технологій розробки та програмна реалізація бота в кросплатформенній пропрієтарній системі Discord».

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2023.

У кваліфікаційній роботі було розроблено прототип Discord-бота на мові програмування Java. Discord-бот є актуальним у контексті електронного навчання та розважальних цілей, оскільки він використовується для виконання унікальних функцій у віртуальних спільнотах.

Проблема полягає у тому, що більшість українських розробників створюють стандартні боти з обмеженим функціоналом, які не привертають увагу користувачів та не сприяють розвитку українського штучного інтелекту, тому обрана тема є актуальною.

Метою даної кваліфікаційної роботи є створення унікального Discord-бота на мові програмування Java, що використовує штучний інтелект.

Потенціал такого проекту полягає в можливості привернути широку аудиторію гравців та користувачів, зацікавлених у взаємодії з інтелектуальними алгоритмами бота.

Вирішенням проблеми є створено проекту з унікальним функціоналом та візуальним стилем. Під унікальним функціоналом вважається розробка унікального інтерфейсу для розваги та допомоги. Бот розроблювався на середовищі розробки IntelliJ IDEA, який на час 2024 року, є одним із популярніших в Україні.

Ключові слова: бот, робот, discord-бот, java, розваги, discord, штучний інтелект, допомога, онлайн, користувачі.

ANNOTATION

Domansky F.R. «Investigation of features of development technologies and software implementation of a bot in the cross-platform proprietary system Discord».

Qualification work for obtaining a bachelor's degree in the specialty 122 "Computer Science". – University of Customs and Finance, Dnipro, 2023.

In this qualifying work, a prototype of the Discord-bot was developed in Java. The Discord-bot is relevant in the context of e-learning and entertainment purposes because it is used to perform unique functions in virtual communities. The problem is that most Ukrainian developers create standard bots with limited functionality that do not attract the attention of users and do not contribute to the development of Ukrainian artificial intelligence.

The purpose of this qualification work is to create a unique Discord-bot in the Java programming language that uses artificial intelligence. The potential of such a project lies in the ability to attract a wide audience of players and users interested in interacting with intelligent bot algorithms.

The solution to the problem will be the creation of a project with a unique functionality and visual style. Unique functionality means developing a unique interface for entertainment and assistance. The bot was developed on the IntelliJ IDEA development environment, which as of 2024 is one of the most popular in Ukraine.

Keywords: bot, robot, discord bot, java, entertainment, discord, artificial intelligence, help, online, users.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ООП – Об’єктно-орієнтоване програмування.

JDA – Java Discord API

API – Application Programming Interface

BOT – Автоматизована програма

HTTP – протокол передачі тексту

IDE – комплексне програмне рішення

Faceit – кіберспортивна платформа

ЗМІСТ

ВСТУП

Індустрія Discord-ботів – це галузь, що займається створенням і розробкою ботів. Вона об’єднує творчих людей – розробників, програмістів та багатьох інших, які працюють разом, щоб створити розважальних та багатофункціональних ботів для користувачів Discord. Вона дуже різноманітна і включає в себе різні типи ботів, такі як універсальні боти, ігрові боти, помічники, інформаційні боти, музикальні боти та боти для модерації сервера. Розробка ботів потребує поєднання креативності, технічних знань і спільної роботи команди.

Індустрія Discord-ботів є дуже популярною і має широку аудиторію користувачів по всьому світу. Боти стають засобом розваги, соціалізації та зручності. Вони можуть бути веселими, захоплюючими, навчальними або навіть викликати емоції. Крім того, індустрія Discord-ботів має значний економічний вплив. Вона створює автоматизацію та оптимізацію робочих процесів, приносить дохід від нових ринків та послуг, підвищує конкурентоспроможність і сприяє розвитку технологій. Багато компаній залучають інвестиції для створення нових ботів та інноваційних розробок. Індустрія Discord-ботів постійно змінюється та розвивається, впроваджуючи нові технології, тренди та ідеї. Вона стала невід’ємною частиною сучасної культури та медіа, і продовжує радувати користувачів новими автоматизованими витворами.

Тема «Розробка кросплатформенного Discord-бота з використанням штучного інтелекту» є актуальною із-за декількох причин:

- зростання популярності системи Discord;
- популярність інтеграції штучного інтелекту;
- потреба в інноваціях;
- розвиток кросплатформенних рішень.

З розвитком технологій ботів стали актуальними в нашому світі. Вони допомагають зняти труднощі у житті, покращити настрій автоматизацією і допомогти з чимось. Багато ботів підтримують універсальний режим, що дозволяє користувачам використовувати налаштування бота для автоматизації та багатофункціональності для чого завгодно.

Це створює можливість для зручної взаємодії бота та соціалізації з іншими користувачами з усього світу [1]. Галузь ботів постійно розвивається та впроваджує нові технології, такі як машинне навчання та природну мову, розподілені системи та обчислення в хмарі, автоматизація процесів, інтеграція зі сторонніми сервісами та API. Боти стають платформою для впровадження та випробування нових інновацій., таких як штучний інтелект та машинне навчання.

Боти стають платформою для випробування різних методів штучного інтелекту та машинного навчання. Вони можуть використовуватися для розробки інтелектуальних систем, які навчаються взаємодіяти з користувачами та вирішувати різноманітні завдання.

Метою кваліфікаційної роботи є аналіз сучасних технологій розробки та програмна реалізація кросплатформового Discord-бота.

Основними завданнями кваліфікаційної роботи є: вибір та обґрунтування парадигми програмування для подальшої роботи; дослідження бот-індустрії; вибір програмного забезпечення для розробки та обґрунтування обраних засобів; аналіз існуючих конкурентів; формування технічного завдання; розробка програмної реалізації та її опис.

Об’єктом дослідження є: технології розробки Discord-бота.

Предмет дослідження: Розробка кросплатформового Discord-бота з використанням штучного інтелекту , яка об’єднала у собі ознаки покращення користувацького досвіду, розширення функціональності Discord, створення нових можливостей взаємодій та дослідження можливостей інтеграції.

У кваліфікаційній роботі були використанні такі методи дослідження: метод узагальнення, метод аналізу та синтезу, метод аналогії, описовий метод.

У результаті виконання кваліфікаційної роботи були підтвердженні наступні результати навчання, які відповідають освітній програми 122 «Комп’ютерні науки»:

– Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проектування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв’язності та нерозв’язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем.

– Застосувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації в предметній області комп’ютерних наук.

– Використовувати методи обчислювального інтелекту, машинного навчання, нейромережевої та нечіткої обробки даних, генетичного та еволюційного програмування для розв’язання задач розпізнавання, прогнозування, класифікації, ідентифікації об’єктів керування тощо.

– Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі комп’ютерних наук.

– Застосовувати знання методології та CASE-засобів проектування складних систем, методів структурного аналізу систем, об’єктно орієнтованої методології проектування при розробці і дослідженні.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел, що містить 23 джерел, додатку на 33 сторінках. Кваліфікаційна роботу містить 27 рисунків, загальна кількість сторінок – 54 сторінок.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис розвитку штучного інтелекту

Початки (1950-і роки): Перші кроки в розвитку штучного інтелекту були зроблені в 1950-их роках. У цей період було запропоновано та розроблено перші моделі штучного інтелекту, такі як програми гри в шахи та програми, що виконують логічні операції [2].

Літній період (1956-1974 роки): У 1956 році відбувся відомий Дартмутський літній дослідницький проект, який вважається початком активного розвитку галузі. У цей період було розроблено перші мови програмування для штучного інтелекту, такі як Lisp, а також виникли нові концепції, такі як вирішення задач на основі знань.

Період занепаду (1974-1980 роки): Наступні роки були відзначені регресом у розвитку штучного інтелекту. Це було пов'язано з обмеженнями технологій та недоліками в алгоритмах, які не дозволяли досягти значних досягнень.

Відродження (1980-1990 роки): У 1980-их роках розвиток штучного інтелекту отримав друге подих. Були введені нові методи та підходи, такі як експертні системи, нейронні мережі та генетичні алгоритми. Цей період відзначився значними досягненнями в області розпізнавання образів, обробки природної мови та ігрового програмування.

Експансія та розвиток (1990-2000 роки): У 1990-і роки штучний інтелект продовжив свій розвиток, зокрема завдяки зростанню обчислювальної потужності та доступності великих обсягів даних для навчання моделей машинного навчання.

Сучасний період (2000-нинішність): Сучасний період розвитку штучного інтелекту характеризується великими досягненнями в області глибокого навчання, розпізнавання образів, обробки природної мови,

автономної навігації, медицини та інших сферах. Штучний інтелект знаходить застосування у багатьох галузях, від технологічних компаній до медичних установ та фінансових установ [3].

Українські компанії активно сприяють створенню нових можливостей для нашої країни шляхом реалізації нових проектів та ініціатив. Інноваційні технології, такі як штучний інтелект, криптовалюти та блокчейн, відіграють ключову роль у цьому процесі. Універсальні AI-системи: Створення багатофункціональних систем штучного інтелекту. Інтеграція у повсякденне життя: Використання AI у різних сферах, від медицини до розваг..

Наприклад, компанія WhiteBIT, одна з найбільших криптовалютних бірж в Україні, активно використовує та інтегрує штучний інтелект у всі сфери своєї діяльності. Команда компанії успішно впроваджує штучний інтелект у різні напрями роботи, що дозволяє їй ефективно функціонувати та розвиватись [4].

Технологічна компанія Roosh, яка спеціалізується на штучному інтелекті та машинному навчанні, впроваджує штучний інтелект в Україні та допомагає підприємцям та технологічним компаніям розширюватись на міжнародному ринку. Їхній внесок полягає у створенні сприятливого середовища для розвитку нових штучно-інтелектуальних стартапів та розширення галузі інформаційних технологій .

У грудні 2020 року Кабмін затвердив концепцію розвитку штучного інтелекту до 2030 року. Для нашої країни концепція розвитку ШІ — це великий крок, який допоможе інтегрувати інноваційні технології в економічно важливі сектори держави.

Огляд розвитку штучного інтелекту підкреслює ключові етапи та досягнення, що сприяли еволюції технології від її зародження до сучасного стану.

Концепція розвитку штучного інтелекту в Україні на рисунку 1.1

Рисунок 1.1 – Картина розвитку ШІ

1.2 Розвиток розробників та незалежних проектів в галузі ботів

Проаналізував загальну історію розробок штучного інтелекту, можна сказати, що за популяризацію ботів стоять дві головні події. По-перше – це поява такої платформи як Flow XO. Ця платформа дозволила поширювати ботів та інший контент, при цьому не залежати від великих видавництв. З цього витікає факт – «Якби Flow XO не провів дорогу для невеликих команд розробників, боти не існували би». Навчання та розвиток: Flow XO зосереджується на підтримці розробників штучного інтелекту, надаючи доступ до різноманітних навчальних ресурсів, вебінарів та онлайн-курсів. Це допомагає розробникам поглибити свої знання в галузі штучного інтелекту та розвинути свої навички [5].

Підтримка інноваційних проектів: Flow XO активно підтримує незалежних розробників та їхні проекти в галузі штучного інтелекту. Це може включати надання фінансової підтримки, доступу до інфраструктури та технічної експертизи.

Забезпечення інфраструктури для розробки: Flow XO надає розробникам доступ до інструментів та сервісів для створення та впровадження

інноваційних проектів в галузі штучного інтелекту. Це може включати SDK, API, платформи розробки та тестування.

Створення спільноти: Flow XO створює сприятливу спільноту для розробників штучного інтелекту, де вони можуть обмінюватися досвідом, ідеями та кращими практиками. Це допомагає збільшити співпрацю та стимулює інновації в галузі.

Багато невеликих незалежних компаній та окремі розробники ботів, ніколи не матимуть таких великих ресурсів, які можуть витратити великі компанії на розробку складних проектів. Тому вони змушені шукати альтернативні шляхи, такі як створення оригінального та цікавого контенту. В інтернеті є думка, що саме незалежні розробники ботів створюють найбільш цікаві та унікальні продукти у цій галузі. Коли розробники ботів не обмежені вимогами великих компаній, вони можуть дійсно втілити свої ідеї і створити проекти, які відповідають їх власним потребам та візії. У світі розробки ботів ігрові проекти стають місцем, де розробники можуть вільно творити, не обтяжені великими вимогами. Також є і повністю експериментальні та унікальні рішення. Чат бот “Bing”, на 2023 рік, є дуже популярним в середовищі штучного інтелекту. Самого бота можна описати наступними термінами:

Мовний модель: ChatGPT - це мовна модель, розроблена для розуміння та генерації тексту. Вона використовується для різних завдань, включаючи відповіді на запитання, генерацію тексту та проведення діалогу з користувачем.

Штучний інтелект: ChatGPT базується на передових методах штучного інтелекту, зокрема на глибокому навчанні та нейромережевих архітектурах, які дозволяють моделі розуміти та генерувати текст.

Контекстно-залежні моделі: ChatGPT використовує контекстно-залежні моделі, які враховують попередні слова або речення при генерації наступного тексту, що допомагає забезпечити більш природні та зв'язані відповіді.

Генерація тексту з контекстом: ChatGPT вміє враховувати контекст в тексті, що вводить користувач, та відповідати адекватно на запитання або коментарі, використовуючи цей контекст для кращого розуміння.

На 2023 рік можна легко завантажити середовище для розробки ботів комерційного рівня і почати з ним працювати. Але при всіх цих плюсах, потрібно не забувати про мінуси розробки ботів. Наприклад, взаємодія з аудиторією, її залучення, і взагалі – великою проблемою більшості компаній є саме донесення їх проектів до тої аудиторії, до якої вони повинні дійти по всьому світу.

1.3 Автоматизація рутинних завдань

Обробка даних: Discord-бот може автоматизувати процес збору, аналізу та обробки великих обсягів даних. Наприклад, він може аналізувати великі набори даних та виділяти в них зв'язки та закономірності без необхідності вручну переглядати кожен запис.

Комунікація та взаємодія з клієнтами: Discord-бот з використанням штучного інтелекту може автоматизувати комунікацію з клієнтами через чат-ботів або віртуальних асистентів. Він може відповідати на запитання, надавати інформацію та навіть виконувати операції обробки замовлень без необхідності участі живої людини.

Моніторинг та управління системами: Він може виявляти аномалії, прогнозувати відмови та приймати рішення щодо їх управління для запобігання можливим проблемам [6].

1.4 Аналіз технологій реалізації Discord-ботів конкурентів

Важливим завданням кваліфікаційної роботи є створення унікального бота та функціонал для подальшого процесу розробки бота, тому потрібно проаналізувати готові проекти.

Першою асоціацією, яка може прийти у голову користувача є бот “Dyno Bot” від компанії Swift Media Entertainment. Бот Dyno - це один з найпопулярніших ботів для Discord, який надає різноманітні корисні функції для адміністрування та керування сервером. Він допомагає у забезпеченні порядку на сервері та надає ряд інструментів для полегшення роботи адміністраторів та модераторів.

Деякі з основних можливостей бота Dyno включають модерацію чату, автоматичні повідомлення, налаштування рівнів доступу та прав, автоматизовані повідомлення про привітання нових учасників сервера, а також захист від спаму та небажаних повідомлень. Крім того, він також має можливість працювати з музикою, створюючи можливість для користувачів прослуховувати музику безпосередньо на сервері.

Завдяки своїй широкій функціональності та простому у використанні інтерфейсу, бот Dyno став популярним інструментом для багатьох серверів Discord, допомагаючи їм забезпечувати ефективне управління та комунікацію [7].

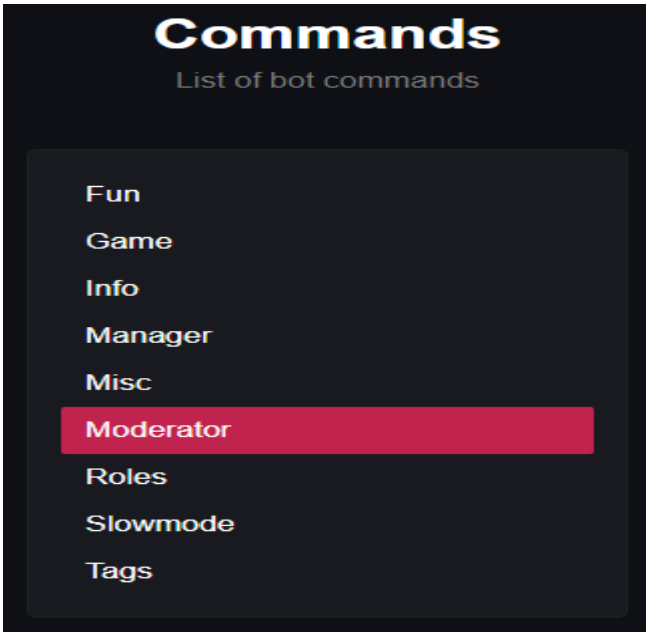

Рисунок 1.2 – Функціонал бота Dyno

Як видно з рисунку 1.2, бот має зручний та яскравий візуальний стиль функціоналу, який відображається у вигляді таблиці та зручних для перегляду

команд, карикатурний підхід до дизайну персонажів та об’єктів, що створює своєрідну атмосферу гри. Модерація чату:

Dyno Bot дозволяє адміністраторам контролювати поведінку учасників чату, включаючи заборону небажаної мови, спаму та інших порушень правил сервера.

Автоматичні повідомлення: Бот може налаштовувати автоматичні повідомлення, які вітають нових учасників сервера або надають іншу інформацію, наприклад, правила сервера.

Ролі та права доступу: Dyno Bot дозволяє налаштовувати ролі та права доступу для учасників сервера, що дозволяє забезпечити правильну ієрархію та контроль доступу до різних функцій.

Автоматичні реакції: Бот може реагувати на певні слова або фрази, надсилаючи автоматичні відповіді або реагуючи на них певним чином.

Музичні команди: Dyno Bot підтримує музичні команди, які дозволяють користувачам прослуховувати музику безпосередньо на сервері.

Система голосового чату: Бот може створювати та керувати голосовими каналами для спілкування учасників сервера.

Система автоматичних повідомлень: Dyno Bot може надсилати автоматичні повідомлення на основі певних умов або розкладу, наприклад, повідомлення про розклад подій або оновлення.

Загалом, Dyno Bot - це потужний інструмент, який дозволяє адміністраторам серверів забезпечити ефективне керування та модерацію, а також створити приємне середовище для спілкування учасників.

Іншою, менш популярною асоціацією, може стати бот MEE6 . Це теж AI Discord. Рівні доступу та система рангів: MEE6 дозволяє налаштовувати систему рівнів, де користувачі отримують ранги за активність на сервері. Це може бути використано для мотивації та стимулювання учасників [8].

Автоматичні повідомлення: Бот може надсилати автоматичні повідомлення при певних подіях, таких як привітання нових користувачів, попередження про правила чату, або нагадування про важливі події.

Модерація чату: MEE6 надає інструменти для модерації чату, такі як фільтри для спаму, можливість видалення повідомлень або надання попереджень користувачам, що порушують правила.

Музика: Бот може відтворювати музику на сервері через команди, що дозволяє учасникам насолоджуватися музичним контентом безпосередньо в Discord. Користувальницькі команди: Крім вбудованих функцій, MEE6 дозволяє створювати користувачам власні команди для автоматизації певних дій чи запуску власних скриптів. Економіка сервера: Бот може вести систему економіки на сервері, де користувачі можуть заробляти валюту за активність та виконання певних завдань. Статистика сервера: MEE6 надає зручний інтерфейс для перегляду статистики сервера, такої як активність користувачів, кількість повідомлень тощо.

Загалом, MEE6 - це потужний бот з багатофункціональним набором інструментів, який допомагає адміністраторам та модераторам забезпечувати ефективне управління сервером та зручну взаємодію з користувачами.

Інтерфейс бота MEE6 в Discord простий та зрозумілий для користувачів. Основні функції та команди можна викликати за допомогою спеціальних команд, які вводяться у текстовий канал чату. Інтерфейс MEE6 спроектований для зручної та ефективної взаємодії з користувачами та адміністраторами серверів у середовищі Discord. Головний екран: Відображає поточні налаштування та активність. Кнопки дії для збереження змін, підключення додаткових плагінів або відновлення попередніх налаштувань. Інструменти для модерації, такі як автоматичні попередження, заборона та відключення звуку для порушників.

Можливість налаштування автоматичних дій для певних тригерів, наприклад, вживання ненормативної лексики або спам. Налаштування привітальних повідомлень для нових учасників сервера.

Можливість надсилати приватні повідомлення новим користувачам або публічні повідомлення в певний канал.

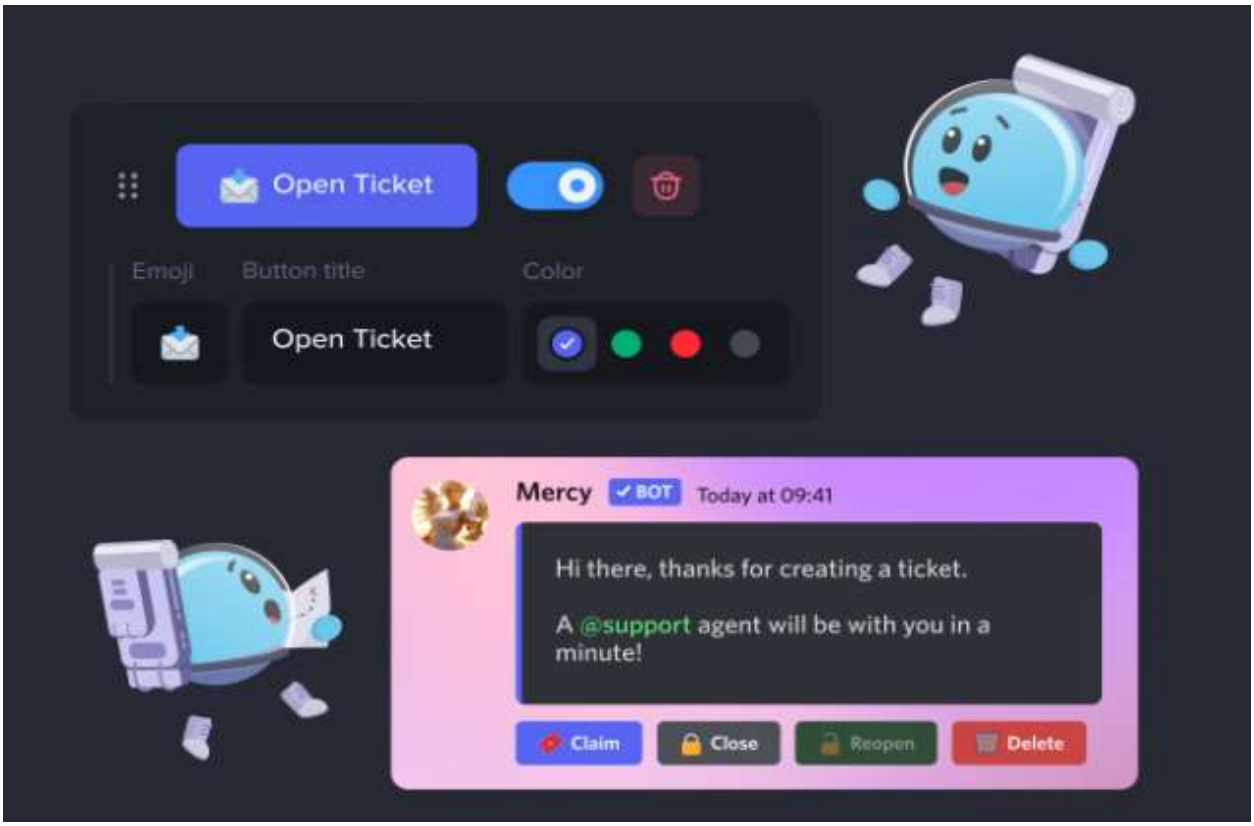

Рисунок 1.3 – Інтерфейс першої функції бота MEE6

Mafia Bot - це бот для Discord, який дозволяє користувачам грати у популярну психологічну гру "Мафія" прямо на сервері. Ось докладний опис його функціоналу та можливостей:

Створення ігрових сесій: Користувачі можуть створювати нові ігрові сесії "Мафії", встановлюючи правила та параметри гри. Обрання ролей: Учасники можуть обирати ролі для гри, такі як "Мафія", "Мирний житель", "Лікар" тощо, в залежності від налаштувань гри.

Автоматизований геймплей: Бот керує ходом гри, розподіляючи ролі, ведучи голосування та оголошуючи результати кожного раунду. Приватні повідомлення: Учасники можуть використовувати приватні повідомлення з ботом для надсилання голосів чи обговорення стратегії гри.

Графічний інтерфейс: Деякі версії бота можуть мати графічний інтерфейс, який відображає інформацію про гравців та стан гри. Налаштовані правила гри: Адміністратори сервера можуть налаштовувати правила гри, включаючи кількість гравців, типи ролей, тривалість раундів тощо.

Статистика гри: Після закінчення гри бот може надавати статистику, яка включає в себе переможців, дії гравців та інші важливі показники.

Mafia Bot дозволяє користувачам насолоджуватися грою "Мафія" прямо в Discord, створюючи веселу та захоплюючу атмосферу для всіх учасників сервера.

Інтерфейс Mafia Bot в Discord є досить простим та зрозумілим для користувачів. Для взаємодії з ботом користувачі використовують спеціальні команди, які вводяться в текстові канали. Наприклад, команда "/mafia start" може почати нову гру "Мафія" на сервері. Деякі версії Mafia Bot можуть мати графічний інтерфейс, який відображає інформацію про гравців, ролі та стан гри у зручному форматі. Інтерфейс Mafia Bot розроблений з урахуванням зручності користувачів та надає їм можливість насолоджуватися грою "Мафія" прямо у середовищі Discord без зайвих складнощів [9].

Рисунок. 1.4 – Інтерфейс функціоналу бота Mafia Bot

Groovy - це музичний бот для Discord, який дозволяє користувачам прослуховувати музику з різних джерел безпосередньо у голосових каналах. Ось докладний опис його функціоналу та можливостей [10]:

Доступ до різних джерел: Groovy підтримує доступ до мільйонів пісень з популярних музичних платформ, таких як YouTube, Spotify, SoundCloud, Bandcamp, Twitch та інші.

Просте керування відтворенням: Користувачі можуть використовувати команди бота для пошуку музики, відтворення певних пісень або плейлистів, регулювання гучності, призупинення або відновлення відтворення тощо.

Система плейлистів: Groovy дозволяє користувачам створювати власні плейлисти або додавати музику до існуючих, щоб зручно слухати улюблені треки.

Функції адміністрування: Адміністратори серверів можуть керувати доступом до функцій бота, встановлювати обмеження на кількість пісень у черзі або виключати певні можливості для учасників сервера.

Спеціальні функції: Groovy має додаткові функції, такі як можливість виконання музичних команд у текстових каналах, відображення інформації про виконавців та пісні у текстовому форматі тощо.

Groovy - це потужний та надійний музичний бот, який дозволяє користувачам насолоджуватися улюбленою музикою прямо на серверах Discord без зайвих зусиль.

Інтерфейс Groovy в Discord дуже зручний і дружній до користувача. Ось докладний опис його основних елементів:

Команди бота: Користувачі можуть взаємодіяти з Groovy, використовуючи спеціальні команди, які вони вводять у текстових каналах чату. Ці команди дозволяють керувати відтворенням музики, додавати пісні у чергу, створювати плейлисти та багато іншого.

Візуальний інтерфейс: Groovy може відображати інформацію про відтворювані треки прямо у текстових каналах, включаючи назву пісні, виконавця, обкладинку альбому та іншу інформацію. Це робить процес слухання музики більш інтерактивним і зручним для користувачів.

Управління чергою відтворення: Користувачі можуть додавати пісні у чергу відтворення, видаляти їх, переміщати вгору або вниз у черзі, змінювати

порядок відтворення та багато іншого. Це дозволяє користувачам налаштувати своє відтворення музики за своїми власними уподобаннями.

Підтримка плейлистів: Groovy підтримує роботу з плейлистами, які користувачі можуть створювати, редагувати та ділитися з іншими учасниками сервера. Це дозволяє зручно організувати свою музичну колекцію та легко знаходити улюблені треки.

Інтерактивні повідомлення: Groovy може відправляти інтерактивні повідомлення у текстових каналах з пропозиціями для вибору треків, підтвердженням операцій або відображенням інших важливих повідомлень, пов'язаних з відтворенням музики.

Настройка налаштувань: Користувачі можуть налаштовувати різні параметри відтворення, такі як гучність, якість аудіо та інші параметри зручної музичної адаптації до власних уподобань.

Загалом, інтерфейс Groovy створений з урахуванням зручності користувачів, надаючи їм зручний та простий спосіб насолоджуватися музикою прямо у текстових каналах Discord.

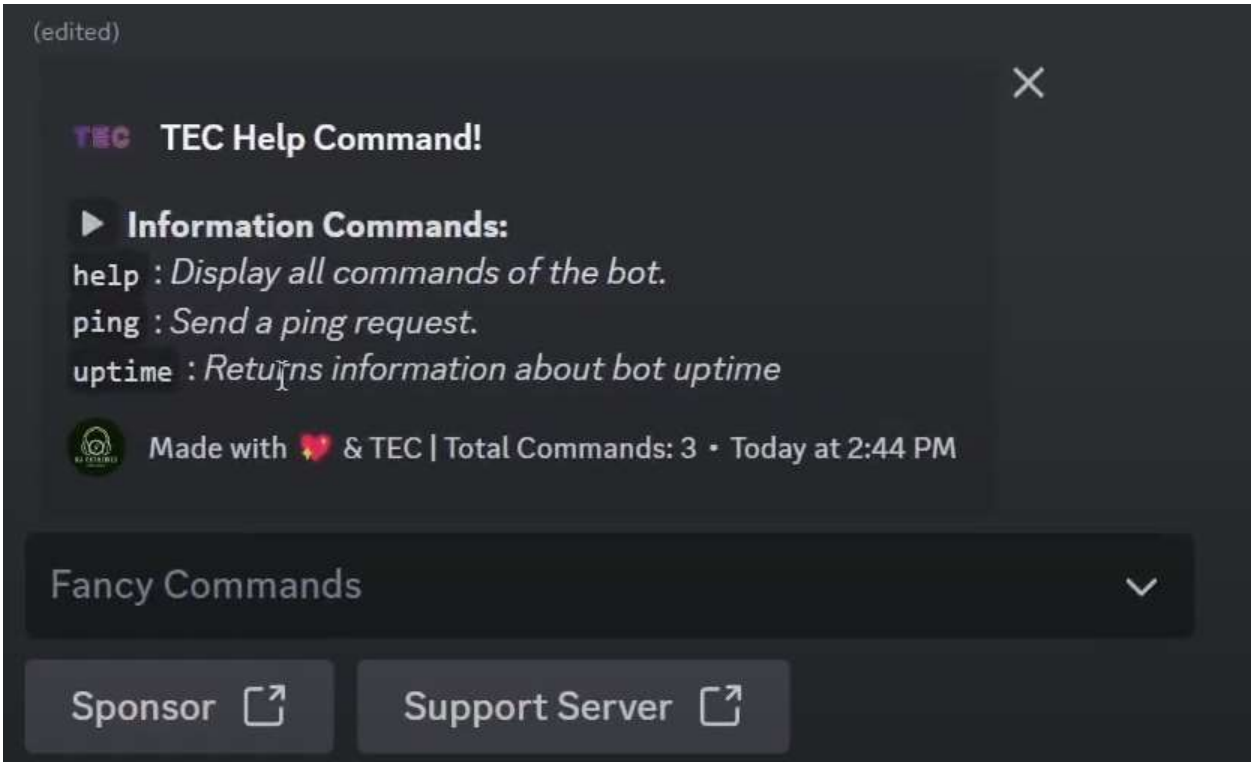

Рисунок. 1.5 - Інтерфейс функціонала бота Groovy

1.5 Висновки до першого розділу

Розробка ботів, зокрема для платформ таких як Discord, демонструє значний ріст завдяки участі незалежних розробників та малих команд. Цей розвиток сприяє інноваціям та створенню унікальних рішень, які здатні задовольнити потреби різноманітних користувачів. Незалежні проекти часто випереджають великі корпорації у впровадженні нових ідей та функціональних можливостей.

На момент написання кваліфікаційної роботи, будь-який програміст може проявити себе у розробці ботів, збагачуючи цю сферу свіжими та креативними ідеями, працювати з великими компаніями та отримувати від них підтримку. Сміливі маленькі компанії у цьому плані зробили великий внесок в розвиток індустрії штучного інтелекту та допомогли розробникам по всьому світу зрозуміти, що для того щоб створювати ботів, не обов’язково мати величезних бюджетів.

Аналіз технологій, використовуваних конкурентами для створення Discord-ботів, показує широкий спектр підходів та інструментів, що дозволяють досягти високої продуктивності та функціональності. Вивчення цих технологій допомагає ідентифікувати найкращі практики та можливі покращення для власних проектів. Конкурентний аналіз є важливим елементом стратегії розвитку, що сприяє підтриманню високого рівня якості та інноваційного продукту.

Проаналізував індустрію штучного інтелекту в Україні, можна побачити, що вона зростає великими темпами та має великий економічний вплив. Вона приваблює іноземних інвесторів та розвиває економіку країни.

РОЗДІЛ 2.
АНАЛІЗ ГОТОВИХ РІШЕНЬ ТА ЗАСОБІВ РОЗРОБКИ БОТІВ

2.1 Основні етапи створення Discord-бота

Головним технічним завданням кваліфікаційної роботи є створення бота з функціонально-автоматизованим аспектом, яка буде у собі об’єднувати цілі в розважальному аспекті та зручної допомоги. Це завдання можна сформулювати у наступні під-завдання:

- розробка розважальних аспектів функцій;
- розробка функціональних можливостей;
- розробка оптимізації інтерфейсу та взаємодії;
- тестування та вдосконалення.

Під розважальним аспектом вважається розробка мережевих функціональних ігор всередині бота. Під функціональних можливостей – створення умов, при яких кожному користувачу потрібно аналізувати ситуацію та прочитати кожен функціонал бота, щоб не виникли помилки всередині користувачів. Оптимізації інтерфейсу та взаємодії будуть самі внутрішні технології бота, її функціонал, взаємодія з іншими користувачами.

Прототипом функціоналу з грою є популярна настільна психологічна гра «Бункер» з такими ключовими характеристиками, як «Вік», «Професія», «Здоров’я» та інші [11]. Тому концепція гри повинна розвинути цей аспект та додати щось нове.

Інтерфейс функціонального меню повинен містити у собі такі речі, як:

- функція для створення ролей, перегляд пояснювальної інформації, перегляд кількісті учасників на сервері;
- функція для переходу у ігровий режим, створення кімнати очікувань для гри “Бункер” та інші;
- функція для виходу з ігрового режиму;

- функція для перевірки ігрового рейтингу користувача;
- функція для модерації сервера і чату;
- функція для заявок на покупку ролі на сервері;
- функція для перегляду випадкових жартів;
- функція для привітання нового користувача сервера.

Функціонал гри “Бункер” повинен відповідати наступним вимогам:

- створення кімнати для користувачів;
- створення каналів для тих хто приєднався;
- створення текстових каналів;

Свій псевдонім користувач вводить при реєстрації на кросплатформенній системи Discord. Також важливою деталлю є те, що тільки користувач, який створив кімнату, може запустити гру. Але, якщо цей користувач вийде із кімнати, право власника кімнати передається наступному користувачу. Також кімната повинна містити у собі її ім’я та список усіх користувачів, які у ній знаходяться.

Інтерфейс в грі у кожного користувача повинен відображати псевдоніми інших гравців та інші важливі характеристики.

2.2 Вибір програмних засобів для створення проекту

Для створення проекту було обрана бібліотека Java Discord Api(JDA) та об’єктно орієнтовану мову Java. Увесь код буде компілюватися у інтегрованому середовищі розробки IntelliJ IDEA [12]. На це є декілька причин:

- кросплатформенність (на JDA розробляють ботів для різних платформ, що дозволяє у майбутньому портування проекту);
- спільнота розробників (по JDA можна знайти величезну кількість інформації, документації, завжди можна знайти допомогу у вирішенні проблем);

- інтерфейс (JDA має дружній та інтуїтивний інтерфейс, що полегшує розробку ботів навіть для розробників початкового рівня. Також він надає зручні інструменти для створення функціоналу, автоматизацію та інших аспектів бота);
- широкий функціонал (JDA має величезну кількість вже вбудованих функцій, які зменшують затрати часу на розробку бота);
- можливості розширення (JDA має свій інформаційний сайт, за допомогою якого можна додавати в проект вже готові різноманітні рішення від інших розробників, або, наприклад, зручно додавати готові функції прямо із сайту. Також JDA підтримує модифікацію методів, тобто будь-який розробник, при наявності досвіду та знань, може створити зручні методи для розробки для себе, які будуть виконуватися прямо в бібліотеці без потреби компілювання коду).

Java Discord API (JDA) є потужним інструментом, який забезпечує розробникам широкі можливості для створення складних інтерактивних ботів для платформи Discord. Ця бібліотека володіє великим рівнем гнучкості та функціональності, що дозволяє реалізовувати різноманітні інтерфейси та взаємодію з користувачами на основі використання мови програмування Java [13].

JDA надає розробникам доступ до широкого спектру інструментів і можливостей, включаючи обробку подій, керування серверами та каналами, взаємодію з повідомленнями, створення реакцій, роботу з асинхронними запитами та багато іншого. Це дозволяє створювати ботів, які можуть виконувати різноманітні завдання, від управління груповими чатами до автоматизації різних процесів.

Однією з ключових переваг JDA є його висока продуктивність та ефективність. Бібліотека оптимізована для роботи з великим обсягом даних та великою кількістю одночасних запитів, що дозволяє створювати ботів, які працюють швидко та надійно навіть при великому навантаженні.

Крім того, JDA є добре документованою бібліотекою з активною спільнотою розробників. Це дозволяє отримувати підтримку та допомогу у вирішенні проблем та розробці нових функцій від інших користувачів.

Узагальнюючи, Java Discord API (JDA) є важливим інструментом для розробки ботів для Discord, який забезпечує широкі можливості та високу продуктивність, що робить його популярним вибором серед розробників, що працюють з платформою Discord.

Сам інтерфейс IntelliJ IDEA з бібліотекою виглядає на рисунку 2.5

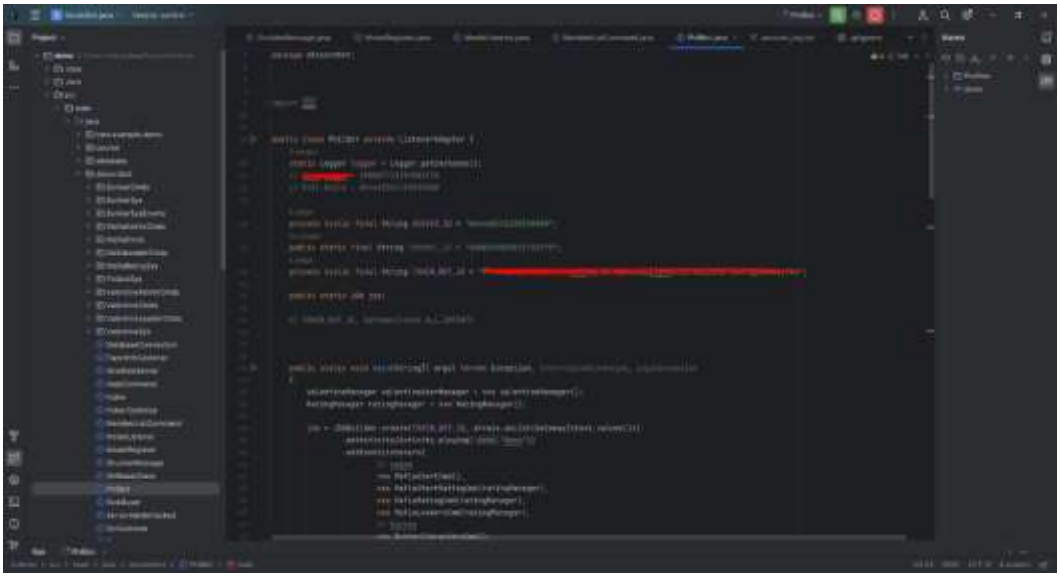

Рисунок 2.1 – Інтерфейс IntelliJ IDEA

Але, окрім усіх плюсів, бібліотека має і свої мінуси. Складність використання: JDA, як і будь-яка інша потужна бібліотека, може бути складною для новачків. Розуміння її архітектури та вивчення всіх її можливостей може вимагати значного часу та зусиль [14].

Оновлення та сумісність: Підтримка JDA може змінюватися з часом, і оновлення можуть призвести до руйнування існуючого коду або зміни у взаємодії з Discord API. Це може вимагати постійного оновлення бібліотеки та адаптації коду до нових версій.

Обмежена документація: Хоча JDA має досить добру документацію, вона може бути не вистачити деталізованою або неофіційною у деяких

випадках. Це може ускладнити розуміння деяких аспектів роботи бібліотеки та розробку відповідного коду.

Відсутність підтримки з боку Discord: JDA є сторонньою бібліотекою, тому розробка з її використанням не гарантує підтримки з боку Discord. Це означає, що ви можете зіткнутися з проблемами, які не можна вирішити безпосередньо з Discord API або внутрішніх інструментів Discord.

Обмежена спільнота: У порівнянні з деякими іншими бібліотеками для Discord, спільнота розробників JDA може бути менш активною або менш розгалуженою. Це може ускладнити отримання допомоги або знайдення відповідей на ваші запитання.

Хоча JDA є потужним інструментом для розробки ботів для Discord, важливо бути свідомим цих недоліків і враховувати їх при виборі цієї бібліотеки для вашого проекту.

2.3 Порівняння альтернатив для розробки.

В альтернативу IntelliJ IDEA, можна привести середовище як Visual Studio. На сьогоднішній день це одна із провідних платформ для розробки програми, і є кілька причин, щоб обрати її серед конкурентів:

- Широкі можливості: Visual Studio надає розробникам широкий спектр інструментів та можливостей для створення різноманітних програм. Від підтримки мов програмування до інтегрованих середовищ для роботи з базами даних та іншими компонентами, Visual Studio дозволяє розробникам працювати над проектами будь-якої складності;
- Інтегрована розробка для платформи Microsoft: Visual Studio пов'язаний з екосистемою Microsoft і надає інтегровані інструменти для розробки програм для Windows, Xbox, Azure та інших платформ Microsoft. Це робить його привабливим вибором для розробників, які працюють в екосистемі Microsoft або планують розробляти програми для цих платформ;

- Розширюваність: Visual Studio підтримує розширення, що дозволяє розробникам налаштовувати і розширювати середовище розробки під їхні потреби. Це дозволяє використовувати сторонні інструменти, плагіни та інші розширення для покращення продуктивності та розширення можливостей Visual Studio [15].

- Спільноти та підтримка: Visual Studio має активне співтовариство розробників та широкий спектр документації та підтримки. Це означає, що ви можете легко знайти відповіді на ваші запитання, розв'язати проблеми та отримати поради від інших розробників

- Інтеграція з іншими інструментами: Visual Studio інтегрується з іншими популярними інструментами розробки, такими як Git, Docker, Kubernetes тощо. Це спрощує розробку програм та робить процес розробки більш ефективним.

Обидва IDE мають свої сильні сторони і обидва є потужними інструментами для розробки програмного забезпечення. Вибір між IntelliJ IDEA та Visual Studio часто залежить від мов програмування та технологій, з якими ви працюєте, а також від ваших особистих вподобань і потреб вашої команди.

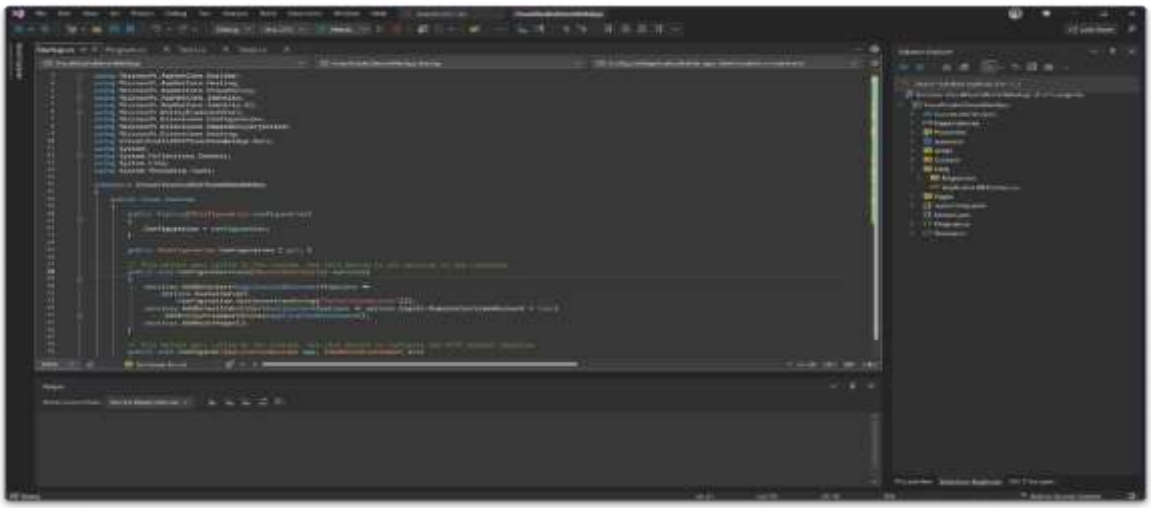

Рисунок 2.2 – Інтерфейс Microsoft Visual Studio

Отже, Visual Studio - це потужне інструментальне забезпечення для розробки програм, яке надає широкий функціонал, глибоку інтеграцію та широкі можливості для розробників будь-якого рівня досвіду (рисунок 2.2).

Ще одною альтернативою може стати інше популярне середовище – PyCharm. Він був інтегрованим середовищем розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Воно надає розробникам широкий спектр інструментів та можливостей для зручної і продуктивної роботи над проектами Python. Ось докладний огляд функцій та характеристик PyCharm [16]:

Редактор коду з підсвічуванням синтаксису: PyCharm має потужний редактор коду, який надає підсвічування синтаксису для кращого розуміння коду та швидкого виявлення помилок.

Автоматичне завершення коду: Інтелектуальне автоматичне завершення коду допомагає розробникам швидше писати код, пропонуючи можливі варіанти завершення методів, змінних та інших елементів коду.

Інтеграція з системами контролю версій: PyCharm легко інтегрується з популярними системами контролю версій, такими як Git, SVN та Mercurial, дозволяючи розробникам зручно працювати зі своїми репозиторіями коду.

Відладка коду: Вбудований відладчик PyCharm дозволяє розробникам легко виявляти та виправляти помилки у своєму коді. Він надає можливості крокування по коду, встановлення точок зупинки та перегляд значень змінних.

Управління проектами: PyCharm надає зручні інструменти для управління проектами, включаючи можливість створення віртуальних середовищ, налаштування конфігурацій проектів та організацію файлів у зручні структури.

Підтримка інструментів розробки: PyCharm інтегрується з широким спектром інструментів розробки, таких як системи керування базами даних, фреймворки веб-розробки та інші.

Підтримка роботи з Django та іншими фреймворками: PyCharm надає спеціальні інструменти для розробки веб-додатків на основі фреймворків, таких як Django, Flask та інші, що спрощує процес створення веб-програм.

Розширюваність: PyCharm підтримує розширення, які дозволяють розробникам розширювати його функціонал та налаштовувати середовище розробки під їхні потреби.

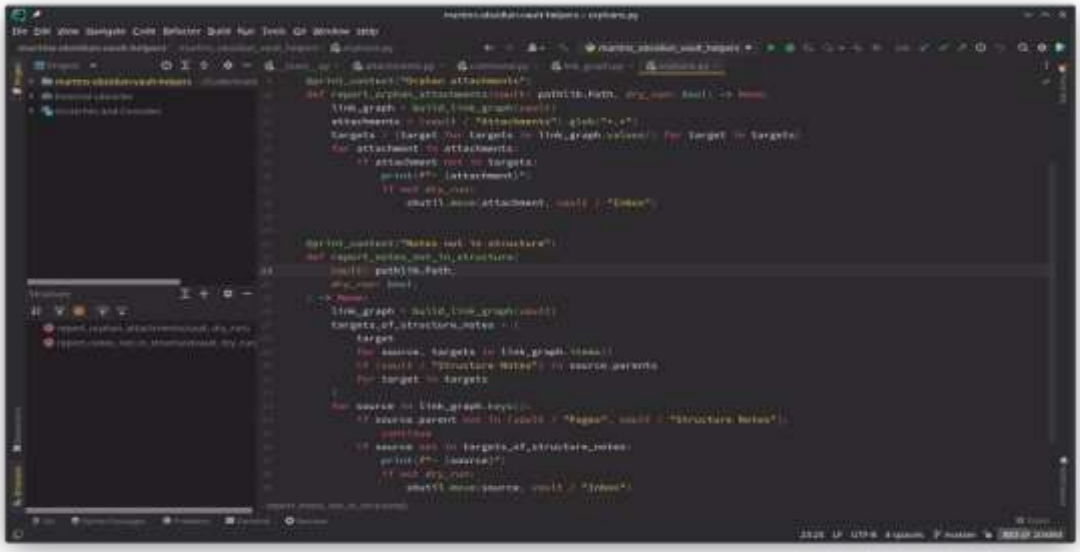

Рисунок 2.3 – Інтерфейс PyCharm

В цілому, PyCharm є потужним та універсальним інструментом для розробки програм на мові Python, який надає широкий спектр інструментів та можливостей для зручної та продуктивної роботи розробників.

Менш популярною альтернативою може стати NetBeans (рисунок 2.4) інтегроване середовище розробки (IDE), призначене для роботи з мовами програмування, такими як Java, PHP, JavaScript, HTML, CSS та інші. Розроблене компанією Apache Software Foundation, NetBeans надає розробникам широкий спектр інструментів для створення різноманітних програмних продуктів. Ось огляд функцій та можливостей NetBeans:

Підтримка мов програмування: NetBeans підтримує багато мов програмування, включаючи Java, PHP, JavaScript, HTML, CSS, Python та інші,

що дозволяє розробникам працювати з різноманітними технологіями в одному середовищі.

Редактор коду з підсвічуванням синтаксису: NetBeans має потужний редактор коду з підсвічуванням синтаксису та автоматичним завершенням коду, що сприяє підвищенню продуктивності розробки.

Відладка коду: Вбудований відладчик NetBeans дозволяє розробникам виявляти та виправляти помилки у коді, надаючи можливість крокувати по коду та переглядати значення змінних.

Управління проектами: NetBeans має зручні інструменти для управління проектами, включаючи можливість створення та налаштування проектів, організацію файлів та ресурсів у проекті.

Підтримка систем контролю версій: NetBeans інтегрується з популярними системами контролю версій, такими як Git та SVN, що дозволяє розробникам ефективно керувати версіями свого коду.

Розширюваність: NetBeans підтримує розширення, які дозволяють розробникам розширювати його функціонал та налаштовувати середовище розробки під свої потреби [17].

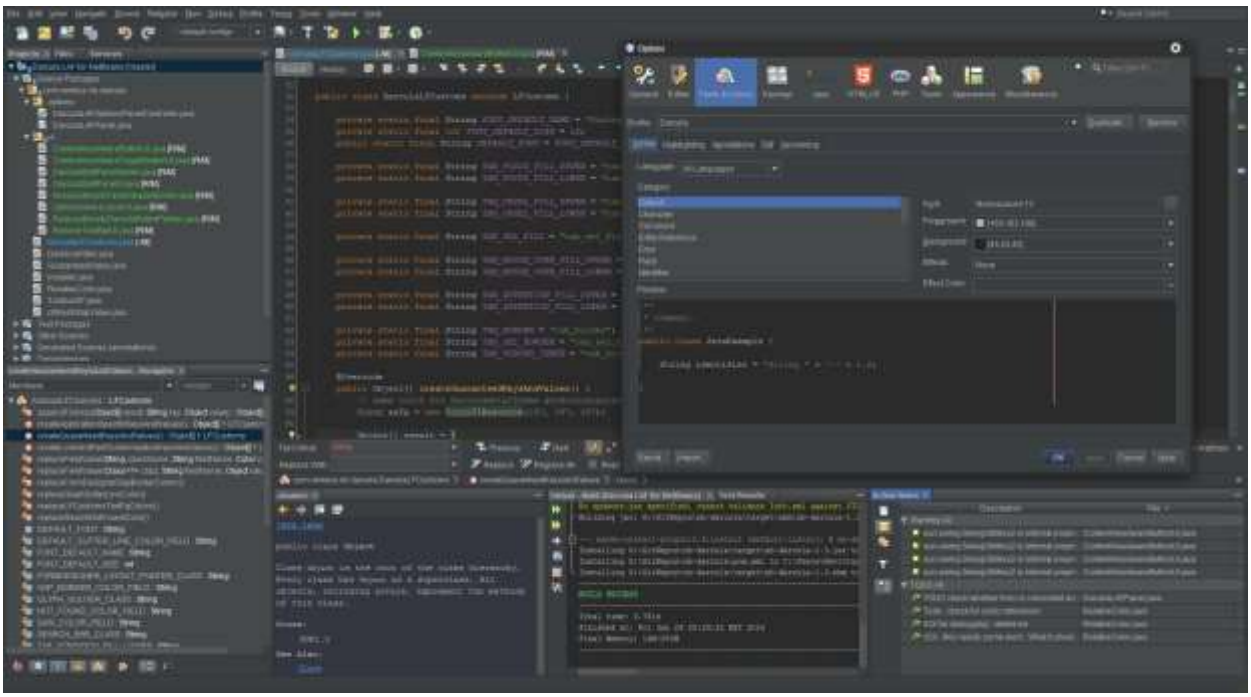

Рисунок 2.4 – Інтерфейс NetBeans

Уся архітектура проектів складається з концепції «дерева» із унаслідуванням класів бібліотеки Discord-бота. Процес програмування не потребує окремих інструментів, а саме написання коду забезпечується використанням скриптової мови програмування під назвою GDScript, синтаксис якого чим-то схожий з мовою Python. Середовище розробки поширюється під ліцензією MIT, що означає, що будь-який розробник може використовувати його повністю безкоштовно, модифікувати середовище та продавати свої програми чи ботів [18]. Він добре оптимізований, має ефективну систему керування ресурсами, яка допомагає створити швидкий та плавний програмний досвід.

Якщо порівняти інтерфейс IntelliJ та NetBeans – обидві середовища мають простий та інтуїтивний інтерфейс та мають ієрархічну структуру проекту. Але, у відмінності від IntelliJ IDEA, NetBeans має ієрархічну структуру із вкладеними вкладками, що дозволяє зручніше переключатися між об’єктами класів, редактором коду та ресурсами.

2.4 Висновки до другого розділу

Сучасні Discord-боти впроваджують нові механіки та інноваційні підходи до функціоналу, щоб зберегти інтерес користувачів, бути здатною конкурувати та приваблювати нових користувачів. Також розмаїття додає різноманіття середовищ і бібліотек, які були використанні у створенні цих ботів. З цього витікає висновок, що конкретного середовища для створення ботів не існує і кожен розробник обирає те, що йому потрібно під концепт ботів.

Інтерфейс таких ботів показує користувачу лише ті команди, які вже були використані користувачем. Також основною вимогою інтерфейсу є показ основних команд: кількість користувачів на сервері та ігрова статистика, створення гри для користувачів з назвою “Бункер”, особиста статистика, заявки на покупку ролей та інше.

Обраним середовищем і бібліотекою став для розробки став IntelliJ IDEA та бібліотека Java Discord API(JDA) – кросплатформна середовище, який використовує для скриптів мову Java із-за його можливостей та зручності. Але, як будь-яке середовище, він має свої мінуси, які були описані у розділі.

Альтернативами для проекту могли бути такі середовища, як Visual Studio, PyCharm, PhpStorm або NetBeans, але із-за непотрібності головних переваг цих середовищ (Visual Studio та PyCharm – структура та функціонал, можливість повної або часткової розробки логіки ботів; PhpStorm – інтеграція з Git documantation, розробка інтерфейсу з використанням JavaScript; NetBeans – повна безкоштовність як середовища, так і створених проектів) та недостатності досвіду розробки проектів у них, було вирішено зупинитися на середовище IntelliJ IDEA.

У цьому розділі було здійснено детальний аналіз існуючих рішень та засобів для розробки ботів. Зокрема, розглянуто різні платформи та інструменти, які дозволяють створювати ботів для різних потреб та платформ, таких як месенджери, соціальні мережі, веб-сайти тощо. Сучасні інструменти для розробки ботів значно спрощують процес створення та тестування ботів. Багато з них пропонують інтуїтивно зрозумілі графічні інтерфейси, що дозволяють налаштовувати логіку роботи бота без глибоких знань програмування.

Таким чином, аналіз показав, що на сьогодні існує широкий вибір інструментів для розробки ботів, що дозволяє обрати оптимальне рішення для конкретних завдань та потреб. Правильний вибір платформи та інструментів залежить від конкретних вимог проекту, досвіду команди та очікувань від бота.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ БОТА DISCORD

3.1 Постановка завдання, організація та основні ідеї бота Discord

Технологія розробки ботів для платформи Discord включає в себе використання різних мов програмування, такий як Java, та інші, а також бібліотек і фреймворків, призначених спеціально для роботи з Discord API. Для створення бота необхідно зареєструвати додаток на сайті Discord Developers, після чого отримати токен доступу, який буде використовуватися для автентифікації бота в системі Discord [19-20].

При розробці бота в системі Discord важливо враховувати функціональні вимоги та можливості API. Боти можуть виконувати різні завдання, такі як модерація сервера, автоматизація певних дій, надання інформації та розваги учасникам сервера. Також важливо дотримуватися правил використання платформи Discord і дотримуватися їх при розробці та використанні бота, щоб уникнути блокування або інших негативних наслідків з боку адміністрації Discord.

При розробці бота виникає ряд питань, пов’язаних з вибором парадигми програмування, вибором середовища розробки та існуючими тенденціями програми, створенням зручного інтерфейсу, вибором між самостійною або комерційною розробкою, тому, виходячи з актуальності, вважаємо, що постановкою нашого завдання є розробка штучного інтелекту в кросплатформенній системи Discord. Для розробки бота, для початку, потрібно:

- обрати та проаналізувати парадигму програмування, за допомогою якої буде створюватися проект;
- проаналізувати та обрати спосіб розробки бота – розробка у великій компанії;

- розглянути або, при відсутності, сформулювати технічне завдання до проекту;
- Налаштувати робочі середовища під потреби проекту та порівняти його з альтернативами;
- проаналізувати готові проекти по заданій програми та їх функціоналу;
- Розробка функціональності бота;
- розробити засоби для подальшого удосконалення проекту;
- Тестування та налагодження та розгортання на сервері.

Після створення проекту також важливим буде:

- проаналізувати оцінки користувачів або тестерів;
- розробити рекомендації щодо удосконалення проекту для подальшого розвитку;
- оцінити переваги та недоліки свого проекту.

Для початку потрібно розібратися у концепції бота для того, щоб був зрозумілий контекст та прийняті рішення для розробки. Концепцією бота є створення програмного забезпечення, яке інтегрується в платформу Discord і виконує певні завдання автоматично або за командою користувача. Такий бот використаний для різних цілей: управління сервером, розваги, надання корисної інформації, автоматизація рутинних процесів, інтеграція з іншими сервісами тощо.

Бот починається з того, що визначається його концепція та призначення. На початковому етапі розробки важливо чітко усвідомити, які задачі буде виконувати бот, які команди підтримуватиме, які дані оброблятиме та які сервіси інтегруватиме. Це допомагає сформувати загальне уявлення про структуру бота та необхідні компоненти для його реалізації:

- Ініціалізація та налаштування функціональності бота;
- Налаштування залежностей за допомогою Maven;

3.2 Використання платформи Steam та Faceit у Discord-бота

Steam — це цифрова платформа для розповсюдження відеоігор і програмного забезпечення, розроблена компанією Valve Corporation [21].

Faceit — це платформа для кіберспортивних змагань, яка надає гравцям можливість брати участь у турнірах, лігах та змагатися за рейтинги в різних відеоіграх [22].

Команда “/faceit steamId” – є частиною функціональності Discord-бота, який інтегрується з платформою Faceit для надання інформації про гравців на основі їх Steam ID. Ця команда дозволяє користувачам швидко отримати інформацію про свій або чужий профіль на Faceit, що включає статистику ігрової активності, рейтинг та інші релевантні дані.

Команда /faceit steamId реєструється під час ініціалізації бота. Це може бути здійснено через API Discord для створення інтерактивних команд. Зазвичай це робиться шляхом створення нового інтерфейсу команд за допомогою бібліотек, таких як JDA (Java Discord API) або Discord.js для Node.js. Важливо забезпечити, щоб команда була доступна та впізнавана ботом після його запуску.

Коли користувач вводить команду /faceit steamId <steamId>, бот отримує подію SlashCommandEvent. Далі бот обробляє цю подію, витягуючи переданий Steam ID. Важливо перевірити правильність введеного Steam ID та надати відповідний зворотний зв'язок користувачу, якщо ID невірний або відсутній.

Використовуючи отриманий Steam ID, бот відправляє запит до API Faceit для отримання інформації про гравця. Для цього може використовуватися HTTP-клієнт на Java, наприклад, HttpClient з пакету java.net.http. Запит до API повинен бути налаштований таким чином, щоб включати всі необхідні параметри для отримання повної інформації про профіль гравця.

Після отримання відповіді від API Faceit бот аналізує дані, які включають статистику гравця, рейтинг, кількість зіграних матчів та інші метрики [23]. Важливо врахувати можливі помилки у відповіді API, такі як відсутність даних або помилки сервера, і обробляти їх відповідним чином.

На основі отриманої інформації бот формує зрозумілу та інформативну відповідь для користувача. Використання об'єктно-орієнтованого програмування (ООП) дозволяє моделювати дані у вигляді класів і об'єктів, що спрощує обробку та відображення інформації. Наприклад, можна створити клас PlayerProfile, який міститиме всі необхідні дані про профіль гравця, а потім використовувати його для генерації відповіді.

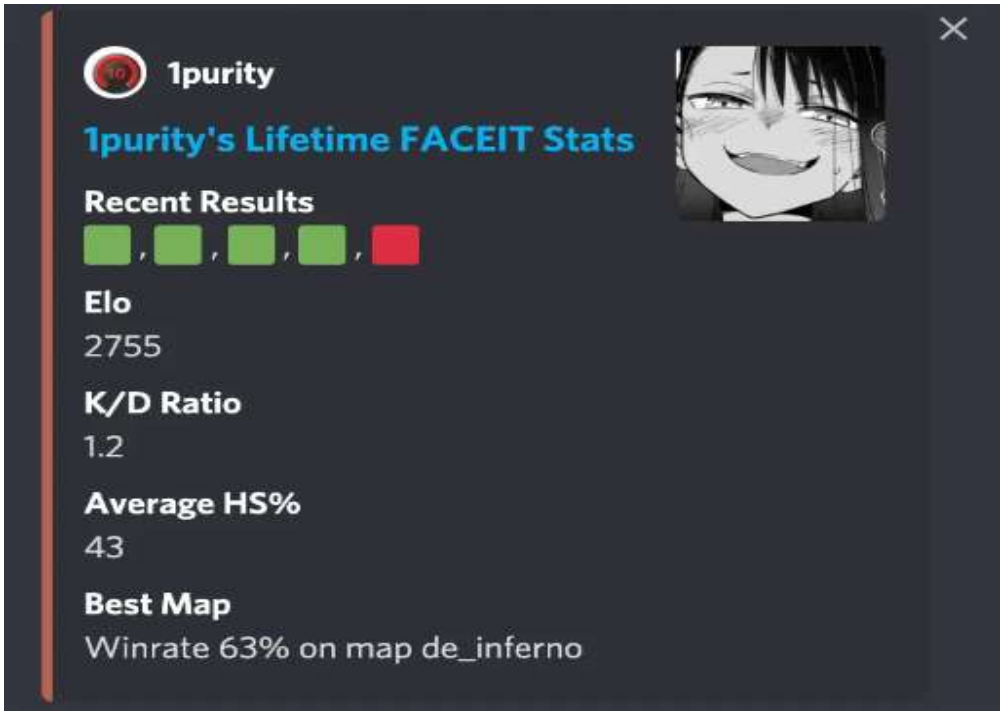

Рисунок 3.1 – Візуалізація функції /faceit

3.3 Інтерфейс та розробка програмної частини

Весь функціонал для кваліфікаційної роботи будуть проходити у тестовому каналі для розробників. При запускі бота, користувач може ввести командну функцію !help (рисунок 3.2):

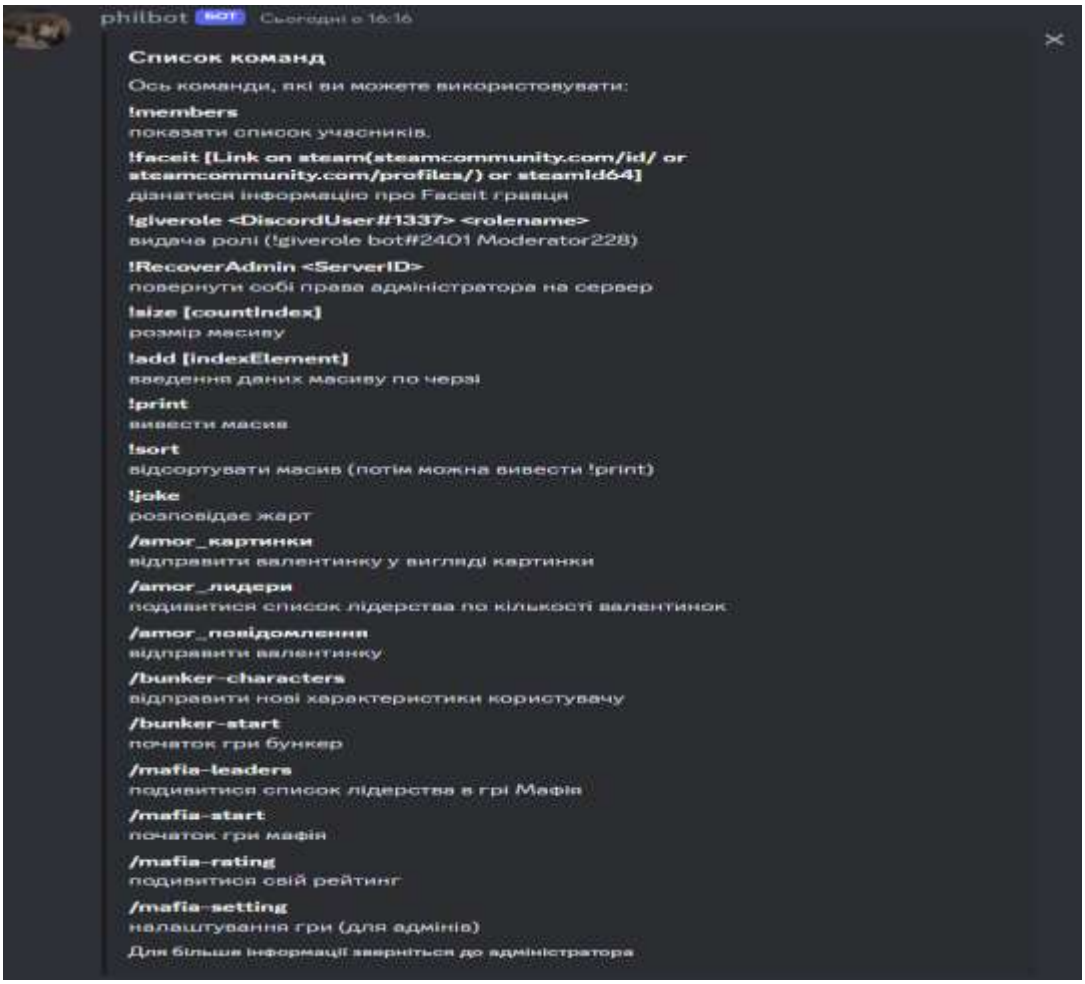

Рисунок 3.2 – Головні функції Бота

Команда "!members" ініціює виведення детальної інформації про учасників сервера. Вона є важливим інструментом для адміністраторів та користувачів, щоб мати змогу переглянути ідентифікатори та інші важливі дані про кожного учасника. Ця інформація може бути корисною для керування спільнотою, забезпечення безпеки та встановлення контакту з іншими користувачами. Вона дозволяє збирати статистику про активність учасників та керувати доступом до різноманітних функцій сервера. Таким чином, команда "!members" є важливим елементом для забезпечення ефективного функціонування та управління сервером у Discord.

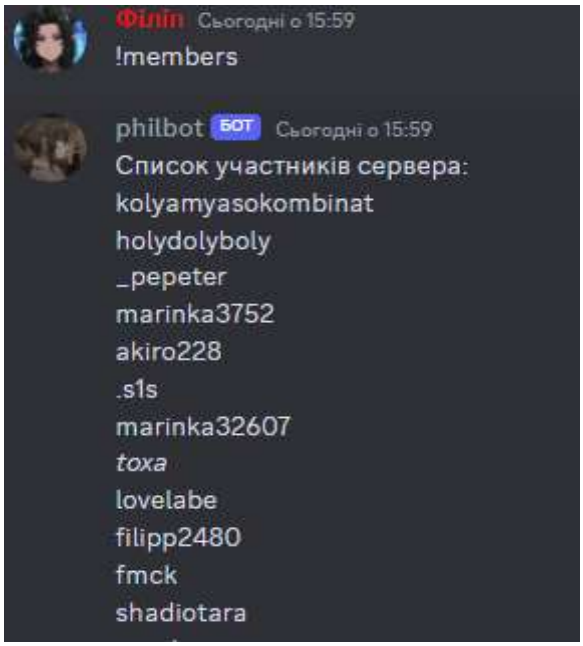

Рисунок 3.3 – Функція “!members” список користувачів

Команда “!giverole” призначена для видачі ролей користувачам на сервері Discord. Ця команда є інструментом управління правами доступу і дозволяє адміністраторам чи модераторам надавати специфічні ролі окремим користувачам. Використовуючи цю команду, адміністратор може легко і швидко змінювати ролі користувачів, надаючи їм доступ до різних функцій і каналів на сервері.

Наприклад, якщо адміністратор хоче надати користувачу з ім'ям "DiscordUser" та номером ідентифікації "#1337" роль з назвою "rolename", він може скористатися командою "!giverole DiscordUser#1337 rolename". Це дозволить встановити відповідні права доступу та функціональні можливості для цього користувача згідно з наданою роллю.

Використання команди "!giverole" забезпечує зручний та ефективний спосіб управління ролями користувачів на сервері Discord, що допомагає підтримувати порядок та контрольований доступ до різних функцій спільноти.

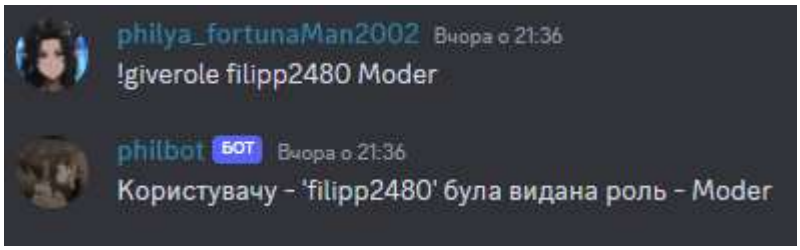

Рисунок 3.4 – Функція “!giverole” видача ролей

Команда "!RecoverAdmin" призначена для відновлення адміністративних прав на сервері Discord у випадку, якщо вони були випадково втрачені. Ця команда є надзвичайно корисною для адміністраторів, оскільки дозволяє швидко і без зайвих труднощів повернути собі повний контроль над сервером.

Використання команди виглядає наступним чином: адміністратор вводить команду разом із ідентифікатором сервера, наприклад, "!RecoverAdmin 123456789012345678". Після виконання команди користувач, який її ввів, знову отримує роль адміністратора на вказаному сервері.

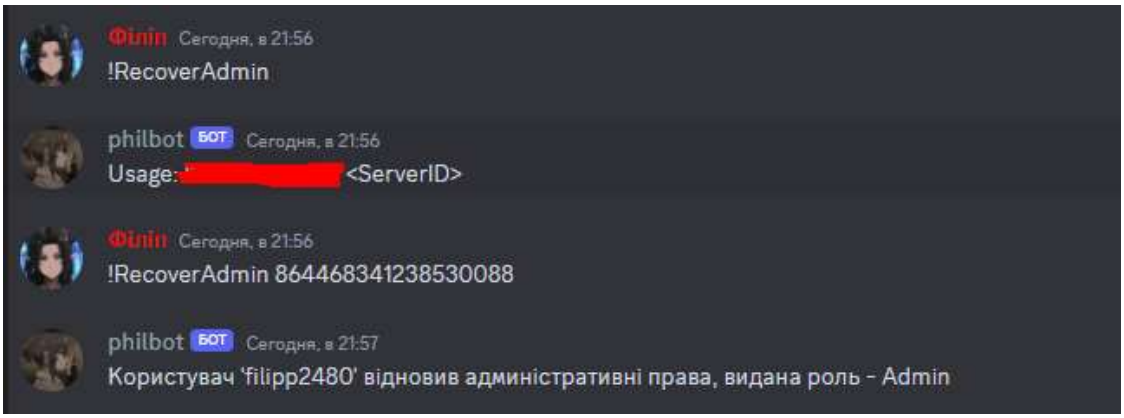

Рисунок 3.4 – Функція “!RecoverAdmin” відновлення прав

Команда "/amor_картинки" призначена для відправлення валентинки у вигляді картинки іншим користувачам на сервері Discord. Ця команда дозволяє користувачам надсилати зображення валентинки, що може бути зроблено як анонімно, так і з зазначенням відправника.

Основні функції та особливості команди "/amor_картинки":

Надсилання валентинки: Користувач може вибрати красиве зображення валентинки і відправити його іншому користувачу на сервері. Це створює атмосферу дружелюбності та підтримує позитивну комунікацію між учасниками спільноти.

Анонімність: Користувач може вибрати опцію відправити валентинку анонімно, тобто без зазначення свого імені. Це може бути використано для сюрпризу або просто для того, щоб зберегти інтригу.

Вказання відправника: Якщо користувач бажає, він може відправити валентинку з зазначенням свого імені, щоб одержувач знав, від кого прийшло привітання.

Підтримка візуального контенту: Команда дозволяє відправляти зображення, що робить валентинки яскравими та привабливими, створюючи більш емоційний та значущий жест уваги.

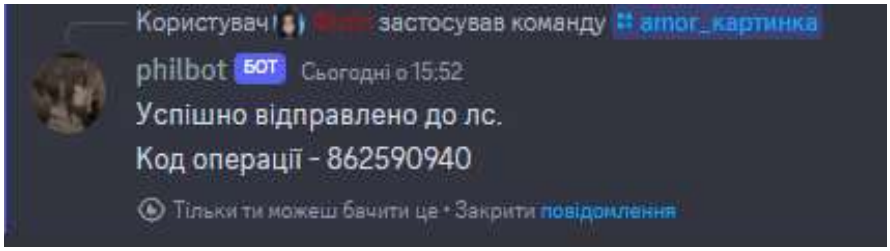

Рисунок 3.5 – Команда “/amor_картинка” відправка валентинки

Таким чином, команда /amor_картинки є чудовим інструментом для вираження почуттів та підтримки дружньої атмосфери на сервері Discord. Вона дозволяє користувачам обмінюватися валентинками у вигляді яскравих зображень, додаючи нотки тепла і радості до спілкування. В Discord-боті, написаному на Java, можна використовувати JDA (Java Discord API). Це дозволяє створити команду, яка реагуватиме на запити користувачів і надсилатиме відповідні зображення. Бот може бути налаштований так, щоб зберігати зображення у базі даних або інтегруватися з зовнішніми сервісами для отримання нових картинок.

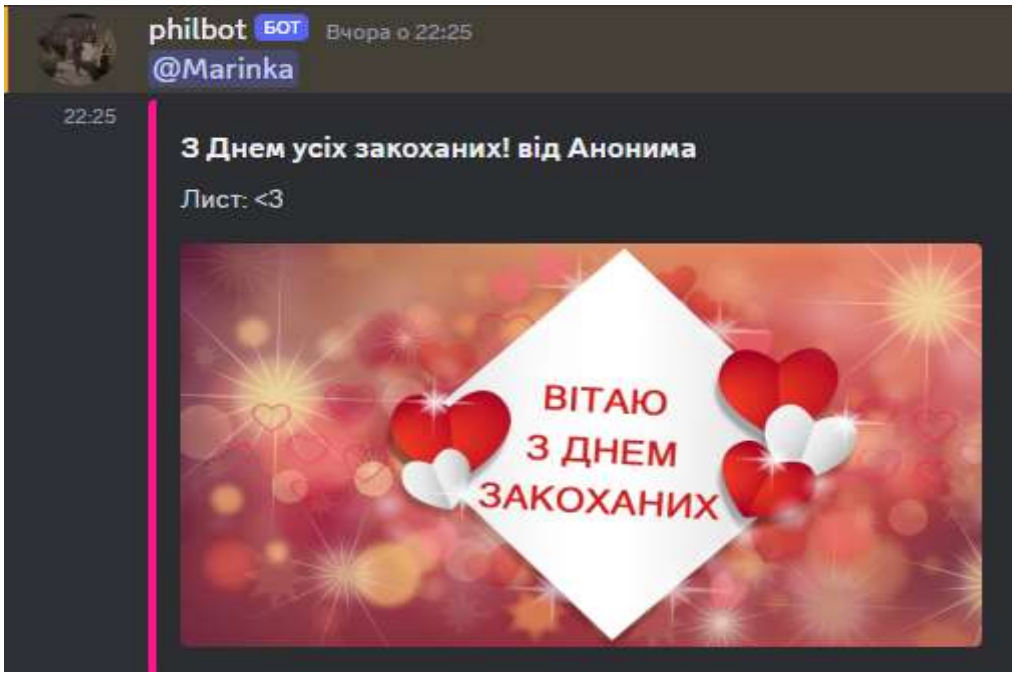

Рисунок 3.6 – Відправлена валентинка через команду

Команда "/amor_лідери" призначена для відображення списку користувачів, які отримали найбільшу кількість валентинок на сервері Discord. Ця команда є чудовим способом підтримувати дружню конкуренцію та мотивацію серед учасників спільноти, оскільки вона показує, хто є найпопулярнішим або найбільш активним у надсиланні та отриманні валентинок.

Відображення рейтингу: Команда виводить список користувачів, упорядкований за кількістю отриманих валентинок. Це дозволяє всім учасникам бачити, хто знаходиться на вершині рейтингу та скільки валентинок вони отримали.

Мотивація та взаємодія: Рейтингова система стимулює користувачів активно брати участь у надсиланні валентинок, що сприяє підвищенню рівня взаємодії та соціальної активності на сервері.

Підтримка духу спільноти: Команда допомагає зміцнити відчуття спільності та дружби серед учасників, заохочуючи їх виявляти увагу та турботу один до одного.

Рисунок 3.6 – Команда “/amor_лідери” топ лідерів

Команда "/bunker-characters" призначена для генерування та збереження характеристик користувачів у форматах JSON і TXT. Ця команда використовується в грі "Бункер" на сервері Discord і допомагає створювати та зберігати унікальні характеристики для кожного гравця. Після виконання команди користувач отримує текстовий файл з характеристиками, а JSON файл використовується для збереження даних на сервері.

Основні функції та особливості команди "/bunker-characters":

- Генерація характеристик,
- Створення файлів,
- Відправка файлів;

Приклад використання команди:

Користувач вводить команду "/bunker-characters", після чого бот генерує характеристики та створює відповідні файли.

Бот відправляє текстовий файл з характеристиками користувачу через особисті повідомлення.

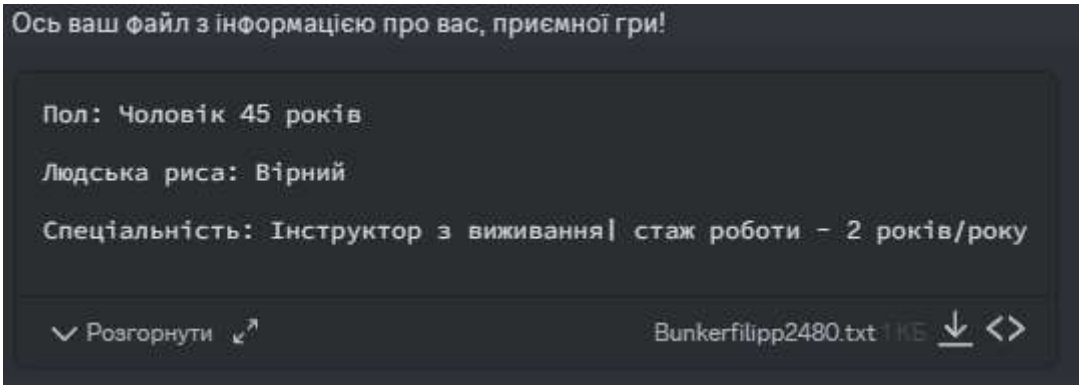

Рисунок 3.7 – Відправлений файл через команду “/bunker-characters”

JSON файл виглядає так:

- FactOne: Історія або факт про персонажа, який додає глибину та індивідуальність.
- FactTwo: Ще один факт або інформація про персонажа.
- Health: Стан здоров'я персонажа, у цьому випадку - туберкульоз.
- Sex: Стать персонажа та його вік.
- StageJob: Досвід роботи на певній посаді у роках або роках/рік.
- SpecialCharacter1 та SpecialCharacter2: Особливі властивості або можливості персонажа, які відзначаються в ігровому контексті.
- Phobia: Фобія або страх персонажа.
- HumanTrait: Характерна особливість або риса характеру.
- Hobby: Захоплення або хобі персонажа.
- Job: Постійна робота або заняття персонажа.
- DegreeHealth: Ступінь здоров'я персонажа у відсотках.
- Inventory: Інвентар персонажа або речі, які він має при собі.

Цей файл містить детальну інформацію про персонажа, яка може бути використана для розробки ігор, написання історій або створення інтерактивного контенту.

JSON (JavaScript Object Notation) є важливим компонентом у цьому проекті, оскільки він забезпечує зручний та ефективний спосіб зберігання і передачі даних. JSON використовується для збереження характеристик користувачів, згенерованих командою "/bunker-characters".

Структурованість та читабельність: JSON дозволяє зберігати дані в чіткій і зрозумілій структурі, яка легко читається як людиною, так і машиною. Це спрощує процес налагодження та підтримки коду. Це забезпечує легкість інтеграції та обробки даних у різних середовищах.

Легкість передачі даних: JSON файли легко передаються між клієнтом та сервером через мережу, що робить його ідеальним для зберігання та обміну даними в реальному часі. У цьому проекті це означає, що згенеровані характеристики можуть бути швидко доступні та оброблені. Гнучкість: JSON дозволяє зберігати різноманітні типи даних, такі як рядки, числа, масиви та об'єкти.

Команда "/bunker-start" призначена для початку гри "Бункер" на сервері Discord. Вона створює інтерактивний інтерфейс з використанням вбудованих повідомлень (embed) та кнопок, що дозволяють користувачам приєднатися до гри та керувати її процесом. Ця команда полегшує організацію та проведення гри, забезпечуючи зручний і зрозумілий інтерфейс для всіх учасників.

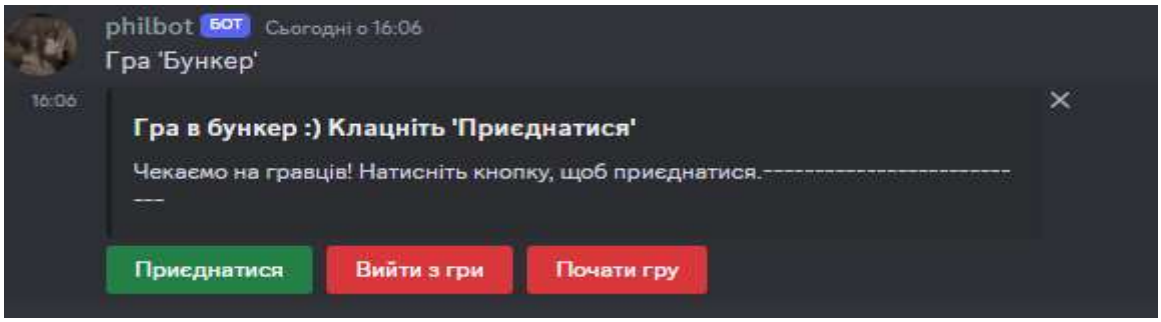

Рисунок 3.8 – Створення лобі через команду

Кнопка "Приєднатися": Користувачі можуть натискати на кнопку "Приєднатися" для того, щоб стати учасниками гри. Бот зберігає список учасників, які приєдналися до гри.

Кнопка "Почати гру": Коли достатня кількість гравців приєдналася, ведучий (адміністратор або модератор) може натиснути кнопку "Старт гри". Це ініціює початок гри та виводить велике вбудоване повідомлення з деталями про процес гри.

Процес гри: Після натискання кнопки "Почати гру" бот створює вбудоване повідомлення, яке містить таблицю з характеристиками кожного гравця та повідомлення з катаклізмом, їх ролями та іншими важливими елементами гри. Це повідомлення постійно оновлюється у міру просування гри.

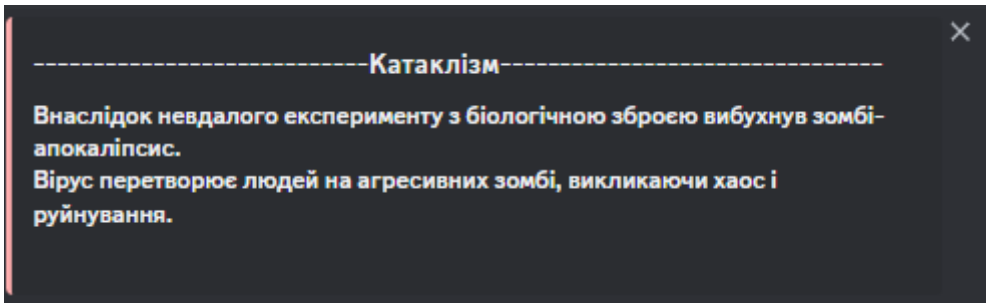

Рисунок 3.9 – Вбудоване повідомлення про катаклізм

Ця реалізація показує, як використовувати команду "/bunker-start" з використанням JDA. Він створює вбудоване повідомлення з кнопками для приєднання та старту гри, обробляє натискання кнопок та оновлює вбудоване повідомлення з процесом гри. Це забезпечує зручний та інтерактивний спосіб керування грою "Бункер" на сервері Discord.

Після вбудованого повідомлення, виводиться таблиця з характеристиками кожного гравця. Характеристики зберігаються в мапі playerCharacteristics, яка асоціює ідентифікатор кожного гравця зі списком його характеристик. Таблиця з характеристиками додається у вигляді полів до вбудованого повідомлення, що забезпечує зручний та зрозумілий інтерфейс для гравців.

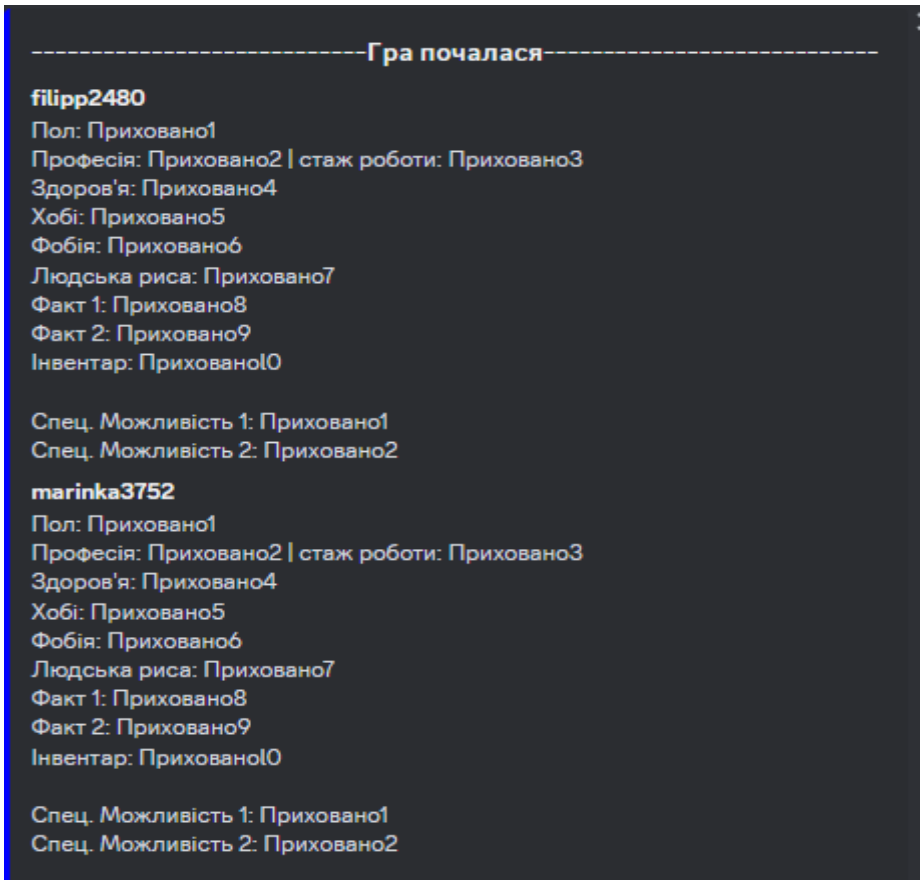

Рисунок 3.10 – Основний інтерфейс гри “Бункер”

Команда "/mafia-leaders" призначена для відображення списку лідерів у грі "Мафія" на сервері Discord. Вона показує рейтинг гравців, упорядкований за їх досягненнями в грі. Це дозволяє учасникам бачити, хто є найкращим гравцем та стимулює здорову конкуренцію серед учасників спільноти.

Основні функції та особливості команди "/mafia-leaders":

Відображення рейтингу: Команда виводить список гравців, упорядкований за їх рейтингом у грі "Мафія". Це дозволяє всім учасникам бачити, хто досяг найвищих результатів.

Мотивація для гравців: Показуючи рейтинг, команда стимулює гравців грати більше та покращувати свої навички, щоб піднятися вище у списку лідерів.

Прозорість і чесність: Відкритий рейтинг створює прозору систему оцінки, де кожен гравець може бачити свій прогрес та прогрес інших гравців.

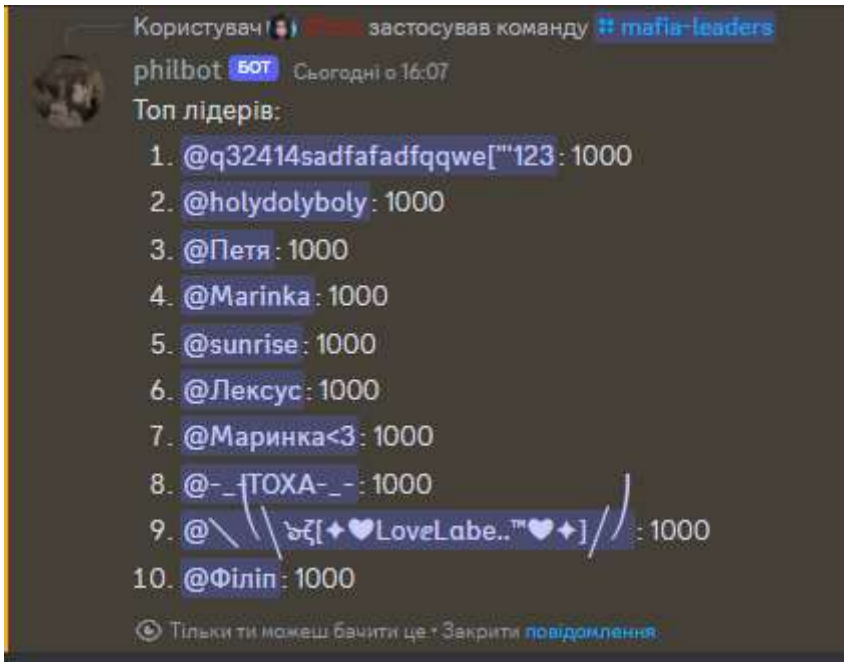

Рисунок 3.11 – Топ лідерів по рейтингу в грі “Мафія”

Команда "/mafia-start" призначена для запуску лобі гри "Мафія" на сервері Discord. Вона створює інтерактивний інтерфейс з кнопкою для приєднання, а після завершення приєднання гравців створює необхідні канали для гри та призначає кожному гравцю відповідні ролі (Мафія, Мирний житель, Шериф тощо). Створення лобі гри, команда яка буде виводить вбудоване повідомлення з кнопкою "Приєднатися", щоб гравці могли приєднатися до гри. Створення каналів, після завершення приєднання гравців, бот створює текстові та голосові канали, необхідні для гри "Мафія". Призначення ролей , кожному гравцю, який приєднався до гри, бот випадковим чином призначає роль (Мафія, Мирний житель, Шериф тощо).

Ігрова сесія: Коли користувач вводить команду "/mafia-start", бот відповідає вбудованим повідомленням з кнопкою "Приєднатися". Гравці можуть натиснути цю кнопку, щоб приєднатися до гри.

Старт гри: Коли достатня кількість гравців приєдналася, ведучий натискає кнопку "Старт гри". Це спричиняє створення категорії та каналів для гри "Мафія" на сервері.

Призначення ролей: Бот випадковим чином призначає ролі кожному гравцю та виводить ці ролі у вбудованому повідомленні.

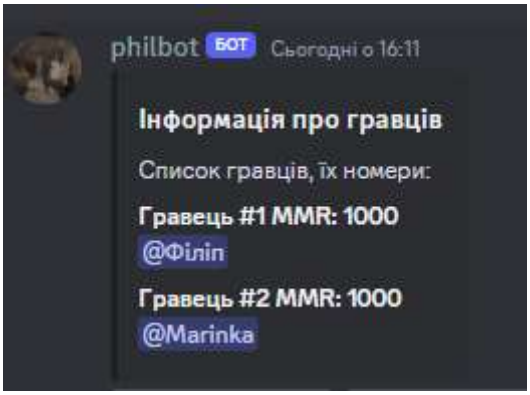

Рисунок 3.12 – Інтерфейс початку гри “Мафія”

Кнопка "Faceit Checker" - це функціонал, що додається до Discord-бота, який дозволяє користувачам перевіряти статистику гравців на платформі Faceit безпосередньо через Discord.

Основна ідея полягає в тому, щоб користувач міг ввести посилання на обліковий запис Steam гравця, або інше ідентифікуюче значення (наприклад, ім'я гравця), і бот автоматично витягує статистику цього гравця з платформи Faceit за допомогою офіційного API. Після цього бот відображає отриману інформацію у вигляді повідомлення в каналі Discord, де користувач може бачити детальну статистику гравця, таку як рейтинг, кількість перемог, статистика гри в різних режимах і т.д.

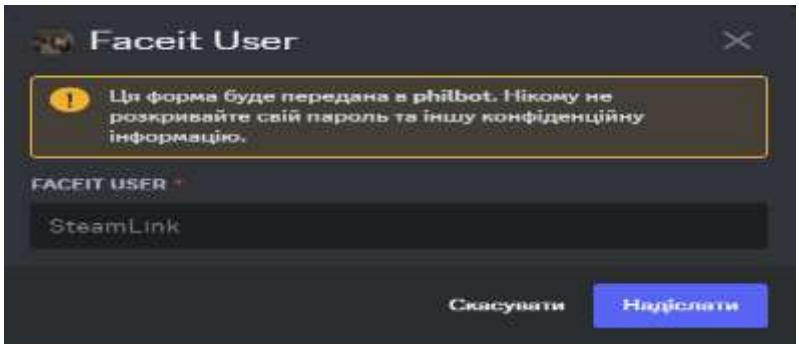

Рисунок 3.13 – Інтерфейс “Faceit Checker”

Отже, кнопка "Faceit Checker" на Discord відкриває можливість швидко і зручно отримувати інформацію про гравців на платформі Faceit без необхідності переходити на інші сайти або застосунки.

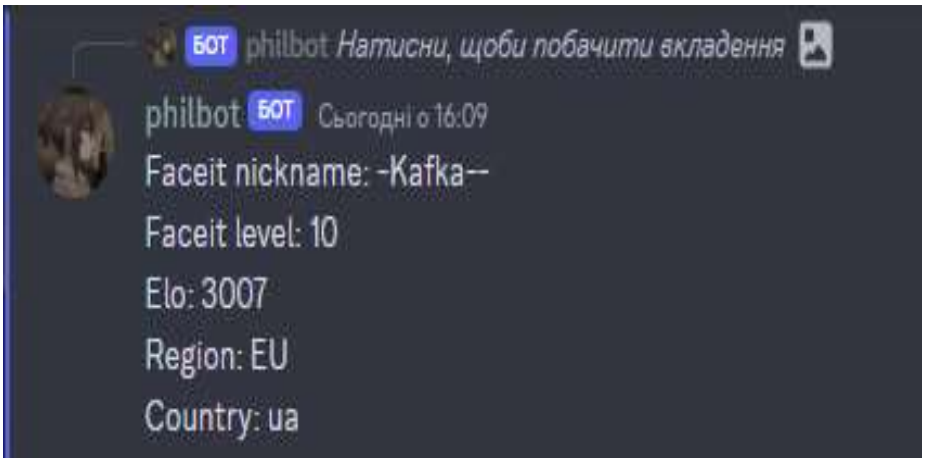

Рисунок 3.14 – Інформація після виконання операції “Faceit Checker”

Реалізація штучного інтелекту в Discord-боті включають декілька етапів, які дозволяють інтегрувати модель штучного інтелекту для обробки запитів та генерації відповідей у режимі реального часу.

Для інтеграції штучного інтелекту використовуються моделі GPT від OpenAI, зокрема GPT-3.5 або GPT-4. Для цього необхідно налаштувати API OpenAI на рисунку 3.15, отримати API-ключ та налаштувати функціонал для відправки запитів до моделі GPT і отримання відповідей.

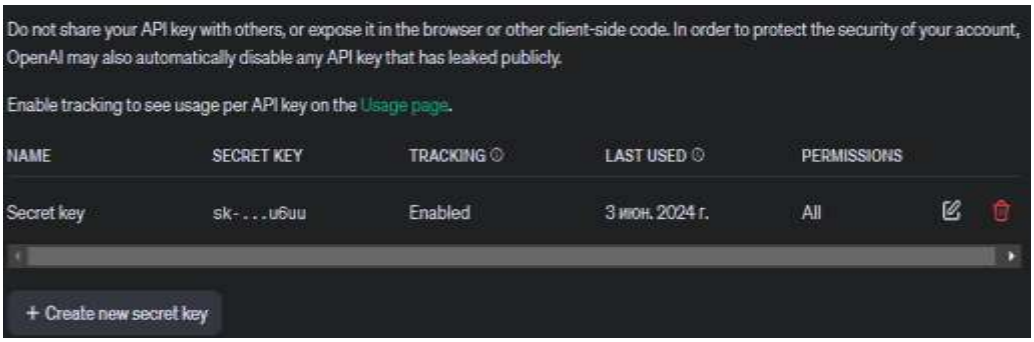

Рисунок 3.15 – Створення API ключа

Основним завданням бота є обробка повідомлень користувачів. Коли користувач надсилає повідомлення, бот отримує це повідомлення, формує відповідний запит до моделі GPT і відправляє його до API OpenAI. Відповідь моделі отримується у форматі JSON, з якого витягується потрібний текст для відправки назад користувачеві. Щоб забезпечити коректність та релевантність відповідей, важливо формувати точні запити до Discord-бота на рисунку 3.16.

Це може включати очищення тексту від зайвих символів або маркерів, а також налаштування параметрів запитів до моделі

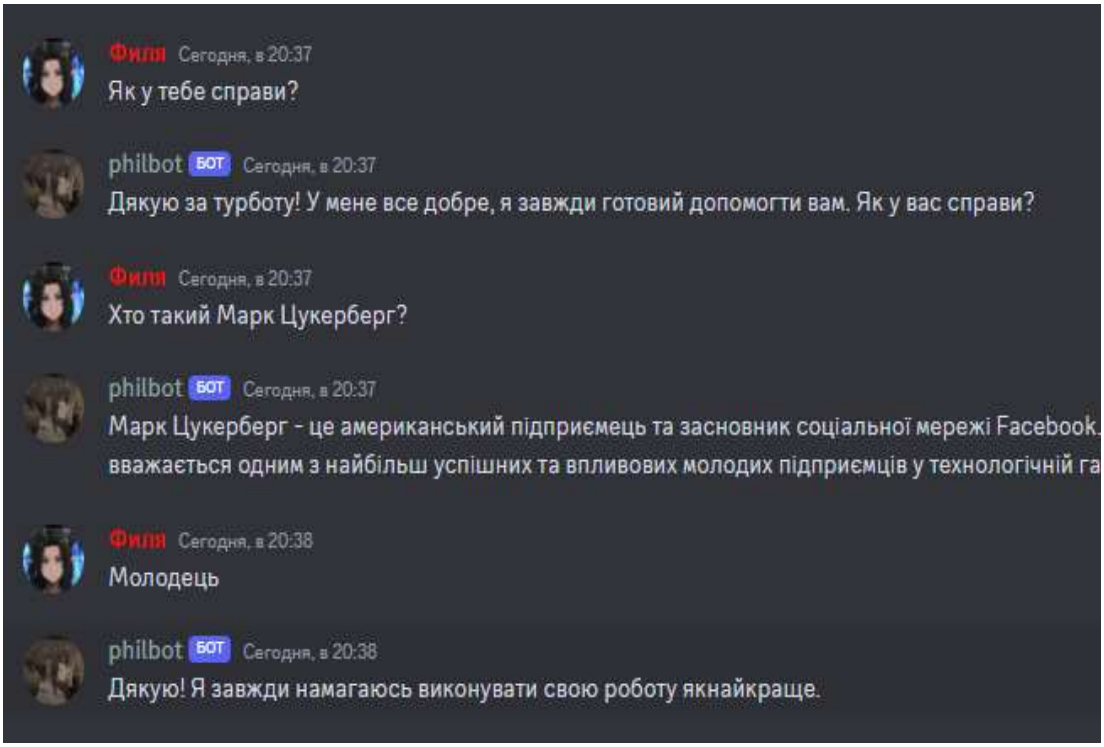

Рисунок 3.16 – Запити до Discord-бота

3.4 Висновки до третього розділу

У бота була розроблена унікальна стилістика і структура, концепція та зручний інтерфейс. Також було використано сучасний підхід до реалізації різноманітних функцій та команд.

Бот має унікальні функції для свого штучного інтелекту. Кожен користувач може використовувати різні команди або функції у разі необхідності для зручності та підтримки користувачів. Успішна робота команд та кнопок на Discord-сервері розширює можливості спільноти та підвищує її ефективність.

Команди дозволяють автоматизувати різні процеси, такі як керування користувачами, надання ролей та управління даними. Кнопки додають інтерактивність та швидкість реакції на події, спрощуючи взаємодію для учасників. За допомогою цих інструментів адміністратори можуть

забезпечити зручне та організоване середовище для спілкування та спільної гри для учасників. Все це сприяє покращенню спільноти, підвищенню її активності та залученню нових учасників.

Реалізація штучного інтелекту в Discord-боті включає поєднання кількох технологій та кроків: створення та налаштування самого бота, інтеграція з API моделі штучного інтелекту, обробка повідомлень користувачів і форматування відповідей. Це дозволяє забезпечити взаємодію з користувачами на новому рівні, використовуючи можливості сучасних моделей штучного інтелекту для генерації контекстно-залежних і змістовних відповідей.

ВИСНОВКИ

У кваліфікаційній роботі було досліджено можливості бібліотеки Java Discord API для створення ботів, а також способи створення ботів, їх історію розвитку, проаналізовано готові рішення різних компаній, було обране програмне середовище для створення та реалізації Discord-боту.

Бот має унікальні функції та, проаналізувавши ботів інших розробників, можна стверджувати, що він не має прямого аналогу. Розроблена у кваліфікаційній роботі система модерації дозволяє легко створювати нові канали та керувати ними. Унікальним є використання штучного інтелекту моделі GPT та створення модуля для розваг з психологічними іграми та функціоналу для перегляду ігрової статистики. Аспектами користування перегляду ігрової статистики є: автентифікація та авторизація користувача та отримання обробки даних через інтеграцію Discord.

На основі проведеного дослідження і аналізу можна надати такі рекомендації щодо подальшого розвитку та вдосконалення бота.

Розширення інтеграцій також буде корисним. Інтеграція з іншими ігровими платформами або сервісами, такими як Steam, дозволить отримувати більше даних. Використання Webhooks для отримання оновлень в реальному часі про події на Faceit, таких як завершення матчів, покращить актуальність інформації.

Оптимізація продуктивності може включати впровадження механізмів кешування для зменшення навантаження на Faceit API та підвищення швидкості відповіді. Використання асинхронних запитів покращить продуктивність та зменшить час очікування користувачів.

Покращення користувацького досвіду можливе через використання інтерактивних відповідей з кнопками та реакціями для зручної навігації та взаємодії. Додавання можливості налаштування профілів дозволить користувачам зберігати команди або налаштування.

Безпека та конфіденційність також є важливими аспектами. Потрібно застосувати додаткові заходи безпеки для захисту API ключа, такі як шифрування та обмеження доступу. Забезпечення відповідності політикам конфіденційності при обробці даних користувачів також необхідне.

Документація та підтримка користувачів допоможуть користувачам та розробникам краще розуміти функціонал бота. Створення детальної документації, яка включатиме приклади використання команд та опис функціоналу, а також налаштування системи підтримки користувачів через окремий канал у Discord або інший сервіс.

Моніторинг та аналіз використання бота дозволять виявляти та усувати проблеми, а також оптимізувати функціональність. Впровадження інструментів для відстеження використання команд та активності користувачів допоможе в подальшому розвитку.

Недоліком програмної реалізації, є невипадкова реалізація ролей у грі “Мафія”. Це створено у першу чергу із-за складності синхронізації випадкових величин у всіх користувачів та недостачі досвіду у створенні штучного інтелекту.

Перевагами бота можна назвати наступне:

- зручний доступ до статистики;
- широкий функціонал;
- унікальна стилістика і структура, у якого не було знайдено аналогів;
- реалізовані унікальні функції для зручної роботи;
- велика кількість команд, що допомагає швидше оптимізувати сервер;
- управління за допомогою різноманітних команд.

На основі відгуків користувачів, які тестували цього бота, можна прийти до факту, що бот був позитивно сприйнятий та отримала гарні оцінки від цільової аудиторії. Це підтверджує успішність розробки, вдале задоволення потреб користувачів. З аналізу відгуків, при подальшому доповненні

контентом та виправленні недоліків, можна зробити висновок, що бот матиме комерційний успіх при добрій маркетинговій компанії.

Бот відповідає таким вимогам, як можливість перегляду статистики (в даному випадку реалізація через вбудованого інтерфейсу Discord), створення соціального та розважального аспекту, реалізація аспектів ботів.

Усі завдання кваліфікаційної роботи, тобто вибір та обґрунтування парадигми програмування для подальшої роботи; дослідження індустрії штучного інтелекту; вибір програмного забезпечення для розробки та обґрунтування обраних засобів; аналіз існуючих конкурентів; формування технічного завдання; розробка програмної реалізації та її опис; висновки були успішно виконанні.

На основі проведеного дослідження та створеної програмної реалізації можна зробити висновок, що створений бот має потенціал стати успішним бізнес-проектом. Однак для цього потрібно докласти багато зусиль та проявити терпіння.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інтеграція цифрових технологій в освітній процес
URL: https://ela.kpi.ua/server/api/core/bitstreams/a6118daf-1629-43e7-80c9-7432267a51b6/content (дата звернення: 14.03.2024)
2. Історія розвитку штучного інтелекту
URL: https://cases.media/article/evolyuciya-shtuchnogo-intelektu-shi-viznachni-momenti-v-istoriyi-ta-zastosuvannya (дата звернення 25.03.2024)
3. Сучасний стан розвитку штучного інтелекту
URL: http://obrii.org.ua/usec/storage/article/Sharov_2023_136.pdf (дата звернення: 27.03.2024)
4. Компанія криптовалюти WhiteBit
URL: https://whitebit.com/ua/about-us (дата звернення 03.04.2024)
5. Аналіз Flow XO
URL: https://flowxo.com/ (дата звернення 07.04.2024)
6. Моніторинг та управління ресурсами штучного інтелекту
URL: https://naurok.com.ua/test/shtuchniy-intelekt-internet-rechey-smart-tehnologi-577934.html (дата звернення 08.04.2024)
7. Dyno bot ШІ
URL: https://dyno.gg/bot (дата звернення 08.04.2024)
8. MEE6 bot ШІ
URL: https://mee6.xyz/en/tutorials (дата звернення 09.04.2024)
9. Mafia bot ШІ
URL: https://discord.com/application-directory/758999095070687284 (дата звернення 10.04.2024)
10.Музичний бот Groovy
URL: https://groovy.bot/ (дата звернення 11.04.2024)
11. Функціонал психологічної гри Бункер
URL: https://bunker-online.com/ (дата звернення 13.04.2024)

12. Середовище розробки IntelliJ IDEA

URL: https://www.jetbrains.com/idea/ (дата звернення 14.04.2024)

13. Бібліотека Java Discord API

URL: https://jda.wiki/introduction/jda/ (дата звернення 14.04.2024)

14. Складність використання бібліотеки ботів

URL: https://avada-media.ua/ua/services/razrabotka-discord-botov/ (дата звернення 20.04.2024)

15. Розширення плагінів Visual Studio

URL: https://learn.microsoft.com/uk-ua/power-pages/configure/vs-code-extension (дата звернення 21.04.2024)

16. Середовище розробки PyCharm

URL: https://www.jetbrains.com/pycharm/ (дата звернення 23.04.2024)

17. Інтегрована середа розробки NetBeans

URL: https://netbeans.apache.org/ (дата звернення 09.05.2024)

18. Імперативна мова GDScript

URL: https://docs.godotengine.org/ua/stable/tutorials (дата звернення 20.05.2024)

19. Портал розробників Discord

URL: https://discord.com/developers/docs/intro (дата звернення 21.05.2024)

20. Кросплатформенна система Discord

URL: https://discord.com/ (дата звернення 22.05.2024)

21. Ігрова платформа «Steam»

URL: https://store.steampowered.com/?l=english (дата звернення 22.05.2024)

22. Кіберспортивна платформа «Faceit»

URL: https://www.faceit.com (дата звернення 23.05.2024)

23. Faceit Developer Portal

URL: https://docs.faceit.com/ (дата звернення 24.05.2024)

ДОДАТОК

ЛІСТИНГ ПРОГРАМИ

```
import discordbot.BunkerCmds.BunkerButtons.BunkerStartBtn;
import discordbot.BunkerCmds.BunkerCharactersCmd;
import discordbot.BunkerCmds.BunkerSpecActivity.BunkerRealizeSpecCharacters;
import discordbot.BunkerCmds.BunkerStartCmd;
import discordbot.MafiaCmds.MafiaRatingCmd;
import discordbot.MafiaCmds.MafiaStartCmd;
import discordbot.MafiaCmds.MafiaStartRatingCmd;
import discordbot.MafiaLeaderCmds.MafiaLeadersCmd;
import discordbot.MafiaRatingSys.RatingManager;
import discordbot.ValentineAdminCmds.ValentineSetCountArrowCmd;
import discordbot.ValentineAdminCmds.ValentineSetCountCmd;
import discordbot.ValentineCmds.*;
import discordbot.ValentineLeaderCmds.ValentineLeadersCmd;
import discordbot.PhilbotSys.Logger;
import discordbot.ValentineSys.ValentineManager;
import net.dv8tion.jda.api.JDA;
import net.dv8tion.jda.api.JDABuilder;
import net.dv8tion.jda.api.entities.*;


import net.dv8tion.jda.api.hooks.ListenerAdapter;

import net.dv8tion.jda.api.interactions.commands.OptionType;
import net.dv8tion.jda.api.interactions.commands.build.Commands;
import net.dv8tion.jda.api.requests.GatewayIntent;


// MTA4MDE4ODM3MTA1OTYxNzkwMg.GfJvm6.OOI2EsgnkEclEYXdLSzMBrS05h9gbUD4N1yFCc
import javax.security.auth.login.LoginException;
import java.util.Arrays;


public class PhilBot extends ListenerAdapter {
    static Logger logger = Logger.getInstance();
    // хата ивана - 598887733787885570
    // Evil Guild - 864468341238530088

    private static final String SERVER_ID = "864468341238530088";
    public static final String CHANNEL_ID = "1080259055911710770";
    private static final String TOKEN_BOT_ID =
"MTA4MDE4ODM3MTA1OTYxNzkwMg.GfJvm6.OOpa462EsgnkEclEYXdLSzMBrS05h9gbUD4N1yFCc";

    public static JDA jda;

    // TOKEN_BOT_ID, GatewayIntent.ALL_INTENTS
```

```
public static void main(String[] args) throws Exception, InterruptedException, LoginException
{
    ValentineManager valentineUserManager = new ValentineManager();
    RatingManager ratingManager = new RatingManager();

    jda = JDABuilder.create(TOKEN_BOT_ID, Arrays.asList(GatewayIntent.values()))
            .setActivity(Activity.playing("Филю"))
            .addEventListeners(
                    // мафия
                    new MafiaStartCmd(),
                    new MafiaStartRatingCmd(ratingManager),
                    new MafiaRatingCmd(ratingManager),
                    new MafiaLeadersCmd(ratingManager),
                    // Бункер
                    new BunkerCharactersCmd(),
                    new BunkerStartCmd(),
                    new BunkerStartBtn(),
                    new BunkerRealizeSpecCharacters(),

                    // Валентинки
                    new ValentineSendMessageCmd(valentineUserManager),
                    new ValentineSendImageCmd(valentineUserManager),
                    new ValentineLeadersCmd(valentineUserManager),
                    new ValentineSetCountCmd(valentineUserManager),
                    new ValentineAmorArrowCmd(valentineUserManager),
                    new ValentineCheckMessageCmd(valentineUserManager),
                    new ValentineCheckArrowCmd(valentineUserManager),
                    new ValentineSetCountArrowCmd(valentineUserManager),

                    // Авторские команды бота
                    //new KickerOptimize(),
                    new MemberListCommand(),
                    new OnReadyClass(),
                    new ServerAdminHacked(),
                    new HelpCommand(),
                    new FaceitInfoListener(),
                   new GiveRoleServer(),
                    new TriggerNewUser(),
                    new OnJokeMessage(),
                    new ModalRegister(),
                    new ModalListener(),
                    new RoleBuyer(),
                  //  new SortListener(),
                    new Test(),
                 new Kicker()

            )
```

```
            .build().awaitReady(); logger.logEvent("[ConsolePhilBot] Все EventListener работают!");
        // 1131547249470607390 - канал тестинга
        // 1080259055911710770 - мой канал
        logger.setLogChannel(jda.getTextChannelById(1080259055911710770L));

        Guild guild = jda.getGuildById(SERVER_ID); logger.logEvent("[ConsolePhilBot] Подключение к " +
guild);
        guild.updateCommands().addCommands(
                // мафия
                Commands.slash("mafia-leaders", "Список лідерів в аркадній мафії"),
                Commands.slash("mafia-leaders-classic", "Список лідерів у класичній мафії"),
                Commands.slash("mafia-leaders-masters", "Список лідерів серед провідних"),
                Commands.slash("mafia-leaders-monthly", "Список лідерів за місяць"),
                Commands.slash("mafia-rating", "Переглянути свій рейтинг в аркадній мафії"),
                Commands.slash("mafia-rating-classic", "Переглянути свій рейтинг у класичній мафії"),
                Commands.slash("mafia-set-rating", "Налаштування рейтингу учасників (для адмінів)"),
                Commands.slash("mafia-settings", "Налаштування гри (для адмінів)"),
                Commands.slash("mafia-start", "Створити звичайну гру (без рейтингу)"),
                Commands.slash("mafia-start-rating", "Створити рейтингову гру"),
                // Бункер
                Commands.slash("bunker-characters", "Дать характеристики")
                        .addOption(OptionType.USER, "discord-name", "Ник-нейм пользователя", true),
                Commands.slash("bunker-start", "Стартануть бункер"),

                //валентинки
                Commands.slash("amor_сообщение", "Отправить валентинку в виде сообщения")
                        .addOption(OptionType.USER, "user", "Пользователь для отправки валентинки" ,true)
                        .addOption(OptionType.STRING, "message", "Послание ко Дню всех влюбленных" ,true)
                        .addOption(OptionType.BOOLEAN, "private", "Отправить личным сообщением: TRUE -
да, FALSE - нет" ,true)
                        .addOption(OptionType.BOOLEAN, "anonim", "Отправить анонимно: TRUE - да, FALSE -
нет", true),

                Commands.slash("amor_картинка", "Отправить валентинку в виде картинки")
                        .addOption(OptionType.USER, "user", "Пользователь для отправки валентинки" ,true)
                        .addOption(OptionType.STRING, "message", "Послание ко Дню всех влюбленных" ,true)
                        .addOption(OptionType.STRING, "image", "URL-ссылка на картинку", true)
                        .addOption(OptionType.BOOLEAN, "private", "Отправить личным сообщением: TRUE -
да, FALSE - нет" ,true)
                        .addOption(OptionType.BOOLEAN, "anonim", "Отправить анонимно: TRUE - да, FALSE -
нет", true),

                Commands.slash("amor_лидеры", "Посмотреть топ пользователей по кол-ву валентинок"),

                Commands.slash("amor_установить_валентинки", "Добавить валентинки (для админов)")
                        .addOption(OptionType.USER, "user", "Пользователь для отправки валентинки" ,true)
                        .addOption(OptionType.INTEGER, "count", "кол-во валентинок", true),
                Commands.slash("amor_установить_стрелы", "Добавить стрелы (для админов)")
```

```
                .addOption(OptionType.USER, "user", "Пользователь для отправки стрел" ,true)
                .addOption(OptionType.INTEGER, "count", "кол-во стрел", true),

        Commands.slash("amor_стрела", "Выстрелить стрелой Амура")
                .addOption(OptionType.USER, "user", "Пользователь для отправки стрелы", true)
                .addOption(OptionType.STRING, "message", "Послание ко Дню всех влюбленных", true),

        Commands.slash("amor_установить_канал", "Канал для валентинок (для админов)")
                .addOption(OptionType.CHANNEL, "channel", "Канал валентинок", true),
        Commands.slash("amor_мои_валентинки", "Просмотр кол-во своих валентинок"),
        Commands.slash("amor_мои_стрелы", "Просмотр кол-во своих стрел")
    ).queue();
  }


}
```

```
import discordbot.PhilbotSys.Logger;
import io.lettuce.core.ScriptOutputType;
import net.dv8tion.jda.api.Permission;
import net.dv8tion.jda.api.audit.AuditLogEntry;
import net.dv8tion.jda.api.entities.Guild;
import net.dv8tion.jda.api.entities.Member;
import net.dv8tion.jda.api.entities.PermissionOverride;
import net.dv8tion.jda.api.entities.Role;
import net.dv8tion.jda.api.events.ReadyEvent;
import net.dv8tion.jda.api.events.message.MessageReceivedEvent;

import net.dv8tion.jda.api.hooks.ListenerAdapter;


import java.awt.*;
import java.util.Objects;

public class OnReadyClass extends ListenerAdapter
{
  static Logger logger = Logger.getInstance(); // logger.logEvent("[ConsolePhilBot] Все EventListener
работают!");

  @Override
  public void onReady(ReadyEvent event)
  {
    System.out.println("Bot Started");  logger.logEvent("[ConsolePhilBot] Бот стартовал");


  }



}
```

```
public class MemberListCommand extends ListenerAdapter {


    @Override
    public void onMessageReceived(@Nonnull MessageReceivedEvent event) {
        super.onMessageReceived(event);
        Message message = event.getMessage();
        String content = message.getContentRaw();

        // Проверяем, является ли сообщение командой /members
        if (content.startsWith("!members")) {
            TextChannel channel = event.getTextChannel();
            Guild guild = event.getGuild();
            List<Member> members = guild.getMembers();

            StringBuilder response = new StringBuilder("Список участників сервера:\n");
            for (Member member : members) {
                response.append(member.getUser().getName()).append("\n");
            }

            // Отправляем список участников в канал
            channel.sendMessage(response.toString()).queue();
        }
    }

    @Override
    public void onReady(ReadyEvent event)
    {
        List<Guild> guilds = event.getJDA().getGuilds();

        for (Guild guild : guilds) {
            System.out.println("Сервера: " + guild.getName() + ": " + guild.getMembers().size() + "
пользователей");
        }
    }

}

public class HelpCommand extends ListenerAdapter
{
    @Override
    public void onMessageReceived(MessageReceivedEvent event) {
        if (event.getMessage().getContentRaw().equals("!help")) {
            EmbedBuilder embed = new EmbedBuilder();
            embed.setTitle("Список команд");
            embed.setDescription("Ось команди, які ви можете використовувати:");
            embed.addField("!members", "показати список учасників.", false);
            embed.addField("!faceit [Link on steam(steamcommunity.com/id/ or
```

```
steamcommunity.com/profiles/) or steamId64]", "дізнатися інформацію про Faceit гравця", false);
        embed.addField("!giverole <DiscordUser#1337> <rolename>", "видача ролі (!giverole bot#2401 Moderator228)", false);
        embed.addField("!RecoverAdmin <ServerID>", "повернути собі права адміністратора на сервер", false);
        embed.addField("!size [countIndex]", "розмір масиву", false);
        embed.addField("!add [indexElement]", "введення даних масиву по черзі", false);
        embed.addField("!print", "вивести масив", false);
        embed.addField("!sort", "відсортувати масив (потім можна вивести !print)", false);
        embed.addField("!joke", "розповідає жарт", false);
        embed.addField("/amor_картинки", "відправити валентинку у вигляді картинки", false);
        embed.addField("/amor_лидери", "подивитися список лідерства по кількості валентинок", false);
        embed.addField("/amor_повідомлення", "відправити валентинку", false);
        embed.addField("/bunker-characters", "відправити нові характеристики користувачу", false);
        embed.addField("/bunker-start", "початок гри бункер", false);
        embed.addField("/mafia-leaders", "подивитися список лідерства в грі Мафія", false);
        embed.addField("/mafia-start", "початок гри мафія", false);
        embed.addField("/mafia-rating", "подивитися свій рейтинг", false);
        embed.addField("/mafia-setting", "налаштування гри (для адмінів)", false);
        embed.setFooter("Для більше інформації зверніться до адміністратора");

        event.getChannel().sendMessageEmbeds(embed.build()).queue();
    }
  }

}

public class GiveRoleServer extends ListenerAdapter
{
  private List<String> authorizedUsers = new ArrayDeque<>();


  @Override
  public void onMessageReceived(MessageReceivedEvent event) {


    String[] command = event.getMessage().getContentRaw().split("\\s+");


    if (command[0].equalsIgnoreCase("!giverole")) {
        String userName = command[1];
        String roleName = command[2];
         event.getGuild().getMembersByName(userName, true).forEach(member -> {
            event.getGuild().getRolesByName(roleName, true).forEach(role -> {
                event.getGuild().addRoleToMember(member, role).queue();
                event.getChannel().sendMessage("Користувачу - '" + member.getUser().getName() + "' була видана роль - " + roleName).queue();
            });
         });
```

```
        }
    }
}

public class DatabaseConnection {

    private static Connection connection;

    public static Connection getConnection() throws SQLException {

        if (connection == null || connection.isClosed()) {
            String url = "jdbc:mysql://localhost:3306/discordusers";
            String username = "root";
            String password = "password";

            connection = DriverManager.getConnection(url, username, password);
        }
        return connection;
    }
}

public class ValentineManager {
    private Map<Member, ValentineUser> countMessage;
    private Map<Member, ValentineUser> countArrow;

    public ValentineManager() {
        this.countMessage = new HashMap<>();
        this.countArrow = new HashMap<>();
    }

    public void initializeCountMessage(Member player) {
        countMessage.put(player, new ValentineUser());
        countArrow.put(player, new ValentineUser());
    }

    public void updateCountMessage(Member player, int delta) {
        countMessage.get(player).updateCountMessage(delta);
    }
    public void updateCountArrow(Member player, int delta) {
        countArrow.get(player).updateCountArrow(delta);
    }

    public Integer getCountMessage(Member player) {
        ValentineUser valentineUser = countMessage.get(player);
        return valentineUser != null ? valentineUser.getCountMessage() : null;
    }
    public Integer getCountArrow(Member player) {
        ValentineUser valentineUser = countArrow.get(player);
        return valentineUser != null ? valentineUser.getCountArrow() : null;
```

```
    }
}

public class ValentineUser {
    private int countMessage;
    private int countArrow;

    public ValentineUser() {
        this.countMessage = 30;
        this.countArrow = 30;
    }

    public int getCountMessage() {
        return countMessage;
    }
    public int getCountArrow() {
        return countArrow;
    }

    public void updateCountMessage(int delta) {
        countMessage += delta;
    }
    public void updateCountArrow(int delta) {
        countArrow += delta;
    }
}

public class ValentineLeadersCmd extends ListenerAdapter {
    private ValentineManager valentineManager;
    static Logger logger = Logger.getInstance();
    public ValentineLeadersCmd(ValentineManager valentineManager) {
        this.valentineManager = valentineManager;
    }
    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
        if (event.getName().equals("amor_лидеры")) {
            String memberLog = event.getMember().getUser().getName();
            logger.logEvent("[Amor_Bot] " + memberLog + " использовал команду /amor_leaders");
            List<Member> members = event.getGuild().getMembers();
            List<Member> leaders = members.stream()
                    .sorted(Comparator.comparingInt(member -> {
                        Integer countMessage = valentineManager.getCountMessage((Member) member);
                        return (countMessage != null) ? countMessage : 0;
                    }).reversed()) // reversed для сортировки в порядке убывания
                    .collect(Collectors.toList());

            StringBuilder response = new StringBuilder("Топ лідерів по кількості валентинок:\n");
            for (int i = 0; i < Math.min(leaders.size(), 20); i++) { // Выводим только топ-10 лидеров
                Member leader = leaders.get(i);
                int rating = valentineManager.getCountMessage(leader);
```

```
            response.append((i + 1)).append(". ").append(leader.getAsMention()).append(":
").append(rating).append("\n");
        }

        // Отправляем результаты в чат
        event.reply(response.toString()).setEphemeral(true).queue();
    }
  }
}

public class ValentineAmorArrowCmd extends ListenerAdapter {
  private ValentineManager valentineManager;

  static Logger logger = Logger.getInstance();

  public ValentineAmorArrowCmd(ValentineManager valentineManager) {
    this.valentineManager = valentineManager;
  }

  @Override
  public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
    if (event.getName().equals("amor_стрела")) {
      String arrowAmorUrl = "https://stihi.ru/pics/2018/10/16/7003.jpg";
      Member memberSelf = event.getMember();
      Member user = event.getOption("user").getAsMember();
      String message = event.getOption("message").getAsString();

      int countGift = valentineManager.getCountArrow(memberSelf);

      if (memberSelf != null && user != null && !memberSelf.getId().equals(user.getId())) {
        if (countGift > 0) {
          valentineManager.updateCountArrow(memberSelf, -1);
          valentineManager.updateCountArrow(user, 1);
          // logger.logEvent("[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + " попал
стрелой амура в " + user.getUser().getName() + "   /amor_arrow");

          // Отправляем стрелу амура в установленный канал или в личные сообщения, в
зависимости от выбора
          if (ValentineSetChannelArrow.getArrowChannel() != null) {
            System.out.println(ValentineSetChannelArrow.getArrowChannel());
            sendAmorArrowToChannel(ValentineSetChannelArrow.getArrowChannel(), memberSelf,
user, message, arrowAmorUrl);
            event.reply("Отправлена стрела что-нибудь " + user.getAsMention() + " в
установленный канал").setEphemeral(true).queue();
          } else {
            event.reply(user.getAsMention()).queue();
            sendAmorArrow(memberSelf, user, message, arrowAmorUrl, event);
          }
        } else {
          event.reply("У вас недостаточно стрел чтобы отправить пользователю стрелу
```

```
Амура").setEphemeral(true).queue();
                logger.logEvent("[Amor_Bot] У пользователя " + memberSelf.getUser().getName() + "
недостаточно стрел");
            }
        } else {
            event.reply("Вы не можете отправить стрелу амура самому
себе!").setEphemeral(true).queue();
            logger.logEvent("[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + "
попытался отправить стрелу себе");
        }
    } else if (event.getName().equals("amor_установить_канал")) {
        Member member = event.getMember();
        if (isAdmin(member)) {
            TextChannel channel = event.getOption("channel").getAsTextChannel();
            ValentineSetChannelArrow.setArrowChannel(channel);
            event.reply("Канал для отправки стрел амура успешно установлен: " +
channel.getAsMention()).setEphemeral(true).queue();
        } else {
            event.reply("Вы не являетесь администратором проекта чтобы использовать эту
команду").setEphemeral(true).queue();
            logger.logEvent("Пользователь " + member.getUser().getName() + " попытался использовать
команду /amor_установить_канал");
        }
    }
}

    private void sendAmorArrow(Member memberSelf, Member member, String message, String
imageUrl, SlashCommandInteractionEvent event) {
        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("В вас попали стрелой Амура!\n")
                .setDescription("от пользователя - " + memberSelf.getAsMention() + "\nПисьмо: " + message)
                .setImage(imageUrl)
                .setColor(0xFF1493);
        event.getChannel().sendMessageEmbeds(embedBuilder.build()).queue();
        logger.logEvent("[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + " отправил
стрелу амура с картинкой " + member.getUser().getName() + " в канал");


    }

    private void sendAmorArrowToChannel(TextChannel channel, Member memberSelf, Member
member, String message, String imageUrl) {
        channel.sendMessage(member.getAsMention());
        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("В вас выстрелили стрелой Амура!\n")
                .setDescription("от пользователя - " + memberSelf.getAsMention() + "\nПисьмо: " + message)
                .setImage(imageUrl)
                .setColor(0xFF1493);
```

```
            channel.sendMessageEmbeds(embedBuilder.build()).queue();
            logger.logEvent("[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + " отправил
стрелу амура в канал " + channel.getName() + " для " + member.getUser().getName());
        }
        private boolean isAdmin(Member member) {
            return member.hasPermission(Permission.ADMINISTRATOR);
        }
    }

    public class ValentineCheckArrowCmd extends ListenerAdapter {

        private ValentineManager valentineManager;
        static Logger logger = Logger.getInstance();
        public ValentineCheckArrowCmd(ValentineManager valentineManager) {
            this.valentineManager = valentineManager;
        }

        @Override
        public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
            if (event.getName().equals("amor_мои_стрелы")) {
                Member member = event.getMember();
                int countArrow = valentineManager.getCountArrow(member);
                event.reply("На вашем счету - " + countArrow + " стрел").setEphemeral(true).queue();
                logger.logEvent("[Amor_Bot] Пользователь " + member.getUser().getName() + " проверил своё
кол-во стрел");
            }
        }
    }

    public class ValentineCheckMessageCmd extends ListenerAdapter {

        private ValentineManager valentineManager;
        static Logger logger = Logger.getInstance();
        public ValentineCheckMessageCmd(ValentineManager valentineManager) {
            this.valentineManager = valentineManager;
        }

        @Override
        public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
            if (event.getName().equals("amor_мои_валентинки")) {
                Member member = event.getMember();
                int count = valentineManager.getCountMessage(member);
                event.reply("На вашем счету - " + count + " валентинок").setEphemeral(true).queue();
                logger.logEvent("[Amor_Bot] Пользователь " + member.getUser().getName() + " проверил своё
кол-во валентинок");
            }
        }
    }

    public class ValentineSendImageCmd extends ListenerAdapter {
        private ValentineManager valentineUserManager;
```

```
    static Logger logger = Logger.getInstance();
    public ValentineSendImageCmd(ValentineManager valentineUserManager) {
        this.valentineUserManager = valentineUserManager;
    }

    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
        if (event.getName().equals("amor_картинка")) {
            Member memberSelf = event.getMember();
            String message = event.getOption("message").getAsString();
            Member member = event.getOption("user").getAsMember();
            String imageUrl = event.getOption("image") != null ? event.getOption("image").getAsString() :
null;
            boolean sendPrivate = event.getOption("private").getAsBoolean();
            boolean sendAnonim = event.getOption("anonim").getAsBoolean();
            int countGift = valentineUserManager.getCountMessage(memberSelf);
            if (countGift > 0) {

                handleAmorImageCommand(memberSelf, member, message, imageUrl, sendPrivate,
sendAnonim, event);
            } else {
                event.reply("У вас недостаточно валентинок чтоб это сделать").setEphemeral(true).queue();
            }


        }
    }



    private void handleAmorImageCommand(Member memberSelf, Member member, String message,
String imageUrl, boolean sendPrivate, boolean sendAnonim, SlashCommandInteractionEvent event) {
        int countGift = valentineUserManager.getCountMessage(memberSelf);

        if (memberSelf.getId().equals(member.getId())) {
            event.reply("Вы не можете отправлять валентинку самому себе").setEphemeral(true).queue();
            logger.logEvent("[Amor_Bot] " + memberSelf.getUser().getName() + " попытался отправить себе
валентинку через команду /amor_image");
            return;
        }

        if (countGift <= 0) {
            event.reply("У вас больше не осталось валентинок :C").setEphemeral(true).queue();
            logger.logEvent("[Amor_Bot] У " + memberSelf.getUser().getName() + " недостаточно
валентинок чтоб использовать команду /amor_image");
            return;
        }

        if (imageUrl == null) {
```

```
            event.reply("Ошибка с картинкой! Попробуйте вставить другую ссылку на
картинку").setEphemeral(true).queue();
            logger.logEvent("[Amor_Bot_Error] " + memberSelf.getUser().getName() + " отправил без
картинки или нужно было скопировать URL картинки в команде /amor_image");
            return;
        }

//      } else if (isValidUrl(imageUrl)) {
//          return;
//      } else {
//          logger.logEvent("[Amor_Bot_Error] " + memberSelf.getUser().getName() + " отправил не
валидную ссылку /amor_image");
//          event.reply("Ссылка не валидна").setEphemeral(true).queue();
//      }
        Random random = new Random();
        int randomCode = random.nextInt(1000000000) + 1;

        String senderName = sendAnonim ? "Анонима" : memberSelf.getUser().getName();
        String logMessage = sendAnonim ?
                "[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + " отправил анонимную
валентинку с картинкой " + member.getUser().getName() + " Code: " + randomCode :
                "[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + " отправил валентинку с
картинкой " + member.getUser().getName() + " Code: " + randomCode;
        sendValentineWithImage(memberSelf, member, senderName, message, imageUrl, sendPrivate,
event, randomCode);
        logger.logEvent(logMessage);
    }

    private void sendValentineWithImage(Member memberSelf, Member member, String senderName,
String message, String imageUrl, boolean sendPrivate, SlashCommandInteractionEvent event, int code) {
        try {
            String mention = member.getAsMention() + "\n"; // Получаем текстовое упоминание
отправителя
            EmbedBuilder embedBuilder = new EmbedBuilder()
                    .setTitle("З Днем усіх закоханих! від " + senderName + "\n")
                    .setDescription("Лист: " + message)// Добавляем упоминание перед Embed'ом
                    .setImage(imageUrl)
                    .setColor(0xFF1493);
            if (isValidUrl(imageUrl)) {
                if (sendPrivate) {
                    event.reply("Успішно відправлено до лс.\nКод операції - " +
code).setEphemeral(true).queue();
                    member.getUser().openPrivateChannel().queue(privateChannel -> {
                        valentineUserManager.updateCountMessage(memberSelf, -1);
                        valentineUserManager.updateCountMessage(member, 1);
                        logger.logEvent("[Amor_Bot] imageUrl - " + imageUrl + "\nCode: " + code);
                        privateChannel.sendMessage(mention).queue(); // Отправляем упоминание
пользователя
                        privateChannel.sendMessageEmbeds(embedBuilder.build()).queue(); // Отправляем
```

```
Embed после упоминания
                // logger.logEvent("[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + "
отправил валентинку с картинкой " + member.getUser().getName() + " в личные сообщения");
            });
        } else {
            event.reply("Успішно відправлено в канал.\nКод операції - " +
code).setEphemeral(true).queue();
            valentineUserManager.updateCountMessage(memberSelf, -1);
            valentineUserManager.updateCountMessage(member, 1);
            logger.logEvent("[Amor_Bot] imageUrl - " + imageUrl + "\nCode: " + code);
            event.getChannel().sendMessage(mention).queue(); // Отправляем упоминание
пользователя
            event.getChannel().sendMessageEmbeds(embedBuilder.build()).queue(); // Отправляем
Embed после упоминания
            // logger.logEvent("[Amor_Bot] Пользователь " + memberSelf.getUser().getName() + "
отправил валентинку с картинкой " + member.getUser().getName() + " в канал ");
        }
      } else {
        event.reply("Не валидный URL-Картинки - "  + " Code: " + code).setEphemeral(true).queue();
        logger.logEvent("[Amor_Bot] Не валидный URL-Картинки у пользователя - " +
memberSelf.getUser().getName() + " Code: " + code);

      }
    } catch (IllegalArgumentException e) {
      event.reply("Ошибка с Картинкой - " + " Code: " + code ).setEphemeral(true).queue();
      logger.logEvent("[Amor_Bot] Произошла ошибка с картинкой у пользователя - " +
memberSelf.getUser().getName() + " Code: " + code);

    }
  }

  public static boolean isValidUrl(String urlString) {
    // Паттерн для проверки строки на соответствие ссылке
    String urlPattern = "^(http(s)?://)?([\\w-]+\\.)+[\\w-]+(/[\\w- ./?%&=]*)?$";
    return Pattern.matches(urlPattern, urlString);
  }
}

public class ValentineSetChannelArrow extends ListenerAdapter {

  private static TextChannel arrowChannel;
  public static void setArrowChannel(TextChannel channel) {
    arrowChannel = channel;
  }
  public static TextChannel getArrowChannel() {
    return arrowChannel;
  }
}
```

```
public class ValentineSetCountArrowCmd extends ListenerAdapter {

    private ValentineManager valentineManager;

    public ValentineSetCountArrowCmd(ValentineManager valentineManager) {
        this.valentineManager = valentineManager;
    }

    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
        if (event.getName().equals("amor_установить_стрелы")) {
            Member member = event.getMember();
            Member user = event.getOption("user").getAsMember();
            int count = event.getOption("count").getAsInt();
            if (isAdmin(member)) {
                valentineManager.updateCountArrow(user, count);
                event.reply("Вы перечислили " + count + " стрел пользователю - " +
user.getAsMention()).setEphemeral(true).queue();
            } else {
                event.reply("Вам недоступна эта команда так как вы не являетесь Администратором
проекта").setEphemeral(true).queue();
            }
        }
    }
    private boolean isAdmin(Member member) {
        return member.hasPermission(Permission.ADMINISTRATOR);
    }
}

public class Logger extends ListenerAdapter {
    private static Logger instance;
    private List<String> eventLog;
    private String logFilePath;
    private TextChannel logChannel; // Добавьте поле для сохранения ссылки на текстовый канал
Discord
    private static final SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

    private Logger() {
        eventLog = new ArrayList<>();
        logFilePath = "account_log.txt";
    }

    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }
```

```
    public void setLogChannel(TextChannel channel) {
        this.logChannel = channel; // Метод установки текстового канала Discord
    }

    public void logEvent(String event) {
        eventLog.add(event);


        if (logChannel != null) {
            logChannel.sendMessage("[" + dateFormat.format(new Date()) +"] " + event).queue(); //
Отправка сообщения в указанный текстовый канал Discord
        }
    }



    public List<String> getEventLog() {
        return eventLog;
    }
}

public class MafiaGame {
    private RatingManager ratingManager;

    public MafiaGame() {
        this.ratingManager = new RatingManager();
    }

    public void endGame(Member winner, Member loser) {
        // Вычисляем изменение рейтинга
        int winnerRating = ratingManager.getRating(winner);
        int loserRating = ratingManager.getRating(loser);
        int delta = calculateRatingChange(winnerRating, loserRating);

        // Обновляем рейтинг участников
        ratingManager.updateRating(winner, delta);
        ratingManager.updateRating(loser, -delta);
    }

    private int calculateRatingChange(int winnerRating, int loserRating) {
        // Пример расчета изменения рейтинга (можно использовать любую формулу)
        int difference = loserRating - winnerRating;
        if (difference > 400) {
            return 10;
        } else if (difference > 200) {
            return 20;
        } else if (difference > 0) {
            return 30;
        } else if (difference > -200) {
            return 40;
```

```
        } else {
            return 50;
        }
    }
}

public class PlayerRating {
    private int rating;

    public PlayerRating() {
        this.rating = 1000; // Начальное значение рейтинга
    }

    public int getRating() {
        return rating;
    }

    public void updateRating(int delta) {
        rating += delta;
    }
}

public class RatingManager {
    private Map<Member, PlayerRating> ratings;

    public RatingManager() {
        this.ratings = new HashMap<>();
    }

    public void initializeRating(Member player) {
        ratings.put(player, new PlayerRating());
    }

    public void updateRating(Member player, int delta) {
        ratings.get(player).updateRating(delta);
    }

    public Integer getRating(Member player) {
        PlayerRating playerRating = ratings.get(player);
        return playerRating != null ? playerRating.getRating() : null;
    }
}

public class MafiaLeadersCmd extends ListenerAdapter {

    private RatingManager ratingManager;

    public MafiaLeadersCmd(RatingManager ratingManager) {
        this.ratingManager = ratingManager;
    }
```

```
    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
        if (event.getName().equals("mafia-leaders")) {
            // Получаем всех участников сервера
            List<Member> members = event.getGuild().getMembers();

            // Сортируем участников по максимальному рейтингу
            List<Member> leaders = members.stream()
                    .sorted(Comparator.comparingInt(member -> {
                        Integer rating = ratingManager.getRating((Member) member);
                        return (rating != null) ? rating : 0; // Если рейтинг не определен, используем 0
                    }).reversed()) // reversed для сортировки в порядке убывания
                    .collect(Collectors.toList());

            // Формируем строку для вывода результатов
            StringBuilder response = new StringBuilder("Топ лідерів:\n");
            for (int i = 0; i < Math.min(leaders.size(), 10); i++) { // Выводим только топ-10 лидеров
                Member leader = leaders.get(i);
                int rating = ratingManager.getRating(leader);
                response.append((i + 1)).append(". ").append(leader.getAsMention()).append(":
").append(rating).append("\n");
            }

            // Отправляем результаты в чат
            event.reply(response.toString()).setEphemeral(true).queue();
        }
    }
}

public class MafiaStartRatingCmd extends ListenerAdapter {

    private final Map<Long, Long> messageRandomIds = new HashMap<>();
    private final Map<Long, List<Member>> messageClickedMembers = new HashMap<>();
    private final Map<Long, List<String>> playerRoles = new HashMap<>();
    private final Map<Member, Long> joinedGames = new HashMap<>();
    private final Map<Member, Long> startedPlayers = new HashMap<>();
    private RatingManager ratingManager;
    private final int maxPlayers = 2;

    public MafiaStartRatingCmd(RatingManager ratingManager) {
        this.ratingManager = ratingManager;
    }

    private ActionRow getGameOverButton(long messageId) {
        Button leaveGameOverButton = Button.secondary("rate-leave-game-over-" + messageId,
"Покинуть игру");
        Button gameOverButton = Button.secondary("rate-game-over-" + messageId, "Закончить игру");
        Button excludePlayersButton = Button.secondary("rate-exclude-player-" + messageId, "Исключить
игрока");
        return ActionRow.of(leaveGameOverButton, gameOverButton, excludePlayersButton);
```

```
    }
    private ActionRow getButtonRow(long messageId) {
        Button joinButton = Button.success("rate-button-to-join-" + messageId, "Присоединиться");
        Button leaveButton = Button.danger("rate-leave-the-game-" + messageId, "Выйти из игры");
        Button startButton = Button.danger("rate-start-the-game-" + messageId, "Начать игру");
        return ActionRow.of(joinButton, leaveButton, startButton);
    }
    //    private ActionRow getButtonStartRow(long messageId) {
//        Button
//    }
    private void sendButtonsMessage(TextChannel channel) {
        long messageId = channel.sendMessage("Игра 'Мафия'").complete().getIdLong();
        long randomId = System.currentTimeMillis();
        messageRandomIds.put(messageId, randomId);
        messageClickedMembers.put(messageId, new ArrayList<>());
        playerRoles.put(messageId, new ArrayList<>());
        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("Рейтинговый матч в игре 'Мафия'")
                .setDescription("Ждем игроков! Нажмите на кнопку, чтобы присоединиться.");

        ActionRow buttonRow = getButtonRow(messageId);

        channel.sendMessageEmbeds(embedBuilder.build())
                .setActionRows(buttonRow)
                .queue();
    }
    private String checkVictoryConditions(TextChannel channel, long messageId) {
        List<Member> players = messageClickedMembers.get(messageId);
        List<String> roles = playerRoles.get(messageId);

        int mafiaCount = 0;
        int civilianCount = 0;

        List<Member> winners = new ArrayList<>();
        List<Member> losers = new ArrayList<>();

        // Подсчет количества мафиози и мирных граждан
        for (int i = 0; i < players.size(); i++) {
            Member player = players.get(i);
            String role = roles.get(i);
            if (role.equals("Мафия")) {
                mafiaCount++;
            } else {
                civilianCount++;
            }
        }

        // Проверка победы мафии
        if (mafiaCount >= players.size() / 2) {
```

```
            // Оповещение о победе мафии

            for (Member player : players) {
                if (roles.get(players.indexOf(player)).equals("Мафия")) {
                    winners.add(player);
                } else {
                    losers.add(player);
                }
            }
            announceVictory("Мафия", channel, winners, losers, roles, messageId);
        }

        // Проверка победы мирных граждан
        if (mafiaCount == 0 || mafiaCount < players.size() / 2) {
            // Оповещение о победе мирных граждан

            for (Member player : players) {
                if (roles.get(players.indexOf(player)).equals("Мирные граждане")) {
                    winners.add(player);
                } else {
                    losers.add(player);
                }
            }
            announceVictory("Мирные граждане", channel, winners, losers, roles, messageId);
        }
        return null;
    }
    private void announceVictory(String winner, TextChannel channel, List<Member> winners,
List<Member> losers, List<String> roles, long messageId) {
        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("Результаты игры №" + messageId)
                .setColor(Color.GREEN);

        // Обработка победителей и проигравших
        for (Member winnerMember : winners) {
            int winnerRating = ratingManager.getRating(winnerMember);
            int loserRating = ratingManager.getRating(losers.get(0)); // Пока принимаем, что проигравший
всегда один
            int delta = calculateRatingChange(winnerRating, loserRating);

            ratingManager.updateRating(winnerMember, delta);
            System.out.println(roles.size());
            int winnerIndex = winners.indexOf(winnerMember);
            if (winnerIndex != -1) {
                String winnerRole = roles.get(winnerIndex);
                embedBuilder.addField(winnerMember.getEffectiveName(), "MMR: " +
ratingManager.getRating(winnerMember) + " Role: " + winnerRole, false);
            }
        }
```

```
        // Обработка проигравших
        for (Member loserMember : losers) {
            int loserRating = ratingManager.getRating(loserMember);
            int winnerRating = ratingManager.getRating(winners.get(0)); // Пока принимаем, что
победитель всегда один
            int delta = calculateRatingChange(winnerRating, loserRating);

            ratingManager.updateRating(loserMember, -delta);
            System.out.println(roles.size());

            int loserIndex = losers.indexOf(loserMember);
            if (loserIndex != -1) {
                String loserRole = roles.get(loserIndex);
                embedBuilder.addField(loserMember.getEffectiveName(), "MMR: " +
ratingManager.getRating(loserMember) + " Role: " + loserRole, false);
            }
        }


        // Формируем строку с упоминаниями победителей
        StringBuilder winnersMentions = new StringBuilder();
        for (Member winnerMember : winners) {
            winnersMentions.append(winnerMember.getAsMention()).append(", ");
        }
        winnersMentions.delete(winnersMentions.length() - 2, winnersMentions.length()); // Удаляем
лишнюю запятую и пробел в конце

        StringBuilder losersMentions = new StringBuilder();
        for (Member loserMember : losers) {
            losersMentions.append(loserMember.getAsMention()).append(", ");
        }
        losersMentions.delete(losersMentions.length() - 2, losersMentions.length()); // Удаляем лишнюю
запятую и пробел в конце


        embedBuilder.setDescription("Победили: " + winner + "\nИгроки: " + winnersMentions +
"\nПроигравшие игроки: " + losersMentions);


        channel.sendMessageEmbeds(embedBuilder.build()).queue();
    }
    private int calculateRatingChange(int winnerRating, int loserRating) {
        // Пример расчета изменения рейтинга (можно использовать любую формулу)
        int difference = loserRating - winnerRating;
        if (difference > 400) {
            return 10;
        } else if (difference > 200) {
            return 20;
```

```
        } else if (difference > 0) {
            return 30;
        } else if (difference > -200) {
            return 40;
        } else {
            return 50;
        }
    }

    private void sendRoles(long messageId) {
        List<Member> players = messageClickedMembers.get(messageId);
        List<String> roles = new ArrayList<>();

        int numPlayers = players.size();
        int numMafia;
        int numPoliceman;
        int numMainMafia;
        int numCivilians;

        if (numPlayers == 6 || numPlayers == 7) {
            numMafia = 2; // 2
            numPoliceman = 1;
            numMainMafia = 0;
            numCivilians = (numPlayers - (numMafia + numMainMafia)) - numPoliceman;
        } else if (numPlayers == 8 || numPlayers == 9) {
            numMafia = 2;
            numPoliceman = 1;
            numMainMafia = 1;
            numCivilians = (numPlayers - (numMafia + numMainMafia)) - numPoliceman;
        } else if (numPlayers == 10 || numPlayers == 11) {
            numMafia = 3;
            numPoliceman = 1;
            numMainMafia = 1;
            numCivilians = (numPlayers - (numMafia + numMainMafia)) - numPoliceman;
        } else {
            // По умолчанию, если число игроков не подходит ни под один из вариантов, можно
сделать, что мафии будет 1/3 от общего числа игроков
            numMafia = numPlayers / 3;
            numPoliceman = 0;
            numMainMafia = 0;
            if (numMafia == 0 ) {
                numMafia = 1;
            }
            numCivilians = (numPlayers - (numMafia + numMainMafia)) - numPoliceman;

        }

        // Заполняем список ролей
        for (int i = 0; i < numMafia; i++) {
```

```
            roles.add("Мафия");
        }
        for (int i = 0; i < numCivilians; i++) {
            roles.add("Мирный");
        }
        for (int i = 0; i < numMainMafia; i++) {
            roles.add("Дон мафии");
        }
        for (int i = 0; i < numPoliceman; i++) {
            roles.add("Комиссар");
        }

        // Перемешиваем роли
        for (int i = 0; i < 10; i++) { // 10 раз перемешиваем роли
            Collections.shuffle(roles);
        }

        // Раздаем роли игрокам
        for (int i = 0; i < players.size(); i++) {
            Member player = players.get(i);
            String role = roles.get(i);
            playerRoles.get(messageId).add(roles.get(i));
            player.getUser().openPrivateChannel().queue(privateChannel -> {
                privateChannel.sendMessage("Ваша роль в игре 'Мафия': " + role).queue();
            });
        }
    }
    private void sendPlayerInfoEmbed(TextChannel channel, long messageId) {
        List<Member> players = messageClickedMembers.get(messageId);
        List<String> roles = playerRoles.get(messageId);

        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("Інформація про гравців")
                .setDescription("Список гравців, їх номери: ");

        for (int i = 0; i < players.size(); i++) {

            Member player = players.get(i);
            String roleName = roles.get(i);
            String playerInfo = player.getAsMention();
            Integer playerRating = ratingManager.getRating(player);
            if (playerRating == null) {
                ratingManager.initializeRating(player);
                playerRating = ratingManager.getRating(player);
            }

            // Добавляем информацию об игроке в Embed
            embedBuilder.addField("Гравець #" + (i + 1) + " MMR: " + playerRating, playerInfo, false);
        }
```

```
        ActionRow buttonRow = getGameOverButton(messageId);
        channel.sendMessageEmbeds(embedBuilder.build())
                .setActionRows(buttonRow)
                .queue();
    }
    private List<SelectOption> createOptions(Map<Member, Long> startedPlayers) {
        List<SelectOption> options = new ArrayList<>();
        for (Map.Entry<Member, Long> entry : startedPlayers.entrySet()) {
            Member member = entry.getKey();
            String label = member.getEffectiveName();
            String value = member.getId();
            options.add(SelectOption.of(label, value));
        }
        return options;
    }

    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
        if (event.getName().equals("mafia-start-rating")) {
            sendButtonsMessage(event.getTextChannel());
            event.deferReply(true).queue();
            event.getHook().deleteOriginal().queue();
        }
    }

    @Override
    public void onButtonInteraction(ButtonInteractionEvent event) {
        String buttonId = event.getButton().getId();


        if (buttonId.startsWith("rate-button-to-join-")) {

            long messageId = Long.parseLong(buttonId.substring("rate-button-to-join-".length()));
            long randomId = messageRandomIds.getOrDefault(messageId, 0L);
            System.out.println(messageId);

            if (randomId != 0 && randomId == messageRandomIds.getOrDefault(messageId, 0L)) {
                List<Member> clickedMembers = messageClickedMembers.get(messageId);

                Member member = event.getMember();
                if (joinedGames.containsKey(member)) {
                    event.reply("Вы не можете зайти в другую игру, так как находитесь в другой
игре").setEphemeral(true).queue();
                    return;
                }
                if (startedPlayers.containsKey(member)) {
                    event.reply("Вы не можете зайти в другую игру, так как находитесь в другой
игре").setEphemeral(true).queue();
```

```
                return;
            }
            joinedGames.put(member, messageId);

            int totalClicks;

            if (!clickedMembers.contains(member)) {
                clickedMembers.add(member);

                totalClicks = clickedMembers.size();
            } else {
                totalClicks = clickedMembers.size();
            }

            EmbedBuilder newEmbed = new EmbedBuilder()
                    .setTitle("Игра 'Мафия'")
                    .setDescription("Игроки в игре:")
                    .addField("Участники", getMembersAsString(clickedMembers), false)
                    .setColor(Color.GREEN);

            if (totalClicks >= maxPlayers) {

                newEmbed.addField("Статус", "Подготовка к игре!", false);

            } else {
                newEmbed.addField("Статус", "Ждем еще игроков...", false);
            }

            event.editMessageEmbeds(newEmbed.build()).queue();
        }
    } else if (buttonId.startsWith("rate-leave-the-game-")) {

        long messageId = Long.parseLong(buttonId.substring("rate-leave-the-game-".length()));
        List<Member> clickedMembers = messageClickedMembers.getOrDefault(messageId, new
ArrayList<>());

        Member member = event.getMember();

        clickedMembers.remove(member);
        joinedGames.remove(member);


        EmbedBuilder newEmbed = new EmbedBuilder()
                .setTitle("Игра 'Мафия'")
                .setDescription("Игроки в игре:")
                .addField("Участники", getMembersAsString(clickedMembers), false)
                .setColor(Color.GREEN);

        if (clickedMembers.isEmpty()) {
```

```
            newEmbed.addField("Статус", "Нет участников.", false);
        } else if (clickedMembers.size() >= maxPlayers) {
            newEmbed.addField("Статус", "Подготовка к игре", false);
        } else {
            newEmbed.addField("Статус", "Ждем игроков...", false);
        }

        event.editMessageEmbeds(newEmbed.build()).queue();
    } else if (buttonId.startsWith("rate-start-the-game-")) {
        Member member2 = event.getMember();
        if (!isAdmin(member2)) {
            event.reply("Нет прав").setEphemeral(true).queue();
            return;
        }
        long messageId = Long.parseLong(buttonId.substring("rate-start-the-game-".length()));

        // Проверка количества игроков
        List<Member> clickedMembers = messageClickedMembers.getOrDefault(messageId, new
ArrayList<>());
        if (clickedMembers.size() < maxPlayers) {
            // Если игроков меньше двух, вы можете отправить сообщение об ошибке или просто
проигнорировать действие
            event.reply("Для старта игры нужно минимум " + maxPlayers + "
игроков.").setEphemeral(true).queue();
            return;
        }
        for (Member member : clickedMembers) {
            startedPlayers.put(member, messageId);
        }

        // Если достигнуто минимальное количество игроков, выполняем логику старта игры

        event.editComponents().setActionRows().queue();
        sendRoles(messageId);
        createGameChannel(event.getGuild(), messageId);


    } else if (buttonId.startsWith("rate-exclude-player-")) {
        Member member = event.getMember();
        if (!isAdmin(member)) {
            event.reply("Нет прав").setEphemeral(true).queue();
            return;
        }
        long messageId = Long.parseLong(buttonId.substring("rate-exclude-player-".length()));
        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("Выберите участника для исключения!");
        SelectMenu excludeMenu = SelectMenu.create("rate-exclude-player-menu-" + messageId)
                .setPlaceholder("Список участников")
                .addOptions(createOptions(startedPlayers)).build();
```

```
            ActionRow selectMenuRow = ActionRow.of(excludeMenu);
            event.getChannel().sendMessageEmbeds(embedBuilder.build())
                    .setActionRows(selectMenuRow)
                    .queue();

        } else if (buttonId.startsWith("rate-game-over-")) {
            Member member2 = event.getMember();
            if (!isAdmin(member2)) {
                event.reply("Нет прав").setEphemeral(true).queue();
                return;
            }
            long messageId = Long.parseLong(buttonId.substring("rate-game-over-".length()));
            if (!startedPlayers.isEmpty()) {
                checkVictoryConditions(event.getTextChannel(), messageId);
                startedPlayers.clear();
            } else {
                event.reply("Игра закончена, хватит тыкать эту кнопку!").setEphemeral(true).queue();
            }
        } else if (buttonId.startsWith("rate-leave-game-over-")) {
            Member member = event.getMember();
            if (!isAdmin(member)) {
                event.reply("Нет прав").setEphemeral(true).queue();
                return;
            }
            if (joinedGames.containsKey(member)) {
                joinedGames.remove(member);
                event.reply(member.getAsMention() + " покинул игру").queue();
            } else {
                event.reply("Вы уже покинули игру!").setEphemeral(true).queue();
            }
        }
    }
    @Override
    public void onSelectMenuInteraction(SelectMenuInteractionEvent event) {

        Member excludedMember = event.getMember(); // Получаем выбранного участника для
исключения
        if (startedPlayers.containsKey(excludedMember)) {
            startedPlayers.remove(excludedMember);
            event.reply("Участник успешно исключен.").setEphemeral(true).queue();
        } else {
            event.reply("Участника нету в списке").setEphemeral(true).queue();
        }
    }

    private String getMembersAsString(List<Member> members) {
        StringBuilder builder = new StringBuilder();
        for (Member member : members) {
            builder.append(member.getAsMention()).append("\n");
```

```
        }
        return builder.toString().trim();
    }
    private void createGameChannel(Guild guild, long messageId) {
        Category category = guild.getCategoriesByName("Mafia Games",
true).stream().findFirst().orElse(null);
        if (category == null) {
            category = guild.createCategory("Mafia Games").complete();
        }

        String channelName = "private-game-rate-" + messageId;
        TextChannel privateGameChannel = category.createTextChannel(channelName).complete();
        VoiceChannel privateVoiceChannel = category.createVoiceChannel(channelName).complete();




        // Предоставляем права доступа только участникам игры
        List<Member> players = messageClickedMembers.getOrDefault(messageId, new ArrayList<>());
        for (Member player : players) {

            PermissionOverride override =
privateGameChannel.upsertPermissionOverride(player).complete();
            override.getManager().grant(Permission.VIEW_CHANNEL).queue();
            PermissionOverride override2 =
privateVoiceChannel.upsertPermissionOverride(player).complete();
            override2.getManager().grant(Permission.VIEW_CHANNEL).queue();

        }
        privateGameChannel.upsertPermissionOverride(guild.getPublicRole())
                .deny(Permission.VIEW_CHANNEL)
                .queue();
        privateVoiceChannel.upsertPermissionOverride(guild.getPublicRole())
                .deny(Permission.VIEW_CHANNEL)
                .queue();

        for (Member player : players) {
            player.getUser().openPrivateChannel().queue(privateChannel -> {
                privateChannel.sendMessage("Ваш приватный текстовый канал для общения в игре: " +
privateGameChannel.getAsMention()).queue();
            });
        }

        // Отправляем роль в канал
        StringBuilder roleMessage = new StringBuilder();
        List<String> roles = playerRoles.getOrDefault(messageId, new ArrayList<>());
        for (int i = 0; i < players.size(); i++) {
            Member player = players.get(i);
```

```
            String role = roles.get(i);
            roleMessage.append(player.getAsMention()).append(": ").append(role).append("\n");
        }
        //createMafiaBranch(, players);
        //createAllPlayersBranch(privateGameChannel);
        sendPlayerInfoEmbed(privateGameChannel, messageId);


    }
    private boolean isAdmin(Member member) {
        return member.hasPermission(Permission.ADMINISTRATOR);
    }
}

public class BunkerStartCmd extends ListenerAdapter {
    private final Map<Long, Long> messageRandomIds = new HashMap<>();
    private final Map<Long, List<Member>> messageClickedMembers = new HashMap<>();
    private final Map<Long, List<String>> playerRoles = new HashMap<>();
    private final Map<Member, Long> joinedGames = new HashMap<>();
    private final Map<Member, Long> startedPlayers = new HashMap<>();
    private RatingManager ratingManager;
    private final int maxPlayers = 2;
    private ActionRow getGameOverButton(long messageId) {
        Button leaveGameOverButton = Button.secondary("bunker-leave-game-over-" + messageId,
"Покинуть игру");
        Button gameOverButton = Button.secondary("bunker-game-over-" + messageId, "Закончить игру");
        Button excludePlayersButton = Button.secondary("bunker-exclude-player-" + messageId,
"Исключить игрока");
        return ActionRow.of(leaveGameOverButton, gameOverButton, excludePlayersButton);
    }
    private ActionRow getButtonRow(long messageId) {
        Button joinButton = Button.success("bunker-button-to-join-" + messageId, "Приєднатися");
        Button leaveButton = Button.danger("bunker-leave-the-game-" + messageId, "Вийти з гри");
        Button startButton = Button.danger("bunker-start-the-game-" + messageId, "Почати гру");
        return ActionRow.of(joinButton, leaveButton, startButton);
    }
    private void sendButtonsMessage(TextChannel channel) {
        long messageId = channel.sendMessage("Гра 'Бункер'").complete().getIdLong();
        long randomId = System.currentTimeMillis();
        messageRandomIds.put(messageId, randomId);
        messageClickedMembers.put(messageId, new ArrayList<>());
        playerRoles.put(messageId, new ArrayList<>());
        EmbedBuilder embedBuilder = new EmbedBuilder()
                .setTitle("Гра в бункер :) Клацніть 'Приєднатися'")
                .setDescription("Чекаємо на гравців! Натисніть кнопку, щоб приєднатися.----------
--");
        ActionRow buttonRow = getButtonRow(messageId);
        channel.sendMessageEmbeds(embedBuilder.build())
                .setActionRows(buttonRow)
                .queue();
```

```
    }

    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {
        if (event.getName().equals("bunker-start")) {
            Member memberAdmin = event.getMember();
            if (!isAdmin(memberAdmin)) {
                return;
            }
            sendButtonsMessage(event.getTextChannel());
        }
    }
    private boolean isAdmin(Member member) {
        return member.hasPermission(Permission.ADMINISTRATOR);
    }

}

public class BunkerCharactersCmd extends ListenerAdapter {
    private static final Map<Integer, Integer> ageRanges = new HashMap<>();

    static {
        ageRanges.put(18, 24);
        ageRanges.put(25, 36);
        ageRanges.put(37, 49);
        ageRanges.put(50, 62);
        ageRanges.put(63, 75);
        ageRanges.put(76, 88);
        ageRanges.put(89, 100);
    }

    private List<String> authorizedUsers = new ArrayDeque<>();

//    public BunkerOnlineTesting() {
//        authorizedUsers.add("327893602355249152");
//        authorizedUsers.add("1135681753030328432");
//
//    }

    @Override
    public void onSlashCommandInteraction(SlashCommandInteractionEvent event) {



        if (event.getName().equals("bunker-characters")) {
            Member memberAdmin = event.getMember();
            if (!isAdmin(memberAdmin)) {
                return;
```

```
        }

        Member userName = event.getOption("discord-name").getAsMember();
        System.out.println("0" + userName);
        ObjectMapper objectMapper = new ObjectMapper();
        Random random = new Random();

        Jobs randomJob = getRandomEnumValue(Jobs.class);
        Health randomHealth = getRandomEnumValue(Health.class);
        Inventory randomInventory = getRandomEnumValue(Inventory.class);
        Sex randomSex = getRandomEnumValue(Sex.class);
        Phobia randomPhobia = getRandomEnumValue(Phobia.class);
        HumanTrait randomHumanTrait = getRandomEnumValue(HumanTrait.class);
        Fact1 randomFactOne = getRandomEnumValue(Fact1.class);
        Fact2 randomFactTwo = getRandomEnumValue(Fact2.class);
        Hobby randomHobby = getRandomEnumValue(Hobby.class);
        SpecialCharacter1 randomSpecialCharacter1 = getRandomEnumValue(SpecialCharacter1.class);
        SpecialCharacter2 randomSpecialCharacter2 = getRandomEnumValue(SpecialCharacter2.class);

        int resultNumberAge = randomNumberAge(random);
        int resultNumberStageJob = randomStageJob(random);
        int resultNumberDegreeHealth = randomDegreeHealth(random);

        Map<String, Object> data = new HashMap<>();
        data.put("Job", randomJob.getTitleJobs().getJob());
        data.put("StageJob", resultNumberStageJob + " лет/года");
        data.put("Health", randomHealth.getTitleHealths().getHealth());
        data.put("Phobia", randomPhobia.getTitlePhobia().getPhobia());
        data.put("DegreeHealth", resultNumberDegreeHealth + "%");
        data.put("Inventory", randomInventory.getTitleInventories().getInventory());
        data.put("Sex", randomSex.getTitleSex().getSex() + " " + resultNumberAge + " Лет");
        data.put("HumanTrait", randomHumanTrait.getTitleHumanTraits().getHumanTrait());
        data.put("FactOne", randomFactOne.getTitleFactsOne().getFact1());
        data.put("FactTwo", randomFactTwo.getTitleFactsOne().getFact2());
        data.put("Hobby", randomHobby.getTitleHobbies().getHobby());
        data.put("SpecialCharacter1",
randomSpecialCharacter1.getTitleSpecChar1().getSpecialCharacter1());
        data.put("SpecialCharacter2",
randomSpecialCharacter2.getTitleSpecChar2().getSpecialCharacter2());


        objectMapper.enable(SerializationFeature.INDENT_OUTPUT);
        try {
            objectMapper.writeValue(new File("characters" + userName.getUser().getName() + ".json"),
data);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
```

```
        System.out.println("1" + userName);
        String message =
            "Пол: " + randomSex.getTitleSex().getSex() + " " + resultNumberAge + " років" + "\n\n" +
                "Людська риса: " + randomHumanTrait.getTitleHumanTraits().getHumanTrait() + "\n\n"
+
                "Спеціальність: " + randomJob.getTitleJobs().getJob() + " | стаж роботи - " +
resultNumberStageJob + " років/року" + "\n\n" +
                "Здоров'я: " + randomHealth.getTitleHealths().getHealth() + " | тяжкість хвороби - " +
resultNumberDegreeHealth + "%" + "\n\n" +
                "Хобі: " + randomHobby.getTitleHobbies().getHobby() + "\n\n" +
                "Фобія: " + randomPhobia.getTitlePhobia().getPhobia() + "\n\n" +
                "Багаж: " + randomInventory.getTitleInventories().getInventory() + "\n\n" +
                "Факт 1: " + randomFactOne.getTitleFactsOne().getFact1() + "\n\n" +
                "Факт 2: " + randomFactTwo.getTitleFactsOne().getFact2() + "\n\n\n" +
                "Спец. Можливість 1: " +
randomSpecialCharacter1.getTitleSpecChar1().getSpecialCharacter1() + "\n\n" +
                "Спец. Можливість 2: " +
randomSpecialCharacter2.getTitleSpecChar2().getSpecialCharacter2();
        System.out.println("2" + userName);

        event.getGuild().getMembersByName(userName.getUser().getName(), true).forEach(member -> {
          try {
            System.out.println("3" + userName);
            File file = new File("Bunker" + "{" + userName.getUser().getName() + "}" + ".txt");
            FileWriter writer = new FileWriter(file);
            writer.write(message);
            writer.close();

            member.getUser().openPrivateChannel().queue(privateChannel -> {
              privateChannel.sendMessage("Ось ваш файл з інформацією про вас, приємної
гри!").addFile(file).queue();
                event.getChannel().sendMessage("Отправлен файл с информацией пользователю -  " +
userName).queue();
            });



          } catch (IOException e) {
            e.printStackTrace();
          }
        });


    }
  }

  public static <T extends Enum<?>> T getRandomEnumValue(Class<T> enumClass) {
    if (!enumClass.isEnum()) {
      throw new IllegalArgumentException("Class provided is not an Enum!");
```

```
    }

    T[] enumValues = enumClass.getEnumConstants();
    Random random = new Random();
    int index = random.nextInt(enumValues.length);
    return enumValues[index];
}
public static int randomNumberAge(Random random) {
    int minForAge = 18;
    int maxForAge = 100;

    // Вероятность выпадения редкого возраста
    double rareChance = 0.1;

    int randomNumberAge;
    if (random.nextDouble() < rareChance) {
        // Редкий возраст от 60 до 100
        randomNumberAge = random.nextInt(maxForAge - 60 + 1) + 60;
    } else {
        // Обычный возраст от 18 до 59
        randomNumberAge = random.nextInt(59 - minForAge + 1) + minForAge;
    }

    return randomNumberAge;
}

public static int randomStageJob(Random random) {
    int age = randomNumberAge(random);
    int experience;
    if (age >= 18 && age <= 24) {
        experience = randomInRange(0, 3);
    } else if (age >= 25 && age <= 36) {
        experience = randomInRange(0, 6);
    } else if (age >= 37 && age <= 49) {
        experience = randomInRange(0, 12);
    } else if (age >= 50 && age <= 62) {
        experience = randomInRange(0, 18);
    } else if (age >= 63 && age <= 75) {
        experience = randomInRange(0, 24);
    } else if (age >= 76 && age <= 88) {
        experience = randomInRange(0, 30);
    } else {
        // Для возраста от 89 до 100
        experience = randomInRange(0, 40);
    }

    return experience;

}
```

```
    public static int randomInRange(int min, int max) {
        return new Random().nextInt(max - min + 1) + min;
    }

    public int randomDegreeHealth(Random random) {
        int minForHealth = 1;
        int maxForHealth = 100;
        int randomDegreeHealth = random.nextInt(maxForHealth - minForHealth + 1) + minForHealth;


        return randomDegreeHealth;
    }
    private boolean isAdmin(Member member) {
        return member.hasPermission(Permission.ADMINISTRATOR);
    }
}
```