Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
Дослідження та програмна реалізація Інтернет-боту для адміністрування глобальних серверів в Discord

Виконав здобувач вищої освіти
II курсу, групи КІ-21М 1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна
інженерія»
_____Сільніцький В.І.
«____» _____ 20 р.

Керівник проекту
кандидат технічних наук, доцент
_____Минайленко Р.М.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *магістр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123 "Комп'ютерна інженерія"*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
«___»_____2022 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Сільніцькому Валерію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація Інтернет-боту для адміністрування глобальних серверів в Discord*

2. Керівник роботи *Минайленко Роман Миколайович, кандидат техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №19-13 від 17.08.2022 року

3. Строк подання роботи до захисту *23.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмна реалізація Інтернет-боту для адміністрування глобальних серверів в Discord*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуші*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2022	12.11.2022
Охорона праці	Оришака О.В., к.т.н., доцент	04.11.2022	20.11.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
«__»_____2022 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____2022 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Сільніцький В.І. Дослідження та програмна реалізація Інтернет-боту для адміністрування глобальних серверів в Discord. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для системи для адміністрування глобальних серверів в Discord.

Метою розробки є дослідження та програмна реалізація системи адміністрування глобальних серверів в Discord.

Об'єктом дослідження є процес адміністрування глобальних серверів в Discord.

Предметом дослідження є методи адміністрування глобальних серверів в Discord.

Методи дослідження базуються на методах математичної статистики, методах розробки програмного забезпечення, методах ООП, методах роботи з БД.

Результат роботи – програмна реалізація системи адміністрування глобальних серверів в Discord.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено на мові програмування Python та в середовищі розробки PyCharm.

Ключові слова: комп'ютерна інженерія, бот для адміністрування, глобальні сервера Discord

ABSTRACT

Silnitskyi V.I. Research and software implementation of an Internet bot for the administration of global servers in Discord. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2022

In this master's thesis, software designed for the system administration global servers in Discord.

The purpose of the development is the research and program implementation of the system administration of global servers in Discord.

The object of the study is the process administration of global servers in Discord.

The subject of the study is the methods administration of global servers in Discord.

Research methods are based on mathematical statistics methods, software development methods, OOP methods, and database work methods.

The result of the work is the software implementation of the system administration of global servers in Discord.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on a PC with Windows XP / Vista / 7/8/10.

The program was developed in the Python programming language and in the PyCharm development environment.

Keywords: computer engineering, bot of administration, global Discord servers

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД – база даних

ПЗ – програмне забезпечення

ООП – об'єктоорієнтоване програмування

ВСТУП

Актуальність теми. Сьогодні Discord далеко не додаток для геймерів, туди заходять великі компанії та освітні програми.

Спільнота Сервер Discord ніби велика квартира, в якій можна зайти в різні кімнати, послухати інших людей і взяти участь у дискусіях. Люди знайомляться та знаходять спільні теми в каналі певної компанії, що додає респекту бренду. Наприклад у каналі Reddit присвяченому крипті є «Трибуна», на яку приходять спікери та обговорюють різні теми, нагадує колись популярний ClubHouse, але тут є великий плюс, крім аудіо контенту користувачі можуть спілкуватися в чаті та гортати стрічку новин, що перетворює Discord на повноцінну соціальну мережу. Окрема тема – це амбассадорські програми, багато користувачів Discord за ексклюзивні стікери та підпис до ніку готові підтримувати вашу компанію в соціальних мережах, допомагати новим користувачам та стати учасниками beta-версій. Цим охоче користуються FinTech проекти, які дають шанс «долучитися» до звичайного користувача до великого і сучасного продукту.

Популярність з кожним днем користувачів та серверів стає все більше, великі компанії приходять і створюють свої ком'юніті. Зайти в Discord ще не пізно, майданчик тільки набирає популярності у масового користувача, і коли як не зараз час розвивати свої сервери.

В розроблюваній кваліфікаційній роботі стоїть за мету створити програмний засіб, який дозволить автоматизувати певні процеси адміністрування Discord серверів.

Із розвитком інформаційних технологій, і появою можливості проведення дистанційного спілкування між людьми, виникла і необхідність в створенні таких засобів. Від найпримітивніших на самому початку додатків для обміну повідомленнями, розвиток технологій прийшов до додатків які дозволяють проводити голосове спілкування, надаючи при цьому широкий функціонал

користувачам. Одним з таких прикладів можна назвати дуже популярний додаток Skype, яким користувалась без применшення більша кількість людей.

В свою чергу навіть системи для голосового спілкування поділяються на системи з різним призначенням. Тоді як Skype в основному використовується для просто голосового спілкування, такі додатки як Slack, Teamspeak, мали своє призначення.

Усі перелічені додатки мають свої плюси і мінуси, проте жоден з них не надає можливості для доступу великої кількості людей на одному сервері, а такий додаток як Skype і взагалі не надає доступу до створення великих серверів.

Для вирішення цієї проблеми, розробниками Станіславом Вишневським та Джейсоном Цитроном було створено додаток для спілкування Discord. Необхідність в додатку була в тому, що використання уже наявних рішень було занадто важким, оскільки така система як Skype занадто навантажувала комп'ютер, а Slack та Teamspeak використовували погані голосові кодеки, що в свою чергу приводило до проблем із комунікацією.

І хоч при розробці вони не націлювались на якусь певну аудиторію, програма широко розійшлась в ігрових спільнотах, а з часом поширилась на весь світ завдяки своєму функціоналу.

Діскорд дозволяє будь якому користувачу, абсолютно безкоштовно створити свій приватний сервер, що в свою чергу відкриває майже безмежні цілі в яких додаток може використовуватись.

В період останніх років, додаток також став популярним як в робочих організаціях, так і в багатьох навчальних закладах, в основному через те що велика кількість таких закладів перейшли на дистанційний тип роботи, і Discord став альтернативою таким додаткам як Zoom та Google Meet, пропонуючи більш широкий функціонал і безкоштовний доступ до майже всіх своїх функцій.

Розглядаючи навчальну сферу, програму Discord можна використовувати для проведення лабораторних робіт, оскільки викладач може завантажувати роботи напряму в додаток, та створювати чати для обговорення, відповідей на запитання, або обліку виконаних завдань.

А також для проведення лекцій. Для забезпечення голосового супроводження лекцій, викладач має можливість створити голосовий канал на сервері, та запросити усіх студентів напряму, або за допомогою посилання. Також в системі є можливість проведення демонстрації екрану, що дозволяє викладачу детально пояснити матеріал, а також підтримка відео-спілкування, для забезпечення присутності студентів на навчанні.

В робочих сферах програма використовується часто в тих самих цілях що і в навчальних, тому відмінності в призначенні мінімальні, проте можливо налаштувати сервер в залежності від рівнів доступу працівників. Таким чином, можна обмежити доступ до каналів з фінансовою інформацією для тих, хто не працює з нею.

Розроблювана кваліфікаційна робота це інформаційна систем автоматизації адміністрування таких серверів. Оскільки популярність програми Discord тільки зростає, а деякі публічні сервери налічують кілька тисяч учасників, повстало питання автоматизації певних процесів адміністрування.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи адміністрування глобальних серверів в Discord.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем адміністрування глобальних серверів в Discord.

– Дослідження системи адміністрування глобальних серверів в Discord.

– Програмна реалізація системи адміністрування глобальних серверів в Discord.

Об’єктом дослідження є процес адміністрування глобальних серверів в Discord.

Предметом дослідження є методи адміністрування глобальних серверів в Discord.

Методи дослідження базуються на методах математичної статистики, методах розробки програмного забезпечення, методах ООП, методах роботи з БД.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод адміністрування глобальних серверів в Discord, який відрізняється від існуючих аналогів, таких як MEE6, Minerea, PancakeBot, має широкі функції по адмініструванню серверу та програванню музики, має логування своїх дій для полегшення адміністрування серверу.

– Розроблено вітчизняний продукт адміністрування глобальних серверів в Discord, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені підпрограми та алгоритми дають легко і добре справлятися с задачами адміністрування глобальних серверів в Discord.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи адміністрування глобальних серверів в Discord, є актуальною задачею, яка потребує вирішення у даній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Інформаційна система призначена для автоматизованого керування Discord сервера, а також для надання певних можливостей адміністраторам, таких як:

– Блокування користувачів.
– Розблокування користувачів.
– Заборона писати/говорити користувачам.
– Включення та виключення музики.
– Логування своїх дій.
– Збір статистики.

1.2 Область застосування

Розроблювана інформаційна система не функціонує як окреме рішення, а працює сумісно з додатком для спілкування та комунікацій Discord.

Discord – одна з великої серії програм, призначених для вирішення питання спілкування та комунікації. Це безкоштовна платформа, яка поєднує доступний інтерфейс чату в таких програмах, як Slack, із відео- та голосовим чатом у стилі Skype, швидко стала однією з найпопулярніших, повідомляючи про 250 мільйонів користувачів і 14 мільйонів людей, які входять у систему щодня. Discord чудово підходить для спілкування з друзями під час гри, але він також корисний для створення місць, де люди можуть збиратися, зустрічатися, знаходити інших гравців і спілкуватися.

Discord – це програма для чату, схожа на такі програми, як Skype, TeamSpeak, і професійні комунікаційні платформи, як-от Slack. Він спеціально призначений для гравців у відеоігри, надаючи їм способи знаходити один

одного, координувати гру та розмовляти під час гри. Він підтримує відеодзвінки, голосовий чат і текстові повідомлення, що дозволяє користувачам зв'язуватися, як їм заманеться.

Discord особливо корисний, якщо ви намагаєтеся грати в комп'ютерні ігри. Програма робить спілкування в чаті досить легким і пропонує функції пошуку, які допоможуть вам знайти інших людей і додати їх до списку друзів для швидкого спілкування. Багато людей використовують його не лише для спілкування один з одним під час гри, але й як організаційний та соціальний інструмент.

Завдяки такій широкій функціональності користувачі також прийняли Discord як напівпублічну платформу спільноти у стилі форуму. Групи гравців зі спільними інтересами, наприклад шанувальники певної гри чи студії, можуть створювати або приєднуватися до «серверів», як загальнодоступних, так і приватних, де багато людей можуть зустрічатися та проводити час, спілкуватися за допомогою тексту, відео чи голосу.

Хоча більшість серверів пов'язані з іграми, також можливо знайти загальнодоступні сервери Discord, які зосереджені на різних темах, включаючи такі речі, як аніме, криптовалюта, самовдосконалення, розробка програмних засобів, а також просто знайомство та спілкування. Також немає ніяких вимог, щодо створення серверів, тож якщо ви хочете мати місце в Discord для обговорення теми, ви завжди можете його створити.

Відмінність Discord від інших програм

Хоча в Інтернеті є багато безкоштовних програм для спілкування, Discord виділяється завдяки широкому спектру варіантів чату. Він поєднує в собі всі найкращі функції більш поширених програм, таких як Skype і Slack, із простим у використанні інтерфейсом. Програма голосового чату не принесе користі, якщо вона сповільнює ваші ігри під час її використання, тому команда, яка створює Discord, прагне зробити її максимально ефективною.

Ця універсальність спонукала великі групи користувачів прийняти Discord як місце для зустрічей і спілкування з людьми, які мають схожі інтереси,

а не лише з друзями. Це частково додаток для спілкування, частково портал соціальних мереж. У той час як сторона чату програми, у якій користувачі можуть приєднуватися до публічних або приватних «серверів», є, мабуть, найпопулярнішою, вона також надає соціальний форум, який чудово допомагає людям грати в ігри. По суті, вам не потрібні відеоігри, щоб зробити Discord корисним - це надзвичайно зручно для приєднання до груп друзів на приватному сервері або зустрічей з однодумцями на загальнодоступних.

Коли створюється сервер Discord, його можна налаштувати багатьма способами, налаштувавши різні ролі для учасників, наприклад створивши ролі для інших адміністраторів і модераторів або створивши спеціальні ролі для ваших найактивніших учасників. Налаштування різних ролей учасників має ряд переваг для вашого сервера, зокрема полегшує керування сервером, винагороджує учасників за активність і надає різні дозволи доступу.

Discord також доступний кількома способами, що спрощує використання, навіть якщо ви не сидите перед ігровим ПК. Додаток має програму для ПК, яку можна завантажувати, яку можна запустити на комп’ютері — зручну, полегшену версію, яка найкраще працює у фоновому режимі під час гри, — а також веб-версію, мобільну версію та бета-версію на консолі. Це означає, що ви можете взаємодіяти з людьми на своїх серверах чату Discord практично будь-де, розширюючи соціальні можливості програми.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення для адміністрування глобальних серверів в Discord, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній випускній роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

Для розробки інформаційної системи одним із головних етапів є виконання аналізу існуючих рішень, та виокремлення їх плюсів та мінусів.

Для аналізу було вибрані такі системи як MEE6, PancakeBot, Minerea.

MEE6

MEE6 – це бот, який дозволяє вам автоматизувати такі завдання, як надсилання привітальних повідомлень. Ви також можете створювати власні команди, щоб виганяти користувачів, які публікують рекламу, посилання та спам.

Але його найкраща функція – можливість призначати ролі користувачам і надсилати повідомлення в поточному каналі або DM.

Існує навіть конкурентоспроможна система підвищення рівня, оскільки учасники можуть боротися, щоб потрапити на вершину таблиці лідерів або отримати настроювані картки рангів, щоб продемонструвати свою лояльність.

Також можливо тимчасово або назавжди заблокувати користувачів на основі встановлених вами параметрів. Наприклад, будь-який користувач, який отримав три порушення, може отримати тайм-аут на 30 хвилин.

Що стосується ботів для модерації, MEE6 є одним із найкращих.

PancakeBot

Pancake – це бот для серверу Discord, який надає численні функції, як-от додавання будь-якої пісні до музичного каналу, додавання списків відтворення з різних платформ, зокрема YouTube і Spotify. Він був розроблений ShyPianoGuy і Fren. Зараз Pancake став настільки популярним, що до нього підключено понад 850 тисяч серверів.

Крім того, цей бот також має деякі фантастичні параметри, як і будь-який інший музичний плеєр, включаючи відтворення, паузу, чергу, наступне, перемішування, повернення тощо.

Minerea

Minerea – це музикальний бот для серверу Discord, який надає функціонал для відтворення музики в режимі реального часу в голосових каналах Discord. Він має можливість відтворення “радіостанцій” різного жанру, що виділяє його серед решти. Що до інших функцій, таких як можливість програвання музики в по добовому режимі, вона дозволена лише для преміум користувачів.

Розроблювана під час кваліфікаційної роботи інформаційна система вміщує в собі як і функціонал з розглянутих відомих аналогів, так і абсолютно новий, що дозволяє виокремити себе серед інших. Оскільки головною метою є розроблення системи адміністрування, є можливість ведення статистики користувачів сервера, яка не надається напряму, можливість вести облік дій адміністрації та автоматизувати обмеження використання певних конструкцій слів, які будуть визначені персоналом адміністрації.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Оскільки розроблюване програмне забезпечення працює не самостійно, а в парі з іншим додатком, який надає доступ до свого функціоналу, вибір мови програмування залежить також від можливостей взаємодії з додатком Discord.

Вибір програмного забезпечення визначається тими функціями, які будуть реалізовані в інформаційній системі. Одним із головних етапів є вибір мови програмування на якій буде розроблена система, вибір мови програмування залежить від кількох чинників:

1. Тип розроблюваного додатку.
2. Важкість розроблюваного додатку.
3. Підтримка після випуску продукту.

4. Масштабованість та продуктивність.

5. Безпека.

Проте наскільки б хорошим вибір мови програмування не був, навіть на найкращій мові можна написати як програму яка буде функціонувати погано, так і програму яка буде функціонувати добре.

Для більш продуктивного процесу розробки, також і потрібні інші засоби окрім самої мови програмування, а саме середовища розробки, середовища проектування та керування базами даних, та інші.

В залежності від вибору середовища розробки, процес виконання роботи може як і значно пришвидшитись, так і сповільнитись, оскільки різні середовища надають різні переваги під час виконання роботи, і можуть значно спростити процес.

Для будування інформаційної системи потрібно визначити її структуру, необхідні функції та мету розробки. Реалізація системи виконуваної в кваліфікаційній роботі потребує наступних засобів:

- Вибір мови програмування.
- Середовище розробки яке надає найкращі умови для створення інформаційної системи.
- Сервер бази даних та система керування, яка буде використовуватись у розробці.
- Додаткові бібліотеки-розширення, які спрощують розробку функціоналу інформаційної системи.
- Сервер на якому буде розміщуватись система, оскільки передбачається що вона повинна працювати в режимі без зупинок.

Оскільки інформаційна система повинна працювати без простою, потрібно щоб вона розміщувалася на вибраному сервері, який забезпечує роботу цілодобово. Багато компаній надають доступ до серверів за плату в місяць, що дозволяє розміщати свої сайти/проекти. В якості компанії було вибрано Namecheap, які надають доступ до віртуального хостингу, де можливо

розмістити інформаційну систему реалізовану як на Javascript, так і на Python, які є головними варіантами для вибору мови програмування реалізації системи.

Для розробки інформаційної системи адміністрування Discord серверів, було проаналізовано такі мови програмування, як Javascript та Python.

Javascript

JavaScript – це легка мова програмування, яку веб-розробники зазвичай використовують для створення більш динамічної взаємодії під час розробки веб-сторінок, програм, серверів і навіть ігор.

Розробники зазвичай використовують JavaScript разом із HTML і CSS. Мова скриптів добре працює з CSS у форматуванні елементів HTML. Однак JavaScript все ще підтримує взаємодію з користувачем, чого CSS не може зробити сам по собі.

Застосування JavaScript в Інтернеті, мобільних додатках та розробці ігор робить цю мову програмування вартою вивчення. Це можна за допомогою навчальних платформ, як-от BitDegree, або досліджуючи безкоштовні шаблони та програми JavaScript на платформах для розміщення коду, як-от GitHub.

Початкові версії мови скриптів були лише для внутрішнього використання. Після того, як Netscape подав його в ECMA International як стандартну специфікацію для веб-браузерів, JavaScript був піонером у випуску ECMAScript.

Це була мова скриптів загального призначення для забезпечення взаємодії веб-сторінок у різних браузерах і пристроях.

Відтоді JavaScript продовжував розвиватися разом із новими браузерами, такими як Mozilla Firefox і Google Chrome. Останній навіть почав розробляти перший сучасний двигун JavaScript під назвою V8, який компілює байт-код у рідний машинний код.

Сьогодні JavaScript має багато фреймворків і бібліотек для спрощення складних проектів, таких як AngularJS, jQuery та ReactJS.

Спочатку реалізація JavaScript працювала на стороні клієнта, але після розробки Node.js була розгалужена на стороні сервера – міжплатформного серверного середовища, створеного на механізмі Google Chrome JavaScript V8.

Хоча він найбільше підходить для веб-програм, функції програмування JavaScript мають інші реалізації в різних областях.

Python

Проте, не дивлячись на всі переваги та недоліки Javascript, для розробки інформаційної системи було обрано мову програмування Python.

Python - це інтерпретована об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Його високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та динамічним зв’язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання як мови скриптів або з’єднувальної мови для з’єднання існуючих компонентів. Простий, легкий для вивчення синтаксис Python підкреслює читабельність і, отже, знижує вартість обслуговування програми. Python підтримує модулі та пакети, що заохочує модульність програми та повторне використання коду. Інтерпретатор Python і обширна стандартна бібліотека доступні у вихідному або двійковому вигляді безкоштовно для всіх основних платформ і можуть вільно поширюватися.

Python виділяється серед інших мов програмування через підвищену продуктивність, яку він забезпечує. Оскільки етапу компіляції немає, цикл редагування-тестування-налагодження відбувається неймовірно швидко. Налагоджувати програми на Python легко: помилка чи неправильний вхід ніколи не спричинить помилку сегментації. Натомість, коли інтерпретатор виявляє помилку, він викликає виняток. Якщо програма не вловлює виняток, інтерпретатор друкує трасування стека. Налагоджувач рівня вихідного коду дозволяє перевіряти локальні та глобальні змінні, оцінювати довільні вирази, встановлювати контрольні точки, покроково виконувати код по рядках і так далі. Сам налагоджувач написаний на Python, що свідчить про інтроспективну силу Python. З іншого боку, часто найшвидшим способом налагодження програми є

додавання кількох операторів друку до джерела: швидкий цикл редагування-тестування-налагодження робить цей простий підхід дуже ефективним.

- Python — це високорівнева, інтерпретована, інтерактивна та об'єктно-орієнтована мова сценаріїв. Python створений для високої читабельності. Він часто використовує англійські ключові слова там, де інші мови використовують знаки пунктуації, і має менше синтаксичних конструкцій, ніж інші мови.

- Python обробляється під час виконання інтерпретатором. Вам не потрібно компілювати програму перед її виконанням. Це схоже на PERL і PHP.

- Python є інтерактивним – ви можете сидіти за підказкою Python і взаємодіяти з інтерпретатором безпосередньо, щоб написати свої програми.

- Python підтримує об'єктно-орієнтований стиль або техніку програмування, яка інкапсулює код всередині об'єктів.

- Python це чудова мова для програмістів початківців, яка підтримує розробку широкого спектру додатків від простої обробки тексту до браузерів WWW та ігор.

Python підтримує програми графічного інтерфейсу користувача, які можна створювати та переносити на багато системних викликів, бібліотек і систем Windows, таких як Windows MFC, Macintosh і система X Window Unix. Не дивлячись на всі плюси, мова має свої недоліки:

- Високорівневість мови змушує розбиратися з нюансами програмування, роботою з пам'яттю, процесами та іншими низькорівними концепціями

- ООП в мові Python не на 100% відповідає усім започаткованим принципам

- Динамічна типізація — через динамічну типізацію Python споживає більше ресурсів, ніж міг би, але це часто компенсується внутрішнім кешуванням.

- Global Interpriter Lock. На даний момент це є основною проблемою продуктивності в Python, а також цим обумовлена погана реалізація багатопоточності. Код GIL не змінювався з першої версії мови, що вказує на те, що він застарів.

Перш ніж вибрати середовище розробки, яке буде використовуватися для проекту, потрібно ретельно вирішити, які інструменти та функції повинні мати середовище. Оскільки не всі середовища розробки однакові, деякі можуть мати корисне програмне забезпечення для налагодження, а інші можуть бути тривіальними у використанні. Хоча список функцій і властивостей, наведений нижче, не є повним (існує багато інших визначальних моментів, які може мати середовище розробки), це мінімальні критерії за якими слід розглядати та вибирати середовище:

- Вартість.
- Швидкість.
- Налагодження.
- Простота використання.
- Додатки.
- Бібліотеки.
- Надійність.

Оскільки Python дуже популярна мова розробки, для неї вийшла велика кількість середовищ розробки, одне з яких і було вибрано для виконання кваліфікаційної роботи – Pycharm.

PyCharm – це гібридна платформа, розроблена JetBrains як IDE для Python. Він зазвичай використовується для розробки додатків Python. Деякі з організацій єдинорога, такі як Twitter, Facebook, Amazon і Pinterest, використовують PyCharm як своє середовище розробки Python.

Він підтримує дві версії: v2.x і v3.x.

PyCharm можна запустити як на Windows, так і на Linux або Mac OS. Крім того, він містить модулі та пакети, які допомагають програмістам розробляти програмне забезпечення за допомогою Python за менший час і з мінімальними зусиллями. Крім того, його також можна налаштувати відповідно до вимог розробників.

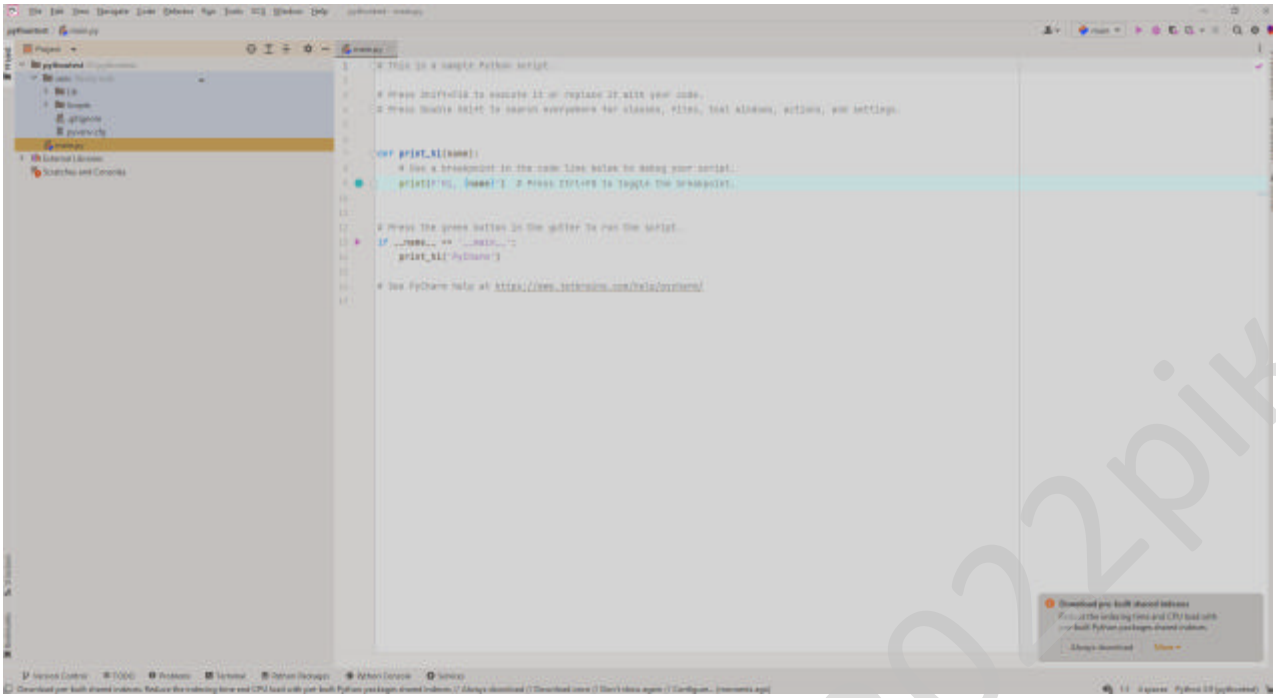

Рисунок 2.1 - Зовнішній вигляд середовища розробки PyCharm

Одним із великих плюсів PyCharm є те, що для нього існує велика кількість модулів, які підключаються до проекту за допомогою однієї стрічки коду, дозволяючи урізноманітнити функціонал розроблюваної системи.

Також для студентів, компанія – розробник середовища Pycharm надає безкоштовний доступ до преміум версії, що відкриває додаткові функції для користувача, такі як: синхронна розробка коду з іншими користувачами, більш широкий доступ до функціоналу середовища розробки, інтеграція з платними модулями PyCharm.

Сервери баз даних використовуються для зберігання та керування базами даних, які зберігаються на сервері, а також для надання доступу до даних авторизованим користувачам. Цей тип сервера зберігає дані в центральному місці, для якого можна регулярно створювати резервні копії. Це також дозволяє користувачам і програмам централізовано отримувати доступ до даних у мережі. Велику кількість баз даних, які використовуються у вашій організації, можна зберігати на одному сервері або групі серверів, які спеціально налаштовані для захисту даних і запитів клієнтів послуг.

Майстер налаштування вашого сервера не містить настроюваної ролі для серверів баз даних. Сервер бази даних – це будь-який сервер, який запускає програму мережевої бази даних і підтримує файли бази даних, наприклад Microsoft SQL Server або Oracle. SQL Server - це високопродуктивна система керування базами даних. Він використовується для зберігання та аналізу даних і надає користувачам можливість швидкого доступу до величезних обсягів даних через мережу. Оскільки SQL Server забезпечує додаткові заходи безпеки, які інакше були б недоступні (як обговорюється в розділі «Захист серверів баз даних» далі в цьому розділі), а обробка відбувається на сервері, транзакції можуть відбуватися безпечно та швидко.

Для роботи над інформаційною системою було вибрано сервер бази даних MySQL.

MySQL – це система керування реляційною базою даних з відкритим кодом. Як і в інших реляційних базах даних, MySQL зберігає дані в таблицях, що складаються з рядків і стовпців. Користувачі можуть визначати, маніпулювати, контролювати та запитувати дані за допомогою мови структурованих запитів, більш відомої як SQL.

Гнучка та потужна програма MySQL є найпопулярнішою системою баз даних з відкритим кодом у світі. Будучи частиною широко використовуваного стеку технології LAMP (який складається з операційної системи на базі Linux, веб-сервера Apache, бази даних MySQL і PHP для обробки), він використовується для зберігання та отримання даних у широкому спектрі популярних програм , веб-сайти та служби.

Для роботи з MySQL було вибрано PHPMyAdmin.

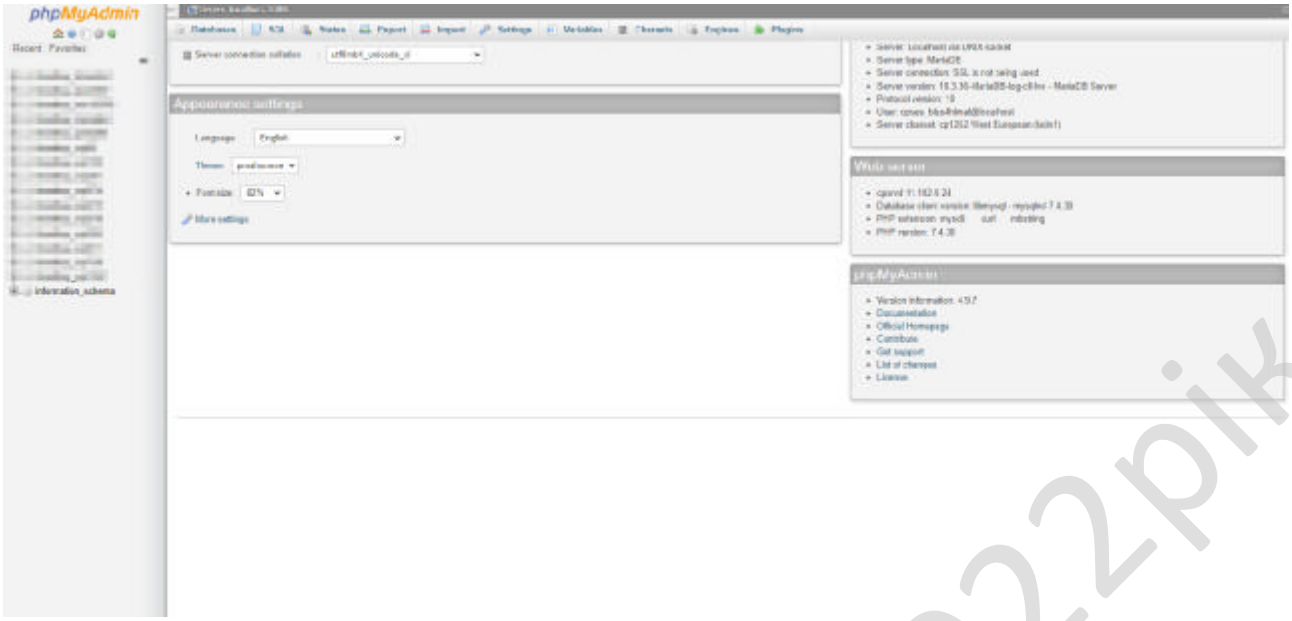

Рисунок 2.2 - Зовнішній вигляд середовище phpmyadmin

phpMyAdmin – це програмний інструмент із відкритим вихідним кодом, представлений 9 вересня 1998 року, написаний на PHP. По суті, це інструмент третьої сторони для керування таблицями та даними в базі даних. phpMyAdmin підтримує різні типи операцій у MariaDB і MySQL. Основна мета phpMyAdmin - це адміністрування MySQL через Інтернет.

Це найпопулярніша програма для керування базами даних MySQL. За допомогою цього програмного забезпечення ми можемо створювати, оновлювати, видаляти, змінювати, видаляти, імпортувати та експортувати таблиці бази даних MySQL. phpMyAdmin також підтримує широкий спектр операцій, як-от керування базами даних, зв'язками, таблицями, стовпцями, індексами, дозволами та користувачами тощо в MySQL і MariaDB. Ці операції можна виконувати через інтерфейс користувача, але ми все ще маємо можливість виконати будь-який оператор SQL.

phpMyAdmin перекладено на 72 мови, а також підтримує мови RTL і LTR, щоб широке коло людей могло легко використовувати це програмне забезпечення. Ми можемо виконувати запити MySQL, ремонтувати, оптимізувати, перевіряти таблиці, а також виконувати інші команди керування

базами даних. phpMyAdmin також можна використовувати для виконання адміністративних завдань, таких як створення бази даних, виконання запитів.

phpMyAdmin — це програма на основі графічного інтерфейсу користувача, яка використовується для керування базою даних MySQL. Ми можемо вручну створити базу даних і таблицю та виконати до них запит. Він забезпечує веб-інтерфейс і може працювати на будь-якому сервері. Оскільки він є веб-сайтом, ми можемо отримати доступ до нього з будь-якого комп'ютера.

Переваги phpMyAdmin:

- phpMyAdmin може працювати на будь-якому сервері чи будь-якій ОС, оскільки він має веб-браузер.

- Ми можемо легко створювати, видаляти та редагувати базу даних і можемо керувати всіма елементами за допомогою графічного інтерфейсу phpMyAdmin, який набагато легший, ніж редактор командного рядка MySQL.

- phpMyAdmin допомагає нам контролювати дозволи користувача та керувати кількома серверами одночасно.

- Ми також можемо створювати резервні копії нашої бази даних і експортувати дані в різні формати, такі як XML, CSV, SQL, PDF, OpenDocument Text, Excel, Word, електронні таблиці тощо.

- Ми можемо виконувати складні інструкції та запити SQL, створювати та редагувати функції, тригери та події за допомогою графічного інтерфейсу phpMyAdmin.

Сервери – це потужні комп'ютери, створені для зберігання, обробки та керування мережевими даними, пристроями та системами. Сервери — це двигуни, які забезпечують організації мережевими пристроями та системами відповідними ресурсами. Для компаній сервери пропонують важливі можливості масштабованості, ефективності та безперервності бізнесу.

Незалежно від того, чи йдеться про розміщення веб-сайту з великою кількістю даних, налаштування спільного диска для відділу чи керування тисячами запитів щохвилини, сервери є транспортними засобами для розміщення

та обробки інтенсивних робочих навантажень, які виходять за межі можливостей традиційного комп’ютера.

Сервери можуть робити все, що може робити стандартний настільний комп’ютер, і багато іншого. І навпаки, комп’ютери можуть запускати серверні процеси, але роблять це набагато менш продуктивно. Загалом сервери пропонують такі функції мережам, які вони обслуговують:

- Масштабованість для обслуговування зростаючої або коливаючої кількості пристроїв, користувачів і робочих навантажень.

- Висока обчислювальна потужність із зростаючими характеристиками процесора та оперативної пам’яті для роботи з мережевими навантаженнями.

- Надійність, яка гарантує, що критичні системи залишаються онлайн і доступними.

- Економія з часом, оскільки сервери можуть зменшити навантаження на мережеві пристрої.

Для розроблюваної інформаційної системи було вибрано сервер спільного хостингу.

Це служба веб-хостингу, де багато віртуальних серверів, розміщуються на одному веб-сервері, підключеному до Інтернету. Безумовно, це найпростіший спосіб розміщення, оскільки загальна вартість обслуговування сервера розподілена між багатьма клієнтами.

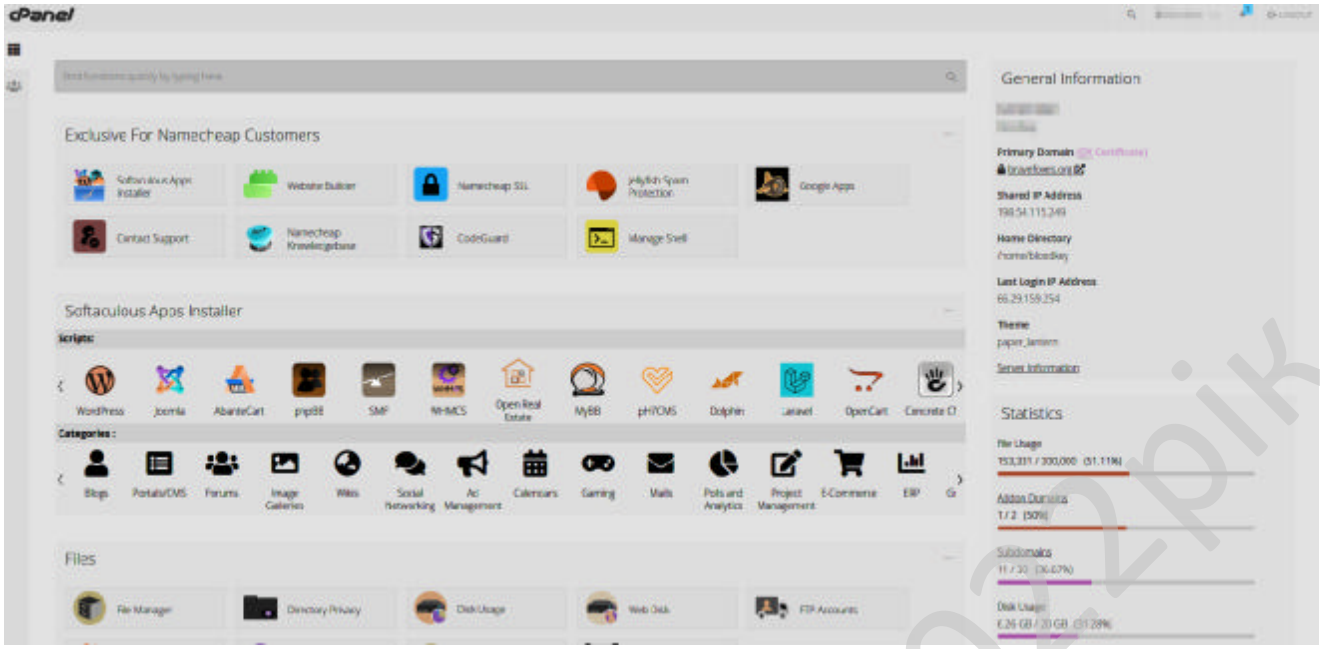

Рисунок 2.3 - Головна контрольна панель серверу

Вибраний сервер працює на операційній системі CloudLinux.

CloudLinux – це орієнтований на хостинг дистрибутив Linux на основі CentOS. Він використовує технологію ядра LVE (полегшене віртуальне середовище), яка в деяких аспектах схожа на OpenVZ або на інші технології віртуалізації на основі ОС.

У спільному хостингу найпоширенішою причиною простою є один обліковий запис, який забирає більшість спільних ресурсів через важкі сайти або неоптимізовані скрипти. ОС CloudLinux розроблена спеціально для серверів, які використовуються таким чином, що унеможливлює ситуацію, коли один або кілька користувачів сильно навантажують або навіть зависають сервер.

CloudLinux покращує стабільність сервера, інкапсулюючи кожного клієнта в безпечне ізольоване середовище. Це означає, що кожен користувач на сервері обмежений кількістю ресурсів, які він може використовувати, будучи розміщеним у так званому «контейнері» (відокремлене операційне середовище).

Наступний рисунок демонструє роботу традиційного серверного середовища та CloudLinux:

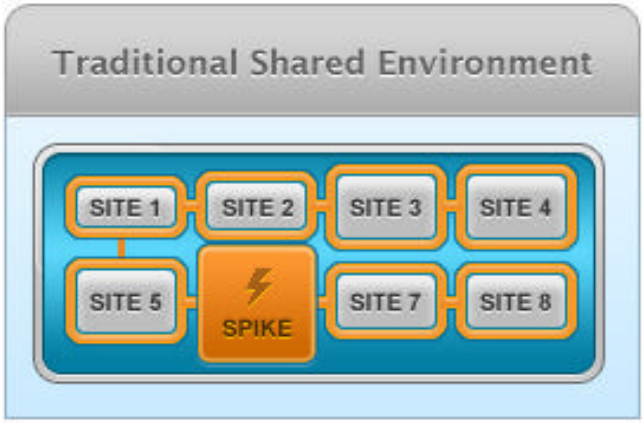

Рисунок 2.4 - Робота серверів CloudLinux

Якщо говорити про специфіку апаратного забезпечення на вибраному сервері, використовується обладнання Dell або Supermicro, і всі сервери мають найновіші процесори Intel Xeon, багато оперативної пам’яті та швидкі SSD-диски.

Технічні характеристики серверу:

- Dell R430
- Подвійний процесор Intel(R) Xeon(R) E5-2660 v4 @ 2,00 ГГц / подвійний процесор Intel Xeon Gold 6140 2,3 ГБ
- Оперативна пам'ять 256 Гб
- RAID6 SSD
- Усі сервери працюють під керуванням операційної системи CloudLinux 6.x із Apache, MySQL, PHP, Perl тощо.

Оскільки Discord сервери використовуються людьми для різних цілей, однією з яких може бути, наприклад, організація робочого місця для розробки, в системі планується реалізувати таку можливість як підключення до Github, що дозволить відображати зміни на репозиторії шляхом виведення сповіщень в відповідний канал на сервері.

Подібний функціонал реалізується за допомогою “Вебхуків”.

Вебхук – це функція зворотного виклику на основі HTTP, яка забезпечує легкий, керований подіями зв’язок між 2 інтерфейсами прикладного програмування (API). Веб-хуки використовуються багатьма веб-програмами для

отримання невеликих обсягів даних з інших програм, але веб-хуки також можна використовувати для запуску автоматизованих робочих процесів у середовищах GitOps.

Веб-хуки для розробки додатків

API – це набір визначень і протоколів для створення та інтеграції прикладного програмного забезпечення. Зв'язок між API іноді називають контрактом між користувачем інформації та постачальником інформації — встановлення вмісту, який вимагається від споживача (виклик), і вміст, який вимагається виробником (відповідь). Цей зв'язок також описується як клієнтська програма, яка викликає серверну програму, але ці ролі можна змінити залежно від того, яка програма запитує дані в певній ситуації.

Веб-інтерфейси API зазвичай використовують HTTP для запиту даних від інших програм і визначення структури повідомлень-відповідей, які зазвичай мають форму файлу XML або JSON. І XML, і JSON є кращими форматами, оскільки вони представляють дані у спосіб, яким легко маніпулювати іншими програмами.

Коли API клієнта запитує дані від API сервера, він викликає, щоб перевірити, чи сталася певна подія, іншими словами, чи змінилися дані сервера таким чином, що може бути корисним для клієнта. У цьому процесі (відомому як опитування) клієнт надсилає запит HTTP через регулярні проміжки часу, доки API сервера не надішле відповідні дані, які іноді називають корисним навантаженням.

Клієнтська програма не знає про стан серверної програми, тому вона опитує API сервера для оновлення, викликаючи знову і знову, доки не відбудеться певна подія, але сервер надішле запитувані дані лише тоді, коли ця інформація стане доступною. Клієнтська програма має продовжувати запитувати оновлення та чекати, доки відбудеться відповідна подія.

Метою багатьох інформаційних систем є перетворення даних в інформацію, щоб генерувати знання, які можна використовувати для прийняття рішень. Для цього система повинна мати можливість отримувати дані, поміщати

їх у контекст і надавати інструменти для агрегації та аналізу. База даних створена саме для такої мети.

База даних – це організований набір пов’язаної інформації. Це організована колекція, оскільки в базі даних усі дані описані та пов’язані з іншими даними. Уся інформація в базі даних також має бути пов’язаною; слід створити окремі бази даних для керування непов’язаною інформацією. Наприклад, база даних, яка містить інформацію про студентів, не повинна також містити інформацію про ціни акцій компанії. Бази даних не завжди є цифровими – наприклад, картотеку можна вважати формою бази даних. Для цілей цього тексту ми розглядатимемо лише цифрові бази даних.

Реляційні бази даних

Бази даних можуть бути організовані багатьма різними способами і, таким чином, приймати різні форми. Найбільш популярною формою бази даних сьогодні є реляційна база даних. Популярними прикладами реляційних баз даних є Microsoft Access, MySQL і Oracle. Реляційна база даних — це база даних, у якій дані організовано в одну або декілька таблиць. Кожна таблиця має набір полів, які визначають характер даних, що зберігаються в таблиці. Запис — це один екземпляр набору полів у таблиці. Щоб уявити це, подумайте про записи як про рядки таблиці, а про поля – як про стовпці таблиці. У наведеному нижче прикладі ми маємо таблицю інформації про студента, де кожен рядок представляє студента, а кожен стовпець – одну інформацію про студента.

У реляційній базі даних усі таблиці пов’язані одним або декількома полями, тому можна з’єднати всі таблиці в базі даних через спільні для них поля. Для кожної таблиці одне з полів визначається як первинний ключ. Цей ключ є унікальним ідентифікатором для кожного запису в таблиці. Щоб допомогти вам краще зрозуміти ці терміни, давайте розглянемо процес розробки бази даних.

Нереляційні бази даних

Незважаючи на те, що реляційні бази даних є найбільш розповсюдженими в наш час, і її модель використовують найбільше при розробці баз даних та імплементації в інформаційні системи, існує також багато інших моделей

сховища даних, які мають свої переваги та недоліки перед реляційними системами. База даних, яка об’єднує принципи батьківського/дочірнього зв’язку між даними, називається ієрархічною моделлю даних, і була популярна в 1960-х і1970-х роках.

Документ орієнтована модель даних дозволяє забезпечити неструктуроване зберігання, шляхом маніпулювання інформацією, розміщеною у так званих “документах”

Хоч і описані попередньо моделі даних використовуються при розробці певних систем, все ж вони не настільки популярні як реляційні, або нереляційні бази даних.

Нереляційна, або ж NoSQL база даних це вже не нова концепція, і свою назву вона заслужила тим що є прямою протилежністю реляційних баз даних. NoSQL системи було розроблено для вирішення проблеми використання масштабних баз даних, які розташовані на декількох серверах або навіть по всьому світу. Для правильного функціонування реляційних баз даних, важливо, щоб над маніпулювання фрагментами даних працювала лише одна людина, концепцією, відомою як блокування записів. З сьогоднішніми масштабами даних це просто неможливо. NoSQL бази даних можуть працювати з даними більш вільно, оскільки дозволяють більш неструктурованому середовищі, передавати зміни даних з часом на всі сервери, які є частиною бази даних.

Реляційні бази даних погано масштабуються, що є недоліком перед NoSQL базами даних. Термін масштаб тут відноситься до бази даних, яка стає все більшою і більшою, розподіляючись на великій кількості комп’ютерів або серверів, підключених через мережу.

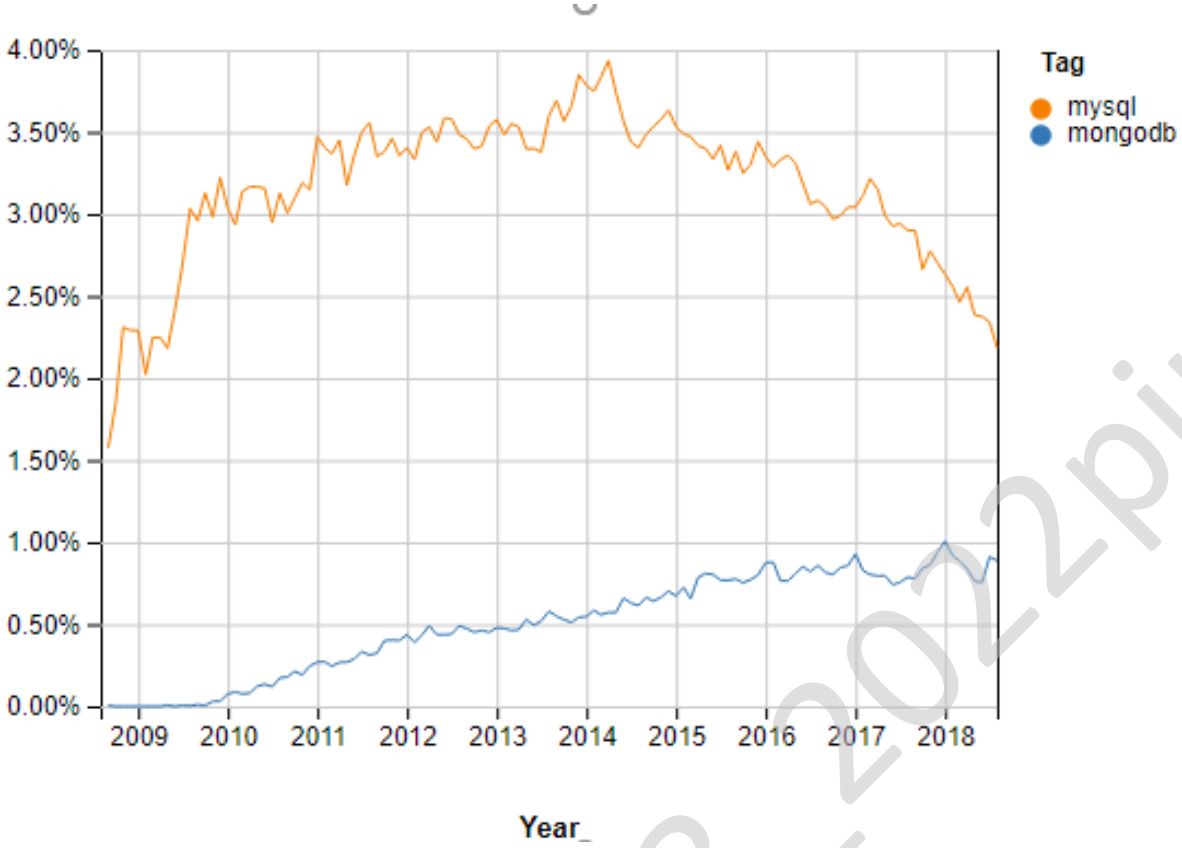

Рисунок 2.5 - Популярність реляційних/нереляційних баз даних

З цих графіків видно що mysql починає втрачати свою популярність, таким чином нереляційні БД з кожним роком стають популярніше і актуальніше, але ще не достатньо.

MongoDB – яскравий приклад грамотної реалізації NoSQL-системи управління базами даних. Програмне забезпечення розповсюджується відповідно до ліцензії SSPL – це невільна Source Available ліцензія зі значними обмеженнями.

Таблиця 2.1 - Головні відмінності SQL та NoSQL баз даних

	SQL	NoSQL
Визначення	Бази даних SQL переважно називаються RDBMS або реляційними базами даних	Бази даних NoSQL в основному називаються нереляційними або розподіленими базами даних
Розроблені для	Традиційна RDBMS використовує синтаксис і запити SQL для аналізу та отримання даних для подальшого аналізу. Вони використовуються для систем OLAP.	Система баз даних NoSQL складається з різних технологій баз даних. Ці бази даних були розроблені у відповідь на вимоги, пред'явлені для розробки сучасної програми.
Мова запитів	Мова структурованих запитів (SQL)	Немає декларативної мови запитів
Тип	Бази даних SQL є табличними базами даних	Бази даних NoSQL можуть базуватися на документах, парах ключ-значення, графічних базах даних
Приклади	Oracle, Postgres і MS-SQL.	MongoDB , Redis, Neo4j, Cassandra, Hbase.
Найкраще підходить для	Ідеальний вибір для середовища інтенсивного виконання складних запитів.	Погано підходить для складних запитів.
Ієрархічне зберігання даних	Бази даних SQL не підходять для ієрархічного зберігання даних.	Більш підходить для ієрархічного сховища даних, оскільки підтримує метод пари ключ-значення.
Відкритий доступ	Поєднання відкритого коду, як Postgres і MySQL, і комерційного, як Oracle Database.	Відкрите джерело
Тип зберігання	Високодоступне сховище (SAN, RAID тощо)	Звичайні накопичувачі (стандартні HDD, JBOD)

Продовження таблиці 2.1

	SQL	NoSQL
Найкращі характеристики	Підтримка між платформами, безпечно та безкоштовно	Простий у використанні, високопродуктивний і гнучкий інструмент.
Модель	ACID (Atomicity, Consistency, Isolation, and Durability) є стандартом для RDBMS	Base (Basically Available, Soft state, Eventually Consistent) — це модель багатьох систем NoSQL

Нормалізація баз даних

На відміну від NoSQL, для правильного функціонування реляційних баз даних, інформація яка зберігається повинна підлягати процесу нормалізування. Під час проектування бази даних важливо розуміти цю концепцію нормалізації. Простими словами, нормалізувати базу даних означає спроектувати її таким чином, щоб:

1) зменшити дублювання даних між таблицями;

2) надати таблиці якомога більше гнучкості.

Хостинг, у найзагальнішому розумінні, — це послуга, за допомогою якої особі чи організації надаються ресурси для зберігання та обчислювальні ресурси для розміщення та обслуговування одного або кількох веб-сайтів і відповідних послуг. Хоча хостинг не обов'язково базується на IP-адресі, переважна більшість випадків – це веб-сервіси, які дозволяють веб-сайту або веб-сервісу бути глобально доступними з Інтернету.

Хостинг також відомий як веб-хостинг.

Як надзвичайно важлива послуга, хостинг сприяв розвитку та зростанню Інтернету. Хостинг в основному надається постачальником послуг хостингу, який створює спеціалізовану серверну обчислювальну інфраструктуру. У свою чергу власник/розробник веб-сайту використовує інфраструктуру для розміщення свого веб-сайту через завантажений вихідний код, де кожен веб-сайт відрізняється своїм унікальним доменним іменем і логічно розподіленим веб-

простором і сховищем. Після вказівки доменного імені у веб-браузері здійснюється доступ до веб-сайту через Інтернет.

З розвитком технологій і моделей доставки хостинг перетворився на різноманітні формати, включаючи спільний хостинг, виділений хостинг і хмарний хостинг. Окрім веб-сайтів, хостинг також може включати хостинг даних/сховищ, хостинг програм/програмного забезпечення та хостинг IT-послуг. Межа також стирається з хмарними обчисленнями та віртуалізацією, які дозволяють інший рівень складності та налаштування.

Віртуальний хостинг – це тип веб-хостингу, де на одному фізичному сервері розміщено кілька сайтів. Багато користувачів використовують ресурси на одному сервері, що дозволяє знизити витрати. Кожен користувач отримує розділ сервера, на якому вони можуть розміщувати файли свого веб-сайту. Спільні сервери можуть приймати сотні користувачів. Кожен клієнт, який користується сервером спільної платформи хостингу, має доступ до таких функцій, як бази даних, щомісячний трафік, дисковий простір, облікові записи електронної пошти, облікові записи FTP та інші додаткові функції, які пропонує хост. Клієнти на сервері спільно використовують системні ресурси за вимогою, і кожен отримує певний відсоток від оперативної пам’яті, центрального процесора та інших елементів, таких як єдиний сервер MySQL, сервер Apache і поштовий сервер.

Спільний хостинг пропонує найбільш економічно ефективний спосіб розмістити сайт в Інтернеті, оскільки витрати на підтримку сервера розподіляються між усіма користувачами. Цей стиль хостингу найкраще підходить для невеликого веб-сайту чи блогу, який не потребує розширених конфігурацій або високої пропускної здатності. Оскільки спільного хостингу недостатньо для сайтів із високим трафіком, сайти з великою кількістю відвідувачів повинні шукати VPS або виділені рішення для хостингу.

Переваги спільного веб-хостингу

Вибір спільного хостингу має багато переваг. Давайте розглянемо основні особливості спільного веб-хостингу:

1) Ціна

Віртуальний хостинг – це найбільш економічно ефективне рішення для розміщення. Оскільки багато людей беруть участь у витратах на сервер, витрати хостингової компанії розподіляються між ними. Базові плани починаються приблизно з 30 доларів на рік, тоді як ви можете розраховувати платити понад 100 доларів на рік за плани преміум-класу з необмеженим дисковим простором, необмеженою пропускною здатністю та необмеженою кількістю веб-сайтів.

2) Гнучкість

Нові онлайн-підприємства можуть починатися зі спільного плану та без проблем оновлюватись у міру зростання сайту.

3) Легкий в керуванні

Віртуальний хостинг потребує відносно низького обслуговування. Поставник хостингу подбає про базові адміністративні завдання сервера. Якщо ви не готові запускати власний сервер, веб-керування є найзручнішим варіантом. Залиште професіоналам турбуватися про ваш веб-хостинг. Завдяки спільному хостингу ви можете розраховувати на професійну технічну допомогу щодо будь-чого: від оновлення обладнання та обслуговування, оновлення програмного забезпечення, DDoS-атак, збоїв у мережі тощо.

Головною альтернативою і наступним кроком при масштабуванні системи, є вибір VPS серверів. VPS надає доступ до власного персонального сервера з виділеною кількістю ресурсів і попередньо встановленою операційною системою на ваш вибір. Один комп’ютер поділяється на кілька VPS. Це має багато переваг для власників веб-сайтів, наприклад наявність виділеного обсягу дискового простору та пропускної здатності. VPS рекомендується для сайтів да додатків, які хочуть розширити масштаб. Якщо потрібно збільшити ресурси для додатку, VPS-хостинг цілком може стати правильним рішенням для вас.

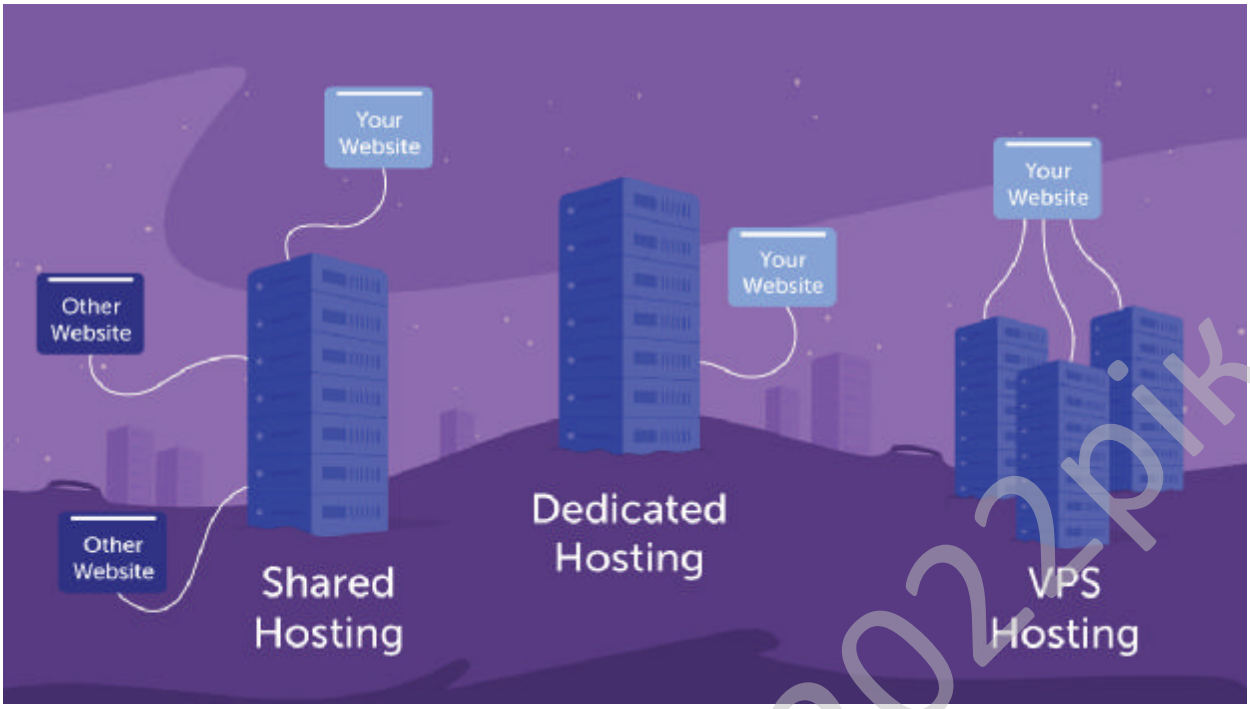

Рисунок 2.6 - Різниця між типами хостингів

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на магістерську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи адміністрування глобальних серверів в Discord.

В процесі розробки магістерської роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації

в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У даній роботі було розроблено Інтернет-бот для адміністрування глобальних серверів Discord. Для початку його розробки потрібно зайти на developers сайт Discord, для створення аплікації зображено на рисунку 3.1.

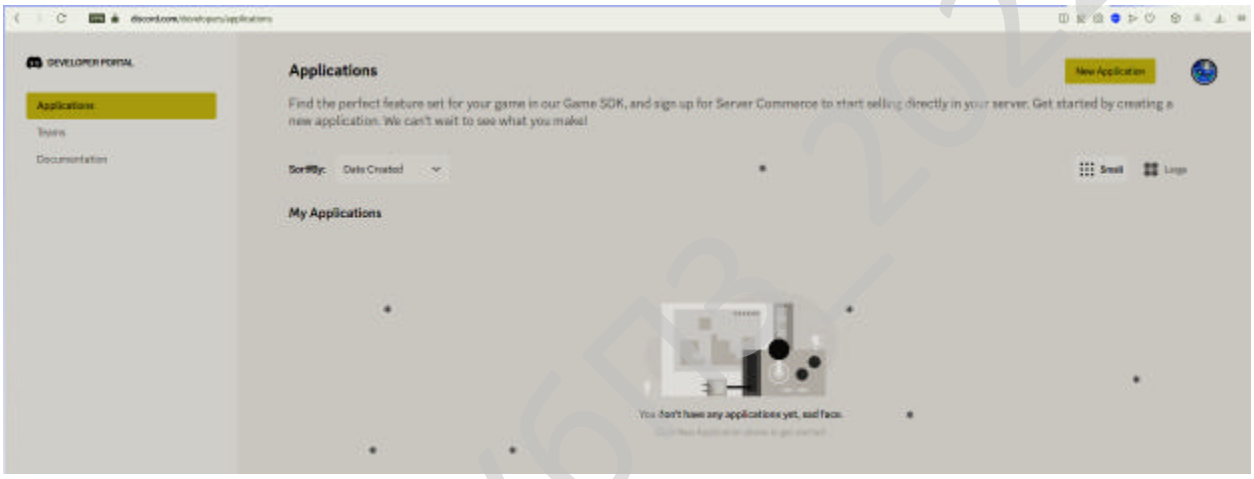

Рисунок 3.1 - Developers сайт Discord

Після цього потрібно створити аплікацію зображено на рисунку 3.2.

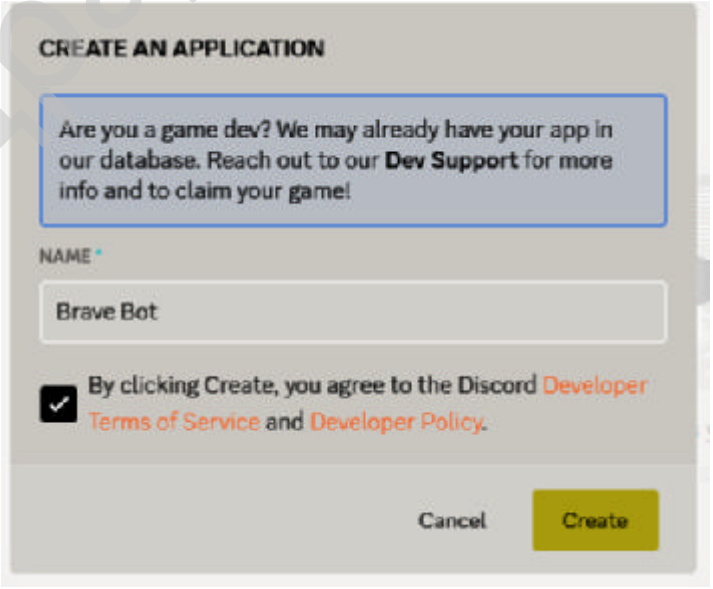

Рисунок 3.2 - Створення аплікації

Для того щоб можна було додати бота та писати для нього код потрібно його створити у аплікації (рисунок 3.3) та дати йому ті дозволи які потрібно для обслуговування серверу, так як у нас бот для адміністрування ми дамо йому функцію адміністратора (рисунок 3.4) та додати його на свій сервер для розробки.

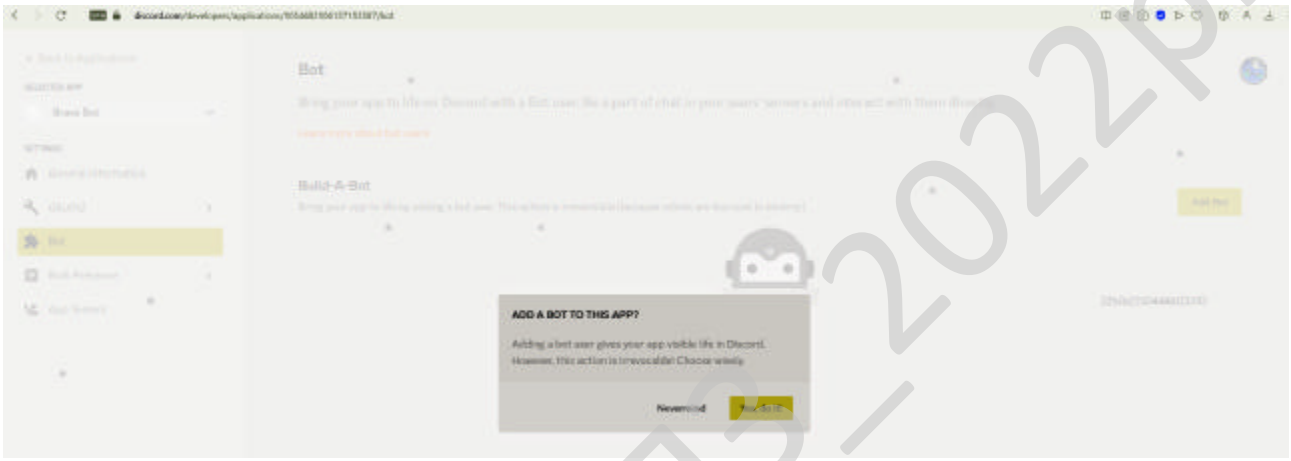

Рисунок 3.3 - Створення бота в аплікації

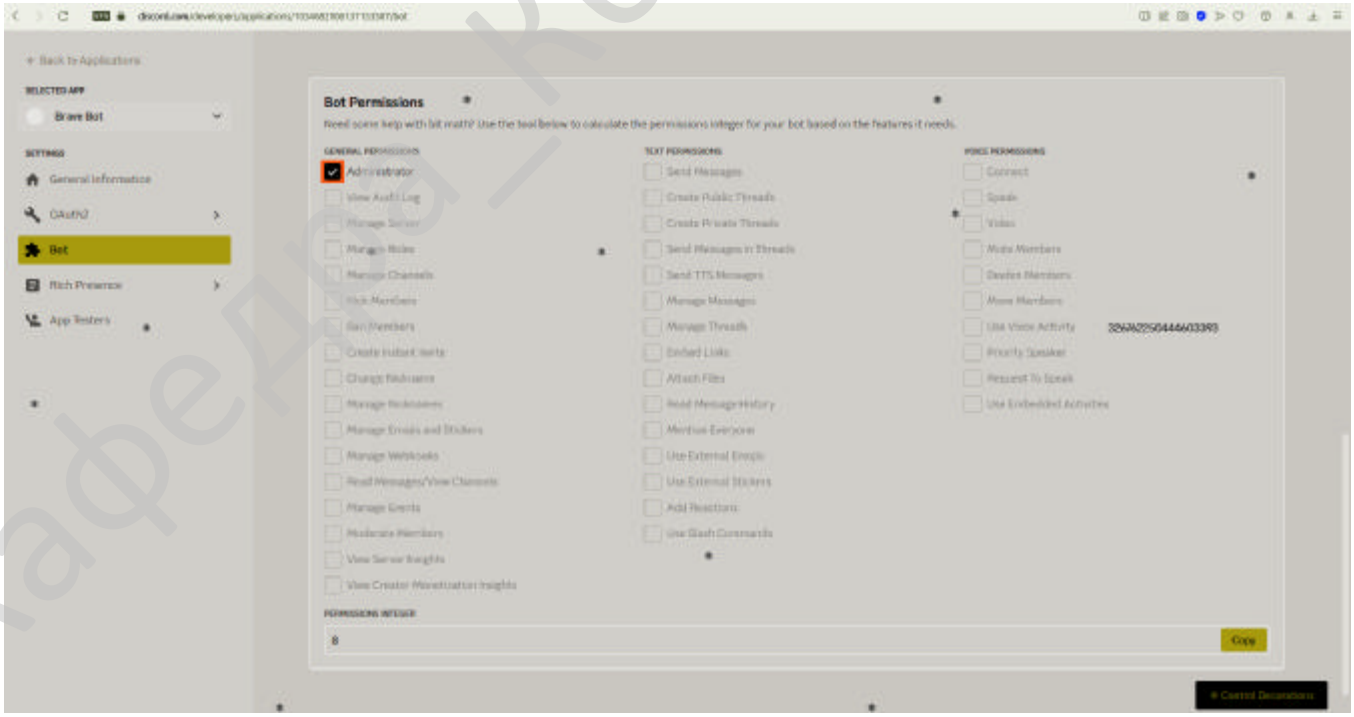

Рисунок 3.4 - Надання прав боту

Після цього потрібно додати бота на сервер за допомогою створення URL показано на рисунку 3.5 та 3.6.

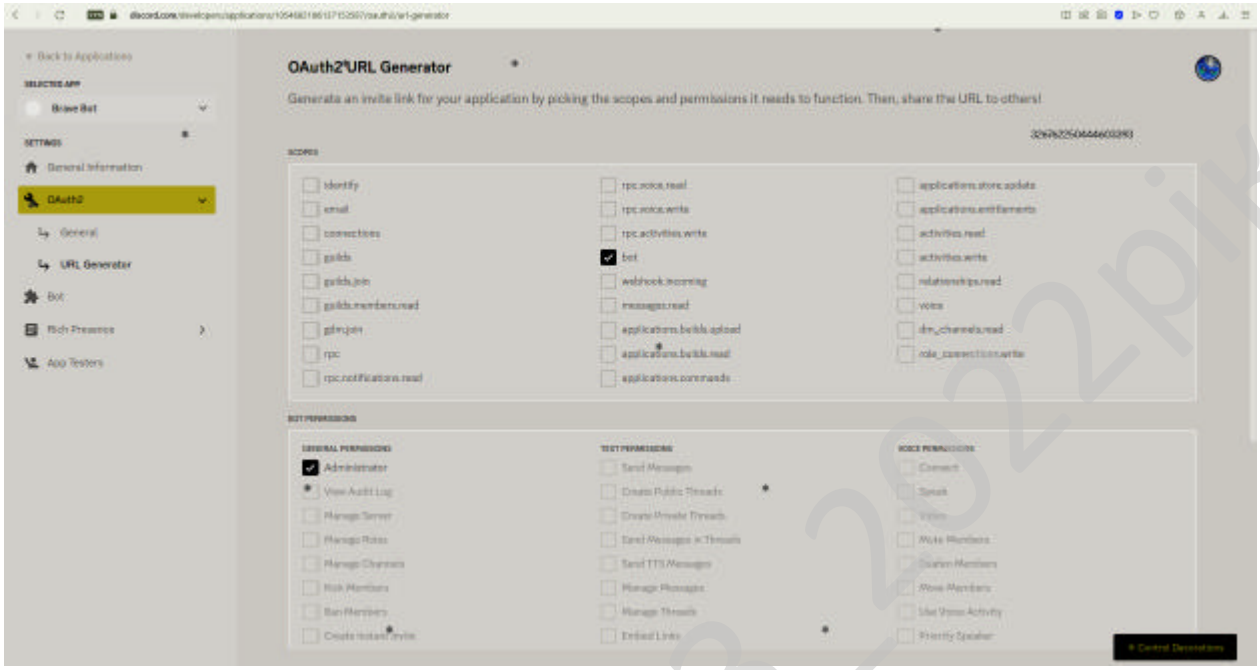

Рисунок 3.5 - Надання прав та створення посилання

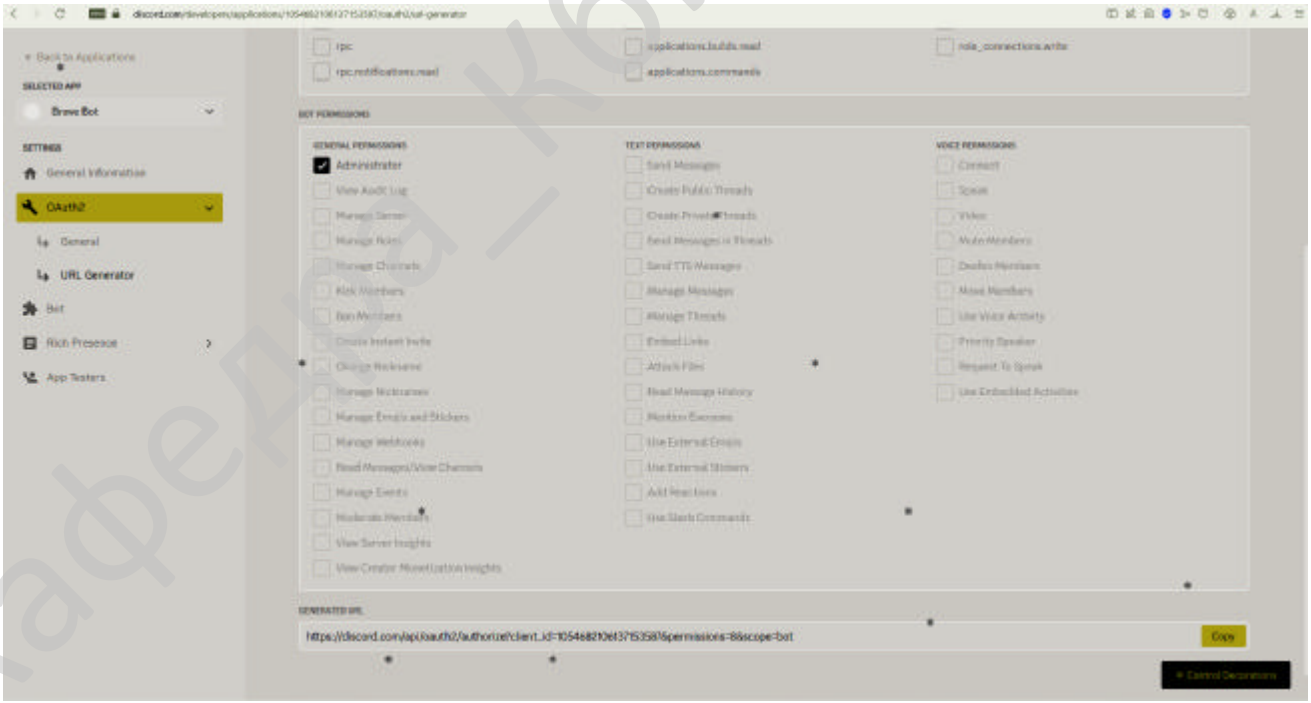

Рисунок 3.6 - Отримання URL посилання на додавання боту

Скопіювавши посилання вставляємо його в браузер та додаємо на сервер який потрібно показано на рисунку 3.7.

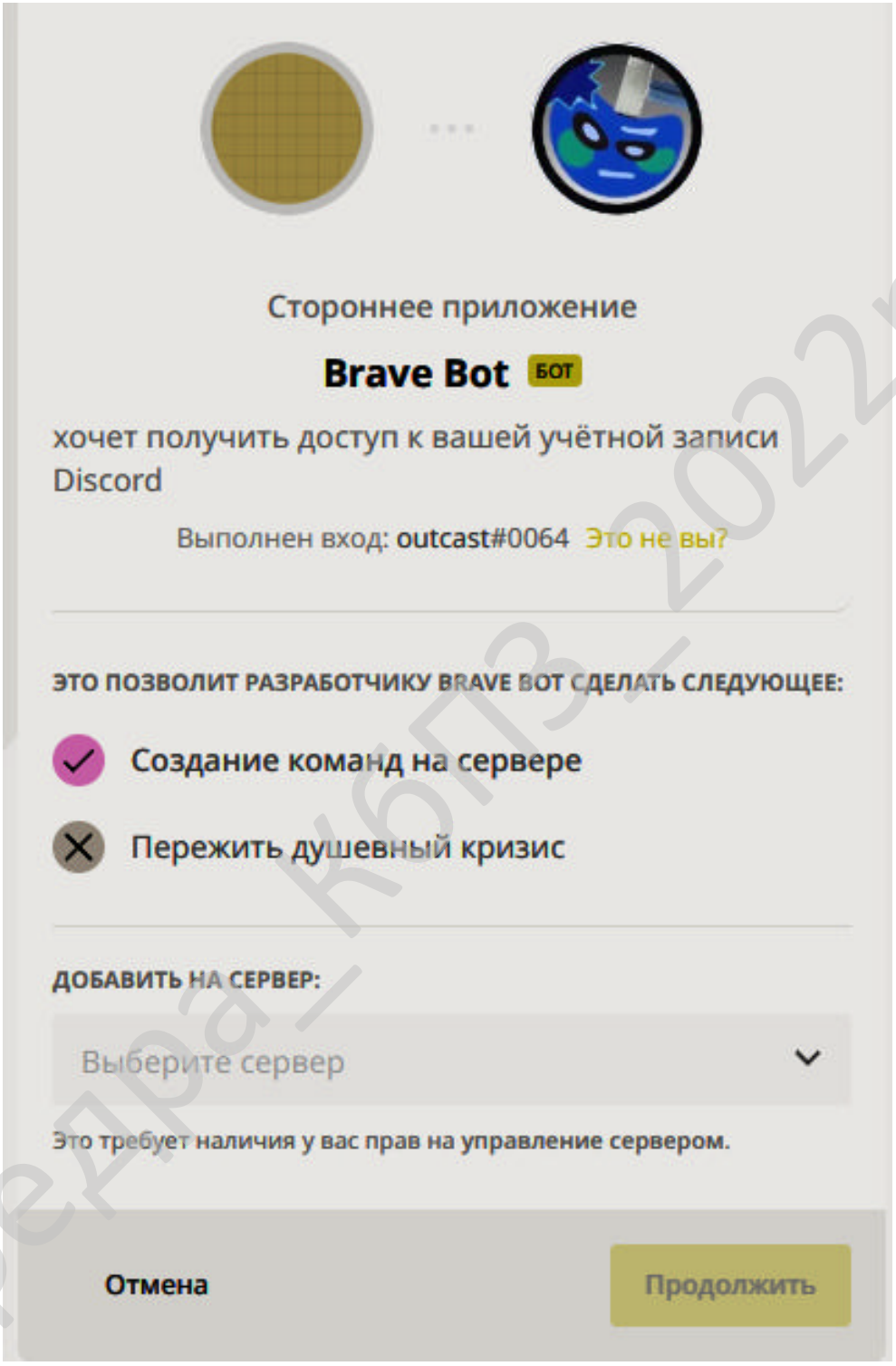

Рисунок 3.7 - Додавання боту на сервер

Для того щоб тепер працювати с ботом потрібно отримати його токен (рисунок 3.8) та додати його підключення за допомогою коду який показано на рисунку 3.9.

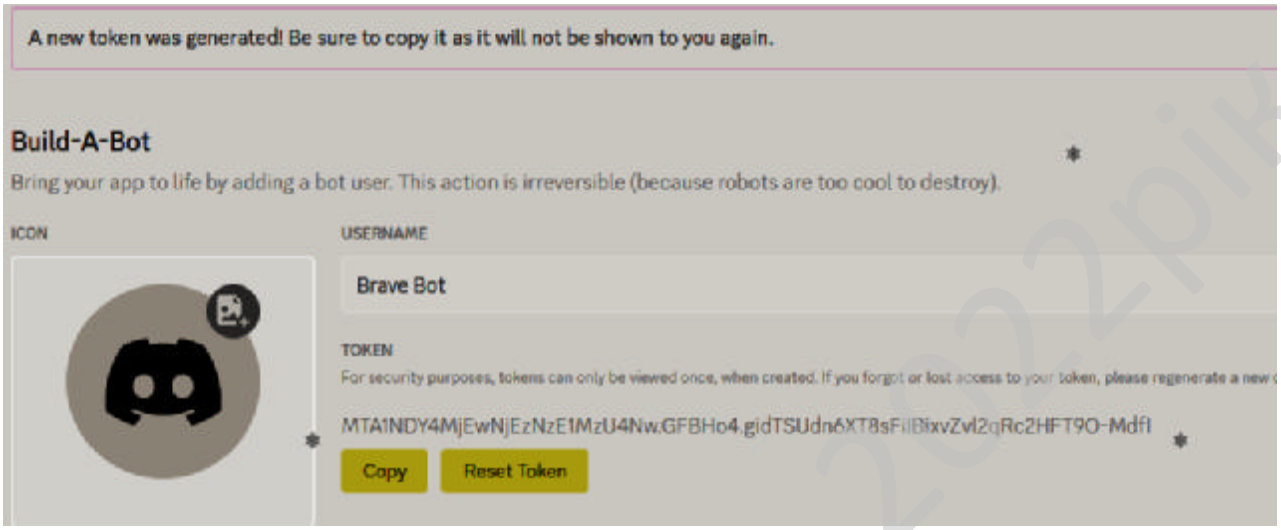

Рисунок 3.8 - Отримання токену бота

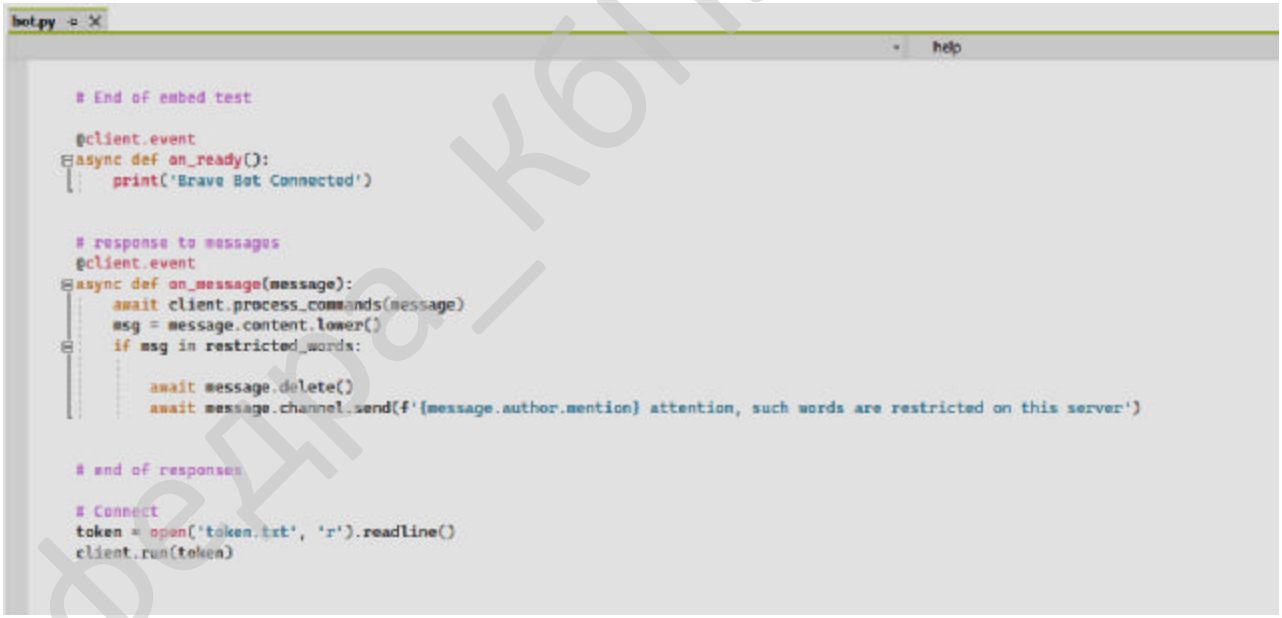

Рисунок 3.9 - Підключення токену за допомогою TXT файла

Підключення зроблено за допомогою зчитування токена з файла для того, якщо програмний код отримає зловмисник то він не отримав доступа до боту.

Під час роботи з великими публічними серверами, процес їх адміністрування та керування може займати багато часу. Оскільки велика

кількість людей користується ресурсом одночасно, керування може займати стільки людських ресурсів, скільки може не бути. Для вирішення подібних проблем були створені автоматизовані інформаційні системи, або ж як їх іноді простіше називають - “боти”, які виконують певну частину функціоналу замість персоналу адміністрації.

Також в якості додаткового функціоналу реалізовано розширення функціоналу серверів, що дозволяє урізноманітнити використання інформаційної системи, а також виділити її серед решти існуючих аналогів.

Кожна функція розроблена в інформаційній системі направлена на зменшення роботи виконуваної адміністративним персоналом серверу, що в свою чергу збільшує швидкість виконання запитів користувачів, або ж адміністрування серверу.

Інформаційна система дозволяє в певних варіантах розширити функціонал додатку Discord, а в деяких впроваджує альтернативу для користувачів та адміністрації, надаючи можливість виконувати дії в більш прискореному варіанті.

В залежності від відправленої команди, актор має доступ до виконання певних дій. Адміністратор має доступ до всіх команд, оскільки у нього абсолютний контроль над сервером:

1) Керування доступами користувачів.

2) Створення нових текстових / голосових каналів.

3) Додавання посилань для сповіщень про початок трансляції.

4) Перегляд логів.

5) Створення карток-завдань.

6) Програвання музики.

7) Перегляд статистики профілів.

Відвідувач або ще по іншому користувач, має доступ до певного функціоналу, такого як:

1) Створення нових тимчасових текстових/голосових каналів.

2) Створення карток-завдань.

3) Програвання музики.

4) Перегляд статистики профілів.

Оскільки основною метою створення інформаційної системи є реалізація функції адміністрування та модерації в публічних та приватних Discord серверах, потрібно визначити які будуть виконувати автоматизоване керування, а також реалізацію додаткового функціоналу для керуючого персоналу.

До функцій адміністрування серверів відноситься наступне:

1) Керування доступом користувачів до серверів, його обмеження як тимчасове так і постійне.

2) Створення записів дій адміністрації, користувачів.

3) Допомога користувачам в створенні голосових та текстових каналів тимчасового призначення.

4) Автоматичне розподілення ролей між користувачами, в залежності від необхідності та рівнів доступу.

5) Статистика профілів користувачів.

Описане – лише головна частина розроблюваного функціоналу, яка відноситься напряму до розроблюваної інформаційної системи. Оскільки інформаційна система розроблюється в форматі “боту”, для розширення функціоналу серверів можливо додавати сторонні бібліотеки, та запрограмувати систему на виконання додаткового функціоналу.

3.2 Розробка структурної схеми

На структурній схемі зображено структуру розроблюваного ПЗ. Такі частини як бот, Discord API та БД. Структура бота складається з:

- Блок команд.
- Блок алгоритму нотифікацій Twitch.
- Блок алгоритму музики.

Частина Discord API складається з таких модулей як:

- Модуль User
- Модуль Client

- Модуль Member
- Модуль Guild
- Модуль Message

Та остання частина БД, яка має додаткову інформацію про користувачів. Користувач за допомогою команд через інтернет може направити свій запит до бота, при обробці цього запита, бот вирішує куди йому потрібно звернутись щоб виконати команду користувача, це може бути як Discord API, до якого отримується доступ через інтернет, так і БД, яка підключена до боту. Сам бот має декілька блоків алгоритмів та команд.

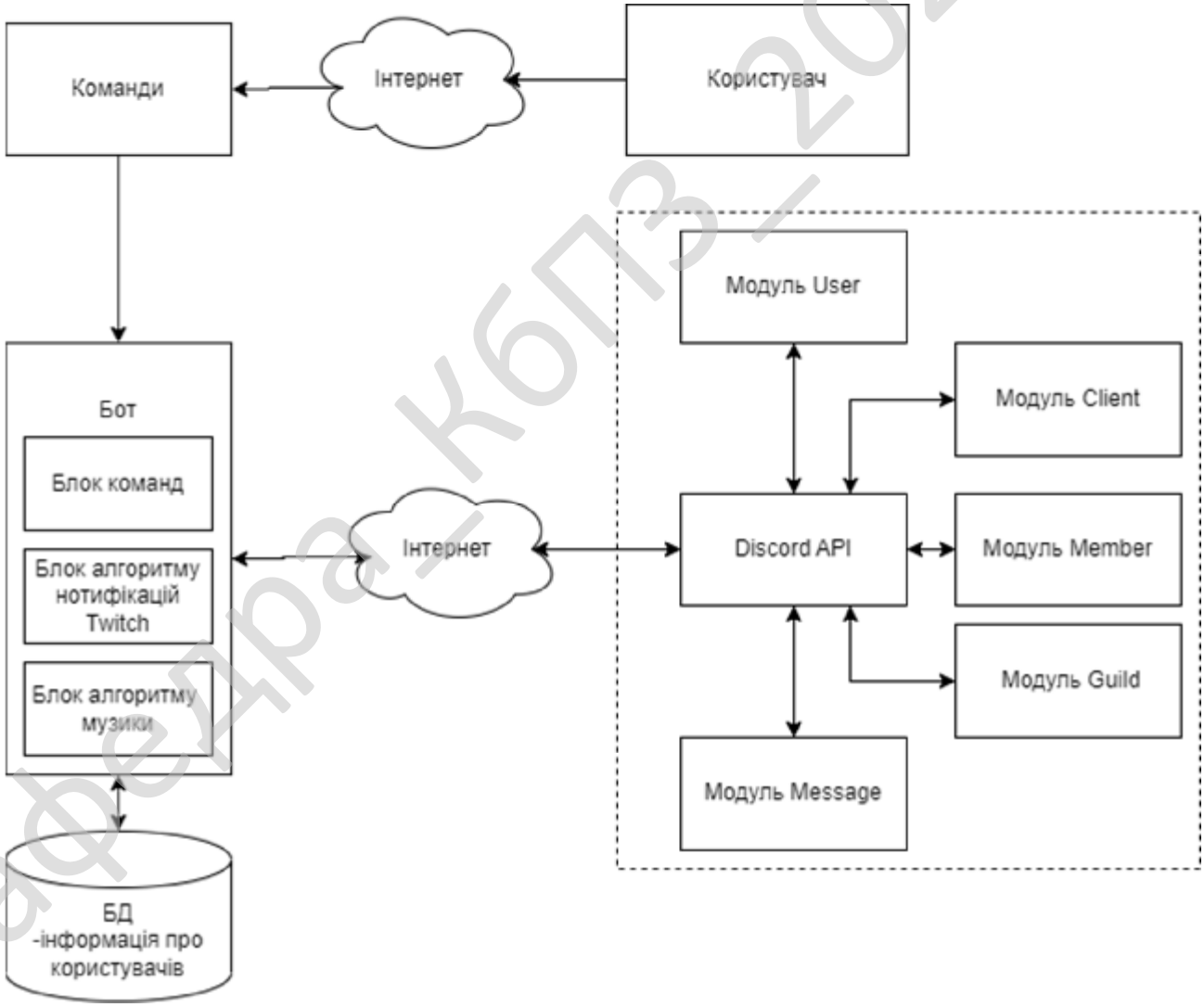

Рисунок 3.10 - Структурна схема системи

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2.

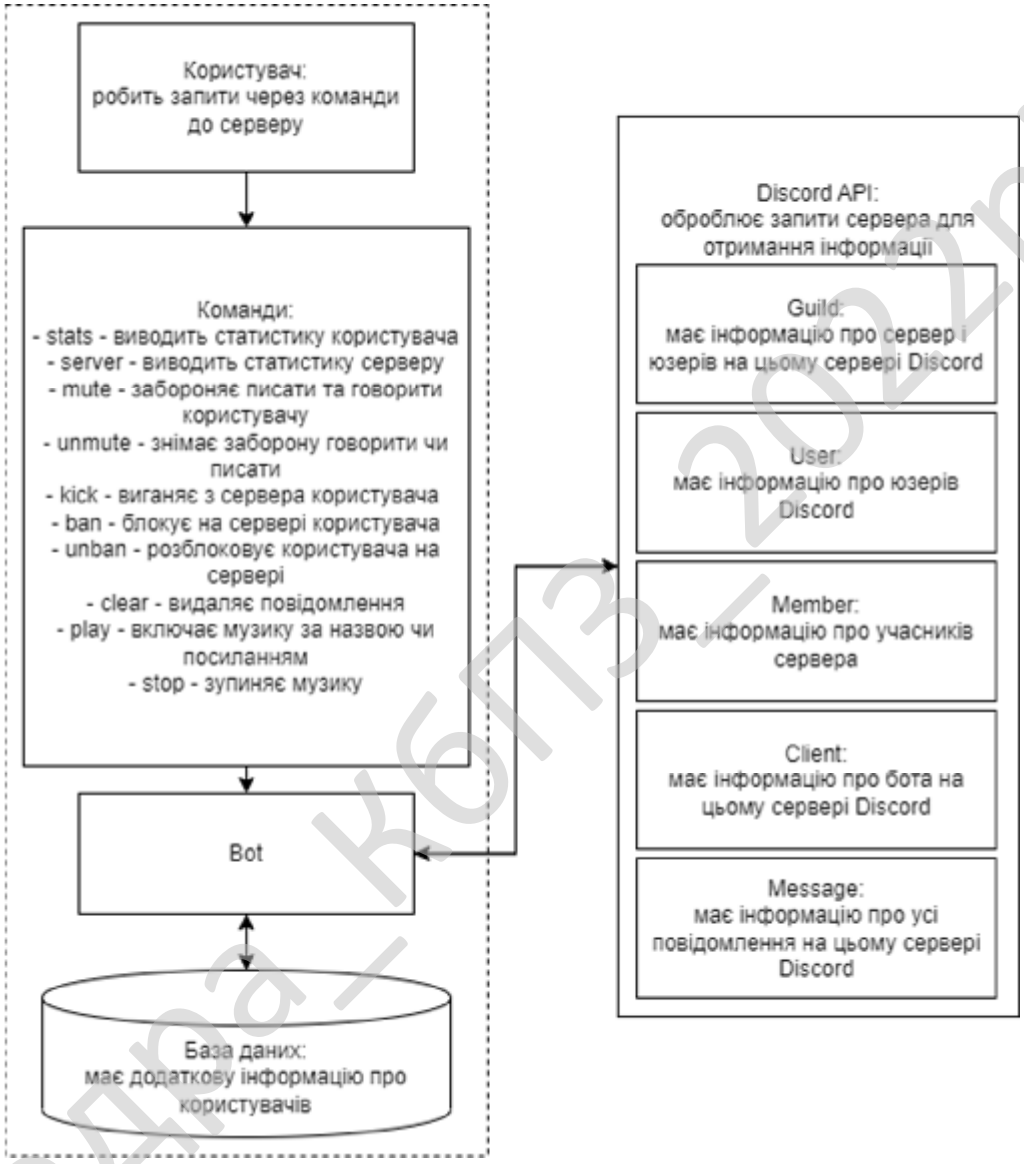

Рисунок 3.11 - Функціональна схема

З рисунку видно, що розроблена система складається з наступних частин:

- **Локальна частина**. Складається з користувача на сервері та його запити з командами до боту

- **Мережева частина**. Складається з Discord API де знаходится уся інформація про сервер та його учасників

Розглянемо більш детально кожний з блоків розробленої програми.

Головний користувач

Має такі атрибути як:

- User_ID – персональний номер користувача;
- Mention – персональний тег користувача в системі Discord.
- Avatar_URL –картинка профілю.
- Guild_ID – номер команди (якщо є).
- UserName – ім’я користувача.

Та виконує функцію:

- sendCommand()

Клас Server:

Має такі атрибути як

- Connect – для підключення до серверу.

Та функції:

- ExecuteCommand() – виконання отриманої від користувача команди.

Клас Bot:

Має такі атрибути як:

- Command – отримана команда;
- Permission – для обробки команди згідно з дозволами користувача.

Також має такі функції як:

- KickUser() – відключення користувача від активного каналу.
- Ban() – перманентне відключення користувача від серверу, відбір усіх привілеїв використання.
- Unban() – надання дозволу відключеному користувачу повернутись на сервер
- Mute() – відбір можливості спілкування в голосових / текстових каналах.
- Unmute() – повернення можливості спілкування в голосових / текстових каналах
- Stats() – виведення статистики користувача по серверу.
- PlayMusic() – відтворення музики в голосових каналах.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3.

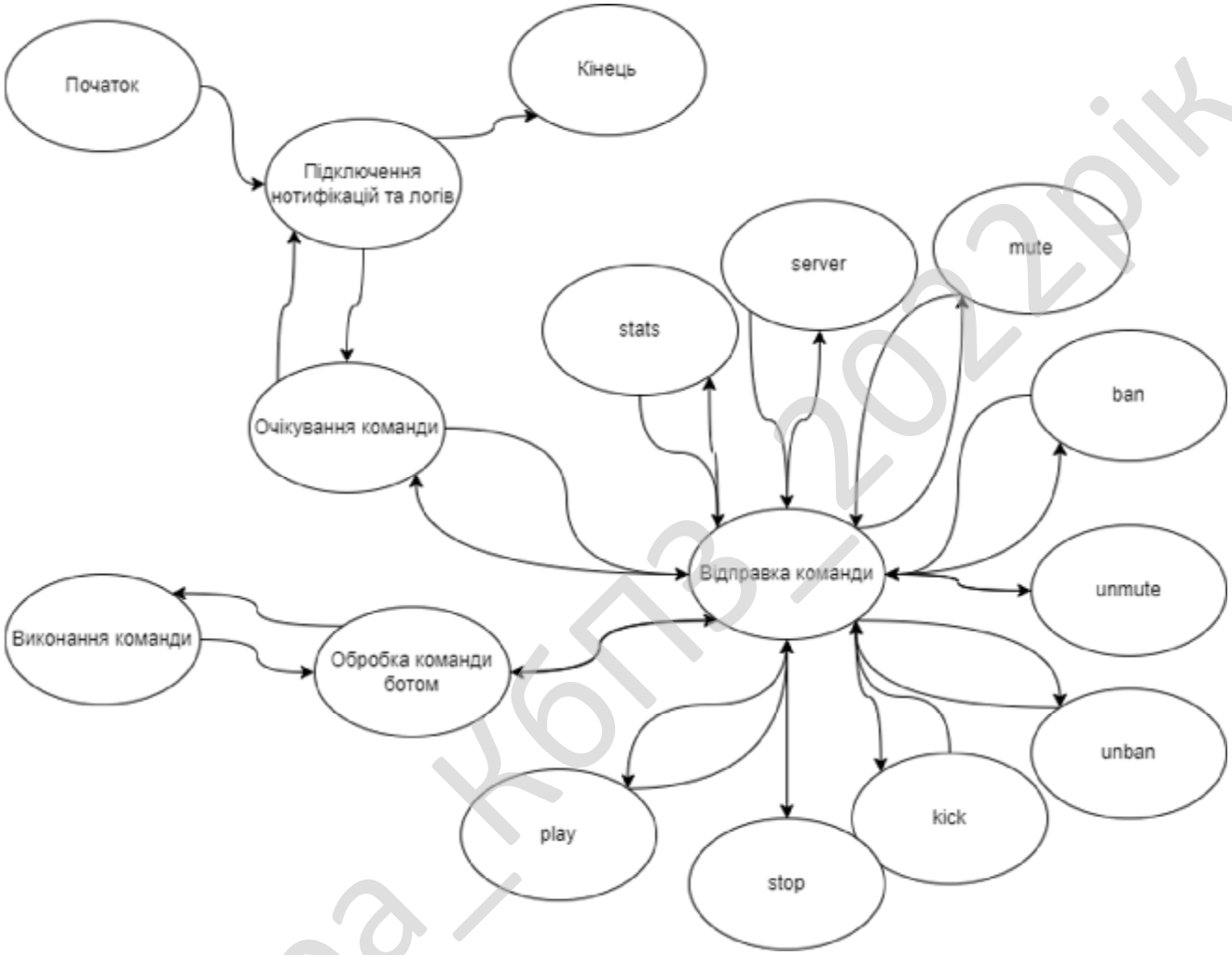

Рисунок 3.12 - Діаграма процесів

Діаграма процесів показує нам як взаємодіють модулі між собою. Головнім процесом тут є відправка і обробка команди ботом. Процес відправки взаємодіє з такими процесами як:

- Команда “stats”
- Команда “server”
- Команда “mute”
- Команда “ban”
- Команда “unmute”

- Команда “unban”
- Команда “kick”
- Команда “stop”
- Команда “play”

Процес обробки команди ботом як раз виконує ці команди. На діаграмі процесів зображено як відбуваються взаємодії в запровадженій системі. Для початку бот включає нотифікації та логи, це перший процес який відбувається при підключенні бота до серверу і після цього очікує команди, якщо команда відправлена він обирає той модуль який йому потрібно та обробляє запит, знову очікуючи команду.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

Реалізована під час виконання кваліфікаційної роботи інформаційна система є набором функцій, які функціонують з API програми Discord, а також із сторонніми за допомогою використання бібліотек.

Для більш наглядного прикладу, розроблювана інформаційна система, хоч і подається у вигляді цільного додатку, насправді функціонує як велика кількість функцій, які лише частково залежить одна від однієї, що гарантує роботу додатку вразі будь яких змін у коді однієї функції.

Програмний додаток складається з серверної частини, яка забезпечую роботу без збоїв, з клієнтської частини, якою є програма Discord, яка одночасно є прошарком між користувачами та інформаційною системою, дозволяючи передавати запити користувачів на серверну частину.

Оскільки мова програмування Python підтримує велику кількість сторонніх бібліотек які можна використовувати під час розробки, що спрощує написання коду, для створення функцій інформаційної системи було використано бібліотеку Discord.py.

Discord.py це бібліотека що дозволяє спростити роботу з API додатку Discord, і дозволяє створювати необхідні запити на мові програмування python, за допомогою взаємодії з веб-інтерфейсом. Фактично це спрощує процес розробки, оскільки замість прямих запитів до API, можна використовувати бібліотеку, яка робить ці запити і видає необхідній результат.

У випадку коли все ж таки необхідно зробити якісь запити, які можуть стосуватись сторонніх API, використовується бібліотека Requests.

Requests дозволяє надсилати HTTP/1.1 запити дуже просто, без необхідності вручну створювати черги запитів до посилань чи формувань

власних PUT & POST даних. Це є одна із найпопулярніших бібліотек Python, яка створює близько 30 мільйонів запитів на тиждень.

Для реалізації однією з функцій було використано таку бібліотеку як PIL (Python Imaging Library). Ця бібліотека додає можливість інтерпретатору Python обробляти картинки. Бібліотека надає розширений доступ до форматування файлів, які потім можуть бути використані в статистичних аналізах та інших цілях, а також дуже потужну можливість обробки зображень.

Ядро бібліотеки розроблено для швидкого доступу до даних зображених в кількох базових піксельних форматах. Також надає основу для загальної обробки зображень при розробці програмних засобів.

Першою з основних функцій є автоматизація керування доступами для користувачів сервера, інформаційна система надає модератору можливість як тимчасово обмежити користувачів в доступі, так і назавжди. Можливість відновлення доступу є, але це повинна робити вручну адміністрація серверу, оскільки ця функція використовується в крайньому випадку.

Також, коли користувач вперше заходить на сервер, у нього відсутній доступ до будь якого іншого каналу, окрім спеціально створеного каналу з правилами. Тільки після того як він ознайомлюється з правилами, та натискає відповідну кнопку яка створюється інформаційною системою, йому автоматично видається доступ до перегляду каналів.

При цьому, доступ не обов’язково надається до всіх каналів, деякі можуть буде приховані адміністраторами і надалі, і надаватись уже відповідно від інших ролей, таким чином реалізується ієрархія доступу, щоб прості користувачі, наприклад, не бачили каналів з логуванням дій на сервері (рисунок 4.2).

Для адміністрування дуже часто використовують команду бану, яка є у стандартних фунціях Discord, але за допомогою бота можна додати час на який можна забанити людину, написати причину і все це однією командую, автоматизовано, тому що у стандартних функціях Discord не можно банити на певний час. На рисунку 4.1 показано програмну реалізацію команди бану на сервері.

```python
# Ban
@client.command(pass_context=True)
@commands.has_permissions(administrator=True)
async def ban(ctx, member: discord.Member, *, reason):
    emb = discord.Embed(title='Ban', colour=discord.Color.red())
    await ctx.channel.purge(limit=1)

    await member.ban(reason=reason)

    emb.set_author(name=member.name, icon_url=member.avatar)
    emb.add_field(name='Ban action', value='User has been banned: {}'.format(member.mention) + ' Why: ' + reason)
    emb.set_footer(text='Who banned: {}'.format(ctx.author.name), icon_url=ctx.author.avatar)

    await ctx.send(embed=emb)
    # await ctx.send(f'User {member.mention} has been banned, farewell: ' + reason)
```

Рисунок 4.1 - Реалізація команди “бану” користувачів

Рисунок 4.2 - Реалізація системи логування дій (зміна та видалення повідомлення)

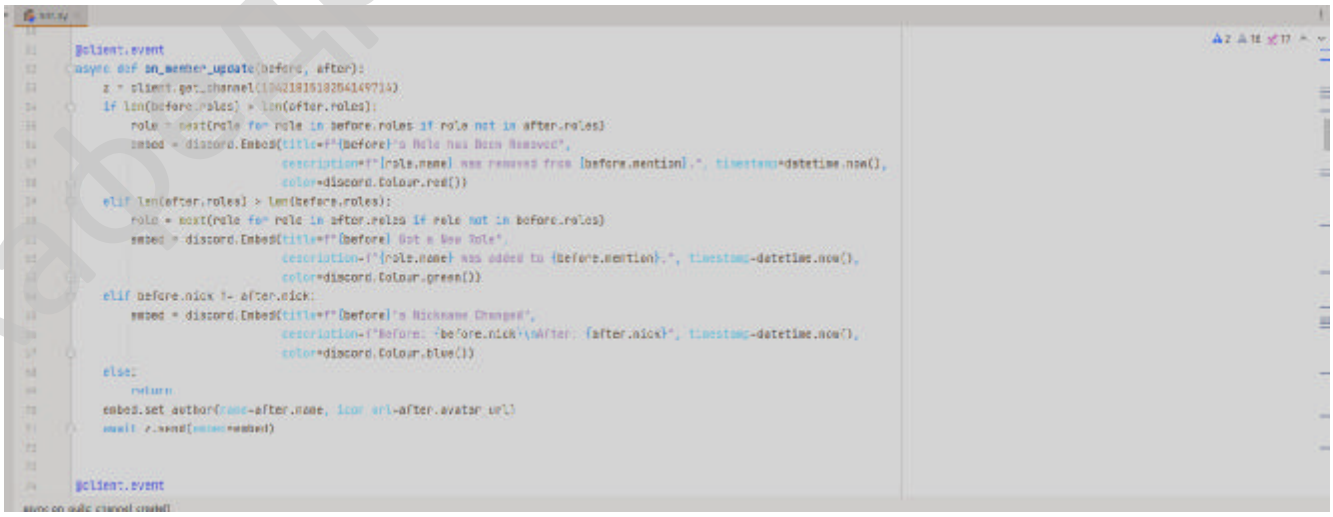

Рисунок 4.3 - Реалізація системи логування дій (зміна або видача ролі)

Рисунок 4.4 - Реалізація системи логування дій (створення та видалення каналів)

Однією з функцій реалізованою системою є виведення статистики про користувачів показана на рисунку 4.5. Оскільки наданий програмою Discord функціонал надає статистику в різних частинах, для більш компактного її відображення було створено більш детальну та компактну статистику, яка за допомогою бібліотеки Pillow реалізовує виведення її на сервері в якості зображення.

Аналогічну систему було створено і для серверу, за допомогою API запитів виводиться точна інформація про сервер, подана у форматі зображення показано на рисунку 4.6.

Використовуючи логування, система безпеки зможе швидко встановити вид порушення, оцінити завдані збитки, ще й виявити зловмисника, якщо наприклад хтось з адміністрації зловживає своїми повноваженнями.

```python
@client.command(aliases=['я', 'stats', 'me'])
async def card_user(ctx):
    await ctx.channel.purge(limit=1)
    img = Image.new('RGBA', (400, 200), '#232529')
    url = str(ctx.author.avatar_url)[:-10]

    response = requests.get(url, stream=True)
    response = Image.open(io.BytesIO(response.content))
    response = response.convert('RGBA')
    response = response.resize((100, 100), Image.ANTIALIAS)

    img.paste(response, (15, 15, 115, 115))
    idraw = ImageDraw.Draw(img)
    name = ctx.author.name
    tag = ctx.author.discriminator

    headline = ImageFont.truetype('arial.ttf', size=20)
    undertext = ImageFont.truetype('arial.ttf', size=12)

    idraw.text((145, 15), f'{name}#{tag}', font=headline)
    idraw.text((145, 50), f'Top role: {ctx.author.top_role}', font=undertext)
    idraw.text((145, 85), f'Join date: {ctx.author.joined_at.strftime(date_format)}', font=undertext)
    idraw.text((145, 120), f'Registration date: {ctx.author.created_at.strftime(date_format)}', font=undertext)
    idraw.text((145, 155), f'User status: {ctx.author.status}', font=undertext)

    img.save('user_card.png')

    await ctx.send(file=discord.File(fp='user_card.png'))
```

Рисунок 4.5 - Реалізація системи статистики користувачів

```python
@client.command(aliases=['server', 'сервер'])
async def card_server(ctx):
    await ctx.channel.purge(limit=1)
    img = Image.new('RGBA', (400, 200), '#232529')
    url = str(ctx.guild.icon_url)[:-10]

    response = requests.get(url, stream=True)
    response = Image.open(io.BytesIO(response.content))
    response = response.convert('RGBA')
    response = response.resize((100, 100), Image.ANTIALIAS)

    img.paste(response, (15, 15, 115, 115))
    idraw = ImageDraw.Draw(img)
    name = ctx.guild.name

    headline = ImageFont.truetype('arial.ttf', size=20)
    undertext = ImageFont.truetype('arial.ttf', size=12)

    idraw.text((145, 15), f'{name}', font=headline)
    idraw.text((145, 50), f'Owner: {ctx.guild.owner}', font=undertext)
    idraw.text((145, 85), f'Number of users: {ctx.guild.member_count}', font=undertext)
    idraw.text((145, 120), f'Number of channels: {len(ctx.guild.channels)}', font=undertext)
    idraw.text((145, 140), f'Number of text channels: {len(ctx.guild.text_channels)}', font=undertext)
    idraw.text((145, 160), f'Number of voice channels: {len(ctx.guild.voice_channels)}', font=undertext)
    img.save('server_card.png')

    await ctx.send(file=discord.File(fp='server_card.png'))
```

Рисунок 4.6 - Реалізація системи інформації про сервер

При приєднанні до серверу, користувачам автоматично надається роль (рисунок 4.7) з базовим функціоналом, роль може змінюватись в залежності від дій адміністрації.

```
async def on_member_join(member):
    channel = client.get_channel(861970912628705233)

    role=discord.utils.get(member.guild.roles, id=1042188049355911249)

    await member.add_roles(role)
    await channel.send(embed=discord.Embed(description=f'User  {member.name} , has joined our team!',
                                           color_=_0x0c0c0c))
```

Рисунок 4.7 - Реалізація системи автоматичної видачі ролі користувачу

Автоматична видача ролі потрібна для того щоб користувач пройшов певну авторизацію на сервері, щоб це не був бот який може зашкодити серверу, чи буде розповсюджувати рекламу.

Також для серверів з розважальною тематикою, власники яких проводять онлайн трансляції на платформі Twitch, було реалізовано систему автоматичного сповіщення при початку трансляцій. Для великих онлайн спільнот це корисна функція, оскільки забирає на себе те, що потрібно було б робити самому власнику, чи іншим людям з персоналу адміністрації показано на рисунку 4.8.

Ця функція може використовуватися на глобальних серверах для зкріплення аудиторії, показує чим займатиметься учасник на своїй трансляції та зверне увагу на нього. Таким чином людина може просувати себе щоб інші люди могли побачити його творчість та можливо знайти друга. Це дуже корисна функція для серверів з наприклад постійними стримерами які проводять свої трансляції на таких платформах як Twitch, Trovo, Facebook Gaming, YouTube.

```python
async def on_presence_update(before, after):
    if after.guild.id == 361970912058408960:
        if before.activity == after.activity:
            return
        global time1
        guild = client.get_guild(361970912058408960)
        role = guild.get_role(1041826271178543205)
        channel = guild.get_channel(1044229509412573294) #trash:816217033800286239 testing:1044229509412573294
        async for message in channel.history(limit=200):
            if before.mention in message.content and "is now streaming" in message.content:
                if isinstance(after.activity, Streaming):
                    return
        if isinstance(after.activity, Streaming):

            await after.add_roles(role)
            emb = discord.Embed(title = ":red_circle: " + after.activity.twitch_name + " is now streaming",
                                colour=0x5865F2, url=after.activity.url, timestamp=datetime.datetime.now())

            emb.set_author(name=after.display_name, icon_url=after.avatar)
            emb.add_field(name="Name stream", value=after.activity.name)
            emb.add_field(name="Category", value=after.activity.game, inline= False)
            emb.add_field(name="Platform", value=after.activity.platform)
            emb.set_footer(text=guild.name, icon_url=guild.icon)
#            emb.set_image(url=after.activity.url)
            emb.set_thumbnail(url=after.activity.url)

            time1 = datetime.datetime.now()
            streaming_service = after.activity.platform
            await channel.send(f":red_circle: **LIVE**\n{before.mention} is now streaming on {streaming_service}!")
            await channel.send(embed=emb)
        elif isinstance(before.activity, Streaming):
            await after.remove_roles(role)
            async for message in channel.history(limit=200):
                if before.mention in message.content and f' is now streaming' in message.content:
                    emb = discord.Embed(title = ":checkered_flag: " + before.activity.twitch_name + " stream was over",
                                        colour=0xF25F58, timestamp=datetime.datetime.now())
                    time2 = datetime.datetime.now()
                    tdelta = time2 - time1
                    if tdelta.seconds>=60:
                        m = tdelta.seconds//60
                        s = tdelta.seconds%60
                        timestream = datetime.time(minute=m, second=s)
                        if m>60:
                            h = m//60
                            m = m%60
                            timestream = datetime.time(hour=h, minute=m, second=s)
                    else: timestream = datetime.time(second=tdelta.seconds)
                    emb.set_author(name=before.display_name, icon_url=before.avatar)
                    emb.add_field(name="Name stream", value=before.activity.name)
                    emb.add_field(name="Time stream", value=timestream, inline=False)
                    emb.add_field(name="Channel", value=f'Link: {before.activity.url}')
                    emb.set_footer(text=guild.name, icon_url=guild.icon)
                    emb.set_thumbnail(url=before.avatar)
                    await channel.send(embed=emb)
                    await message.delete()
        else:
            return
#end of twitch notify
```

Рисунок 4.8 - Реалізація системи сповіщень Twitch

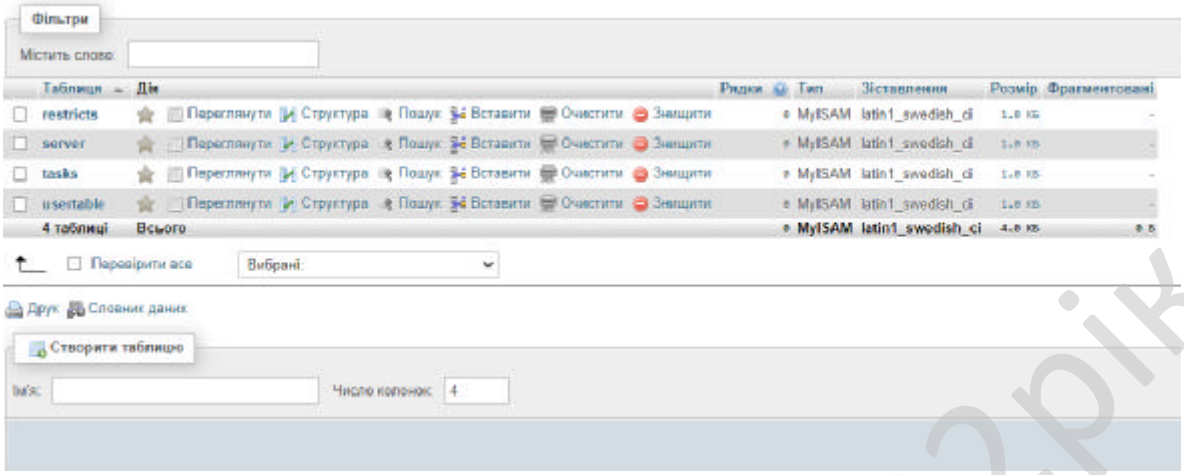

Рисунок 4.9 - Структура бази даних проекту

В базі даних створені такі таблиці:

- Restricts – для створення списку заборонених слів чи структур для використання на сервері.
- Server – головна таблиця, містить дані про сервер на якому оперує бот.
- Tasks – табличка для створення карток-завдань на сервері.
- Usertable – табличка для користувачів.

Така мала кількість табличок обумовлена тим, що велика частина процесів відбувається за допомогою звертань до API програми Discord, що дозволяє реалізувати функціонал без зберігання інформації, що в свою чергу дозволяє зменшити навантаження на сервер.

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.10.

Рисунок 4.10 - Блок-схема основної програми

З рисунку видно, що після запуску програми спочатку відбувається підключення нотифікацій та логів. Після цього бот очікує введення команди з наведеного списка:

- #clear – очистка попередніх повідомлень, без вказування числа очищується 100 повідомлень.
- #ban – блокування певного користувача на сервері, без можливості повернення якщо блокування не буде знято.
- #kick – команда для відключення користувача від серверу, з можливістю в користувача повернутись в будь який момент по посиланню.
- #unban – зняття блокування.
- #stats – відображення статистики користувача на сервері.
- #server – відображення інформації про сервер.
- #play – команда для початку відтворення музики в голосовому каналі.
- #stop – команда для зупинення відтворення музики.

Обробляючи команду бот перевіряє чи потрібен доступ до БД, якщо потрібен то надає туди запит, потім бот перевіряє чи є посилання і якщо є він посилає запит у підпрограму включення музики, блок-схема якого наведена нижче.

З блок-схеми на рисунку 4.11 видно як працює підпрограма виконання музики, спочатку перевіряється чи є посилання, якщо є то йде пошук за посиланням, якщо не нема то йде пошук за назвою та обирає найкращий варіант та включає музику і каналі з користувачем. Після виконання команди бот очікує наступну команду.

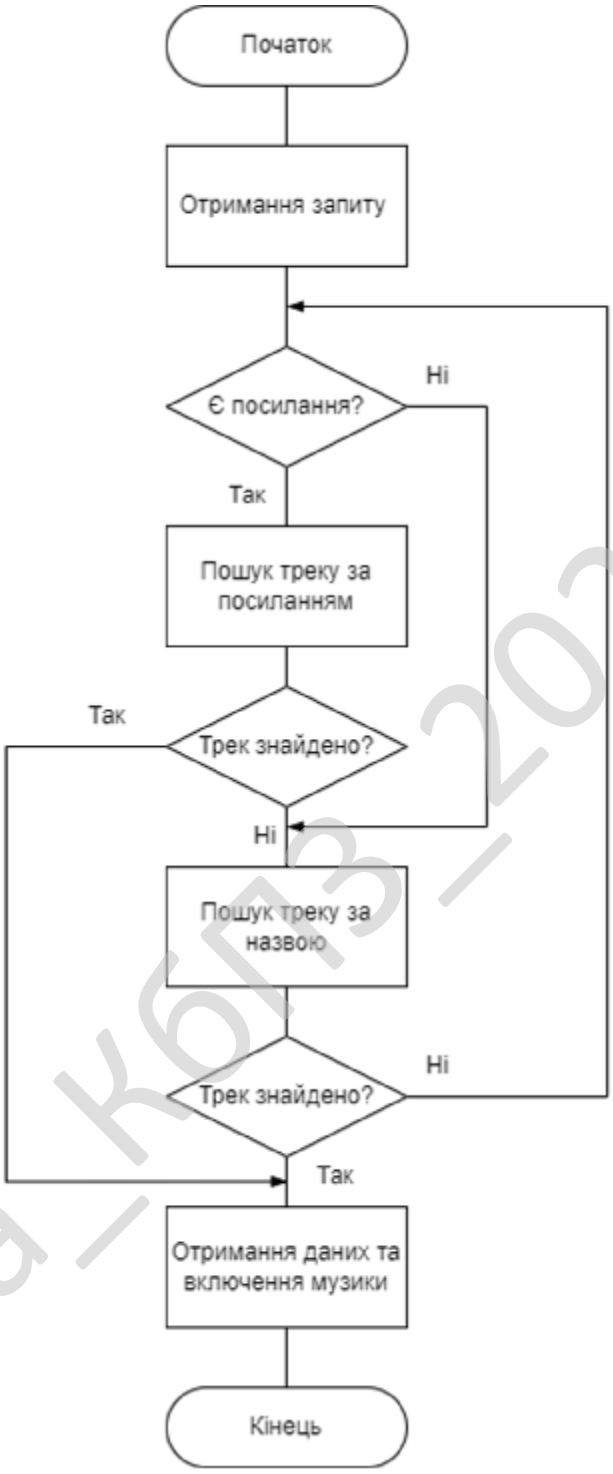

Рисунок 4.11 - Блок-схема підпрограми

4.2 Захист розробленого програмного забезпечення

Було прийнято рішення що розроблене ПЗ буде поширюватися за допомогою вільної ліцензії. Тому було розглянуто декілька видів вільних ліцензій

Існують наступні види вільних ліцензій:

GNU General Public License

Ця ліцензія розшифровується як універсальна громадська ліцензія. Основним її принципом є те, що будь-яка зміна кода має бути опублікованою. Програма з цією ліцензією не може бути включена до складу іншої програми, якщо остання не має вільної ліцензії. ПЗ може бути вільно розповсюджено користувачами, за умови якщо зміни будуть опубліковані. Існують три версії цієї ліцензії у яких були дещо послаблені обмеження стосовно програм які не мають вільну ліцензію.

MIT License

Цей вид ліцензії є дозвільною ліцензією, тобто користувач майже не обмежений у користуванні та розробці програмного забезпечення. Також це означає що програми з цією ліцензією можуть використовуватись як частина будь якого програмного забезпечення

Apache License 2.0

Ця ліцензія є також дозвільною ліцензією. Програми з цією ліцензією не обмежені у розповсюдженні та можуть бути вбудовані у невільне програмне забезпечення. Але заборонено змінювати назву, та файли повинні містити інформацію про всі зміни та ліцензію.

BSD License 2.0

Ця ліцензія на програмне забезпечення є схожою на ліцензію MIT та програми з цією ліцензію дозволенно вбудовувати як частину невільного програмного забезпечення. Але забороняється використовувати оригінальну назву вільного проекту.

Microsoft Public License (MS-PL)

Ця вільна ліцензія надає право на розповсюдження використання та зміну коду. При розповсюдженні необхідно зберігати інформацію про авторські права.

Виходячи з опису основних ліцензій вільного програмного забезпечення та вимог розробленого програмного забезпечення була обрана ліцензія GNU General Public License. Ця ліцензія дає змогу вільну розповсюджувати розроблене програмне забезпечення, та не забороняє вдосконалювати його, у випадку якщо вдосконалена версія також буде знаходитись у вільному доступі.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1

Для додавання на свій сервер створеної інформаційної системи, потрібно перейти на офіційний сайт Discord, знайти в пошуку BraveBot, або ж отримати прями посилання та перейти за ним. Для цього також потрібно мати в наявності доступ до адміністрування серверу, або мати власний сервер, оскільки для того щоб бот функціонував, у нього повинні також бути права адміністрування.

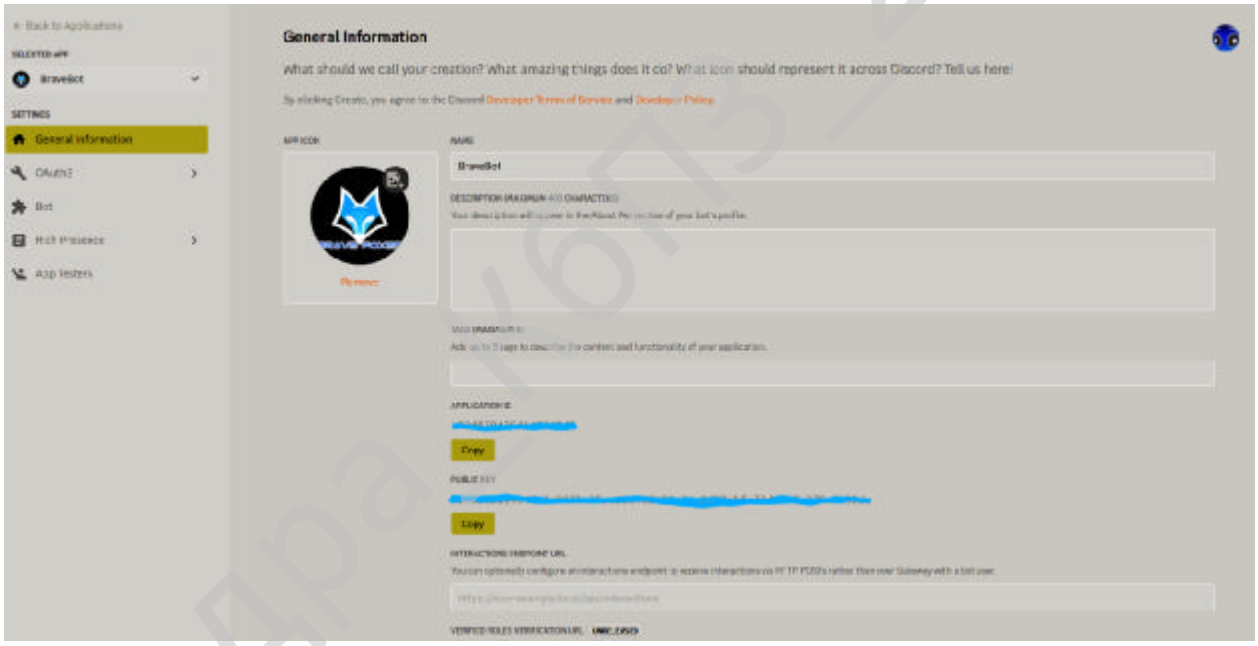

Рисунок 5.1 - Створена система на сайті Discord

Після того як бота приєднали до серверу, надається доступ до створеного функціоналу, для перегляду доступних команд в текстовий чат потрібно прописати *#help*.

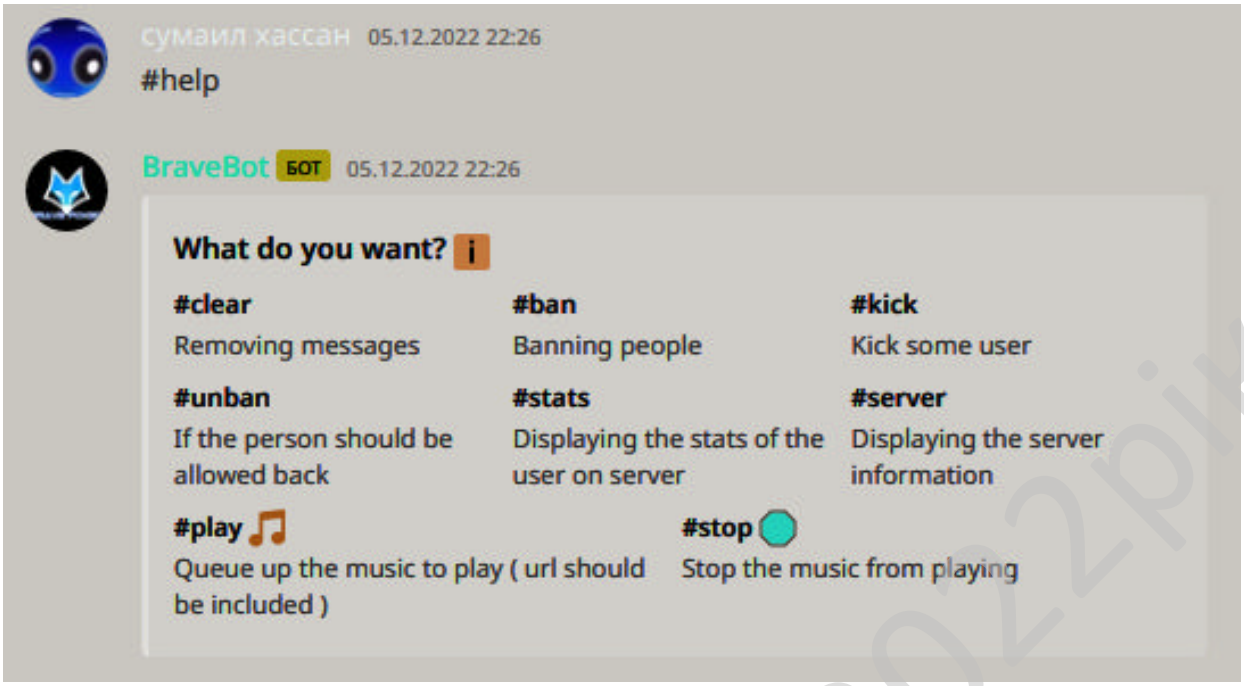

Рисунок 5.2 - Відображення доступ команд

Для адміністрування, було створено наступні команди:

- #clear – очистка попередніх повідомлень, без вказування числа очищується 100 повідомлень.

- #ban – блокування певного користувача на сервері, без можливості повернення якщо блокування не буде знято.

- #kick – команда для відключення користувача від серверу, з можливістю в користувача повернутись в будь який момент по посиланню.

- #unban – зняття блокування.

- #stats – відображення статистики користувача на сервері.

- #server – відображення інформації про сервер.

- #play – команда для початку відтворення музики в голосовому каналі.

- #stop – команда для зупинення відтворення музики.

На наведених далі рисунках приведено приклади роботи команд:

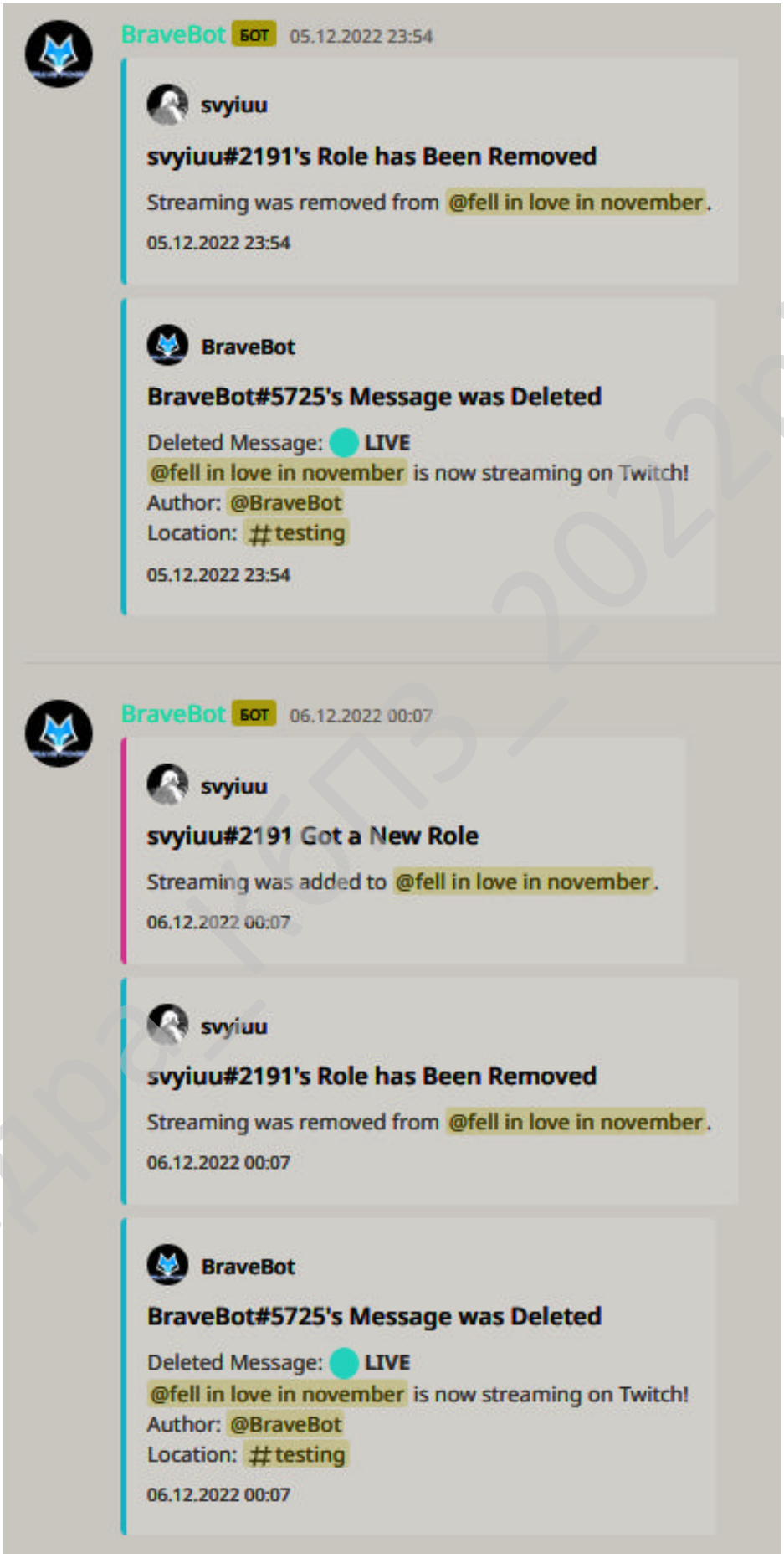

Рисунок 5.3 - Приклад виведення логування дій користувачів

Рисунок 5.4 - Приклад відтворення музики

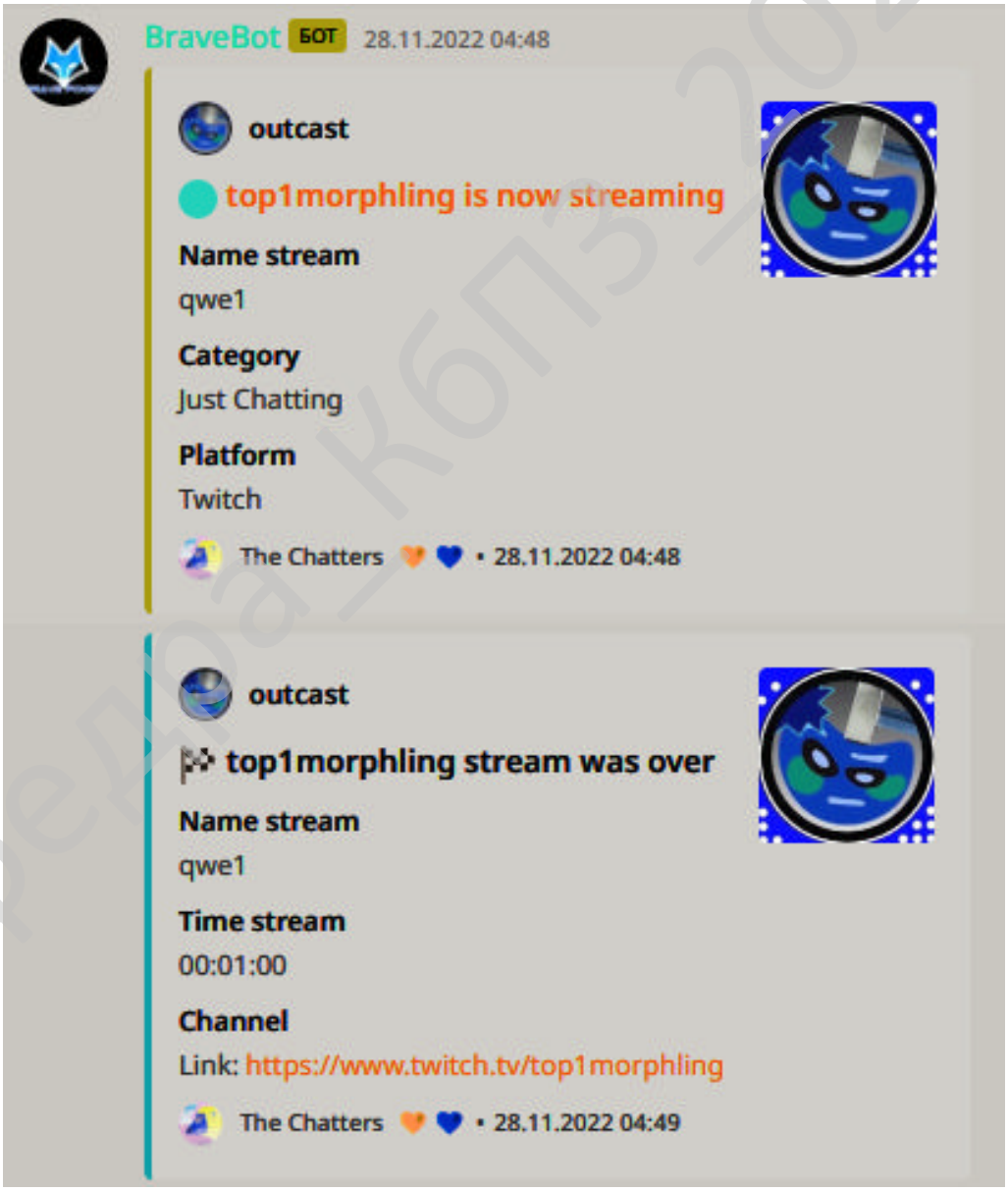

Рисунок 5.4 - Виведення сповіщень про початок онлайн трансляції

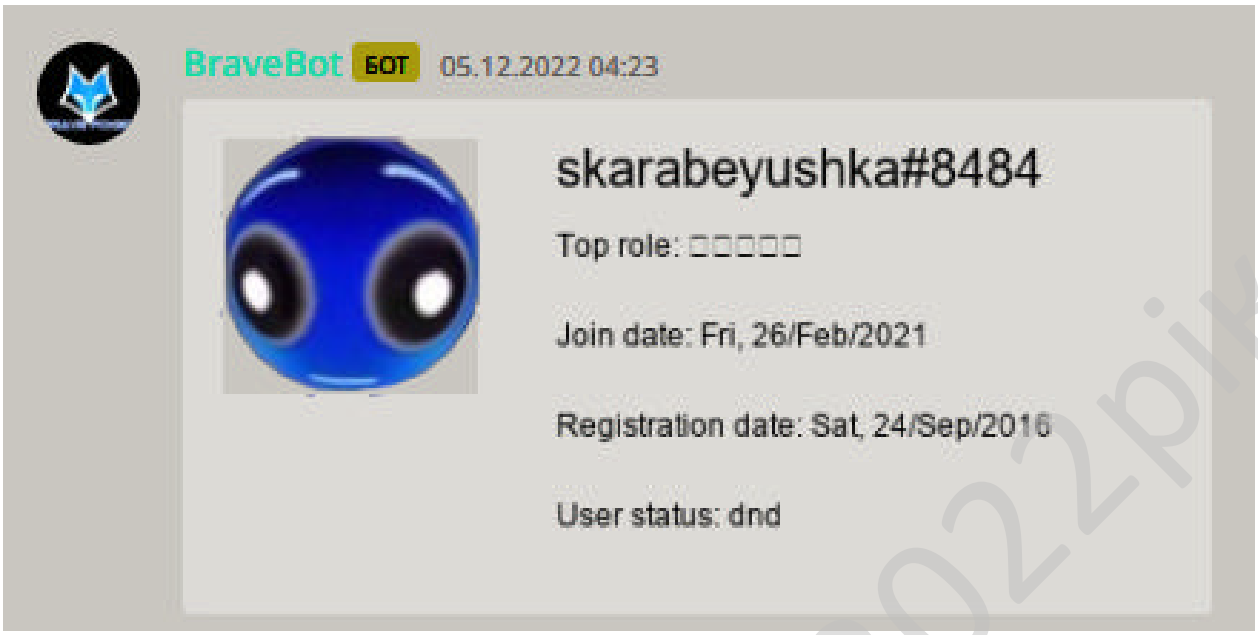

Рисунок 5.5 - Виведення статистики про користувачів

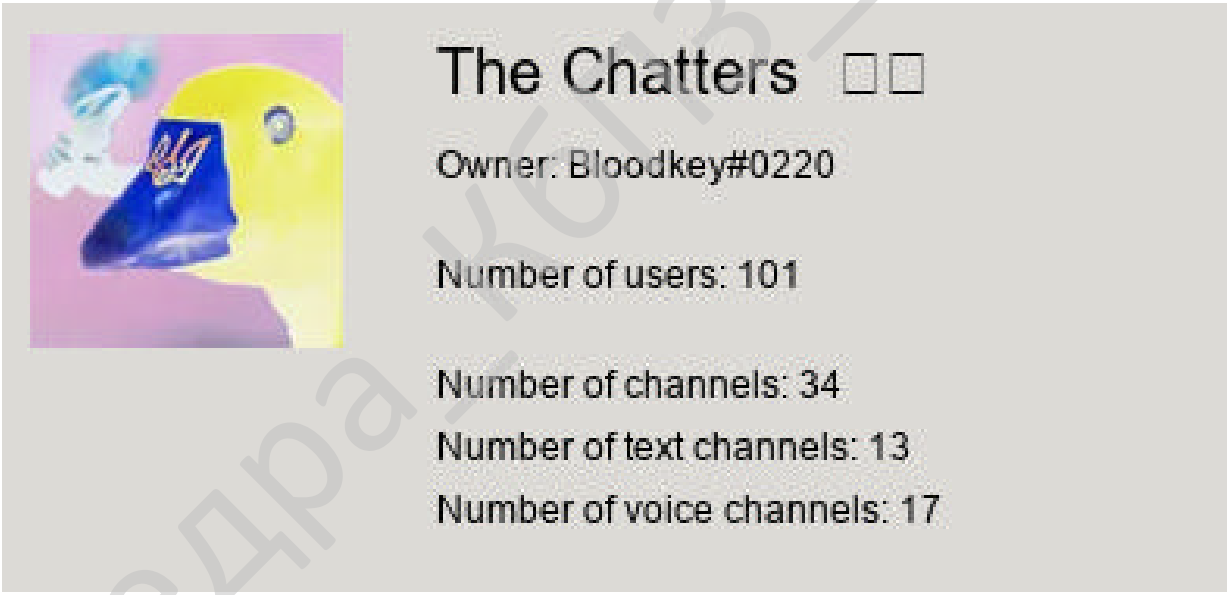

Рисунок 5.6 - Виведення статистики про сервер

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для системи адміністрування глобальних серверів в Discord.

Метою розробки є дослідження та програмна реалізація системи адміністрування глобальних серверів в Discord.

Об'єктом дослідження є процес адміністрування глобальних серверів в Discord.

Предметом дослідження є методи адміністрування глобальних серверів в Discord.

Методи дослідження базуються на методах математичної статистики, методах розробки програмного забезпечення, методах ООП, методах роботи з БД.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод адміністрування глобальних серверів в Discord, який відрізняється від існуючих аналогів, таких як MEE6, Minerea, PancakeBot, має широкі функції по адмініструванню серверу та програванню музики, має логування своїх дій для полегшення адміністрування серверу.

– Розроблено вітчизняний продукт адміністрування глобальних серверів в Discord, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми дипломного проекту

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація Інтернет-боту для адміністрування глобальних серверів в Discord.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп’ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позна-чення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	900
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	90000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \text{ Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

В – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i\,, \tag{7.2}$$

де: Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45\cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \tag{7.3}$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78\cdot(0{,}88\cdot 0{,}93\cdot 0{,}88\cdot 0{,}91\cdot 0{,}95\cdot 1\cdot 1\cdot 0{,}87\cdot 1{,}22\cdot 1{,}16\cdot 1{,}1\cdot 1{,}1\cdot$$
$$\cdot 1{,}12\cdot 1{,}1\cdot 1\cdot 1{,}1\cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S\,, \tag{7.4}$$

де: C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3\cdot 2{,}66\cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot 70 = 118 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	МВ Додаток Е
Ескізний проект	10	МВ Додаток Ж
Технічний проект	9	МВ Додаток З
Робочий проект	118	(7.1-7.4)
Впровадження	13	МВ Додаток О
Всього	159	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{159 \cdot 1}{60 - 5} = 2,9 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	9	810	13,5
Монітор	60	9	540	9
Клавіатура	30	9	270	4,5
Маніпулятор «мишка»	30	9	270	4,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	0	0	0,0
Сканер	20	0	0	0,0
Концентратор–маршрутизатор	30	1	30	0,5
Кабельні господарства ЛВС на 1 м. п.	2,5	150	375	6,25
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	42,58

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{42,58 \cdot 3}{1,2} = 106,5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$Ч_{ел}$ = 106,5/(60·8) = 0,2 ставки.

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру, Windows Server 2012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього			
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього			

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	0,25	8000	6000
Продакт-менеджер	0,25	8000	6000
Інженер-програміст	2,9	8000	69600
Інженер-електронщик	0,2	7000	4200
Адміністратор мережі	0,5	8000	12000
Інженер-верстальник	0,25	7408	5556
Всього за період розробки	$R_{сп}$ = 4,35	-	$Ф_{роб}$ = 103356

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{103356}{4,35 \cdot 60} = 396 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8\,м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 23200 = 1484800 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 148480 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$I_{нв} = R^1_{сп} \cdot Ц_м, \qquad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались за прайсом Інтернет магазину Компбест за 07.11.22 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11186
Системний блок		6488
Процесор	Intel Core I5-750 S-1156 (8M Cache, 2.66 GHz)	=
Системна плата	MB Asrock H55M-LE s1156 (H55, s1156, DDR3 2600(OC)x, ntel(R) HD Graphics, 1xPCI-E 16x, 4xSATA2, Lan 1000 Mb/s, SB 7.1) mATX	=
Відеокарта	AMD Radeon HD 8570, 1 GB GDDR3, 128-bit / LowProfile / DP, DVI	=
Жорсткий диск	SSD 120 Gb + HDD 640 Gb WD 7200 64Mb	=
Оперативна пам’ять	DIMM 8Gb DDR3 PC3-10600 Samsung (2x4 Gb)	=
DVD-привод	Super Multi LG SATA DVD±RW R+22x/-22x, RW+8x/-6x, DL+16x/-12x, RAM 12x, SecurDisc, black (GH22NS40RBB)	=
Корпус	ASUS TA-861 500W FSP ATX-500W (Black/Silver panel) ATX (90-PL861AF5C4-53CZ)	=
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	240

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	3200
Принтер лазерний	CANON LBP-3010 Black	929
Принтер струминний	EPSON STYLUS PHOTO R390	1690
Сканер	Epson Perfection V200 Photo	968
Копіювальний апарат	Copier: CANON IR-1022A	5965
Пристрій безперебійного живлення	UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS)	1498

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість ,шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	8	11186	8948,8	98436,8
Принтер лаз.	2	929	185,8	2043,8
Принтер струм.	1	1690	169	1859

Продовження таблиці 7.7

Найменування обчислювальної техніки	Кількість ,шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Сканери	1	968	96,8	1064,8
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	109966

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1484800	-	-
2. Передавальні пристрої	148480	-	-
Всього по групі	1633280	5	81664
Група 4			
3. Обчислювальна техніка	109966	-	-
Всього по групі	109966	50	54983
Нематеріальні активи			
4. Нематеріальні активи	90000	10	9000

Продовження таблиці 7.8

1	2	3	4
Група 5, 6			
5. Вимірювальні пристрої	5190	25	1297,5
6. Транспортні засоби	109200	20	21840
7. Господарський інвентар	28000	25	7000
Всього по групі	142390	-	30137,5
Разом	$К_р$ = 1975636		$А_р$ = 175785

Примітка: вартість автомобіля взята по даним з автосалону автотрейдінг, вкладки автобазар, джерело http: //www.auto-trading.com.ua/sale/lot20772.html, складає 109200 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 396 \cdot 159/900 = 70 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 70 \cdot 10 \cdot 0{,}01 = 7 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: H_c – відрахування на соціальні потреби, %.

$$C_{оц} = 0{,}01 \cdot 22(70+7) = 28{,}5 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $H_г$ = 15% від основної зарплати:

$$Г_{осп} = З_о \cdot H_г \cdot 0{,}01, \quad (7.14)$$

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 70 \cdot 15 \cdot 0{,}01 = 10{,}5 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Кількість паперу на період розробки визначаємо по середній його витраті за попередній період (одна пачка на місяць). Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м \cdot n. \quad (7.16)$$

$$З_{М1} = 200 \cdot 3 \cdot 1 = 600 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 400):

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 21,5 грн./шт., DVD-R box – 27 грн./шт.

$$З_{М2} = 400 \cdot 27 = 10800 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: картридж для CANON LBP-3010 Black Canon 712 – 574 грн.; картридж для EPSON STYLUS PHOTO R390 – 558 грн.; картридж для CANON IR-1022A – LJ Q2612A Cart. HP LJ 1010/1012/1015/3015/3020/3030 (2500 стр.) – 570 грн.

$$З_{МЗ} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (600 + 10800 + 1702)/900 = 14{,}5 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \tag{7.19}$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 70 \cdot 15 \cdot 0{,}01 = 10{,}5 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 900 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 175785 \cdot 3/(900 \cdot 12) = 49 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \tag{7.21}$$

$$С_п = 70+7+28{,}5+10{,}5+14{,}5+10{,}5+49 = 190 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($Р_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 55%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \tag{7.22}$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 55 \cdot 190 = 104{,}5 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	70
2. Додаткова зарплата виконавців	$З_д$	7
3. Відрахування на соціальні потреби	$С_{оц}$	28,5
4. Загальногосподарські витрати	$Г_{осп}$	10,5
5. Витрати на матеріали	$З_М$	14,5
6. Освоєння нових операційних систем, мов програмування	$О_п$	10,5
7. Амортизація основних фондів	$А_м$	49
8. Повна собівартість програмного забезпечення	$С_п$	190
9. Плановий прибуток	$П_р$	104,5
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	294,5
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	58,9
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	353,4

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну,

транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	353
Всього капітальних витрат	–	353

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування	$З_р$	25546	18788
2. Витрати на електроенергію	$З_{ел}$	173	120
3. Витрати на амортизацію	$З_{ам}$	0	89
Всього витрат за рік	$І$	25719	18997

Витрати на обслуговування системи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування системи за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 240 годин на рік до 200 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 240 \cdot 70 \cdot 1{,}1 \cdot 1{,}22 = 25546\ грн,$$

до:

$$З_{р\ нов} = 200 \cdot 70 \cdot 1{,}1 \cdot 1{,}22 = 18788\ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}35 \cdot 130 \cdot 3{,}8 = 173 \text{ грн.}$$

$$З_{ел\ нов}\ 0{,}35 \cdot 90 \cdot 3{,}8 = 120 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	353	–	88,25
Всього відрахувань	-	–	353	–	88,25

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - E_н \cdot K_р, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.

$$Ев = (294,5 - 190) \cdot 900 - 0,15 \cdot 1975636 \cdot 3/12 = 19963,65 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

$$T_в = \frac{1975636}{(294,5 - 190) \cdot 900 \cdot 12 / 3} = 5 \text{ років}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (25719\text{-}18997)\text{-}0,25 \cdot 353 = 6634 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{353}{25719 - 18997} = 0,1 \text{ року}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	900
2. Повна собівартість розробленої програми	Грн.	190
3. Ціна розробленої програми	Грн.	294,5
4. Плановий прибуток від реалізації розробленої програми	Грн.	104,5
5. Рентабельність програмної продукції	%	55
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1975636
7. Загальний прибуток від реалізації програмної продукції	Грн.	94050
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	19964
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Рік	5
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	353
11.Величина економічного ефекту у користувача програмної продукції	Грн.	6634
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Рік	0,1

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Вступ

Програмісти у процесі роботи отримують негативний вплив на органи зору, а також мають значну розумову напругу і нервово-емоційне навантаження. Руки (м'язи рук та суглоби пальців) при роботі з клавіатурою мають теж істотне навантаження. До шкідливих факторів, які впливають на робітників галузі інформаційних технологій спеціалісти відносять високочастотні електромагнітні коливання роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

До недоліків умов праці користувачів комп'ютерної техніки можна віднести:

- недостатню площу і обсяг виробничого приміщення;
- недотримання вимог, мікроклімату на робочих місцях;
- низький рівень освітленості у приміщеннях і на робочих поверхнях апаратури;
- підвищений рівень низькочастотних магнітних полів від моніторів;
- порушення вимог організації робочих місць;
- недотримання вимог до режимам праці та відпочинку;
- надмірне виробничу навантаження працівників;
- відсутність навичок зниження впливу психоемоційного напруги.

Відповідно до ст.14 Закону «Про охорони праці» на роботодавця покладено обов'язок забезпечити: безпеку працівників при експлуатації устаткування; застосування коштів індивідуальної захисту працівників; відповідні вимоги охорони праці, умови праці в кожному робоче місце; дотримання режиму праці та відпочинку працівників; навчання безпечним методам і прийомам виконання; інструктаж з охорони праці; організацію

контролю над станом умов праці в робочих місць; проведення атестації робочих місць в умовах праці.

Максимально зменшити кількість шкідливих впливів на людину при високій продуктивності праці, створити комфортні умови для роботи людей – ось одна з головних задач охорони праці.

Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальни машини (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

Шкідливими факторами при роботі з персональним комп'ютером є неонізуюче випромінювання промислової частоти, збільшене нервово-емоційне навантаження на оператора, збільшення навантаження на органи зору та дрібні стереостатичні рухи кінцівок.

Ці фактори можуть викликати у працівника певні розлади здоров'я, зокрема підвищення артеріального тиску, кон'юктивіти, тендовагініти ті інші захворювання.

Комп'ютер, як і будь-який електричний прилад, особливо при його неправильному підключенні, може бути джерелом ураження оператора електричним струмом. Саме тому всі працівники, які працюють з персональним комп'ютером, повинні мати першу(або другу) групу допуску з електробезпеки.

Через наявність зазначених факторів працівники, які працюють з персональними комп'ютерами, підлягають попередньому та періодичному медичному огляду згідно з пунктом 6.2.3 додатку 4 до наказу Міністерства охорони здоров'я України "Про затвердження Порядку проведення медичних оглядів працівників певних категорій" від 21 травня 2007 року №246.

Аналіз умов праці на робочому місці фахівця

Робота програміста пов'язана з постійною роботою на ЕОМ, яка відбувається у кімнаті розмірами 4,4 м×6,2 м×2,9 м. Одна з її більших стін має шість двостулкових вікон, розмірами 2 м×1,8 м, які виходять на північний захід. Вікна розташовані рівномірно по всій довжині стіни. Підлога в кімнаті покрита леноліумом, всі стіни пофарбовані світло оранжевого кольору до висоти 2,8 м, а далі підвісна стеля. Уздовж стін розташовані комп'ютерні столи. На них розташовуються 2 персональні комп'ютери й інша оргтехніка (сканер принтери, телефони й ксерокс). Столи мають пластикове покриття. Габарити їхньої робочої поверхні 1255 мм×845 мм. Висота столів 760 мм. Висота стільців від рівня підлоги становить 430 мм.

Згідно НПАОП 0.00 – 1.28 – 10 «Правила охорони праці під час електронно-обчислювальних машин» площа повинна задовольняти умові - не менш 6 м^2 на одне робоче місце. Кратність повітрообміну в приміщенні вузла також регламентується ДСанПіН 3.3.2.007-98, вона повинна становити 20 м^3/годину на одне місце. Виконання даних вимог забезпечить підтримку в приміщенні вузла оптимального значення вологості й складу повітря.

Відповідно ДБН В.2.5 – 28 – 2006 роботу програміста можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення вузла можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при сполученому висвітленні), повинен становити 0,5%, освітленість при штучному висвітленні повинна становити 300 лк.

За результатами виміру освітленості відділом охорони праці величина освітленості від системи загального штучного висвітлення лежить у межах 200-250 лк, що не відповідає вимогам, які пред'являються до приміщення.

Відповідно ДСанПіН 3.3.2.007-98 рівні звукового тиску в робочому приміщенні не повинні перевищувати в октавних смугах із середньо геометричними частотами наступних значень, наведених у таблиці 8.1.

Таблиця 8.1 – Допустимі спектри рівнів звукового тиску

Робоче місце	Рівень звукового тиску, дБ, в октавних смугах із середньогеометричними частотами, Гц								Рівень звуку і еквівалентний рівень звуку, дБА
Приміщення конструкторських бюро, програмістів обчислювальних машин, лабораторій для теоретичних робіт і опрацювання експериментальних даних, прийому хворих в медпунктах									

У приміщенні перебувають наступні джерела шуму: електродвигуни внутрішнього вентилятора ЕОМ; працюючі принтери; працюючі дисководи. Шум, вироблений вентилятором можна класифікувати як постійний, всі інші джерела шуму, як імпульсні. Відповідно паспорта на приміщення рівень звуку, Дб(А), обмірюваний за шкалою (А) шумоміра досяг величини 28,3 Дб(А) при роботі всього встаткування вузла, включаючи й ксерокс. Це дозволяє зробити

висновок про відповідність рівня звуку в приміщенні вимогам нормативних актів.

Ергономічні вимоги до робочого місця працюючого з ВДТ ЕОМ і ПЕОМ нормуються НПАОП 0.00 – 1.28 – 10**.** Оптимальне положення тіла того, що працює забезпечується відповідною конструкцією робочого місця, а також регуляцією висоти робочої поверхні, сидіння, простори й підставки для ніг. Даного місця програміста не мають регульованих параметрів. Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту дані в таблиці 8.2.

Таблиця 8.2 – Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту

Ріст людини, см	Висота робочої поверхні мм,	Висота простору для ніг, мм	Висота робочого сидіння, мм

У дужках зазначені реальні значення параметрів робочого місця; всі вони не відповідають параметрам, зазначеним у стандарті.

Параметри мікроклімату можуть мінятися в широких межах, тоді як необхідною умовою життєдіяльності людини є підтримка сталості температури тіла завдяки властивості терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище.

У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах ДСН 3.3.6.042 - 99 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 8.3).

Таблиця 8.3 - Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	22 - 24°С до 0,1 м/с
Теплий	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	23 - 25°С 0,1 ... 0,2 м / с

Розробка заходів з умов поліпшення охорони праці

Провівши аналіз умов праці в розглянутому вище приміщенні, було отримано наступні результати:

значення мікроклімату в приміщенні не перевищує норму;

розрахунки розміру робочого місця на одного працівника відповідають нормі;

рівень шуму в приміщенні не становить вище норми.

З вище перечислених результатів можна зробити висновок, що основний вплив на продуктивність ІТ-спеціалістів є його психологічний стан. Тому є доцільним зменшити рівень стресу на робочому місці.

Рекомендовані наступні заходи: За потреби особливої концентрації уваги під час виконання робіт суміжні робочі місця операторів необхідно відділяти одне від одного перегородками висотою 1,5 – 2м. Конструкція робочого місця користувача персонального комп’ютера має забезпечити підтримання оптимальної робочої пози офісного працівника. Конструкція робочого столу має відповідати сучасним вимогам ергономіки і забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання (дисплея, клавіатури, принтера) і документів. Висота робочої поверхні робочого столу має регулюватися в межах 680-800 мм, а ширина і глибина – забезпечувати

можливість виконання операцій у зоні досяжності моторного поля (рекомендовані розміри: 600-1400мм, глибина – 800-1000мм). Робочий стіл повинен мати простір для ніг заввишки не менше ніж 600 мм, завширшки не менше ніж 500 мм, завглибшки (на рівні колін) не менше ніж 450 мм, на рівні простягнутої ноги не менше ніж 650 мм. Робочий стілець має бути підйомно-поворотним, регульованим за висотою, з кутом і нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння поверхня сидіння має бути плоскою, передній край – заокругленим.

Розрахункова частина

Для захисного штучного заземлення застосовуються ветикальні електроди: метелевий куток *80·80·8 мм*., (згідно з ДСТУ 2251:2018 «Кутики сталеві гарячекатані рівнополичні. Сортамент») довжиною *L=1,6 м.,* та горизонтальний електрод — метелева полоса з перетином *60·5 мм*. Напруга — *220/380 В*. Розрахункова схема розташування заземлюючих електродів — у ряд.

Розрахунок проведем за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта — чорнозем, нижнього шару грунта — глина (питомий опір $\rho_2 = 40$ *Ом·м)*. Умовна товщина верхнього шару грунта: *H=0,5 м*. Відстань між вертикальними заземлювачами (електродами) *A=1,6 м.* Глибина закладення горизонтального контура заземлення *t=0,6 м*. Опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок

Відстань від центра вертикального заземлювача до поверхні землі:

$T=t+L/2=0,6+1,6/2=1,4$ *м.*

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$\rho = \psi\rho = 1{,}36 \cdot 40 = 54{,}5$ *Ом·м.*

де $\psi = 1{,}36$ - табличне значення коефіцієнта сезонності для відповідної климатичной зони у багатошаровому грунті;

$\rho_2 = 40$ *Ом·м.* - табличне значення питомого опору нижнього шару грунта (глина).

Еквівалентний діаметр ветикальньного електрода (кутка):

$D_в = 0{,}95 \cdot K = 0{,}95 \cdot 80 = 76{,}85$ *мм.* $= 0{,}076$ *м.*

де $K = 80$ *мм.* - розмір полиці метелевого кутка.

Відношення $A/L = 1{,}6/1{,}6 = 1$

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача:

$$R_O = 0{,}366(\rho/L)[\lg(2L/D_в) + (1/2)\lg((4T+L)/(4T-L))] =$$

$$= 21{,}8 \text{ Ом.}$$

Визначаємо коефіцієнт екранування ветикальньних електродів $K_{ев} = 0{,}8$ при орієтовній кількості ветикальньних електродів, яке дорівнює 5.

Визначаємо необхідну кількість вертикальних електродів заземлювача (без вхарування горизонтального заземлювача), при $R_{ЗН} = 4$ *Ом*:

$$R_O / (K_{ев} R_{ЗН}) = 21{,}8 / (0{,}8 \cdot 4) = 6{,}8 \approx 7 \text{ шт.}$$

Визначаємо длину з’єднуючої полоси:

$$L_П = 1{,}05 \cdot A \cdot N = 1{,}05 \cdot 1{,}6 \cdot 7 = 12{,}2 \approx 12 \text{ м.}$$

Опір розтіканню електричного струму з’єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$:

$$R_П = 0{,}366(\rho \cdot K_П / L_П)\lg(2 \cdot L_П^2 / (B \cdot t)) =$$

0,06·0,6))=20,16 Ом.

де $K_П=5$ - табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної климатичної зони для з'єднуючої полоси :

$B = 60$ *мм.* $= 0,06$ *м.*- ширина з'єднуючої полоси (задана).

Загальньний опір розтіканню електричного струму заземлювача:

$$_O \cdot R_П)/(R_O \cdot \eta_П + N \cdot R_П \cdot Кев) =$$

$$=(21,8 \cdot 20,16)/(21,8 \cdot 0,75 + 9 \cdot 20,16 \cdot 0,8) = 3,46 \text{ Ом.}$$

де $\eta_П = 0,75$ - табличне значення коефіцієнта екранування з'єднуючої полоси.

Умова $R \le R_{ЗН}$ виконується ($3,46 \le 4$).

Так як при *7* вертикальних електродах R суттєво менше $R_{ЗН}$, зменшимо кількість вертикальних електродів N до *6* і виконаємо перерахунок. У результаті остаточно отримали: $R = 3,96$ *Ом.* при кількості вертикальних електродів $N=6$.

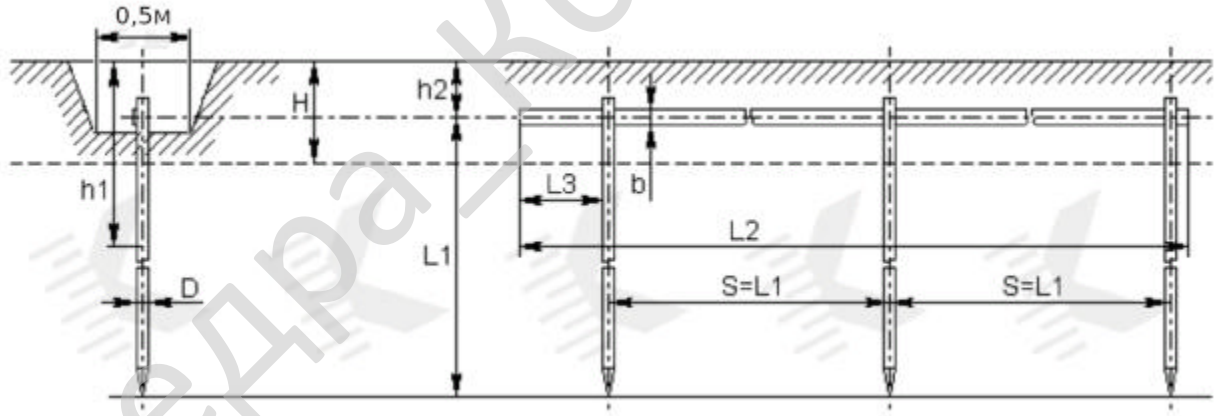

Рисунок 8.1 — Схема штучного заземлення.

Висновок

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз умов праці на робочому місці програміста, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для системи адміністрування глобальних серверів в Discord.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів адміністрування глобальних серверів в Discord.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем адміністрування глобальних серверів в Discord.

– Досліджена система адміністрування глобальних серверів в Discord.

– На основі отриманих результатів досліджень створена програмна реалізація системиадміністрування глобальних серверів в Discord.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання адміністрування глобальних серверів в Discord.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило

мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows XP/Vista/7/8/10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 6634 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,1 років.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Information system [Електронний ресурс] Режим доступу до ресурсу: https://www.britannica.com/topic/information-system/Computer-software

2. Марушко Н.С. , Гончарук Я.А., Лозовицький Д.С., Воляник Г.М., Інформаційні системи і технології в обліку– Магнолія, 2006. – 352 с.

3. What is discord? [Електронний ресурс] Режим доступу до ресурсу: https://www.digitaltrends.com/gaming/what-is-discord/

4. What are webhooks and how they work [Електронний ресурс] Режим доступу до ресурсу: https://hookdeck.com/webhooks/guides/what-are-webhooks-how-they-work#post-or-get-webhooks

5. SQL vs NoSQL [Електронний ресурс] Режим доступу до ресурсу: https://www.integrate.io/blog/the-sql-vs-nosql-difference/

6. Difference between Shared hosting and VPS [Електронний ресурс] Режим доступу до ресурсу: https://www.namecheap.com/support/knowledgebase/article.aspx/525/48/what-is-the-difference-between-shared-hosting-and-vps/

7. What is a Discord bot? [Електронний ресурс] Режим доступу до ресурсу: https://www.xenioo.com/whats-is-a-discord-bot/

8. Що таке системний аналіз [Електронний ресурс] Режим доступу до ресурсу: https://ba-ds.lviv.ua/system-analysis/

9. Ладанюк А. П., Основи системного аналізу – Нова книга, 2004. – 176 с.

10. Goal Tree. An example of Goal Tree. [Електронний ресурс] Режим доступу до ресурсу: https://www.researchgate.net/figure/An-example-of-a-Goal-Tree_fig1_274953315

11. Метод аналізу ієрархій [Електронний ресурс]Режим доступу до ресурсу: https://dss.tg.ck.ua/ahp-help

12. Для чого потрібні діаграми процесів [Електронний ресурс] Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html

13. М. Фаулер., UML. Основи – Символ-Т, 2013

14. Python and MySQL Database [Електронний ресурс] Режим доступу до ресурсу: https://realpython.com/python-mysql/

15. What is a Pycharm? [Електронний ресурс] Режим доступу до ресурсу: https://intellipaat.com/blog/what-is-pycharm/

16. Building a Discord bot in node or python [Електронний ресурс] Режим доступу до ресурсу: https://soyokaze.medium.com/build-a-discord-bot-in-node-or-python-3a4028099657

17. MySQL vs SQL Server [Електронний ресурс] Режим доступу до ресурсу: https://www.dnsstuff.com/mysql-vs-mssql-performance

18. What is hosted services? [Електронний ресурс] Режим доступу до ресурсу: https://www.techtarget.com/searchitchannel/definition/hosted-services

19. Starting with Discord.py [Електронний ресурс] Режим доступу до ресурсу: https://discordpy.readthedocs.io/en/stable/

20. Python: створення і використання бібліотек [Електронний ресурс] Режим доступу до ресурсу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/66_Python/index.html

21. How to make a discord bot in python [Електронний ресурс] Режим доступу до ресурсу: https://realpython.com/how-to-make-a-discord-bot-python/

22. Discord developer portal [Електронний ресурс] Режим доступу до ресурсу: https://discord.com/developers/docs/intro

23. Python requests module [Електронний ресурс] Режим доступу до ресурсу: https://www.w3schools.com/python/module_requests.asp

24. Изучаем Python / Марк Лутц., 2020. – 832 с.

25. Ерік Метіз «Вивчаємо Python. Програмування ігор, візуалізація даних, веб-застосунки»/ Ерік Метіз, 2016. – 496 с.

26. Пол Бэрри «Изучаем программирование на Python» / Пол Бэрри, 2017. – 611 с.

27. Эл Свейгарт «Автоматизация рутинных задач с помощью Python. Практическое руководство для начинающих» / Эл Свейгарт, 2017. – 573 с.

28. Майкл Доусон «Программируем на Python» / Майкл Доусон, 2016. – 480 с.

29. Зед Шоу «Легкий способ выучить Python» / Зед Шоу 2016. – 352 с.

30. Дэн Бейдер «Чистый Python. Тонкости программирования для профи», Дэн Бейдер, 2018. – 288 с.

31. Марк Лутц «Программирование на Python» / Марк Лутц, 2011. – 992 с.

32. Франсуа Шолле «Глубокое обучение на Python» / Франсуа Шолле, 2016. – 480 с.

33. Лучано Рамальо «Python. К вершинам мастерства» / Лучано Рамальо, 2016. – 768 с.

34. Developer portal [Електронний ресурс] Режим доступу до ресурсу: https://discord.com/developers/docs/intro

35. Бот Discord: Как создать и добавить на сервер [Електронний ресурс] Режим доступу до ресурсу: https://appmaster.io/ru/blog/bot-discord-kak-sozdat-i-dobavit-na-server

36. Создайте профессиональный Discord сервер [Електронний ресурс] Режим доступу до ресурсу: https://probot.io/uk

37. Создайте лучший дискорд сервер [Електронний ресурс] Режим доступу до ресурсу: https://mee6.xyz

38. Juniper bot [Електронний ресурс] Режим доступу до ресурсу: https://juniper.bot

39. Welcome to red – Discord bots documentation [Електронний ресурс] Режим доступу до ресурсу: https://docs.discord.red/en/latest/.

Installing Red on Windows [Електронний ресурс] Режим доступу до ресурсу: https://docs.discord.red/en/latest/install_guides/windows.html

40. Державні будівельні норми України ДБН В.2.5 – 28 – 2006: Інженерне обладнання будинків і споруд. Природне і штучне освітлення. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0168667-06#Text (дата звернення 19.10.22).

41. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98 (дата звернення 19.10.22).

42. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення 19.10.22).

43. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення 19.09.22).

44. Методичні рекомендації до виконання розділу "Заходи з охорони праці та техніки безпеки" випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти для здобувачів вищої освіти спеціальностей 123 "Комп'ютерна інженерія" та 122 "Комп'ютерні науки" / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програм. забезпечення; [укл. О.В. Оришака, К.М. Марченко]. - Кропивницький: ЦНТУ, 2022. — 19 с. [Електронний ресурс]. – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12240 (дата звернення 19.09.22).

45. Охорона праці. Ч. 1. Захисне заземлення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд. ; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград : КІСМ, 1997. - 20 с. – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358 (дата звернення 19.09.22).

46. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : "Колос", 1973. – 238 с.

47. Програмування числових методів мовою Python / А. В.Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий. – Київ: Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с. – (Підручник).

48. Національна стратегія розвитку освіти в Україні на 2012–2021 роки. [Електронний ресурс]. Режим доступу: http://www.mon.gov.ua/images/files/news/12/05/4455.pdf.

49. Логи дискорд боту [Електронний ресурс] Режим доступу до ресурсу: https://ru.stackoverflow.com/questions/1237847/Логи-дискорд-боту

50. Налаштування серверу Discord [Електронний ресурс] Режим доступу до ресурсу: https://support.discord.com/hc/ru/categories/200404378-Настройка-сервера

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи адміністрування глобальних серверів в Discord

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі програмування та захисту інформації (нак. №19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація системи адміністрування глобальних серверів в Discord

4 Джерела розробки

Джерелом цієї магістерської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи адміністрування глобальних серверів в Discord;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Компонент повинен бути реалізований на ЕОМ типу IBM PC в операційному середовищі Windows 10 і орієнтований на сумісні з цією платформою зовнішні пристрої, мережне обладнання і прикладне програмне забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2018 року.

8 Вимоги щодо охорони праці

В частині охорони праці магістерської роботи повинна бути розглянута аналіз умов праці програміста в лабораторії К505 ЦНТУ і вироблений розрахунок категорії важкості праці програміста.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 1 аркуш.
– Блок-схема алгоритму підпрограми – 1 аркуш
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 102 аркуша.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист 10.12.2022 р.

11.2 Подання магістерської роботи на захист 23.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за другим (магістерським) рівнем вищої освіти

_____ Р.М. Минайленко

Дослідження та програмна реалізація Інтернет-боту для адміністрування глобальних серверів в Discord

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2022 року

Основна програма

Файл bot.py основної програми

```
from ast import Delete
import asyncio
from sshtunnel import SSHTunnelForwarder
from inspect import getcoroutinelocals
from msilib.schema import Class
from turtle import title
import discord
from discord import Streaming
from discord import Intents
from discord.utils import get
from discord.ext import commands
import datetime
from datetime import timedelta
from youtube_dl import YoutubeDL
import requests
from bs4 import BeautifulSoup as BS
import PIL
from PIL import Image, ImageFont, ImageDraw
import io
import mysql.connector

date_format = "%a, %d/%b/%Y"
intents = discord.Intents.all()
PREFIX = '#'
YDL_OPTIONS = {'format': 'worstaudio/best', 'noplaylist': 'False', 'simulate':
'True', 'key': 'FFmpegExtractAudio'}
FFMPEG_OPTIONS = {'before_options': '-reconnect 1 -reconnect_streamed 1 -
reconnect_delay_max 5', 'options': '-vn'}


client = commands.Bot(command_prefix=PREFIX, intents=intents)
client.remove_command('help')
messagecounts = {}
# Words
restricted_words = ['restricted']
hello_words = ['hello', 'hi', 'привіт']
# songs_url_queue = asyncio.Queue()
# songs_names_queue = asyncio.Queue()
bot_voice_status = 0
global player
time1 = datetime.datetime.now()

# auto role distribution

async def on_member_join(member):
    channel = client.get_channel(361970912628703233)

    role = discord.utils.get(member.guild.roles, id=1042188049355911249)

    await member.add_roles(role)
    await channel.send(emb=discord.Embed(description=f'User ``{member.name}``,
has joined our team!',
                                         color=0x0c0c0c))


# end of auto role distribution

# logs

@client.event
async def on_message_delete(message):
    z = client.get_channel(1042181518254149714)
    embed = discord.Embed(title=f"{message.author}'s Message was Deleted",
```

```
                            description=f"Deleted Message:
{message.content}\nAuthor: {message.author.mention}\nLocation:
{message.channel.mention}",
                            timestamp=datetime.datetime.now(),
color=discord.Colour.red())
    embed.set_author(name=message.author.name, icon_url=message.author.avatar)
    await z.send(embed=embed)


@client.event
async def on_message_edit(before, after):
    z = client.get_channel(1042181518254149714)
    embed = discord.Embed(title=f"{before.author} Edited Their Message",
                            description=f"Before: {before.content}\nAfter:
{after.content}\nAuthor: {before.author.mention}\nLocation:
{before.channel.mention}",
                            timestamp=datetime.datetime.now(),
color=discord.Colour.blue())
    embed.set_author(name=after.author.name, icon_url=after.author.avatar)
    await z.send(embed=embed)


@client.event
async def on_member_update(before, after):
    z = client.get_channel(1042181518254149714)
    if len(before.roles) > len(after.roles):
        role = next(role for role in before.roles if role not in after.roles)
        embed = discord.Embed(title=f"{before}'s Role has Been Removed",
                              description=f"{role.name} was removed from
{before.mention}.", timestamp=datetime.datetime.now(),
                              color=discord.Colour.red())
    elif len(after.roles) > len(before.roles):
        role = next(role for role in after.roles if role not in before.roles)
        embed = discord.Embed(title=f"{before} Got a New Role",
                              description=f"{role.name} was added to
{before.mention}.", timestamp=datetime.datetime.now(),
                              color=discord.Colour.green())
    elif before.nick != after.nick:
        embed = discord.Embed(title=f"{before}'s Nickname Changed",
                              description=f"Before: {before.nick}\nAfter:
{after.nick}", timestamp=datetime.datetime.now(),
                              color=discord.Colour.blue())
    else:
        return
    embed.set_author(name=after.name, icon_url=after.avatar)
    await z.send(embed=embed)


@client.event
async def on_guild_channel_create(channel):
    z = client.get_channel(1042181518254149714)
    embed = discord.Embed(title=f"{channel.name} was Created",
description=channel.mention, timestamp=datetime.datetime.now(),
                            color=discord.Colour.green())
    await z.send(embed=embed)


@client.event
async def on_guild_channel_delete(channel):
    z = client.get_channel(1042181518254149714)
    embed = discord.Embed(title=f"{channel.name} was Deleted",
timestamp=datetime.datetime.now(), color=discord.Colour.red())
    await z.send(embed=embed)


# end of logs

# stop music
@client.command(pass_context=True)
```

```
async def stop(ctx):
    global bot_voice_status
    if bot_voice_status == 0:
        await ctx.send(f'{ctx.author.mention}, im not in channel')
        return
    await ctx.send(f'{ctx.author.mention}, Your wish is my command, music has
been stopped.')
    await ctx.voice_client.disconnect()
    bot_voice_status = 0


# play music
@client.command()
async def play(ctx, url):
    global bot_voice_status
    vc = await ctx.message.author.voice.channel.connect()
    bot_voice_status = 1
    with YoutubeDL(YDL_OPTIONS) as ydl:
        if 'https://' in url:
            info = ydl.extract_info(url, download=False)
        else:
            info = ydl.extract_info(f"ytsearch:{url}",
download=False)['entries'][0]
    link = info['formats'][0]['url']

    vc.play(discord.FFmpegPCMAudio(executable="ffmpeg\\ffmpeg.exe", source=link,
**FFMPEG_OPTIONS))


# End of play music

# User stats

@client.command(aliases=['я', 'stats', 'me'])
async def card_user(ctx, member: discord.Member):
    await ctx.channel.purge(limit=1)
    img = Image.new('RGBA', (400, 200), '#232529')
    url = str(member.avatar)[:-10]

    response = requests.get(url, stream=True)
    response = Image.open(io.BytesIO(response.content))
    response = response.convert('RGBA')
    response = response.resize((100, 100), Image.ANTIALIAS)

    img.paste(response, (15, 15, 115, 115))
    idraw = ImageDraw.Draw(img)
    name = member.name
    tag = member.discriminator

    headline = ImageFont.truetype('arial.ttf', size=20)
    undertext = ImageFont.truetype('arial.ttf', size=12)

    idraw.text((145, 15), f'{name}#{tag}', font=headline)
    idraw.text((145, 50), f'Top role: {member.top_role}', font=undertext)
    idraw.text((145, 85), f'Join date:
{member.joined_at.strftime(date_format)}', font=undertext)
    idraw.text((145, 120), f'Registration date:
{member.created_at.strftime(date_format)}', font=undertext)
    idraw.text((145, 155), f'User status: {member.status}', font=undertext)

    img.save('user_card.png')

    await ctx.send(file=discord.File(fp='user_card.png'))


# End of user stats

# Server stats
```

```
@client.command(aliases=['server', 'сервер'])
async def card_server(ctx):
    await ctx.channel.purge(limit=1)
    img = Image.new('RGBA', (400, 200), '#232529')
    url = str(ctx.guild.icon)[:-10]

    response = requests.get(url, stream=True)
    response = Image.open(io.BytesIO(response.content))
    response = response.convert('RGBA')
    response = response.resize((100, 100), Image.ANTIALIAS)

    img.paste(response, (15, 15, 115, 115))
    idraw = ImageDraw.Draw(img)
    name = ctx.guild.name

    headline = ImageFont.truetype('arial.ttf', size=20)
    undertext = ImageFont.truetype('arial.ttf', size=12)

    idraw.text((145, 15), f'{name}', font=headline)
    idraw.text((145, 50), f'Owner: {ctx.guild.owner}', font=undertext)
    idraw.text((145, 85), f'Number of users: {ctx.guild.member_count}',
font=undertext)
    idraw.text((145, 120), f'Number of channels: {len(ctx.guild.channels)}',
font=undertext)
    idraw.text((145, 140), f'Number of text channels:
{len(ctx.guild.text_channels)}', font=undertext)
    idraw.text((145, 160), f'Number of voice channels:
{len(ctx.guild.voice_channels)}', font=undertext)
    img.save('server_card.png')

    await ctx.send(file=discord.File(fp='server_card.png'))


# End of server stats

#check act test command

@client.command(pass_context=True)
@commands.has_permissions(administrator=True, ban_members=True)
async def act(ctx, member: discord.Member):
     await ctx.send(member.id)
     test = '326762250444603393'
     user = await client.fetch_user(326762250444603393)
     await ctx.guild.unban(user)

#end

#twitchnotify
@client.event
async def on_presence_update(before, after):
    if after.guild.id == 361970912058408960:
        if before.activity == after.activity:
            return
        global time1
        guild = client.get_guild(361970912058408960)
        role = guild.get_role(1041826271178543205)
        channel = guild.get_channel(1044229509412573294)
#trash:816217033800286239 testing:1044229509412573294
        async for message in channel.history(limit=200):
            if before.mention in message.content and "is now streaming" in
message.content:
                if isinstance(after.activity, Streaming):
                    return
        if isinstance(after.activity, Streaming):

            await after.add_roles(role)
            emb = discord.Embed(title = ":red_circle: " +
after.activity.twitch_name + " is now streaming", colour=0x5865F2,
url=after.activity.url, timestamp=datetime.datetime.now())
```

```
            emb.set_author(name=after.display_name, icon_url=after.avatar)
            emb.add_field(name="Name stream", value=after.activity.name)
            emb.add_field(name="Category", value=after.activity.game, inline=
False)
            emb.add_field(name="Platform", value=after.activity.platform)
            emb.set_footer(text=guild.name, icon_url=guild.icon)
#             emb.set_image(url=after.activity.url)
            emb.set_thumbnail(url=after.activity.url)

            time1 = datetime.datetime.now()
            streaming_service = after.activity.platform
            await channel.send(f":red_circle: **LIVE**\n{before.mention} is now
streaming on {streaming_service}!")
            await channel.send(embed=emb)
        elif isinstance(before.activity, Streaming):
            await after.remove_roles(role)
            async for message in channel.history(limit=200):
                if before.mention in message.content and f' is now streaming' in
message.content:
                    emb = discord.Embed(title = ":checkered_flag: " +
before.activity.twitch_name + " stream was over", colour=0xF25F58,
timestamp=datetime.datetime.now())
                    time2 = datetime.datetime.now()
                    tdelta = time2 - time1
                    if tdelta.seconds>=60:
                        m = tdelta.seconds//60
                        s = tdelta.seconds%60
                        timestream = datetime.time(minute=m, second=s)
                        if m>60:
                            h = m//60
                            m = m%60
                            timestream = datetime.time(hour=h, minute=m,
second=s)
                    else: timestream = datetime.time(second=tdelta.seconds)
                    emb.set_author(name=before.display_name,
icon_url=before.avatar)
                    emb.add_field(name="Name stream",
value=before.activity.name)
                    emb.add_field(name="Time stream", value=timestream,
inline=False)
                    emb.add_field(name="Channel", value=f'Link:
{before.activity.url}')
                    emb.set_footer(text=guild.name, icon_url=guild.icon)
                    emb.set_thumbnail(url=before.avatar)
                    await channel.send(embed=emb)
                    await message.delete()
        else:
            return
#end of twitch notify

# Clear message
@client.command(pass_context=True)
@commands.has_permissions(administrator=True)
async def clear(ctx, amount=100):
    await ctx.channel.purge(limit=amount)


# Kick
@client.command(pass_context=True)
@commands.has_permissions(administrator=True)
async def kick(ctx, member: discord.Member, *, reason):
    await ctx.channel.purge(limit=1)

    await member.kick(reason=reason)
    await ctx.send(f'User {member.mention} has been kicked, reason: ' + reason)


# Ban
```

```
@client.command(pass_context=True)
@commands.has_permissions(administrator=True)
async def ban(ctx, member: discord.Member, *, reason):
    emb = discord.Embed(title='Ban', colour=discord.Color.red())
    await ctx.channel.purge(limit=1)

    await member.ban(reason=reason)

    emb.set_author(name=member.name, icon_url=member.avatar)
    emb.add_field(name='Ban action', value='User has been banned:
{}'.format(member.mention) + ' Why: ' + reason)
    emb.set_footer(text='Who banned: {}'.format(ctx.author.name),
icon_url=ctx.author.avatar)

    await ctx.send(embed=emb)
    # await ctx.send(f'User {member.mention} has been banned, farewell: ' +
reason)

# unban
@client.command(pass_context=True)
@commands.has_permissions(administrator=True, ban_members=True)
async def unban(ctx, *, member: discord.User):
    emb = discord.Embed(title='unban', colour=discord.Color.blue())

    await ctx.guild.unban(member)

    emb.set_author(name=member.name, icon_url=member.avatar)
    emb.add_field(name='Unban action', value='User has been unbanned:
{}'.format(member.mention))
    emb.set_footer(text='Who unbanned: {}'.format(ctx.author.name),
icon_url=ctx.author.avatar)

    await ctx.send(embed=emb)

# end of ban/kick/unban

# Mute
@client.command()
@commands.has_permissions(administrator=True)
async def mute(ctx, member: discord.Member):
    await ctx.channel.purge(limit=1)

    mute_role = discord.utils.get(ctx.message.guild.roles, name='Muted')

    await member.add_roles(mute_role)
    await ctx.send(f'User: {member.mention}, has been muted')


# End of mute

# unmute
@client.command()
@commands.has_permissions(administrator=True)
async def unmute(ctx, member: discord.Member):
    await ctx.channel.purge(limit=1)

    mute_role = discord.utils.get(ctx.message.guild.roles, name='Muted')

    await member.remove_roles(mute_role)
    await ctx.send(f'User: {member.mention}, has been unmuted')


# End of unmute

# Help
@client.command(pass_context=True)
async def help(ctx):
    emb = discord.Embed(title='What do you want? :information_source:')
```

```
    emb.add_field(name='{}clear'.format(PREFIX), value='Removing messages')
    emb.add_field(name='{}ban '.format(PREFIX), value='Banning people')
    emb.add_field(name='{}kick'.format(PREFIX), value='Kick some user')
    emb.add_field(name='{}unban'.format(PREFIX), value='If the person should be
allowed back')
    emb.add_field(name='{}stats'.format(PREFIX), value='Displaying the stats of
the user on server')
    emb.add_field(name='{}server '.format(PREFIX), value='Displaying the server
information')
    emb.add_field(name='{}play :musical_note:'.format(PREFIX), value='Queue up
the music to play ( url should be included )')
    emb.add_field(name='{}stop :stop_sign:'.format(PREFIX), value='Stop the
music from playing')

    await ctx.send(embed=emb)

# End of Help

# Embed test
@client.command(pass_context=True)
@commands.has_permissions(administrator=True)
async def check(ctx, title_input):
    emb = discord.Embed(title=title_input, colour=discord.Color.orange(),
                        url='https://www.youtube.com/watch?v=dQw4w9WgXcQ')
    emb.set_author(name=client.user.name, icon_url=client.user.avatar)
    emb.set_footer(text=ctx.author.name, icon_url=ctx.author.avatar)

emb.set_image(url='https://content2.rozetka.com.ua/goods/images/big/16079684.jpg
')
    emb.set_thumbnail(

url='https://isport.ua/i/63/16/53/9/6316539/image_main/c362439388a50f8b52404dee9
b8dc855-resize_crop_1Xquality_100Xallow_enlarge_0Xw_1200Xh_630.jpg')

    now_date = datetime.datetime.now()
    emb.add_field(name='Time', value='Time : {}'.format(now_date))

    await ctx.send(embed=emb)


# End of embed test

@client.event
async def on_ready():
    print('Brave Bot Connected')


# response to messages
@client.event
async def on_message(message):
    await client.process_commands(message)
    msg = message.content.lower()
    if msg in restricted_words:

        await message.delete()
        await message.channel.send(f'{message.author.mention} attention, such
words are restricted on this server')


# end of responses

# Connect
token = open('token.txt', 'r').readline()
client.run(token)
```

Client.py

```
from __future__ import annotations

import asyncio
import datetime
import logging
from typing import (
    Any,
    AsyncIterator,
    Callable,
    Coroutine,
    Dict,
    Generator,
    List,
    Optional,
    Sequence,
    TYPE_CHECKING,
    Tuple,
    Type,
    TypeVar,
    Union,
)

import aiohttp

from .user import User, ClientUser
from .invite import Invite
from .template import Template
from .widget import Widget
from .guild import Guild
from .emoji import Emoji
from .channel import _threaded_channel_factory, PartialMessageable
from .enums import ChannelType
from .mentions import AllowedMentions
from .errors import *
from .enums import Status
from .flags import ApplicationFlags, Intents
from .gateway import *
from .activity import ActivityTypes, BaseActivity, create_activity
from .voice_client import VoiceClient
from .http import HTTPClient
from .state import ConnectionState
from . import utils
from .utils import MISSING, time_snowflake
from .object import Object
from .backoff import ExponentialBackoff
from .webhook import Webhook
from .appinfo import AppInfo
from .ui.view import View
from .stage_instance import StageInstance
from .threads import Thread
from .sticker import GuildSticker, StandardSticker, StickerPack,
_sticker_factory

if TYPE_CHECKING:
    from typing_extensions import Self
    from types import TracebackType
    from .types.guild import Guild as GuildPayload
    from .abc import SnowflakeTime, Snowflake, PrivateChannel
    from .guild import GuildChannel
    from .channel import DMChannel
    from .message import Message
    from .member import Member
    from .voice_client import VoiceProtocol

# fmt: off
__all__ = (
```

```
    'Client',
)
# fmt: on

Coro = TypeVar('Coro', bound=Callable[..., Coroutine[Any, Any, Any]])

_log = logging.getLogger(__name__)

class _LoopSentinel:
    __slots__ = ()

    def __getattr__(self, attr: str) -> None:
        msg = (
            'loop attribute cannot be accessed in non-async contexts. '
            'Consider using either an asynchronous main function and passing it
to asyncio.run or '
            'using asynchronous initialisation hooks such as Client.setup_hook'
        )
        raise AttributeError(msg)

_loop: Any = _LoopSentinel()

class Client:
    r"""Represents a client connection that connects to Discord.
    This class is used to interact with the Discord WebSocket and API.

    .. container:: operations

        .. describe:: async with x

            Asynchronously initialises the client and automatically cleans up.

            .. versionadded:: 2.0

    A number of options can be passed to the :class:`Client`.

    Parameters
    -----------
    max_messages: Optional[:class:`int`]
        The maximum number of messages to store in the internal message cache.
        This defaults to ``1000``. Passing in ``None`` disables the message
cache.

        .. versionchanged:: 1.3
            Allow disabling the message cache and change the default size to
``1000``.
    proxy: Optional[:class:`str`]
        Proxy URL.
    proxy_auth: Optional[:class:`aiohttp.BasicAuth`]
        An object that represents proxy HTTP Basic Authorization.
    shard_id: Optional[:class:`int`]
        Integer starting at ``0`` and less than :attr:`.shard_count`.
    shard_count: Optional[:class:`int`]
        The total number of shards.
    application_id: :class:`int`
        The client's application ID.
    intents: :class:`Intents`
        The intents that you want to enable for the session. This is a way of
        disabling and enabling certain gateway events from triggering and being
sent.

        .. versionadded:: 1.5

        .. versionchanged:: 2.0
            Parameter is now required.
    member_cache_flags: :class:`MemberCacheFlags`
        Allows for finer control over how the library caches members.
        If not given, defaults to cache as much as possible with the
        currently selected intents.
```

```
        .. versionadded:: 1.5
    chunk_guilds_at_startup: :class:`bool`
        Indicates if :func:`.on_ready` should be delayed to chunk all guilds
        at start-up if necessary. This operation is incredibly slow for large
        amounts of guilds. The default is ``True`` if :attr:`Intents.members`
        is ``True``.

        .. versionadded:: 1.5
    status: Optional[:class:`.Status`]
        A status to start your presence with upon logging on to Discord.
    activity: Optional[:class:`.BaseActivity`]
        An activity to start your presence with upon logging on to Discord.
    allowed_mentions: Optional[:class:`AllowedMentions`]
        Control how the client handles mentions by default on every message
sent.

        .. versionadded:: 1.4
    heartbeat_timeout: :class:`float`
        The maximum numbers of seconds before timing out and restarting the
        WebSocket in the case of not receiving a HEARTBEAT_ACK. Useful if
        processing the initial packets take too long to the point of
disconnecting
        you. The default timeout is 60 seconds.
    guild_ready_timeout: :class:`float`
        The maximum number of seconds to wait for the GUILD_CREATE stream to end
before
        preparing the member cache and firing READY. The default timeout is 2
seconds.

        .. versionadded:: 1.4
    assume_unsync_clock: :class:`bool`
        Whether to assume the system clock is unsynced. This applies to the
ratelimit handling
        code. If this is set to ``True``, the default, then the library uses the
time to reset
        a rate limit bucket given by Discord. If this is ``False`` then your
system clock is
        used to calculate how long to sleep for. If this is set to ``False`` it
is recommended to
        sync your system clock to Google's NTP server.

        .. versionadded:: 1.3
    enable_debug_events: :class:`bool`
        Whether to enable events that are useful only for debugging gateway
related information.

        Right now this involves :func:`on_socket_raw_receive` and
:func:`on_socket_raw_send`. If
        this is ``False`` then those events will not be dispatched (due to
performance considerations).
        To enable these events, this must be set to ``True``. Defaults to
``False``.

        .. versionadded:: 2.0
    http_trace: :class:`aiohttp.TraceConfig`
        The trace configuration to use for tracking HTTP requests the library
does using ``aiohttp``.
        This allows you to check requests the library is using. For more
information, check the
        `aiohttp documentation
<https://docs.aiohttp.org/en/stable/client_advanced.html#client-tracing>`_.

        .. versionadded:: 2.0
    max_ratelimit_timeout: Optional[:class:`float`]
        The maximum number of seconds to wait when a non-global rate limit is
encountered.
        If a request requires sleeping for more than the seconds passed in, then
```

```
        :exc:`~discord.RateLimited` will be raised. By default, there is no
timeout limit.
        In order to prevent misuse and unnecessary bans, the minimum value this
can be
        set to is ``30.0`` seconds.

        .. versionadded:: 2.0

    Attributes
    -----------
    ws
        The websocket gateway the client is currently connected to. Could be
``None``.
    """

    def __init__(self, *, intents: Intents, **options: Any) -> None:
        self.loop: asyncio.AbstractEventLoop = _loop
        # self.ws is set in the connect method
        self.ws: DiscordWebSocket = None  # type: ignore
        self._listeners: Dict[str, List[Tuple[asyncio.Future, Callable[...,
bool]]]] = {}
        self.shard_id: Optional[int] = options.get('shard_id')
        self.shard_count: Optional[int] = options.get('shard_count')

        proxy: Optional[str] = options.pop('proxy', None)
        proxy_auth: Optional[aiohttp.BasicAuth] = options.pop('proxy_auth',
None)
        unsync_clock: bool = options.pop('assume_unsync_clock', True)
        http_trace: Optional[aiohttp.TraceConfig] = options.pop('http_trace',
None)
        max_ratelimit_timeout: Optional[float] =
options.pop('max_ratelimit_timeout', None)
        self.http: HTTPClient = HTTPClient(
            self.loop,
            proxy=proxy,
            proxy_auth=proxy_auth,
            unsync_clock=unsync_clock,
            http_trace=http_trace,
            max_ratelimit_timeout=max_ratelimit_timeout,
        )

        self._handlers: Dict[str, Callable[..., None]] = {
            'ready': self._handle_ready,
        }

        self._hooks: Dict[str, Callable[..., Coroutine[Any, Any, Any]]] = {
            'before_identify': self._call_before_identify_hook,
        }

        self._enable_debug_events: bool = options.pop('enable_debug_events',
False)
        self._connection: ConnectionState = self._get_state(intents=intents,
**options)
        self._connection.shard_count = self.shard_count
        self._closed: bool = False
        self._ready: asyncio.Event = MISSING
        self._application: Optional[AppInfo] = None
        self._connection._get_websocket = self._get_websocket
        self._connection._get_client = lambda: self

        if VoiceClient.warn_nacl:
            VoiceClient.warn_nacl = False
            _log.warning("PyNaCl is not installed, voice will NOT be supported")

    async def __aenter__(self) -> Self:
        await self._async_setup_hook()
        return self

    async def __aexit__(
```

```
        self,
        exc_type: Optional[Type[BaseException]],
        exc_value: Optional[BaseException],
        traceback: Optional[TracebackType],
    ) -> None:
        if not self.is_closed():
            await self.close()

    # internals

    def _get_websocket(self, guild_id: Optional[int] = None, *, shard_id:
Optional[int] = None) -> DiscordWebSocket:
        return self.ws

    def _get_state(self, **options: Any) -> ConnectionState:
        return ConnectionState(dispatch=self.dispatch, handlers=self._handlers,
hooks=self._hooks, http=self.http, **options)

    def _handle_ready(self) -> None:
        self._ready.set()

    @property
    def latency(self) -> float:
        """:class:`float`: Measures latency between a HEARTBEAT and a
HEARTBEAT_ACK in seconds.

        This could be referred to as the Discord WebSocket protocol latency.
        """
        ws = self.ws
        return float('nan') if not ws else ws.latency

    def is_ws_ratelimited(self) -> bool:
        """:class:`bool`: Whether the websocket is currently rate limited.

        This can be useful to know when deciding whether you should query
members
        using HTTP or via the gateway.

        .. versionadded:: 1.6
        """
        if self.ws:
            return self.ws.is_ratelimited()
        return False

    @property
    def user(self) -> Optional[ClientUser]:
        """Optional[:class:`.ClientUser`]: Represents the connected client.
``None`` if not logged in."""
        return self._connection.user

    @property
    def guilds(self) -> Sequence[Guild]:
        """Sequence[:class:`.Guild`]: The guilds that the connected client is a
member of."""
        return self._connection.guilds

    @property
    def emojis(self) -> Sequence[Emoji]:
        """Sequence[:class:`.Emoji`]: The emojis that the connected client
has."""
        return self._connection.emojis

    @property
    def stickers(self) -> Sequence[GuildSticker]:
        """Sequence[:class:`.GuildSticker`]: The stickers that the connected
client has.

        .. versionadded:: 2.0
        """
```

```
        return self._connection.stickers

    @property
    def cached_messages(self) -> Sequence[Message]:
        """Sequence[:class:`.Message`]: Read-only list of messages the connected
client has cached.

        .. versionadded:: 1.1
        """
        return utils.SequenceProxy(self._connection._messages or [])

    @property
    def private_channels(self) -> Sequence[PrivateChannel]:
        """Sequence[:class:`.abc.PrivateChannel`]: The private channels that the
connected client is participating on.

        .. note::

            This returns only up to 128 most recent private channels due to an
internal working
            on how Discord deals with private channels.
        """
        return self._connection.private_channels

    @property
    def voice_clients(self) -> List[VoiceProtocol]:
        """List[:class:`.VoiceProtocol`]: Represents a list of voice
connections.

        These are usually :class:`.VoiceClient` instances.
        """
        return self._connection.voice_clients

    @property
    def application_id(self) -> Optional[int]:
        """Optional[:class:`int`]: The client's application ID.

        If this is not passed via ``__init__`` then this is retrieved
        through the gateway when an event contains the data or after a call
        to :meth:`~discord.Client.login`. Usually after
:func:`~discord.on_connect`
        is called.

        .. versionadded:: 2.0
        """
        return self._connection.application_id

    @property
    def application_flags(self) -> ApplicationFlags:
        """:class:`~discord.ApplicationFlags`: The client's application flags.

        .. versionadded:: 2.0
        """
        return self._connection.application_flags

    @property
    def application(self) -> Optional[AppInfo]:
        """Optional[:class:`~discord.AppInfo`]: The client's application info.

        This is retrieved on :meth:`~discord.Client.login` and is not updated
        afterwards. This allows populating the application_id without requiring
a
        gateway connection.

        This is ``None`` if accessed before :meth:`~discord.Client.login` is
called.

        .. seealso:: The :meth:`~discord.Client.application_info` API call
```

```
        .. versionadded:: 2.0
        """
        return self._application

    def is_ready(self) -> bool:
        """:class:`bool`: Specifies if the client's internal cache is ready for
use."""
        return self._ready is not MISSING and self._ready.is_set()

    async def _run_event(
        self,
        coro: Callable[..., Coroutine[Any, Any, Any]],
        event_name: str,
        *args: Any,
        **kwargs: Any,
    ) -> None:
        try:
            await coro(*args, **kwargs)
        except asyncio.CancelledError:
            pass
        except Exception:
            try:
                await self.on_error(event_name, *args, **kwargs)
            except asyncio.CancelledError:
                pass

    def _schedule_event(
        self,
        coro: Callable[..., Coroutine[Any, Any, Any]],
        event_name: str,
        *args: Any,
        **kwargs: Any,
    ) -> asyncio.Task:
        wrapped = self._run_event(coro, event_name, *args, **kwargs)
        # Schedules the task
        return self.loop.create_task(wrapped, name=f'discord.py: {event_name}')

    def dispatch(self, event: str, /, *args: Any, **kwargs: Any) -> None:
        _log.debug('Dispatching event %s', event)
        method = 'on_' + event

        listeners = self._listeners.get(event)
        if listeners:
            removed = []
            for i, (future, condition) in enumerate(listeners):
                if future.cancelled():
                    removed.append(i)
                    continue

                try:
                    result = condition(*args)
                except Exception as exc:
                    future.set_exception(exc)
                    removed.append(i)
                else:
                    if result:
                        if len(args) == 0:
                            future.set_result(None)
                        elif len(args) == 1:
                            future.set_result(args[0])
                        else:
                            future.set_result(args)
                        removed.append(i)

            if len(removed) == len(listeners):
                self._listeners.pop(event)
            else:
                for idx in reversed(removed):
                    del listeners[idx]
```

```
        try:
            coro = getattr(self, method)
        except AttributeError:
            pass
        else:
            self._schedule_event(coro, method, *args, **kwargs)

    async def on_error(self, event_method: str, /, *args: Any, **kwargs: Any) ->
None:
        """|coro|

        The default error handler provided by the client.

        By default this logs to the library logger however it could be
        overridden to have a different implementation.
        Check :func:`~discord.on_error` for more details.

        .. versionchanged:: 2.0

            ``event_method`` parameter is now positional-only
            and instead of writing to ``sys.stderr`` it logs instead.
        """
        _log.exception('Ignoring exception in %s', event_method)

    # hooks

    async def _call_before_identify_hook(self, shard_id: Optional[int], *,
initial: bool = False) -> None:
        # This hook is an internal hook that actually calls the public one.
        # It allows the library to have its own hook without stepping on the
        # toes of those who need to override their own hook.
        await self.before_identify_hook(shard_id, initial=initial)

    async def before_identify_hook(self, shard_id: Optional[int], *, initial:
bool = False) -> None:
        """|coro|

        A hook that is called before IDENTIFYing a session. This is useful
        if you wish to have more control over the synchronization of multiple
        IDENTIFYing clients.

        The default implementation sleeps for 5 seconds.

        .. versionadded:: 1.4

        Parameters
        ------------
        shard_id: :class:`int`
            The shard ID that requested being IDENTIFY'd
        initial: :class:`bool`
            Whether this IDENTIFY is the first initial IDENTIFY.
        """

        if not initial:
            await asyncio.sleep(5.0)

    async def _async_setup_hook(self) -> None:
        # Called whenever the client needs to initialise asyncio objects with a
running loop
        loop = asyncio.get_running_loop()
        self.loop = loop
        self.http.loop = loop
        self._connection.loop = loop

        self._ready = asyncio.Event()

    async def setup_hook(self) -> None:
        """|coro|
```

```
        A coroutine to be called to setup the bot, by default this is blank.

        To perform asynchronous setup after the bot is logged in but before
        it has connected to the Websocket, overwrite this coroutine.

        This is only called once, in :meth:`login`, and will be called before
        any events are dispatched, making it a better solution than doing such
        setup in the :func:`~discord.on_ready` event.

        .. warning::

            Since this is called *before* the websocket connection is made
therefore
            anything that waits for the websocket will deadlock, this includes
things
            like :meth:`wait_for` and :meth:`wait_until_ready`.

        .. versionadded:: 2.0
        """
        pass

    # login state management

    async def login(self, token: str) -> None:
        """|coro|

        Logs in the client with the specified credentials and
        calls the :meth:`setup_hook`.

        Parameters
        -----------
        token: :class:`str`
            The authentication token. Do not prefix this token with
            anything as the library will do it for you.

        Raises
        ------
        LoginFailure
            The wrong credentials are passed.
        HTTPException
            An unknown HTTP related error occurred,
            usually when it isn't 200 or the known incorrect credentials
            passing status code.
        """

        _log.info('logging in using static token')

        if self.loop is _loop:
            await self._async_setup_hook()

        if not isinstance(token, str):
            raise TypeError(f'expected token to be a str, received
{token.__class__!r} instead')
        token = token.strip()

        data = await self.http.static_login(token)
        self._connection.user = ClientUser(state=self._connection, data=data)
        self._application = await self.application_info()
        if self._connection.application_id is None:
            self._connection.application_id = self._application.id

        if not self._connection.application_flags:
            self._connection.application_flags = self._application.flags

        await self.setup_hook()

    async def connect(self, *, reconnect: bool = True) -> None:
        """|coro|
```

```
        Creates a websocket connection and lets the websocket listen
        to messages from Discord. This is a loop that runs the entire
        event system and miscellaneous aspects of the library. Control
        is not resumed until the WebSocket connection is terminated.

        Parameters
        -----------
        reconnect: :class:`bool`
            If we should attempt reconnecting, either due to internet
            failure or a specific failure on Discord's part. Certain
            disconnects that lead to bad state will not be handled (such as
            invalid sharding payloads or bad tokens).

        Raises
        -------
        GatewayNotFound
            If the gateway to connect to Discord is not found. Usually if this
            is thrown then there is a Discord API outage.
        ConnectionClosed
            The websocket connection has been terminated.
        """

        backoff = ExponentialBackoff()
        ws_params = {
            'initial': True,
            'shard_id': self.shard_id,
        }
        while not self.is_closed():
            try:
                coro = DiscordWebSocket.from_client(self, **ws_params)
                self.ws = await asyncio.wait_for(coro, timeout=60.0)
                ws_params['initial'] = False
                while True:
                    await self.ws.poll_event()
            except ReconnectWebSocket as e:
                _log.debug('Got a request to %s the websocket.', e.op)
                self.dispatch('disconnect')
                ws_params.update(sequence=self.ws.sequence, resume=e.resume,
session=self.ws.session_id)
                if e.resume:
                    ws_params['gateway'] = self.ws.gateway
                continue
            except (
                OSError,
                HTTPException,
                GatewayNotFound,
                ConnectionClosed,
                aiohttp.ClientError,
                asyncio.TimeoutError,
            ) as exc:

                self.dispatch('disconnect')
                if not reconnect:
                    await self.close()
                    if isinstance(exc, ConnectionClosed) and exc.code == 1000:
                        # clean close, don't re-raise this
                        return
                    raise

                if self.is_closed():
                    return

                # If we get connection reset by peer then try to RESUME
                if isinstance(exc, OSError) and exc.errno in (54, 10054):
                    ws_params.update(
                        sequence=self.ws.sequence,
                        gateway=self.ws.gateway,
                        initial=False,
```

```
                        resume=True,
                        session=self.ws.session_id,
                    )
                    continue

                # We should only get this when an unhandled close code happens,
                # such as a clean disconnect (1000) or a bad state (bad token,
no sharding, etc)
                # sometimes, discord sends us 1000 for unknown reasons so we
should reconnect
                # regardless and rely on is_closed instead
                if isinstance(exc, ConnectionClosed):
                    if exc.code == 4014:
                        raise PrivilegedIntentsRequired(exc.shard_id) from None
                    if exc.code != 1000:
                        await self.close()
                        raise

                retry = backoff.delay()
                _log.exception("Attempting a reconnect in %.2fs", retry)
                await asyncio.sleep(retry)
                # Always try to RESUME the connection
                # If the connection is not RESUME-able then the gateway will
invalidate the session.
                # This is apparently what the official Discord client does.
                ws_params.update(
                    sequence=self.ws.sequence,
                    gateway=self.ws.gateway,
                    resume=True,
                    session=self.ws.session_id,
                )

    async def close(self) -> None:
        """|coro|

        Closes the connection to Discord.
        """
        if self._closed:
            return

        self._closed = True

        await self._connection.close()

        if self.ws is not None and self.ws.open:
            await self.ws.close(code=1000)

        await self.http.close()

        if self._ready is not MISSING:
            self._ready.clear()

        self.loop = MISSING

    def clear(self) -> None:
        """Clears the internal state of the bot.

        After this, the bot can be considered "re-opened", i.e.
:meth:`is_closed`
        and :meth:`is_ready` both return ``False`` along with the bot's internal
        cache cleared.
        """
        self._closed = False
        self._ready.clear()
        self._connection.clear()
        self.http.clear()

    async def start(self, token: str, *, reconnect: bool = True) -> None:
        """|coro|
```

```
        A shorthand coroutine for :meth:`login` + :meth:`connect`.

        Parameters
        -----------
        token: :class:`str`
            The authentication token. Do not prefix this token with
            anything as the library will do it for you.
        reconnect: :class:`bool`
            If we should attempt reconnecting, either due to internet
            failure or a specific failure on Discord's part. Certain
            disconnects that lead to bad state will not be handled (such as
            invalid sharding payloads or bad tokens).

        Raises
        -------
        TypeError
            An unexpected keyword argument was received.
        """
        await self.login(token)
        await self.connect(reconnect=reconnect)

    def run(
        self,
        token: str,
        *,
        reconnect: bool = True,
        log_handler: Optional[logging.Handler] = MISSING,
        log_formatter: logging.Formatter = MISSING,
        log_level: int = MISSING,
        root_logger: bool = False,
    ) -> None:
        """A blocking call that abstracts away the event loop
        initialisation from you.

        If you want more control over the event loop then this
        function should not be used. Use :meth:`start` coroutine
        or :meth:`connect` + :meth:`login`.

        This function also sets up the logging library to make it easier
        for beginners to know what is going on with the library. For more
        advanced users, this can be disabled by passing ``None`` to
        the ``log_handler`` parameter.

        .. warning::

            This function must be the last function to call due to the fact that
it
            is blocking. That means that registration of events or anything
being
            called after this function call will not execute until it returns.

        Parameters
        -----------
        token: :class:`str`
            The authentication token. Do not prefix this token with
            anything as the library will do it for you.
        reconnect: :class:`bool`
            If we should attempt reconnecting, either due to internet
            failure or a specific failure on Discord's part. Certain
            disconnects that lead to bad state will not be handled (such as
            invalid sharding payloads or bad tokens).
        log_handler: Optional[:class:`logging.Handler`]
            The log handler to use for the library's logger. If this is ``None``
            then the library will not set up anything logging related. Logging
            will still work if ``None`` is passed, though it is your
responsibility
            to set it up.
```

```
            The default log handler if not provided is
:class:`logging.StreamHandler`.

            .. versionadded:: 2.0
        log_formatter: :class:`logging.Formatter`
            The formatter to use with the given log handler. If not provided
then it
            defaults to a colour based logging formatter (if available).

            .. versionadded:: 2.0
        log_level: :class:`int`
            The default log level for the library's logger. This is only applied
if the
            ``log_handler`` parameter is not ``None``. Defaults to
``logging.INFO``.

            .. versionadded:: 2.0
        root_logger: :class:`bool`
            Whether to set up the root logger rather than the library logger.
            By default, only the library logger (``'discord'``) is set up. If
this
            is set to ``True`` then the root logger is set up as well.

            Defaults to ``False``.

            .. versionadded:: 2.0
        """

        async def runner():
            async with self:
                await self.start(token, reconnect=reconnect)

        if log_handler is not None:
            utils.setup_logging(
                handler=log_handler,
                formatter=log_formatter,
                level=log_level,
                root=root_logger,
            )

        try:
            asyncio.run(runner())
        except KeyboardInterrupt:
            # nothing to do here
            # `asyncio.run` handles the loop cleanup
            # and `self.start` closes all sockets and the HTTPClient instance.
            return

    # properties

    def is_closed(self) -> bool:
        """:class:`bool`: Indicates if the websocket connection is closed."""
        return self._closed

    @property
    def activity(self) -> Optional[ActivityTypes]:
        """Optional[:class:`.BaseActivity`]: The activity being used upon
        logging in.
        """
        return create_activity(self._connection._activity, self._connection)

    @activity.setter
    def activity(self, value: Optional[ActivityTypes]) -> None:
        if value is None:
            self._connection._activity = None
        elif isinstance(value, BaseActivity):
            # ConnectionState._activity is typehinted as ActivityPayload, we're
passing Dict[str, Any]
            self._connection._activity = value.to_dict()  # type: ignore
```

```
            else:
                raise TypeError('activity must derive from BaseActivity.')

        @property
        def status(self) -> Status:
            """:class:`.Status`:
            The status being used upon logging on to Discord.

            .. versionadded: 2.0
            """
            if self._connection._status in set(state.value for state in Status):
                return Status(self._connection._status)
            return Status.online

        @status.setter
        def status(self, value: Status) -> None:
            if value is Status.offline:
                self._connection._status = 'invisible'
            elif isinstance(value, Status):
                self._connection._status = str(value)
            else:
                raise TypeError('status must derive from Status.')

        @property
        def allowed_mentions(self) -> Optional[AllowedMentions]:
            """Optional[:class:`~discord.AllowedMentions`]: The allowed mention
configuration.

            .. versionadded:: 1.4
            """
            return self._connection.allowed_mentions

        @allowed_mentions.setter
        def allowed_mentions(self, value: Optional[AllowedMentions]) -> None:
            if value is None or isinstance(value, AllowedMentions):
                self._connection.allowed_mentions = value
            else:
                raise TypeError(f'allowed_mentions must be AllowedMentions not
{value.__class__!r}')

        @property
        def intents(self) -> Intents:
            """:class:`~discord.Intents`: The intents configured for this
connection.

            .. versionadded:: 1.5
            """
            return self._connection.intents

        # helpers/getters

        @property
        def users(self) -> List[User]:
            """List[:class:`~discord.User`]: Returns a list of all the users the bot
can see."""
            return list(self._connection._users.values())

        def get_channel(self, id: int, /) -> Optional[Union[GuildChannel, Thread,
PrivateChannel]]:
            """Returns a channel or thread with the given ID.

            .. versionchanged:: 2.0

                ``id`` parameter is now positional-only.

            Parameters
            -----------
            id: :class:`int`
                The ID to search for.
```

```
        Returns
        --------
        Optional[Union[:class:`.abc.GuildChannel`, :class:`.Thread`,
:class:`.abc.PrivateChannel`]]
            The returned channel or ``None`` if not found.
        """
        return self._connection.get_channel(id)  # type: ignore # The cache
contains all channel types

    def get_partial_messageable(
        self, id: int, *, guild_id: Optional[int] = None, type:
Optional[ChannelType] = None
    ) -> PartialMessageable:
        """Returns a partial messageable with the given channel ID.

        This is useful if you have a channel_id but don't want to do an API call
        to send messages to it.

        .. versionadded:: 2.0

        Parameters
        -----------
        id: :class:`int`
            The channel ID to create a partial messageable for.
        guild_id: Optional[:class:`int`]
            The optional guild ID to create a partial messageable for.

            This is not required to actually send messages, but it does allow
the
            :meth:`~discord.PartialMessageable.jump_url` and
            :attr:`~discord.PartialMessageable.guild` properties to function
properly.
        type: Optional[:class:`.ChannelType`]
            The underlying channel type for the partial messageable.

        Returns
        --------
        :class:`.PartialMessageable`
            The partial messageable
        """
        return PartialMessageable(state=self._connection, id=id,
guild_id=guild_id, type=type)

    def get_stage_instance(self, id: int, /) -> Optional[StageInstance]:
        """Returns a stage instance with the given stage channel ID.

        .. versionadded:: 2.0

        Parameters
        -----------
        id: :class:`int`
            The ID to search for.

        Returns
        --------
        Optional[:class:`.StageInstance`]
            The stage instance or ``None`` if not found.
        """
        from .channel import StageChannel

        channel = self._connection.get_channel(id)

        if isinstance(channel, StageChannel):
            return channel.instance

    def get_guild(self, id: int, /) -> Optional[Guild]:
        """Returns a guild with the given ID.
```

```
        .. versionchanged:: 2.0

            ``id`` parameter is now positional-only.

        Parameters
        -----------
        id: :class:`int`
            The ID to search for.

        Returns
        --------
        Optional[:class:`.Guild`]
            The guild or ``None`` if not found.
        """
        return self._connection._get_guild(id)

    def get_user(self, id: int, /) -> Optional[User]:
        """Returns a user with the given ID.

        .. versionchanged:: 2.0

            ``id`` parameter is now positional-only.

        Parameters
        -----------
        id: :class:`int`
            The ID to search for.

        Returns
        --------
        Optional[:class:`~discord.User`]
            The user or ``None`` if not found.
        """
        return self._connection.get_user(id)

    def get_emoji(self, id: int, /) -> Optional[Emoji]:
        """Returns an emoji with the given ID.

        .. versionchanged:: 2.0

            ``id`` parameter is now positional-only.

        Parameters
        -----------
        id: :class:`int`
            The ID to search for.

        Returns
        --------
        Optional[:class:`.Emoji`]
            The custom emoji or ``None`` if not found.
        """
        return self._connection.get_emoji(id)

    def get_sticker(self, id: int, /) -> Optional[GuildSticker]:
        """Returns a guild sticker with the given ID.

        .. versionadded:: 2.0

        .. note::

            To retrieve standard stickers, use :meth:`.fetch_sticker`.
            or :meth:`.fetch_premium_sticker_packs`.

        Returns
        --------
        Optional[:class:`.GuildSticker`]
            The sticker or ``None`` if not found.
        """
```

```
        return self._connection.get_sticker(id)

    def get_all_channels(self) -> Generator[GuildChannel, None, None]:
        """A generator that retrieves every :class:`.abc.GuildChannel` the
client can 'access'.

        This is equivalent to: ::

            for guild in client.guilds:
                for channel in guild.channels:
                    yield channel

        .. note::

            Just because you receive a :class:`.abc.GuildChannel` does not mean
that
            you can communicate in said channel.
:meth:`.abc.GuildChannel.permissions_for` should
            be used for that.

        Yields
        ------
        :class:`.abc.GuildChannel`
            A channel the client can 'access'.
        """

        for guild in self.guilds:
            yield from guild.channels

    def get_all_members(self) -> Generator[Member, None, None]:
        """Returns a generator with every :class:`.Member` the client can see.

        This is equivalent to: ::

            for guild in client.guilds:
                for member in guild.members:
                    yield member

        Yields
        ------
        :class:`.Member`
            A member the client can see.
        """
        for guild in self.guilds:
            yield from guild.members

    # listeners/waiters

    async def wait_until_ready(self) -> None:
        """|coro|

        Waits until the client's internal cache is all ready.

        .. warning::

            Calling this inside :meth:`setup_hook` can lead to a deadlock.
        """
        if self._ready is not MISSING:
            await self._ready.wait()
        else:
            raise RuntimeError(
                'Client has not been properly initialised. '
                'Please use the login method or asynchronous context manager
before calling this method'
            )

    def wait_for(
        self,
        event: str,
```

```
        /,
        *,
        check: Optional[Callable[..., bool]] = None,
        timeout: Optional[float] = None,
    ) -> Any:
        """|coro|

        Waits for a WebSocket event to be dispatched.

        This could be used to wait for a user to reply to a message,
        or to react to a message, or to edit a message in a self-contained
        way.

        The ``timeout`` parameter is passed onto :func:`asyncio.wait_for`. By
default,
        it does not timeout. Note that this does propagate the
        :exc:`asyncio.TimeoutError` for you in case of timeout and is provided
for
        ease of use.

        In case the event returns multiple arguments, a :class:`tuple`
containing those
        arguments is returned instead. Please check the
        :ref:`documentation <discord-api-events>` for a list of events and their
        parameters.

        This function returns the **first event that meets the requirements**.

        Examples
        ---------

        Waiting for a user reply: ::

            @client.event
            async def on_message(message):
                if message.content.startswith('$greet'):
                    channel = message.channel
                    await channel.send('Say hello!')

                    def check(m):
                        return m.content == 'hello' and m.channel == channel

                    msg = await client.wait_for('message', check=check)
                    await channel.send(f'Hello {msg.author}!')

        Waiting for a thumbs up reaction from the message author: ::

            @client.event
            async def on_message(message):
                if message.content.startswith('$thumb'):
                    channel = message.channel
                    await channel.send('Send me that \N{THUMBS UP SIGN}
reaction, mate')

                    def check(reaction, user):
                        return user == message.author and str(reaction.emoji) ==
'\N{THUMBS UP SIGN}'

                    try:
                        reaction, user = await client.wait_for('reaction_add',
timeout=60.0, check=check)
                    except asyncio.TimeoutError:
                        await channel.send('\N{THUMBS DOWN SIGN}')
                    else:
                        await channel.send('\N{THUMBS UP SIGN}')

        .. versionchanged:: 2.0

            ``event`` parameter is now positional-only.
```

```
        Parameters
        ------------
        event: :class:`str`
            The event name, similar to the :ref:`event reference <discord-api-
events>`,
            but without the ``on_`` prefix, to wait for.
        check: Optional[Callable[..., :class:`bool`]]
            A predicate to check what to wait for. The arguments must meet the
            parameters of the event being waited for.
        timeout: Optional[:class:`float`]
            The number of seconds to wait before timing out and raising
            :exc:`asyncio.TimeoutError`.

        Raises
        -------
        asyncio.TimeoutError
            If a timeout is provided and it was reached.

        Returns
        --------
        Any
            Returns no arguments, a single argument, or a :class:`tuple` of
multiple
            arguments that mirrors the parameters passed in the
            :ref:`event reference <discord-api-events>`.
        """

        future = self.loop.create_future()
        if check is None:

            def _check(*args):
                return True

            check = _check

        ev = event.lower()
        try:
            listeners = self._listeners[ev]
        except KeyError:
            listeners = []
            self._listeners[ev] = listeners

        listeners.append((future, check))
        return asyncio.wait_for(future, timeout)

    # event registration

    def event(self, coro: Coro, /) -> Coro:
        """A decorator that registers an event to listen to.

        You can find more info about the events on the :ref:`documentation below
<discord-api-events>`.

        The events must be a :ref:`coroutine <coroutine>`, if not,
:exc:`TypeError` is raised.

        Example
        ---------

        .. code-block:: python3

            @client.event
            async def on_ready():
                print('Ready!')

        .. versionchanged:: 2.0

            ``coro`` parameter is now positional-only.
```

```
        Raises
        --------
        TypeError
            The coroutine passed is not actually a coroutine.
        """

        if not asyncio.iscoroutinefunction(coro):
            raise TypeError('event registered must be a coroutine function')

        setattr(self, coro.__name__, coro)
        _log.debug('%s has successfully been registered as an event',
coro.__name__)
        return coro

    async def change_presence(
        self,
        *,
        activity: Optional[BaseActivity] = None,
        status: Optional[Status] = None,
    ) -> None:
        """|coro|

        Changes the client's presence.

        Example
        ---------

        .. code-block:: python3

            game = discord.Game("with the API")
            await client.change_presence(status=discord.Status.idle,
activity=game)

        .. versionchanged:: 2.0
            Removed the ``afk`` keyword-only parameter.

        .. versionchanged:: 2.0
            This function will now raise :exc:`TypeError` instead of
            ``InvalidArgument``.

        Parameters
        ----------
        activity: Optional[:class:`.BaseActivity`]
            The activity being done. ``None`` if no currently active activity is
done.
        status: Optional[:class:`.Status`]
            Indicates what status to change to. If ``None``, then
            :attr:`.Status.online` is used.

        Raises
        ------
        TypeError
            If the ``activity`` parameter is not the proper type.
        """

        if status is None:
            status_str = 'online'
            status = Status.online
        elif status is Status.offline:
            status_str = 'invisible'
            status = Status.offline
        else:
            status_str = str(status)

        await self.ws.change_presence(activity=activity, status=status_str)

        for guild in self._connection.guilds:
            me = guild.me
```

```
            if me is None:
                continue

            if activity is not None:
                me.activities = (activity,)  # type: ignore # Type checker does
not understand the downcast here
            else:
                me.activities = ()

            me.status = status

    # Guild stuff

    async def fetch_guilds(
        self,
        *,
        limit: Optional[int] = 200,
        before: Optional[SnowflakeTime] = None,
        after: Optional[SnowflakeTime] = None,
    ) -> AsyncIterator[Guild]:
        """Retrieves an :term:`asynchronous iterator` that enables receiving
your guilds.

        .. note::

            Using this, you will only receive :attr:`.Guild.owner`,
:attr:`.Guild.icon`,
            :attr:`.Guild.id`, and :attr:`.Guild.name` per :class:`.Guild`.

        .. note::

            This method is an API call. For general usage, consider
:attr:`guilds` instead.

        Examples
        ---------

        Usage ::

            async for guild in client.fetch_guilds(limit=150):
                print(guild.name)

        Flattening into a list ::

            guilds = [guild async for guild in client.fetch_guilds(limit=150)]
            # guilds is now a list of Guild...

        All parameters are optional.

        Parameters
        -----------
        limit: Optional[:class:`int`]
            The number of guilds to retrieve.
            If ``None``, it retrieves every guild you have access to. Note,
however,
            that this would make it a slow operation.
            Defaults to ``200``.

            .. versionchanged:: 2.0

                The default has been changed to 200.

        before: Union[:class:`.abc.Snowflake`, :class:`datetime.datetime`]
            Retrieves guilds before this date or object.
            If a datetime is provided, it is recommended to use a UTC aware
datetime.
            If the datetime is naive, it is assumed to be local time.
        after: Union[:class:`.abc.Snowflake`, :class:`datetime.datetime`]
            Retrieve guilds after this date or object.
```

```
            If a datetime is provided, it is recommended to use a UTC aware
datetime.
            If the datetime is naive, it is assumed to be local time.

        Raises
        ------
        HTTPException
            Getting the guilds failed.

        Yields
        --------
        :class:`.Guild`
            The guild with the guild data parsed.
        """

        async def _before_strategy(retrieve: int, before: Optional[Snowflake],
limit: Optional[int]):
            before_id = before.id if before else None
            data = await self.http.get_guilds(retrieve, before=before_id)

            if data:
                if limit is not None:
                    limit -= len(data)

                before = Object(id=int(data[0]['id']))

            return data, before, limit

        async def _after_strategy(retrieve: int, after: Optional[Snowflake],
limit: Optional[int]):
            after_id = after.id if after else None
            data = await self.http.get_guilds(retrieve, after=after_id)

            if data:
                if limit is not None:
                    limit -= len(data)

                after = Object(id=int(data[-1]['id']))

            return data, after, limit

        if isinstance(before, datetime.datetime):
            before = Object(id=time_snowflake(before, high=False))
        if isinstance(after, datetime.datetime):
            after = Object(id=time_snowflake(after, high=True))

        predicate: Optional[Callable[[GuildPayload], bool]] = None
        strategy, state = _after_strategy, after

        if before:
            strategy, state = _before_strategy, before

        if before and after:
            predicate = lambda m: int(m['id']) > after.id

        while True:
            retrieve = min(200 if limit is None else limit, 200)
            if retrieve < 1:
                return

            data, state, limit = await strategy(retrieve, state, limit)

            # Terminate loop on next iteration; there's no data left after this
            if len(data) < 200:
                limit = 0

            if predicate:
                data = filter(predicate, data)
```

```
            for raw_guild in data:
                yield Guild(state=self._connection, data=raw_guild)

    async def fetch_template(self, code: Union[Template, str]) -> Template:
        """|coro|

        Gets a :class:`.Template` from a discord.new URL or code.

        Parameters
        -----------
        code: Union[:class:`.Template`, :class:`str`]
            The Discord Template Code or URL (must be a discord.new URL).

        Raises
        -------
        NotFound
            The template is invalid.
        HTTPException
            Getting the template failed.

        Returns
        --------
        :class:`.Template`
            The template from the URL/code.
        """
        code = utils.resolve_template(code)
        data = await self.http.get_template(code)
        return Template(data=data, state=self._connection)

    async def fetch_guild(self, guild_id: int, /, *, with_counts: bool = True) -
> Guild:
        """|coro|

        Retrieves a :class:`.Guild` from an ID.

        .. note::

            Using this, you will **not** receive :attr:`.Guild.channels`,
:attr:`.Guild.members`,
            :attr:`.Member.activity` and :attr:`.Member.voice` per
:class:`.Member`.

        .. note::

            This method is an API call. For general usage, consider
:meth:`get_guild` instead.

        .. versionchanged:: 2.0

            ``guild_id`` parameter is now positional-only.

        Parameters
        -----------
        guild_id: :class:`int`
            The guild's ID to fetch from.
        with_counts: :class:`bool`
            Whether to include count information in the guild. This fills the
            :attr:`.Guild.approximate_member_count` and
:attr:`.Guild.approximate_presence_count`
            attributes without needing any privileged intents. Defaults to
``True``.

            .. versionadded:: 2.0

        Raises
        ------
        Forbidden
            You do not have access to the guild.
        HTTPException
```

```
            Getting the guild failed.

        Returns
        --------
        :class:`.Guild`
            The guild from the ID.
        """
        data = await self.http.get_guild(guild_id, with_counts=with_counts)
        return Guild(data=data, state=self._connection)

    async def create_guild(
        self,
        *,
        name: str,
        icon: bytes = MISSING,
        code: str = MISSING,
    ) -> Guild:
        """|coro|

        Creates a :class:`.Guild`.

        Bot accounts in more than 10 guilds are not allowed to create guilds.

        .. versionchanged:: 2.0
            ``name`` and ``icon`` parameters are now keyword-only. The
``region`` parameter has been removed.

        .. versionchanged:: 2.0
            This function will now raise :exc:`ValueError` instead of
            ``InvalidArgument``.

        Parameters
        ----------
        name: :class:`str`
            The name of the guild.
        icon: Optional[:class:`bytes`]
            The :term:`py:bytes-like object` representing the icon. See
:meth:`.ClientUser.edit`
            for more details on what is expected.
        code: :class:`str`
            The code for a template to create the guild with.

            .. versionadded:: 1.4

        Raises
        ------
        HTTPException
            Guild creation failed.
        ValueError
            Invalid icon image format given. Must be PNG or JPG.

        Returns
        -------
        :class:`.Guild`
            The guild created. This is not the same guild that is
            added to cache.
        """
        if icon is not MISSING:
            icon_base64 = utils._bytes_to_base64_data(icon)
        else:
            icon_base64 = None

        if code:
            data = await self.http.create_from_template(code, name, icon_base64)
        else:
            data = await self.http.create_guild(name, icon_base64)
        return Guild(data=data, state=self._connection)

    async def fetch_stage_instance(self, channel_id: int, /) -> StageInstance:
```

```
        """|coro|

        Gets a :class:`.StageInstance` for a stage channel id.

        .. versionadded:: 2.0

        Parameters
        -----------
        channel_id: :class:`int`
            The stage channel ID.

        Raises
        -------
        NotFound
            The stage instance or channel could not be found.
        HTTPException
            Getting the stage instance failed.

        Returns
        --------
        :class:`.StageInstance`
            The stage instance from the stage channel ID.
        """
        data = await self.http.get_stage_instance(channel_id)
        guild = self.get_guild(int(data['guild_id']))
        # Guild can technically be None here but this is being explicitly
silenced right now.
        return StageInstance(guild=guild, state=self._connection, data=data)  #
type: ignore

    # Invite management

    async def fetch_invite(
        self,
        url: Union[Invite, str],
        *,
        with_counts: bool = True,
        with_expiration: bool = True,
        scheduled_event_id: Optional[int] = None,
    ) -> Invite:
        """|coro|

        Gets an :class:`.Invite` from a discord.gg URL or ID.

        .. note::

            If the invite is for a guild you have not joined, the guild and
channel
            attributes of the returned :class:`.Invite` will be
:class:`.PartialInviteGuild` and
            :class:`.PartialInviteChannel` respectively.

        Parameters
        -----------
        url: Union[:class:`.Invite`, :class:`str`]
            The Discord invite ID or URL (must be a discord.gg URL).
        with_counts: :class:`bool`
            Whether to include count information in the invite. This fills the
            :attr:`.Invite.approximate_member_count` and
:attr:`.Invite.approximate_presence_count`
            fields.
        with_expiration: :class:`bool`
            Whether to include the expiration date of the invite. This fills the
            :attr:`.Invite.expires_at` field.

            .. versionadded:: 2.0
        scheduled_event_id: Optional[:class:`int`]
            The ID of the scheduled event this invite is for.
```

```
            .. note::

                It is not possible to provide a url that contains an
``event_id`` parameter
                when using this parameter.

            .. versionadded:: 2.0

        Raises
        -------
        ValueError
            The url contains an ``event_id``, but ``scheduled_event_id`` has
also been provided.
        NotFound
            The invite has expired or is invalid.
        HTTPException
            Getting the invite failed.

        Returns
        --------
        :class:`.Invite`
            The invite from the URL/ID.
        """

        resolved = utils.resolve_invite(url)

        if scheduled_event_id and resolved.event:
            raise ValueError('Cannot specify scheduled_event_id and contain an
event_id in the url.')

        scheduled_event_id = scheduled_event_id or resolved.event

        data = await self.http.get_invite(
            resolved.code,
            with_counts=with_counts,
            with_expiration=with_expiration,
            guild_scheduled_event_id=scheduled_event_id,
        )
        return Invite.from_incomplete(state=self._connection, data=data)

    async def delete_invite(self, invite: Union[Invite, str], /) -> None:
        """|coro|

        Revokes an :class:`.Invite`, URL, or ID to an invite.

        You must have the :attr:`~.Permissions.manage_channels` permission in
        the associated guild to do this.

        .. versionchanged:: 2.0

            ``invite`` parameter is now positional-only.

        Parameters
        ----------
        invite: Union[:class:`.Invite`, :class:`str`]
            The invite to revoke.

        Raises
        -------
        Forbidden
            You do not have permissions to revoke invites.
        NotFound
            The invite is invalid or expired.
        HTTPException
            Revoking the invite failed.
        """

        resolved = utils.resolve_invite(invite)
        await self.http.delete_invite(resolved.code)
```

```
    # Miscellaneous stuff

    async def fetch_widget(self, guild_id: int, /) -> Widget:
        """|coro|

        Gets a :class:`.Widget` from a guild ID.

        .. note::

            The guild must have the widget enabled to get this information.

        .. versionchanged:: 2.0

            ``guild_id`` parameter is now positional-only.

        Parameters
        -----------
        guild_id: :class:`int`
            The ID of the guild.

        Raises
        -------
        Forbidden
            The widget for this guild is disabled.
        HTTPException
            Retrieving the widget failed.

        Returns
        --------
        :class:`.Widget`
            The guild's widget.
        """
        data = await self.http.get_widget(guild_id)

        return Widget(state=self._connection, data=data)

    async def application_info(self) -> AppInfo:
        """|coro|

        Retrieves the bot's application information.

        Raises
        -------
        HTTPException
            Retrieving the information failed somehow.

        Returns
        --------
        :class:`.AppInfo`
            The bot's application information.
        """
        data = await self.http.application_info()
        if 'rpc_origins' not in data:
            data['rpc_origins'] = None
        return AppInfo(self._connection, data)

    async def fetch_user(self, user_id: int, /) -> User:
        """|coro|

        Retrieves a :class:`~discord.User` based on their ID.
        You do not have to share any guilds with the user to get this
information,
        however many operations do require that you do.

        .. note::
```

```
            This method is an API call. If you have
:attr:`discord.Intents.members` and member cache enabled, consider
:meth:`get_user` instead.

        .. versionchanged:: 2.0

            ``user_id`` parameter is now positional-only.

        Parameters
        -----------
        user_id: :class:`int`
            The user's ID to fetch from.

        Raises
        -------
        NotFound
            A user with this ID does not exist.
        HTTPException
            Fetching the user failed.

        Returns
        --------
        :class:`~discord.User`
            The user you requested.
        """
        data = await self.http.get_user(user_id)
        return User(state=self._connection, data=data)

    async def fetch_channel(self, channel_id: int, /) -> Union[GuildChannel,
PrivateChannel, Thread]:
        """|coro|

        Retrieves a :class:`.abc.GuildChannel`, :class:`.abc.PrivateChannel`, or
:class:`.Thread` with the specified ID.

        .. note::

            This method is an API call. For general usage, consider
:meth:`get_channel` instead.

        .. versionadded:: 1.2

        .. versionchanged:: 2.0

            ``channel_id`` parameter is now positional-only.

        Raises
        -------
        InvalidData
            An unknown channel type was received from Discord.
        HTTPException
            Retrieving the channel failed.
        NotFound
            Invalid Channel ID.
        Forbidden
            You do not have permission to fetch this channel.

        Returns
        --------
        Union[:class:`.abc.GuildChannel`, :class:`.abc.PrivateChannel`,
:class:`.Thread`]
            The channel from the ID.
        """
        data = await self.http.get_channel(channel_id)

        factory, ch_type = _threaded_channel_factory(data['type'])
        if factory is None:
            raise InvalidData('Unknown channel type {type} for channel ID
{id}.'.format_map(data))
```

```
        if ch_type in (ChannelType.group, ChannelType.private):
            # the factory will be a DMChannel or GroupChannel here
            channel = factory(me=self.user, data=data,
state=self._connection)  # type: ignore
        else:
            # the factory can't be a DMChannel or GroupChannel here
            guild_id = int(data['guild_id'])  # type: ignore
            guild = self._connection._get_or_create_unavailable_guild(guild_id)
            # the factory should be a GuildChannel or Thread
            channel = factory(guild=guild, state=self._connection, data=data)  #
type: ignore

        return channel

    async def fetch_webhook(self, webhook_id: int, /) -> Webhook:
        """|coro|

        Retrieves a :class:`.Webhook` with the specified ID.

        .. versionchanged:: 2.0

            ``webhook_id`` parameter is now positional-only.

        Raises
        --------
        HTTPException
            Retrieving the webhook failed.
        NotFound
            Invalid webhook ID.
        Forbidden
            You do not have permission to fetch this webhook.

        Returns
        ---------
        :class:`.Webhook`
            The webhook you requested.
        """
        data = await self.http.get_webhook(webhook_id)
        return Webhook.from_state(data, state=self._connection)

    async def fetch_sticker(self, sticker_id: int, /) -> Union[StandardSticker,
GuildSticker]:
        """|coro|

        Retrieves a :class:`.Sticker` with the specified ID.

        .. versionadded:: 2.0

        Raises
        --------
        HTTPException
            Retrieving the sticker failed.
        NotFound
            Invalid sticker ID.

        Returns
        --------
        Union[:class:`.StandardSticker`, :class:`.GuildSticker`]
            The sticker you requested.
        """
        data = await self.http.get_sticker(sticker_id)
        cls, _ = _sticker_factory(data['type'])
        # The type checker is not smart enough to figure out the constructor is
correct
        return cls(state=self._connection, data=data)  # type: ignore

    async def fetch_premium_sticker_packs(self) -> List[StickerPack]:
        """|coro|
```

```
        Retrieves all available premium sticker packs.

        .. versionadded:: 2.0

        Raises
        -------
        HTTPException
            Retrieving the sticker packs failed.

        Returns
        ---------
        List[:class:`.StickerPack`]
            All available premium sticker packs.
        """
        data = await self.http.list_premium_sticker_packs()
        return [StickerPack(state=self._connection, data=pack) for pack in
data['sticker_packs']]

    async def create_dm(self, user: Snowflake) -> DMChannel:
        """|coro|

        Creates a :class:`.DMChannel` with this user.

        This should be rarely called, as this is done transparently for most
        people.

        .. versionadded:: 2.0

        Parameters
        -----------
        user: :class:`~discord.abc.Snowflake`
            The user to create a DM with.

        Returns
        -------
        :class:`.DMChannel`
            The channel that was created.
        """
        state = self._connection
        found = state._get_private_channel_by_user(user.id)
        if found:
            return found

        data = await state.http.start_private_message(user.id)
        return state.add_dm_channel(data)

    def add_view(self, view: View, *, message_id: Optional[int] = None) -> None:
        """Registers a :class:`~discord.ui.View` for persistent listening.

        This method should be used for when a view is comprised of components
        that last longer than the lifecycle of the program.

        .. versionadded:: 2.0

        Parameters
        ------------
        view: :class:`discord.ui.View`
            The view to register for dispatching.
        message_id: Optional[:class:`int`]
            The message ID that the view is attached to. This is currently used
to
            refresh the view's state during message update events. If not given
            then message update events are not propagated for the view.

        Raises
        -------
        TypeError
            A view was not passed.
```

```
        ValueError
            The view is not persistent. A persistent view has no timeout
            and all their components have an explicitly provided custom_id.
        """

        if not isinstance(view, View):
            raise TypeError(f'expected an instance of View not
{view.__class__!r}')

        if not view.is_persistent():
            raise ValueError('View is not persistent. Items need to have a
custom_id set and View must have no timeout')

        self._connection.store_view(view, message_id)

    @property
    def persistent_views(self) -> Sequence[View]:
        """Sequence[:class:`.View`]: A sequence of persistent views added to the
client.

        .. versionadded:: 2.0
        """
        return self._connection.persistent_views
```