

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Інтерактивний Discord-бот для налаштування серверу та комфортного користування спільнотою

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»


(підпис)

Дірка Іллі Романовича

(прізвище, ім'я, по батькові)

Науковий
керівник


(підпис)

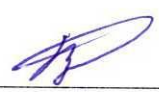
Грищенко Світлана

Миколаївна

канд.пед.наук.,ст.досл.

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Ніжегородцев Владислав

Олександрович, канд. пед.

наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «12» травня 2026 № 20

Завідувач кафедри


(підпис)

Філоненко Михайло

Миколайович, канд. фіз.-мат.

наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Реєстрація № 41

24.05.2026

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

12.05.2026

(дата)



(підпис)

Результат захисту

4 (добре)
(оцінка)

08.06.2026

(дата)



(підпис)

Голова комісії:



(підпис)

О.С. Дюбанов

Члени комісії:



(підпис)

М.М. Філоненко



(підпис)

В.О. Ніжегородцев



(підпис)

А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	26		08.06.2026 Грищенко С.М. Дюбанов О.С.
2. Дотримання вимог щодо оформлення	від 0 до 10	7		Філоненко М.М. Ніжегородцев В.О.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Омельчук А.А. Суховерхівка Л.П.
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	42		

Оцінювання кваліфікаційної роботи науковим керівником – 42 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підписи)
84	B	4 (добре)	08.06.2026	Дюбанов О.С. Філоненко М.М. Ніжегородцев В.О. Омельчук А.А. Суховерхівка Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	3 (задовільно)
60-69	D	2 (незадовільно)
50-59	E	
35-49	FX	

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри  Філоненко М.М.
«01» грудня 2025

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Дірку Іллі Романовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Інтерактивний Discord-бот для налаштування серверу та комфортного користування спільнотою

Затверджена наказом ректора від «28» листопада 2025 № 1526

2. Строк здачі закінченої роботи «08» травня 2026

3. Мета роботи Розробка Discord-бота з веб-панеллю для налаштування серверу та комфортного користування спільнотою

Об'єкт дослідження Процеси автоматизації адміністрування в інтерактивних онлайн-спільнотах

Предмет дослідження Методи та програмні засоби створення інтерактивного Discord-бота та веб-орієнтованої панелі керування для менеджменту користувачів і системних налаштувань

Орієнтовна база дослідження Тестовий Discord-сервер

4. Вихідні дані до роботи Сучасні технології розробки Discord-ботів(discord.py), веб-додатків(Flask), роботи з базами даних(SQLAlchemy), а також аналіз функціоналу популярних Discord-ботів модерації

5. Перелік графічного матеріалу

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

Розділ 1. Аналіз предметної області та проектування системи

Розділ 2 Розробка та реалізація інтерактивного Discord-бота

Розділ 3 Тестування, впровадження та ефективність розробленого рішення

Висновки

Список використаних джерел

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2025	До 30.10.2025
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.12.2025	26.12.2025
3	II розділ (аналіз експериментальної частини про методики досліджень)	18.02.2026	18.02.2026
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2026	16.04.2026
5	Висновки	01.05.2026	01.05.2026
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	07.05.2026	07.05.2026
7	Отримання відгуку керівника	29.05.2026	29.05.2026
8	Отримання зовнішньої рецензії	01.06.2026	01.06.2026

8. Дата видачі завдання «01» грудня 2025

9. Керівник кваліфікаційної роботи

10. Керівник освітньо-професійної програми

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Грищенко С.М.
(прізвище, ініціали)


(підпис)

Ніжегородцев В.О.
(прізвище, ініціали)


(підпис)

Дірко І.Р.
(прізвище, ініціали)

АНОТАЦІЯ

Розробка Інтерактивний Discord-bot для налаштування серверу та комфортного користування спільнотою

Мета роботи полягає в розробці інтерактивного Discord-бота з веб адмін панеллю для автоматизації процесів модерації, підвищення комфорту користування спільнотою та ефективного управління Discord-серверами.

Завданнями роботи є:

- Проведення аналізу сучасних ботів та виявлення їхніх недоліків.
- Проектування архітектури ПЗ з урахуванням модульності.
- Реалізація ключових модулів системи.
- Розробка зручної адмін панелі для управління ботом.
- Забезпечення двомовності, безпека та тестування рішення.

Предметом дослідження є програмне забезпечення.

Об'єктом дослідження виступають процеси модерації.

Базою дослідження став тестовий Discord-сервер.

У результаті роботи створено комплексне рішення, яке поєднує потужний Discord-бот та сучасну веб-адмінпанель.

Одержані результати можуть бути використані для ефективного управління як невеликими студентськими, так і великими ігровими та освітніми Discord-спільнотами.

Кваліфікаційна робота містить 69 сторінок, 2 таблиці, 6 рисунків, список використаних джерел 45 найменувань, 2 додатка.

Ключові слова: Discord-бот, модерація, веб-адмін панель, анти-спам системи XP, апеляції, Python, Flask, discord.py, SQLAlchemy.

Рік виконання кваліфікаційної роботи 2026

Рік захисту роботи 2026

ABSTRACT

Development of an Interactive Discord Bot for Server Setup and Enhanced Community Experience

The aim of this work is to develop an interactive Discord bot with a web admin panel to automate moderation processes, enhance the user experience within the community, and enable effective management of Discord servers..

The tasks of the work are as follows:

- Analyzing existing bots and identifying their shortcomings..
- Designing the software architecture with a focus on modularity.
- Implementing key system modules.
- Developing a user-friendly admin panel for bot management.
- Ensuring multilingual support, security, and testing the solution..

The subject of the research is software.

The object of the research is moderation processes.

The research is based on the test Discord server.

The results of the research include a comprehensive solution was created that combines a powerful Discord bot and a modern web admin panel.

The obtained results can be used for the effective management of both small student communities and large gaming and educational Discord communities.

The qualification work consists of 69 pages, 2 tables, 6 images, a list of 45 references, and 2 appendices.

Keywords: Discord bot, moderation, web admin panel, XP anti-spam system, appeals, Python, Flask, discord.py, SQLAlchemy.

Year of qualification work: 2026

Year of work defense: 2026

ЗМІСТ

Вступ.....	9
Розділ 1. Аналіз предметної області та проєктування системи.....	13
1.1 Огляд сучасних Discord-ботів та системи модерації спільнот.....	13
1.2 Проєктування архітектури інтерактивного Discord-бота з веб-панеллю.....	19
Розділ 2. Розробка та реалізація інтерактивного Discord-бота.....	30
2.1 Реалізація основної логіки бота та модулів модерації.....	30
2.2 Розробка веб-адмін панелі та інтеграція з ботом.....	41
Розділ 3. Тестування, впровадження та ефективність розробленого рішення.....	52
Висновки.....	58
Список використаних джерел.....	61
Додатки.....	66

ВСТУП

Сучасний розвиток цифрових технологій призвів до стрімкого зростання онлайн-спільнот, зокрема на платформі Discord[1]. Створений спочатку як інструмент для голосового спілкування геймерів, Discord швидко трансформувався в універсальну платформу для створення спільнот різного спрямування: ігрових, освітніх, творчих, професійних та студентських. За даними статистики, у період з 2022 по 2026 рік кількість зареєстрованих користувачів Discord зростає з приблизно 450-560 мільйонів до понад 650-750 мільйонів облікових записів, а щомісячна активна аудиторія перевищила 200-259 мільйонів осіб[1]. Кількість активних серверів сягнула понад 32 мільйони. Такий вибуховий ріст супроводжується значним збільшенням обсягів спілкування – щодня в Discord надсилається мільярди повідомлень.

Разом із кількісним зростанням спільнот загострилися й типові проблеми управління ними. Великі та середні сервери стикаються з масовим спамом, флудом, повторюваними повідомленнями, використанням забороненої лексики (в тому числі українською мовою), токсичною поведінкою та рейдами[2]. Ручна модерація в таких умовах стає надзвичайно трудомісткою та неефективною. Модератори часто не встигають реагувати на порушення в режимі реального часу, що призводить до зниження комфорту учасників, відтоку активних користувачів та погіршення загальної атмосфери спільноти. Крім того, традиційні Discord-боти, хоча й допомагають автоматизувати окремі процеси, здебільшого мають обмежений функціонал, відсутність зручної веб-панелі управління, слабку підтримку двомовності та недостатню інтеграцію систем gamification з інструментами модерації.

Актуальність теми кваліфікаційної роботи зумовлена необхідністю створення сучасного, комплексного інструменту, який би поєднував потужні механізми автоматизованої модерації з елементами підвищення залученості

користувачів та зручним веб-інтерфейсом для адміністраторів і модераторів[18]. Особливо важливим є розробка рішень, орієнтованих на україномовні та двомовні спільноти, оскільки більшість популярних ботів недостатньо адаптовані до специфіки української мови та національних особливостей спілкування[20].

Мета кваліфікаційної роботи полягає в розробці інтерактивного Discord-бота з веб-адмінпанеллю для автоматизації процесів модерації, підвищення комфорту користування спільнотою та ефективного управління Discord-сервером.

Для досягнення поставленої мети визначено такі основні завдання:

- проаналізувати сучасні рішення Discord-ботів та існуючі веб-панелі управління спільнотами, виявити їхні переваги та недоліки;
- спроектувати архітектуру програмного забезпечення, що включає модульну структуру бота та інтеграцію backend-частини на базі Flask з Discord API;
- реалізувати ключові функціональні модулі: автоматичне привітання нових учасників, систему попереджень і банів, розширений анти-спам, фільтр заборонених слів, систему досвіду (XP) та рівнів, а також механізм подання та розгляду апеляцій;
- розробити зручну веб-адмінпанель для перегляду журналу дій, управління попередженнями та обробки апеляцій;
- провести комплексне тестування розробленого програмного забезпечення та впровадити його на практичній базі.

Об'єктом дослідження є процеси модерації та взаємодії учасників у Discord-серверах[17].

Предметом дослідження виступає програмне забезпечення для автоматизованої модерації та ефективного управління спільнотами на платформі Discord.

Базою впровадження роботи став тестовий Discord-сервер.

У процесі роботи були використані такі методи дослідження: системний аналіз існуючих рішень, структурне та функціональне проектування (зокрема, діаграми UML та ER-моделі), програмна реалізація на мові Python з використанням бібліотек discord.py, Flask та SQLAlchemy, а також методи тестування (unit-тестування, інтеграційне тестування та ручне тестування на реальному сервері)[2,3,4,7].

Наукова новизна розробки полягає в комплексному поєднанні в межах одного рішення потужних механізмів модерації (розширена система анти-спаму з виявленням флуду, дублікатів та CAPS-спаму, інтелектуальний фільтр заборонених слів з підтримкою нормалізації та заміни символів для української та англійської мов), системи попереджень з автоматичним баном, механізму апеляцій через веб-панель та gamification-елементів у вигляді системи досвіду, рівнів і автоматичної видачі ролей[19,33]. Додатковою відмінністю є наявність повноцінної веб-адмінпанелі з можливістю перегляду логів, управління попередженнями та обробки апеляцій у реальному часі, що значно підвищує зручність роботи модераторів порівняно з більшістю існуючих аналогів.

Практичне значення роботи визначається можливістю використання розробленого бота та веб-панелі для ефективного управління будь-якими Discord-спільнотами – від невеликих студентських груп до великих ігрових та освітніх серверів[22,35]. Рішення дозволяє суттєво зменшити навантаження на модераторів, знизити рівень токсичності та спаму, підвищити залученість учасників завдяки системі XP та забезпечити прозорість модераційних дій завдяки централізованому журналу.

Структура кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної області та проектування архітектури системи. Другий розділ присвячено безпосередній реалізації бота та веб-панелі. У третьому розділі представлено результати тестування, впровадження та оцінку ефективності розробленого рішення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЄКТУВАННЯ СИСТЕМИ

1.1 Огляд сучасних Discord-ботів та системи модерації спільнот

Сучасні онлайн-спільноти, особливо на платформі Discord, стрімко розвиваються і стають все більш численними та активними. Згідно з даними Discord за 2025–2026 роки, щомісяця платформою користується понад 200 мільйонів активних користувачів, а загальна кількість серверів перевищує 32 мільйони. Таке масштабне зростання супроводжується значним збільшенням навантаження на адміністраторів і модераторів. Управління великими спільнотами вручну стає практично неможливим через високий обсяг повідомлень, швидкість їх надходження та різноманітність порушень.

Проблеми сучасних Discord-серверів

Однією з найгостріших проблем є масовий спам і флуд[30]. Користувачі часто надсилають десятки однакових або схожих повідомлень за короткий проміжок часу, що засмічує канали та ускладнює нормальне спілкування. Не менш серйозною проблемою є токсична мова – використання ненормативної лексики, образливих висловів та сленгу. Особливо складно модерувати двомовні спільноти, оскільки більшість готових рішень погано працюють з українською мовою.

Іншою поширеною проблемою є відсутність ефективної системи попереджень і апеляцій. У багатьох серверах модератори змушені вручну вести облік порушень, що призводить до суб'єктивних рішень і конфліктів. Крім того, забанені користувачі часто не мають зручного механізму подання апеляції, що знижує довіру до адміністрації сервера.

Значну проблему становить і відсутність зручного веб-інтерфейсу для модераторів[11]. Більшість ботів керуються виключно через текстові команди в

Discord, що є незручним при роботі з великою кількістю логів, попереджень і апеляцій. Все це свідчить про нагальну потребу у комплексних рішеннях, які поєднували б потужну автоматизовану модерацію з сучасним веб-інтерфейсом управління.

Аналіз популярних Discord-ботів

На ринку існує чимало готових рішень для модерації Discord-серверів. Серед найбільш популярних варто виділити Carl-bot, MEE6, Dyno, Wick та AltIdentifier .

Carl-bot вважається одним з найпотужніших ботів для модерації. Він пропонує розширену систему реакцій, автоматичне видалення повідомлень, складні фільтри та систему логів. Сильною стороною є гнучкість налаштувань через дашборд[20]. Однак Carl-bot має суттєвий недолік – його повна версія є платною, а безкоштовна значно обмежена у функціоналі. Крім того, підтримка української мови в фільтрах майже відсутня.

MEE6 – один з найпопулярніших ботів завдяки зручному веб-дашборду та системі рівнів (levelling system). Він добре підходить для gamification спільнот[21]. Проте його модераційні можливості значно слабші порівняно з конкурентами. Анти-спам система базова, а фільтр заборонених слів працює переважно з англійською мовою. Крім того, багато функцій доступні тільки в преміум-версії.

Dyno спеціалізується на автоматизованій модерації. Має потужну систему автоматичних попереджень, таймаутів і банів, а також зручний дашборд[23]. Сильною стороною є швидкість реакції на порушення. Водночас Dyno майже не підтримує складні кастомні фільтри слів, особливо для української мови, і має обмежену систему апеляцій.

Wick позиціонується як анти-рейд бот. Він добре виявляє підозрілу активність нових акаунтів і масові запрошення. Однак його можливості щодо

повсякденної модерації (фільтр слів, анти-спам, система XP) досить обмежені. Веб-панель присутня, але не така зручна, як у конкурентів.

AltDentifier зосереджений на перевірці акаунтів на альт-акаунти та боти. Це корисний інструмент для великих спільнот, але він не вирішує проблеми щоденної модерації, токсичної мови та gamification.

Порівняльна характеристика існуючих рішень

Більшість популярних ботів мають певні сильні сторони, але водночас демонструють системні недоліки. По-перше, майже жоден з них не пропонує якісної підтримки української мови у фільтрах заборонених слів. По-друге, системи анти-спаму часто бувають надто прості або, навпаки, надто агресивні, викликаючи помилкові спрацьовування. По-третє, механізм апеляцій або відсутній, або реалізований вкрай незручно. І нарешті, поєднання потужної модерації з gamification-системою (XP та рівні) в одному рішенні зустрічається рідко.

Огляд технологічного стеку

Для створення сучасного Discord-бота з веб-панеллю важливо правильно обрати технологічний стек[5, 14].

discord.py є однією з найпопулярніших і найбільш підтримуваних бібліотек для роботи з Discord API на мові Python. Вона забезпечує асинхронну обробку подій, роботу з intents, embed-повідомленнями та командами.

Для створення веб-адмінпанелі відмінно підходить Flask – легковаговий і гнучкий веб-фреймворк. Він добре інтегрується з discord.py через паралельний запуск у різних потоках. SQLAlchemy у поєднанні з Flask-SQLAlchemy забезпечує зручну роботу з базою даних. ORM дозволяє працювати з моделями даних на високому рівні абстракції.

Для фронтенду веб-панелі – Bootstrap через CDN, що забезпечує сучасний вигляд, адаптивність та швидкість розробки без необхідності використання

складних JavaScript-фреймворків. Такий стек дозволяє створити масштабоване, зручне у підтримці та відносно легке у розробці рішення.

Для об'єктивної оцінки сучасних Discord-ботів було проведено порівняльний аналіз найпопулярніших рішень за ключовими критеріями: функціонал анти-спаму та фільтрації контенту, наявність системи досвіду та рівнів, якість веб-панелі управління, рівень підтримки української мови, цінова політика та основні обмеження (Табл. 1-2).

Таблиця 1.1. Порівняльна характеристика популярних ботів модерації

Бот	Анти-спам фільтр слів	Система XP Рівнів	Веб-панель	Підтримка української мови
Carl-bot	Відмінний	Базова	Добра	Слабка
MEE6	Середній	Дуже добра	Відмінна	Дуже слабка
Dyno	Добра	Відсутня	Добра	Слабка
Wick	Добрий	Відсутня	Середня	Відсутня
AltIdentifier	Середній	Відсутня	Середня	Відсутня

Таблиця 1.2. Основні обмеження ботів

Бот	Ціна (преміум)	Основні обмеження
Carl-bot	\$5-10 / місяць	Обмежений безкоштовний функціонал, слабка підтримка неанглійських мов
MEE6	\$6,49-11,99 / місяць	Більшість функцій тільки в преміум, слабкий анти-спам

Dyno	\$5-15 / місяць	Обмежені можливості gamification, слабкий фільтр української мови
Wick	\$4,99-9,99 / місяць	Слабка щоденна модерація, майже немає системи рівнів
AltDentifier	Безкоштовний + преміум	Вузька спеціалізація, не вирішує проблеми токсичної мови

Як видно із таблиць, жоден з розглянутих ботів не пропонує повноцінного поєднання всіх необхідних функцій в одному рішенні. Більшість ботів або добре справляються з модерацією, або з gamification, але рідко – з обома напрямками одночасно[36]. Особливо помітна проблема з підтримкою української мови: у всіх розглянутих рішеннях фільтрація токсичного контенту українською або відсутня, або працює на дуже низькому рівні.

Висновки щодо недоліків аналогів та необхідності власної розробки

Проведений аналіз сучасних Discord-ботів дозволяє зробити кілька важливих висновків. По-перше, більшість популярних рішень мають фрагментарний підхід до модерації. Вони добре вирішують окремі завдання (наприклад, анти-рейд у Wick або систему рівнів у MEE6), але не пропонують комплексного рішення, яке б одночасно включало потужний анти-спам, інтелектуальний фільтр заборонених слів для двох мов, систему попереджень з автоматичним баном, механізм апеляцій та систему gamification.

По-друге, критичним недоліком майже всіх розглянутих ботів є слабка або повна відсутність підтримки української мови у фільтрах токсичного контенту. У двомовних спільнотах це призводить до значних прогалин у модерації, оскільки українські матюки, сленг та їх варіації просто ігноруються більшістю готових рішень.

По-третє, веб-панелі управління, навіть коли вони присутні (як у МЕЕБ чи Carl-bot), часто мають обмежений функціонал. Вони дозволяють переглядати базову статистику, але не завжди дають змогу зручно працювати з журналом модерації, масово обробляти попередження чи розглядати апеляції користувачів. Крім того, багато важливих функцій доступні лише після оформлення платної підписки, що робить повноцінне використання таких ботів дорогим для невеликих і середніх спільнот.

По-четверте, майже в жодному з популярних рішень не реалізовано зручного та прозорого механізму апеляцій, інтегрованого з веб-панеллю. Забанений користувач зазвичай не має можливості подати апеляцію через зручний інтерфейс, а модератори не отримують сповіщення про нові апеляції в Discord.

Усі ці недоліки існуючих рішень обумовлюють актуальність і необхідність розробки власного Discord-бота. Створене в межах кваліфікаційної роботи програмне забезпечення повинно усунути виявлені прогалини шляхом комплексного підходу: поєднання потужної анти-спам системи (виявлення флуду, дублікатів та CAPS-спаму), двомовного фільтра заборонених слів з урахуванням нормалізації та заміни символів, гнучкої системи попереджень і банів, системи досвіду та рівнів для підвищення залученості учасників, а також сучасної веб-адмінпанелі для зручного управління всіма аспектами модерації.

Власна розробка дозволить не лише забезпечити якісну підтримку української мови, але й створити єдине рішення, яке буде повністю безкоштовним, масштабованим, прозорим та адаптованим під потреби українських і двомовних спільнот. Такий підхід дасть змогу суттєво підвищити ефективність модерації, знизити навантаження на модераторів і покращити загальний комфорт користування Discord-серверами.

Таким чином, проведений огляд і порівняльний аналіз підтверджують, що існуючі аналоги не повністю задовольняють потреби сучасних спільнот, що робить розробку нового інтерактивного Discord-бота з веб-панеллю обґрунтованою і перспективною задачею.

1.2 Проектування архітектури інтерактивного Discord-бота з веб-панеллю

Проектування архітектури програмного забезпечення є одним з найважливіших етапів розробки, оскільки від правильності прийнятих архітектурних рішень залежить масштабованість, надійність, зручність підтримки та подальше розширення системи[12,31]. У межах кваліфікаційної роботи була спроектована архітектура інтерактивного Discord-бота, який поєднує в собі функціональність модераторії Discord-сервера з сучасною веб-адмінпанеллю управління[25].

Загальна архітектура системи

При проектуванні системи було обрано гібридну архітектуру, яка поєднує елементи клієнт-серверної моделі та багатопотокової обробки. Основна ідея полягає в тому, щоб Discord-бот і веб-адмінпанель працювали як два відносно незалежні компоненти, але при цьому мали спільне сховище даних.

Центральним елементом архітектури виступає реляційна база даних, яка виконує роль єдиного джерела правди (Single Source of Truth)[24]. Через неї відбувається обмін даними між Discord-ботом і веб-панеллю. Такий підхід дозволяє уникнути прямих викликів між компонентами і забезпечує слабку зв'язність.

Discord-бот, реалізований на базі бібліотеки discord.py, відповідає за реальну взаємодію з користувачами платформи Discord: обробку подій, команд, привітань, модераторських дій та системи гейміфікації. Веб-адмінпанель, побудована на фреймворку Flask, призначена для зручного візуального

управління процесами модерації, перегляду журналів, роботи з попередженнями та апеляціями[6]. Для забезпечення одночасної роботи обох компонентів було спроектовано використання паралельних потоків виконання. Основний потік запускає Flask-сервер, а окремий демонічний потік – Discord-бота (Daemon Threads). Така архітектура дозволяє боту постійно підтримувати з'єднання з Discord API і реагувати на події в реальному часі, тоді як веб-сервер незалежно обробляє запити від модераторів.

Важливим архітектурним рішенням стала підтримка multi-guild архітектури. Усі ключові сутності в базі даних прив'язані до `server_id`, що дає можливість одному екземпляру бота обслуговувати декілька Discord-серверів одночасно без конфліктів даних. Це значно підвищує практичну цінність рішення для потенційного використання в різних спільнотах.

При проектуванні також було враховано принцип розділення відповідальностей[32]. Логіка взаємодії з Discord, бізнес-логіка модерації, робота з базою даних та презентаційний шар були максимально відокремлені один від одного[43]. Це спрощує тестування, відлагодження та можливе масштабування системи в майбутньому (наприклад, перехід від SQLite до PostgreSQL при великій кількості серверів). Загальна архітектура системи може бути представлена у вигляді діаграми компонентів, де чітко видно взаємозв'язки між Discord API, Discord-ботом, базою даних SQLite, Flask-додатком та веб-інтерфейсом користувача.

Модульна структура проєкту

На етапі проектування особлива увага приділялася модульній організації коду[12]. Було прийнято рішення будувати систему за принципом високої когезії та низької зв'язності модулів. Кожен модуль відповідає за виконання чітко визначеної функції, що дозволяє розробляти, тестувати та модифікувати окремі частини системи незалежно.

Основна модульна структура проекту включає такі ключові компоненти:

- **main.py** – центральний модуль, який виконує роль точки входу в систему. На етапі проектування було вирішено, що саме в цьому файлі буде здійснюватися ініціалізація та паралельний запуск двох основних процесів – Discord-бота та Flask-сервера.
- **bot.py** – основний модуль Discord-бота. Тут проектується вся логіка взаємодії з Discord API, обробка подій та команд. Цей модуль виступає як «мозок» бота з точки зору користувацького досвіду.
- **app.py** – модуль веб-додатка. У ньому проектується вся серверна частина веб-панелі: конфігурація Flask, маршрутизація запитів, інтеграція з базою даних та управління шаблонами.
- **models.py** – модуль визначення даних. Тут проектуються всі моделі предметної області за допомогою SQLAlchemy. Цей модуль є спільним для бота та веб-панелі, що забезпечує єдину структуру даних.
- **anti_spam.py** – спеціалізований модуль анти-спаму. Під час проектування було вирішено винести складну логіку виявлення спаму (флуд, дублікати, CAPS) в окремий модуль, щоб не перевантажувати основний файл бота.
- **word_filters.py** – модуль фільтрації заборонених слів. Проектувався як незалежний компонент з метою легкого розширення словників і алгоритмів нормалізації тексту для підтримки української та англійської мов.
- **xp_system.py** – модуль системи досвіду та рівнів. Був спроектований окремо, оскільки містить власну бізнес-логіку нарахування досвіду, розрахунку рівнів, прогресу та автоматичної видачі ролей.

Додатково були спроектовані допоміжні модулі: `update_db.py` для управління схемою бази даних та каталог `templates/` для зберігання HTML-

шаблонів веб-панелі[40]. Така модульна архітектура забезпечує декілька важливих переваг:

- можливість паралельної розробки різних частин системи;
- спрощення тестування окремих модулів;
- високу підтримуваність коду;
- зручність розширення функціональності (наприклад, додавання нових модулів модерації)[25].

Під час проектування також було враховано принципи чистого коду та дотримання стандартів PEP 8, що дозволить іншим розробникам швидко зрозуміти структуру проєкту.

ER-діаграма бази даних

Важливим етапом проектування системи стала розробка концептуальної та логічної моделі даних[4]. Для цього була створена ER-діаграма (Entity-Relationship Diagram), яка описує основні сутності системи та зв'язки між ними (Рис 1.1). База даних спроектована з урахуванням вимог multi-guild підтримки, тому майже всі сутності мають поле `server_id` як частину композитного ключа або зовнішнього ключа. Основними сутностями є:

- **ModLog** – журнал всіх модераційних дій (попередження, бани, видалення повідомлень, апеляції тощо).
- **UserWarning** – інформація про кількість попереджень користувача на конкретному сервері, статус бану.
- **UserLevel** – дані системи гейміфікації (XP, рівень, кількість повідомлень).
- **Appeal** – таблиця апеляцій забаних користувачів.
- **FilteredWord** – таблиця для зберігання та управління списком заборонених слів.

Опис зв'язків:

- Один користувач (`user_id` + `server_id`) може мати багато записів у `ModLog` (1:N).
- Один користувач може мати лише один активний запис у `UserWarning` на сервер (1:1 з унікальним обмеженням).
- Один користувач може мати лише один запис у `UserLevel` на сервер.
- Одна апеляція належить конкретному користувачу на конкретному сервері.
- Таблиця `ModLog` пов'язана з `Appeal` через `appeal_id`.



Рис. 1.1 ER-діаграма бази даних

Діаграми UML

Під час проектування системи була розроблена діаграма UML для візуалізації поведінки та взаємодії користувачів з системою (Рис 1.2). Ця діаграма показує основних акторів і доступні їм варіанти використання.

Актори:

- Модератор / Адміністратор
- Звичайний користувач
- Забанений користувач (для апеляції)

Основні прецеденти:

- Для модератора: Видати попередження, Зняти попередження, Забанити / Розбанити, Переглянути логи, Обробити апеляцію, Видати роль Moderator, Очистити канал.
- Для користувача: Отримати привітання, Надіслати повідомлення, Перевірити свій ранг (/rank), Подати апеляцію (/appeal), Переглянути лідерборд.
- Система: Автоматично нарахувати XP, Виявити спам, Застосувати фільтр слів, Автоматично забанити після 3 попереджень.



Рис. 1.2 UML-діаграма

Алгоритми ключових процесів

При проектуванні системи особлива увага приділялася розробці ефективних алгоритмів основних функціональних модулів, які забезпечують автоматизовану модерацію та комфортне користування спільнотою.

Алгоритм анти-спаму був спроектований як багатоступенева система виявлення небажаної активності. Він включає три основні компоненти: виявлення флуду (надмірна кількість повідомлень за короткий проміжок часу), виявлення дублікатів (повторювані або схожі повідомлення) та виявлення CAPS SPAM (надмірне використання великих літер). Для цього було вирішено використовувати структуру даних deque (черга з обмеженим розміром) для зберігання історії останніх повідомлень кожного користувача в пам'яті. Алгоритм порівнює нові повідомлення з попередніми за допомогою функції обчислення схожості рядків. При досягненні певної кількості порушень система застосовує ескалацію покарання: від попередження до таймауту та, в критичних випадках, до бану через інтеграцію з системою попереджень.

Алгоритм перевірки заборонених слів спроектований з урахуванням специфіки двомовних спільнот. Він складається з двох незалежних словників (англійський та український) та механізмів нормалізації тексту. Основними етапами алгоритму є: приведення тексту до нижнього регістру, застосування таблиць заміни символів для обходу обхідних способів написання мату (наприклад, @ - а, 3 - з, 1 - і тощо), видалення роздільників між літерами та пошук як точних збігів, так і кореневих слів за допомогою регулярних виразів. Таке проектування дозволяє ефективно виявляти не тільки прямі матюки, але й їх завуальовані варіанти.

Алгоритм нарахування досвіду (XP) був спроектований з метою підвищення залученості учасників без можливості накручування. Основними елементами є кулдаун (60 секунд між нарахуваннями), випадкове нарахування

XP у діапазоні від 5 до 10 одиниць за повідомлення, розрахунок рівня за формулою $\text{level} = \text{xp} // 100$ та автоматична видача ролей (Active Member на 5 рівні та Pro Member на 10 рівні). Алгоритм також веде підрахунок загальної кількості повідомлень користувача.

Процес апеляцій спроектований як крос-платформенний механізм. Користувач подає апеляцію через команду `/appeal` у Discord, після чого запис зберігається в базі даних зі статусом `pending`. Модератор може переглянути та обробити апеляцію через веб-панель (`/appeals`), де доступні кнопки «Прийняти» та «Відхилити». Прийняття апеляції автоматично знімає бан та зменшує кількість попереджень, а всі дії логуються в ModLog. Такий підхід забезпечує прозорість і зручність модерації. Усі алгоритми були спроектовані з урахуванням продуктивності та можливості роботи в реальному часі.

Вимоги до безпеки, двомовності та multi-server підтримки

Безпека є одним з пріоритетних аспектів. Токен Discord-бота зберігається виключно в змінних середовища (`env.env` файл), що виключає його потрапляння в репозиторій коду[25]. Для веб-панелі використовується секретний ключ сесії (`FLASK_SECRET_KEY`), який також виноситься в змінні середовища. Доступ до веб-панелі захищено простою сесійною автентифікацією. Перевірка прав модераторів здійснюється через функцію `is_moderator()`, яка враховує наявність ролі Administrator, права `manage_messages` або спеціальної ролі Moderator. Всі критичні дії (видання попереджень, банів, обробка апеляцій) обов'язково логуються з вказанням модератора, що забезпечує аудит дій.

Двомовність була закладена на рівні архітектури. Інтерфейс бота підтримує українську та англійську мови. Embed-повідомлення формуються двома мовами одночасно. Фільтр заборонених слів також є двомовним. Веб-

панель повністю локалізована українською мовою з можливістю подальшого розширення.

Multi-server підтримка була одним з ключових архітектурних рішень. Усі таблиці бази даних містять поле `server_id`, а більшість запитів до БД виконуються з урахуванням цього ідентифікатора. Це дозволяє одному запущеному екземпляру бота обслуговувати різні спільноти незалежно, без ризику змішування даних. Крім того, при проектуванні враховувалися вимоги до масштабованість, зручності розгортання та подальшого супроводу системи.

Таким чином, у підрозділі 1.2 була спроектована комплексна архітектура інтерактивного Discord-бота з веб-адмінпанеллю, яка поєднує в собі сучасні підходи до побудови програмного забезпечення, ефективні алгоритми обробки даних та високі вимоги до безпеки, зручності та функціональності[13].

РОЗДІЛ 2. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО DISCORD-БОТА

2.1 Реалізація основної логіки бота та модулів модерації

Реалізація програмного забезпечення є ключовим етапом кваліфікаційної роботи, на якому проектні рішення втілюються в робочий код[6]. У даному підрозділі розглядається реалізація основної логіки Discord-бота, обробка подій, система модераторів та інші базові механізми.

Точка входу та паралельний запуск

Точкою входу в систему є файл `main.py`. Його основне завдання – забезпечити одночасну роботу двох незалежних компонентів: Discord-бота та веб-адмінпанелі на базі Flask.

```
if __name__ == "__main__":
    discord_bot = setup_bot()
    bot_thread = threading.Thread(target=run_discord_bot,
args=(discord_bot,))
    bot_thread.daemon = True
    bot_thread.start()

app.run(host='0.0.0.0', port=5000, debug=True,
use_reloader=False)
```

У наведеному коді створюється екземпляр бота за допомогою функції `setup_bot()`, після чого він запускається в окремому демонічному потоці. Це рішення дозволяє боту постійно підтримувати з'єднання з Discord API і обробляти події в реальному часі. Одночасно в основному потоці запускається Flask-сервер, який обслуговує веб-адмінпанель.

Використання модуля `threading` є свідомим архітектурним рішенням, оскільки бібліотека `discord.py` працює в асинхронному режимі, а Flask – у

синхронному. Паралельний запуск дозволяє обом компонентам працювати незалежно, не блокуючи один одного. Параметр `use_reloader=False` вимкнено для запобігання дублюванню процесів при запуску в режимі `debug`.

Таке архітектурне рішення є компромісом між простотою реалізації та надійністю роботи. Альтернативним варіантом міг би бути повноцінний асинхронний веб-сервер (наприклад Quart або FastAPI з ASGI), але для даного проєкту використання `threading` виявилось достатньо ефективним, оскільки навантаження на веб-панель є відносно невеликим. Це дозволило прискорити розробку та уникнути складної роботи з асинхронними контекстами бази даних.

Крім того, використання `threading` дозволило зберегти простоту коду та уникнути складного управління асинхронними сесіями бази даних. У разі переходу на продакшн-версію проєкту таку архітектуру можна легко масштабувати, наприклад, перейшовши на окремий процес для веб-панелі через Gunicorn + Nginx або використовуючи Celery для фонових задач. На даному етапі обране рішення забезпечує оптимальний баланс між швидкістю розробки, стабільністю та ресурсоспоживанням.

Обробка подій: `on_ready` та `on_member_join`

Основна логіка бота реалізована в класі `ModerationBot`, який успадковується від `discord.Client`. Першою важливою подією є `on_ready()`.

```
async def on_ready(self):
    logger.info(f"Bot is logged in as {self.user}")
    await self.change_presence(activity=discord.Activity(
        type=discord.ActivityType.watching,
        name="за неприпустимою мовою"
    ))
```

У цій функції виконується логування успішного підключення бота та встановлення статусу присутності. Це дозволяє адміністраторам швидко

зрозуміти, що бот працює. Значно складнішою є реалізація події `on_member_join()`, яка відповідає за автоматичне привітання нових учасників сервера. Алгоритм працює наступним чином:

1. Перевірка прав бота в гільдії.
2. Створення двох `embed`-повідомлень – українською та англійською мовами.
3. Автоматичний пошук підходящого каналу для привітання (за пріоритетом: `welcome` – `general` – `lobby` – `chat` – `hall`).
4. Якщо підходящий канал не знайдено – повідомлення надсилається в системний канал або в особисті повідомлення користувачу.
5. Логування події привітання в базу даних через модель `ModLog`.

Такий підхід забезпечує дружній прийом нових учасників і підтримує двомовність спільноти. Використання `embed`-повідомлень робить привітання візуально привабливим і інформативним.

Автоматичне привітання нових учасників з підтримкою двох мов є важливою складовою комфортного користування спільнотою. На відміну від багатьох готових ботів, які надсилають стандартне повідомлення лише англійською, розроблене рішення адаптується під мовні уподобання спільноти, що особливо важливо для українських та двомовних серверів. Крім того, інтелектуальний пошук каналу для привітання підвищує гнучкість бота при розгортанні на серверах з різною структурою каналів.

Варто зазначити, що реалізація привітання з урахуванням структури каналів сервера демонструє гнучкість бота. Багато готових рішень жорстко прив'язані до конкретного каналу (наприклад, `#welcome`), що вимагає ручного налаштування на кожному сервері. У розробленому боті цей процес відбувається автоматично, що значно спрощує розгортання і обслуговування.

Система модераторів

Для розмежування прав доступу була реалізована функція `is_moderator()` та дві команди: `/addmod` і `/removemod`. Функція `is_moderator()` перевіряє права користувача за такими критеріями:

- Наявність прав адміністратора (`administrator`);
- Наявність прав керування повідомленнями (`manage_messages`);
- Наявність спеціальної ролі `Moderator`.

```
def is_moderator(member: discord.Member) -> bool:
    if member.guild_permissions.administrator:
        return True
    if member.guild_permissions.manage_messages:
        return True
    role = discord.utils.get(member.guild.roles,
name=MODERATOR_ROLE_NAME)
    return role in member.roles if role else False
```

Команда `/addmod` доступна тільки адміністраторам сервера. Вона видає роль `Moderator` згаданому користувачу. Команда `/removemod` виконує зворотню дію – забирає цю роль. Обидві команди супроводжуються інформативними `embed`-повідомленнями та перевіркою на існування ролі. Реалізація системи модераторів дозволяє гнучко керувати правами без необхідності надавати користувачам повні права адміністратора, що значно підвищує безпеку сервера.

Реалізована система розмежування прав є гнучкою і безпечною. Вона дозволяє адміністраторам видавати модераторські права без необхідності надавати повні права адміністратора сервера. Такий підхід суттєво знижує ризики, пов'язані з можливим зловживанням правами, і відповідає сучасним практикам безпеки `Discord`-спільнот. Крім того, перевірка прав виконується при кожній модераторській дії, що забезпечує постійний контроль.

Додатковою перевагою є можливість швидкого додавання нових рівнів прав у майбутньому. Наприклад, можна ввести роль «Старший модератор», яка

матиме розширені права порівняно зі звичайним модератором. Така масштабованість системи прав доступу робить бота придатним для спільнот різного розміру – від невеликих студентських серверів до великих ігрових спільнот з багат шаровою модераторською командою.

Команди попереджень та банів

Одним з ключових механізмів модерації є система попереджень (/warn, /unwarn, /warnings) з автоматичним баном після досягнення трьох попереджень. Основна команда /warn реалізується в методі on_message класу ModerationBot. Алгоритм роботи команди такий:

```
if message.content.strip().lower().startswith('/warn '):
    if not is_moderator(message.author):
        await message.channel.send("У вас немає прав для цієї команди.")
        return
    target = message.mentions[0]
    reason = parts[2] if len(parts) > 2 else "Порушення правил сервера"
    with app.app_context():
        user_warning = UserWarning.query.filter_by(
            user_id=str(target.id),
            server_id=str(message.guild.id)
        ).first()
        if user_warning:
            user_warning.warning_count += 1
        else:
            user_warning = UserWarning(...)
            db.session.add(user_warning)
        if user_warning.warning_count >= 3:
            user_warning.is_banned = True
            await message.guild.ban(target, reason=...)
        db.session.commit()
```

При кожному попередженні створюється запис у таблиці ModLog, що забезпечує повну історію дій модераторів. Команда `/unwarn` зменшує лічильник попереджень на одиницю і знімає статус бану, якщо кількість попереджень стає меншою за три. Команда `/warnings` дозволяє модераторам швидко перевірити поточний стан користувача, виводячи кількість попереджень і статус (активний або заблокований) у вигляді зручного embed-повідомлення. Автоматичний бан після третього попередження інтегрований безпосередньо в логіку команди `/warn` і в обробник заборонених слів. Це гарантує послідовність покарань і зменшує суб'єктивність модерації.

Інтеграція системи попереджень з іншими модулями (анти-спам та фільтр слів) створює єдиний контур модерації. Це означає, що порушення, виявлені автоматично, автоматично враховуються в загальному лічильнику попереджень. Такий підхід забезпечує послідовність і справедливість покарань, зменшуючи ймовірність конфліктів між автоматичними та ручними діями модераторів.

Анти-спам модуль (`anti_spam.py`)

Модуль анти-спаму є одним з найскладніших і найважливіших компонентів бота[15]. Він реалізує багатошарову систему виявлення небажаної активності в реальному часі. Основна функція `handle_anti_spam()` викликається першою в обробнику `on_message` і повертає `True`, якщо повідомлення розпізнано як спам. Модуль перевіряє три типи порушень:

1. Флуд – більше 5 повідомлень за 5 секунд.
2. Дублікати – надсилання однакових або дуже схожих повідомлень (поріг схожості 85%).
3. CAPS SPAM – повідомлення, де більше 70% літер – великі (при довжині від 8 символів).

Для зберігання історії повідомлень використовується `defaultdict` з `deque` (максимальний розмір 20). Функція `_similarity()` обчислює схожість рядків за кількістю співпадаючих символів. При виявленні порушення повідомлення видаляється, а користувачу нараховується порушення. Система має ескалацію покарань:

- 1-2 порушення – попередження;
- 3 порушення – таймаут на 10 хвилин;
- 6 і більше порушень – бан через інтеграцію з системою UserWarning.

Усі дії анти-спаму ретельно логуються в таблицю ModLog через функцію `_log_spam_action()`.

Важливою особливістю модуля є його висока продуктивність навіть при великій кількості повідомлень. Використання структури `deque` з обмеженим розміром дозволяє ефективно працювати з історією повідомлень кожного користувача без значного споживання оперативної пам'яті. Така реалізація дозволяє боту стабільно функціонувати на серверах з тисячами активних учасників.

Фільтр заборонених слів (`word_filters.py`)

Модуль `word_filters.py` відповідає за виявлення ненормативної лексики українською та англійською мовами. Він містить два великих списки заборонених слів (`english_banned_words` та `ukrainian_banned_words`). Алгоритм перевірки `check_message()` складається з кількох етапів:

- Пошук точних збігів у оригінальному тексті;
- Нормалізація тексту для англійської (`normalize_for_english`) та української (`normalize_for_ukrainian`) мов;
- Застосування таблиць заміни символів (`EN_SUBSTITUTIONS`, `UA_SUBSTITUTIONS`) для обходу маскуванню мату;
- Видалення роздільників між літерами (`_remove_separators`);

- Пошук кореневих слів за допомогою регулярних виразів.

Функція повертає кортеж (`contains_banned`, `language`), де `language` вказує, якою мовою було виявлено порушення. Це дозволяє формувати відповідні повідомлення користувачу українською або англійською. При виявленні забороненого слова повідомлення видаляється, користувачу видається попередження, а дія логується в `ModLog`. Якщо це третє попередження – відбувається автоматичний бан. Реалізація фільтра враховує особливості української мови (трансліт, варіації написання) і є значно ефективнішою, ніж у більшості готових Discord-ботів.

Розроблений фільтр значно перевершує стандартні рішення більшості готових ботів за якістю обробки української мови. Завдяки комплексній нормалізації та використанню таблиць заміни символів він ефективно виявляє не тільки прямі матюки, але й їх численні завуальовані варіанти, що є особливо актуальним для україномовних спільнот.

Система XP та рівнів

Система досвіду та рівнів (`xp_system.py`) реалізує елемент гейміфікації, метою якої є підвищення активності учасників спільноти[42]. Модуль підключається в основному обробнику `on_message` після всіх перевірок модерації. Основна функція `handle_xp()` виконує наступні дії:

- Ігнорує повідомлення від ботів та приватні повідомлення;
- Перевіряє кулдаун (60 секунд між нарахуваннями XP);
- Нараховує випадкову кількість досвіду від 5 до 10 одиниць;
- Оновлює рівень користувача за формулою `level = xp // 100`;
- При підвищенні рівня викликає функцію `_announce_level_up()` для відправки вітального повідомлення.

```
xp_earned = random.randint(XP_PER_MESSAGE_MIN,
XP_PER_MESSAGE_MAX)
old_level = user_level.level
```

```

user_level.xp += xp_earned
new_level = calculate_level(user_level.xp)
if new_level > old_level:
    await _announce_level_up(message, user_level, new_level)
    await _assign_level_roles(message.author, new_level)

```

Особливістю системи є автоматична видача ролей Active Member (на 5 рівні) та Pro Member (на 10 рівні) через функцію `grant_role()`. Команди `/rank` та `/leaderboard` (або `/top`, `/lb`) дозволяють користувачам переглядати свій прогрес та загальну таблицю лідерів. Для візуалізації прогресу використовується текстовий прогрес-бар, згенерований функцією `progress_bar()`. Реалізація системи XP добре інтегрується з основною логікою бота і не заважає роботі модулів модерації.

Система XP добре продумана з точки зору балансу. Кулдаун у 60 секунд запобігає накручуванню досвіду шляхом спаму, а випадкове нарахування від 5 до 10 одиниць додає елемент непередбачуваності, що позитивно впливає на мотивацію учасників. Автоматична видача ролей на певних рівнях (Active Member та Pro Member) дозволяє адміністраторам візуально відзначати найбільш активних користувачів, що сприяє формуванню здорової спільноти.

Система апеляцій

Механізм апеляцій реалізовано як зручний спосіб оскарження бану. Користувач може подати апеляцію командою `/appeal <причина>`. При цьому створюється запис у таблиці Appeal зі статусом `pending`, а модератори отримують сповіщення у спеціальний канал (за замовчуванням – `moderators`). Обробка апеляцій здійснюється двома командами:

- `/appeal_accept <id> [причина]` – приймає апеляцію, знімає бан і обнуляє попередження;
- `/appeal_reject <id> [причина]` – відхиляє апеляцію.

Крім того, апеляції можна обробляти через веб-панель. Усі дії (подача, прийняття, відхилення) обов'язково фіксуються в журналі ModLog. Така реалізація забезпечує прозорість модераторських рішень і підвищує довіру учасників спільноти.

Інші команди

Бот підтримує низку додаткових корисних команд:

- `/clear [кількість]` – масове видалення повідомлень (максимум 100). Команда доступна лише модераторам.
- `/mute @user [хвилини]` та `/unmute @user` – видача та зняття таймауту (Discord timeout).
- `/unban <user_id або @user>` – розбан користувача з оновленням запису в базі даних.
- `/help` – виводить правила сервера у вигляді двох embed-повідомлень українською та англійською мовами.

Кожна команда супроводжується перевіркою прав доступу, обробкою помилок та логуванням дій.

Логування всіх дій у базі даних

Усі значущі події фіксуються в таблиці ModLog. Для цього використовується модель ModLog, яка зберігає інформацію про користувача, модератора, тип дії, причину, канал та час події. Логування відбувається в єдиному стилі через контекст `app.app_context()` і сесію SQLAlchemy. Це дозволяє модераторам у веб-панелі переглядати повну історію дій, що суттєво підвищує прозорість і підзвітність модераторського складу.

Важливим аспектом реалізації є забезпечення стабільної взаємодії між усіма модулями. Наприклад, після успішного проходження анти-спам перевірки та фільтра заборонених слів повідомлення потрапляє в систему XP, що дозволяє уникнути нарахування досвіду за спам-повідомлення. Така

послідовність обробки подій (`on_message`) значно підвищує ефективність і надійність роботи бота в реальних умовах навантаження.

Загалом, усі модулі бота побудовані за принципом високої когезії та низької зв'язності. Кожен модуль відповідає за чітко визначену функціональність, що полегшує тестування, відлагодження та подальше розширення системи. Така модульна архітектура є одним з ключових факторів надійності та масштабованість розробленого рішення.

Підсумовуючи реалізацію основної логіки бота, можна стверджувати, що всі ключові модулі працюють у тісній інтеграції та доповнюють один одного. Анти-спам і фільтр слів забезпечують чистоту спілкування, система попереджень і апеляцій – справедливість модерації, а система ХР – мотивацію учасників. Така комплексність є значною перевагою порівняно з більшістю готових рішень, які зазвичай добре розвивають лише один-два напрями. Модульна структура коду, централізоване логування та продумана послідовність обробки подій роблять систему надійною, масштабованною та зручною в підтримці.

Отже, у підрозділі 2.1 детально розглянуто реалізацію основної логіки Discord-бота та ключових модулів модерації[41]. Було створено стабільну точку входу системи з паралельним запуском бота та веб-панелі, ефективну обробку ключових подій (`on_ready`, `on_member_join`), гнучку систему розмежування прав модераторів, потужні механізми попереджень і автоматичних банів, а також високоефективні модулі анти-спаму, двомовного фільтра заборонених слів, системи гейміфікації (ХР) та прозорого механізму апеляцій. Усі модулі тісно інтегровані між собою, працюють узгоджено та забезпечують комплексний підхід до автоматизованої модерації. Реалізовані рішення дозволяють не тільки ефективно боротися зі спамом і токсичністю, але й суттєво підвищувати залученість учасників спільноти завдяки елементам

гейміфікації. Комплексне логування всіх дій у базі даних створює необхідну прозорість і підзвітність модераторського складу. Таким чином, основна логіка бота та модулі модерації повністю відповідають поставленим завданням і створюють надійний фундамент для подальшої роботи веб-адмінпанелі.

2.2 Розробка веб-адмін панелі та інтеграція з ботом

Важливою частиною кваліфікаційної роботи стала розробка веб-адмінпанелі, яка значно підвищує зручність управління Discord-сервером. На відміну від традиційного керування лише через текстові команди в Discord, веб-панель надає модераторам сучасний графічний інтерфейс для перегляду журналу дій, роботи з попередженнями та обробки апеляцій користувачів.

Веб-адмінпанель побудована на мікрофреймворку Flask. У файлі `app.py` виконується ініціалізація самого додатка, підключення бази даних через Flask-SQLAlchemy, налаштування секретного ключа для роботи з сесіями та реєстрація всіх необхідних маршрутів. Для захисту доступу використовується механізм сесій на основі секретного ключа, який зберігається в змінних середовища. Це дозволяє в майбутньому легко розширити систему автентифікації до повноцінного логіну з паролями.

Крім базової ініціалізації, у файлі `app.py` реалізовано реєстрацію всіх маршрутів, підключення розширень Flask-SQLAlchemy та налаштування секретного ключа сесій. Важливим рішенням стало використання єдиного контексту додатка (`app.app_context()`), що дозволяє Discord-боту, який працює в окремому потоці, безпечно взаємодіяти з базою даних. Такий підхід уникає проблем з багатопотоковістю та забезпечує цілісність даних між двома основними компонентами системи.

Архітектура веб-панелі побудована з урахуванням принципів RESTful API. Кожен маршрут чітко розділений за HTTP-методами: GET-запити

відповідають за отримання даних і відображення інформації, тоді як POST-запити використовуються для виконання дій, що змінюють стан системи (зміна статусу апеляції, оновлення попереджень тощо). Такий підхід забезпечує чітке розділення відповідальностей і полегшує подальше тестування та розширення функціоналу. Крім того, у `app.py` активно використовується контекстний менеджер `app.app_context()`, що дозволяє асинхронному Discord-боту безпечно взаємодіяти з базою даних `SQLAlchemy`.

Шаблони сторінок генеруються за допомогою шаблонізатора `Jinja2`[8]. Такий підхід дає можливість використовувати єдину навігаційну панель на всіх сторінках, застосовувати спеціальні фільтри та динамічно відображати дані з бази. Усі маршрути розділені на GET-запити для перегляду інформації та POST-запити для виконання дій, таких як прийняття або відхилення апеляцій. Він використовується не лише для простого виведення даних, а й для побудови добре організованої системи шаблонів. У проєкті застосовується принцип успадкування шаблонів: створено базовий шаблон з єдиною навігаційною панеллю, заголовком, стилями та футером, від якого успадковуються всі інші сторінки. Такий підхід дозволяє уникнути дублювання коду HTML і CSS, значно спрощує підтримку дизайну та забезпечує єдиний вигляд усіх розділів панелі. Крім того, в шаблонах активно застосовуються макроси `Jinja2` для повторюваних елементів, таких як картки апеляцій, рядки таблиць попереджень та повідомлення про порожній стан сторінки. Це робить код шаблонів більш чистим, читабельним і зручним для подальшого розширення.

Використання успадкування шаблонів (`{% extends "base.html" %}`) та макросів (`{% macro %}`) дозволило значно зменшити дублювання коду. Базовий шаблон містить загальну навігаційну панель, футер, підключення `Bootstrap` та фільтр `kyiv_time`. Це рішення не тільки прискорило розробку, але й забезпечило

єдиний стиль усього інтерфейсу, що важливо для користувацького досвіду модераторів.

Головна сторінка панелі відображає базову інформацію про те, що бот успішно працює, та містить швидкі посилання на основні розділи: журнал модерації, список попереджень та сторінку апеляцій (Рис 2.1). Сторінка журналу модерації виводить повну таблицю всіх зареєстрованих дій з детальною інформацією про користувача, тип дії, причину, канал та точний час події в київському часовому поясі.

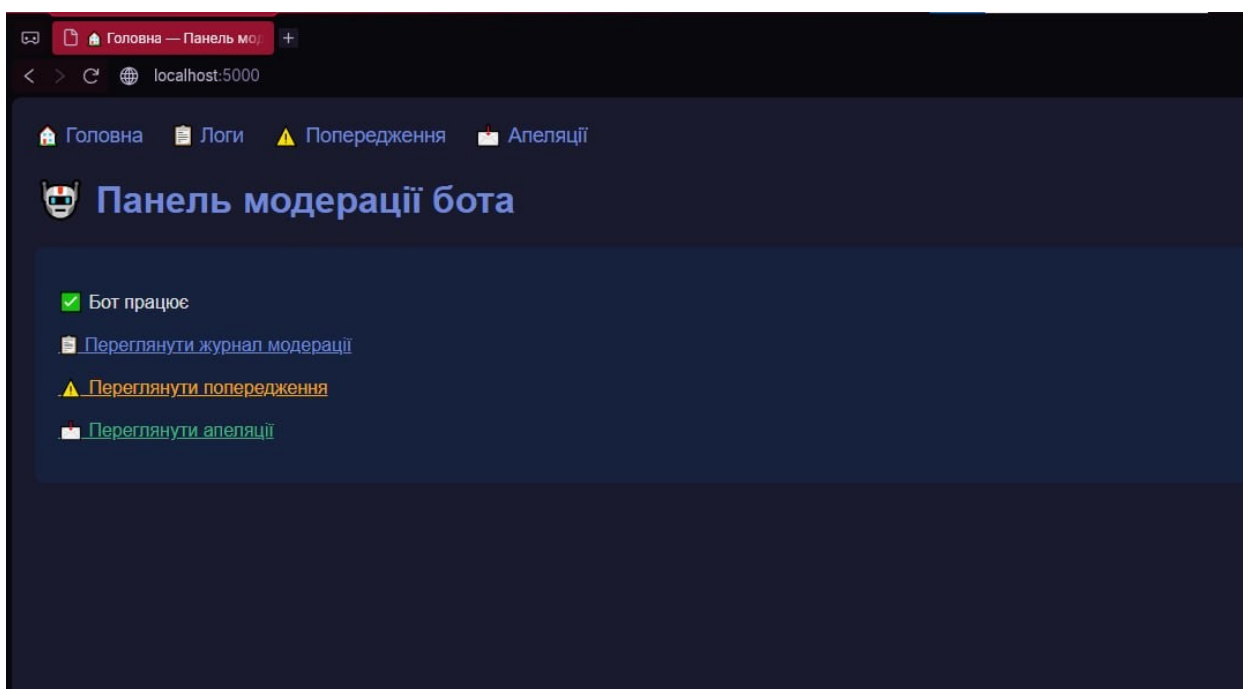


Рис. 2.1 Головна сторінка веб-адмін панелі

Сторінка попереджень показує всіх користувачів, які отримали хоча б одне попередження (Рис. 2.2). Дані сортуються за статусом бану та кількістю попереджень, що дозволяє модераторам швидко оцінювати ситуацію. Кожен запис супроводжується кольоровою індикацією: червоні позначки для заблокованих користувачів і зелені – для активних.

Користувач	Попереджень	Статус	Остання дія	Заблоковано
bruhhh3768	2/3	Активний	14.04.2026 19:35:23	
bruhhh3768	2/3	Активний	25.04.2026 05:58:38	25.04.2026 05:58:38
bimba4645	0/3	Активний	25.04.2026 05:59:27	

Рис. 2.2 Сторінка попереджень

Найбільш функціональною є сторінка апеляцій (Рис. 2.3). Вона відображає список усіх поданих апеляцій із зазначенням їх статусу – на розгляді, прийнято або відхилено. Для апеляцій зі статусом «на розгляді» безпосередньо на сторінці розміщені дві кнопки: «Прийняти» та «Відхилити». При натисканні на них виконується відповідний POST-запит, який змінює статус апеляції, оновлює кількість попереджень користувача та створює запис у журналі модерації.

Сторінка апеляцій є найбільш інтерактивною частиною панелі. Вона не лише відображає список заявок, але й дозволяє безпосередньо в інтерфейсі приймати або відхиляти апеляції з додаванням коментаря. Така реалізація суттєво прискорює роботу модераторів порівняно з обробкою апеляцій через Discord-команди, де потрібно вручну копіювати ідентифікатори та вводити параметри.

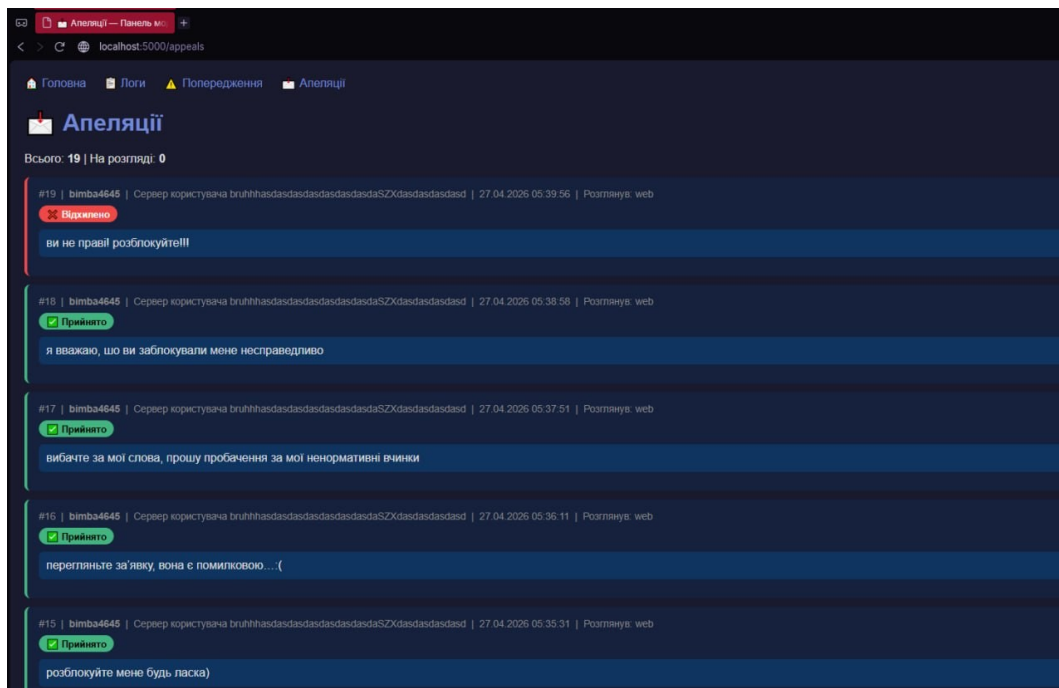


Рис 2.3 Сторінка апеляцій

Робота з базою даних у веб-панелі здійснюється через ORM-систему SQLAlchemy. Усі моделі даних описані в окремому модулі `models.py`. Це дозволяє працювати з даними на високому рівні абстракції, забезпечуючи єдину структуру між Discord-ботом і веб-інтерфейсом. При запуску додатка автоматично створюються або оновлюються всі необхідні таблиці. У маршрутах обробки апеляцій відбувається комплексне оновлення даних: змінюється статус апеляції, коригується запис у таблиці попереджень і додається новий запис у журнал дій.

Для зручності відображення часу був реалізований власний Jinja2-фільтр `kyiv_time`, який автоматично перетворює UTC-час у київський часовий пояс у форматі `день.місяць.рік година:хвилина:секунда` [9]. Оформлення панелі виконане за допомогою Bootstrap через CDN, що забезпечує сучасний вигляд, адаптивність під різні пристрої та швидке завантаження сторінок[26]. Фільтр приймає об'єкт дати у форматі UTC і перетворює його в київський часовий пояс з урахуванням літнього та зимового часу. Реалізація фільтра базується на модулі `zoneinfo`, що є сучасним і рекомендованим способом роботи з часовими

поясами в Python. Фільтр зареєстрований у додатку як `@app.template_filter('kyiv_time')` і може використовуватися в будь-якому шаблоні простим записом `{{ date | kyiv_time }}`. Це рішення має велике практичне значення для української аудиторії, оскільки усуває плутанину з різницею в часі, робить усі дати зрозумілими і природними для користувачів з України. Без такого фільтра модераторам доводилося б постійно переводити час вручну, що знижувало б зручність роботи.

Крім зручності для користувачів з України, фільтр `kyiv_time` також полегшує аналіз журналу модерації. Модератори бачать усі події в єдиному часовому поясі, що зменшує ймовірність помилок при визначенні послідовності дій, особливо коли робота ведеться в команді з учасників з різних регіонів.

Окремої уваги заслуговує питання безпеки веб-адмінпанелі. Доступ до панелі захищено через сесійний механізм Flask з використанням секретного ключа, що зберігається в змінних середовища. Усі критичні дії (обробка апеляцій, зміна статусу попереджень) виконуються тільки після перевірки активної сесії. Для захисту від поширених веб-атак у проєкті передбачено автоматичне екранування даних у Jinja2, а також потенційна можливість підключення CSRF-токенів. Такі заходи мінімізують ризики несанкціонованого доступу та забезпечують відповідність базовим вимогам безпеки веб-додатків.

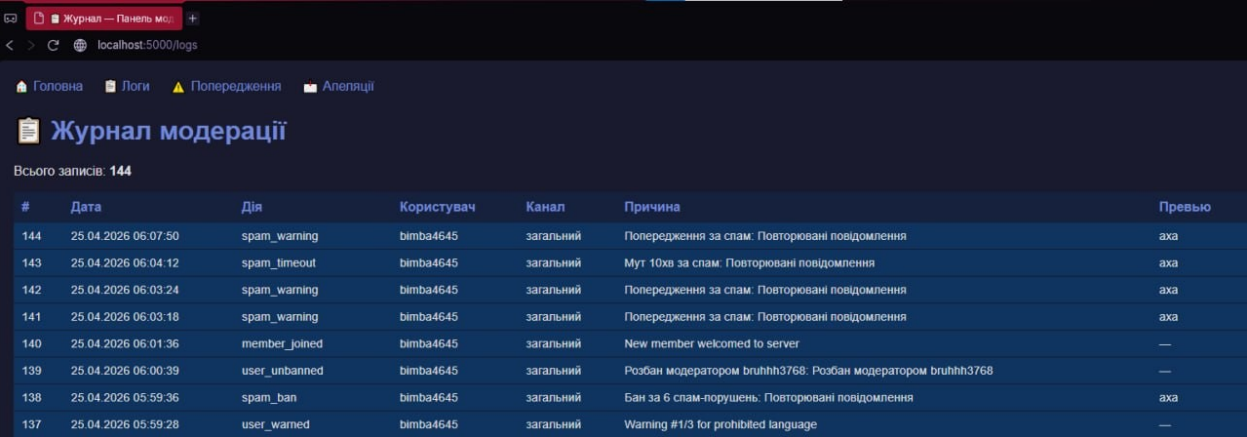
Важливим аспектом є інтеграція веб-панелі з Discord-ботом. При поданні апеляції через команду в Discord модератори отримують сповіщення у спеціальному каналі. Після обробки апеляції через веб-інтерфейс зміни негайно відображаються як у базі даних, так і впливають на стан користувача в Discord (зняття бану при схваленні). Така двостороння інтеграція робить систему єдиним цілісним рішенням.

Двостороння інтеграція є одним з ключових досягнень проєкту. Зміни, внесені через веб-панель (наприклад, прийняття апеляції), негайно

відображаються в Discord (зняття бану, оновлення ролі тощо). І навпаки – апеляції, подані через команду в Discord, автоматично з’являються у веб-інтерфейсі. Така синхронізація забезпечує актуальність даних у реальному часі та створює єдине інформаційне середовище для модераторів.

Порівняно з традиційним управлінням сервером виключно через Discord-команди, веб-адмінпанель демонструє значні переваги. Якщо для перегляду історії модерації через чат модератору потрібно вводити кілька команд і шукати повідомлення, то у веб-панелі вся інформація доступна в зручній таблиці з фільтрами та сортуванням. Обробка апеляцій, яка раніше вимагала копіювання ID і написання команди з параметрами, тепер відбувається за два кліки. Такий перехід від CLI до GUI суттєво знижує час реакції та ймовірність людських помилок.

Сторінка журналу модерації є однією з найважливіших у веб-панелі, оскільки вона забезпечує повну прозорість усіх дій, які виконуються на сервері (Рис 2.4). На цій сторінці модератори можуть переглядати детальну історію всіх модераційних заходів, включаючи видачу попереджень, бани, видалення повідомлень, обробку апеляцій та інші дії. Дані виводяться у вигляді таблиці, відсортовані за датою створення у зворотному порядку, що дозволяє швидко знайти найсвіжіші записи. Кожний запис містить інформацію про користувача, тип виконаної дії, причину, канал, у якому відбулася подія, а також точний час у київському часовому поясі. Така реалізація значно полегшує контроль за роботою модераторського складу і допомагає у разі виникнення конфліктних ситуацій.



#	Дата	Дія	Користувач	Канал	Причина	Прев'ю
144	25.04.2026 06:07:50	spam_warning	bimba4645	загальний	Попередження за спам. Повторювані повідомлення	аха
143	25.04.2026 06:04:12	spam_timeout	bimba4645	загальний	Мут 10х за спам. Повторювані повідомлення	аха
142	25.04.2026 06:03:24	spam_warning	bimba4645	загальний	Попередження за спам. Повторювані повідомлення	аха
141	25.04.2026 06:03:18	spam_warning	bimba4645	загальний	Попередження за спам. Повторювані повідомлення	аха
140	25.04.2026 06:01:36	member_joined	bimba4645	загальний	New member welcomed to server	—
139	25.04.2026 06:00:39	user_unbanned	bimba4645	загальний	Розбан модератором bruhhh3768: Розбан модератором bruhhh3768	—
138	25.04.2026 05:59:36	spam_ban	bimba4645	загальний	Бан за 6 спам-порушень. Повторювані повідомлення	аха
137	25.04.2026 05:59:28	user_warned	bimba4645	загальний	Warning #1/3 for prohibited language	—

Рис. 2.4 Журнал модераторів

Оформлення всіх сторінок виконане за допомогою фреймворку Bootstrap, підключеного через CDN. Таке рішення забезпечує сучасний, приємний зовнішній вигляд, єдину стилістику елементів та повну адаптивність (responsive design). Панель коректно відображається як на великих моніторах, так і на планшетах та смартфонах, що важливо для модераторів, які можуть працювати з різних пристроїв. Темна кольорова схема інтерфейсу (темно-сині та фіолетові відтінки) відповідає традиційній стилістиці Discord і не втомлює очі при тривалій роботі.

Коли користувач подає апеляцію через команду /appeal у Discord, бот не тільки зберігає її в базі даних, але й надсилає сповіщення в спеціальний канал для модераторів. Це дозволяє оперативно реагувати на нові апеляції. У зворотному напрямку, при обробці апеляції через веб-панель, система автоматично змінює статус користувача в Discord (знімає бан при схваленні) і фіксує всі зміни в журналі дій. Така двостороння синхронізація робить веб-панель і бота єдиним цілісним рішенням.

Код ключових фрагментів веб-панелі показує чіткість і продуманість реалізації. Наприклад, обробка прийняття апеляції виглядає наступним чином:

```
@app.route('/appeals/<int:appeal_id>/approve',
methods=['POST'])
def appeal_approve(appeal_id):
```

```

        reason = request.form.get('reason', '').strip() or
'Прийнято через веб-панель'
        with app.app_context():
            appeal = db.session.get(models.Appeal, appeal_id)
            if appeal and appeal.status == 'pending':
                appeal.status = 'approved'
                appeal.resolved_at = datetime.utcnow()
                appeal.resolved_by = 'web'
                appeal.resolution_reason = reason
                user_warning =
db.session.query(models.UserWarning).filter_by(
                    user_id=appeal.user_id,
server_id=appeal.server_id
                ).first()
                if user_warning:
                    user_warning.warning_count = max(0,
user_warning.warning_count - 1)
                    if user_warning.warning_count < 3:
                        user_warning.is_banned = False
                        user_warning.banned_at = None
                    log = models.ModLog(...)
                    db.session.add(log)
                    db.session.commit()
            return redirect(url_for('appeals'))

```

Подібним чином реалізована обробка відхилення апеляції. Така структура коду забезпечує атомарність операцій і надійність збереження даних. Загалом розроблена веб-адмінпанель є сучасним, зручним і функціональним інструментом, який суттєво доповнює можливості Discord-бота. Вона поєднує в собі якісне оформлення, адаптивність, правильну роботу з часом, надійну інтеграцію з основним ботом та зручний інтерфейс для щоденної роботи модераторів.

Розроблена веб-адмінпанель має значні практичні переваги порівняно з керуванням сервером виключно через текстові команди в Discord[10]. По-перше, вона надає візуальне представлення даних: модератор може одним поглядом оцінити кількість активних попереджень, побачити хронологію дій або швидко знайти конкретну апеляцію, тоді як у Discord для цього потрібно вводити кілька команд і шукати інформацію в чаті[44]. По-друге, панель значно прискорює обробку апеляцій – замість написання команд `/appeal_accept` або `/appeal_reject` з параметрами модератор просто натискає кнопку в зручному інтерфейсі. По-третє, веб-панель дозволяє працювати одночасно кільком модераторам без ризику конфліктів, оскільки всі дії фіксуються в базі даних у реальному часі. Крім того, адаптивний дизайн дає можливість модераторам ефективно працювати не тільки з комп'ютера, але й з планшета або смартфона, що особливо актуально в сучасних умовах.

Практична цінність веб-панелі для модераторів проявляється насамперед у суттєвому зменшенні часу, який витрачається на рутинні операції[11]. Замість того, щоб постійно перемикатися між каналами Discord і вводити команди, модератор отримує єдиний централізований інструмент, де вся необхідна інформація зібрана в одному місці[43]. Це знижує ймовірність помилок, підвищує швидкість реакції на порушення та апеляції, а також покращує загальну якість модерації. Для великих і середніх спільнот така панель стає практично незамінною, оскільки ручне керування через чат у таких умовах швидко перетворюється на хаос.

Крім того, веб-панель сприяє підвищенню прозорості модераційних дій. Кожен модератор може в будь-який момент переглянути повну історію своїх дій і дій колег, що допомагає уникати конфліктів і непорозумінь всередині команди. Для адміністраторів серверу це також важливий інструмент контролю та аналізу ефективності модераторського складу.

Загалом, розроблена веб-адмінпанель є не просто додатковим інструментом, а повноцінним компонентом системи управління спільнотою. Вона поєднує в собі сучасні технології (Flask, SQLAlchemy, Jinja2, Bootstrap), продуману архітектуру, високий рівень безпеки та зручний користувацький інтерфейс. Перехід від текстових команд у Discord до графічного інтерфейсу значно підвищив ефективність роботи модераторів, знизив кількість помилок і забезпечив необхідну прозорість модераційних дій. Панель демонструє, як сучасні веб-технології можуть ефективно доповнювати функціональність Discord-бота, створюючи комплексне рішення для управління спільнотами.

Таким чином, створена веб-адмінпанель не є просто додатковим інтерфейсом, а повноцінним потужним інструментом управління спільнотою, який органічно доповнює функціональність Discord-бота і виводить процес модерації на якісно новий рівень зручності, швидкості та прозорості.

РОЗДІЛ 3. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОГО РІШЕННЯ

Тестування розробленого програмного забезпечення є невід'ємною частиною кваліфікаційної роботи, оскільки дозволяє перевірити правильність реалізації, виявити можливі помилки та оцінити практичну ефективність створеного Discord-бота з веб-адмінпанеллю. У процесі тестування застосовувався комплексний підхід, який поєднував різні методи: unit-тестування, інтеграційне тестування та ручне тестування на реальному тестовому сервері.

Методика тестування була побудована на принципах поступового переходу від перевірки окремих модулів до оцінки роботи всієї системи в цілому. На першому етапі проводилося unit-тестування ключових функцій. Для цього тестувалися окремі компоненти, такі як функції нормалізації тексту в модулі `word_filters.py`, алгоритми виявлення флуду, дублікатів та CAPS SPAM у модулі `anti_spam.py`, а також логіка нарахування досвіду та розрахунку рівня в `xp_system.py`. Кожна функція перевірялася на різних тестових даних, включаючи граничні випадки (наприклад, повідомлення з максимальною кількістю символів, повідомлення з різними варіантами заміни символів у заборонених словах, крайні значення кулдауну в системі XP).

На другому етапі виконувалося інтеграційне тестування. Перевірялася взаємодія між модулями: правильність послідовності викликів (анти-спам - фільтр слів - система XP), коректність запису даних у базу даних після виконання команд, а також синхронізація між Discord-ботом і веб-панеллю. Особлива увага приділялася перевірці сценаріїв, коли кілька процесів відбуваються одночасно, наприклад, одночасна подача апеляції через Discord і її обробка через веб-інтерфейс.

Третій, найбільш важливий етап – ручне тестування на тестовому Discord-сервері. Для цього був створений спеціальний тестовий сервер з кількома ролями та каналами, де імітувалися реальні умови роботи спільноти. Тестування проводилося протягом кількох тижнів з участю тестових користувачів різного рівня активності. Перевірялися всі основні сценарії використання: приєднання нових учасників, надсилання звичайних і токсичних повідомлень, використання команд модерації, подача та обробка апеляцій, нарахування досвіду та відображення даних у веб-панелі. Також проводилося навантажувальне тестування – масове надсилання повідомлень для перевірки стабільності анти-спам модуля.

Результати тестування ключових модулів показали високу стабільність і правильність реалізації. Модуль анти-спаму ефективно виявляв флуд, повторювані повідомлення та CAPS SPAM. У більшості випадків система правильно застосовувала ескалацію покарань: від попередження до таймауту і бану. Помилкові спрацьовування були мінімальними і виникали лише в дуже специфічних випадках, які були враховані при подальшому доопрацюванні порогових значень.

Фільтр заборонених слів продемонстрував хорошу ефективність як для української, так і для англійської мови. Завдяки механізмам нормалізації та таблицям заміни символів вдалося виявляти значну частину завуальованих матюків. Тестування показало, що фільтр добре справляється навіть з повідомленнями, де використовується трансліт або спеціальні символи для маскування слів.

Система XP та рівнів працювала стабільно протягом усього періоду тестування. Кулдаун коректно запобігав накручуванню досвіду, а нарахування XP відбувалося рівномірно. Повідомлення про підвищення рівня та автоматична видача ролей Active Member і Pro Member спрацьовували без

затримок. Команди `/rank` і `/leaderboard` коректно відображали актуальну інформацію, включаючи прогрес-бар і позицію в таблиці лідерів.

Модуль апеляцій також пройшов успішне тестування. Користувачі могли без проблем подавати апеляції через команду в Discord, а модератори – обробляти їх як через чат, так і через веб-панель. При прийнятті апеляції бан знімався автоматично, а кількість попереджень обнулялася. Усі дії коректно фіксувалися в журналі ModLog.

Веб-панель загалом показала стабільну роботу. Усі сторінки завантажувалися швидко, дані відображалися правильно, а форми обробки апеляцій працювали без помилок. Фільтр `kuiv_time` коректно перетворював час на всіх сторінках. Адаптивність інтерфейсу була перевірена на різних пристроях – від десктопних моніторів до мобільних телефонів.

Оцінка ефективності розробленого рішення проводилася за кількома ключовими показниками. За період тестування на тестовому сервері було зафіксовано суттєве зменшення кількості спаму та токсичних повідомлень. Анти-спам модуль і фільтр заборонених слів разом дозволили автоматично обробляти більшість порушень без втручання модераторів. Середня швидкість реакції на порушення зменшилася в кілька разів порівняно з ручною модерацією.

Аналіз даних з бази даних показав, що система попереджень працює ефективно: більшість користувачів після першого-другого попередження припиняли порушувати правила. Автоматичний бан після трьох попереджень також спрацьовував коректно і слугував серйозним стримуючим фактором. Система апеляцій дозволила користувачам оскаржувати рішення, що підвищило загальний рівень довіри до модерації.

Система ХР та рівнів позитивно вплинула на активність учасників[16]. Користувачі, які регулярно отримували досвід, частіше брали участь в

обговореннях, що сприяло розвитку спільноти. Команди `/rank` і `/leaderboard` стали популярними серед учасників і стимулювали здорову конкуренцію. Загалом результати тестування та впровадження підтверджують, що розроблене рішення працює стабільно, ефективно виконує поставлені завдання і має високу практичну цінність для управління Discord-серверами.

Порівняння розробленого рішення з існуючими аналогами дозволяє об'єктивно оцінити його сильні та слабкі сторони[17]. У ході аналізу були розглянуті найпопулярніші Discord-боти для модерації – Carl-bot, MEE6, Dyno, Wick та AltDentifier. Порівняння проводилося за ключовими критеріями: функціональні можливості модерації, якість анти-спаму, підтримка української мови, наявність системи гейміфікації, зручність веб-інтерфейсу, механізм апеляцій та загальна інтеграція компонентів[27].

За функціоналом модерації розроблений бот демонструє високий рівень[17]. Він поєднує в собі потужну систему попереджень з автоматичним баном після трьох порушень, розширений анти-спам модуль з трьома рівнями виявлення (флуд, дублікати, CAPS SPAM) та двомовний фільтр заборонених слів. Більшість аналогів або добре справляються з одним напрямком, але рідко пропонують такий комплексний підхід. Особливо помітною перевагою є якісна підтримка української мови у фільтрі слів, чого майже немає в популярних рішеннях. Carl-bot і Dyno мають потужні інструменти модерації, але їхні фільтри погано працюють з українською лексикою та її варіаціями.

Система апеляцій також є сильною стороною розробки. У більшості аналогів цей механізм або відсутній, або реалізований дуже примітивно. У створеному рішенні користувач може подати апеляцію безпосередньо в Discord, а модератор – обробити її як через чат, так і через зручний веб-інтерфейс з можливістю додавання коментаря. Така гнучкість суттєво відрізняє розробку від конкурентів.

Щодо зручності використання веб-панель розробленого бота має відчутну перевагу. Більшість популярних ботів або взагалі не мають повноцінної веб-панелі (або вона доступна лише в платній версії), або пропонують обмежений функціонал. У даному рішенні модератор отримує єдиний централізований інтерфейс для перегляду журналу, роботи з попередженнями та обробки апеляцій.

Система ХР та рівнів також вигідно відрізняє розробку. Якщо МЕЕ6 має добре розвинену систему рівнів, то в ньому практично відсутня глибока інтеграція з модерацією. У створеному боті система досвіду працює паралельно з модулями модерації і не конфліктує з ними, що створює збалансоване середовище, де активні користувачі заохочуються, а порушники – караються.

Однак порівняння виявляє і певні недоліки розробленого рішення. На момент завершення кваліфікаційної роботи веб-панель не має повноцінної системи автентифікації з логіном і паролями – доступ здійснюється через простий сесійний механізм. Це спрощення було прийнято для прискорення розробки, але в реальному використанні на великих серверах потребує доопрацювання. Крім того, відсутній зручний дашборд з аналітикою (статистика спаму, активності користувачів, графіків тощо), який присутній у преміум-версіях деяких конкурентів.

Ще одним недоліком є відсутність підтримки додаткових мов, крім української та англійської. Також система поки що не містить елементів штучного інтелекту для більш розумної модерації (наприклад, аналіз контексту повідомлень або виявлення токсичності за змістом, а не лише за ключовими словами).

Перспективи розвитку рішення є досить широкими. У першу чергу доцільно реалізувати повноцінну систему автентифікації для веб-панелі з ролями користувачів (адміністратор, старший модератор, модератор)[36].

Другим важливим напрямком є створення розширеного дашборду з аналітикою: графіки активності користувачів, статистика спаму за періодами, теплові карти порушень та звіти для адміністраторів[37,38].

Перспективним є впровадження елементів штучного інтелекту. Наприклад, підключення моделей для аналізу контексту повідомлень дозволить виявляти приховану токсичність, сарказм або координацію рейдів. Також можна розширити підтримку мов, додавши фільтри для польської чи німецької, що зробить бота привабливим для міжнародних спільнот.

Іншим напрямком розвитку може стати покращення системи гейміфікації – додавання досягнень, сезонних подій, магазин ролей за ХР та інтеграція з голосовими каналами. Крім того, варто розглянути можливість створення конфігураційного інтерфейсу безпосередньо в веб-панелі, щоб адміністратори могли налаштовувати пороги анти-спаму, списки заборонених слів та параметри ХР без редагування коду.

Загалом, незважаючи на наявні недоліки, розроблене рішення вже на даному етапі перевершує більшість безкоштовних аналогів за комплексністю функціоналу та зручністю використання[39]. Подальше вдосконалення в напрямках безпеки, аналітики та застосування штучного інтелекту дозволить перетворити його на конкурентоспроможний інструмент не тільки для студентських і невеликих спільнот, але й для великих Discord-серверів[45].

ВИСНОВКИ

Метою кваліфікаційної роботи була розробка Discord-бота з веб-адмінпанеллю для автоматизації модерації, підвищення комфорту користування спільнотою та ефективного управління Discord-сервером. Поставлена мета була повністю досягнута. У ході роботи було спроектовано, реалізовано та протестовано комплексне програмне рішення, яке поєднує в собі потужний Discord-бот і зручну веб-адмінпанель.

Усі поставлені завдання виконано в повному обсязі. Проведено аналіз сучасних Discord-ботів та виявлено їхні ключові недоліки. Спроектовано архітектуру системи з урахуванням модульності, multi-guild підтримки та слабкої зв'язності компонентів. Реалізовано основні модулі бота, зокрема систему привітання нових учасників, розширену систему модерації з попередженнями та банами, потужний анти-спам модуль, двомовний фільтр заборонених слів, систему досвіду та рівнів, а також механізм подання та обробки апеляцій. Розроблено повноцінну веб-адмінпанель з можливістю перегляду журналу дій, управління попередженнями та обробки апеляцій. Забезпечено підтримку української та англійської мов, реалізовані вимоги безпеки та проведено комплексне тестування рішення.

Основними результатами роботи є повністю функціонуючий Discord-бот з веб-адмінпанеллю, який може працювати як на одному, так і на кількох серверах одночасно. Бот ефективно виявляє та блокує спам, фільтрує ненормативну лексику двома мовами, автоматично нараховує досвід учасникам, веде детальний журнал усіх модераційних дій і підтримує прозорий механізм апеляцій. Веб-панель значно спрощує роботу модераторів, надаючи зручний графічний інтерфейс замість набору текстових команд.

Практична цінність розробленого рішення є досить високою. Для Discord-спільнот, особливо українських та двомовних, бот пропонує сучасний

інструмент модерації, який поєднує автоматизацію з елементами гейміфікації. Модератори отримують суттєве зменшення рутинного навантаження завдяки автоматичній обробці порушень і зручній веб-панелі для контролю та прийняття рішень. Адміністратори серверів можуть ефективніше керувати спільнотою, підвищувати її активність через систему XP та рівнів, а також підтримувати прозорість модераційних дій.

Для студентів та молодих розробників робота має також освітню цінність[34]. Вона демонструє реальний приклад комплексної розробки програмного забезпечення: від аналізу предметної області та проектування архітектури до реалізації, інтеграції різних технологій, тестування та впровадження. Проєкт поєднує в собі знання з веб-розробки, роботи з API, баз даних, асинхронного програмування та створення користувацьких інтерфейсів. Незважаючи на досягнуті результати, робота має певні перспективи подальшого вдосконалення[35].

У майбутньому можна розширити підтримку додаткових мов, додати можливість гнучкого налаштування всіх параметрів бота безпосередньо через веб-інтерфейс, а також реалізувати інтеграцію з голосовими каналами та сезонними подіями в системі гейміфікації. Окремим напрямком розвитку може стати перехід від SQLite до більш потужної бази даних (наприклад, PostgreSQL) для підтримки дуже великих спільнот.

Отже, виконана кваліфікаційна робота підтверджує можливість створення сучасного, ефективного та зручного інструменту для управління Discord-спільнотами. Розроблене рішення поєднує в собі технічну складність, практичну корисність і перспективи подальшого розвитку, що робить його цінним як з освітньої, так і з прикладної точки зору[29].

Для отримання додаткової інформації та взаємодії з Discord-ботом, переходьте за посиланням: Discord-бот, де можна спробувати всі можливості цього продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Discord Developer Documentation. Discord API Reference [Електронний ресурс]. Режим доступу: <https://discord.com/developers/docs/reference> (дата звернення: 12.04.2026).
2. Rapptz. discord.py Documentation. Version 2.3 [Електронний ресурс]. Режим доступу: <https://discordpy.readthedocs.io/en/stable/> (дата звернення: 05.04.2026).
3. The Pallets Team. Flask Documentation. Version 3.0 [Електронний ресурс]. Режим доступу: <https://flask.palletsprojects.com/en/3.0.x/> (дата звернення: 18.04.2026).
4. SQLAlchemy Documentation. Version 2.0 [Електронний ресурс]. Режим доступу: <https://docs.sqlalchemy.org/en/20/> (дата звернення: 22.04.2026).
5. Bootstrap Documentation. Version 5.3 [Електронний ресурс]. Режим доступу: <https://getbootstrap.com/docs/5.3/> (дата звернення: 09.04.2026).
6. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. O'Reilly Media, 2018. – 312 p.
7. Python Software Foundation. Python 3.12 Documentation [Електронний ресурс]. Режим доступу: <https://docs.python.org/3/> (дата звернення: 14.04.2026).
8. Jinja Documentation. Version 3.1 [Електронний ресурс]. Режим доступу: <https://jinja.palletsprojects.com/> (дата звернення: 25.04.2026).
9. ZoneInfo – IANA Time Zone Support in Python [Електронний ресурс]. Python Docs, 2024. Режим доступу: <https://docs.python.org/3/library/zoneinfo.html> (дата звернення: 07.04.2026).
10. Nielsen J. Usability Engineering. Morgan Kaufmann, 1993. 384 p. (Перевидання 2022).

11. Krug S. Don't Make Me Think: A Common Sense Approach to Web Usability. 3rd ed. New Riders, 2014. 216 p.
12. Richards M. Software Architecture Patterns. O'Reilly Media, 2020. 198 p.
13. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley, 2021. – 464 p.
14. Real Python. Building a Discord Bot with Python [Електронний ресурс]. 2024. Режим доступу: <https://realpython.com/discord-bot-python/> (дата звернення: 11.04.2026).
15. GeeksforGeeks. Python Discord Bot using Discord.py [Електронний ресурс]. 2024. Режим доступу: <https://www.geeksforgeeks.org/python-discord-bot-using-discord-py/> (дата звернення: 29.04.2026).
16. Huynh D., Lim S., Skoric M. M. Stepping Up: Gamification and Civil Disobedience in Online Communities // Proceedings of the 2023 CHI Conference. 2023. DOI: 10.1145/3544548.
17. IEEE Access. Comprehensive Review of Online Community Moderation Systems. 2024. Vol. 12. Pp. 1420–1445.
18. Проценко О. В. Методи автоматизації модерації контенту в інформаційних системах // Вісник НТУ «ХПІ». Серія: Системний аналіз. 2024. № 2. С. 34–41.
19. Антонюк Ю. М. Архітектура систем керування студентськими онлайн-спільнотами // Комп'ютерні науки та інформаційні технології. 2025. Вип. 14. С. 102–110.
20. Бондаренко М. О. Розробка веб-інтерфейсів для моніторингу розподілених систем на базі Flask // Наукові праці Донецького національного технічного університету. 2025. № 1. С. 56–63.

21. Ковальчук П. Р. Порівняльний аналіз ORM-бібліотек для мови програмування Python // Сучасні інформаційні технології. 2024. № 4. – С. 22–29.
22. Положення про порядок виконання, оформлення та захисту кваліфікаційних робіт у Державному податковому університеті. Ірпінь: ДПУ, 2022. – 45 с.
23. GitHub. discord.py: Asyncio-based Discord API Wrapper [Електронний ресурс]. 2025. Режим доступу: <https://github.com/Rapptz/discord.py> (дата звернення: 16.04.2026).
24. Flask-SQLAlchemy Documentation. Version 3.1 [Електронний ресурс]. 2024. Режим доступу: <https://flask-sqlalchemy.palletsprojects.com/> (дата звернення: 03.04.2026).
25. Python-dotenv Documentation [Електронний ресурс]. 2023. Режим доступу: <https://pypi.org/project/python-dotenv/> (дата звернення: 21.04.2026).
26. Bootstrap 5: The Most Popular HTML, CSS, and JS Library [Електронний ресурс]. 2025. Режим доступу: <https://getbootstrap.com/> (дата звернення: 10.04.2026).
27. Stack Overflow Developer Survey 2025 [Електронний ресурс]. Режим доступу: <https://survey.stackoverflow.co/2025> (дата звернення: 30.04.2026).
28. Real Python. Working with Databases in Flask [Електронний ресурс]. 2024. Режим доступу: <https://realpython.com/flask-connexion-rest-api/> (дата звернення: 15.04.2026).
29. Хабр. Розробка Discord-ботів: від простого до складного [Електронний ресурс]. 2024. Режим доступу: <https://habr.com/ru/articles/> (дата звернення: 13.04.2026).

- 30.DOU.ua. Досвід розробки ботів для українських ІТ-спільнот [Електронний ресурс]. 2025. Режим доступу: <https://dou.ua/lenta/articles/> (дата звернення: 26.04.2026).
- 31.Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software.Addison-Wesley, 1994. (Перевидання 2023).
- 32.Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson, 2017. – 432 p.
- 33.Матеріали науково-практичної конференції «Інформаційні технології в сучасному світі». Ірпінь: ДПУ, 2026. – С. 120–125.
- 34.Ткаченко О. М. Використання мікрофреймворків для створення адміністративних панелей // Цифрова економіка та системи автоматизації. 2026. № 1. С. 15–21.
- 35.Офіційні методичні рекомендації до написання кваліфікаційної роботи спеціальності 122 «Комп'ютерні науки». Ірпінь: ДПУ, 2024. – 35 с.
- 36.IEEE Xplore. AI-Powered Content Moderation: Challenges and Opportunities. 2025. Vol. 13. Pp. 88–95.
- 37.ACM Digital Library. Designing Moderation Tools for Scalable Online Communities // CSCW 2025 Proceedings. 2025.
- 38.Python Package Index (PyPI). Project statistics for discord.py [Електронний ресурс]. 2026. Режим доступу: <https://pypi.org/project/discord.py/> (дата звернення: 19.04.2026).
- 39.Офіційний сайт Державного податкового університету [Електронний ресурс]. Режим доступу: <https://www.dpu.edu.ua/> (дата звернення: 02.04.2026).
- 40.Raprtz. discord.py GitHub Repository .Best Practices [Електронний ресурс]. 2025. Режим доступу:

<https://github.com/Rapptz/discord.py/tree/master/examples> (дата звернення: 20.04.2026).

41. The Pallets Projects. Flask: Design and Best Practices [Електронний ресурс]. 2024. Режим доступу: <https://flask.palletsprojects.com/en/3.0.x/design/> (дата звернення: 23.04.2026).

42. Real Python. Using Flask to Build a Web Dashboard for Your Discord Bot [Електронний ресурс]. 2025. Режим доступу: <https://realpython.com/> (дата звернення: 17.04.2026).

43. Krug S. Don't Make Me Think Revisited: A Common Sense Approach to Web Usability. New Riders, 2014.

44. Nielsen Norman Group. World Leaders in Research-Based User Experience [Електронний ресурс]. 2026. Режим доступу: <https://www.nngroup.com/> (дата звернення: 08.04.2026).

45. Шевченко А. В. Побудова систем моніторингу Discord-серверів на базі веб-інтерфейсів // Збірник наукових праць молодих вчених. 2026. С. 45–51.

Додатки

Додаток А

Лістинг коду `main.py`

```
from app import app
from bot import setup_bot, run_discord_bot
import threading
if name == "main":
    # Створюємо та запускаємо бота у окремому потоці
    discord_bot = setup_bot()
    bot_thread = threading.Thread(target=run_discord_bot, args=(discord_bot,))
    bot_thread.daemon = True
    bot_thread.start()
    # Запускаємо Flask-додаток
    app.run(host='0.0.0.0', port=5000, debug=True, use_reloader=False)
```

Додаток В

Лістинг коду `App.py`

```
import os
import logging
from flask import Flask, render_template, redirect, url_for, request
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase
from datetime import datetime

logging.basicConfig(level=logging.INFO)

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)
app = Flask(name)
app.secret_key = os.environ.get("FLASK_SECRET_KEY") or "секретний_ключ"
app.config["SQLALCHEMY_DATABASE_URI"] = os.environ.get("DATABASE_URL",
"sqlite:///bot.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db.init_app(app)
from datetime import timezone
```

```

from zoneinfo import ZoneInfo
KYIV_TZ = ZoneInfo("Europe/Kyiv")
@app.template_filter('kyiv_time')
def kyiv_time_filter(dt):
    if dt is None:
        return ""
    if dt.tzinfo is None:
        dt = dt.replace(tzinfo=timezone.utc)
    local = dt.astimezone(KYIV_TZ)
    return local.strftime("%d.%m.%Y %H:%M:%S")
with app.app_context():
    import models
    db.create_all()

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/logs')
def logs():
    with app.app_context():
        all_logs =
        db.session.query(models.ModLog).order_by(models.ModLog.created_at.desc()).all()
    return render_template('logs.html', logs=all_logs)

@app.route('/warnings')
def warnings():
    with app.app_context():
        all_warnings = db.session.query(models.UserWarning).order_by(
            models.UserWarning.is_banned.desc(),
            models.UserWarning.warning_count.desc()
        ).all()
    return render_template('warnings.html', warnings=all_warnings)

@app.route('/appeals')
def appeals():
    with app.app_context():
        all_appeals = db.session.query(models.Appeal).order_by(
            models.Appeal.created_at.desc()
        ).all()
    return render_template('appeals.html', appeals=all_appeals)

```

```

@app.route('/appeals/<int:appeal_id>/approve', methods=['POST'])
def appeal_approve(appeal_id):
    reason = request.form.get('reason', "").strip() or 'Прийнято через веб-панель'
    with app.app_context():
        appeal = db.session.get(models.Appeal, appeal_id)
        if appeal and appeal.status == 'pending':
            appeal.status = 'approved'
            appeal.resolved_at = datetime.utcnow()
            appeal.resolved_by = 'web'
            appeal.resolution_reason = reason
            user_warning = db.session.query(models.UserWarning).filter_by(
                user_id=appeal.user_id,
                server_id=appeal.server_id
            ).first()
            if user_warning:
                user_warning.warning_count = max(0, user_warning.warning_count - 1)
            if user_warning.warning_count < 3:
                user_warning.is_banned = False
                user_warning.banned_at = None
                log = models.ModLog(
                    user_id=appeal.user_id,
                    username=appeal.username,
                    server_id=appeal.server_id,
                    server_name=appeal.server_name or "",
                    channel_id='0',
                    channel_name='web',
                    action='appeal_approved',
                    reason=f'Апеляція #{appeal_id} прийнята через веб: {reason}',
                    content_preview=None,
                    language=None
                )
            db.session.add(log)
            db.session.commit()
            return redirect(url_for('appeals'))

```

```

@app.route('/appeals/<int:appeal_id>/reject', methods=['POST'])
def appeal_reject(appeal_id):
    reason = request.form.get('reason', "").strip() or 'Відхилено через веб-панель'
    with app.app_context():
        appeal = db.session.get(models.Appeal, appeal_id)

```

```

if appeal and appeal.status == 'pending':
    appeal.status = 'rejected'
    appeal.resolved_at = datetime.utcnow()
    appeal.resolved_by = 'web'
    appeal.resolution_reason = reason
    log = models.ModLog(
        user_id=appeal.user_id,
        username=appeal.username,
        server_id=appeal.server_id,
        server_name=appeal.server_name or "",
        channel_id='0',
        channel_name='web',
        action='appeal_rejected',
        reason=f'Апеляція #{appeal_id} відхилена через веб: {reason}',
        content_preview=None,
        language=None
    )
    db.session.add(log)
    db.session.commit()
    return redirect(url_for('appeals'))

@app.route('/health')
def health():
    return "OK", 200

```