Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка алгоритмів та програмних засобів для керування каналами і чатами у месенджерах»

Виконав: студент IV курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Стецула М. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Мельник О. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Богомолов С. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Стецулі Мар'яну Васильовичу

1. Тема роботи – розробка алгоритмів та програмних засобів для керування каналами і чатами у месенджерах.

Керівник роботи: Мельник Олександр Васильович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи: 2 червня 2025.

3. Вихідні дані до роботи: базові методи керування ботами – методи універсальної обробки команд і повідомлень; моделі реагування – моделі визначення команд користувача, моделі класифікації повідомлень; режим обробки даних – реальний час; вихідні дані для обробки – текстові та медіа повідомлення; дані про користувачів у чаті, права доступу, типи повідомлень; коефіцієнт пріоритетності обробки; ідентифікатори чатів, каналів, користувачів; вихідні дані – виконані дії бота відповідно до обраного алгоритму, оновлений стан каналів та чатів.

4. Зміст текстової частини: розрахунок оптимальної обробки повідомлень; логічні та структурні властивості алгоритмів обробки; аналіз моделей класифікації типів команд; класифікація алгоритмів реагування бота; аналіз практичної цінності розроблених алгоритмів; застосування симетричних структур даних для оптимізації роботи; методи обробки, засновані на аналізі частоти команд та повідомлень; підвищення ефективності алгоритмів шляхом урахування пріоритетності завдань та розподілу ресурсів; розробка структури системи керування ботом; розробка інтерфейсу керування; розробка загального алгоритму роботи системи; розробка алгоритму моделювання сценаріїв реагування; економічна частина.

5. Перелік ілюстративного матеріалу: галузі застосування алгоритмів керування ботами; основні етапи обробки повідомлень у чатах та каналах; моделі класифікації типів повідомлень; інтерфейс системи для керування алгоритмами та налаштування бота.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Мельник Олександр Васильович	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз API месенджерів та існуючих ботів	25.03.25 – 06.04.25	Вик
2	Формування функціональних вимог до бота	07.04.25 – 20.04.25	Вик
3	Розробка архітектури бота та алгоритмів	20.04.25 – 07.05.25	Вик
4	Реалізація програмного модуля та тестування	07.05.25 – 20.05.25	Вик

Студент _____ (підпис) **М. В. Стецула** (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **О. В. Мельник** (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.4

Стецула М.В. Бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 92 с.

На укр. мові. Бібліогр.: назва: розробка алгоритмів та програмних засобів для керування каналами і чатами у месенджерах; розділів: 4; рисунків: 43; таблиць: 5.

У бакалаврській кваліфікаційній роботі проведено детальний аналіз алгоритмів для керування чатами та каналами у месенджерах.

Запропоновано універсальні алгоритми обробки повідомлень, модерації, розподілу прав доступу та керування подіями в реальному часі.

Розроблено систему, яка використовує мову програмування Python, бібліотеку telegram.ext та середовище розробки Visual Studio Code для побудови гнучкого інструменту керування.

Реалізовано модулі для блокування та розблокування користувачів, створення таймерів, модерації, логування, обробки callback-запитів і багаторівневих меню.

Отримані результати можна застосовувати для зниження навантаження на адміністраторів та підвищення ефективності управління великими спільнотами.

Ключові слова: боти, модерація, автоматизація, API, алгоритми.

ABSTRACT

Stetsula M.V. Bachelor's thesis in specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 92 c.

In Ukrainian. Bibliogr.: title: development of algorithms and software tools for managing channels and chats in messengers; titles: 4; figures: 43; tables: 5.

In the bachelor's thesis, a detailed analysis of algorithms for managing chats and channels in messengers was carried out.

Universal algorithms for message processing, moderation, access rights distribution, and real-time event management were proposed.

A system has been developed that uses the Python programming language, the telegram.ext library, and the Visual Studio Code development environment to build a flexible management tool.

Modules for blocking and unblocking users, creating timers, moderation, logging, processing callback requests, and multi-level menus have been implemented.

The results obtained can be used to reduce the workload of administrators and increase the efficiency of managing large communities.

Keywords: bots, moderation, automation, API, algorithms.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Швидкий розвиток цифрових технологій формує нові вимоги до інструментів взаємодії з користувачами. Зростає актуальність автоматизації управління каналами та чатами у месенджерах, що стали важливою частиною особистої, ділової й суспільної комунікації. Зі збільшенням кількості користувачів потрібні більш ефективні засоби управління великими спільнотами й автоматизації однотипних процесів [1].

Керування каналами і чатами сьогодні виходить за межі ручного керування повідомленнями: воно охоплює організацію доступу, модерацію контенту, аналіз учасників і повідомлень, забезпечення безпеки та швидке реагування. Ручне виконання цих завдань неефективне і веде до хаосу, особливо у великих групах. Автоматизація через ботів значно знижує навантаження на адміністраторів і підвищує ефективність керування. Боти надають широкі можливості: обробку запитів, управління спільнотами, автоматизацію процесів, ведення журналів. Проте більшість рішень мають обмежений функціонал або потребують оплати за розширені можливості. Бракує універсальних безкоштовних систем для повного управління [2, 3].

Тому актуальним є розробка і впровадження алгоритмів, які керують спільнотами та автоматизують модерацію, блокування користувачів, аналіз повідомлень, керування правами доступу та інтерактивну взаємодію з користувачами. Це дозволяє підвищити ефективність адміністрування каналів і чатів, зменшити навантаження на модераторів та забезпечити стабільну роботу спільнот.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності керування каналами та чатами за рахунок створення

універсального бота для месенджера, який автоматизує базові дії, забезпечує модерацію, управління доступами та логування подій, що дозволяє мінімізувати витрати часу на адміністративні завдання.

Основні задачі досліження:

– розробка алгоритмів обробки адміністративних команд та подій у чатах і каналах;

– створення модуля для управління доступами учасників відповідно до заданих правил;

– забезпечення простого та інтуїтивно зрозумілого інтерфейсу взаємодії з ботом;

– проведення тестування всіх модулів для виявлення та усунення можливих помилок.

Об’єкт дослідження. Об’єктом дослідження є процес організації ефективного керування спільнотами у месенджерах.

Предмет дослідження. Предметом дослідження є методи та засоби автоматизації керування чатами та каналами в месенджерах.

Методи дослідження. Для реалізації поставлених задач використовувалися емпіричні методи розробки ботів на Python через бібліотеку telegram.ext; елементарна алгебра для інтеграції з API месенджера; алгоритмічні методи обробки повідомлень у реальному часі; методи організації прав доступу, а також методи тестування програмних систем.

Новизна отриманих результатів.

1. Подальшого розвитку набули методи автоматизації керування чатами та каналами, що об’єднуються в єдину систему функцій, зокрема модерацію, автоматичне блокування та розблокування користувачів, динамічне формування інтерактивних меню через callback-запити та розподіл прав доступу, що дозволило оптимізувати процес модерації.

2. Запропоновано метод підвищення ефективності управління спільнотами, який відрізняється від відомих методів тим, що не використовує інструменти штучного інтелекту, що дозволяє зменшити втручання людини та

забезпечити стабільність каналів навіть у періоди високої активності користувачів.

Практична цінність отриманих результатів. Практична цінність результатів роботи полягає в тому, що на основі теоретичних положень, розроблено програмні засоби для месенджера, що можуть бути використані адміністраторами каналів і чатів для ефективного керування спільнотами, автоматизації рутинних задач та забезпечення якісної модерації без постійного втручання з боку людини.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати бакалаврської кваліфікаційної роботи доповідалися на LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії 2025.

Публікації. Результати роботи доповідалися на LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії 2025 [4].

1 АНАЛІЗ СТАНУ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ІНСТРУМЕНТІВ КЕРУВАННЯ ЧАТАМИ В МЕСЕНДЖЕРАХ

1.1. Аналіз сучасних месенджерів та їх API

Месенджер – це програма або платформа на телефоні або комп’ютері, що дозволяє користувачам вести приватні і групові чати в інтернеті [5]. Він дозволяє надсилати повідомлення, виконувати голосові та відео дзвінки, обмінюватися файлами, фотографіями, відео та іншим контентом. Месенджери стали невід'ємною частиною повсякденного життя, замінивши традиційні SMS і часто навіть електронну пошту завдяки зручності, швидкості та додатковим можливостям.

До найпопулярніших месенджерів належать WhatsApp, Facebook Messenger, Telegram, Discord і Viber [6]. Кожний із них має різні додаткові функції, наприклад створення групових чатів, ботів для автоматизації чи каналів для новин або комерційного використання.

Сучасні месенджери є багатофункціональними платформами, що забезпечують обмін різними типами повідомлень між користувачами, а також мають підтримку каналів, чатів, групових обговорень і різних інструментів для автоматизації. Основною особливістю таких платформ є наявність відкритих API, що дозволяють розробникам створювати сторонні застосунки, зокрема ботів, які можуть інтегруватися у внутрішню систему месенджера.

API (Application Programming Interface) – це інтерфейс програмування застосунків, який дозволяє стороннім розробникам або сервісам взаємодіяти з функціональністю месенджера програмним шляхом, за допомогою якого можна наступне [7]:

- отримування інформації про користувачів;
- надсилання й отримування повідомлень;
- створення чатів;
- інтегрування ботів;
- керування учасниками;

- налаштування прав доступу;
- автоматичне керування та адміністрування;
- відстежування активності.

Перевагою використання API месенджерів є можливість створення різних програмних рішень, які можуть адаптуватися до потреб конкретних спільнот. Це включає не лише елементи автоматизації, а й побудову складної взаємодії, яка базуються на повідомленнях, поведінці користувачів та інших подій. У результаті API не лише реагує на події, а й може прогнозувати або попереджати про небажану активність.

API месенджерів безперервно розвивається, тому з’являються нові методи, події та типи об'єктів, що розширюють можливості функціоналу алгоритмів. Це дозволяє розробникам покращувати свої існуючі алгоритми для середовища, підтримувати сумісність і використовуючи нові можливості їх API.

1.2. Існуючі методи керування каналами та чатами

Існуючі методи базуються на використанні спеціальних алгоритмів і логічних конструкцій, які дозволяють автоматизувати роботу з великими обсягами інформації та значною кількістю учасників. Основна мета таких методів полягає в спрощенні адміністрування, автоматизації рутинних завдань, модерації контенту, розподілі прав доступу, плануванні публікацій, а також аналізі користувачів [8]. Відомі методи включають системи тригерів, що реагують на певні слова або події, які виконують заздалегідь визначені сценарії, де дії прописані наперед.

Найпоширеніший метод – це використання системи фільтрації контенту, що дозволяють в автоматичному режимі видаляти небажані повідомлення, блокувати спам або попереджати учасників про порушення правил спільноти.

Наступний метод – це керування правами доступу корисувачів до алгоритмів, що передбачає надання або обмеження команд залежно від ролі користувача.

Інший метод заключається в планувані контенту, коли бот за розкладом публікує заготовлені повідомлення або проводить інформаційні новини.

Ще один метод пов'язаний з логуванням повідомлень спільноти чи користувачів, коли алгоритми збирають дані про активність, кількість повідомлень, частоту порушень, а потім формують звіти на основі отриманих показників.

Всі ці методи можна об'єднати в одну систему, яка називається бот.

Боти – це спеціалізовані програмні системи, що автоматично виконують визначені дії у відповідь на події чи команди. Боти стали одним із основних інструментів для спільнот, оскільки дозволяють швидко і зручно реагувати на запити, виконувати задачі без втручання людини та забезпечують постійний контроль над спілкуванням. Їх функціональність значною мірою залежить від можливостей API, через який бот отримує доступ до інформації, надсилає повідомлення, модерує чат або виконує дії з учасниками.

Боти можуть виконувати самі різні функції – від простого реагування на текстові команди до складної логіки. Наприклад, бот може як просто привітати нового учасника в чаті, так і профільтрувати і проаналізувати весь чат за короткий проміжок часу. Таким чином, бот виконує роль адміністратора в управліні каналом або чатом. Головна перевага ботів полягає в їх здатності працювати автономно, цілодобово та без помилок, що часто притаманні людині.

1.3. Аналіз існуючих ботів для керування каналами та чатами

У сучасних месенджерах з активними спільнотами вимагається не просто ручне адміністрування, а цілісний процес автоматичного керування. Це включає модерацію, логування, захист від спаму, автоматизацію взаємодії з учасниками та оптимізацію рутинних завдань.

Найбільш відомими аналогами є:

– @PuzzleBot;

– @MissRose_bot;

– @junction_bot;
– @ChatNorrisBot;
– @TheFeedReaderBot.

PuzzleBot є популярним ботом, який дозволяє планувати публікації, редагувати повідомлення, додавати інтерактивні кнопки та збирати статистику. Проте його функціонал у базовій версії обмежений, а розширені можливості потребують оплати. Інтерфейс взаємодії реалізований через командну систему та інлайн-кнопки, що передбачає певне навчання користувача.

Rose Bot забезпечує широку підтримку модерації за допомогою детальної системи команд, що дозволяє ефективно реагувати на порушення правил чату та швидко блокувати небажаних учасників. Основний акцент зроблено на оперативному управлінні спільнотою, що є важливим у динамічних групах. Однак бот має слабко розвинену аналітичну складову, що обмежує його ефективність у великих спільнотах, де необхідний більш комплексний підхід до збору та обробки даних про активність учасників.

Junction Bot орієнтований переважно на адміністраторів каналів, пропонуючи автоматизацію агрегування контенту з різних джерел, що дозволяє спрощено публікувати новини та інший матеріал у каналах. Він має просту інтеграцію, що робить його зручним для швидкого налаштування. Однак його можливості обмежуються лише пересиланням повідомлень без глибокої модерації чи логування, що робить його менш ефективним для великих спільнот, де важливо управляти взаємодією користувачів і моніторити активність.

ChatNorrisBot відзначається простотою у використанні, тим що він призначений для створення коротких резюме чату та підсумків активностей. Це робить його ідеальним для динамічних груп, але його функціонал обмежується лише збором та відображенням повідомлень без глибокої модерації чи автоматизації.

Feed Reader Bot пропонує розширені можливості інтеграції RSS-стрічок у чати або канали, дозволяючи автоматично публікувати новини з різних джерел.

Водночас його модуль логування обмежений, а налаштування може вимагати деякого часу для адаптації під конкретні джерела новин, що може бути незручним для адміністраторів, які шукають швидке налаштування та безперебійний процес публікації. Окрім того, цей бот не надає розширених можливостей для модерації або інтерактивної взаємодії з користувачами, що обмежує його функціональність у великих і активних спільнотах [9].

Результати порівняння аналогів зведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	**Stellar Bot (Власний бот)**	**Puzzle Bot**	**Rose Bot**	**Junction Bot**	**ChatNorris Bot**	**Feed Reader Bot**
Автоматизація керування	+	+	+	+	-	+
Модерація контенту	+	+	+	-	-	-
Інтуїтивність взаємодії	+	-	-	-	+	-
Простота налаштувань	+	+	+	+	+	+
Швидкість обробки команд	+	+	+	+	+	+
Загальна оцінка (1-10)	9	8	7	6	6	7

Попри наявність різноманітних готових ботів, жоден із них не пропонує повного набору цих функцій, який би задовольняв усі можливі потреби сучасних адміністраторів.

Власний бот перевершує аналоги за більшістю важливих критеріїв, особливо у зручності використання та інтеграції, однак показник зростатиме разом із вдосконаленнями. Це дозволить не лише покрити потреби сучасних адміністраторів, а й забезпечити потенціал для розширення функціоналу у майбутньому відповідно до вимог користувачів.

Отже, розробка бота для керування каналами та чатами у месенджерах відкриває широкі можливості для підвищення ефективності керування спільнотами, автоматизації рутинних процесів та покращення взаємодії між учасниками завдяки зручній, інтуїтивній роботі через чат-інтерфейси.

1.4. Аналіз аналогічних рішень і бібліотек

Перед розробкою алгоритмів потрібно враховувати не лише архітектурні особливості месенджерів, а і їх API, в якому варто виділити обмеження швидкості запитів, ліміти на кількість повідомлень, політики безпеки, обробку помилок, а також можливості інтегрування майбутнього функціоналу. Кожен месенджер реалізує свої протоколи взаємодії, що впливає на побудову універсальних алгоритмів. Тому доцільно розглядати не лише функціональні можливості API, а й специфічні особливості обробки даних, формати відповіді, структуру запитів та методи автоматизації.

Боти, які працюють через API – це одне із багатьох рішень для керування каналами та чатами.

Іншим рішеням є інтегровані інструменти в самому месенджері. Багато сучасних месенджерів мають вбудовані засоби керування. Це повністю клієнтські засоби, які не потребують зовнішніх систем.

Другим рішенням є ручне керування. Це найпростіша і базова форма керування каналами і чатами, яка здійснюється без використання сторонніх засобів автоматизації. Адміністратор або модератор особисто виконує всі дії в режимі реального часу.

Окрему увагу при розробці варто приділяти архітектурним рішенням, що використовуються в бібліотеках. Це стосується підтримки модульної

структури, можливості створення середовища тестування, масштабованості, а також обробки виключних ситуацій у режимі реального часу.

1.5. Висновки

У першому розділі було розглянуто існуючі рішення для керування каналами та чатами в месенджерах і їх сильні та слабкі сторони. Розглянуто специфіку API месенджерів, що дозволило зрозуміти технічні можливості та обмеження при створенні системи для адміністрування спільнот. Проаналізовано основні методи керування, зокрема автоматичну модерацію, планування контенту, логування та управління доступом. Значну увагу приділено ботам як основному засобу реалізації таких методів. Проведено аналіз існуючих ботів, їх функціональності, недоліків і переваг.

2 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ КЕРУВАННЯ КАНАЛАМИ НА ЧАТАМИ

2.1. Постановка задачі керування каналами та чатами

Сучасний розвиток технологій супроводжується активним поширенням месенджерів, які стали головними платформами для комунікації людей між собою. З ростом кількості користувачів та об'ємом інформації, що проходить через канали та чати, виникає потреба у нових інструментах для ефективного керування цими спільнотами [10].

Спочатку адміністрування чатів і каналів здійснювалося вручну, однак зі зростанням розмірів цей підхід виявився малоефективним. Проблеми модерації контенту, управління учасниками, забезпечення безпеки та реагування на порушення правил спільноти стали проблемою. Це призвело до появи різноманітних рішень, серед яких особливе місце займають боти, які керують або допомагають з каналами та чатами.

Основними напрямами розвитку цієї технології є видалення спаму, бан учасників за порушення правил, фільтрація забороненого контенту, аналіз користувачів, складання звітів, налаштування прав учасників, організація реєстрації та авторизації новачків у спільнотах та інтеграція з іншими сервісами для розширення можливостей ботів [11].

Технічний прогрес у сфері обчислювальних потужностей і розвитку API месенджерів суттєво полегшив створення ботів для месенджерів. Сучасні рішення можуть забезпечувати роботу у великих спільнотах без затримок, підтримувати одночасну обробку тисяч запитів та мати високий рівень надійності [12].

Доступні сьогодні рішення часто мають обмежений функціонал або вимагають плати за використання більшості функцій. Існує дефіцит універсальних безкоштовних рішень, які б поєднували автоматизацію і допомогу в керувані, гнучке налаштування та простий інтерфейс взаємодії.

Постановка задачі полягає у створенні універсальних і гнучких

алгоритмів, які дозволяють автоматизувати широкий спектр завдань: від модерації контенту та обробки запитів користувачів до планування публікацій і логування. Важливо, щоб такі алгоритми не були жорстко прив’язані до одного конкретного середовища, а мали адаптивну структуру, здатну працювати з різними конфігураціями чатів, каналів і груп.

Отже, аналіз показує високу актуальність розробки власного рішення, орієнтованого на автоматизацію часто виконуваних завдань, зменшення спаму в каналах та чатах і зручність для адміністраторів. Необхідність постійного вдосконалення технологій управління спільнотами зумовлює інтерес до розробки нових програмних продуктів, які зможуть відповідати сучасним вимогам ефективності, надійності та простоти використання.

2.2. Визначення основних функціональних вимог до бота

Розробка бота для месенджеру вимагає комплексного підходу до вибору метода реалізації основних функцій, таких як модерація, захист від спаму та автоматизація функцій. Серед можливих методів можна виокремити пару основних.

1. Перший метод полягає у використанні готової системи через інтеграцію з існуючими API. Таке рішення дозволяє реалізувати майже весь функціонал месенджера через бібліотеки. Вони забезпечують стабільність роботи з месенджером та мають офіційну підтримку. Проте цей підхід обмежує можливості створення унікального функціоналу.

2. Другий метод базується на розробці власних алгоритмів бота з мінімальним використанням сторонніх бібліотек, що дає змогу створити унікальні алгоритми взаємодії та більш гнучко керувати данними. Такий метод вимагає більше часу на розробку і тестування, однак забезпечує більшу функціональність системи під потреби конкретної спільноти.

Найкращим рішенням стало комбінувати ці два підхода. Тобто, створення власної архітектури бота з підтримкою готових бібліотек для обробки повідомлень і команд і розширення функціоналу за допомогою самостійно

реалізованих модулів модерації та автоматизації. Такий підхід забезпечить баланс між швидкістю розробки, гнучкістю рішення та можливістю подальшого додавання функцій.

В результаті буде створенний універсальний алгоритм, який поєднує інтуїтивність взаємодії, розширені можливості керування спільнотами та підвищену ефективність адміністрування.

Було визначено такі завдання, які необхідно виконати для успішної розробки програмного забезпечення:

– розробити алгоритм керування каналами та чатами через текстовий чат-інтерфейс, що забезпечить зручність взаємодії без використання традиційних візуальних елементів;

– створити алгоритми автоматизації модерації контенту, включаючи фільтрацію спаму, заборонених слів та небажаного контенту;

– створити модуль гнучкого керування правами доступу адміністраторів та модераторів;

– реалізувати систему сповіщень про важливі події в чатах і каналах без перевантаження інтерфейсу;

– забезпечити високу надійність, захист даних і стабільність роботи програмного рішення;

– провести тестування програмного забезпечення відповідно до поставлених функціональних вимог.

Отже, були сформульовані основні функціональні вимоги до бота, які наведено в таблиці 2.1 в якій структуровано подано перелік доступних дій залежно від рівня доступу, назви функцій та короткий опис їх призначення.

Таблиця 2.1 – Функціональні вимоги до бота

Права доступу	**Назва**	**Опис**
Модератор	Блокування користувачів	Модератор блокує користувачів через чат.

Продовження таблиці 2.1.

Права доступу	**Назва**	**Опис**
Модератор	Розблокування користувачів	Модератор розблоковує користувача за ID або у відповідь на повідомлення.
Модератор	Автоматичне блокування	Включає автоматичне блокування користувачів за заборонені слова.
Модератор	Запуск і зупинка таймера	Модератор запускає або зупиняє зворотний таймер у каналі з повідомленням по завершенню.
Модератор	Надсилання повідомлень у канал	Модератор надсилає повідомлення в канал від імені бота.
Модератор	Режим аналізу	Вмикає або вимикає режим аналізу повідомлень в реальному часі. Логування повідомлень відбувається завжди.
Модератор	Отримання листів	Отримує приватні листи від користувачів з темою й вкладеннями.
Модератор	Адмін-команди	Доступ до повного списку команд для адміністраторів.
Користувач	Список команд	Доступ до обмеженого списку команд для адміністраторів.
Користувач	Отримання ID	Отримання свого або чужого ID.
Користувач	Отримання дати	Можна отримати поточну дату й час.
Користувач	Надсилання листа	Написати адміну повідомлення з темою.
Користувач	Отримання інформації	Дозволяє отримати відповіді на ЧаПи, правила та контакти.
Користувач	Гра в шашки	Запускати гру в шашки в чаті між користувачами з можливістю здатися або отримати правила гри.

2.3. Побудова логіки взаємодії з користувачем і API

Особливістю всіх ботів в месенджерах є специфічна взаємодія з користувачем: замість традиційного графічного інтерфейсу застосовуються текстові команди, натискання кнопок у повідомленнях та використання інлайн-меню. Саме тому при розробці власного рішення слід врахувати потребу в інтуїтивнішому сценарії взаємодії через зручні меню і повідомлення без необхідності запам'ятовувати складні команди.

Набір команд поділений за ролями: для звичайних користувачів передбачено базовий функціонал, тоді як адміністратори мають доступ до розширеного переліку команд для глибшого модерування та управління каналами та чатами. Поділ користувачів зображений на рисунку 2.1.

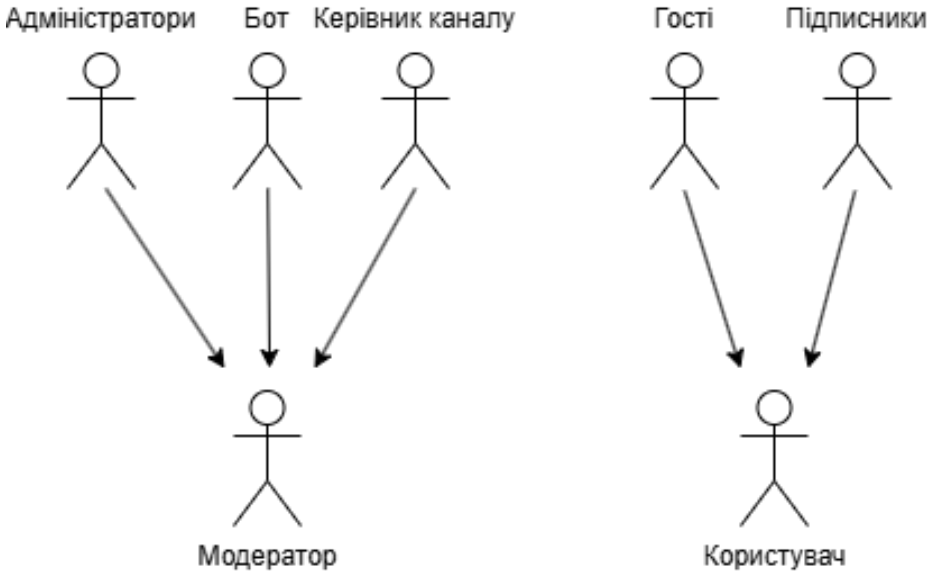


Рисунок 2.1 – Системні суб'єкти

Модератор відповідає за управління каналами та чатами, включає функції додавання, видалення, модерування повідомлень, встановлення правил і прав доступу для користувачів.

Користувачі використовують канали та чати для обміну повідомленнями, перегляду контенту, надсилання файлів і реакцій на повідомлення.

Для кращого розуміння логіки роботи бота для керування каналами та

чатами була розроблена загальна модель роботи бота. Розробка орієнтована на діалоговий інтерфейс у чаті.

Розробленою моделлю є діаграма діяльності взаємодії користувача з ботом, яка зображена на рисунку 2.2. На діаграмі послідовно зображено основні етапи роботи з ботом: початкова базова команда, надсилання запиту на отримання меню доступних команд, вибір однієї з запропонованих опцій, виконання відповідної дії системою та отримання користувачем відповіді або результату виконаної операції [13].

Рисунок 2.2 – Загальна модель взаємодії бота та користувача

Для забезпечення коректної роботи системи також були враховані можливі виключні ситуації, такі як неправильний вибір команди, відсутність прав доступу до певних функцій або технічні збої під час обробки запитів.

Модель передбачає обробку таких випадків через повідомлення про помилку або пропозицію альтернативних дій, що підвищує надійність та зручність використання програмного продукту.

Для розробки бота для керування каналами та чатами необхідно розробити загальний алгоритм роботи команд. Алгоритм зображений на рисунку 2.3.

Рисунок 2.3 – Загальний алгоритм роботи команд

На початку роботи здійснюється старт бота та його підключення до API через наданий токен. Після запуску бота відбувається ініціалізація основних обробників команд, які дозволяють користувачам взаємодіяти із системою.

Далі бот обробляє базову команду, яка виводить перелік доступних команд для користувача залежно від його прав. Користувач обирає одну із запропонованих команд і надсилає її боту.

Після введення команди відбувається обробка отриманого запиту – зокрема, провірка на присутність і правильність команди, провірка на доступ через роль користувача і парсинг параметрів. Система визначає, до якого саме функціоналу відноситься команда, та запускає відповідний модуль обробки.

Після обробки запиту бот виводить результат виконаної команди – надсилає повідомлення користувачу або адміністратору, оновлює повідомлення у чаті або надсилає файл чи документ.

На завершальному етапі обробки запит завершується, і бот продовжує очікувати нові команди або події.

Розробка діаграм здійснювалася з використанням спеціалізованого середовища draw.io. Завдяки використанню цього інструменту вдалося швидко і якісно сформувати візуальні представлення процесів, що сприяло подальшій розробці та тестуванню системи [14].

2.4. Розробка алгоритму обробки команд і ролей

Процес розробки алгоритмів для системи керування каналами та чатами включає побудову логічної структури, що забезпечує обробку користувацьких команд, визначення ролей учасників і управління їхніми правами доступу.

Першим етапом є формалізація структури команд, які може ініціювати користувач або система. Команди поділяються на кілька категорій:

1. Команди користувача охоплюють усі дії, які ініціюються безпосередньо користувачем через інтерфейс бота. Це можуть бути запити на отримання інформації, звернення до адміністратора чи отримання даних. Вони передбачають реакцію бота на команду без складної обробки чи логіки.

2. Напівавтоматизовані команди поєднують ручний тригер із частково автоматизованими сценаріями, наприклад, блокування або розблокування користувача чи вмикання аналізу повідомлень.

3. Повністю автоматизовані дії працюють у фоновому режимі, не потребуючи безпосереднього виклику з боку користувача і автоматично виконують відповідні дії без участі оператора. Вони реагують на події в чаті, як-от автоматичне блокування за погані слова, відлік таймера чи логування повідомлень.

Для кожної команди будується алгоритм обробки, який визначає права користувача в спільноті. Алгоритм зображений на рисунку 2.4.

Рисунок 2.4 – Визначення прав доступу

Розроблені алгоритм дозволяє легко змінювати ролі користувачів, а також можливість виконання заданих команд в залежності від їх ролі.

2.5. Висновки

У другому розділі було сформульовано основні задачі для розробки власного бота, який має поєднати високу функціональність, стабільність, автоматизацію та зручність керування. Було доведено доцільність створення власного рішення, що буде конкурентоспроможним і ефективним для керування спільнотами у месенджерах. На основі отриманої інформації визначено конкретний перелік задач для подальшої розробки системи. Приділено увагу в зручності використання через текстовий інтерфейс. Створенно діаграми роботи бота з користувачами, загальний алгоритм роботи і алгоритм визначення ролі.

3 РОЗРОБКА БОТА ДЛЯ КЕРУВАННЯ КАНАЛАМИ ТА ЧАТАМИ

3.1. Вибір мови програмування, середовища та API

При розробці програмного забезпечення для керування каналами та чатами в месенджері виникла необхідність обрати найбільш оптимальні інструменти, які б забезпечили надійність, функціональність та простоту у подальшій підтримці й розвитку програмного забезпечення. Оскільки розробка орієнтована на взаємодію через чат-інтерфейс, важливо було обрати такі технології, які легко інтегруються з месенджером, дозволяють ефективно працювати з даними та повідомленнями, мають великі бібліотеки для обробки тексту, а також забезпечують просту організацію багатопотокової роботи для обробки великої кількості запитів.

Порівняння мов за критеріями зручності, продуктивності та доступності бібліотек для роботи з месенджерами наведенна в таблиці 3.1 [15].

Таблиця 3.1 – Порівняння мов програмування

Критерії	**Python**	**Go**	**Java**	**C#**	**Rust**
Безпека	±	+	±	±	++
Продуктивність	±	+	+	++	++
Кількість бібліотек	++	±	+	+	±
Готові рішення для API	++	–	+	+	±
Підтримка багатопотоковості	±	+	+	+	+
Налагодження	++	±	+	+	±
Час на розробку	+	±	–	–	–
Загальна оцінка (1–10)	8	7	7	7	6

Python – це високорівнева, інтерпретована мова програмування, яка характеризується простим синтаксисом і широкими можливостями для

розробки серверних та клієнтських додатків [16].

Саме тому було прийнято рішення використати мову програмування Python, яка завдяки своїй простоті, великій кількості бібліотек та активній підтримці спільнот ідеально підходить для швидкої розробки ботів будь-якої складності. Він забезпечує швидку розробку, а також має численні бібліотеки, що значно спрощують реалізацію чат-ботів. Python має вбудовану підтримку асинхронного програмування, що є критичним для обробки великої кількості одночасних запитів.

Середовищем розробки було обрано Visual Studio Code за рахунок його гнучкості, підтримки великої кількості розширень, інтеграції з системами контролю версій та зручного інтерфейсу для роботи з Python [17]. Це дозволило забезпечити швидке прототипування, налагодження та тестування алгоритмів, а також зручно структурувати проєкт.

Месенджер було обрано в якості Telegram для реалізації бота з огляду на його відкритий і добре задокументований API, високу стабільність платформи, гнучку систему управління каналами та чатами, а також широкі можливості для автоматизації без обмеження функціональності з боку розробників сервісу. Крім того, Telegram активно підтримує інтеграцію з ботами на рівні системного інтерфейсу, має вбудовані механізми обробки подій, підтримку асинхронної взаємодії та багаторівневої системи прав, що дозволяє повноцінно реалізувати логіку керування, модерації, призначення ролей і сценаріїв взаємодії.

Telegram включає в себе бібліотеки python-telegram-bot, aiogram, pyrogram та telethon, що дозволяють зручно працювати із його API [18]. У таблиці 3.2 наведено порівняння бібліотеки telegram.ext з іншими популярними бібліотеками для розробки ботів.

Таблиця 3.2 – Порівняння бібліотек API

Критерій	**python-telegram-bot**	**aiogram**	**pyrogram**	**telethon**
Рівень абстракції	++	+	+-	-

Продовження таблиці 3.2.

Критерій	**python-telegram-bot**	**aiogram**	**pyrogram**	**telethon**
Простота використання	++	+	+-	-
Підтримка асинхронності	++	++	+-	++
Підтримка спільноти	++	++	+	++
Гнучка логіка	++	++	+-	++
Підтримка складних сценаріїв	++	++	-	+-
Доступ до низькорівневих методів	+-	+	++	++
Стабільність	++	++	+-	++
Загальна оцінка	9/10	8/10	6/10	7/10

Отже, для реалізації зв’язку з API обрана бібліотека telegram.ext – високорівнева обгортка над python-telegram-bot. Вона надає розширену підтримку роботи з командами, фільтрами, обробниками повідомлень, розгалуженою логікою сценаріїв, а також має зручну модель роботи з асинхронністю, що критично важливо при побудові складних ботів [19].

Тому, для реалізації програмного продукту для управління каналами було використано месенджер Telegram і мову програмування Python у поєднанні з бібліотекою python-telegram-bot і базовими бібліотеками Python (asyncio, re, datetime, os, json), за допомогою яких реалізовано функціонал бота.

Вхідними даними для розробленої системи є:

– повідомлення користувачів у каналі чи чаті;

– команди користувачів;

– документи, фотографії, аудіо- і відеофайли, що надсилаються;

– збереженні текстові файли із забороненими словами, логами або листами чи відповідями.

3.2. Розробка інтерфейсу

При розробці інтерфейсу Telegram-бота було приділено особливу увагу його зрозумілості, зручності та інтерактивності у межах стандартного вікна діалогу Telegram. Інтерфейс реалізується через повідомлення бота, кнопки та меню, що максимально просто і логічно організовані для користувача. Основним середовищем взаємодії виступає чат у Telegram, де бот надсилає текстові повідомлення, запити на підтвердження дій, інформаційні повідомлення та інтерактивні кнопки.

Після першого запуску бота користувач отримує вітальне повідомлення із коротким описом доступних можливостей та кнопкою «Розпочати», яка ініціює відкриття головного меню (рис. 3.1). Це повідомлення служить початковою точкою знайомства з ботом та налаштовує користувача на подальшу роботу.

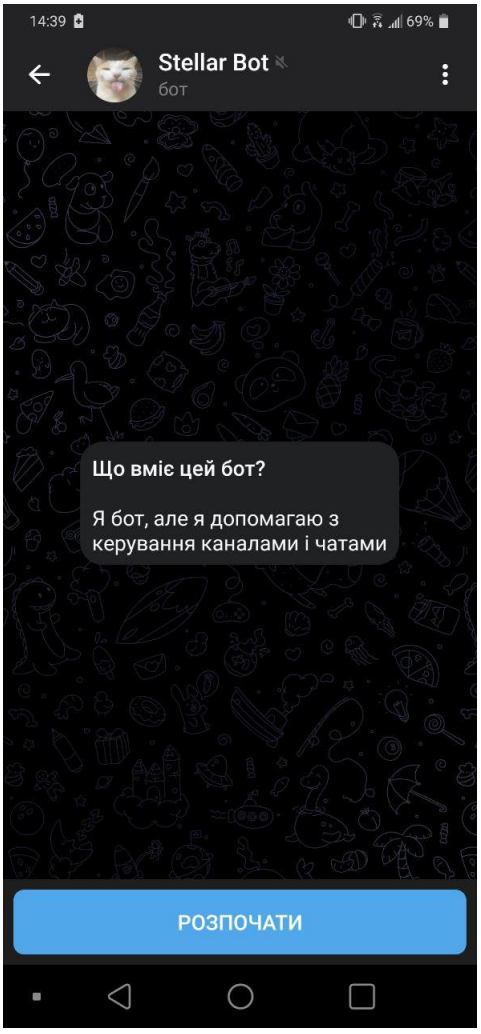


Рисунок 3.1 – Вітальне повідомлення бота з кнопкою запуску

Структура інтерфейсу побудована таким чином, щоб забезпечити мінімальну кількість кроків для виконання основних завдань користувача. Усі функції згруповані в логічні розділи і доступні через прості текстові команди або кнопки з короткими описами. Таке рішення дозволяє користувачу не заплутатися у можливостях бота навіть при першому використанні.

Натиснувши кнопку «Розпочати», бот надсилає головного меню, яке реалізовано у вигляді текстового повідомлення зі списком доступних команд і коротким поясненням кожної з них (рис. 3.2). Команди адаптовані до різних типів користувачів: звичайних користувачів та адміністраторів, що дозволяє гнучко розмежувати доступ до функціоналу.

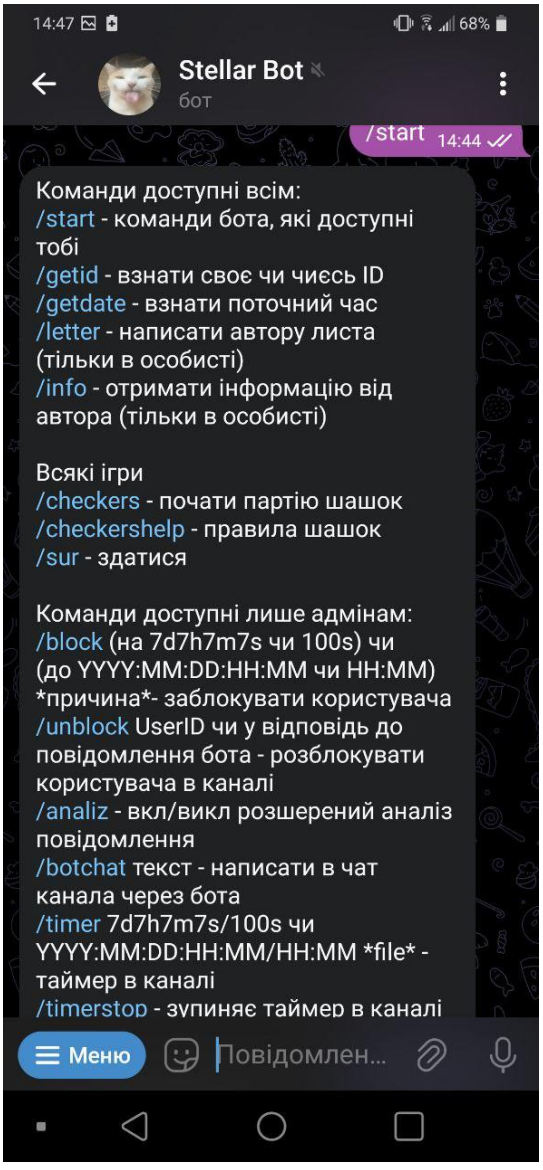


Рисунок 3.2 – Головне меню бота

При виборі функції надсилання листа бот відкриває інтерфейс формату «Лист», де користувач може вибрати тему звернення та скласти повідомлення для власника або модератора каналу (рис. 3.3). Аналогічно працює формат відповідей на часті питання. Завдяки структурі та кнопкам користувачеві легко зрозуміти, які поля йому необхідні.

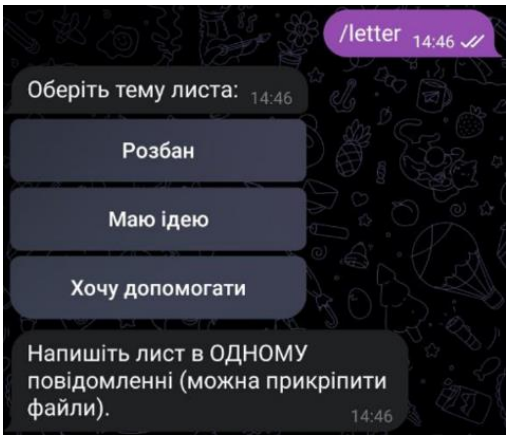


Рисунок 3.3 – Інтерфейст формату «Лист»

У випадках, коли бот виявляє повідомлення із забороненими словами або адміністратор виконує ручне блокування користувача, інтерфейс передбачає автоматичне надсилання сповіщення про виявлене порушення в чат і особисті адміністратору (рис. 3.4). Для розблокування користувача передбачено механізм текстового введення команди у відповідь на повідомлення бота, що спрощує процес адміністрування без потреби пошуку користувача вручну.

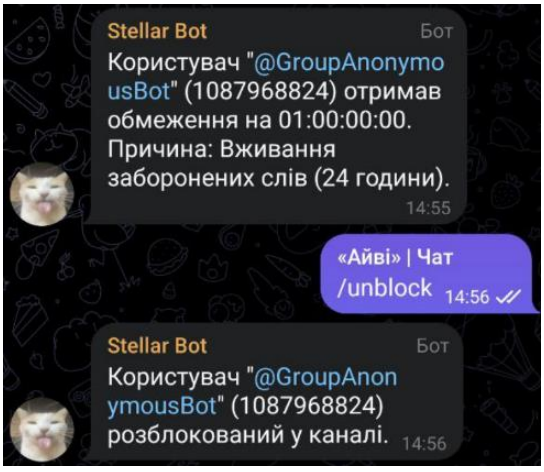


Рисунок 3.4 – Повідомлення про блокування користувача

Також присутня міні-гра в шашки, яка розважає користувачів, щоб підтримувати активність у чаті або змагатися за приз, в якій бот дозволяє розпочати гру прямо в груповому чаті, де двоє людей можуть приєднатися і по черзі робити ходи за допомогою текстових команд (рис. 3.5). Інтерфейс гри реалізовано у вигляді Unicode-дошки, яка оновлюється після кожного ходу, а також містить інформацію про поточного гравця та залишок часу на хід. У разі, якщо гравець хоче здатися, достатньо скористатися командою /sur, після чого бот автоматично завершує гру, повідомляє про переможця в чаті.

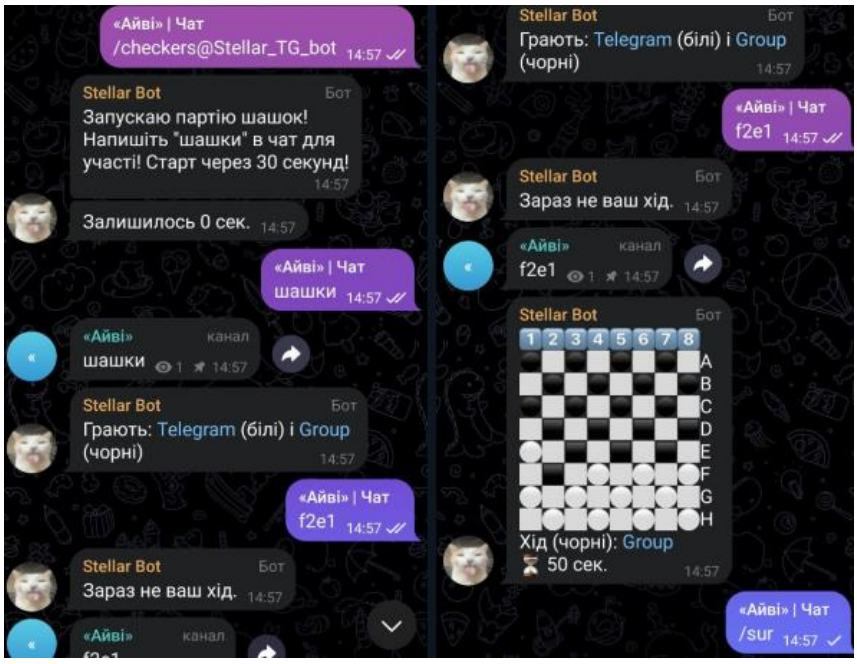


Рисунок 3.5 – Міні-гра шашки через чат

Розробка інтерфейсу Telegram-бота здійснювалась у середовищі Visual Studio Code із використанням мови програмування Python та бібліотеки python-telegram-bot. Усі функціональні елементи розроблені на базі стандартних можливостей Telegram Bot API, що забезпечує стабільну роботу на всіх пристроях незалежно від операційної системи та версії месенджера.

3.3. Розробка основних алгоритмів бота

Основні алгоритми бота охоплюють автоматизацію керування повідомленнями (створення, редагування, видалення), налаштуванням прав,

модерацію контенту, логування, а також інтеракцію з користувачами через меню, кнопки або текстові команди. Для кожного з цих напрямів створюються окремі алгоритми [20].

З метою підвищення надійності й ефективності бота, алгоритми мають модульну структуру. Це дозволяє ізолювати окремі компоненти, наприклад, модерацію, логування або адміністрування, що полегшує розробку, тестування і підтримку.

Для роботи з текстами, обробки вхідних повідомлень, розпізнавання команд, побудови діалогів та автоматизації роботи з повідомленнями застосовується стандартний модуль «re» для регулярних виразів, що дозволяє ефективно проводити фільтрацію текстів, виявляти заборонені слова та здійснювати інші текстові перетворення.

Крім того, використовується модуль «datetime» для обробки часу при роботі з таймерами, що потребують контролю часу. Він забезпечує точне визначення моментів часу, створення відліку до певної дати або години, а також форматування дати для повідомлень бота.

Також важливою частиною програмного рішення є реалізація таймерів, які запускаються та керуються асинхронно за допомогою модуля «asyncio», що дозволяє створювати декілька одночасно працюючих процесів відліку часу без блокування основного потоку бота.

Для доступу до файлової системи та обробки локальних ресурсів використовується модуль «os». Він дозволяє перевіряти наявність файлів, створювати каталоги, відкривати та читати текстові ресурси, необхідні для логіки бота.

Для збереження або передачі структурованих даних у вигляді тексту використовується модуль «json», який дає змогу перетворити об'єкти у формат JSON. Зокрема, це застосовується в режимі аналізу повідомлень, де інформація про оновлення зберігається у вигляді читабельного логу.

У випадках, коли потрібно надсилати файли або великі повідомлення, бот використовує внутрішні можливості API месенджера, забезпечуючи при цьому

обмеження на розмір файлів, перевірку їхньої наявності перед відправкою, а також інформування користувача про статус відправлення файлів чи повідомлень [21].

Метод bad_words_ban перевіряє текст повідомлення на наявність заборонених слів (рис. 3.6). Якщо виявлено заборонене слово з відповідного заданного словника, бот блокує користувача на заданий період, видаляє його повідомлення, повідомляє про блокування у чаті та надсилає це повідомлення адміністраторам з кнопкою для розблокування. Також присутня можливість відключати блокування адміністраторів за погані слова, яка доступна лише власнику бота.

```python
async def bad_words_ban(u: Update, c: CallbackContext):
    if not u.message or not u.message.text:
        return
    if u.effective_user.id in ADMIN_IDS and not Admin_ban_words:
        return
    message_text = re.sub(r"[^\w\s]", "", u.message.text.lower())
    words = set(message_text.split())
    paths = {
        1800: ("C:/D/Project/TGBot/badwords_30min.txt", "Вживання заборонених слів (30 хв)."),
        86400: ("C:/D/Project/TGBot/badwords_24h.txt", "Вживання заборонених слів (24 години).")
    }
    for dur, (file_path, reason) in paths.items():
        try:
            with open(file_path, encoding="utf-8") as f:
                bad_words = set(w.strip().lower() for w in f.read().split(",") if w.strip())
        except FileNotFoundError:
            continue
        if words & bad_words:
            until_time = datetime.datetime.utcnow() + datetime.timedelta(seconds=dur)
            await c.bot.restrict_chat_member(
                u.effective_chat.id, u.effective_user.id,
                permissions=ChatPermissions(can_send_messages=False),
                until_date=until_time
            )
            text = f'Користувач "@{u.effective_user.username}" ({u.effective_user.id}) отримав обмеження на {format_time(dur)}. Причина: {reason}'
            await u.message.reply_text(text)
            await u.message.delete()
            keyboard = InlineKeyboardMarkup([[InlineKeyboardButton("Розблокувати", callback_data=f"unblock_{u.effective_user.id}")]])
            for admin_id in ADMIN_IDS:
                await c.bot.send_message(admin_id, text, reply_markup=keyboard)
            break
```

Рисунок 3.6 – Метод bad_words_ban

Метод block приймає команду повідомлення та контекст, перевіряє, чи користувач є адміністратором і чи команда виконується у відповідь на інше повідомлення і зображений на рисунку 3.7. Якщо так, він парсить тривалість блокування та опціональну причину, після чого блокує користувача у чаті на вказаний період, видаляє його повідомлення, повідомляє про блокування у чаті та надсилає це повідомлення адміністраторам з кнопкою для розблокування.

```python
async def block(u: Update, c: CallbackContext):
    if u.effective_user.id in ADMIN_IDS and u.message.reply_to_message and c.args:
        d = parse_time(c.args[0])
        reason = " ".join(c.args[1:]) if len(c.args) > 1 else ""
        if d:
            user_to_restrict = u.message.reply_to_message.from_user
            await c.bot.restrict_chat_member(
                u.effective_chat.id, user_to_restrict.id,
                permissions=ChatPermissions(can_send_messages=False),
                until_date=asyncio.get_event_loop().time() + d
            )
            await u.message.delete()
            text = f'Користувач "@{user_to_restrict.username}" ({user_to_restrict.id}) отримав обмеження на {format_time(d)}.'
            if reason:
                text += f' Причина: {reason}'
            await u.message.reply_to_message.reply_text(text)
            keyboard = InlineKeyboardMarkup([[InlineKeyboardButton("Розблокувати", callback_data=f"unblock_{user_to_restrict.id}")]])
            for admin_id in ADMIN_IDS:
                await c.bot.send_message(admin_id, text, reply_markup=keyboard)
```

Рисунок 3.7 – Метод block

Метод unblock дозволяє адміністратору розблокувати вибраного користувача або за його ID, або у відповідь на повідомлення бота в якому міститься його ID (рис. 3.8). Розблоковує через API Telegram і надсилає повідомлення про розблокування.

```python
async def unblock(u: Update, c: CallbackContext):
    if u.effective_user.id not in ADMIN_IDS:
        return

    user = None
    if u.message.reply_to_message:
        match = re.search(r'\((\d+)\)', u.message.reply_to_message.text)
        if match:
            user_id = int(match.group(1))
            user = await c.bot.get_chat(user_id)
    if not user and c.args:
        try:
            user = await c.bot.get_chat(int(c.args[0]))
        except ValueError:
            return

    if user:
        await c.bot.restrict_chat_member(ChatID, user.id, permissions=ChatPermissions(can_send_messages=True))
        unblock_message = f'Користувач "@{user.username}" ({user.id}) розблокований у каналі.'

        if u.effective_chat.id == ChatID:
            await u.message.delete()
            await c.bot.send_message(ChatID, unblock_message)
        else:
            await u.message.reply_text(unblock_message)
            await c.bot.send_message(ChatID, unblock_message)
```

Рисунок 3.8 – Метод unblock

Метод parse_time бере рядок із зазначенням тривалості або точного часу і перетворює її у секунду і зображений на рисунку 3.9. Підтримує формати d/h/m/s, HH:MM, YYYY:MM:DD:HH:MM.

```python
def parse_time(t):
    if re.fullmatch(r"\d{4}:\d{2}:\d{2}:\d{2}:\d{2}", t):
        now = datetime.datetime.now()
        target_time = datetime.datetime.strptime(t, "%Y:%m:%d:%H:%M")
        return int((target_time - now).total_seconds())
    if re.fullmatch(r"\d{1,2}:\d{2}", t):
        now = datetime.datetime.now()
        target_time = datetime.datetime.strptime(t, "%H:%M").replace(year=now.year, month=now.month, day=now.day)
        if target_time < now:
            target_time += datetime.timedelta(days=1)
        return int((target_time - now).total_seconds())
    m = re.fullmatch(r"(?:(\d+)d)?(?:(\d+)h)?(?:(\d+)m)?(?:(\d+)s)?", t)
    if not m:
        return None
    d, h, m, s = (int(m.group(i) or 0) for i in range(1, 5))
    return d * 86400 + h * 3600 + m * 60 + s
```

Рисунок 3.9 – Метод parse_time

Метод format_time бере тривалість у секундах та форматує її у вигляді ДД:ГГ:ХХ:СС (рис. 3.10).

```python
def format_time(s):
    d, s = divmod(s, 86400)
    h, s = divmod(s, 3600)
    m, s = divmod(s, 60)
    return f"{d:02}:{h:02}:{m:02}:{s:02}"
```

Рисунок 3.10 – Метод format_time

Метод countdown запускає таймер зворотного відліку, кожну секунду оновлюючи повідомлення у чаті (рис. 3.11). По завершенню надсилає файл, якщо він був заданий. Метод створений для відкладеного надсилання файлу.

```python
active_timers = {}
async def countdown(c, chat_id, d, file_id=None):
    m = await c.bot.send_message(chat_id, format_time(d))
    active_timers[chat_id] = m.message_id
    while d > 0:
        await asyncio.sleep(1)
        if chat_id not in active_timers: return
        d -= 1
        try: await m.edit_text(format_time(d))
        except: return
    if file_id: await c.bot.send_document(chat_id, file_id)
    await m.delete()
    active_timers.pop(chat_id, None)
```

Рисунок 3.11 – Метод countdown

Метод button_unblock_handler обробляє кнопку «Розблокувати», що приходить разом із повідомленням про бан (рис. 3.12). Бот розблоковує користувача та оновлює повідомлення.

```
212 async def button_unblock_handler(u: Update, c: CallbackContext):
213     query = u.callback_query
214     if not query:
215         return
216     user_id = int(query.data.split("_")[1])
217     await c.bot.restrict_chat_member(ChatID, user_id, permissions=ChatPermissions(can_send_messages=True))
218     user = await c.bot.get_chat(user_id)
219     text = f'Користувач "@{user.username}" ({user.id}) розблокований у каналі.'
220     await query.message.edit_text(text)
221     await c.bot.send_message(ChatID, text)
```

Рисунок 3.12 – Метод button_unblock_handler

Метод timer_handler обробляє команду /timer і /timerstop від адміністратора (рис. 3.13). Якщо очікується файл – зберігає його ID та запускає таймер. Інакше запускає таймер одразу або зупиняє активний.

```
148 waiting_for_file = {}
149 async def timer_handler(u: Update, c: CallbackContext):
150     if u.effective_user.id not in ADMIN_IDS:
151         return
152     global waiting_for_file
153     args = c.args
154     if u.message.document and u.effective_user.id in waiting_for_file:
155         d, chat_id = waiting_for_file.pop(u.effective_user.id)
156         file_id = u.message.document.file_id
157         asyncio.create_task(countdown(c, ChannelID, d, file_id))
158         return
159     if not args:
160         return
161     if u.message.text.startswith("/timerstop"):
162         if (message_id := active_timers.pop(ChannelID, None)):
163             try:
164                 await c.bot.delete_message(ChannelID, message_id)
165             except:
166                 pass
167         return
168     if u.message.text.startswith("/timer"):
169         if "file" in args:
170             args.remove("file")
171             d = parse_time(" ".join(args))
172             if d:
173                 waiting_for_file[u.effective_user.id] = (d, u.effective_chat.id)
174                 await u.message.reply_text("Надішліть файл.")
175         else:
176             d = parse_time(" ".join(args))
177             if d:
178                 asyncio.create_task(countdown(c, ChannelID, d))
```

Рисунок 3.13 – Метод timer_handler

Метод botchat дозволяє адміну написати повідомлення від імені бота в загальний чат і зображений на рисунку 3.14. Метод дозволяє писати анонімно.

```python
async def botchat(u: Update, c: CallbackContext):
    if u.effective_chat.type == "private" and c.args:
        text = " ".join(c.args)
        await c.bot.send_message(ChatID, text)
```

Рисунок 3.14 – Метод botchat

Метод letter_handler дозволяє користувачам в особистому чаті бота надіслати лист власнику (рис. 3.15). Користувач обирає тему, після чого надсилає текст або файл у чат, який бот перенаправляє адміну, як лист. Адміну приходить лист з усіма данними користувача і його текстом.

```python
async def letter_handler(u: Update, c: CallbackContext):
    if u.effective_chat.type != "private":
        return
    if u.message and u.message.text and u.message.text.startswith("/letter"):
        topics = [
            ["Розбан", "🔓 Розбан"],
            ["Маю ідею", "💡 Маю ідею"],
            ["Хочу допомогати", "🤝 Хочу допомогати"],
        ]
        keyboard = [[InlineKeyboardButton(name, callback_data=f"letter_{name.lower()}")] for name, _ in topics]
        c.user_data["letter_topics"] = {name.lower(): icon for name, icon in topics}
        await u.message.reply_text("Оберіть тему листа:", reply_markup=InlineKeyboardMarkup(keyboard))
        return
    if u.callback_query and u.callback_query.data.startswith("letter_"):
        topic_key = u.callback_query.data.split("letter_")[1]
        topic = c.user_data.get("letter_topics", {}).get(topic_key)
        if topic:
            c.user_data.update({"letter_topic": topic, "awaiting_letter": True})
            await u.callback_query.message.reply_text("Напишіть лист в ОДНОМУ повідомленні (можна прикріпити файли).")
        return
    if c.user_data.pop("awaiting_letter", False):
        topic = c.user_data.get("letter_topic", "Без теми")
        user, text = u.effective_user, u.message.caption or u.message.text or "Без тексту"
        message = f"📩 НОВИЙ ЛИСТ\nТема: {topic}\nВід: @{user.username} ({user.id})\n\n{text}"
        file_attr = next((attr for attr in ["document", "photo", "video", "audio", "voice"] if getattr(u.message, attr, None)), None)
        if file_attr:
            file_id = u.message.photo[-1].file_id if file_attr == "photo" else getattr(u.message, file_attr).file_id
            await getattr(c.bot, f"send_{file_attr}")(MyID, file_id, caption=message)
        else:
            await c.bot.send_message(MyID, message)
        await u.message.reply_text("✅ Ваш лист відправлено!")
```

Рисунок 3.15 – Метод letter_handler

Метод start надсилає користувачу список доступних команд в залежності його прав доступу (рис. 3.16). Якщо користувач є адміністратором, то бот додає адмінські команди.

```python
async def start(u, c):
    cmds = "\n".join(f"/{cmd}" for cmd in USER_COMMANDS)
    if u.effective_user.id in ADMIN_IDS:
        cmds += f"\n\nКоманди доступні лише адмінам:\n" + "\n".join(f"/{cmd}" for cmd in ADMIN_COMMANDS)
    await u.message.reply_text(f"Команди доступні всім:\n{cmds}")
```

Рисунок 3.16 – Метод start

Метод info_handler надає користувачеві документи по обраній темі з меню (рис. 3.17). Якщо обрано пункт, бот шукає відповідний файл на комп'ютері і надсилає його в чат користувачеві. Працює лише через особисті повідомлення.

```
async def info_handler(u: Update, c: CallbackContext):
    if u.effective_chat.type == "private":
        if u.message and u.message.text and u.message.text.startswith("/info"):
            buttons_info = [
                ["ЧаПи", "C:/D/Project/TGBot/FAQ.txt"],
                ["Правила", "C:/D/Project/TGBot/Rules.txt"],
                ["Контакти", "C:/D/Project/TGBot/Contacts.txt"],
            ]
            buttons = [
                [InlineKeyboardButton(name, callback_data=f"info_{name.lower()}")]
                for name, _ in buttons_info
            ]
            keyboard = InlineKeyboardMarkup(buttons)
            c.user_data["buttons_info"] = {name.lower(): path for name, path in buttons_info}
            await u.message.reply_text("Виберіть потрібну тему:", reply_markup=keyboard)
        elif u.callback_query and u.callback_query.data.startswith("info_"):
            button_name = u.callback_query.data.split("info_")[1]
            file_path = c.user_data.get("buttons_info", {}).get(button_name)
            if file_path and os.path.exists(file_path):
                await u.callback_query.message.reply_document(document=open(file_path, "rb"))
            else:
                await u.callback_query.message.reply_text("Файл не знайдено.")
```

Рисунок 3.17 – Метод info_handler

Метод getid надсилає користувачу його Telegram ID або ID іншої особи, якщо є відповідю на повідомлення (рис. 3.18).

```
async def getid(u: Update, c: CallbackContext):
    if u.message.reply_to_message:
        target_user = u.message.reply_to_message.from_user
        await u.message.reply_text(f"ID цього користувача: {target_user.id}")
    else:
        await u.message.reply_text(f"Ваш ID: {u.effective_user.id}")
```

Рисунок 3.18 – Метод getid

Метод getdate відправляє користувачу поточну дату і час у форматі РРРР:ММ:ДД:ГГ:ХХ і зображений на рисунку 3.19. Метод потрібен для отримання зручної дати, яку потрібно вписувати в команду з баном.

```python
async def getdate(u: Update, c: CallbackContext):
    now = datetime.datetime.now().strftime("%Y:%m:%d:%H:%M")
    await u.message.reply_text(f"Сьогоднішня дата: {now}")
```

Рисунок 3.19 – Метод getdate

Клас CheckersGameManager і методи checkers, handle_checkers_message, surrender_checkers і checkershelp створені для гри в шашки (рис. 3.20).

```python
active_checkers_games = {}
checkers_manager = CheckersGameManager()
async def checkers(update: Update, context: CallbackContext):
    chat_id = update.effective_chat.id
    game = active_checkers_games.get(chat_id)
    if not game:
        game = CheckersGameManager()
        active_checkers_games[chat_id] = game
    await game.initiate_game(update, context)

async def handle_checkers_message(update: Update, context: CallbackContext):
    chat_id = update.effective_chat.id
    game = active_checkers_games.get(chat_id)
    if game:
        await game.handle_message(update, context)

async def surrender_checkers(update: Update, context: CallbackContext):
    chat_id = update.effective_chat.id
    game = active_checkers_games.get(chat_id)
    if game:
        await game.surrender(update, context)

async def checkershelp(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = (
```

Рисунок 3.20 – Клас і методи для шашок

Метод analiz зберігає вхідні дані повідомлень у локальний файл та за потреби пересилає їх адмінам для аналізу (рис. 3.21). Вмикається та вимикається командою /analiz.

```python
analiz_mode = False
async def analiz(update: Update, context: ContextTypes.DEFAULT_TYPE):
    global analiz_mode
    logs_dir = "logs"
    os.makedirs(logs_dir, exist_ok=True)
    raw_data = update.to_dict()
    json_text = json.dumps(raw_data, indent=2, ensure_ascii=False)
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S_%f")
    filename = os.path.join(logs_dir, f"{timestamp}.txt")
    with open(filename, "w", encoding="utf-8") as f:
        f.write(json_text)
    if update.message and update.message.text == '/analiz' and update.effective_user.id in ADMIN_IDS:
        analiz_mode = not analiz_mode
        status = 'увімкнено' if analiz_mode else 'вимкнено'
        await update.message.reply_text(f'Аналіз режим {status}.')
    elif analiz_mode and update.effective_user.id in ADMIN_IDS:
        if len(json_text) > 4000:
            json_text = json_text[:4000] + '\n... (обрізано)'
        await context.bot.send_message(chat_id=ADMIN_IDS[0], text=f'<pre>{json_text}</pre>', parse_mode='HTML')
```

Рисунок 3.21 – Метод analiz

Отже, розроблені алгоритми забезпечують ефективне, безпечне та гнучке керування каналами і чатами через Telegram, що дає змогу автоматизувати рутинні процеси, зменшити навантаження на адміністраторів і підвищити якість взаємодії з учасниками спільнот.

3.4. Інтеграція алгоритмів з Telegram API

У процесі розробки програмного забезпечення для керування каналами та чатами важливим етапом є інтеграція реалізованих алгоритмів з API, який забезпечує доступ до функцій платформи. Ця інтеграція дозволяє забезпечити двосторонню взаємодію між користувачами та системою, з можливістю обробки подій у реальному часі, обміну даними, керування доступом і виконання відповідних дій у відповідь на події, що надходять від сервера.

Інтеграція алгоритмів в Telegram API в коді реалізована через бібліотеку python-telegram-bot, яка здійснює роботу з HTTP-запитами до Telegram Bot API. Ідентифікація бота відбувається за допомогою токена (TOKEN), який видається Telegram після створення бота в BotFather – він дозволяє авторизуватися і виконувати дії від імені бота. Всі взаємодії (надсилання повідомлень, блокування, обмеження, реакція на кнопки тощо) здійснюються через методи об'єкта бота, який доступний через CallbackContext (рис. 3.22).

```
325  TOKEN = "secret"
326  app = Application.builder().token(TOKEN).build()
327  app.add_handler(CommandHandler("start", start))
328  app.add_handler(CommandHandler("timer", timer_handler))
329  app.add_handler(CommandHandler("timerstop", timer_handler))
330  app.add_handler(CommandHandler("block", block))
331  app.add_handler(CommandHandler("getid", getid))
332  app.add_handler(CommandHandler("getdate", getdate))
333  app.add_handler(CommandHandler("unblock", unblock))
334  app.add_handler(CommandHandler("info", info_handler))
335  app.add_handler(CommandHandler("botchat", botchat))
336  app.add_handler(CommandHandler("checkers", checkers))
337  app.add_handler(CommandHandler("checkershelp", checkershelp))
338  app.add_handler(CommandHandler("sur", surrender_checkers))
339  app.add_handler(CallbackQueryHandler(button_unblock_handler, pattern=r"unblock_\d+"))
340  app.add_handler(CallbackQueryHandler(info_handler, pattern=r"info_.*"))
341  app.add_handler(CallbackQueryHandler(letter_handler, pattern=r"letter_.*"))
342  app.add_handler(MessageHandler(filters.Document.ALL, timer_handler), group=0)
343  app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, bad_words_ban), group=1)
344  app.add_handler(MessageHandler(filters.ALL, letter_handler), group=2)
345  app.add_handler(MessageHandler(filters.TEXT & (~filters.COMMAND), handle_checkers_message), group=3)
346  app.add_handler(MessageHandler(filters.ALL, analiz), group=4)
347
348  if __name__ == "__main__":
349      app.run_polling()
```

Рисунок 3.22 – Інтеграція команд через API

Команди додаються через CommandHandler, який пов'язує текстову команду (наприклад, «/start», «/block») з відповідною асинхронною функцією. Обробники (add_handler(...)) прописуються в додатку, який відповідає за отримання оновлень від Telegram через run_polling. Зв'язок між діями користувача та функціями бота реалізовано через об'єкти Update та Context, які містять інформацію про повідомлення, чат, користувача, аргументи команди тощо. Таким чином, весь цикл роботи проходить через систему обробників.

Отже, інтеграція алгоритмів із зовнішнім API здійснювалася через чітке розмежування функцій контролю, обробки і відповіді, що дозволило досягти стабільної роботи, високої швидкості реакції та коректної взаємодії з користувачами в рамках заданих сценаріїв.

3.5. Висновки

У третьому розділі було обґрунтовано вибір мови програмування та бібліотек, розроблено і детально описано розробку основних алгоритмів системи, які забезпечують обробку повідомлень, модерацію користувачів, адміністрування каналів, керування таймерами та організацію зворотного зв'язку. Також представлено принципи побудови інтерфейсу бота, орієнтованого на зручність користувача через використання інтерактивних кнопок та меню в чаті. Результатом є створення функціональної, масштабованої та легко підтримуваної програмної системи, що задовольняє вимоги автоматизації процесів управління спільнотами в месенджері Telegram.

4 ВЕРИФІКАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

4.1. Опис методики тестування

Методика тестування програмного забезпечення для керування каналами та чатами базується на послідовній перевірці функціональності бота в умовах, максимально наближених до реального використання. Оскільки система інтегрується з Telegram API, тестування проводиться у середовищі з тестовими й реальними каналами, групами та користувачами різного доступу.

Основним підходом до тестування є інтерактивне тестування, тобто запуск бота на сервері та поступова перевірка всіх ключових функцій через звичайний інтерфейс Telegram. Це дозволяє імітувати поведінку користувача, який надсилає команди, натискає кнопки, взаємодіє з меню тощо. Кожна функція бота перевіряється у кількох сценаріях, щоб упевнитися в коректності її реалізації та стійкості до некоректних дій.

Особлива увага приділяється перевірці обробки команд, логіки ролей та реакції на події у чатах. Для цього створюються окремі групи користувачів із заздалегідь заданими правами доступу, після чого перевіряється, чи дотримується обмеження на виконання дій: видалення повідомлень, зміна налаштувань, додавання учасників тощо.

Методика також включає тестування навантаження – симуляцію великої кількості запитів до бота за короткий проміжок часу для перевірки його стабільності. Таке тестування проводиться в умовах високої активності в чатах і дозволяє виявити проблеми в продуктивності або обробці [22].

Отже, запропонована методика поєднує тестування в реальному месенджері і перевірку на стресостійкість, що дозволяє повноцінно оцінити функціональність, надійність та зручність використання розробленого бота.

4.2. Проведення функціонального тестування

Тестування системи керування каналами та чатами у месенджерах є етапом розробки, що дозволяє виявити помилки, перевірити відповідність

вимогам і забезпечити стабільність роботи системи у реальних умовах експлуатації. Основною метою тестування є гарантування надійності взаємодії з чатами та каналами, правильності обробки команд користувачів, повідомлень та адміністративних дій.

Перед початком тестування була підготовлена тестова інфраструктура, що включала створення бота, канала і чата у месенджері Telegram. Тестування виконувалося на комп'ютері із встановленою операційною системою Windows 10 та на мобільному пристрої із Android 10 для перевірки коректної роботи системи у різних середовищах.

На початковому етапі тестування було приділено увагу перевірці найважливішої функціональності – обмеження прав доступу до команд користувачів залежно від їхніх ролей у чаті. Система чітко розділяє користувачів на дві категорії: звичайних користувачів і адміністраторів. Якщо звичайний користувач намагається отримати меню з командами, бот надсилає повідомлення з набором звичаних команд (рис. 4.1).

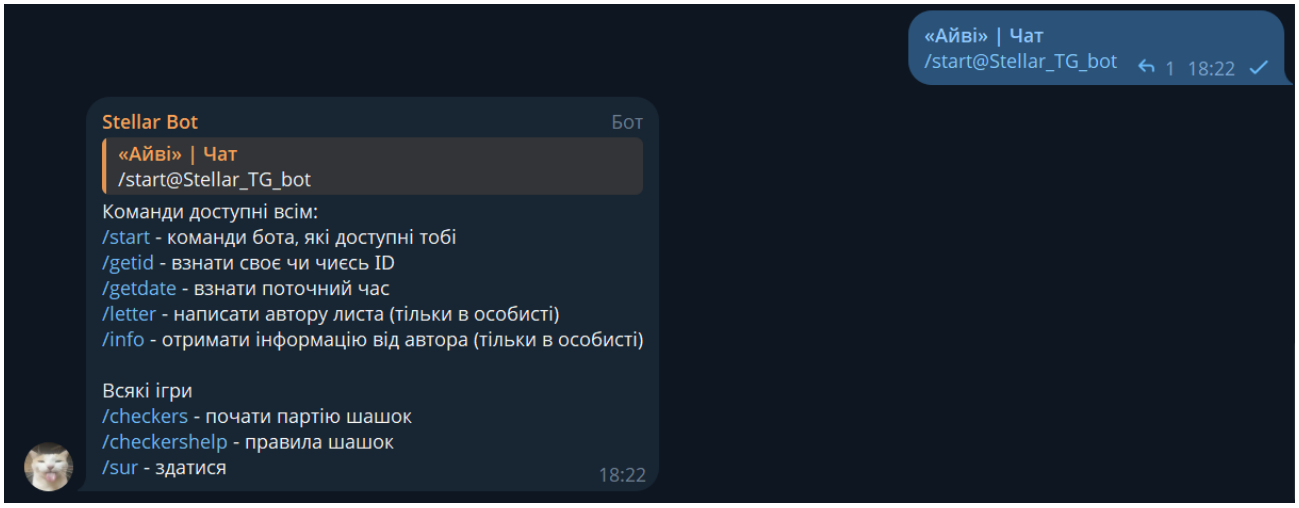


Рисунок 4.1 – Повідомлення без прав

Натомість для адміністратора виводиться розширене повідомлення із додатковими інструкціями або доступними варіантами дій і це зображено на рисунку 4.2. Це тестування дозволило переконатися, що права доступу налаштовані правильно, а механізм перевірки ролей працює стабільно.

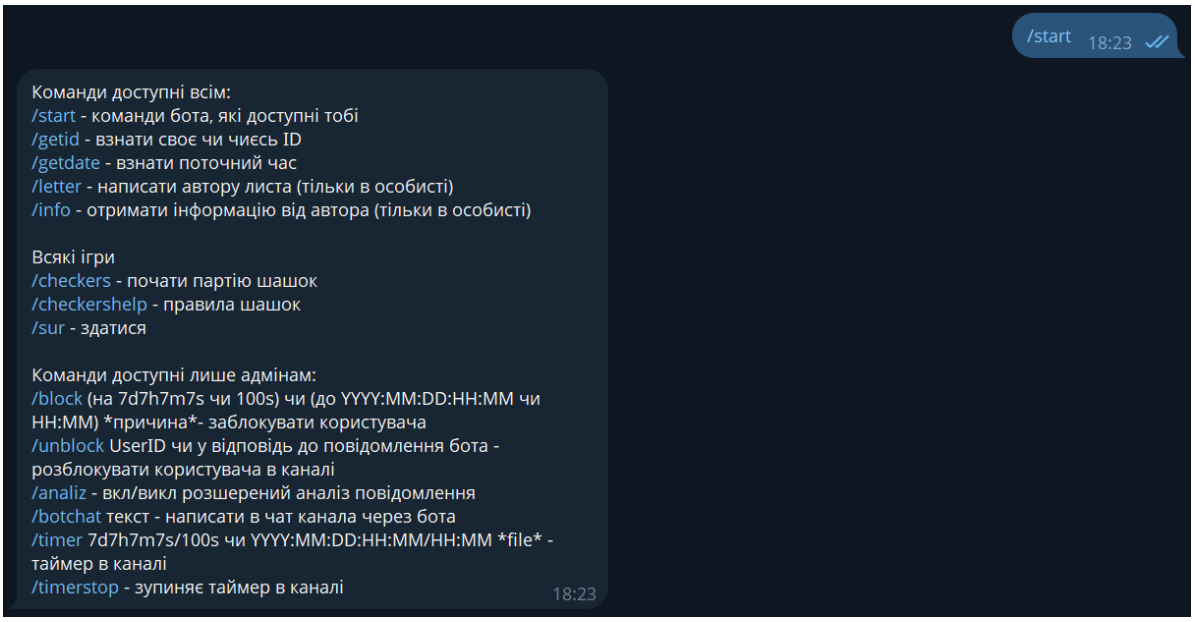


Рисунок 4.2 – Повідомлення з правами

Після успішної перевірки прав доступу було розпочато тестування правильності обробки адміністративних команд. Для прикладу була обрана команда /block, яка має кілька варіантів використання. Спочатку вводилася команда без додаткових параметрів, наприклад «/block», у відповідь на повідомлення користувача. Бот правильно розпізнав недостатність інформації та не виконав дію (рис. 4.3).

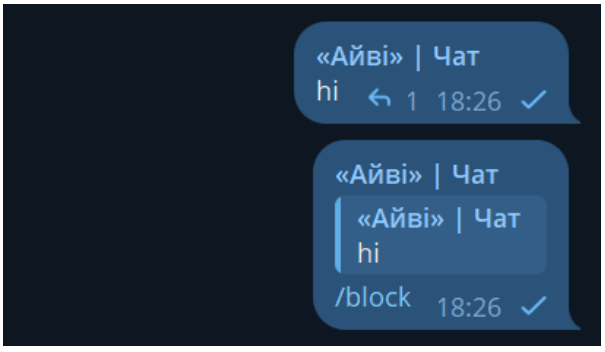


Рисунок 4.3 – Команда без додаткових параметрів

Далі було протестовано команду «/block 1d», яка означає блокування користувача на один день. Бот успішно заблокував користувача, видалив його повідомлення з командою на блокування і залишив службове повідомлення про блокування. Результат зображений на рисунку 4.4.

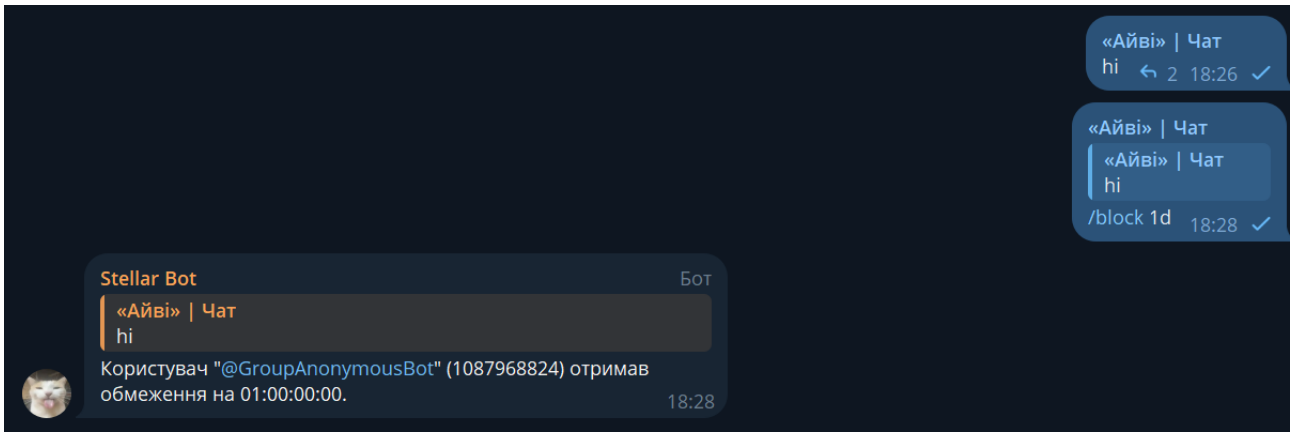


Рисунок 4.4 – Команда з додатковим параметром

Бот автоматично надсилає детальну інформацію про факт блокування: ім’я та ID користувача, тривалість блокування та вказану причину (за наявності). Якщо бот розпізнає заборонене слово в повідомленні користувача, яке знаходиться в двох словниках, то він заблокує автоматично користувача.

Наступний тест стосувався команди «/block 1d Спам», де додатково вказувалась причина блокування. Усі дані були правильно оброблені та зафіксовані у повідомленні (рис. 4.5).

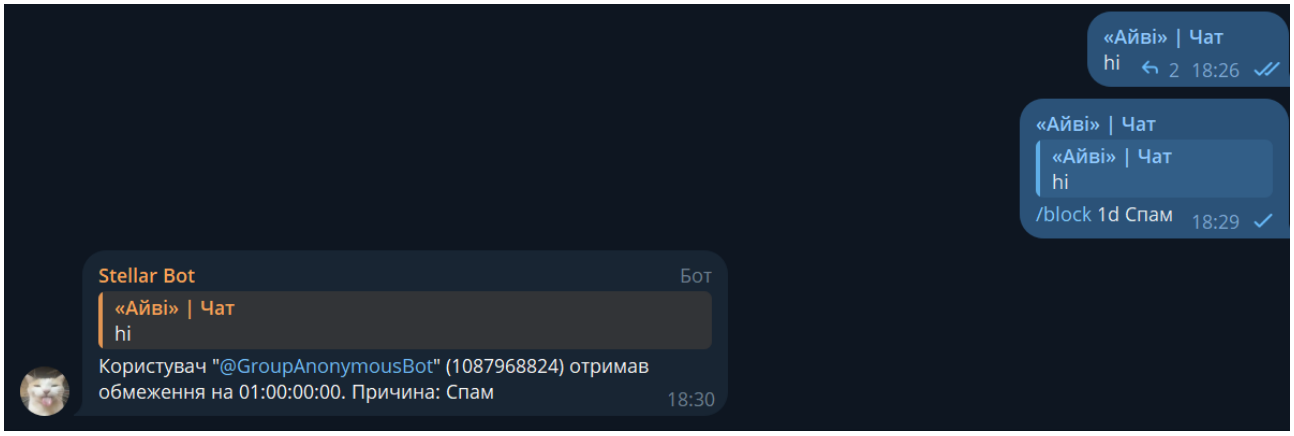


Рисунок 4.5 – Команда з додатковими параметроми

В обох випадках, бот присилає повідомлення про блокування як в чат, де здійснено блокування, так і в особисті адміністратору. В особистих присутня інтерактивна кнопка «Розблокувати», після натискання якої можна швидко розблокувати користувача. Результат зображений на рисунку 4.6.

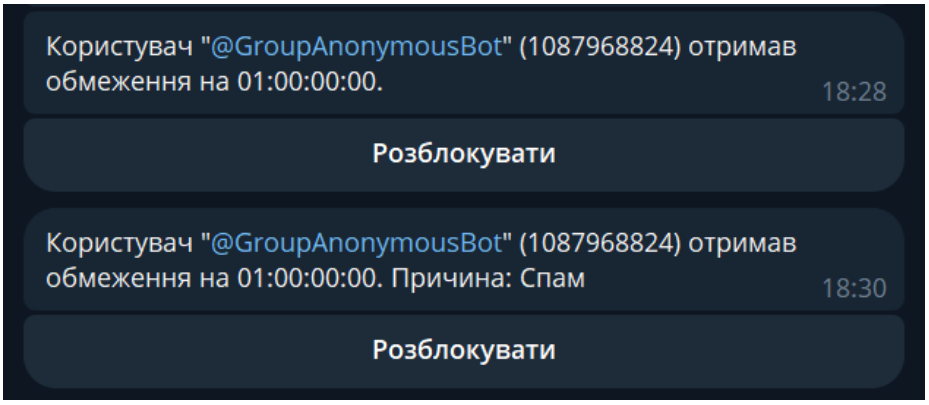


Рисунок 4.6 – Сповіщення в особистих

Наступним етапом стало тестування розблокування користувачів. Якщо адміністратор бажає достроково розблокувати користувача, він має можливість скористатися командою /unblock (рис. 4.7). Це можна зробити як у відповідь на службове повідомлення про блокування в групі, так і в особистому чаті з ботом, використовуючи формат «/unblock user_id». В обох випадках бот успішно знімає обмеження і надсилає підтвердження у групу та в особисті повідомлення.

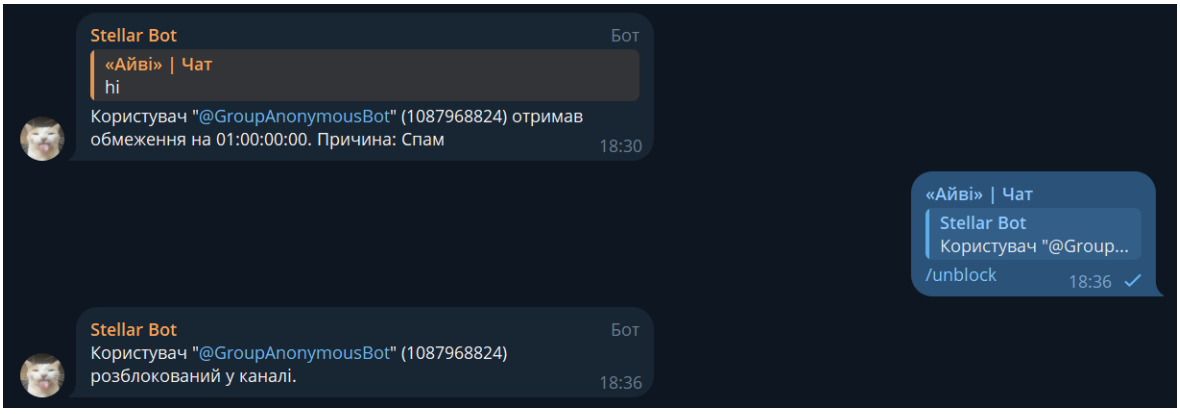


Рисунок 4.7 – Розблокування користувача

Було також протестовано таймер за допомогою команди «/timer» та «/timerstop», який зображений на рисунку 4.8. Після отримання цих команд бот або розпочинав таймер або зупиняв його в каналі. Таймер було запущено на 10 секунд з прикріпленим файлом.

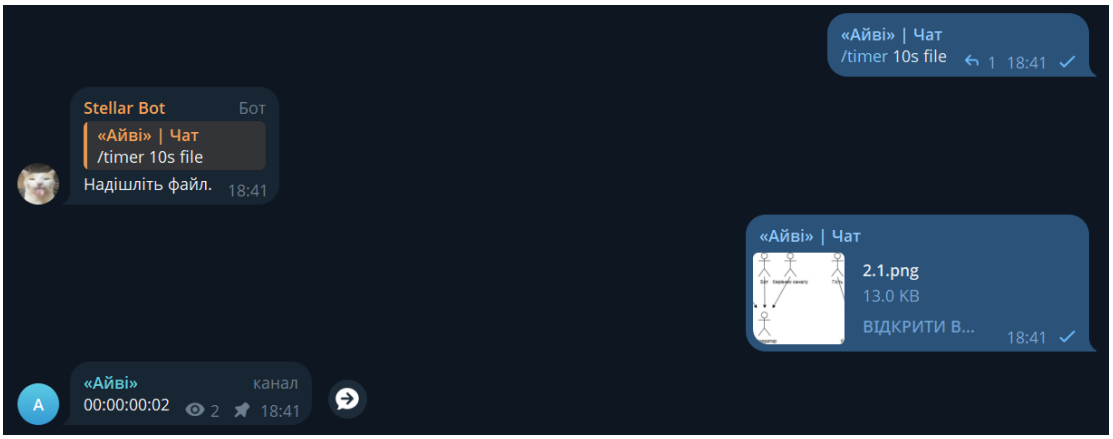


Рисунок 4.8 – Реалізація таймера

Окремо було протестовано функціональність обробки запитів користувачів через алгоритм листування (рис. 4.9). Якщо заблокований користувач бажає оскаржити своє блокування або звернутися до адміністратора з іншим питанням, він може скористатися командою «/letter». Після введення команди бот запропонує вибрати тему звернення та оформити лист. Після заповнення всіх полів бот повідомить користувача про успішну відправку листа, а адміністратор отримає звернення із зазначенням теми та автора. Оскільки лист був надісланий самому собі, його було отримано в тому ж чаті.

Рисунок 4.9 – Лист від користувача

Також була перевірка роботи інформаційного алгоритма бота, який аналогічний листуванню. При введенні команди «/info» користувач отримує

список доступних тем для ознайомлення (рис. 4.10). Після вибору теми бот надсилає відповідний документ або повідомляє, якщо файл відсутній. Це дозволяє ефективно реалізувати функціонал FAQ всередині каналу.

Рисунок 4.10 – Команда «/info»

Додатково тестувалася реакція бота на спробу використання адміністративних команд звичайними користувачами (рис. 4.11). Якщо користувач без відповідних прав вводив команду, бот правильно ігнорував запит без надсилання зайвих повідомлень. Це дозволяє уникнути навантаження на чат та запобігти неправильному інформуванню учасників.

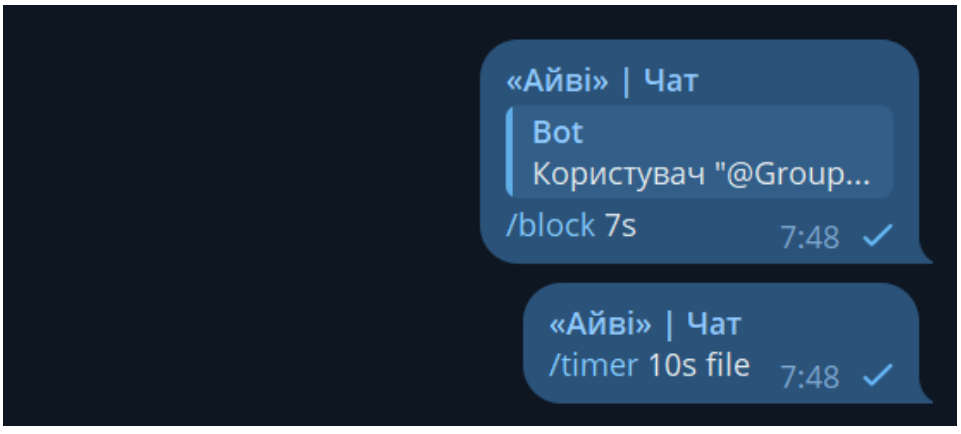


Рисунок 4.11 – Користувач без прав

Протестовано користувацькі команди «/getid» і «/getdate», які дозволяються взнати час, своє чи чиєсь ID, які зображені на рисунку 4.12.

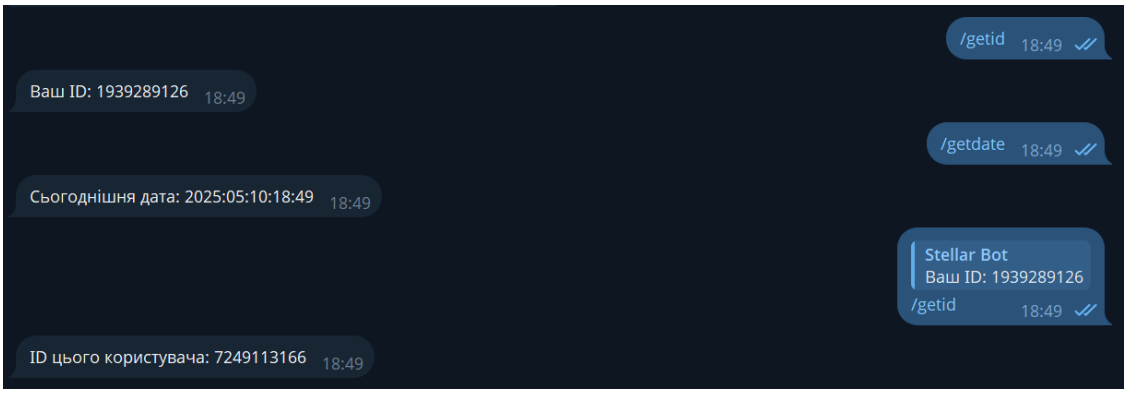


Рисунок 4.12 – Команди «/getid» і «/getdate»

Далі протестовано команду «/analiz» (рис. 4.13). Після активації цього режиму, бот бере початковий JSON-файл, який надсилає сам месенджер і перетворює його повністю на читабельний текст, який містить всю інформацію повідомлення і надсилає в особисті адміністратору. Також метод завжди зберігає цей текст в окремих лог-файлах на комп’ютері.

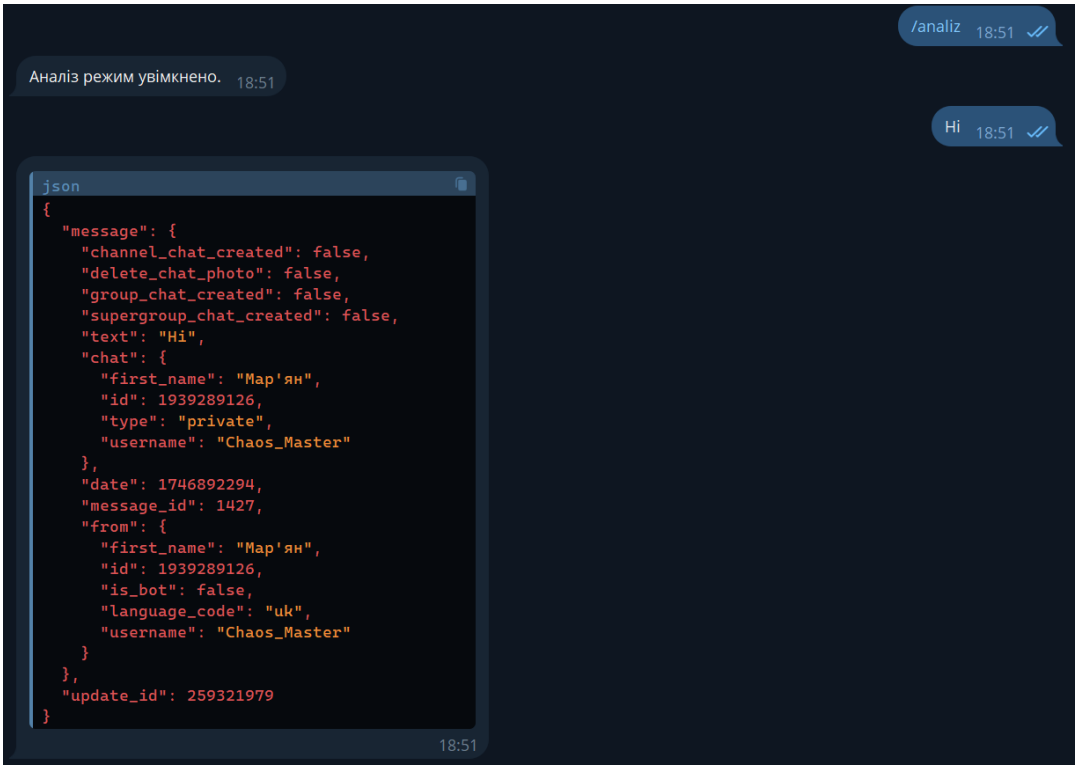


Рисунок 4.13 – Команда «/analiz»

Було протестовано партію з шашок, яку зображено на рисунку 4.14. Спочатку йде набір протягом часу, потім гра, яка закінчилася програшом.

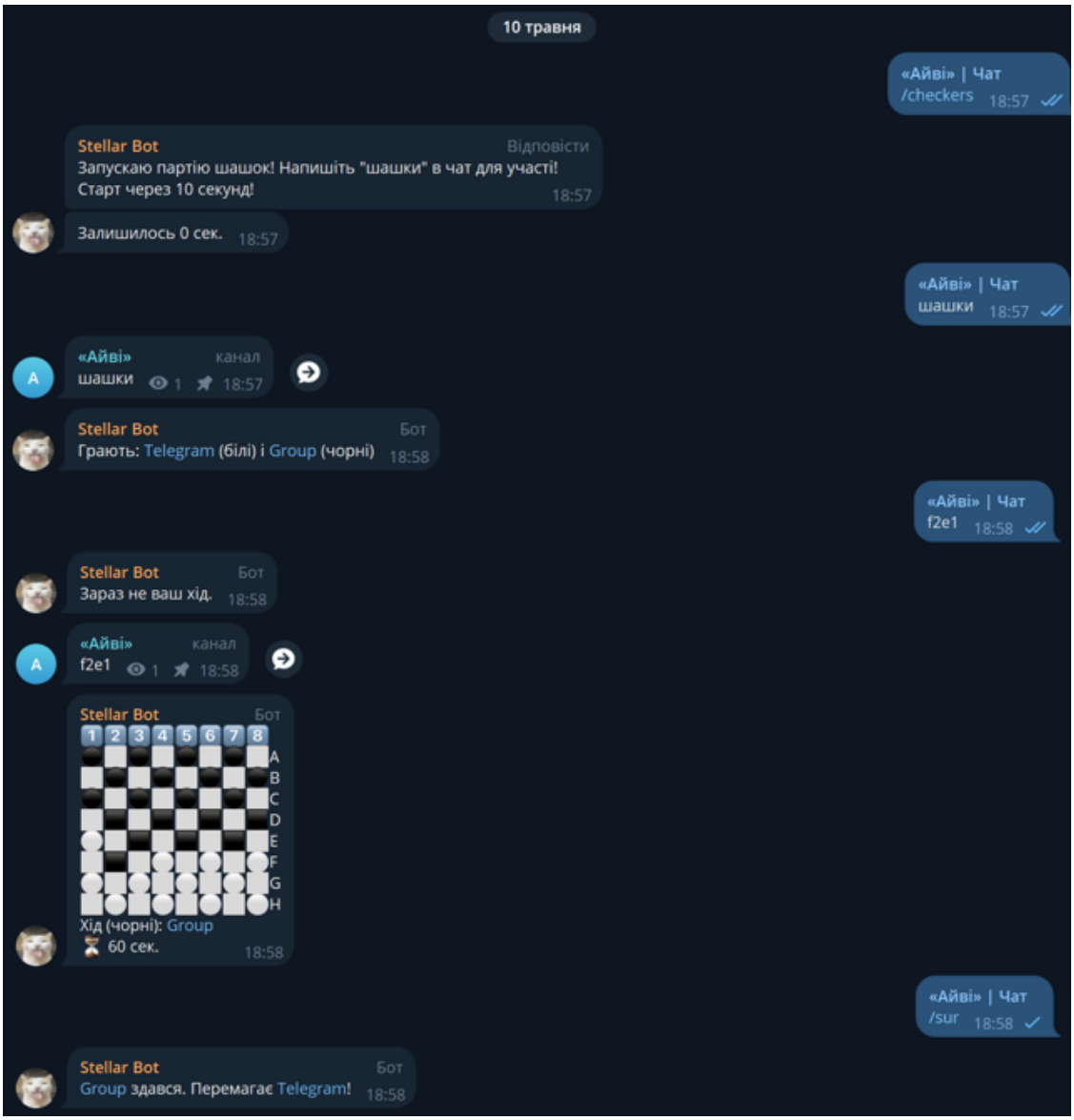


Рисунок 4.14 – Шашки в чаті

Окрім функціонального тестування команд, була проведена перевірка роботи системи на різних пристроях. Програмний модуль тестувався на смартфонах з операційною системою Android 10, а також на десктопних і веб-додатках Telegram для Windows 10. В усіх випадках бот правильно обробляв команди, відправляв повідомлення та взаємодіяв із користувачами без затримок та помилок.

Протестовано можливість одночасної роботи з різними типами пристроїв. Наприклад, користувач міг надіслати запит через мобільний додаток, тоді як адміністратор обробляв звернення через десктопну версію Telegram. Система правильно синхронізувала обробку запитів між різними пристроями.

Оцінка продуктивності показала, що обробка стандартних запитів та команд відбувається миттєво, затримки мінімальні та практично непомітні для користувачів. Бот витримує навантаження навіть при одночасному надходженні великої кількості запитів, що забезпечує стабільну роботу в активних спільнотах та великих каналах.

Крім цього, було перевірено надійність роботи бота при нестабільному інтернет-з’єднанні. У випадках короткочасної втрати зв'язку бот коректно відновлював сесію після відновлення мережі без потреби повторного введення команд.

На основі проведених тестувань можна зробити висновок, що розроблені програмні методи є стабільними, продуктивними та повністю готовими до використання у реальних умовах керування каналами та чатами у месенджері Telegram.

4.3. Аналіз результатів в різних сценаріях

Для повноцінної перевірки працездатності та універсальності бота було протестовано кілька специфічних сценаріїв, що можуть виникнути в взаємодії з спільнотами і користувачами. Ці тести дозволили виявити особливості роботи системи в нестандартних умовах.

Перший сценарій полягає у спробі приховати заборонене слово шляхом вставки сторонніх символів – емодзі, великих літер або знаків пунктуації. Наприклад, користувач може вписати якесь емодзі в заборонене слово, поставити якийсь знак чи змінити букви на великі літери, щоб уникнути автоматичного блокування. Результат зображено на рисунку 4.15.

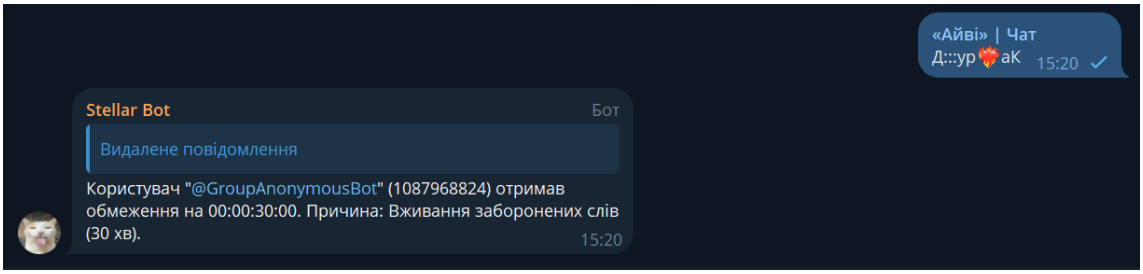


Рисунок 4.15 – Спроба приховати заборонене слово

Бот попередньо очищає повідомлення від зайвих символів, залишаючи лише літери, завдяки чому успішно виявляє і фільтрує заборонене слово.

Другий сценарій – масове надходження спам-повідомлень у групу з забороненими словами (рис. 4.16). Було імітовано ситуацію, коли в групі починаю надсилатися багато повідомлення. Завдяки вбудованим словникам бот автоматично розпізнавав повідомлення, що містять заборонені слова або посилання, і одразу блокував джерело порушення.

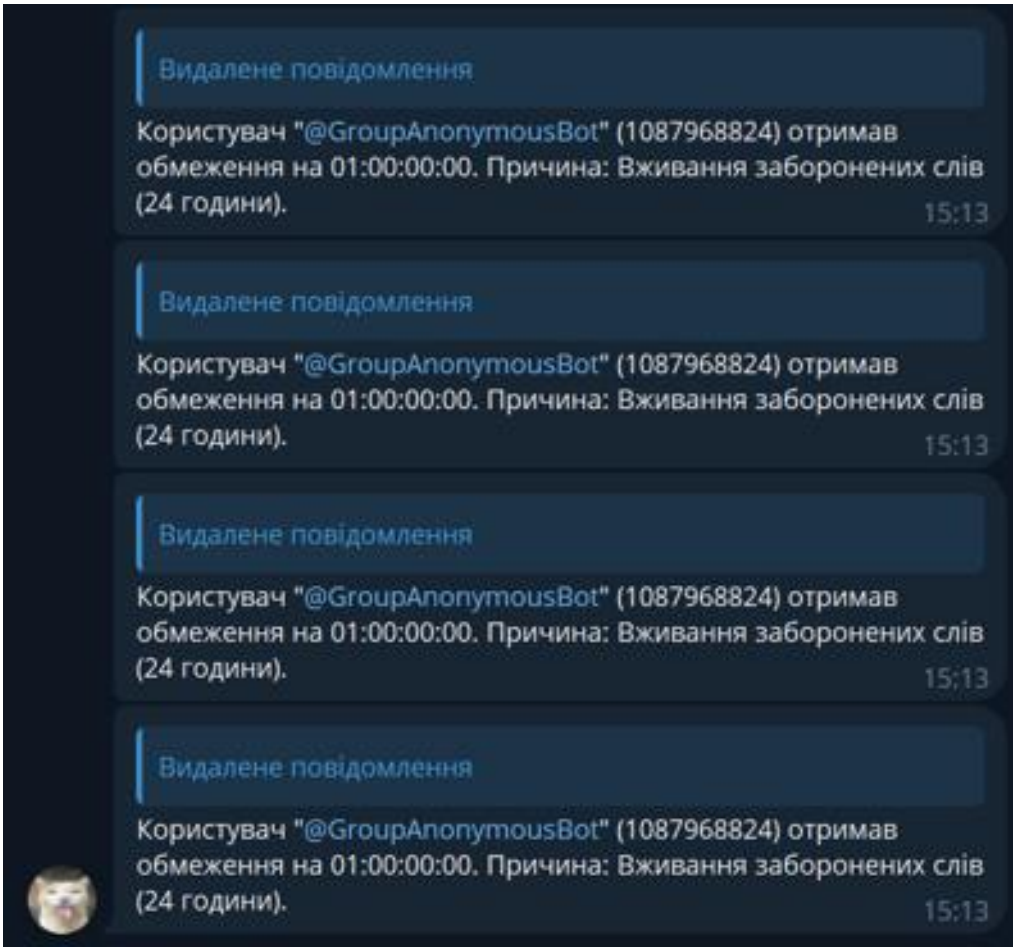


Рисунок 4.16 – Автоматичне блокування під час спаму

Третій сценарій – повторна команда «/block» до вже заблокованого користувача. Адміністратор двічі поспіль намагався заблокувати одного й того ж користувача. Бот коректно обробив повторне звернення, надіслав повідомлення про те, що користувача знову заблоковано, тим самим подовживши дію блокування. Результат зображено на рисунку 4.17

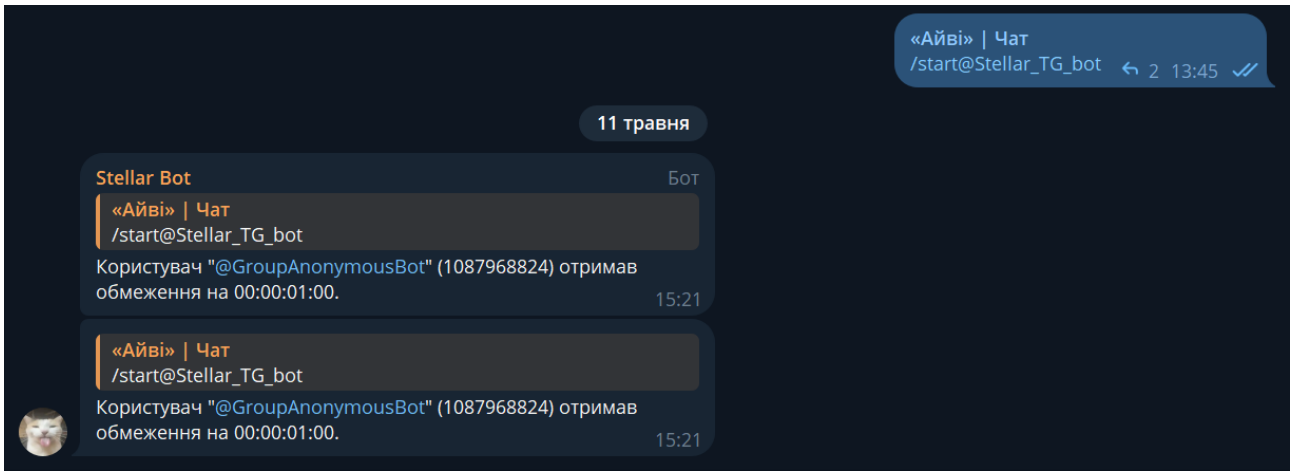


Рисунок 4.17 – Продовження блокування користувача

Сценарії показали, що розроблений бот демонструє високу адаптивність до нестандартних ситуацій, зберігає стабільність роботи під навантаженням і успішно виконує власні функції.

4.4. Висновки щодо продуктивності та надійності

Для ефективної експлуатації бота управління чатами та каналами в Telegram необхідно враховувати технічні вимоги, особливості його роботи та рекомендації щодо використання в різних умовах.

Деталі мінімальної та рекомендованої конфігурації подано в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	**Рекомендовано**	**Мінімально**
Процесор	Intel Core i5	Intel Core i3
Оперативна пам'ять	2 ГБ RAM	1 ГБ RAM
Вільний простір на диску	4 ГБ	2 ГБ
Операційна система	Windows 10 / Linux Ubuntu 20.04+ / macOS 11+	Windows 7 / Linux Ubuntu 16.04+ / macOS 8+

Продовження таблиці 4.1.

Вимоги до конфігурування	**Рекомендовано**	**Мінімально**
Підключення до Інтернету	10 Мбіт/с	2 Мбіт/с
Python	версія 3.10 і вище	версія 3.10 і вище

Висновки щодо продуктивності і надійності:

1. Бот стабільно обробляє до 10000 подій на годину без затримок;
2. Рекомендовано не перевищувати 5 паралельних чатів із великою активністю (>500 повідомлень/хв);
3. Система автоматично відновлює сесію після перезапуску або втрати підключення;
4. Команди, які запускають зміну прав або блокування, завжди дублюються в особисті повідомлення адміністраторам;
5. Бот не зберігає особистих даних користувачів, окрім їх повідомлень;
6. Можливість обмежити виконання команд лише для певних ID або ролей;

4.5. Висновки

У четвертому розділі проведено тестування алгоритмів бота з керування каналами та чатами у Telegram, що підтвердило їхню функціональність, стабільність, стійкість до збоїв, сумісність із різними версіями і видами пристроїв. Бот правильно обробляє команди користувачів та адміністраторів, забезпечує належний контроль доступу, коректно працює при високих навантаженнях і нестабільному інтернет-з’єднанні. Система готова до реального використання.

ВИСНОВКИ

У процесі виконання бакаларської кваліфікаційної роботи було розроблено алгоритми та програмні засоби для керування каналами і чатами у месенджерах – універсального бота для Telegram. Проведено аналіз сучасних месенджерів, їх API, наявних ботів і рішень для автоматизації керування чатами. Сформульовано функціональні вимоги до програмного забезпечення, побудовано логіку взаємодії користувача з ботом та Telegram API, реалізовано обробку команд і права доступу. Розроблено інтерфейс, реалізовано основні алгоритми й виконано інтеграцію з Telegram. Проведено тестування програмного продукту, яке підтвердило його працездатність, відповідність вимогам та стабільну роботу в різних сценаріях. Робота оформлена за методичними вказівками [23, 24, 25, 26].

Розробленого бота можна використовувати для автоматизованого адміністрування каналів та чатів у Telegram з можливістю розширення функціональності. Рекомендується впроваджувати його у проєкти, де важлива централізована та рольова модель управління контентом, повідомленнями і доступом. Також систему можливо адаптувати під інші месенджери з відкритим API.

Подальше удосконалення розробленого бота може включати підтримку мультиплатформеності, додавання нових функцій для обробки повідомлень, впровадження машинного навчання, а також створення більш зручнішого інтерфейсу для зручного адміністрування ботом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Боти для месенджерів [Електроний ресурс] URL: https://chatbotsmagazine.com/what-is-a-chatbot-5ff5e7fbd9e5 (дата звернення 05.04.2025).

2. Безкоштовні боти для керування чатами [Електроний ресурс] URL: https://manybot.io/ (дата звернення 05.04.2025).

3. Розробка універсальних ботів [Електроний ресурс] URL: https://core.telegram.org/bots (дата звернення 06.04.2025).

4. Стецула М.В., Мельник О.В. Розробка та застосування алгоритмів для керування каналами та чатами у месенджерах. LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) [Електроний ресурс] URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23314

5. Що таке месенджер? [Електроний ресурс] URL: https://bukowina.org.ua/blog/technology/shho-take-mesendzher-osnovni-funktsiyi-i-perevagy/ (дата звернення 06.04.2025).

6. Топ популярних месенджерів [Електроний ресурс] URL: https://vseosvita.ua/library/so-take-mesendzeri-aki-u-nih-mozlivosti-top-popularnih-mesendzeriv-281047.html (дата звернення 06.04.2025).

7. API (Application Programming Interface) [Електроний ресурс] URL: https://en.wikipedia.org/wiki/API (дата звернення 06.04.2025).

8. Автоматизація рутинних завдань [Електроний ресурс] URL: https://www.uipath.com/rpa/robotic-process-automation (дата звернення 06.04.2025).

9. Боти в Telegram [Електроний ресурс] URL: https://laba.ua/blog/1973-38-korisnih-botiv-dlya-telegram (дата звернення 08.04.2025).

10. Messenger Communication [Електроний ресурс] URL: https://www.ibm.com/topics/messaging (дата звернення 10.04.2025).

11. Статистика у спільнотах [Електроний ресурс] URL:

https://www.socialbakers.com/blog/social-media-community-management (дата звернення 10.04.2025).

12. Управління правами учасників [Електроний ресурс] URL: https://telegram.org/faq#groups-and-channels (дата звернення 12.04.2025).

13. Python Callback Functions [Електроний ресурс] URL: https://docs.python.org/3/tutorial/controlflow.html#defining-functions (дата звернення 12.04.2025).

14. Python OS Module [Електроний ресурс] URL: https://realpython.com/python-os-module/ (дата звернення 12.04.2025).

15. Порівняння мов програмування [Електроний ресурс] URL: https://uk.itpedia.nl/2023/07/13/6-moderne-programmeertalen-en-hun-downsides/ (дата звернення 12.04.2025).

16. Python for bot development [Електроний ресурс] URL: https://botpress.com/docs/installation/python (дата звернення 12.04.2025).

17. Visual Studio Code [Електроний ресурс] URL: https://code.visualstudio.com (дата звернення 12.04.2025).

18. Node-telegram-bot-api [Електроний ресурс] URL: https://github.com/yagop/node-telegram-bot-api (дата звернення 16.04.2025).

19. Бібліотека python-telegram-bot [Електроний ресурс] URL: https://python-telegram-bot.readthedocs.io (дата звернення 16.04.2025).

20. Telegram message formatting and keyboard [Електроний ресурс] URL: https://python-telegram-bot.readthedocs.io/en/stable/telegram.html#keyboardmarkup (дата звернення 19.04.2025).

21. Python bot moderation features [Електроний ресурс] URL: https://botfather.telegram.org/ (дата звернення 19.04.2025).

22. Experiences of Test Automation: Case Studies of Software Test Automation, Дороті Ґрем, Марк Фьюстер Ґрем, Фьюстер, 2019. С. 672

23. Цисарж В. В. Математические методы компьютерной графики / В. В. Цисарж, Р. И. Марусин. – Киев : Факт, 2004. – 464 с.

24. Романюк О. Н. Квадратична апроксимація BRDF / О. Н. Романюк,

Ю. Л. Ляшенко // Вісник Вінницького політехнічного інституту. – 2007. – № 1. – С. 67–69.

25. Романюк О. Н. Класифікація дистрибутивних функцій відбивної здатності поверхні / О. Н. Романюк // Наукові праці Донецького національного технічного університету. – Серія «Інформатика, кібернетика і обчислювальна техніка». – 2008. – Випуск 9 (132). – С. 145–151.

26. Романюк О. Н. Високопродуктивні методи та засоби зафарбовування тривимірних графічних об’єктів : монографія. / О. Н. Романюк, А. В. Чорний. – Вінниця : УНІВЕСУМ-Вінниця, 2006. – 190 с.

ДОДАТКИ

ДОДАТОК А
Технічне завдання, обов'язковий

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н.
Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка алгоритмів та програмних засобів для керування каналами і чатами у месенджерах» за спеціальністю
121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ О.В. Мельник
«14» березня 2025 р.
Виконав:
студент гр. ЗПІ-21б М.В. Стецула
«14» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка алгоритмів та програмних засобів для керування каналами і чатами у месенджерах». Галузь застосування – системи автоматизації керування месенджерів.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності керування каналами та чатами за рахунок створення універсального бота для месенджера, який автоматизує базові дії, забезпечує модерацію, управління доступами, логування подій, що дозволяє мінімізувати витрати часу на адміністративні завдання.

Призначення роботи – автоматизація модерації, управління контентом і взаємодії з користувачами в каналах і чатах месенджера.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Цисарж В. В. Математические методы компьютерной графики / В. В. Цисарж, Р. И. Марусин. – Киев : Факт, 2004. – 464 с.

2. Романюк О. Н. Квадратична апроксимація BRDF / О. Н. Романюк, Ю. Л. Ляшенко // Вісник Вінницького політехнічного інституту. – 2007. – № 1. – С. 67–69.

3. Романюк О. Н. Класифікація дистрибутивних функцій відбивної здатності поверхні / О. Н. Романюк // Наукові праці Донецького національного технічного університету. – Серія «Інформатика, кібернетика і обчислювальна техніка». – 2008. – Випуск 9 (132). – С. 145–151.

4. Романюк О. Н. Високопродуктивні методи та засоби зафарбовування тривимірних графічних об’єктів : монографія. / О. Н. Романюк, А. В. Чорний. – Вінниця : УНІВЕСУМ-Вінниця, 2006. – 190 с.

5. Технічні вимоги

Базові методи керування ботами – методи універсальної обробки команд і повідомлень; моделі реагування – моделі визначення команд користувача, моделі класифікації повідомлень; режим обробки даних – реальний час; вихідні дані для обробки – текстові та медіа повідомлення; дані про користувачів у чаті, права доступу, типи повідомлень; коефіцієнт пріоритетності обробки; ідентифікатори чатів, каналів, користувачів; вихідні дані – виконані дії бота відповідно до обраного алгоритму, оновлений стан каналів та чатів.

6. Конструктивні вимоги.

Система повинна відповідати технічним вимогам, повинна бути зручною в інтерфейсі та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до БКР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз API месенджерів та існуючих ботів	25.03.25 – 06.04.25
2	Формування функціональних вимог до бота	07.04.25 – 20.04.25
3	Розробка архітектури бота та алгоритмів	20.04.25 – 07.05.25
4	Реалізація програмного модуля та тестування	07.05.25 – 20.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

Протокол перевірки БКР на плагіат, обов'язковий

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка алгоритмів та програмних засобів для керування каналами і чатами у месенджерах»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3,2 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Мельник О. В. доцент
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Стецула М. В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг програми, обов’язковий

Лістинг bot_main.py

```
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup, ChatPermissions
from telegram.ext import Application, CommandHandler, MessageHandler, CallbackQueryHandler, filters, CallbackContext, ContextTypes
from bot_checkers import CheckersGameManager
import asyncio, re, datetime, os, json

TOKEN = "Secret"

ChannelID = -1002192428921
ChatID = -1002168139156
MyID = 1939289126
MyChatID = 1087968824
MyChannelID = 777000

Admin_ban_words = 1

ADMIN_IDS = [MyID, MyChatID, MyChannelID]
ADMIN_COMMANDS = [
    "block (на 7d7h7m7s чи 100s) чи (до YYYY:MM:DD:HH:MM чи HH:MM) *причина*- заблокувати користувача",
    "unblock UserID чи у відповідь до повідомлення бота - розблокувати користувача в каналі",
    "analiz - вкл/викл розшерений аналіз повідомлення",
    "botchat текст - написати в чат канала через бота",
    "timer 7d7h7m7s/100s чи YYYY:MM:DD:HH:MM/HH:MM *file* - таймер в каналі",
```

```
    "timerstop - зупиняє таймер в каналі",
]
USER_COMMANDS = [
    "start - команди бота, які доступні тобі",
    "getid - взнати своє чи чиєсь ID",
    "getdate - взнати поточний час",
    "letter - написати автору листа (тільки в особисті)",
    "info - отримати інформацію від автора (тільки в особисті)\n\nВсякі ігри",
    "checkers - почати партію шашок",
    "checkershelp - правила шашок",
    "sur - здатися",
]

async def block(u: Update, c: CallbackContext):
    if u.effective_user.id in ADMIN_IDS and u.message.reply_to_message and c.args:
        d = parse_time(c.args[0])
        reason = " ".join(c.args[1:]) if len(c.args) > 1 else ""
        if d:
            user_to_restrict = u.message.reply_to_message.from_user
            await c.bot.restrict_chat_member(
                u.effective_chat.id, user_to_restrict.id,
                permissions=ChatPermissions(can_send_messages=False),
                until_date=asyncio.get_event_loop().time() + d
            )
            await u.message.delete()
            text = f'Користувач "@{user_to_restrict.username}" ({user_to_restrict.id})
отримав обмеження на {format_time(d)}.'
            if reason:
                text += f' Причина: {reason}'
            await u.message.reply_to_message.reply_text(text)
```

```
            keyboard =
InlineKeyboardMarkup([[InlineKeyboardButton("Розблокувати",
callback_data=f"unblock_{user_to_restrict.id}")]])
            for admin_id in ADMIN_IDS:
                await c.bot.send_message(admin_id, text, reply_markup=keyboard)


async def bad_words_ban(u: Update, c: CallbackContext):
    if not u.message or not u.message.text:
        return
    if u.effective_user.id in ADMIN_IDS and not Admin_ban_words:
        return
    message_text = re.sub(r"[^\w\s]", "", u.message.text.lower())
    words = set(message_text.split())
    paths = {
        1800: ("C:/D/Project/TGBot/badwords_30min.txt", "Вживання заборонених
слів (30 хв)."),
        86400: ("C:/D/Project/TGBot/badwords_24h.txt", "Вживання заборонених
слів (24 години).")
    }
    for dur, (file_path, reason) in paths.items():
        try:
            with open(file_path, encoding="utf-8") as f:
                bad_words = set(w.strip().lower() for w in f.read().split(",") if w.strip())
        except FileNotFoundError:
            continue
        if words & bad_words:
            until_time = datetime.datetime.utcnow() + datetime.timedelta(seconds=dur)
            await c.bot.restrict_chat_member(
                u.effective_chat.id, u.effective_user.id,
                permissions=ChatPermissions(can_send_messages=False),
```

```
            until_date=until_time
        )
        text = f'Користувач "@{u.effective_user.username}" ({u.effective_user.id})
отримав обмеження на {format_time(dur)}. Причина: {reason}'
        await u.message.reply_text(text)
        await u.message.delete()
        keyboard =
InlineKeyboardMarkup([[InlineKeyboardButton("Розблокувати",
callback_data=f"unblock_{u.effective_user.id}")]])
        for admin_id in ADMIN_IDS:
            await c.bot.send_message(admin_id, text, reply_markup=keyboard)
        break

async def unblock(u: Update, c: CallbackContext):
    if u.effective_user.id not in ADMIN_IDS:
        return

    user = None
    if u.message.reply_to_message:
        match = re.search(r'\((\d+)\)', u.message.reply_to_message.text)
        if match:
            user_id = int(match.group(1))
            user = await c.bot.get_chat(user_id)
    if not user and c.args:
        try:
            user = await c.bot.get_chat(int(c.args[0]))
        except ValueError:
            return

    if user:
```

```
        await c.bot.restrict_chat_member(ChatID, user.id,
permissions=ChatPermissions(can_send_messages=True))
        unblock_message = f'Користувач "@{user.username}" ({user.id})
розблокований у каналі.'

        if u.effective_chat.id == ChatID:
            await u.message.delete()
            await c.bot.send_message(ChatID, unblock_message)
        else:
            await u.message.reply_text(unblock_message)
            await c.bot.send_message(ChatID, unblock_message)

def parse_time(t):
    if re.fullmatch(r"\d{4}:\d{2}:\d{2}:\d{2}:\d{2}", t):
        now = datetime.datetime.now()
        target_time = datetime.datetime.strptime(t, "%Y:%m:%d:%H:%M")
        return int((target_time - now).total_seconds())
    if re.fullmatch(r"\d{1,2}:\d{2}", t):
        now = datetime.datetime.now()
        target_time = datetime.datetime.strptime(t, "%H:%M").replace(year=now.year,
month=now.month, day=now.day)
        if target_time < now:
            target_time += datetime.timedelta(days=1)
        return int((target_time - now).total_seconds())
    m = re.fullmatch(r"(?:(\d+)d)?(?:(\d+)h)?(?:(\d+)m)?(?:(\d+)s)?", t)
    if not m:
        return None
    d, h, m, s = (int(m.group(i) or 0) for i in range(1, 5))
    return d * 86400 + h * 3600 + m * 60 + s
```

```
def format_time(s):
    d, s = divmod(s, 86400)
    h, s = divmod(s, 3600)
    m, s = divmod(s, 60)
    return f"{d:02}:{h:02}:{m:02}:{s:02}"

active_timers = {}
async def countdown(c, chat_id, d, file_id=None):
    m = await c.bot.send_message(chat_id, format_time(d))
    active_timers[chat_id] = m.message_id
    while d > 0:
        await asyncio.sleep(1)
        if chat_id not in active_timers: return
        d -= 1
        try: await m.edit_text(format_time(d))
        except: return
    if file_id: await c.bot.send_document(chat_id, file_id)
    await m.delete()
    active_timers.pop(chat_id, None)

waiting_for_file = {}
async def timer_handler(u: Update, c: CallbackContext):
    if u.effective_user.id not in ADMIN_IDS:
        return
    global waiting_for_file
    args = c.args
    if u.message.document and u.effective_user.id in waiting_for_file:
        d, chat_id = waiting_for_file.pop(u.effective_user.id)
        file_id = u.message.document.file_id
        asyncio.create_task(countdown(c, ChannelID, d, file_id))
```

```
        return
    if not args:
        return
    if u.message.text.startswith("/timerstop"):
        if (message_id := active_timers.pop(ChannelID, None)):
            try:
                await c.bot.delete_message(ChannelID, message_id)
            except:
                pass
        return
    if u.message.text.startswith("/timer"):
        if "file" in args:
            args.remove("file")
            d = parse_time(" ".join(args))
            if d:
                waiting_for_file[u.effective_user.id] = (d, u.effective_chat.id)
                await u.message.reply_text("Надішліть файл.")
        else:
            d = parse_time(" ".join(args))
            if d:
                asyncio.create_task(countdown(c, ChannelID, d))


async def letter_handler(u: Update, c: CallbackContext):
    if u.effective_chat.type != "private":
        return
    if u.message and u.message.text and u.message.text.startswith("/letter"):
        topics = [
            ["Розбан", "🔓 Розбан"],
            ["Маю ідею", "💡 Маю ідею"],
            ["Хочу допомогати", "🤝 Хочу допомогати"],
```

```
        ]
        keyboard = [[InlineKeyboardButton(name,
callback_data=f"letter_{name.lower()}")] for name, _ in topics]
        c.user_data["letter_topics"] = {name.lower(): icon for name, icon in topics}
        await u.message.reply_text("Оберіть тему листа:",
reply_markup=InlineKeyboardMarkup(keyboard))
        return
    if u.callback_query and u.callback_query.data.startswith("letter_"):
        topic_key = u.callback_query.data.split("letter_")[1]
        topic = c.user_data.get("letter_topics", {}).get(topic_key)
        if topic:
            c.user_data.update({"letter_topic": topic, "awaiting_letter": True})
            await u.callback_query.message.reply_text("Напишіть лист в ОДНОМУ
повідомленні (можна прикріпити файли).")
        return
    if c.user_data.pop("awaiting_letter", False):
        topic = c.user_data.get("letter_topic", "Без теми")
        user, text = u.effective_user, u.message.caption or u.message.text or "Без
тексту"
        message = f"📩 НОВИЙ ЛИСТ\nТема: {topic}\nВід: @{user.username}
({user.id})\n\n{text}"
        file_attr = next((attr for attr in ["document", "photo", "video", "audio", "voice"]
if getattr(u.message, attr, None)), None)
        if file_attr:
            file_id = u.message.photo[-1].file_id if file_attr == "photo" else
getattr(u.message, file_attr).file_id
            await getattr(c.bot, f"send_{file_attr}")(MyID, file_id, caption=message)
        else:
            await c.bot.send_message(MyID, message)
        await u.message.reply_text("☑ Ваш лист відправлено!")
```

```
async def button_unblock_handler(u: Update, c: CallbackContext):
    query = u.callback_query
    if not query:
        return
    user_id = int(query.data.split("_")[1])
    await c.bot.restrict_chat_member(ChatID, user_id,
permissions=ChatPermissions(can_send_messages=True))
    user = await c.bot.get_chat(user_id)
    text = f'Користувач "@{user.username}" ({user.id}) розблокований у каналі.'
    await query.message.edit_text(text)
    await c.bot.send_message(ChatID, text)

async def info_handler(u: Update, c: CallbackContext):
    if u.effective_chat.type == "private":
        if u.message and u.message.text and u.message.text.startswith("/info"):
            buttons_info = [
                ["ЧаПи", "C:/D/Project/TGBot/FAQ.txt"],
                ["Правила", "C:/D/Project/TGBot/Rules.txt"],
                ["Контакти", "C:/D/Project/TGBot/Contacts.txt"],
            ]
            buttons = [
                [InlineKeyboardButton(name, callback_data=f"info_{name.lower()}")]
                for name, _ in buttons_info
            ]
            keyboard = InlineKeyboardMarkup(buttons)
            c.user_data["buttons_info"] = {name.lower(): path for name, path in
buttons_info}
            await u.message.reply_text("Виберіть потрібну тему:",
reply_markup=keyboard)
```

```
        elif u.callback_query and u.callback_query.data.startswith("info_"):
            button_name = u.callback_query.data.split("info_")[1]
            file_path = c.user_data.get("buttons_info", {}).get(button_name)
            if file_path and os.path.exists(file_path):
                await
u.callback_query.message.reply_document(document=open(file_path, "rb"))
            else:
                await u.callback_query.message.reply_text("Файл не знайдено.")

async def botchat(u: Update, c: CallbackContext):
    if u.effective_chat.type == "private" and c.args:
        text = " ".join(c.args)
        await c.bot.send_message(ChatID, text)

async def start(u, c):
    cmds = "\n".join(f"/{cmd}" for cmd in USER_COMMANDS)
    if u.effective_user.id in ADMIN_IDS:
        cmds += f"\n\nКоманди доступні лише адмінам:\n" + "\n".join(f"/{cmd}" for
cmd in ADMIN_COMMANDS)
    await u.message.reply_text(f"Команди доступні всім:\n{cmds}")

async def getid(u: Update, c: CallbackContext):
    if u.message.reply_to_message:
        target_user = u.message.reply_to_message.from_user
        await u.message.reply_text(f"ID цього користувача: {target_user.id}")
    else:
        await u.message.reply_text(f"Ваш ID: {u.effective_user.id}")

async def getdate(u: Update, c: CallbackContext):
    now = datetime.datetime.now().strftime("%Y:%m:%d:%H:%M")
```

```
    await u.message.reply_text(f"Сьогоднішня дата: {now}")

analiz_mode = False
async def analiz(update: Update, context: ContextTypes.DEFAULT_TYPE):
    global analiz_mode
    logs_dir = "logs"
    os.makedirs(logs_dir, exist_ok=True)
    raw_data = update.to_dict()
    json_text = json.dumps(raw_data, indent=2, ensure_ascii=False)
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S_%f")
    filename = os.path.join(logs_dir, f"{timestamp}.txt")
    with open(filename, "w", encoding="utf-8") as f:
        f.write(json_text)
    if update.message and update.message.text == '/analiz' and update.effective_user.id
in ADMIN_IDS:
        analiz_mode = not analiz_mode
        status = 'увімкнено' if analiz_mode else 'вимкнено'
        await update.message.reply_text(f'Аналіз режим {status}.')
    elif analiz_mode and update.effective_user.id in ADMIN_IDS:
        if len(json_text) > 4000:
            json_text = json_text[:4000] + '\n... (обрізано)'
        await context.bot.send_message(chat_id=ADMIN_IDS[0],
text=f'<pre>{json_text}</pre>', parse_mode='HTML')

active_checkers_games = {}
checkers_manager = CheckersGameManager()
async def checkers(update: Update, context: CallbackContext):
    chat_id = update.effective_chat.id
    game = active_checkers_games.get(chat_id)
    if not game:
```

```
        game = CheckersGameManager()
        active_checkers_games[chat_id] = game
    await game.initiate_game(update, context)

async def handle_checkers_message(update: Update, context: CallbackContext):
    chat_id = update.effective_chat.id
    game = active_checkers_games.get(chat_id)
    if game:
        await game.handle_message(update, context)

async def surrender_checkers(update: Update, context: CallbackContext):
    chat_id = update.effective_chat.id
    game = active_checkers_games.get(chat_id)
    if game:
        await game.surrender(update, context)

async def checkershelp(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = (
        "Правила гри в шашки:\n\n"
        "1. Щоб почати набір, напишіть команду /checkers у чат. Після цього всі охочі мають написати слово 'шашки' в чат протягом n секунд. Може бути запущена лише одна гра.\n"
        "2. Після завершення набору бот випадково обирає двох гравців і випадково призначає їм кольори: ◯ (білі) або ⬤ (чорні). Білі ходять першими.\n"
        "3. ♡ — дамки білих, ♥ — дамки чорних.\n"
        "4. Щоб зробити хід, введіть координати шляху через пари (літера-цифра) у форматі F2E3 (звідки–куди) в чат.\n"
        "5. Бити шашки суперника можна, стрибнувши через них на порожню клітинку. Якщо шашка противника знаходиться на E3, введіть координати у
```

```
форматі F2D4.\n"
        "6. Можна вказувати послідовність для багаторазових стрибків для мультиудару у форматі F2D4B6.\n"
        "7. Звичайні шашки ходять тільки вперед по діагоналі (бити можна в усі сторони). Дамки можуть ходити в будь-якому напрямку по діагоналях в необмеженій кількості.\n"
        "8. Якщо немає жодного доступного ходу, ви програєте.\n"
        "9. Щоб здатися, напишіть /sur.\n"
    )
    await update.message.reply_text(text, parse_mode="Markdown")

app = Application.builder().token(TOKEN).build()
app.add_handler(CommandHandler("start", start))
app.add_handler(CommandHandler("timer", timer_handler))
app.add_handler(CommandHandler("timerstop", timer_handler))
app.add_handler(CommandHandler("block", block))
app.add_handler(CommandHandler("getid", getid))
app.add_handler(CommandHandler("getdate", getdate))
app.add_handler(CommandHandler("unblock", unblock))
app.add_handler(CommandHandler("info", info_handler))
app.add_handler(CommandHandler("botchat", botchat))
app.add_handler(CommandHandler("checkers", checkers))
app.add_handler(CommandHandler("checkershelp", checkershelp))
app.add_handler(CommandHandler("sur", surrender_checkers))
app.add_handler(CallbackQueryHandler(button_unblock_handler,
pattern=r"unblock_\d+"))
app.add_handler(CallbackQueryHandler(info_handler, pattern=r"info_.*"))
app.add_handler(CallbackQueryHandler(letter_handler, pattern=r"letter_.*"))
app.add_handler(MessageHandler(filters.Document.ALL, timer_handler), group=0)
app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
```

```
bad_words_ban), group=1)
app.add_handler(MessageHandler(filters.ALL, letter_handler), group=2)
app.add_handler(MessageHandler(filters.TEXT & (~filters.COMMAND),
handle_checkers_message), group=3)
app.add_handler(MessageHandler(filters.ALL, analiz), group=4)

if __name__ == "__main__":
    app.run_polling()
```

Лістинг bot_checkers.py

```
import asyncio, random

JOIN_TIME = 30
MOVE_TIME = 60

class CheckersGameManager:
    def __init__(s):
        s.active_game = s.timer_running = False
        s.players, s.participants = [], set()
        s.board_message = s.timer_message = s.countdown_text_message = None
        s.current_turn = 0
        s.board = []
        s.chat_id = None
        s.move_timer_task = None
        s.move_time_left = MOVE_TIME

    async def initiate_game(s, u, c):
        if s.active_game or s.timer_running: return
        s.timer_running = True
```

```
        s.chat_id = u.effective_chat.id
        s.participants = set()
        s.countdown_text_message = await c.bot.send_message(chat_id=s.chat_id,
text=f'Запускаю партію шашок! Напишіть "шашки" в чат для участі! Старт
через {JOIN_TIME} секунд!')
        s.timer_message = await c.bot.send_message(chat_id=s.chat_id,
text=f"Залишилось {JOIN_TIME} сек.")
        asyncio.create_task(s._countdown(c, JOIN_TIME))

    async def _countdown(s, c, sec):
        while sec > 0:
            await asyncio.sleep(5)
            sec -= 5
            try: await s.timer_message.edit_text(f"Залишилось {sec} сек.")
            except: pass
        s.timer_running = False
        participants = list(s.participants)
        if len(participants) < 2:
            f = [f"[{u.full_name}](tg://user?id={u.id})" for u in participants]
            await c.bot.send_message(chat_id=s.chat_id, text=f"В шашках хотіли взяти
участь: {', '.join(f)}. \nНедостатньо гравців.", parse_mode="Markdown")
            return
        selected = random.sample(participants, 2)
        random.shuffle(selected)
        s.players = selected
        f = [f"[{u.full_name}](tg://user?id={u.id})" for u in s.players]
        await c.bot.send_message(chat_id=s.chat_id, text=f"Грають: {f[0]} (білі) і
{f[1]} (чорні)", parse_mode="Markdown")
        s.active_game = True
        s._init_board()
```

```
        await s._send_board(c)

    def _init_board(s):
        s.board = []
        for r in range(8):
            row = []
            for c in range(8):
                if (r + c) % 2 == 0:
                    row.append('●' if r < 3 else '◯' if r > 4 else '■')
                else: row.append('□')
            s.board.append(row)

    async def _send_board(s, c):
        txt = "1 □2 □3 □4 □5 □6 □7 □8 □\n"
        for i, r in enumerate(s.board): txt += "".join(r) + chr(65 + i) + "\n"
        p = s.players[s.current_turn]
        col = "білі" if s.current_turn == 0 else "чорні"
        if not s._has_moves(s.current_turn):
            winner = s.players[1 - s.current_turn]
            loser = s.players[s.current_turn]
            await c.bot.send_message(chat_id=s.chat_id, text=txt +
f"[{loser.full_name}](tg://user?id={loser.id}) не має доступних ходів. Перемагає
[{winner.full_name}](tg://user?id={winner.id})!", parse_mode="Markdown")
            s.active_game = False
            return
        if s.board_message:
            try: await s.board_message.delete()
            except: pass
        s.board_message = await c.bot.send_message(chat_id=s.chat_id, text=txt +
```

```
f"Хід ({col}): [{p.full_name}](tg://user?id={p.id})\n ⏳ {MOVE_TIME} сек.",
parse_mode="Markdown")
        if s.move_timer_task: s.move_timer_task.cancel()
        s.move_time_left = MOVE_TIME
        s.move_timer_task = asyncio.create_task(s._move_timer(c, s.current_turn))

    async def _move_timer(s, c, turn_index):
        while s.move_time_left > 0:
            await asyncio.sleep(5)
            if not s.active_game or s.current_turn != turn_index:
                return
            s.move_time_left -= 5
            col = "білі" if turn_index == 0 else "чорні"
            p = s.players[turn_index]
            txt = "1 □2 □3 □4 □5 □6 □7 □8 □\n"
            for i, r in enumerate(s.board):
                txt += "".join(r) + chr(65 + i) + "\n"
            await s.board_message.edit_text(
                txt + f"Хід ({col}): [{p.full_name}](tg://user?id={p.id})\n ⏳
{s.move_time_left} сек.",
                parse_mode="Markdown"
            )
        if s.active_game and s.current_turn == turn_index:
            loser = s.players[turn_index]
            winner = s.players[1 - turn_index]
            await c.bot.send_message(
                chat_id=s.chat_id,
                text=f"[{loser.full_name}](tg://user?id={loser.id}) не зробив хід за
{MOVE_TIME} секунд. Перемагає
[{winner.full_name}](tg://user?id={winner.id})!",
```

```
            parse_mode="Markdown"
        )
        s.active_game = False

    async def surrender(s, u, c):
        if not s.active_game: return
        user = u.effective_user
        ids = [p.id for p in s.players]
        if user.id in ids:
            loser_index = ids.index(user.id)
            winner = s.players[1 - loser_index]
            await c.bot.send_message(chat_id=s.chat_id,
text=f"[{user.full_name}](tg://user?id={user.id}) здався. Перемагає
[{winner.full_name}](tg://user?id={winner.id})!", parse_mode="Markdown")
            s.active_game = False

    def _is_inside(s, y, x): return 0 <= x < 8 and 0 <= y < 8

    def _get(s, y, x): return s.board[y][x] if s._is_inside(y, x) else None

    def _set(s, y, x, val): s.board[y][x] = val

    def _is_opponent(s, cell, white): return cell in (['⚫', '❤'] if white else ['⚪',
'♡'])

    def _is_empty(s, cell): return cell in ['■', '□']

    def _is_queen(s, cell): return cell in ['♡', '❤']
```

```
    def _has_moves(s, turn):
        white = turn == 0
        own, queen = ('◯', '♡') if white else ('⬤', '❤')
        for y in range(8):
            for x in range(8):
                piece = s._get(y, x)
                if piece not in [own, queen]: continue
                dirs = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
                if piece == own:
                    step = -1 if white else 1
                    for dy, dx in dirs:
                        ny, nx = y + dy, x + dx
                        if s._is_inside(ny, nx) and s._is_empty(s._get(ny, nx)) and dy == step:
                            return True
                        ny, nx = y + 2*dy, x + 2*dx
                        my, mx = y + dy, x + dx
                        if s._is_inside(ny, nx) and s._is_empty(s._get(ny, nx)) and
s._is_opponent(s._get(my, mx), white):
                            return True
                else:
                    for dy, dx in dirs:
                        cy, cx = y + dy, x + dx
                        enemy_found = False
                        while s._is_inside(cy, cx):
                            cell = s._get(cy, cx)
                            if s._is_empty(cell):
                                if enemy_found: return True
                                cy += dy; cx += dx
                            elif s._is_opponent(cell, white) and not enemy_found:
                                enemy_found = True
```

```
                    cy += dy; cx += dx
                else: break
            if not enemy_found:
                cy, cx = y + dy, x + dx
                if s._is_inside(cy, cx) and s._is_empty(s._get(cy, cx)):
                    return True
        return False

    async def handle_message(s, u, c):
        t = u.message.text.strip().upper()
        if s.timer_running and t.lower() == "шашки":
            s.participants.add(u.effective_user)
            return
        if not s.active_game: return
        if u.effective_user.id != s.players[s.current_turn].id:
            await c.bot.send_message(chat_id=s.chat_id, text="Зараз не ваш хід.")
            return

        if len(t) < 4 or len(t) % 2 != 0:
            await c.bot.send_message(chat_id=s.chat_id, text="Неправильний формат
ходу.")
            return

        steps = [(ord(t[i]) - 65, int(t[i+1]) - 1) for i in range(0, len(t), 2)]
        if any(not s._is_inside(y, x) for y, x in steps):
            await c.bot.send_message(chat_id=s.chat_id, text="Хід поза межами
дошки.")
            return

        fy, fx = steps[0]
```

```
        piece = s._get(fy, fx)
        white = s.current_turn == 0
        own, queen = ('◯', '♡') if white else ('⬤', '❤')

        if piece not in [own, queen]:
            await c.bot.send_message(chat_id=s.chat_id, text="Це не ваша шашка.")
            return

        captured = []
        valid = True
        for (y1, x1), (y2, x2) in zip(steps, steps[1:]):
            dy, dx = y2 - y1, x2 - x1
            if abs(dy) != abs(dx): valid = False; break
            if s._get(y2, x2) not in ['■', '□']: valid = False; break
            if s._is_queen(piece):
                sy, sx = (dy > 0) - (dy < 0), (dx > 0) - (dx < 0)
                cy, cx = y1 + sy, x1 + sx
                enemy, epos = None, None
                while cy != y2 and cx != x2:
                    cell = s._get(cy, cx)
                    if s._is_opponent(cell, white):
                        if enemy: valid = False; break
                        enemy, epos = cell, (cy, cx)
                    elif cell not in ['■', '□']: valid = False; break
                    cy += sy; cx += sx
                if not valid: break
                if enemy: captured.append(epos)
            else:
                if abs(dy) == 1 and s._is_empty(s._get(y2, x2)) and ((white and dy == -1)
or (not white and dy == 1)):
```

```
            if len(steps) > 2: valid = False; break
        elif abs(dy) == 2:
            my, mx = (y1 + y2)//2, (x1 + x2)//2
            if s._is_opponent(s._get(my, mx), white): captured.append((my, mx))
            else: valid = False; break
        else: valid = False; break

if not valid:
    await c.bot.send_message(chat_id=s.chat_id, text="Хід недопустимий.")
    return

s._set(fy, fx, '■')
for y, x in captured: s._set(y, x, '■')
ly, lx = steps[-1]
if (white and ly == 0 and piece == own): piece = '♡'
if (not white and ly == 7 and piece == own): piece = '♥'
s._set(ly, lx, piece)
s.current_turn ^= 1
await s._send_board(c)
```

ДОДАТОК Г

Ілюстративна частина, обов’язковий

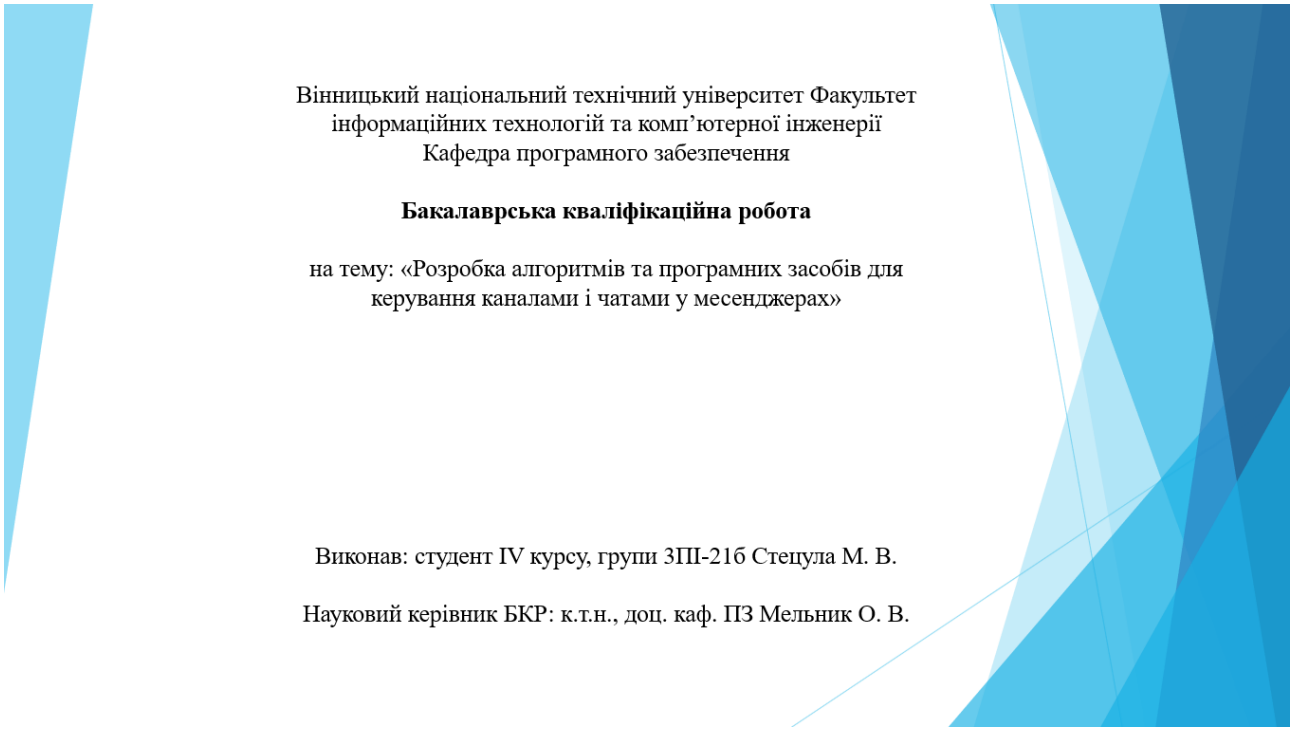


Рисунок Г.1 – Титульна сторінка

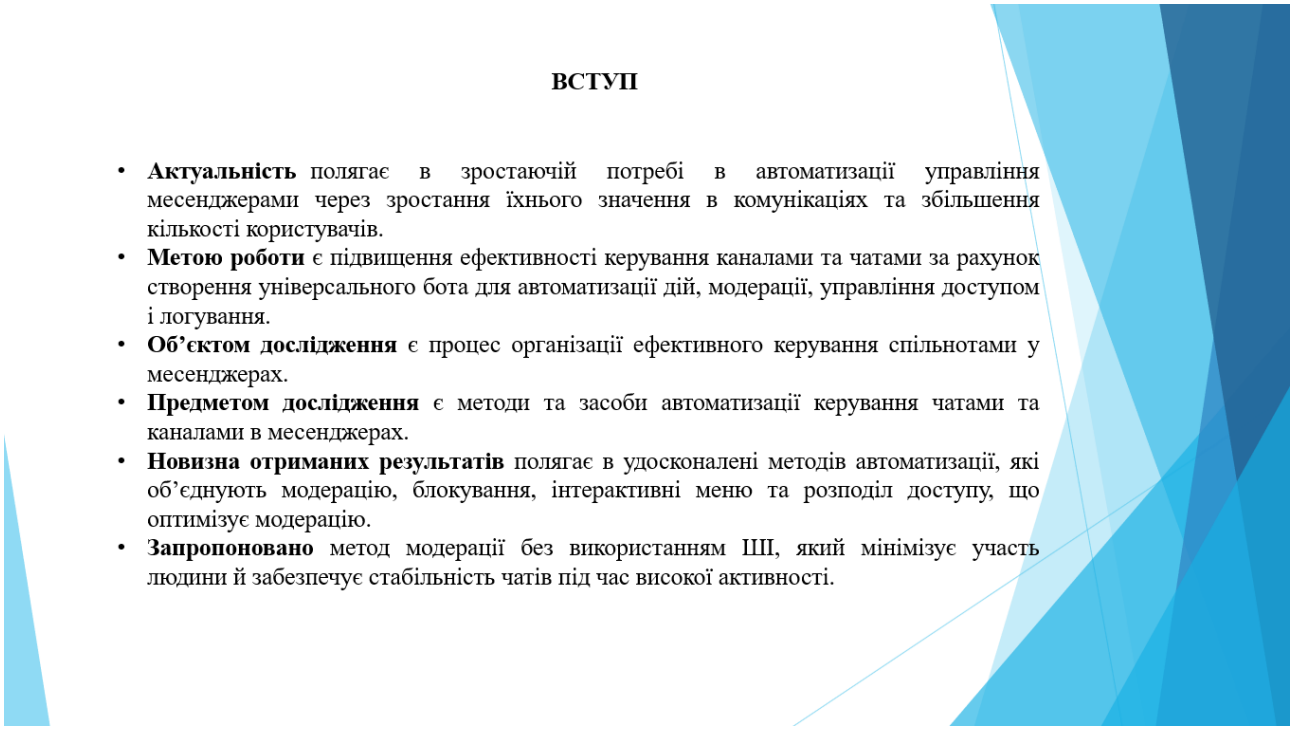


Рисунок Г.2 – Вступ

АНАЛІЗ ПОПУЛЯРНИХ АНАЛОГОВИХ БОТІВ

Критерії	Stellar Bot (Власний бот)	Puzzle Bot	Rose Bot	Junction Bot	ChatNorris Bot	Feed Reader Bot
Автоматизація керування	+	+	+	+	-	+
Модерація контенту	+	+	+	-	-	-
Інтуїтивність взаємодії	+	-	-	-	+	-
Простота налаштувань	+	+	+	+	+	+
Швидкість обробки команд	+	+	+	+	+	+
Загальна оцінка (1-10)	9	8	7	6	6	7

Аналіз популярних ботів показує, що кожен має свої переваги й недоліки: від модерації та публікацій до збору статистики чи інтеграції новин. Власний бот перевершує аналоги завдяки інтуїтивному інтерфейсу, простоті налаштування й широкому функціоналу.

Рисунок Г.3 – Аналіз аналогових ботів

ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ КЕРУВАННЯ КАНАЛАМИ НА ЧАТАМИ

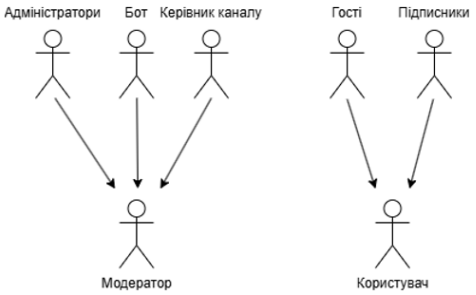
Побудова логіки взаємодії користувача і API

Особливістю ботів є взаємодія через текстові команди, а також вбудовані кнопки й інлайн-меню.
Сценарії взаємодії є інтуїтивними: модератори здійснюють управління і автоматизацію, а користувачі користуються чатом та доступним функціоналом.
Діаграма на рисунку демонструє логіку взаємодії користувача з API.

Рисунок Г.4 – Побудова логіки взаємодії користувача і API

ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ КЕРУВАННЯ КАНАЛАМИ НА ЧАТАМИ

Ролі в системі



Системні суб'єкти можна поділити за ролями:

1. Звичайні користувачі, яким передбачено базовий функціонал;
2. Модератори, які мають доступ до розширеного переліку команд для управління каналами та чатами.

Рисунок Г.5 – Ролі в системі

ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ КЕРУВАННЯ КАНАЛАМИ НА ЧАТАМИ

Визначення прав доступу до команд

Команди поділяються на три групи:

1. Ініційовані користувачем;
2. Напівавтоматизовані;
3. Повністю автоматизовані.

Алгоритм на зображенні перевіряє права користувача і визначає доступ до команди.

Рисунок Г.6 – Визначення прав доступу до команд

Критерій	python-telegram-bot	aiogram	pyrogram	telethon
Рівень абстракції	++	+	+-	-
Простота використання	++	+	+-	-
Підтримка асинхронності	++	++	+-	++
Підтримка спільноти	++	++	+	++
Гнучка логіка	++	++	+-	++
Підтримка складних сценаріїв	++	++	-	+-
Доступ до низькорівневих методів	+-	+	++	++
Стабільність	++	++	+-	++
Загальна оцінка	9/10	8/10	6/10	7/10

Критерії	Python	Go	Java	C#	Rust
Безпека	±	+	±	±	++
Продуктивність	±	+	+	++	++
Кількість бібліотек	++	±	+	+	±
Готові рішення для API	++	–	+	+	±
Підтримка багатопотоковості	±	+	+	+	+
Налагодження	++	±	+	+	±
Час на розробку	+	±	–	–	–
Загальна оцінка (1–10)	8	7	7	7	6

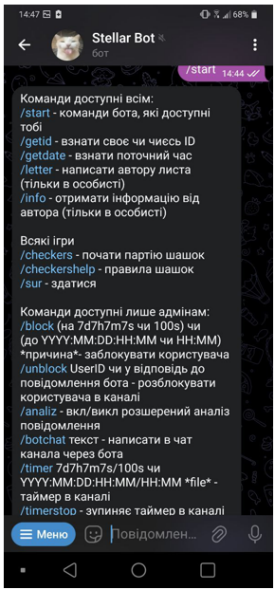
РОЗРОБКА БОТА ДЛЯ КЕРУВАННЯ КАНАЛАМИ ТА ЧАТАМИ

Вибір мови програмування, середовища та API

Для реалізації бота було обрано мову програмування Python, середовище Visual Studio Code, бібліотеку python-telegram-bot і месенджер Telegram, як найкращі варіанти.

Рисунок Г.7 – Вибір мови, середовища та API

РОЗРОБКА БОТА ДЛЯ КЕРУВАННЯ КАНАЛАМИ ТА ЧАТАМИ



Розробка інтерфейсу

Інтерфейс бота побудований у вигляді діалогу у чаті Telegram і містить вітальне повідомлення, головне меню з командами та інтерактивними кнопками. Для виконання дій користувачу достатньо обрати кнопку або надіслати команду, що запускає відповідний сценарій.

Рисунок Г8 – Розробка інтерфейсу

РОЗРОБКА БОТА ДЛЯ КЕРУВАННЯ КАНАЛАМИ ТА ЧАТАМИ

Розробка основних алгоритмів бота і їх інтеграція

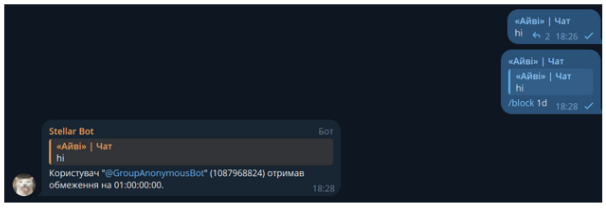
Основні алгоритми включають:
1. Перевірку повідомлень на заборонені слова;
2. Блокування;
3. Розблокування користувачів;
4. Обробку листів;
5. Таймери;
6. Логування;
7. Автоматизовану взаємодію з API.

Алгоритми реалізовано модульно і асихронно, з використанням бібліотек re, datetime, os, json та asyncio.

Рисунок Г.9 – Розробка основних алгоритмів і їх інтеграція

ВЕРИФІКАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

Проведення функціонального тестування

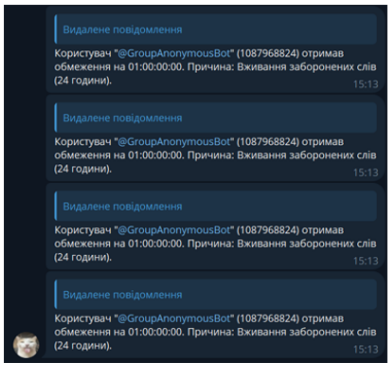


Методика тестування базувалася на перевірці роботи бота в реальному середовищі Telegram. Перевірка охопила як базову функціональність, так і роботу в умовах великої кількості запитів. Тестування включало імітацію дій реальних користувачів з різними сценаріями поведінки, що дозволило оцінити стабільність системи та її стійкість до помилкових або зловмисних дій.

Рисунок Г.10 – Проведення функціонального тестування

ВЕРИФІКАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

Аналіз результатів в різних сценаріях



Тестування охопило сценарії блокування з приховуванням поганих слів, автоматичного реагування на спам, відновлення сесії після втрати зв'язку.
Всі основні функції працювали коректно, без помітних збоїв.

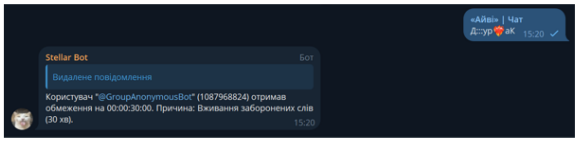


Рисунок Г.11 – Аналіз результатів в різних сценаріях

ВЕРИФІКАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

Висновки щодо продуктивності та надійності

Вимоги до конфігурування	**Рекомендовано**	**Мінімально**
Процесор	Intel Core i5	Intel Core i3
Оперативна пам'ять	2 ГБ RAM	1 ГБ RAM
Вільний простір на диску	4 ГБ	2 ГБ
Операційна система	Windows 10 / Linux Ubuntu 20.04+ / macOS 11+	Windows 7 / Linux Ubuntu 16.04+ / macOS 8+
Підключення до Інтернету	10 Мбіт/с	2 Мбіт/с
Python	версія 3.10 і вище	версія 3.10 і вище

Під час тестування бот стабільно обробляв всі повідомлення без затримок.
Проведені тести довели, що бот є продуктивним, надійним і готовим до повсякденного використання в активних спільнотах Telegram.

Рисунок Г.12 – Висновки щодо продуктивності та надійності

ВИСНОВКИ

- У процесі виконання БКР було створено Telegram-бота для керування чатами та каналами, реалізовано його алгоритми і логіку.
- Проведено аналіз API месенджерів, існуючих ботів і засобів автоматизації, визначено функціональні вимоги до системи.
- Розроблено інтерфейс користувача, реалізовано обробку команд і систему доступу, інтегровано рішення з Telegram API.
- Проведене тестування підтвердило працездатність, відповідність вимогам і стабільну роботу програмного продукту.
- Бот рекомендований до використання в інших системах, його можливо розширювати та адаптувати під інші платформи.

Рисунок Г.13 – Висновки

ДЯКУЮ ЗА УВАГУ!!!

Рисунок Г.14 – Фінальний слайд