КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ
Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від _____2026 року, протокол № _____
В.о.завідувача кафедри к.фіз.-мат.наук, доцент
_____ Ігор БЕХ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
на тему:

«АДАПТИВНА СИСТЕМА КОНТРОЛЮ ТА ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ ЧАТ-КОМУНІКАЦІЙ»

Виконав:
студент 2-го курсу магістратури
денної форми здобуття освіти
спеціальності 172 Електронні комунікації та радіотехніка
Інформаційна безпека телекомунікаційних систем і мереж
Ткаченко Олександр Олександрович_____

Науковий керівник:
к. т. н., с. н. с.
Жиров Генадій Борисович _____

Рецензент:
завідувач кафедри комп'ютерних наук
Державного університету інформаційно-комунікаційних технологій,
доктор технічних наук, професор
Вишнівський Віктор Вікторович_____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____ Олександр Ткаченко

Київ – 2026

РЕФЕРАТ

Дипломна робота: 64 с., 2 табл., 15 рис., 23 джерела.

ІНФОРМАЦІЙНА БЕЗПЕКА, ЧАТ-КОМУНІКАЦІЇ, CHATSHIELD HUB, КЛІЄНТ-СЕРВЕРНА СИСТЕМА, СПАМ, ФІШИНГОВІ ПОСИЛАННЯ, TELEGRAM, DISCORD, SIGNAL, FASTAPI, REACT, JWT.

Об’єкт дослідження — процес моніторингу, аналізу та автоматизованого реагування на загрози в чат-комунікаціях у багатоплатформному середовищі.

Мета роботи — підвищення ефективності моніторингу, аналізу та реагування на загрози в чат-комунікаціях засобами автоматизованої клієнт-серверної системи.

У дипломній роботі розглянуто основні загрози чат-комунікацій, зокрема спам, масові розсилки, фішингові посилання, повторювану зловмисну активність користувачів і ризики несанкціонованого доступу.

Проаналізовано існуючі підходи до захисту чат-середовищ і встановлено, що більшість рішень не забезпечують комплексного централізованого контролю подій у різних месенджерах.

У межах роботи спроєктовано та реалізовано систему ChatShield Hub, яка поєднує серверну частину, вебінтерфейс адміністратора, адаптери для Telegram, Discord і Signal, модулі виявлення спаму та перевірки посилань.

Серверну частину реалізовано з використанням FastAPI, PostgreSQL, Redis і WebSocket-з’єднань, а клієнтський інтерфейс — на основі React і TypeScript. У системі передбачено рольовий контроль доступу, JWT-автентифікацію, аудит адміністративних дій і захист конфігураційних даних.

Проведене тестування підтвердило працездатність системи та її здатність виявляти типові загрози в багатоплатформному середовищі.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування;

CRUD – створення, читання, оновлення та видалення даних;

DB – база даних;

E2E – наскрізне тестування;

HTTP – протокол передавання гіпертексту;

HTTPS – захищений протокол передавання гіпертексту;

IP – інтернет-протокол;

JSON – формат обміну даними JavaScript Object Notation;

JSONB – бінарний формат зберігання JSON-даних;

JWT – JSON Web Token, вебтокен для автентифікації та передавання даних про користувача.

ML – машинне навчання;

MVP – мінімально життєздатний продукт;

ORM – об’єктно-реляційне відображення;

RBAC – розмежування доступу на основі ролей;

REST – архітектурний стиль взаємодії програмних компонентів;

SIEM – система управління інформацією та подіями безпеки;

SPA – односторінковий застосунок;

SQL – мова структурованих запитів;

SVM – метод опорних векторів;

UI – інтерфейс користувача;

URL – уніфікований локатор ресурсу;

UX – користувацький досвід;

VPN – віртуальна приватна мережа;

WebSocket – протокол двостороннього обміну даними в реальному часі.

ВСТУП

Сучасні цифрові платформи комунікації, зокрема Telegram, Discord і Signal, стали важливою частиною повсякденного обміну інформацією, організації онлайн-спільнот та роботи цифрових сервісів [2; 6; 7]. Водночас зростання кількості користувачів і обсягів повідомлень у таких середовищах сприяє поширенню кіберзагроз, серед яких особливо актуальними є автоматизований спам, фішингові посилання, шкідливий контент і масові атаки на чат-спільноти [3; 4; 14]. У великих групах ручна модерація часто є недостатньо ефективною, оскільки адміністратори не завжди можуть своєчасно виявляти небезпечні повідомлення та реагувати на повторювані порушення.

Актуальність теми зумовлена необхідністю створення засобів автоматизованого моніторингу та модерації чатів у багатоканальних середовищах [1; 5; 15; 16]. Більшість наявних рішень орієнтовані на окремі платформи та не забезпечують достатнього рівня уніфікації, масштабованості й інтеграції із зовнішніми сервісами аналізу загроз. У зв’язку з цим важливим є розроблення програмної системи, здатної автоматично аналізувати повідомлення, виявляти ознаки спаму та фішингу, застосовувати правила реагування й надавати адміністраторам зручні інструменти контролю.

Розроблення системи ChatShield Hub спрямоване на вирішення проблеми централізованої модерації чатів шляхом поєднання локальних алгоритмів аналізу повідомлень, інтеграції із зовнішніми сервісами перевірки посилань і вебінтерфейсу адміністрування. Такий підхід передбачає створення багаторівневого механізму виявлення загроз з урахуванням повторюваності повідомлень, інтенсивності їх відправлення, схожості тексту та наявності потенційно небезпечних URL-адрес.

Об’єктом дослідження є процес моніторингу, аналізу та автоматизованого реагування на загрози в чат-комунікаціях у багатоплатформному середовищі.

Предметом дослідження є архітектура та програмні засоби побудови плагінно-орієнтованої системи реального часу для виявлення спаму, фішингових посилань та інших порушень у чат-середовищах, а також механізми адміністрування й автоматизованого реагування.

Метою роботи є підвищення ефективності процесів виявлення, моніторингу та реагування на загрози в чат-комунікаціях шляхом застосування інтегрованої системи ChatShield Hub, що поєднує плагінні модулі аналізу, адаптери комунікаційних платформ і веб-інструменти адміністрування.

Для досягнення поставленої мети в роботі передбачено аналіз сучасних загроз, характерних для чат-платформ, визначення функціональних і нефункціональних вимог до системи, проєктування архітектури ChatShield Hub з урахуванням модульності та масштабованості, реалізацію серверної частини, модулів аналізу повідомлень і механізмів реагування, розроблення вебінтерфейсу для адміністрування, перегляду подій, тікетів і метрик, забезпечення інтеграції з базою даних, кешем, зовнішніми API та контейнеризованим середовищем, а також проведення тестування системи й аналіз отриманих результатів.

Програмна реалізація системи передбачає використання сучасного технологічного стеку. Серверна частина побудована на основі Python, FastAPI, SQLAlchemy і Pydantic, клієнтська частина — з використанням React, TypeScript, Vite, Tailwind CSS і TanStack Query. Для зберігання даних та підтримки інфраструктури застосовано PostgreSQL, Redis і Docker Compose. Обраний технологічний підхід відповідає сучасним принципам побудови клієнт-серверних застосунків і застосунків реального часу [12; 17; 18; 23].

Практичне значення роботи полягає у можливості застосування розроблюваної системи для автоматизації модерації Telegram-чатів, Discord-серверів, Signal-груп та інших комунікаційних середовищ. Запропоноване

рішення орієнтоване на зменшення навантаження на адміністраторів, підвищення оперативності реагування на потенційні порушення та централізацію обліку інцидентів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ ЧАТ-КОМУНІКАЦІЙ

1.1. Особливості чат-комунікацій як об'єкта інформаційної безпеки

Чат-комунікації в сучасному цифровому середовищі є специфічним об'єктом інформаційної безпеки, оскільки поєднують високу інтенсивність обміну повідомленнями, багатоканальність, неструктурований характер контенту та безперервну взаємодію користувачів у режимі реального часу. На відміну від традиційних інформаційних систем, де обробка даних часто має транзакційний або документно-орієнтований характер, у середовищах миттєвого обміну повідомленнями домінують короткі повідомлення, швидка зміна контексту, неформальна мова, посилання, вкладення, повторювані фрази та масова взаємодія великої кількості учасників. Ці особливості істотно ускладнюють своєчасне виявлення небезпечної активності [2; 8; 21].

Суттєвою ознакою чат-комунікацій є їхній подієво-орієнтований характер. Будь-яке повідомлення, реакція системи, спрацювання аналітичного модуля або модераційна дія формують послідовність взаємопов'язаних подій (потік подій), які виникають у стислі часові інтервали та потребують оперативної обробки. Це означає, що безпека чат-комунікацій не може зводитися лише до захисту даних у стані зберігання або до класичного контролю доступу, а повинна охоплювати також динамічний моніторинг потоку повідомлень, аналіз поведінкових патернів і фіксацію інцидентів у момент їх виникнення [13; 15; 18].

Іншою важливою особливістю є мультиплатформеність чат-комунікацій. У межах сучасних організацій та спільнот комунікація часто здійснюється паралельно в Telegram, Discord, Signal та інших месенджерах, причому кожна платформа має власні протоколи взаємодії, формати повідомлень, механізми

ідентифікації користувачів і доступні засоби модерації. За таких умов об'єктом захисту виступає не окремий канал зв'язку, а сукупність розподілених середовищ миттєвого обміну, для яких необхідно забезпечити узгоджений підхід до спостереження, аналізу подій і застосування політик безпеки.

Специфіка чат-комунікацій також полягає у неструктурованому характері інформації. Повідомлення надходять природною мовою, можуть містити жаргонізми, скорочення, навмисні спотворення слів, однакові або майже ідентичні фрази, а також посилання на зовнішні ресурси. Унаслідок цього значна частина загроз не виявляється простими статичними правилами, оскільки шкідлива активність може маскуватися під звичайну комунікацію. Відмінність між легітимною та небезпечною поведінкою часто проявляється лише через сукупність ознак, таких як частота повідомлень, повторюваність змісту, низька варіативність тексту або наявність підозрілих адрес зовнішніх ресурсів.

Для чат-середовищ характерна висока швидкість поширення небажаного або шкідливого контенту. Навіть коротка затримка між появою повідомлення та реакцією системи може призвести до розсилання спаму, поширення фішингових посилань, дезінформаційного впливу або ескалації конфліктної взаємодії між учасниками. Саме тому інформаційна безпека в цій сфері повинна враховувати часовий параметр як критично важливу характеристику, а ефективність захисту має оцінюватися не лише за точністю виявлення, а й за швидкістю переходу від фіксації події до захисної дії.

Окрему складність становить поєднання технічних та соціально-поведінкових аспектів у чат-комунікаціях. Джерелом ризику можуть бути не лише зовнішні порушники, а й легітимні користувачі, чия поведінка змінюється ситуативно, а також автоматизовані облікові записи, боти або координовані групи учасників. Відповідно, чат-комунікації як об'єкт інформаційної безпеки мають розглядатися не лише як сукупність повідомлень, а як середовище взаємодії

суб'єктів, дій, намірів і цифрових слідів, що потребує контекстного аналізу [20; 21; 22].

Ще однією важливою рисою є необхідність поєднання функцій захисту, спостереження та керування. Для чат-середовищ недостатньо лише виявити загрозу; система повинна мати змогу пов'язати подію з конкретним каналом, користувачем, часовим інтервалом і подальшою управлінською реакцією. Це обумовлює значення централізованого журналювання подій, аудиту дій адміністраторів, рольового розмежування доступу та механізмів оперативної модерації, які разом формують цілісний контур забезпечення безпеки платформ миттєвого обміну повідомленнями.

Таким чином, чат-комунікації як об'єкт інформаційної безпеки характеризуються динамічністю, розподіленістю, неструктурованістю даних, високою швидкістю обміну та тісним поєднанням контентних і поведінкових ознак. Саме ці властивості визначають необхідність застосування адаптивних підходів до моніторингу й захисту, здатних працювати в режимі реального часу, враховувати контекст повідомлень і забезпечувати узгоджену реакцію на інциденти в багатоплатформеному середовищі миттєвого обміну повідомленнями.

1.2. Основні загрози та ризики в чат-комунікаціях

Чат-комунікації є середовищем підвищеного ризику, оскільки поєднують відкритість взаємодії, високу швидкість поширення повідомлень, масовість участі користувачів і складність оперативного контролю змісту повідомлень у режимі реального часу. У такому середовищі загрози мають не лише технічний, а й поведінковий характер, а їх реалізація часто відбувається швидше, ніж традиційні засоби ручної модерації або постфактум-аналізу здатні на них

відреагувати. З огляду на це загрози в чат-комунікаціях доцільно розглядати за кількома основними групами: контентні, поведінкові, технічні та організаційні.

До контентних загроз насамперед належать спам і масове надсилання повідомлень, тобто масове надсилання однотипних, коротких або навмисно повторюваних повідомлень у межах малого часового інтервалу. Небезпека такого впливу полягає не лише у засміченні каналу зв'язку, а й у витісненні корисної інформації, зниженні доступності комунікації для легітимних учасників, створенні додаткового навантаження на модераторів та формуванні передумов для подальших шахрайських або деструктивних дій. Для чат-середовищ ця загроза є особливо характерною, оскільки повідомлення надходять безперервно, а їх масовість створює ефект інформаційного перевантаження [3; 4; 9; 10].

Не менш суттєвою контентною загрозою є поширення шкідливих або підозрілих посилань. У середовищах миттєвого обміну повідомленнями URL-адреси зовнішніх ресурсів можуть використовуватися для фішингу, перенаправлення на шкідливі ресурси, завантаження небезпечного вмісту або введення користувача в оману шляхом маскування справжнього призначення посилання. Особливість цього ризику полягає в тому, що такі посилання часто виглядають легітимно в контексті розмови, а сама атака використовує довіру, швидкість реакції користувача та неформальний характер спілкування, який знижує критичність сприйняття повідомлень [14; 20].

Окрему групу становлять поведінкові загрози, пов'язані з характером дій учасників чат-середовища. До них належать координовані або повторювані зловмисні дії, коли небажана активність розподіляється між кількома обліковими записами, каналами або хвилями повідомлень, а також використання ботів для масового розсилання, маніпулювання обговоренням або створення уявної масової підтримки певних тез. У таких випадках окреме повідомлення може не виглядати небезпечним, однак сукупність подій у певному часовому вікні свідчить про цілеспрямований вплив на чат-інфраструктуру або її учасників. Саме тому

виявлення поведінкових загроз потребує аналізу не лише змісту повідомлень, а й частоти дій, повторюваності сценаріїв та взаємозв’язків між обліковими записами.

Суттєвим ризиком є також помилки автоматизованого виявлення загроз. Для систем захисту чат-комунікацій однаково небезпечними є як хибнопозитивні спрацювання, коли легітимні повідомлення помилково визначаються як шкідливі, так і хибнонегативні результати, коли реальна загроза залишається непоміченою. Перший випадок порушує нормальний хід комунікації та знижує довіру користувачів до модераційної системи, тоді як другий безпосередньо створює умови для реалізації інциденту безпеки. У зв’язку з цим для чат-систем критично важливим є баланс між чутливістю механізмів детекції та контролем рівня помилкових спрацювань.

Поряд із контентними та поведінковими загрозами важливу роль відіграють технічні ризики, пов’язані з адміністративним та інфраструктурним контуром чат-систем. Якщо вебінтерфейси адміністрування, API, механізми інтеграції ботів або системи автентифікації не мають належного розмежування доступу, журналювання дій і захисту від підбору облікових даних, це створює умови для несанкціонованих змін конфігурації, зловживання адміністративними функціями, компрометації токенів або прихованого втручання в роботу системи. До цієї ж групи належать ризики перехоплення повідомлень, використання незахищених каналів зв’язку, зараження через вкладення, а також експлуатація вразливостей інтегрованих модулів і зовнішніх сервісів [11; 12; 19; 20].

Окремо слід виділити організаційні ризики, які виникають не стільки через самі повідомлення, скільки через недоліки процесів керування безпекою. За відсутності централізованого журналювання, аудиту дій адміністраторів, механізмів агрегування подій і зрозумілих політик реагування велика кількість однотипних інцидентів швидко перетворюється на інформаційний шум, унаслідок чого знижується якість прийняття рішень і зростає ймовірність

пропуску критичних подій. У таких умовах навіть наявність інструментів виявлення загроз не гарантує належного рівня захисту без належної організації процесу реагування.

Ще однією специфічною рисою ризиків у чат-комунікаціях є їх виражена часова залежність. Ефективність захисту в цьому середовищі визначається не лише фактом виявлення небезпечної активності, а й швидкістю переходу до дії, зокрема попередження, обмеження повідомлень, видалення вмісту або блокування джерела загрози. Навіть коротка затримка між появою небезпечного повідомлення та реакцією системи може сформувати вікно вразливості, у межах якого шкідливий контент уже встигає вплинути на користувачів або поширитися далі.

Таким чином, основні загрози та ризики в чат-комунікаціях охоплюють спам, масове надсилання повідомлень, поширення шкідливих посилань, координовану зловмисну активність, помилки автоматизованого виявлення, несанкціонований доступ до засобів адміністрування, конфігураційні вразливості та організаційні недоліки реагування. Сукупність цих чинників підтверджує, що захист чат-комунікацій повинен будуватися як багаторівневий процес безперервного спостереження, аналізу, фіксації та адаптивного реагування, а не як ізольований механізм фільтрації окремих повідомлень.

1.3. Аналіз існуючих рішень та обґрунтування актуальності розробки

Існуючі підходи до забезпечення безпеки чат-комунікацій можна умовно поділити на кілька груп: вбудовані платформні засоби модерації, спеціалізовані сервіси аналізу окремих типів загроз, правила та евристики для виявлення небажаного контенту, а також комплексні корпоративні рішення інформаційної безпеки. Проте аналіз цих підходів свідчить, що жодна з названих груп окремо не забезпечує повного покриття потреб багатоплатформеного чат-середовища, у

якому необхідно одночасно виконувати моніторинг повідомлень, оцінювання ризику, централізовану реєстрацію подій, оперативну модерацію та контроль адміністративних дій.

Першу групу становлять вбудовані механізми платформ, зокрема засоби ролей, дозволів, ботів і базової модерації у Telegram, Discord або Signal. Їх перевагою є безпосередня інтеграція з конкретним сервісом, однак основним недоліком залишається фрагментованість: кожна платформа має власний формат подій, власні сценарії керування та різну глибину автоматизації, унаслідок чого адміністрування безпеки в кількох каналах одночасно стає розрізненим і складним для уніфікованого контролю. Крім того, самі платформи орієнтовані передусім на забезпечення комунікації, а не на побудову єдиного аналітичного контуру безпеки для кількох середовищ одразу.

Другу групу формують спеціалізовані зовнішні сервіси перевірки загроз, зокрема рішення класу розвідки загроз для аналізу адрес зовнішніх ресурсів. Подібні засоби є ефективними для перевірки репутації посилань і виявлення відомих шкідливих ресурсів, оскільки надають результати аналізу на основі великої кількості антивірусних рушіїв, блоклистів і пов’язаних джерел контексту. Водночас такі сервіси не розв’язують задачу комплексного захисту чат-комунікацій, оскільки орієнтовані лише на окремий тип об’єктів, не враховують поведінкові патерни користувачів і не забезпечують повного циклу реагування всередині чат-платформи. Додатковим обмеженням є залежність від зовнішнього API та встановлених лімітів запитів, що може впливати на оперативність роботи в режимі реального часу.

Третя група рішень ґрунтується на правилах, евристичних моделях і системах зважування ознак, які традиційно використовуються для виявлення спаму. Класичним прикладом такого підходу є Apache SpamAssassin, у якому рішення формується на основі набору тестів, кожен з яких має власну вагу, а підсумковий результат визначається сумарним скоринговим значенням.

Перевагою таких рішень є інтерпретованість, відносна простота адаптації та можливість налаштування правил під конкретну предметну область. Водночас класичні схеми на основі правил здебільшого створювалися для електронної пошти або інших більш структурованих каналів, тому не повною мірою враховують специфіку чат-комунікацій, де критичними є швидкість реакції, короткі повідомлення, неформальна мова, повторювані фрази та постійна зміна контексту [9; 10].

Четверту групу становлять комплексні комерційні продукти безпеки, орієнтовані на централізоване керування, контроль комунікацій і захист корпоративного середовища. Такі рішення зазвичай забезпечують ширший функціонал, включно з аудитом, аналітикою, політиками доступу та засобами реагування, однак вони часто орієнтовані на загальнокорпоративні сценарії, мають закриту логіку ухвалення рішень, потребують значних ресурсів для впровадження або не дають достатньої гнучкості для адаптації під конкретні умови чат-взаємодії в різних месенджерах. Для дослідницької або прикладної розробки важливою є не лише функціональна повнота, а й архітектурна прозорість, можливість модифікації та обґрунтування обраних механізмів із позицій критеріїв ефективності.

Узагальнене порівняння основних груп рішень, їхніх переваг, обмежень і прикладів наведено в табл. 1.1.

Таблиця 1.1 — Узагальнення переваг і недоліків основних груп рішень

Тип рішення	**Переваги**	**Обмеження**	**Приклади**
Платформні засоби модерації	Пряма інтеграція з платформою,	Фрагментованість, відсутність єдиного центру керування	Telegram боти, Discord ролі/модерація,

Тип рішення	Переваги	Обмеження	Приклади
	швидке базове адміністрування		Signal функції безпеки
Сервіси розвідки загроз	Ефективна перевірка адрес і репутації об’єктів	Орієнтація на окремий тип загроз, залежність від зовнішнього API	VirusTotal API
Системи на основі правил та евристик	Інтерпретованість, налаштовуваність, скорингова модель	Слабка адаптованість до динаміки чатів і зміни контексту	Apache SpamAssassin
Комплексні корпоративні рішення	Централізація, аудит, політики доступу, ширший функціонал	Висока складність впровадження, закритість, обмежена гнучкість	SIEM-платформи контролю комунікацій

Аналіз наявних підходів дає підстави стверджувати, що їхнім ключовим обмеженням є ізольованість функцій. Одні рішення добре виявляють шкідливі посилання, але не керують ролями доступу та аудитом; інші підтримують модераційні дії, але не мають централізованої аналітики; треті надають статистику, але не забезпечують адаптивного аналізу потоку повідомлень у реальному часі. У результаті для практичного захисту чат-комунікацій доводиться поєднувати кілька інструментів, що ускладнює інтеграцію, підвищує ризик неузгодженості політик і знижує оперативність реагування.

На цьому тлі актуальність розробки спеціалізованої системи захисту чат-комунікацій обумовлюється необхідністю створення єдиного адаптивного середовища, яке поєднує моніторинг, аналіз, модерацію, журналювання подій, керування інцидентами та захист адміністративного контуру в межах однієї архітектури. Актуальність такого підходу підсилюється тим, що сучасні чат-середовища потребують не статичної фільтрації окремих повідомлень, а комплексного оцінювання загроз за набором контентних, поведінкових і технічних ознак із можливістю централізованого реагування. Саме сукупність невирішених проблем багатоплатформеності, фрагментованості засобів контролю, часової чутливості реагування та потреби в адаптивній аналітиці обґрунтовує доцільність подальшої розробки спеціалізованих систем забезпечення інформаційної безпеки чат-комунікацій [11; 13; 15; 16; 17].

РОЗДІЛ 2. ПРОЄКТУВАННЯ АДАПТИВНОЇ СИСТЕМИ КОНТРОЛЮ

2.1. Формування вимог до системи та аналіз сценаріїв використання

Формування вимог до адаптивної системи контролю та забезпечення інформаційної безпеки чат-комунікацій здійснювалося поетапно на основі аналізу сучасних загроз, характерних для середовищ обміну повідомленнями, зокрема спам-флуду, фішингових посилань і масових розсилок. Під час уточнення вимог використано підходи, узгоджені з положеннями NIST SP 800-53, ISO/IEC 27001, а також з урахуванням актуальних тенденцій, відображених у матеріалах ENISA Threat Landscape 2025, де істотна частка інцидентів пов’язана із соціальною інженерією. Такий підхід дав змогу не лише сформувати перелік функцій системи, а й прив’язати їх до реальних сценаріїв порушень, що виникають у чат-комунікаціях у процесі експлуатації платформи ChatShield Hub.

Для впорядкування вимог застосовано метод MoSCoW, який дозволив розмежувати функціональні пріоритети з урахуванням практичної значущості окремих можливостей системи. До обов’язкових вимог віднесено інтеграцію з платформами Telegram, Discord і Signal через уніфіковані адаптери ботів. Такі адаптери реалізують стандартний набір операцій, необхідних для життєвого циклу підключення та керування ботом, а саме запуск, зупинку, виконання модераційних дій і перевірку поточного стану. До цієї ж групи належить модуль SpamDetector, призначений для виявлення спаму на основі правил, що враховують інтенсивність надходження повідомлень, їхню довжину, ступінь текстової подібності та рівень лексичної різноманітності. Система оцінювання ризику побудована на методі простого адитивного зважування, за якого сукупний бал, що перевищує встановлений поріг, призводить до класифікації повідомлення

як спаму. Окремим обов’язковим елементом є перевірка посилань через VirusTotal API версії 3, що дає змогу віднести підозрілі або шкідливі адреси до відповідного рівня загрози та ініціювати подальші захисні дії.

Крім цього, у складі обов’язкових вимог визначено механізм автоматизованого реагування, який передбачає кілька рівнів впливу на порушника: від відсутності реакції до попередження, видалення повідомлення, тимчасового обмеження активності, вилучення користувача із чату або повного блокування. Така градація є важливою, оскільки дає змогу здійснювати не лише одноразове усунення загрози, а й поетапну ескалацію санкцій залежно від повторюваності порушень і ступеня ризику. У результаті платформа набуває адаптивного характеру, оскільки реакція на інцидент формується не механічно, а з урахуванням контексту події, історії поведінки користувача та природи виявленої загрози.

До часткових вимог віднесено підсистему модераційних тікетів, яка забезпечує облік інцидентів, їхню подальшу обробку адміністраторами або модераторами, а також можливість закриття, амністії чи повторного відкриття записів. Такий механізм є важливим для підтримання зручності адміністрування системи, оскільки поєднує автоматичне виявлення порушення з подальшим контрольованим управлінням процесом обробки інциденту. До можливих вимог зараховано інтеграцію моделей машинного навчання, зокрема SVM і Random Forest, однак на етапі поточної реалізації вони були віднесені до перспектив розширення. Відкладеними вимогами визначено корпоративні сценарії масштабування, зокрема федерацію серверів, які не входили до меж першої робочої версії системи.

Нефункціональні вимоги сформовано з акцентом на продуктивність, безпеку та надійність. У межах продуктивнісних обмежень було зафіксовано прийнятний час виявлення спаму на рівні кількох секунд, а аналіз посилань — у межах, зумовлених зверненням до зовнішнього API. Окремо було встановлено

вимогу до дуже низької затримки обробки одного повідомлення в межах мілісекундного діапазону та пропускної здатності понад 300 подій за хвилину. У частині захисту даних передбачено рольовий контроль доступу з трьома рівнями повноважень, шифрування конфігураційних даних у PostgreSQL за допомогою VaultJSONB, а також обмеження частоти запитів для запобігання атакам методом перебору. Для надійності визначено вимогу до доступності на рівні 99,9 відсотка з використанням механізмів контролю працездатності Docker, що охоплюють перевірку готовності, стану та працездатності окремих сервісів.

Трасування вимог, зафіксоване у файлі methodology_requirements_traceability.md, підтверджує, що ключові функціональні й нефункціональні положення були реалізовані та перевірені у відповідних тестових сценаріях. Зокрема, для адаптерів підтверджено повне проходження сценаріїв запуску, для модуля SpamDetector зафіксовано значення метрики F1 на рівні 1,0, для перевірки посилань через VirusTotal відсутність хибнопозитивних спрацювань, а для механізму ескалації — повну коректність застосування дій. Крім того, підтверджено досягнення заданого значення затримки на рівні 95-го перцентиля в межах мілісекундного порогу.

Аналіз сценаріїв використання системи виконано з урахуванням ролей, які беруть участь у її експлуатації. Адміністратор системи здійснює створення та керування ботами, модулями, користувачами, а також моніторинг стану платформи через головну панель керування. Модератор чату працює з тікетами порушень, ухвалюючи рішення щодо подальшої реакції на інциденти. Бот-адаптер, у свою чергу, виконує автоматичний моніторинг чатів і передає події до центрального контуру обробки. Приклад головної панелі моніторингу, через яку адміністратор контролює стан системи, наведено на рис. 2.1.

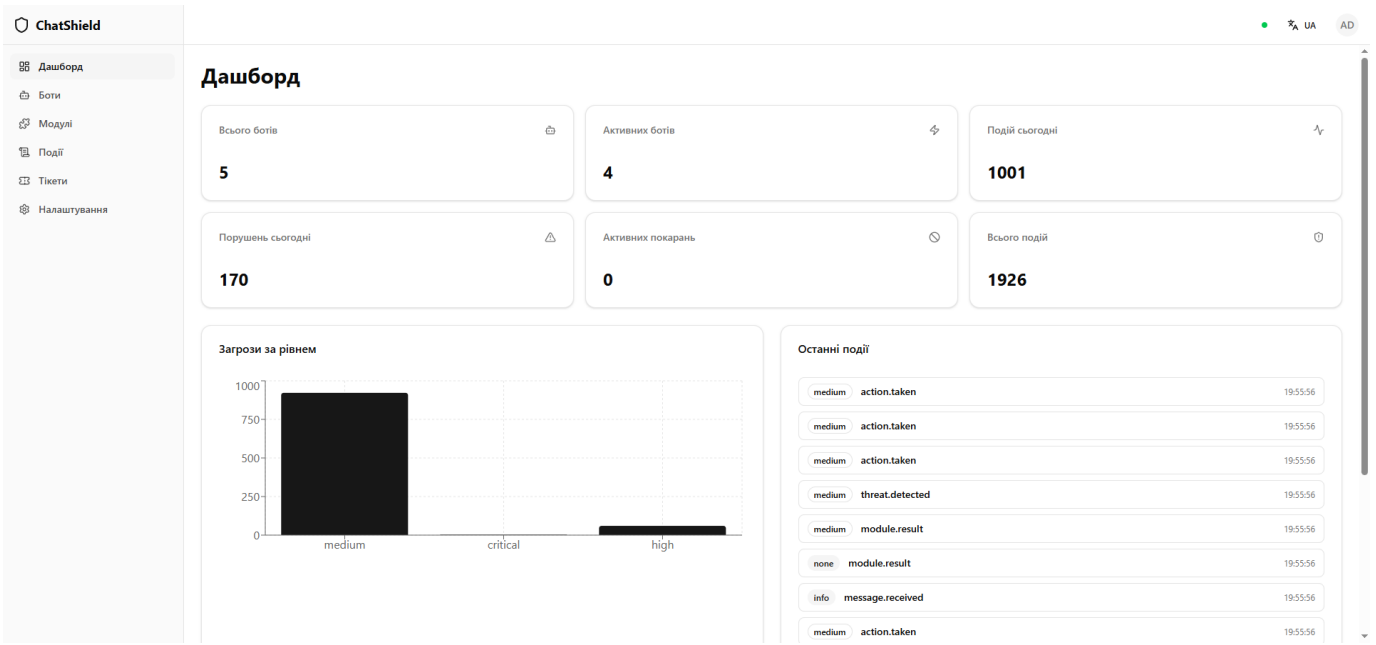

Рисунок 2.1 — Головний Dashboard з графіками, статистикою та останніми подіями

У межах верифікації було сформовано шість базових сценаріїв. Перший сценарій описує нейтральний трафік без загроз, за якого механізм не формує помилкових спрацьовувань. Другий сценарій моделює вибуховий спам, що призводить до виявлення загрози та застосування попередження з подальшим видаленням повідомлення. Третій сценарій стосується фішингових посилань, для яких передбачено диференційоване реагування залежно від рівня загрози. Четвертий сценарій відображає ескалацію повторного порушення після вже здійсненого попередження, унаслідок чого система переходить до більш жорсткого режиму обмеження. П’ятий сценарій описує змішане навантаження, що поєднує різні типи подій, а шостий — перевірку механізмів рольового контролю доступу, де неавторизовані операції мають бути повністю заблоковані. Практична перевірка цих сценаріїв у польових умовах підтвердила здатність системи обробляти значний обсяг подій і зберігати стабільність моніторингу в реальному часі.

2.2. Загальна архітектура та модель обробки повідомлень

Загальна архітектура системи ChatShield Hub побудована на основі подієво-орієнтованого підходу, за якого взаємодія між компонентами здійснюється не через жорстко зв’язані виклики, а через обмін асинхронними подіями. Така організація забезпечує слабке зв’язування підсистем, спрощує масштабування окремих функціональних вузлів і дає змогу реалізувати обробку повідомлень у режимі, наближеному до реального часу, без блокування основного виконуваного потоку. Центральним елементом цієї архітектури є EventBus, який у межах реалізації ChatShield Hub виконує функцію внутрішньої шини подій та забезпечує маршрутизацію повідомлень між адаптерами платформ, аналітичними модулями, службами збереження подій, підсистемою сповіщень і вебінтерфейсом моніторингу.

Подієва модель системи охоплює низку типових подій, які відображають життєвий цикл обробки повідомлення та зміну стану системи. До таких подій належать надходження нового повідомлення, виявлення загрози, виконання модераційної дії, запуск або зупинка бота, помилки роботи адаптерів, результати виконання окремих модулів аналізу, зміни конфігурації та операції, пов’язані з покаранням або амністією користувача. Використання єдиного подієвого контуру є принципово важливим, оскільки саме він забезпечує спостережуваність усіх етапів обробки, повторне використання даних кількома службами одночасно та синхронізацію між серверною частиною й інтерфейсом адміністрування.

Архітектурно систему доцільно розглядати як чотиришарову. Перший шар утворює клієнтська частина, реалізована як односторінковий вебзастосунок на основі React і TypeScript, що включає сторінки моніторингу, керування ботами, модулями, подіями, тікетами та системними налаштуваннями. Цей рівень забезпечує візуалізацію поточного стану системи й отримує оновлення через WebSocket-з’єднання. Другий шар становить серверна частина на FastAPI, яка

надає прикладний програмний інтерфейс для автентифікації, керування ботами та модулями, доступу до журналу подій, тікетів, користувачів і службових налаштувань. На цьому ж рівні реалізовано JWT-автентифікацію, обробку ролей доступу та канал WebSocket для передавання подій у клієнтський інтерфейс.

Третій шар є ядром оркестрації, у межах якого зосереджено доменну логіку обробки повідомлень. Саме тут функціонують реєстр адаптерів BotRegistry, менеджер модулів ModuleManager, конвеєр аналізу (Pipeline) та виконавець дій ActionEngine. У сукупності ці компоненти формують логічний центр системи, який приймає нормалізовані повідомлення від платформ, передає їх на послідовний аналіз, агрегує результати й ініціює відповідні заходи реагування. Четвертий шар становить рівень даних та інтеграцій, де використовуються PostgreSQL для збереження подій, аудиту, тікетів, ботів, модулів і користувачів, Redis для підтримки подієвої взаємодії, кешування та допоміжних служб виконання, а також зовнішні інтерфейси, зокрема VirusTotal API для перевірки URL-адрес.

Окремої уваги потребує розгортання системи, оскільки воно безпосередньо впливає на її працездатність і відтворюваність експериментального середовища. У ChatShield Hub розгортання здійснюється через Docker Compose, де окремими сервісами представлені PostgreSQL, Redis, серверна частина FastAPI, клієнтська частина, зворотний проксі-сервер Nginx та окремий допоміжний компонент signal-cli для інтеграції з платформою Signal. У такій конфігурації забезпечується контейнеризоване середовище виконання, чітке розмежування залежностей і можливість автоматизованих перевірок готовності сервісів. Для моніторингу роботи системи застосовується структуроване JSON-логування з кореляційним ідентифікатором запиту X-Request-ID, що дає можливість відстежувати весь ланцюг проходження події через систему.

Модель обробки повідомлень у ChatShield Hub реалізовано на основі шаблону ланцюга відповідальності, у якому кожен аналітичний модуль

послідовно отримує контекст повідомлення, виконує власну перевірку й або формує результат, або передає обробку далі. Така організація є доцільною для задач інформаційної безпеки, оскільки різні класи загроз можуть аналізуватися незалежними механізмами без зміни загальної схеми роботи системи. На першому етапі адаптер конкретної платформи приймає повідомлення, нормалізує його до уніфікованої структури контекст повідомлення і передає до центрального контуру обробки. Для Telegram це здійснюється через aiogram із циклічним опитуванням Bot API, для Discord — через постійне WebSocket-з'єднання бібліотеки discord.py, а для Signal — через локальну взаємодію із signal-cli на основі JSON-RPC. Приклад форми додавання Telegram-бота з токеном і переліком чатів подано на рис. 2.2.

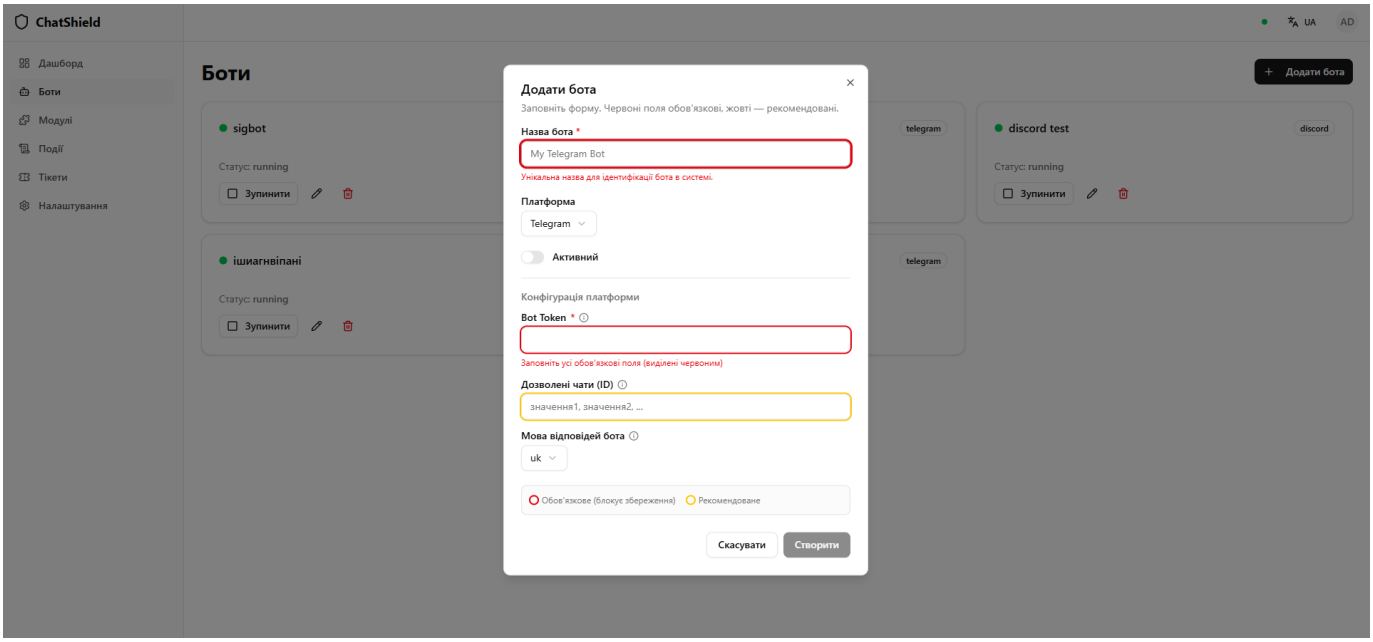

Рисунок 2.2 — Форма додавання нового бота Telegram з токеном та чатами

Після нормалізації повідомлення адаптер генерує подію message.received, яка надходить до EventBus і стає доступною підписаним компонентам. Оркестратор приймає цю подію, формує повний контекст повідомлення разом із метаданими та передає його до конвеєра аналізу. Далі конвеєр аналізу викликає

модулі відповідно до заданого пріоритету. У поточній конфігурації модуль VirusTotalAnalyzer виконується раніше, оскільки перевірка посилань може виявити критичну загрозу незалежно від поведінкових характеристик повідомлення, тоді як SpamDetector аналізує інтенсивність потоку, повторюваність змісту, довжину тексту та інші ознаки спаму. Для звернення до VirusTotal використовується асинхронний HTTP-клієнт, а у випадках перевищення квоти запитів застосовується повторна спроба з поступовим збільшенням інтервалу очікування. SpamDetector, у свою чергу, повертає значення довіри до виявленої загрози на основі обчисленого зваженого балу.

Кожен модуль аналізу формує частковий результат, який містить рівень загрози, обґрунтування та рекомендовану дію. Після завершення послідовного проходження всіх активних модулів конвеєр обробки виконує агрегування результатів. У цій операції враховується найвищий рівень небезпеки серед усіх отриманих оцінок, а також поєднуються причини спрацювання, що дозволяє сформувати консолідований план реагування. Далі ActionEngine перетворює узагальнений результат на конкретну дію для відповідного адаптера платформи. На цьому етапі можуть бути виконані видалення повідомлення, надсилання попередження, тимчасове обмеження користувача, вилучення з чату або блокування. Після успішного виконання формується подія action.taken, а у випадку наявності суттєвої загрози — також подія threat.detected.

Паралельно з виконанням модераційної дії результати обробки зберігаються у базі даних за допомогою асинхронних сесій SQLAlchemy. У таблицях фіксуються як самі події, так і пов’язаний аудит, а в разі необхідності створюються або оновлюються модераційні тікети. Після цього дані повторно передаються до EventBus, а звідти — до WebSocket-шлюзу, який транслює їх у вебінтерфейс. На стороні клієнта події оновлюють локальний стан і ініціюють повторне отримання релевантних вибірок для панелі керування, журналу подій і

тікетів. Саме завдяки такій схемі Dashboard відображає зміни майже в реальному часі, що підтверджено в межах польових випробувань системи.

Важливою особливістю архітектури є можливість динамічної зміни конфігурації аналітичних модулів. Параметри модулів та адаптерів зберігаються у PostgreSQL у JSON-структурах, для яких використовується VaultJSONB з шифруванням конфіденційних полів. Це дає змогу не лише централізовано зберігати конфігурацію, а й формувати на її основі елементи інтерфейсу введення в адміністративній частині системи. Для ботів така схема вже реалізована повністю через серверні схеми конфігурації, тоді як для модулів у поточній версії частина налаштувань ще редагується у форматі сирого JSON, що відображає реальний стан реалізації прототипу та не суперечить його загальній архітектурній концепції.

Узгодженість архітектури з вимогами безпеки досягається завдяки інтеграції принципів Zero Trust. Доступ до серверного прикладного інтерфейсу та WebSocket-каналу контролюється через JWT, а операції керування ботами, модулями та користувачами додатково перевіряються механізмом рольового розмежування доступу. Також застосовується обмеження частоти запитів, аудит критичних змін і шифрування конфігураційних даних. У результаті загальна архітектура ChatShield Hub поєднує подієву модель, модульність, відокремленість платформених адаптерів і централізований контроль обробки повідомлень, формуючи єдину систему захисту чат-комунікацій.

2.3. Проєктування основних модулів системи

Проєктування основних модулів системи ChatShield Hub виконано в межах плагін-архітектури, що передбачає відокремлення алгоритмів аналізу від ядра системи та їхню динамічну підміну без внесення змін до базової логіки оркестрації. Такий підхід узгоджується з принципом відкритості для розширення

та закритості для модифікації, оскільки нові модулі можуть додаватися шляхом реєстрації в системі без порушення роботи вже наявних компонентів. Базовим контрактом для аналітичних модулів виступає абстрактний клас BaseAnalyzer, який визначає метод аналізу повідомлення, метод налаштування конфігурації, метод формування схеми конфігурації для інтерфейсу адміністрування, а також властивості, що задають унікальну назву модуля та його пріоритет у конвеєрі обробки.

Побудова модулів у межах ChatShield Hub реалізована таким чином, щоб кожен аналізатор був автономним функціональним блоком із чітко визначеним інтерфейсом взаємодії. Метод analyze отримує уніфікований контекст повідомлення та повертає результат, що містить рівень загрози, довіру до висновку, текстове обґрунтування і рекомендовану дію. Метод configure забезпечує приймання параметрів у форматі JSON, а get_config_schema формує структуру, на основі якої автоматично створюються елементи форми в адміністративній частині системи. Властивість priority використовується для впорядкування модулів у конвеєрі аналізу: модулі з меншим значенням пріоритету виконуються раніше, що є важливим для побудови послідовності перевірок, коли більш критичні або дешевші з погляду обчислень алгоритми мають застосовуватися першими.

Менеджер модулів ModuleManager виконує роль проміжного рівня між сховищем конфігурацій і конвеєром аналізу. Під час запуску системи він зчитує активні модулі з бази даних, де кожен запис містить ознаку увімкнення, параметри конфігурації та схему налаштувань, після чого впорядковує модулі відповідно до їхнього пріоритету і передає їх до конвеєра аналізу. Така модель забезпечує гнучкість керування складом аналітичних засобів, оскільки адміністратор може змінювати набір активних модулів без перекомпіляції або зміни коду оркестратора. Практично це означає, що контур обробки здатен

адаптуватися до нових класів загроз шляхом додавання нових модулів аналізу або зміни параметрів уже наявних.

Модуль SpamDetector реалізує правило-орієнтований механізм виявлення спаму, побудований на методі простого адитивного зважування. Його сутність полягає у послідовному обчисленні сукупного ризикового балу на основі кількох ознак, кожна з яких має власну вагу та внесок у кінцевий результат. До таких ознак належать надмірна інтенсивність надходження повідомлень, мала довжина тексту, повторюваність повідомлень і низька лексична різноманітність. У разі перевищення встановленого порога повідомлення класифікується як спам, а рекомендованою дією стає переведення користувача у тимчасовий режим обмеження активності.

У науково-практичному сенсі SpamDetector поєднує аналіз повідомлень у межах часових вікон із багатокритеріальною оцінкою ризику. Повідомлення користувача аналізуються в межах фіксованого інтервалу часу, після чого підраховується кількість елементів потоку, ступінь подібності текстів між собою, а також характеристика символічної та лексичної різноманітності. Якщо кількість повідомлень у вікні перевищує порогове значення, якщо текст є надто коротким або якщо спостерігається висока схожість і низька варіативність змісту, сукупний бал зростає. Коли цей бал перевищує поріг, результат набуває статусу середнього рівня загрози з рекомендацією застосувати обмежувальну дію. Саме така логіка дозволяє виявляти не лише очевидний флуд, а й більш приховані форми нав’язливого повторення змісту.

Окремим модулем є VirusTotalAnalyzer, призначений для перевірки URL-адрес у складі тексту повідомлення. Його робота ґрунтується на виділенні посилань із вхідного тексту та передаванні їх до зовнішнього сервісу VirusTotal через асинхронний HTTP-запит. У відповідь система отримує зведену репутаційну оцінку від кількох десятків антивірусних механізмів, після чого здійснюється класифікація посилання як чистого, підозрілого або шкідливого.

Якщо наявність загрози підтверджується хоча б частиною зовнішніх рушіїв, модуль формує високий рівень небезпеки і рекомендує видалення повідомлення з можливим подальшим обмеженням користувача.

Використання VirusTotalAnalyzer є важливим із точки зору розширення захисного контуру системи, оскільки він доповнює локальні евристичні перевірки зовнішньою репутаційною оцінкою. Такий підхід особливо корисний у випадках, коли небезпека прихована не у змісті самого тексту, а в ресурсі, на який він посилається. За наявності підозрілих або явно шкідливих URL-адрес система формує більш жорстку рекомендацію реагування, ніж у випадку звичайного спаму, що відповідає логіці підвищення рівня загрози залежно від характеру індикаторів компрометації.

Центральним компонентом моделі реагування є ActionEngine. Саме він отримує результати від усіх активних аналізаторів, узагальнює їх і перетворює на єдиний план дій. Агрегування виконується за правилом вибору максимального рівня загрози, після чого враховується історія попередніх порушень користувача. Така історія дає змогу реалізувати ескалаційну модель: перше порушення може призводити до попередження або видалення повідомлення, повторне — до обмеження активності, а наступні — до вилучення з чату або повного блокування. Отже, ActionEngine не просто виконує реакцію на окреме повідомлення, а враховує історію попередніх порушень.

Виконання дій уніфіковано через інтерфейс BaseBotAdapter, що забезпечує незалежність моделі реагування від конкретної платформи. Для Telegram реалізовано виклики на видалення повідомлень, закріплення попереджень і встановлення обмежень для користувача; для Discord — механізми масового видалення та тимчасового блокування; для Signal — операції надсилання попереджень і блокування через локальний клієнт signal-cli. Усі ці адаптери використовують однакову логіку виклику, що дозволяє ядру системи оперувати універсальним планом дій без урахування платформених відмінностей.

Додатково передбачено механізм тимчасового інтервалу між повторними санкціями, який запобігає їх надмірному накладанню на одного користувача, якщо попередня модераційна дія вже була виконана нещодавно. Це знижує ризик надмірної реакції системи та запобігає дублюванню подій у разі серійного флуду.

Окремий функціональний блок становить ViolationBlockService, який перетворює повторні модераційні події на узагальнені записи про порушення. У межах цього сервісу фіксуються ідентифікатор порушника, платформа, застосована дія, статус тікета та кількість зафіксованих порушень. Саме через цю підсистему формуються модераційні тікети, які можуть бути використані для подальшого аналізу, закриття, амністії або повторного відкриття випадку. Отже, модульна архітектура ChatShield Hub поєднує як автоматизовану детекцію загроз, так і механізми організаційної обробки інцидентів.

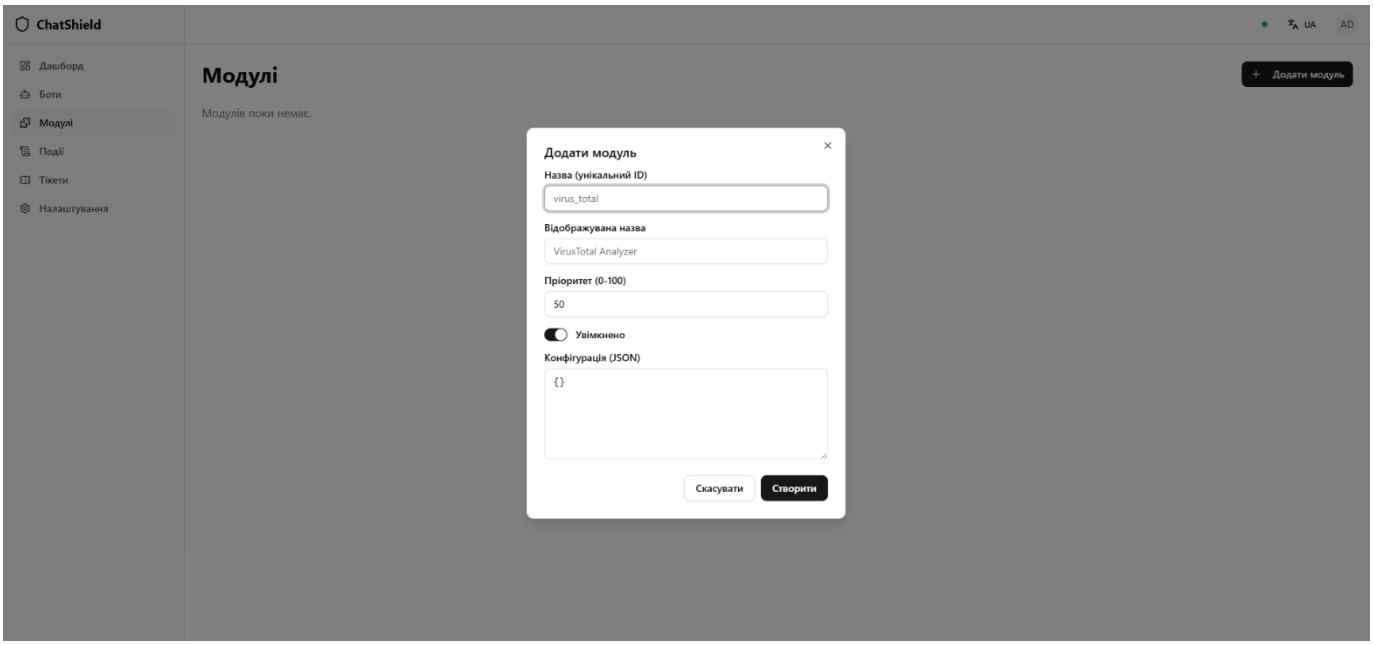

Рисунок 2.3 — Вікно створення нового модуля аналізу

Взаємодія між модулем обробки, конвеєром аналізу, виконавцем дій і подієвою шиною організована так, щоб кожен етап обробки був контрольованим і зафіксованим у системі подій. EventBus на базі Redis pub/sub приймає події від

оркестратора, забезпечує їхнє поширення до служб збереження, WebSocket-шлюзу, модуля тікетів та допоміжних підсистем, що дозволяє централізовано відстежувати стан системи в реальному часі. У підсумку модульна структура ChatShield Hub не є набором ізольованих класів, а формує цілісний механізм, у якому аналітика, реагування, аудит і відображення стану логічно пов’язані між собою. Інтерфейс створення нового аналітичного модуля, що відповідає плагінній структурі системи, показано на рис. 2.3.

2.4. Проєктування інтерфейсу адміністрування та механізмів безпеки

Інтерфейс адміністрування системи ChatShield Hub реалізовано у формі односторінкового вебзастосунку, який після початкового завантаження працює без повного перезавантаження сторінки та забезпечує динамічне оновлення окремих областей інтерфейсу. Такий підхід є доцільним для системи моніторингу інформаційної безпеки, оскільки дає змогу поєднати високу інтерактивність із безперервним відображенням поточного стану подій, ботів і модераційних рішень. Основою клієнтської частини обрано React 19 у поєднанні з TypeScript 5.9, що забезпечує компонентну організацію інтерфейсу та статичну перевірку типів, а інструмент Vite 7.3 використано для швидкого циклу розробки та оптимізованої збірки прикладної частини.

Структура клієнтського інтерфейсу побудована на бібліотеці Shadcn/ui із використанням Tailwind CSS 4, що дозволило реалізувати єдину систему візуальних компонентів, узгоджену з вимогами до адаптивності, читабельності та швидкого проєктування форм і таблиць. У межах цього підходу застосовано сторінкову організацію інтерфейсу, де кожен розділ відповідає окремій функціональній підсистемі: головна панель моніторингу, керування ботами, керування модулями, журнал подій, модераційні тікети та системні

налаштування. Така структура дозволяє логічно згрупувати функціональні можливості системи навколо чітко окреслених робочих сценаріїв адміністратора.

Головна сторінка панелі моніторингу призначена для оперативного контролю стану системи. На ній відображаються ключові показники діяльності, зокрема загальна кількість ботів, число активних ботів, кількість подій і порушень за поточний день, а також обсяг активних санкцій. Окрім цього, у центральній області розміщено графічне відображення розподілу загроз за рівнями, що дає можливість швидко оцінити зміну безпекової ситуації. Нижче подано таблицю останніх подій, у якій фіксуються тип інциденту, рівень загрози, джерело, час і деталі події. Саме ця сторінка є основною точкою спостереження за станом системи в реальному часі, оскільки її дані оновлюються внаслідок надходження подій через WebSocket-з’єднання.

Сторінка Bots використовується для керування адаптерами платформ. Вона реалізована у формі карток, кожна з яких відображає назву бота, платформу, поточний стан і доступні дії керування. Для створення або редагування бота застосовується модальне вікно з формою введення, у якій задаються назва, платформа, ознака активності, секретний токен, перелік дозволених чатів та мова відповідей. Така форма є важливою не лише як елемент керування, а і як засіб первинної перевірки налаштувань, оскільки саме через неї бот вводиться в експлуатацію та підключається до контуру обробки повідомлень.

Сторінка Modules відображає список аналітичних компонентів, зокрема SpamDetector і VirusTotalAnalyzer, та дозволяє керувати їхнім станом, конфігурацією і пріоритетом. Для створення або зміни параметрів модуля використовується окреме модальне вікно, у якому конфігурація подається у форматі JSON. Такий спосіб організації зручний для модульної архітектури, оскільки дає змогу централізовано визначати параметри обробки без необхідності втручання у вихідний код. Крім того, генерація форми з серверної схеми

забезпечує узгодженість між очікуваною структурою даних на сервері та формою їхнього введення в інтерфейсі.

Журнал подій Events реалізовано як таблицю з розширеними засобами фільтрації, пошуку та сторінкового перегляду. У ньому зберігаються всі суттєві події, пов’язані з роботою ботів, виявленням загроз, модераційними діями та змінами конфігурації. Таблиця підтримує вибірку за типом події, рівнем загрози, ботом і часовим діапазоном, а також дозволяє експортувати результати в табличні формати для подальшого аналізу. Така структура журналу є важливою для аудиту, оскільки забезпечує простежуваність усіх критичних операцій і створює основу для подальшого розслідування інцидентів.

Сторінка Tickets містить таблицю модераційних випадків, у якій відображаються відомості про порушника, платформу, характер дії, статус обробки та кількість зафіксованих повторів. Для кожного тікета передбачено можливість його закриття, амністії або повторного відкриття. Це формує окремий адміністративний контур, що доповнює автоматичну модерацію ручним управлінням у випадках, коли потрібне уточнення або перегляд прийнятого рішення. Таким чином, підсистема тікетів поєднує автоматизовану реакцію на загрозу з процедурою керованого адміністративного супроводу.

Розділ Settings/Users містить дві взаємопов’язані частини: глобальні налаштування системи та облік системних користувачів. У блоці користувачів відображається таблиця з адресою електронної пошти, роллю, датою створення, часом останнього входу і діями керування. Створення нового користувача виконується через окреме модальне вікно, де задаються адреса електронної пошти, пароль і роль. Введені дані перевіряються на відповідність формату, а також проходять серверну валідацію, що зменшує ризик помилкових або неповних записів у системі. Приклад сторінки налаштування користувачів і ролей доступу наведено на рис. 2.4.

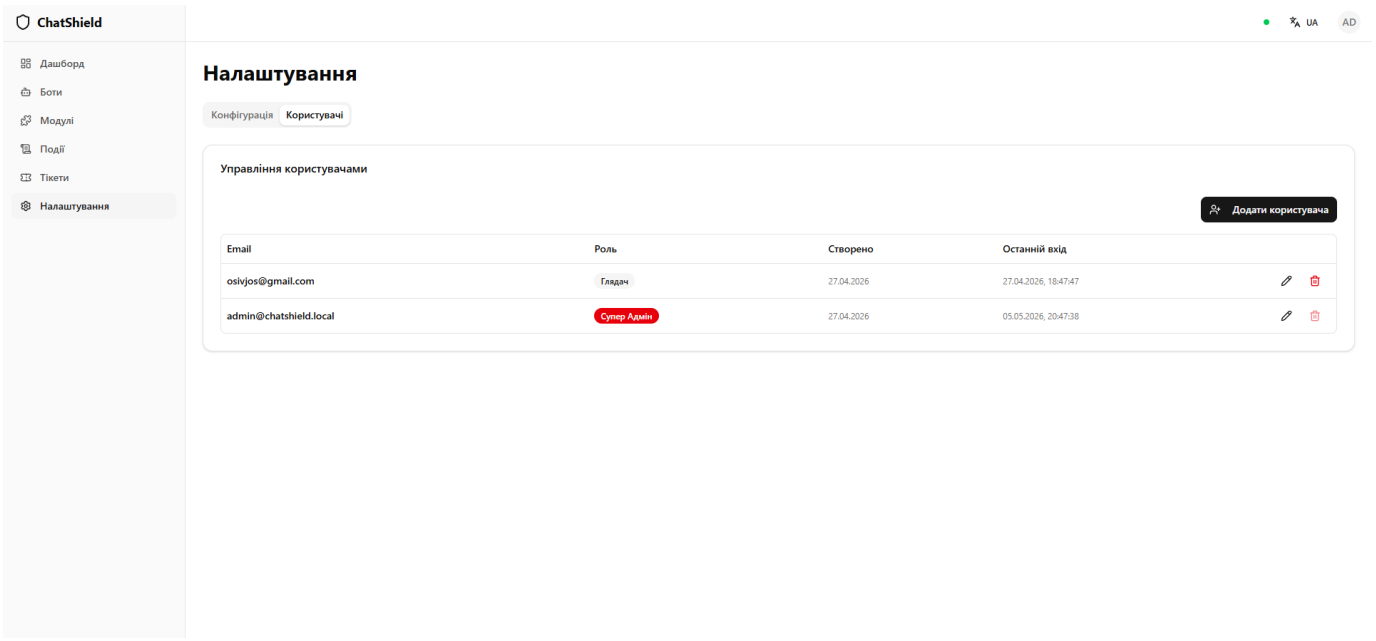

Рисунок 2.4 — Налаштування користувачів та ролей доступу

Синхронізація стану клієнтського інтерфейсу із сервером реалізована через TanStack Query, що забезпечує керування запитами, автоматичне оновлення кешу та повторне отримання даних після змін. Для подій у реальному часі використовується WebSocket-шлюз, завдяки якому головна панель і журнал подій оперативно відображають нові інциденти без дії користувача. Глобальний стан клієнтської частини підтримується через Zustand, що спрощує роботу з автентифікацією, ботами та подіями без надлишкової складності реалізації. Для багатомовного інтерфейсу передбачено локалізацію кількома мовами з можливістю перемикання у верхній частині сторінки.

Механізми безпеки інтегровано на всіх рівнях системи відповідно до підходу Zero Trust, за якого жоден компонент не вважається довіреним за замовчуванням. Автентифікація користувача здійснюється за допомогою JSON Web Token, що використовується для доступу до прикладного програмного інтерфейсу та WebSocket-каналу. Для розмежування повноважень застосовано рольовий контроль доступу з трьома рівнями: superadmin, admin і viewer. Перед

виконанням захищеної операції система перевіряє, чи відповідає роль користувача мінімальному рівню дозволу для конкретного ендпоінта; у разі невідповідності повертається код заборони доступу.

Окремо реалізовано механізми додаткового захисту, що охоплюють зберігання паролів у вигляді стійких хешів, обмеження частоти запитів до критичних ендпоінтів, шифрування чутливих конфігураційних даних у базі та ведення аудиту змін. Конфіденційні параметри ботів і модулів зберігаються в зашифрованому вигляді в PostgreSQL, а операції створення, оновлення й видалення записів фіксуються в окремому журналі аудиту. Така комбінація контролів дозволяє зменшити ризик несанкціонованого доступу, унеможливити непомітну зміну критичних налаштувань і зберегти повну простежуваність дій адміністратора.

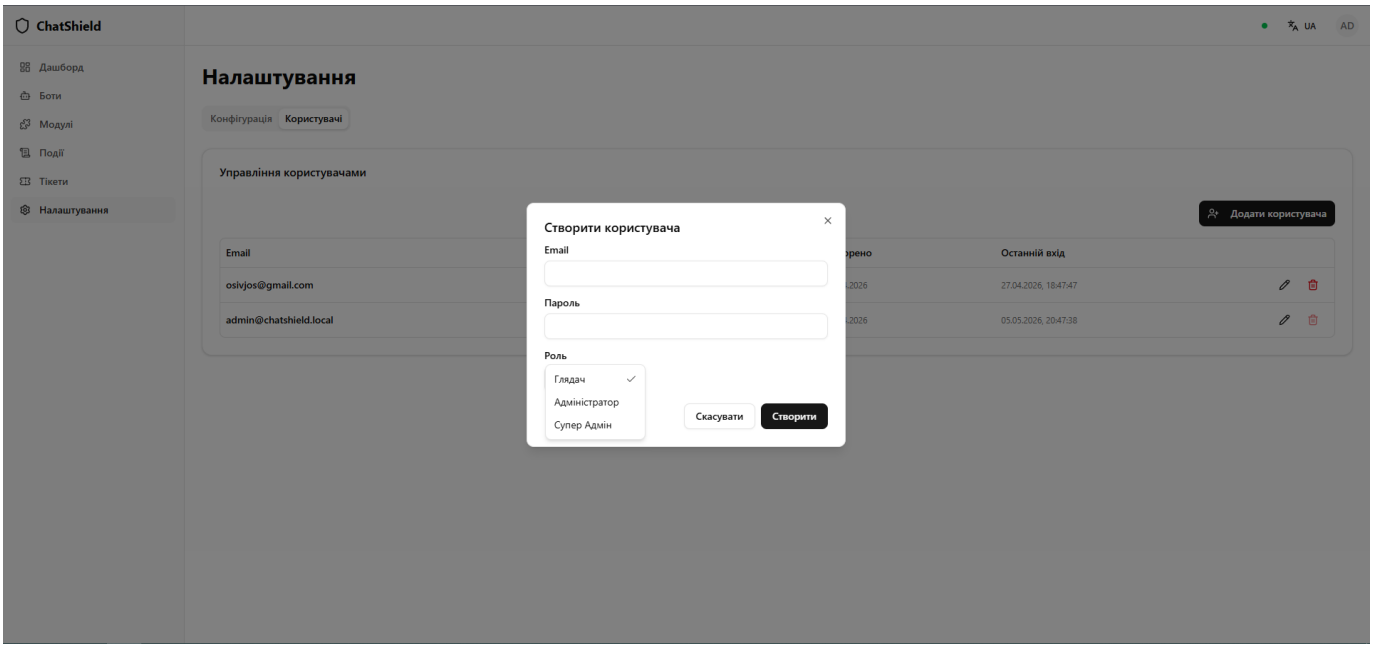

Рисунок 2.5 — Форма створення нового користувача з вибором ролі

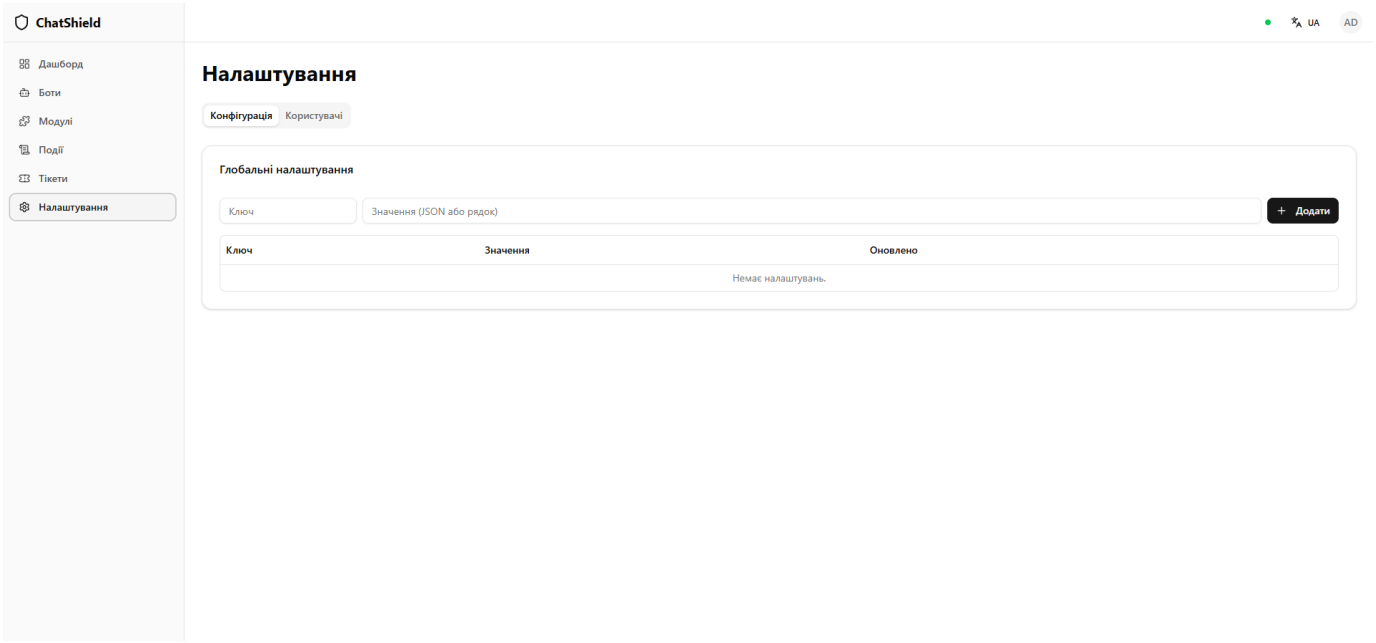

Рисунок 2.6 — Глобальні конфігураційні налаштування системи

У цілому інтерфейс адміністрування ChatShield Hub є не лише засобом візуалізації стану системи, а й повноцінним інструментом керування подієвим контуром безпеки. Його проєктування узгоджено з архітектурою серверної частини, що забезпечує цілісність обміну даними між користувачем, механізмами автентифікації, модераційною логікою та системою журналювання. Саме тому адміністративний інтерфейс виконує в системі подвійну функцію: забезпечує зручну експлуатацію та водночас виступає важливим елементом захисту й контролю безпечної роботи чат-комунікацій. Окремий приклад реалізації адміністративної дії створення користувача з вибором ролі наведено на рис. 2.5.

Приклад сторінки глобальних конфігураційних налаштувань системи подано на рис. 2.6.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

3.1. Вибір технологій та інструментальних засобів реалізації

Реалізація технологічного стеку системи ChatShield Hub була підпорядкована вимогам до асинхронної обробки подій, високої пропускної здатності, надійного зберігання конфігураційних даних і відтворюваного розгортання повного комплексу компонентів. У межах практичної частини це означало необхідність обрати такі програмні засоби, які забезпечують подієву архітектуру, вебінтерфейс адміністратора, взаємодію в реальному часі через WebSocket, а також механізми автентифікації, журналювання, обмеження частоти запитів і тестування на всіх рівнях системи.

Для серверної частини обрано Python 3.11 у поєднанні з FastAPI. Така зв’язка є доцільною для асинхронних застосунків, оскільки поєднує високу ефективність обробки паралельних запитів із вбудованою підтримкою життєвого циклу застосунку, залежностей, WebSocket-з’єднань і автоматичного формування документації програмного інтерфейсу. Саме FastAPI став основою для реалізації OrchestratorService, EventBus, REST API та шлюзу обміну подіями в реальному часі, а також для ініціалізації сервісів під час запуску застосунку. Для доступу до даних використано SQLAlchemy 2.0 в асинхронному режимі разом з Alembic, що забезпечило керовану роботу зі схемою PostgreSQL і послідовне оновлення структури бази даних без порушення цілісності прикладної логіки.

Перевірка та нормалізація структур даних реалізовані на базі Pydantic 2, який використовується для опису моделей запитів, відповідей і динамічних схем конфігурації адаптерів та модулів. Це рішення є особливо важливим у контексті вебінтерфейсу, оскільки форма створення бота або модуля формується на основі

JSON-схеми, що надходить із серверної частини. Автентифікацію та керування сесіями реалізовано за допомогою PyJWT, а для хешування паролів застосовано bcrypt, що відповідає вимогам до захисту облікових даних і підтримки безпечної моделі доступу до адміністративних функцій.

Механізм обмеження частоти запитів побудовано на основі slowapi 0.1.9, який дає змогу окремо контролювати операції входу та загальний трафік з однієї IP-адреси. Для взаємодії із зовнішніми сервісами перевірки посилань обрано httpx, оскільки цей асинхронний HTTP-клієнт підтримує повторні спроби запитів і добре підходить для роботи з VirusTotal API v3. У сукупності це забезпечило поєднання швидкодії локальної обробки з контрольованим доступом до зовнішніх джерел репутаційних даних.

Адаптерний рівень системи спроєктовано з урахуванням особливостей кожної платформи. Для Telegram використано aiogram v3, який ефективно працює з Bot API в асинхронному режимі; для Discord застосовано discord.py 2.7, орієнтований на постійне з’єднання зі шлюзом подій; для Signal інтегровано signal-cli через JSON-RPC або командний інтерфейс. Такий підхід дозволив реалізувати уніфікований інтерфейс адаптерів для різних комунікаційних середовищ і зберегти спільну модель обробки повідомлень без прив’язки ядра системи до конкретної платформи.

Як основну базу даних обрано PostgreSQL 16, оскільки ця система поєднує транзакційну надійність, підтримку JSONB і придатність для зберігання подій, ботів, користувачів, тікетів та аудиторських записів. Для динамічного обміну подіями та збереження допоміжних лічильників використано Redis 7, який забезпечує швидке поширення повідомлень у реальному часі та підтримує тимчасові структури даних, зокрема блокування порушень. Конфігураційні дані, токени та параметри модулів зберігаються в зашифрованому вигляді, що підсилює загальний рівень захисту системи.

Фронтенд реалізовано на базі React 19, TypeScript 5.9 і Vite 7.3. Така комбінація забезпечує сучасну компонентну модель, високу швидкість збірки та зручну інтеграцію із серверним API. Для маршрутизації застосовано React Router, для керування запитами і кешем — TanStack Query, а для локального стану інтерфейсу — Zustand 5.0. Візуалізацію метрик і графіків реалізовано засобами Recharts 3.7, що дало змогу відображати статистику подій, розподіл загроз і поточний стан ботів на інформаційній панелі.

Окрему увагу приділено засобам побудови інтерфейсу. Для цього застосовано Shadcn/ui та Tailwind CSS 4, які забезпечують використання готових компонентів із можливістю гнучкого налаштування, доступність для клавіатурного керування та компактність стилів. Зокрема, у проєкті використовуються типізовані компоненти на зразок таблиць, модальних вікон і форм динамічної конфігурації, що спрощує побудову інтерфейсу адміністратора без дублювання логіки відображення. Інтерфейс побудовано як односторінковий застосунок, що відповідає вимогам до швидкого оновлення даних без перезавантаження сторінок.

Для контейнеризації та розгортання обрано Docker і Docker Compose. Таке рішення дає змогу запускати весь стек однією командою та відтворювати робоче середовище без ручного налаштування залежностей. У проєктній конфігурації передбачено окремі контейнери для PostgreSQL, Redis, signal-cli, backend, frontend та Nginx-проксі, а також перевірки працездатності для контролю стану сервісів. Додатково використовується файл .env для зберігання секретів, JSON-логування з кореляційними ідентифікаторами та підготовка до автоматизованого розгортання через pyproject.toml, package.json і шаблони тестових сценаріїв.

Приклад сторінки керування ботами зі статусами їхньої роботи наведено на рис. 3.1.

Таблиця 3.1 — Порівняння основних технологій з їх функціональним призначенням у системі

Назва	**Версія**	**Для чого**	**Як працює**	**Чому саме ця**
React	19	Побудова інтерфейсу	Формує сторінки як сукупність компонентів, оновлює їх без перезавантаження	Забезпечує зрілу компонентну модель та широкі можливості інтеграції з екосистемою веброзробки
TypeScript	5.9	Типізація коду	Додає статичну перевірку типів і підказки в редакторі	Зменшує ризик помилок під час взаємодії з API та складними структурами даних
Vite	7.3	Збірка проєкту	Підтримує швидке оновлення під час розробки та оптимізовану робочу збірку	Доцільний для сучасних frontend-застосунків із вимогами до оперативної розробки
Shadcn/ui	4.7.0	Готові інтерфейсні блоки	Надає компоненти, які можна адаптувати до потреб проєкту	Спрощує створення форм, таблиць і модальних вікон без надмірної складності

Назва	Версія	Для чого	Як працює	Чому саме ця
Tailwind CSS	4	Стилізація	Використовує утилітарні класи для швидкого компонування інтерфейсу	Забезпечує гнучке оформлення та контроль над розміром стилів
Zustand	5.0	Локальний стан	Дає змогу зберігати стан інтерфейсу у легкому клієнтському сховищі	Підходить для простих і керованих сценаріїв стану в адміністраторській панелі
TanStack Query	5.90	Запити до сервера	Автоматизує завантаження, кешування та оновлення даних	Забезпечує зручну синхронізацію клієнта з API та підтримку актуальності даних
Recharts	3.7	Візуалізація даних	Будує адаптивні діаграми на основі масивів даних	Доцільний для відображення метрик подій і загроз у панелі моніторингу
Playwright	1.52	Тестування інтерфейсу	Автоматизує перевірку сценаріїв у різних браузерах	Дає змогу перевіряти критичні користувацькі сценарії в умовах,

Назва	Версія	Для чого	Як працює	Чому саме ця
				близьких до реальних

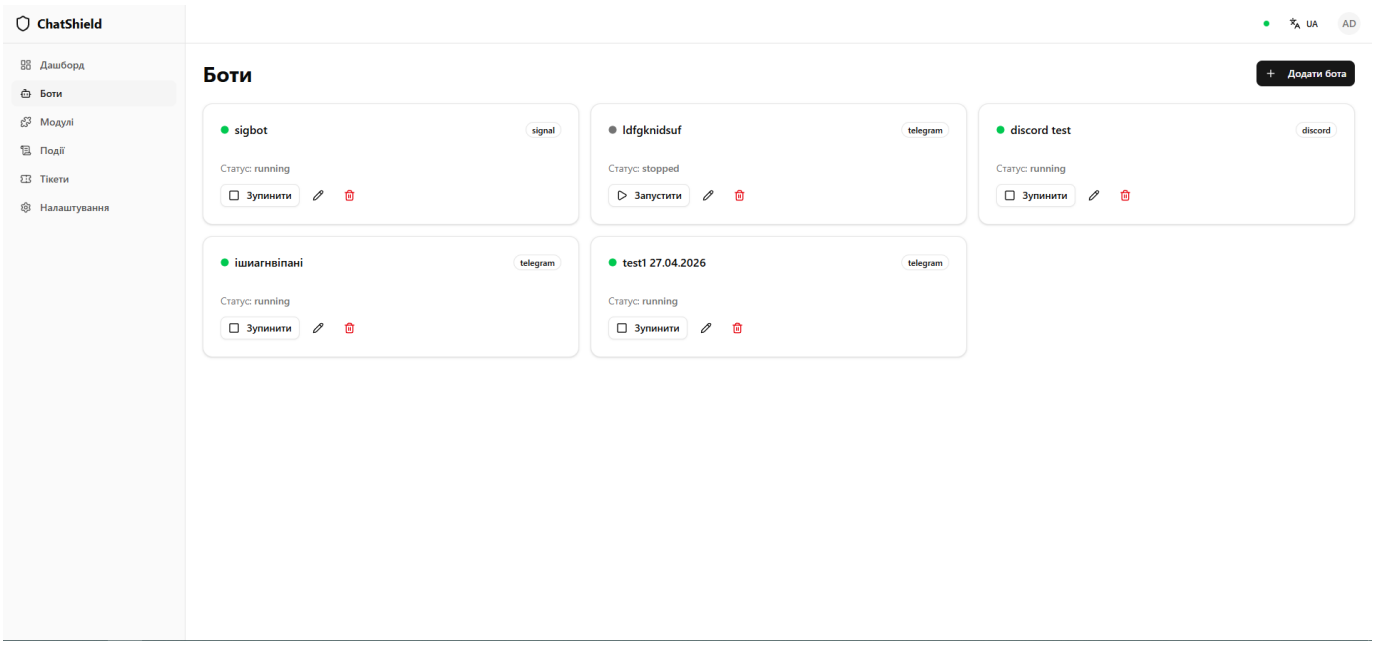

Рисунок 3.1 — Сторінка керування ботами зі статусами роботи

Реалізацію багатомовності інтерфейсу через меню вибору мови показано на рис. 3.2.

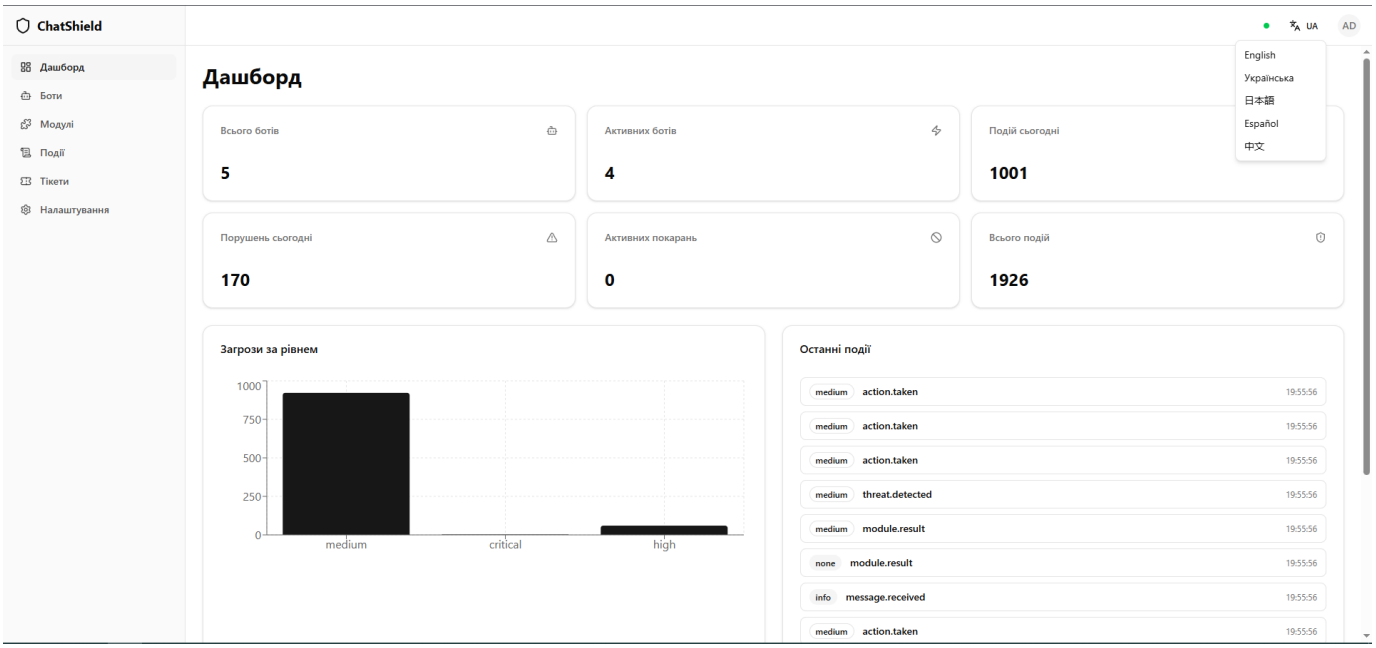

Рисунок 3.2 — Меню вибору мови інтерфейсу

Вибір технологій був підтверджений результатами тестування. Для серверної частини застосовано pytest, а для інтерфейсних сценаріїв — Playwright. Перевірка якості детекції, часових затримок і стійкості механізмів реагування виконувалася за допомогою набору експериментальних тестів. Такий підхід дав змогу оцінити не лише працездатність окремих компонентів, а й стабільність їхньої взаємодії в умовах реального навантаження. За підсумками випробувань система продемонструвала придатність до практичного розгортання та відповідність заданим архітектурним вимогам. Систематизоване порівняння основних технологій, їхнього призначення та ролі в реалізації ChatShield Hub наведено в табл. 3.1.

3.2. Реалізація ядра системи, адаптерів та аналітичних модулів

Реалізація ядра системи ChatShield Hub ґрунтується на сервісі OrchestratorService, який під час запуску FastAPI формує робоче середовище

застосунку та ініціалізує основні компоненти керування подіями, адаптерами і аналітичними модулями. Саме на цьому рівні створюються EventBus, BotRegistry, ModuleManager, AnalysisPipeline та ActionEngine, після чого підключаються сервіси збереження подій і передавання їх до вебінтерфейсу. Під час ініціалізації застосунку також створюються таблиці бази даних, перевіряється наявність адміністративного користувача, відкривається з'єднання з Redis, запускається оркестратор і відновлюються активні боти, що забезпечує безперервність моніторингу після перезапуску системи.

Ключовим механізмом взаємодії між компонентами є шина подій EventBus, яка забезпечує асинхронний обмін повідомленнями типу message.received, threat.detected, module.result і action.taken. Такий підхід усуває прямі залежності між адаптерами, аналізаторами та сервісами реагування, оскільки кожен компонент працює не через безпосередні виклики, а через уніфіковані події. Усі значущі результати надалі зберігаються в таблиці events і паралельно передаються до клієнтської частини через WebSocket, завдяки чому інтерфейс системи фактично синхронізується із серверною логікою в реальному часі.

Для взаємодії з різними платформами у системі реалізовано спільний контракт BaseBotAdapter, який визначає запуск, зупинку, виконання дій, надсилання повідомлень і редагування службових сповіщень. Усі вхідні повідомлення нормалізуються до структури MessageContext, у якій фіксуються ідентифікатор бота, платформа, чат, користувач, текст повідомлення, ідентифікатор повідомлення, час та службові метадані. Завдяки цьому модулі аналізу не залежать від конкретної платформи й працюють з єдиним форматом даних, що суттєво спрощує розширення системи та підключення нових джерел повідомлень.

TelegramAdapter реалізовано на основі aiogram v3 і використано для отримання повідомлень у режимі постійного опитування сервера Telegram з мінімальною затримкою. Після отримання оновлення адаптер формує контекст

повідомлення і передає його до оркестратора, а під час модерації може видаляти повідомлення, надсилати попередження, обмежувати користувача або блокувати його. Для повноцінної роботи бота необхідні права адміністратора та вимкнений режим конфіденційності, що було враховано під час польового тестування. Аналогічно, DiscordAdapter функціонує через постійне з'єднання із сервером Discord, а SignalAdapter інтегровано через signal-cli у форматі окремого допоміжного контейнера, що дає змогу підтримувати різні канали комунікації в межах єдиного ядра системи.

Керування адаптерами здійснює BotRegistry, який пов'язує запис про бота з бази даних із його виконуваним екземпляром у пам'яті. Це означає, що адміністратор керує не лише записом у базі, а й фактичним станом роботи адаптера: запуском, зупинкою та відстеженням помилок. На практиці такий підхід формує єдиний реєстр ботів, доступний через вебінтерфейс, де кожен бот має власну конфігурацію, статус і набір дозволених дій.

Аналітична частина системи побудована на контракті BaseAnalyzer, який визначає назву модуля, пріоритет, метод analyze(), метод configure() та схему конфігурації. Така структура забезпечує плагінність і дає змогу підключати нові аналітичні модулі без зміни загального механізму обробки. У межах реалізації застосовуються два основні модулі: SpamDetector і VirusTotalAnalyzer, які доповнюють один одного за швидкістю та глибиною аналізу.

SpamDetector реалізує евристичне виявлення спаму на основі сукупності ознак, зокрема частоти повідомлень, їхньої довжини, повторюваності тексту та лексичної різноманітності. Оцінювання виконується як зважена сума балів, після чого результат порівнюється з пороговим значенням, а модуль формує результат аналізу із рівнем загрози, рівнем упевненості, причиною спрацювання та рекомендованою дією. Такий підхід дає змогу виявляти не лише масовий спам, а й короткі шаблонні повідомлення, що повторюються в межах короткого проміжку часу.

VirusTotalAnalyzer відповідає за перевірку URL-адрес, виділених із тексту повідомлення. Якщо посилань немає, модуль повертає нейтральний результат, а за наявності URL виконує запит до VirusTotal API v3 і оцінює ресурс як безпечний, підозрілий або шкідливий. Урахування зовнішніх обмежень і повторних запитів дає змогу використовувати цей модуль як точніший, хоча й повільніший механізм зовнішньої репутаційної перевірки.

Об’єднання результатів модулів виконує AnalysisPipeline, який послідовно запускає активні аналізатори за їхнім пріоритетом і формує набір результатів аналізу для подальшого агрегування. Далі ActionEngine перетворює отримані результати на єдиний план реагування, визначаючи найсуворішу дію відповідно до рівня загрози та історії порушень. На нижчому рівні цей план виконується через відповідний адаптер, а успішне виконання супроводжується подією action.taken.

Для модераційних сценаріїв у системі передбачено ViolationBlockService і TicketService. Перший агрегує повторні порушення в один редагований блок службового повідомлення, що зменшує засмічення чату та робить реакцію модератора більш наочною. Другий автоматично створює або оновлює тікети, у яких фіксуються порушник, платформа, тип дії, рівень загрози та історія інциденту. Саме тому платформа не обмежується разовим видаленням повідомлення, а формує повноцінний контур обліку, аналізу та розгляду порушень.

Практичну реалізацію журналу подій, у якому відображаються результати роботи системи, наведено на рис. 3.3.

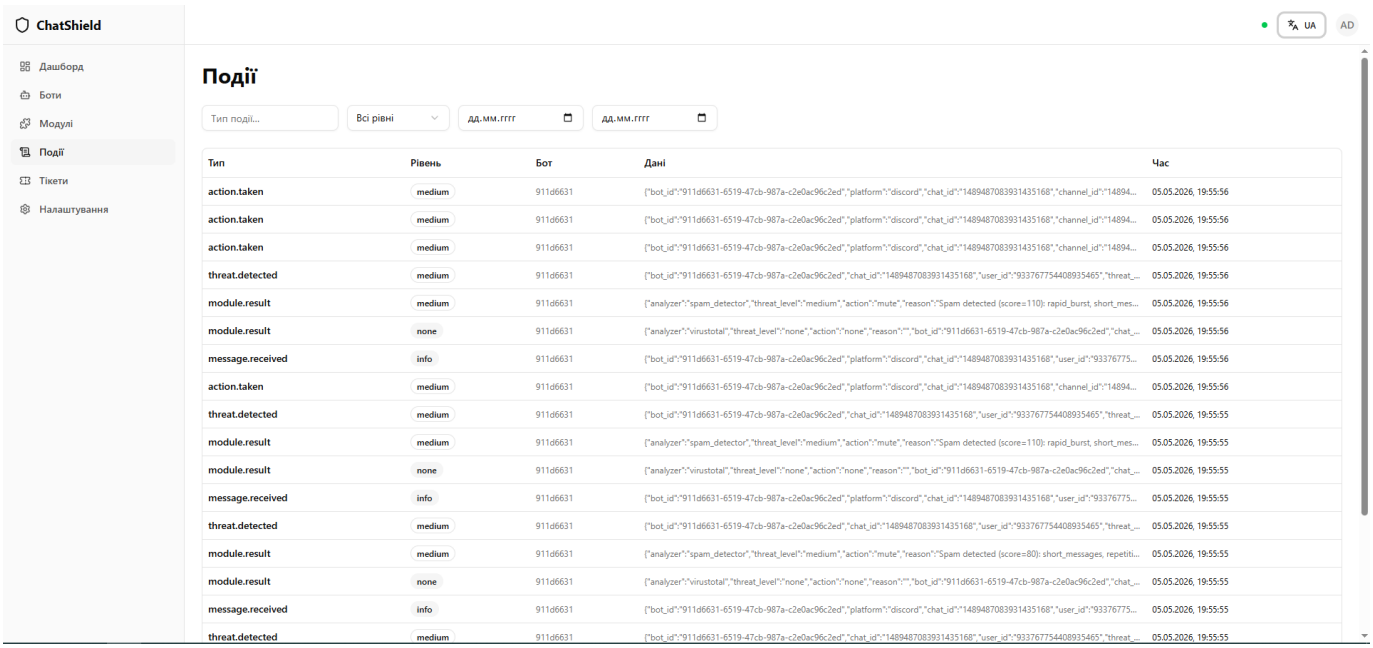

Рисунок 3.3 — Сторінка журналу подій системи

Приклад сторінки тікетів з активними порушеннями та доступними діями модератора подано на рис. 3.4.

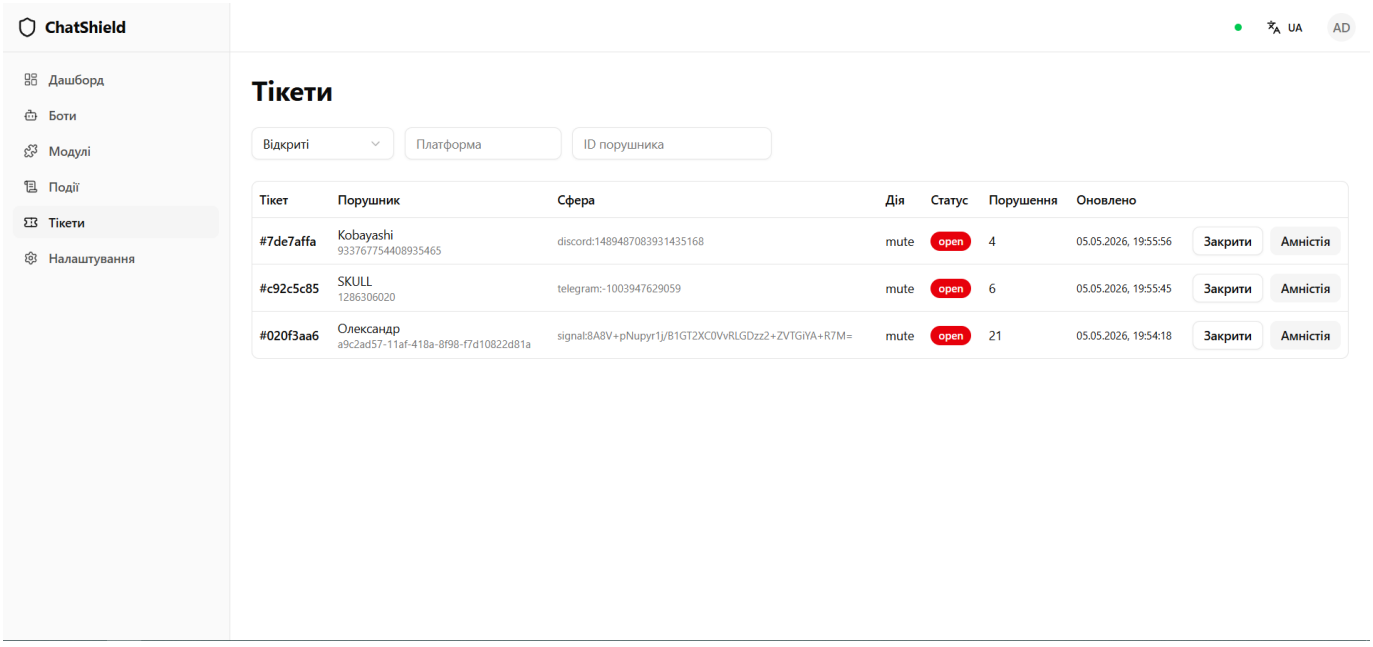

Рисунок 3.4 — Сторінка “Тікети” з активними порушеннями та діями

У підсумку реалізація ядра, адаптерів і аналітичних модулів утворює єдиний цикл обробки повідомлень: повідомлення надходять через адаптер, перетворюються на уніфікований контекст, аналізуються модулями аналізу, після чого формується план реагування, який виконується через відповідний адаптер і фіксується в базі даних та вебінтерфейсі. Саме така архітектура забезпечує слабке зв’язування компонентів, можливість розширення системи та можливість практичного використання ChatShield Hub як адаптивної системи контролю інформаційної безпеки чат-комунікацій.

3.3. Реалізація вебінтерфейсу та механізмів реагування

Реалізація вебінтерфейсу ChatShield Hub ґрунтується на односторінковому застосунку React із маршрутизацією, централізованим керуванням станом і підключенням до API та WebSocket, що забезпечує роботу адміністративної панелі в режимі майже реального часу. Після входу користувач отримує доступ до сторінок Dashboard, Bots, Modules, Events, Tickets і Settings, а вся взаємодія із сервером здійснюється через захищений клієнт API з автоматичною підстановкою JWT і перенаправленням на сторінку входу у разі втрати авторизації. Така організація інтерфейсу забезпечує не лише зручність роботи, а й узгодженість клієнтської логіки з механізмами контролю доступу, реалізованими на серверному боці.

Сторінка головної панелі керування відображає зведені показники роботи системи, зокрема кількість ботів, кількість активних ботів, події за сьогодні, порушення, активні покарання, загальну кількість подій, а також графік розподілу загроз і стрічку останніх подій. Саме ця сторінка виконує роль оперативного центру моніторингу, оскільки дає адміністратору змогу швидко оцінити стан системи та характер поточного навантаження без переходу до журналу подій. Оновлення панелі моніторингу відбувається через WebSocket-канал wsevents,

тому зміни відображаються без ручного перезавантаження сторінки. Після накопичення нових подій змінюється як абсолютне значення показників, так і візуальний розподіл загроз, що підтверджує динамічний характер відображення стану системи.

Сторінка Bots реалізує повний цикл керування адаптерами: створення бота, запуск, зупинку, редагування та видалення. Для створення і редагування використовується динамічна форма, яка будується на основі JSON-схеми, отриманої від сервера, тому набір полів залежить від платформи і не потребує жорсткого кодування в клієнтській частині. Такий підхід є особливо важливим для ChatShield Hub, оскільки Telegram, Discord і Signal мають різні параметри конфігурації, але в інтерфейсі вони подаються через єдину модель введення. У результаті адміністратор працює з уніфікованою формою, а різниця між платформами відображається лише на рівні схеми конфігурації.

Сторінка Modules призначена для керування аналітичними модулями, які відповідають за виявлення спаму та перевірку URL-адрес. Адміністратор може вмикати або вимикати модулі, змінювати їхній пріоритет і редагувати конфігурацію. Це дає змогу адаптувати роботу системи до різних сценаріїв модерації без зміни серверної логіки, а плагінна модель підключення модулів підтримує розширюваність системи та поступове доповнення її новими засобами аналізу. Така організація є доцільною для багатоканального середовища, де вимоги до швидкості реакції та глибини перевірки можуть відрізнятися залежно від платформи або політики безпеки. Приклад відображення головної панелі керування англійською мовою наведено на рис. 3.5.

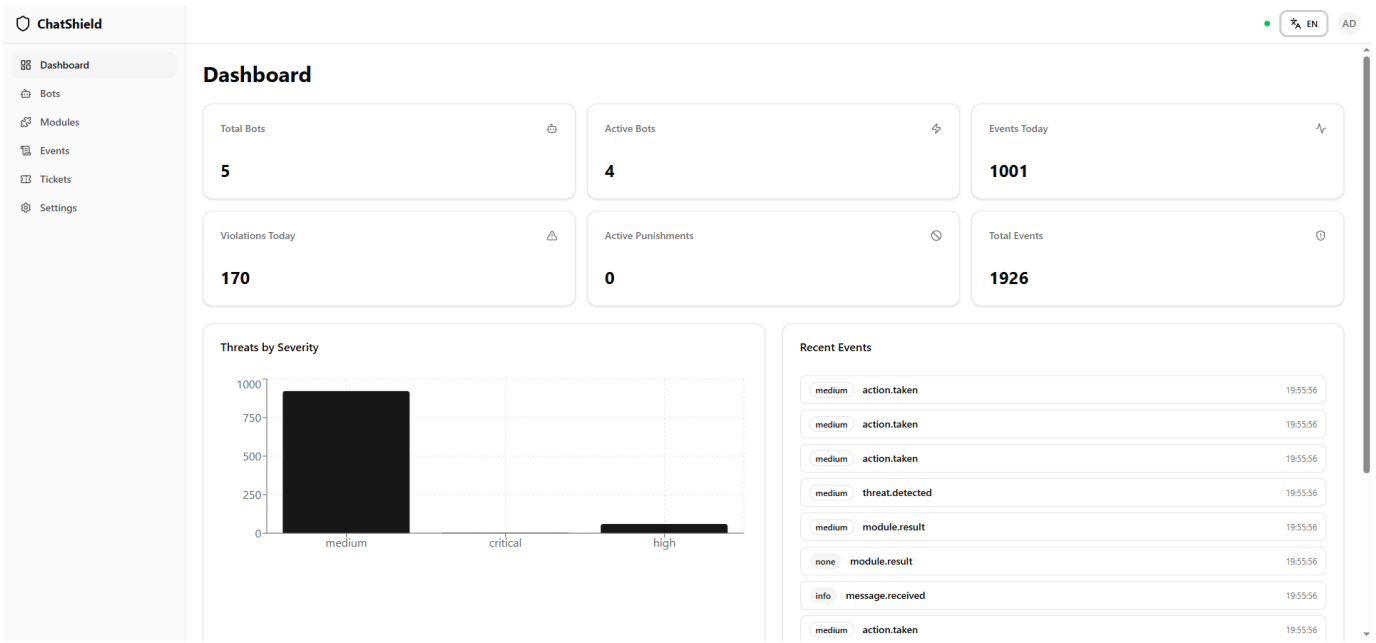

Рисунок 3.5 — Dashboard англійською мовою

Журнал подій на сторінці Events є одним із центральних елементів практичної реалізації, оскільки саме тут відображаються події message.received, module.result, threat.detected і action.taken у вигляді таблиці з фільтрами, рівнями небезпеки, ботами, деталями та часом виникнення. Ця сторінка відтворює серверну подієву модель у зручному для аналізу вигляді та дозволяє швидко відстежувати ланцюг обробки кожного інциденту. Завдяки цьому адміністратор може не лише бачити факт спрацювання захисту, а й аналізувати його причину та послідовність дій системи, що є важливим для подальшого аудиту та налаштування політик реагування.

Сторінка Tickets реалізує процес модерації, у якому кожне суттєве порушення оформлюється як окремий тікет із зазначенням порушника, платформи, типу дії, статусу, кількості порушень і доступних керувальних дій. Передбачено закриття тікета, амністію та відновлення розгляду, причому доступ до цих дій залежить від ролі користувача. Це переводить реагування на інциденти з рівня окремого повідомлення в рівень обліковуваного управлінського процесу,

у межах якого зберігається історія рішень і повторюваність порушень. Приклад Telegram-сповіщення ChatShield про виявлення спаму та видалення повідомлень показано на рис. 3.6.

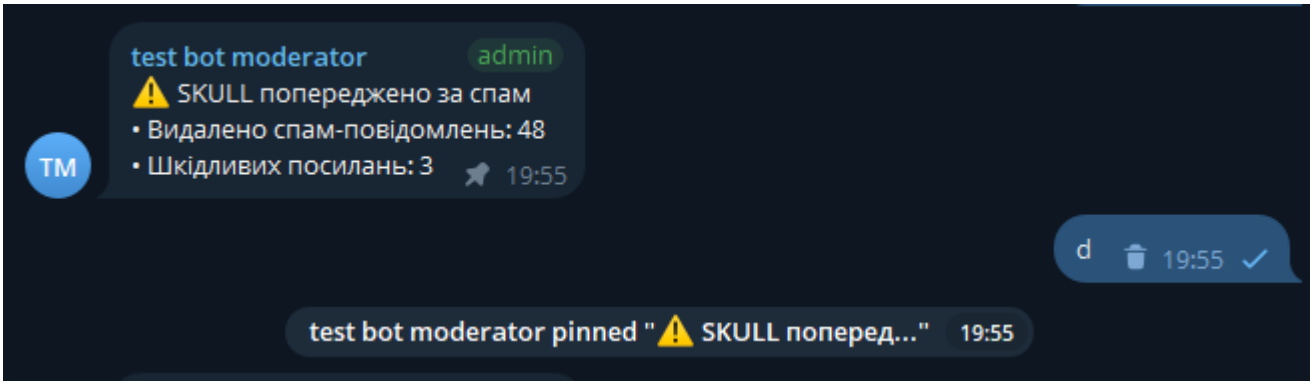

Рисунок 3.6 — Telegram-сповіщення від ChatShield про спам та видалені повідомлення

Сторінка Settings використовується для керування системними параметрами і користувачами, зокрема для створення нового користувача, перегляду таблиці користувачів, їхніх ролей, дати створення та останнього входу. Вебінтерфейс доповнений багатомовністю, тому окремі елементи можуть відображатися українською та іншими мовами інтерфейсу залежно від вибору користувача. Це підвищує придатність системи до роботи в різних мовних середовищах і спрощує експлуатацію, особливо в умовах багатонаціональних або розподілених команд адміністрування.

Механізми реагування у вебінтерфейсі відображають не лише поточний стан об’єктів, а й результат дії серверної логіки: запуск або зупинка бота, зміна конфігурації модуля, обробка тікета, оновлення показників панелі моніторингу і поява нових подій у журналі. Технічно це забезпечується зв’язкою API, React Query і WebSocket, де кожне серверне повідомлення оновлює кеш сторінок панелі керування, Events і Tickets. У результаті адміністратор працює не з окремими

формами, а з єдиним керованим середовищем модерації та моніторингу, у якому кожна дія має перевірюваний і відображуваний у реальному часі наслідок.

3.4. Комплексне тестування системи

Комплексне тестування системи ChatShield Hub було спрямоване на перевірку працездатності всіх рівнів програмного комплексу, зокрема серверної логіки, подієвого контуру, адаптерів платформ, аналітичних модулів, механізмів реагування, вебінтерфейсу та засобів авторизації. У межах цього етапу оцінювалися як функціональні сценарії, так і часові характеристики обробки повідомлень, коректність оновлення інтерфейсу в реальному часі, реакція системи на порушення політик та стійкість до повторюваних інцидентів. Тестування проводилося в кілька послідовних етапів, що дало змогу перевірити систему не лише в ізольованому середовищі, а й у складі повного робочого контуру.

На першому етапі виконувалося регресійне тестування серверної частини. За результатами цього набору було успішно пройдено 139 тестів, які охоплювали оркестрацію подій, збереження журналу, маршрутизацію повідомлень, модераційні дії, автентифікацію та рольовий доступ. Окремо для виправлень, пов’язаних із Dashboard API, було виконано ще 40 тестів, які підтвердили коректність обчислення статистики, вибірки подій і формування актуального зведення щодо стану системи. Таке покриття дало змогу переконатися, що основні серверні механізми працюють узгоджено і не порушують цілісність прикладного контуру.

Другий блок тестування стосувався перевірки аналітичних модулів. Для модуля SpamDetector було використано синтетичні сценарії зі спамом, короткими повторюваними повідомленнями та burst-активністю, у яких система продемонструвала точність, повноту і F1 на рівні 1.00, а також відсутність

хибнопозитивних спрацьовувань у базовому наборі. У полігоні тестів було зафіксовано, що при перевищенні порога за частотою та схожістю повідомлень система переходить до реакції у вигляді попередження або видалення повідомлення, а при повторному порушенні ескалує дію до mute. Це підтвердило, що реалізований механізм евристичного виявлення спаму придатний для практичного використання у чат-комунікаціях.

Окремо перевірявся модуль VirusTotalAnalyzer, який виконує зовнішню репутаційну оцінку URL-адрес. Під час тестів було використано контрольований набір безпечних, підозрілих і шкідливих посилань, що дало змогу перевірити правильність класифікації та реакції системи на різні рівні загрози. Для таких сценаріїв було встановлено, що час виявлення становить приблизно 5–12 секунд з урахуванням зовнішнього API та повторних спроб, а дія реагування після отримання результату виконується приблизно за 0.5 секунди. У підсумку це підтвердило придатність модуля для виявлення мережевих загроз у режимі модерації з відкладеною, але більш достовірною перевіркою.

Третій напрям тестування охоплював інтерфейс користувача та механізми реального часу. Для фронтенд-частини було виконано 11 автоматизованих інтерфейсних тестів засобами Playwright, які перевіряли логін, перехід між сторінками, доступ до панелі керування, перегляд журналу подій, керування ботами, роботу з модулями та модераційними тікетами. Тестування підтвердило коректність маршрутизації, підвантаження даних через API та відображення результатів через WebSocket без необхідності ручного оновлення сторінки. Окремо було відзначено, що при надходженні нових подій оновлюються показники інформаційної панелі, зокрема загальна кількість подій, активність ботів і розподіл загроз.

На рівні польового сценарію механізм перевірявся в тестовій Telegram-групі, де бот працював із правами адміністратора та мав можливість видаляти повідомлення і накладати обмеження на користувачів. У цьому середовищі

модуль виявляв burst-спам, фіксував повтори повідомлень і обробляв повідомлення з підозрілими посиланнями, після чого виконував попередження, видалення або ескалацію дії залежно від типу інциденту. Зафіксований перехід від базового стану з 133 подіями до фінального стану з 434 подіями підтвердив, що події коректно зберігаються у базі, транслюються через подієвий канал і відображаються у вебінтерфейсі. Крім того, у тестах було підтверджено, що підсистема здатна утримувати живу стрічку подій і змінювати показники панелі майже в реальному часі.

Важливим аспектом комплексного тестування стала перевірка рольового доступу та захисту адміністративних дій. Під час тестів було підтверджено, що користувачі без належних прав не можуть виконувати адміністративні операції, а маршрути захищені механізмом JWT та RBAC. Також було перевірено журналювання входів, збереження аудиту змін і обмеження частоти запитів, що є важливим для практичної експлуатації системи в умовах багатокористувацького середовища. У результаті ці перевірки засвідчили не лише працездатність функцій, а й коректність захисного шару системи.

Отримані результати дозволили зробити висновок, що ChatShield Hub проходить комплексну перевірку як цілісний програмний комплекс, а не як набір окремих функцій. Поєднання регресійних тестів, інтерфейсної автоматизації, оцінювання часу спрацювання та польового прогону показало, що система здатна стабільно працювати під навантаженням, виявляти типові загрози та відображати результати модерації в реальному часі. Саме це створює практичну основу для наступного етапу оцінювання ефективності системи.

3.5. Аналіз результатів та оцінювання практичної ефективності

Оцінювання практичної ефективності ChatShield Hub виконувалося на основі сукупності регресійних тестів, синтетичних експериментів, інтерфейсних

перевірок та польового прогону в Telegram-середовищі. Такий підхід дав змогу оцінити систему не лише за формальним фактом працездатності, а й за вимірюваними характеристиками, серед яких точність детекції, час реагування, коректність ескалації, стабільність інтерфейсу та здатність до відображення подій у режимі, з мінімальною затримкою. У результаті було отримано набір показників, який обґрунтовує практичну готовність системи до модерації чат-комунікацій.

Насамперед слід відзначити якість роботи аналітичних модулів. Для вбудованого детектора спаму на контрольному синтетичному наборі було зафіксовано Precision 1.00, Recall 1.00, F1 1.00 та FPR 0.00. Це означає, що в межах підготовленого набору прикладів система не пропустила жодного типового спам-сценарію та не дала хибнопозитивних спрацьовувань. Для перевірки URL-адрес за допомогою VirusTotalAnalyzer також отримано Precision 1.00, Recall 1.00 і F1 1.00 на контрольованому наборі з безпечними та шкідливими посиланнями. У практичному контексті це свідчить про те, що архітектура придатна як для сценаріїв локального фільтрування, так і для сценаріїв підвищеної обережності, де зовнішня репутаційна перевірка є обов’язковою.

Часові характеристики підтвердили, що платформа працює в межах прийнятного режиму для адміністративного моніторингу. Для burst-спаму час виявлення становив приблизно 1.5–2 секунди, що відповідає швидкій реакції на масове надсилання повідомлень. Для URL-загроз, які перевіряються через зовнішній сервіс, час виявлення становив 5–12 секунд через особливості API-обробки та повторних спроб. Час виконання модераційної дії після прийняття рішення становив близько 0.5 секунди, що підтверджує швидку реакцію механізму action.taken після завершення аналізу. Додатково зафіксовано, що оновлення панелі керування через WebSocket має орієнтовну затримку близько 1 секунди, тобто зміни в стані системи відображаються майже безперервно.

Окремий показник практичної ефективності пов’язаний із динамікою подій у системі. У тестах без активного втручання та у сценаріях реального середовища кількість подій зросла з 133 до 434, що дало приріст 301 подію. Така зміна підтверджує, що платформа не лише детектує інциденти, а й коректно реєструє їх у базі даних, транслює у вебінтерфейсі та використовує для формування зведеної статистики. На головній панелі керування це супроводжувалося оновленням сумарних показників, зокрема кількості подій за день та загальної активності ботів. Отже, подієвий контур працює як повноцінний механізм моніторингу, а не як ізольована логіка без зворотного зв’язку.

Ефективність модераційної логіки підтверджується й результатами сценаріїв ескалації. У тестовому наборі було зафіксовано 33 коректні випадки вибору рівня реакції, а також 100-відсоткове блокування 33 несанкціонованих спроб у межах перевірки RBAC. Це означає, що механізм не лише виявляє загрози, а й правильно співвідносить їх із відповідними діями, а доступ до цих дій чітко обмежено ролями. Для практичної експлуатації це принципово важливо, оскільки помилка в механізмі доступу або неправильна ескалація може звести нанівець ефект від якісної детекції.

Порівняння з альтернативними підходами, наведене у матеріалах проєкту, показує, що ChatShield Hub не орієнтувався на абстрактне максимізування окремої метрики, а прагнув до збалансованого практичного результату. Наприклад, у полігоні перевірки для спам-сценаріїв було отримано стабільну реакцію на burst-повідомлення, повторення та низьку варіативність тексту, а для URL-аналізу система зберігала високу якість розпізнавання навіть за наявності затримки з боку зовнішнього API. Це підтверджує, що обрана архітектура добре пристосована до реального середовища, де різні типи загроз мають різну природу та різний допустимий час реагування.

Практична цінність системи підтверджується також результатами інтерфейсного тестування. Було виконано 11 сценаріїв E2E, які охоплювали вхід

у систему, навігацію, відображення панелі моніторингу, роботу з журналом подій, тікетами та керуванням ботами. Усі ці сценарії підтвердили цілісність клієнтської частини, а також коректну взаємодію з API та WebSocket без необхідності ручного оновлення сторінки. Це означає, що адміністративна панель не лише візуалізує інформацію, а й реально підтримує робочий процес реагування на інциденти.

Польовий тест у Telegram-середовищі дав ще більш прикладний результат. У сценаріях burst flood (різке масове надсилання повідомлень), повторного спаму та підозрілих URL бот переходив від попередження до видалення повідомлень і подальшої ескалації, якщо порушення повторювалися. У тестовій послідовності зафіксовано випадки, де після першого попередження архітектура повторно спрацьовувала на тому ж типі поведінки, що призводило до посилення реагування. Це свідчить про те, що механізм реагування не є одноразовим, а враховує накопичення інцидентів у межах часу та контексту. Аналогічний механізм сповіщення було передбачено і для Discord-середовища, де модератор отримує повідомлення про спам або шкідливі посилання, як показано на рис. 3.7.

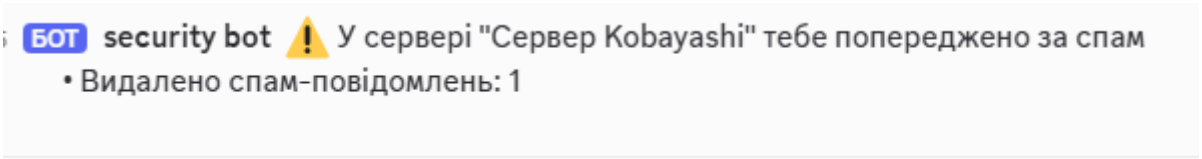

Рисунок 3.7 — Discord-сповіщення модератора про спам і шкідливі посилання

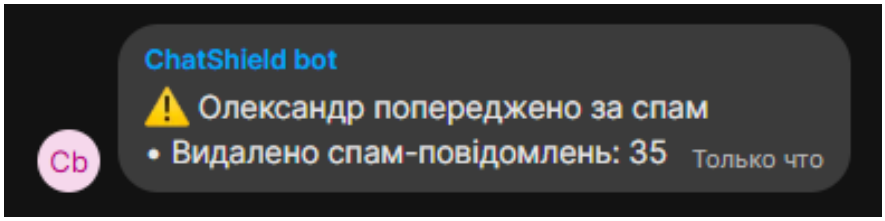

Рисунок 3.8 — Signal-сповіщення про попередження за спам

Додатковий приклад локалізації адміністративної панелі японською мовою подано на рис. 3.9.

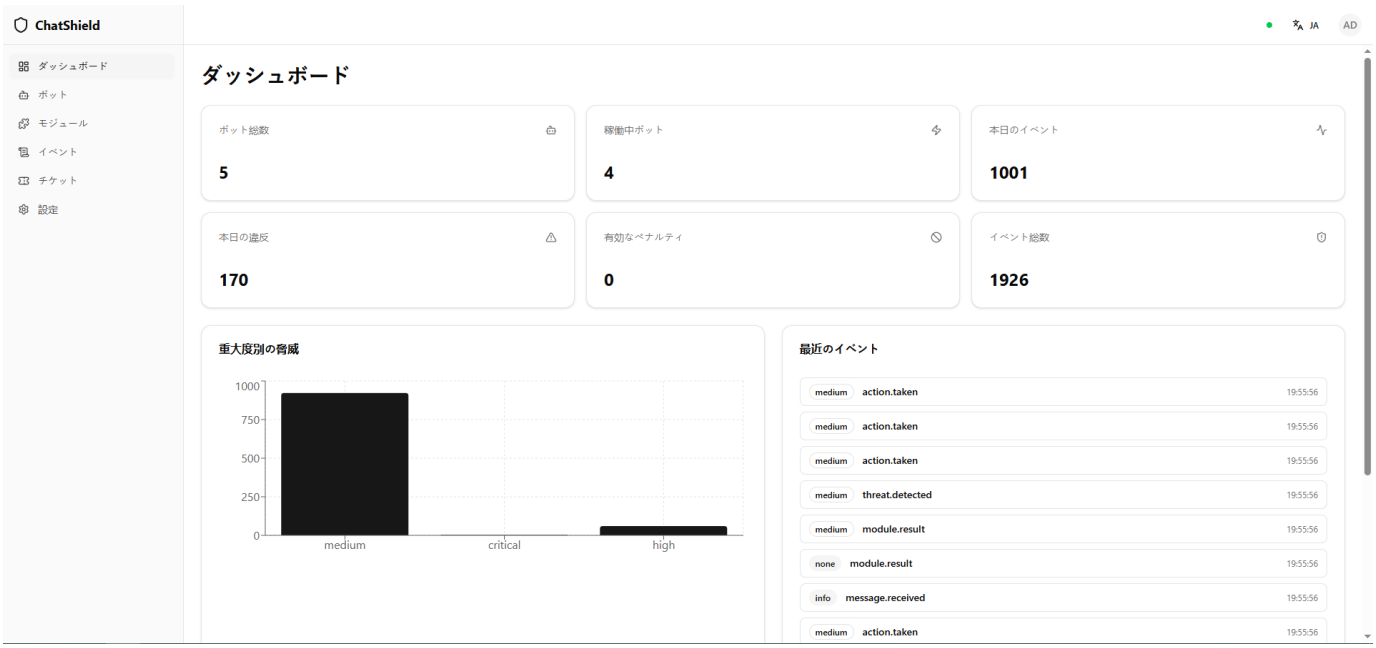

Рисунок 3.9 — Dashboard японською мовою

Узагальнюючи результати, слід зазначити, що ChatShield Hub демонструє високу практичну ефективність на трьох рівнях: аналітичному, операційному та інтерфейсному. На аналітичному рівні система досягла високих показників на контрольних наборах; на операційному рівні забезпечила швидке виявлення та реагування; на інтерфейсному рівні — стабільне відображення подій, тікетів і зведеної статистики. Саме тому отримані результати доцільно використовувати як базу для подальшого розширення системи, тоді як у завершальних висновках можна стисло констатувати факт реалізації та підтвердженої працездатності без повторного наведення всіх числових значень. Приклад попередження користувача про спам у Signal-середовищі наведено на рис. 3.8.

ВИСНОВКИ

У процесі виконання магістерської дипломної роботи було спроєктовано та реалізовано клієнт-серверну систему ChatShield Hub, призначену для моніторингу, аналізу та автоматизованого реагування на загрози в чат-комунікаціях. Розроблене рішення орієнтоване на роботу в багатоплатформному середовищі та забезпечує централізовану обробку повідомлень із Telegram, Discord і Signal через єдину архітектурну модель. У межах роботи було розглянуто основні загрози для чат-середовищ, проаналізовано вимоги до системи, спроєктовано її архітектуру, реалізовано основні програмні компоненти та проведено перевірку їх працездатності.

У результаті розроблено систему, яка поєднує серверну частину, вебінтерфейс адміністратора, платформні адаптери, модулі аналізу повідомлень, механізми автоматизованого реагування, журналювання подій і засоби контролю доступу. Серверна частина забезпечує приймання повідомлень, їх нормалізацію, маршрутизацію до конвеєра аналізу, формування рішення щодо рівня загрози та виконання відповідної модераційної дії. Вебінтерфейс адміністратора дає змогу переглядати події, тікети, статистику, стан підключених ботів і результати роботи модулів у зручній формі.

Одним з основних результатів роботи стало створення модульної архітектури ChatShield Hub. Вона передбачає відокремлення ядра системи від конкретних платформ і аналітичних модулів, що забезпечує можливість подальшого розширення системи без істотної зміни її базової логіки. Адаптери Telegram, Discord і Signal реалізують єдиний підхід до обробки повідомлень та виконання модераційних дій, а модулі аналізу можуть працювати незалежно один від одного в межах спільного конвеєра обробки. Такий підхід підвищує гнучкість системи та дає змогу адаптувати її до різних сценаріїв використання.

У межах програмної реалізації було створено механізми виявлення спаму, повторюваної активності та потенційно небезпечних URL-адрес. Для аналізу повідомлень використано поведінкові ознаки, зокрема інтенсивність надсилання, повторюваність тексту, схожість повідомлень і наявність посилань. Для перевірки URL-адрес передбачено інтеграцію із зовнішнім сервісом аналізу загроз. На основі результатів аналізу система формує рішення щодо подальшої реакції, зокрема попередження, видалення повідомлення, обмеження користувача або застосування інших модераційних дій.

Окрему увагу приділено питанням безпеки та керованості системи. У ChatShield Hub реалізовано JWT-автентифікацію, рольове розмежування доступу, аудит адміністративних дій, обмеження частоти запитів і захист конфігураційних даних. Це дозволяє використовувати систему не лише як інструмент автоматизованої модерації, а й як контрольований адміністративний контур для роботи з інцидентами в чат-комунікаціях. Наявність журналу подій і тікетів забезпечує можливість подальшого аналізу порушень, перегляду історії реагування та контролю дій адміністраторів.

Проведене тестування підтвердило працездатність основних компонентів системи та коректність їхньої взаємодії. За результатами синтетичних експериментів система продемонструвала високі показники точності виявлення спаму та шкідливих посилань, а також коректність виконання модераційних дій. Польовий тест у контрольованому Telegram-середовищі підтвердив, що система здатна приймати реальні повідомлення, виявляти загрози, виконувати відповідні дії та передавати події до вебінтерфейсу в режимі, близькому до реального часу. Операційні показники тестування засвідчили своєчасне виявлення спам-активності, швидке виконання модераційних дій і стабільне оновлення інформації на Dashboard.

Практична цінність розробленої системи полягає в можливості автоматизувати частину роботи адміністраторів чат-спільнот, зменшити час

реагування на типові порушення та забезпечити централізований контроль подій у різних месенджерах. ChatShield Hub може бути використана для модерації Telegram-чатів, Discord-серверів, Signal-груп та інших комунікаційних середовищ, де важливими є швидкість реагування, прозорість дій і можливість подальшого аналізу інцидентів. Завдяки модульній структурі система може бути адаптована до потреб конкретної організації або спільноти.

Отримані результати свідчать про доцільність обраного архітектурного підходу та підтверджують практичну придатність розробленої системи як дипломного програмного рішення. Водночас проведене тестування виконувалося в контрольованих умовах, тому подальше впровадження системи в массованому середовищі потребує додаткової перевірки на більших обсягах даних, з більшою кількістю користувачів і різними сценаріями навантаження.

Подальший розвиток ChatShield Hub може бути пов’язаний із розширенням переліку підтримуваних платформ, зокрема додаванням Slack або Matrix, упровадженням складніших моделей машинного навчання для аналізу контенту, удосконаленням механізмів динамічного оновлення модулів, розгортанням у Kubernetes-середовищі та інтеграцією з корпоративними системами автентифікації й моніторингу. Такі напрями розвитку дозволять підвищити масштабованість, гнучкість і практичну ефективність системи в умовах реального використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Andriotis P., Li Z., et al. (2015). Real-time Monitoring of Privacy Abuses and Intrusion in Android. Proceedings of the HAS Conference. Дата звернення: 05.03.26. Джерело: https://andriotisp.github.io/assets/pdf/has2015li.pdf
2. Putz F., Haesler S., Hollick M. (2022). User Perceptions of Security and Privacy for Group Chat. ACM Transactions on Computer-Human Interaction, 29(1), 1-37. Дата звернення: 08.03.26. Джерело: https://dl.acm.org/doi/10.1145/3491265
3. Abdo A.A., Alhajri K., Alyami A. (2023). AI-based Spam Detection Techniques for Online Social Networks: Challenges and Opportunities. Journal of Internet Services and Information Security, 13(2), 1-20. Дата звернення: 11.03.26. Джерело: https://jisis.org/wp-content/uploads/2023/08/2023.I3.006.pdf
4. Alkaeed M., et al. (2023). AI Methods Used for Spam Detection in Social Systems: An Overview. Proceedings of the 10th SNAMS Conference, IEEE. Дата звернення: 14.03.26. Джерело: https://aau.ac.ae/en/research/aimethodsusedforspamdetectioninsocialsystemsanoverview
5. Sharma R.M., et al. (2024). Active Chat Monitoring and Suspicious Chat Detection Over Internet. International Journal of Scientific Research in Engineering and Management, 8(5), 1-5. Дата звернення: 21.03.26. Джерело: https://www.mendeley.com/catalogue/805ad54b-900b-3afd-a4b2-301c09306ed2/
6. Thomas L. (2022). A Comprehensive Overview of Telegram Services. International Journal of Case Studies in Business, IT, and Education, 6, 288-301. Дата звернення: 25.03.26. Джерело: https://ouci.dntb.gov.ua/en/works/7PxPJMe7/

7. Georgiev G., et al. (2022). Telegram and Discord as a Part in Control System. AIP Conference Proceedings, 2449(1), 020018. Дата звернення: 29.03.26. Джерело: https://pubs.aip.org/aip/acp/article/2449/1/020018/2823892/Telegram-and-discord-as-a-part-in-control-system
8. Putz F., et al. (2024). Sounds Good? Fast and Secure Contact Exchange in Groups. Proceedings of the ACM on Human-Computer Interaction, 8(CSCW2), 1-44. Дата звернення: 05.04.26. Джерело: https://dl.acm.org/doi/10.1145/3686964
9. Mehta R., Gandhi S. (2018). A Survey: SMS Spam Filtering Techniques. International Journal of Trend in Scientific Research and Development, 2(3), 1234-1240. Дата звернення: 09.04.26. Джерело: https://www.slideshare.net/slideshow/a-survey-sms-spam-filtering/115371092
10. Wang L., et al. (2025). Key insights into recommended SMS spam detection methods. Scientific Reports, 15, 92223. Дата звернення: 12.04.26. Джерело: https://www.nature.com/articles/s41598-025-92223-1
11. Anderson R. (2020). Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd Edition, Wiley, 1072 p.
12. Kindervag J., et al. (2017). Zero Trust Networks: Building Secure Systems in Untrusted Networks. O'Reilly Media, 248 p.
13. Fette I., Melnikov A. (2011). The WebSocket Protocol. RFC 6455, IETF. Дата звернення: 10.05.26. Джерело: https://datatracker.ietf.org/doc/html/rfc6455
14. Diachun V. (2024). AI-Based Phishing Attack Detection Using NLP. IC-ITECHS, 5(1), 1-10. Дата звернення: 25.04.26. Джерело: https://jurnal.ubhinus.ac.id/IC-ITECHS/article/view/1590
15. Kumar P., et al. (2023). Real-Time Systems, Architecture, Scheduling, and Verification. IntechOpen. Дата звернення: 28.04.26. Джерело: https://www.intechopen.com/books/1921

16. Santos J., et al. (2023). A Real-Time Platform to Monitoring Misinformation on Telegram. ICISSP Proceedings, 1-8. Дата звернення: 30.04.26. Джерело: https://www.scitepress.org/Papers/2023/120391/120391.pdf
17. Li Y., et al. (2024). Chat Application Architecture Explained. PubNub Blog (technical report). Дата звернення: 02.05.26. Джерело: https://www.pubnub.com/blog/chat-application-architecture-explained/
18. Wang S., et al. (2026). Real-Time Chat Application: A Comprehensive Overview. IJISRT, 24(DEC), 729. Дата звернення: 8.05.26. Джерело: https://www.ijisrt.com/assets/upload/files/IJISRT24DEC729.pdf
19. Strand J. (2017). Offensive Countermeasures: Protecting Against Advanced Adversaries. CreateSpace, 300 p.
20. Limoncelli T., et al. (2020). The Defensive Security Handbook: Best Practices for Securing Infrastructure. O'Reilly Media, 280 p.
21. Abdo A.A. (2023). Unveiling user activities on instant messaging platforms. Knowledge-Based Systems, 95, 1-15. Дата звернення: 8.05.26. Джерело: https://www.sciencedirect.com/science/article/pii/S0950705126006192
22. Al-Mansoori R., et al. (2025). A Scalable Telegram-Based Botnet Framework for Stealthy Attacks. ISJ Journal, 1-12. Дата звернення: 04.05.26. Джерело: https://isj.vn/index.php/journal_STIS/article/download/1102/378/4935
23. Hands-On Authors. (2023). Hands-On Real Time Communications. Packt Publishing, 30 p. Дата звернення: 09.05.26. Джерело: https://content.e-bookshelf.de/media/reading/L-26019429-865423b684.pdf