Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ АВТОМАТИЗОВАНОГО ФІЛЬТРУВАННЯ ОБРАЗЛИВИХ ПОВІДОМЛЕНЬ У ЧАТАХ

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Гнідий К. В.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)
«4» червня 2025 р.

Рецензент: Ph. D., ст. викладач каф. САІТ
Войцеховська О.О.
(прізвище та ініціали)
«5» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«6» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д. т. н. Яровий А.А.
«5» 05. 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Гнідому Костянтину Вікторовичу

1. Тема роботи: Програмний модуль автоматизованого фільтрування образливих повідомлень в чатах.
керівник роботи: Арсенюк Ігор Ростиславович, к. т. н., доц.
затверджені наказом вищого навчального закладу « 20 » 03. 2025 року № 97

2. Термін подання студентом роботи 30.05.2025 р.

3. Вихідні дані до роботи:
Діапазон індексу токсичності повідомлення – від 0 до 1;
Критерії оцінки токсичності повідомлення – не менше 5;
Кросплатформність – веб-додаток та бот для месенджерів;
Мова програмування – об'єктно-орієнтована;
Інтерфейс користувача – інтуїтивно зрозумілий.
Середовища розробки: Visual Studio Code та PHPStorm.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності використання технології автоматизованого фільтрування образливих повідомлень у розробці веб-чатів; проектування програмного модуля фільтрування образливих повідомлень; програмна реалізація модуля фільтрування образливих повідомлень; висновки; список використаних джерел; додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема алгоритму роботи програмного модуля фільтрування образливих повідомлень; загальна структурна схема компонентів програмного модуля фільтрування образливих повідомлень; UML-діаграма класів серверної частини модуля фільтрування образливих повідомлень; UML-діаграма класів серверної частини модуля авторизації; ER-модель бази даних програмного модуля фільтрування образливих повідомлень; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Арсенюк І. Р., к. т. н., доц. каф. КН		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обґрунтування доцільності розробки програмного модуля фільтрування образливих повідомлень	05.05.2025	09.05.2025	
2	Проектування програмного модуля фільтрування образливих повідомлень	12.05.2025	16.05.2025	
3	Програмна реалізація модуля фільтрування образливих повідомлень	19.05.2025	23.05.2025	
4	Висновки та аналіз отриманих результатів	26.05.2025	30.05.2025	
5	Оформлення додатків	02.06.2025	04.06.2025	
6	Оформлення матеріалів до захисту БДР	05.06.2025	06.06.2025	

Студент _____ **Гнідий К. В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Арсенюк І. Р.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 85 сторінок формату А4, які включають 27 рисунків і 6 таблиць. У списку використаних джерел зазначено 21 найменування.

Метою цієї роботи є розширення функціональних можливостей програмного модуля автоматичного фільтрування образливих повідомлень у чатах.

Програмний модуль зроблено в середовищі Visual Studio Code з використанням нових мов програмування та фреймворків. Тестування функцій системи було проведено, результати якого підтверджують ефективність виявлення і блокування образливих повідомлень.

Розроблене рішення може бути інтегроване в різні платформи обміну повідомленнями, щоб забезпечити більш безпечне середовище для користувачів.

Ключові слова: автоматичне фільтрування образливого контенту, чат-платформи, безпека онлайнкомунікації, інтерактивність, кібербулінг.

ABSTRACT

The bachelor's thesis consists of 85 A4 pages, including 27 figures and 6 tables. The list of references contains 21 sources.

The aim of this work is to enhance the functionality of a software module for automatic filtering of offensive messages in chats.

The module was developed in the Visual Studio Code environment using modern programming languages and frameworks.

System functions were tested, and the results confirm the effectiveness of detecting and blocking offensive messages.

The developed solution can be integrated into various messaging platforms to provide a safer communication environment for users.

Keywords: automatic offensive content filtering, chat platforms, online communication security, interactivity, cyberbullying.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасному цифровому середовищі, де онлайн-спілкування стало невід'ємною частиною особистого, професійного та освітнього життя, питання безпечної та відповідальної комунікації набуває дедалі більшої актуальності. Соціальні мережі, чати, форуми та інші платформи обміну повідомленнями об'єднують мільйони користувачів по всьому світу, відкриваючи нові можливості для взаємодії, водночас створюючи й численні ризики. Одним із найбільш серйозних викликів у цій сфері є поширення токсичних повідомлень, мови ворожнечі, цькування, приниження, образ та інших проявів деструктивної поведінки [1].

Згідно з дослідженнями у сфері цифрової безпеки, кількість випадків кібербулінгу постійно зростає, а його наслідки можуть бути надзвичайно серйозними — від психологічного дискомфорту до депресій і суїцидальних настроїв серед користувачів, особливо підлітків. При цьому модерація онлайн-контенту вручну стає все менш ефективною через масштабність і динамічність цифрових платформ. Затримки у виявленні образливого змісту, людський фактор, брак ресурсів — усе це створює передумови для впровадження автоматизованих рішень, здатних у режимі реального часу виявляти й фільтрувати небажаний контент.

У цьому контексті зростає значущість програмних модулів, які застосовують методи штучного інтелекту, обробки природної мови (NLP), машинного навчання та лінгвістичного аналізу для класифікації повідомлень та виявлення потенційно образливих висловлювань. Такі системи здатні не лише зменшити навантаження на модераторів, а й підвищити загальний рівень безпеки цифрового простору. Вони можуть бути використані в соціальних мережах, чат-застосунках, онлайн-іграх, освітніх платформах, службах підтримки тощо.

Особливої актуальності проблема набуває у зв'язку з поширенням анонімного спілкування, коли користувачі відчувають себе безкарними й схильні до проявів агресії. В таких умовах превентивне фільтрування контенту стає не

просто бажаною функцією, а необхідністю для захисту користувачів, збереження репутації платформи та дотримання етичних норм взаємодії. Актуальність дослідження підсилюється й тим, що існуючі інструменти не завжди враховують контекст висловлювань, тональність, іронію чи культурні особливості мови, що створює виклик для подальших наукових і практичних розробок у цій сфері.

Метою дослідження є розширення функціональних можливостей програмного модуля автоматичного фільтрування образливих повідомлень у чатах.

Об'єктом дослідження є процес обробки текстових повідомлень у чатах.

Предметом дослідження є програмні засоби для реалізації фільтрації образливого контенту.

Задачі дослідження:

1. Проаналізувати сучасні методи фільтрації образливих повідомлень.
2. Виконати аналіз відомих аналогів програмного забезпечення для автоматизованого фільтрування образливих повідомлень та виокремити їх переваги та недоліки.
3. Розробити загальний алгоритм роботи програмного модуля для автоматизованого фільтрування образливих повідомлень.
4. Реалізувати програмний модуль фільтрації.
5. Виконати тестування програмного модуля та здійснити аналіз отриманих результатів.
6. Розробка інструкції користувача.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ ФІЛЬТРУВАННЯ ОБРАЗЛИВИХ ПОВІДОМЛЕНЬ

1.1 Аналіз предметної області та методів обробки природної мови

З розвитком інтернету та соціальних мереж проблему негативного впливу ненависті та агресії в онлайн-комунікаціях все частіше досліджують як соціологи і психологи, так і фахівці з обробки мовлення. Токсичні або образливі коментарі у мережах – це висловлювання, що містять грубу лайку, принизливі та провокативні формулювання, спрямовані на вираження ненависті, зневаги або наміру образити іншу особу. В інформатиці часто вживають термін «hate speech», перекладаючи його як «мовлення ненависті», що характеризується агресивним висловлюванням щодо певних соціальних груп. Загалом токсичний коментар визначається як грубий, образливий або неповагаючий вислів, котрий імовірно викличе негативну реакцію – напризаписклад, спонукає співрозмовника залишити дискусію. Тобто основна ознака токсичності – це не просто наявність «поганих» слів, а передусім зловмисний намір зачепити або образити співрозмовника. У таких коментарях часто спостерігається «принизлива лексика, використання нецензурних слів та сарказму чи іронії», спрямованих «зачепити почуття людини, яка прочитає даний коментар». Тому аналіз токсичних коментарів полягає у виявленні цих лінгвістичних і стилістичних ознак агресії [2].

Крім того, одним із аспектів токсичності є зловмисний намір автора. Навіть нейтральні на перший погляд слова можуть бути частиною образливого послання, якщо вони сказані з метою принизити. Тому система аналізу коментарів прагне визначати не лише тематику або тон висловлювання, а й потенційно «небезпечний» намір промовця. Саме ця задача модерації коментарів стала актуальною, коли популярні видання і соцмережі на початку 2000-х почали відчувати потребу блокувати окремі дискусії через різкий ріст агресії серед коментаторів. Наукові праці з цього питання наголошують, що повноцінний

аналіз токсичних повідомлень неможливий без урахування намірів і контексту спілкування, адже людині нерідко достатньо побачити «дружній» фасад (позитивні слова), щоб не зачепити її, а штучному інтелекту – виявити прихований зміст випадає складніше. Такий формалізм лише підкреслює необхідність точного розуміння намірів у кожній конкретній ситуації [3].

Поява іронії та сарказму у виразах вносять додаткову складність у аналіз мови. Обидва ці феномени – випадки «непрямого» вираження наміру: вони спираються на те, що реальний зміст висловлювання відрізняється або прямо суперечить буквальному значенню слів. Іронія – це троп, коли істинний смисл вислову протиставлено формі компліменту чи позитивної оцінки для прихованої насмішки чи викриття. Прикладом може бути фраза «О, дивись, вона знову запізнилася – просто взірець пунктуальності!» – тут комплімент «взірець пунктуальності» іронічно означає протилежне. Іронія може бути використана у різних стилях мовлення: від м'якої жартівливої до саркастичної, і не завжди несе чітку образу – іноді це просто засіб гумору чи коментування абсурду ситуації. Сарказм ж є брутальнішим і специфічнішим феноменом: це гострий, їдкий жарт, який будується на різкому контрасті вигляду фрази і її реального змісту, з чітким завданням принизити адресата або висміяти конкретний об'єкт. У сарказмі завжди присутня мета «зачепити когось»: як зазначається у популярному виданні, «сарказм одразу має на меті когось боляче зачепити, у нього є конкретний адресат, тоді як іронії це не обов'язково». Наприклад, «Такий геній!» – з вуст у сатиричному контексті буде сприйнято як пряма образа здібностей адресата [4].

Для NLP-систем сарказм і іронія становлять помітну перешкоду. Це пояснюється тим, що формальні алгоритми зазвичай опрацьовують текст «буквально»: вони оцінюють семантичну полярність слів і інтонаційні маркери, але не завжди можуть зіставити контекст чи саркастичний намір. Як стверджують дослідники штучного інтелекту, «текст, який споживач ШІ виробляє, схильний сприйматися ним буквально, що призводить до невдалого декодування імплікатур». Так само практичні спостереження підтверджують: якщо в тексті відсутні прямі слова образи, але вислів подано саркастично,

звичайний класифікатор може не помітити агресію. Сучасні дослідження спонукають до впровадження контекстуальних моделей. Наприклад, новітній підхід на базі трансформерів RoBERTa та DistilBERT, що враховує навколишній контекст повідомлення, продемонстрував надзвичайно високі результати у задачі виявлення сарказму – до 99% F1-метрики на англомовному датасеті новинних заголовків. Крім того, автори показали, що без урахування контексту показник F1 на датасеті Reddit зростав з 49% до 75% після додавання відповідних контекстних підказок. Це підтверджує: для правильного розуміння іронії та сарказму системі потрібно враховувати ширший контекст, наприклад попередні репліки в діалозі, знання про ситуацію чи навіть позалінгвістичні сигнали (може, емодзі чи інтонацію, яку в тексті важко передати). Без цього манера вислову сприймається буквально, і справжній намір автора «проглядається» [5].

Важливо зазначити, що іронію не слід плутати з іншими видами непрямого гумору чи фігурами стилю. На кшталт побутового вияву – наприклад, випадкової невдачі або двозначності – часто реагують як на іронію («О, ось тебе й знайшов!» коли нарешті зустріли потрібну людину). Але сарказм завжди має елемент «ножа у словах» – у нього конкретна ціль. У сукупності ці лінгвістичні стратегії (імплікатури, сарказм, «постіронія» як соціальний феномен) є предметом окремих досліджень мовознавців, які відзначають: успішне декодування іронічного змісту потребує когнітивного розуміння, часто виходячи за межі тексту.

Методи обробки природної мови (NLP) стають головним інструментом аналізу коментарів. Класичні підходи передбачають попередню обробку тексту (видалення пунктуації, приведення слів до початкової форми, токенізацію), а потім — побудову фіч для алгоритмів машинного навчання. Наприклад, з початком ери глибокого навчання популярними стали нейронні мережі, зокрема рекурентні й трансформерні архітектури, які здатні вловлювати семантичні зв’язки між словами. У ряді досліджень застосовували прості методи, такі як меш-словниковий підхід (bag-of-words) або TF-IDF з логістичною регресією чи SVM, для класифікації коментарів як «токсичні» чи «нетоксичні». Наприклад, в

одному українському дослідженні на основі коментарів з YouTube було показано, що вже моделі SVM із ядром з випадковим лісом досягали високих показників точності, підтверджуючи практичність автоматичної модерації контенту.

Проте явним трендом останніх років є використання моделей-попередньо навчених трансформерів (BERT, RoBERTa, GPT і їхніх модифікацій). Встановлено, що моделі на базі трансформерної архітектури значно перевершують старі методи у задачах розуміння тону та намірів тексту. Так, у згаданій роботі згрупи вчених було продемонстровано: налаштування BERT на задачу класифікації коментарів дало майже ідеальні результати для датасету Kaggle Toxic, а врахування специфічного контексту при навчанні дозволило підвищити F1 та ефективність розпізнавання сарказму. Це означає, що для машинного аналізу намірів користувача бажано враховувати не тільки окреме речення, але й те, що йому передувало або супроводжувало (наприклад, наявність цитованого повідомлення чи відповіді) [6].

У контексті розпізнавання наміру «образити чи ні», NLP-системи зазвичай формулюють задачу як багатокласову класифікацію: наприклад, «грубі висловлювання», «ненависть», «сарказм без образи», «нейтральне» тощо. Для цього створюють розмічені корпуси коментарів, де приклади сарказму (з наочними жартами) відрізняють від прямих образ чи похвали. У навчанні моделі можуть задіювати як лексичні фічі (наявність нецензурної лексики, слів негативної конотації), так і семантичні (ембедінги, що уловлюють суть речення), а також позалінгвістичні (наявність емодзі, інтенсивність розділових знаків). Однак найскладнішою є інтерпретація саме тих випадків, коли поверхнево фраза здається позитивною або нейтральною, але має прихований негативний підтекст (наприклад, сарказм), або навпаки – коли образа прикрита емоційними маркерами ніжності (так звані «цинично-саркастичні» компліменти). Тому навчання моделі потребує спеціальних прикладів саркастичних та іронічних висловлювань [7].

1.2 Оцінка можливих платформ реалізації програмного забезпечення

Вибір відповідного середовища для реалізації програмного модуля є критично важливим етапом у процесі розробки. Сучасні технології пропонують безліч рішень, що відрізняються мовами програмування, архітектурою, масштабованістю та підтримкою обробки природної мови.

Платформи для реалізації програмного модуля мають свої особливості, які істотно впливають на функціональність, зручність використання та потенціал масштабування. Розглядаючи середовище ПК-додатків, варто зазначити, що такі рішення зазвичай демонструють високу функціональність. Вони дозволяють глибоко інтегруватися з операційною системою, використовувати повноцінні графічні інтерфейси та працювати з великими обсягами даних. Завдяки підтримці таких середовищ розробки, як .NET, Qt чи Electron, розробники мають значний контроль над усіма аспектами роботи програми. Проте основним недоліком ПК-додатків є їхня обмежена мобільність: користувач повинен мати доступ до конкретного пристрою, на якому встановлено програму. Це обмежує гнучкість і оперативність користування модулем у повсякденному житті, особливо в умовах, коли важлива мобільність.

Мобільні додатки, з іншого боку, пропонують значно вищу мобільність. Вони завжди під рукою, оскільки смартфони та планшети є невід'ємною частиною сучасного користувача. Це створює значну перевагу з точки зору доступності та зручності. Крім того, мобільні середовища підтримують сучасні мови програмування, такі як Kotlin, Swift або фреймворки на кшталт Flutter, які дозволяють створювати кросплатформенні рішення. Недоліком є потреба в окремій інсталяції, що для частини користувачів може стати бар'єром, а також обмеження за ресурсами пристрою, що може впливати на продуктивність модуля, особливо при обробці великих обсягів текстових даних чи використанні моделей штучного інтелекту.

WEB-застосунки надають значну гнучкість, універсальність і кросплатформеність. Їхня головна перевага — це відсутність необхідності встановлення, що дає змогу користувачам швидко отримати доступ до сервісу з будь-якого пристрою, який має браузер і підключення до Інтернету. Сучасні фреймворки, такі як React або Vue.js, дозволяють створювати інтерактивні інтерфейси, які забезпечують високий рівень взаємодії. Крім того, серверна частина може бути реалізована з використанням PHP, Python, Node.js або інших технологій, що відкриває широкі можливості для масштабування. Недоліки цієї платформи переважно пов'язані з залежністю від інтернет-з'єднання, а також із необхідністю додаткового захисту даних, особливо коли мова йде про чутливу інформацію, що передається через мережу.

Окрему категорію становлять месенджери, зокрема платформа Telegram. Її можна вважати однією з найзручніших для швидкої реалізації модуля, оскільки Telegram вже надає власну інфраструктуру та API для створення ботів. Ці боти можуть працювати як посередники між користувачем і сервером, обробляючи повідомлення, надсилаючи відповіді та виконуючи перевірку токсичності. Перевагою є також те, що Telegram доступний практично на всіх платформах, тому користувач не мусить встановлювати окремий додаток — достатньо скористатися вже наявним месенджером. Це робить Telegram універсальним інструментом для інтеграції модулів модерації, забезпечуючи як зручність для користувача, так і швидкість розробки для розробника.

У таблиці 1.1 наведено порівняльну характеристику середовищ, які можуть бути використані для розробки модуля автоматизованого фільтрування образливих повідомлень у чатах.

Таблиця 1.1 – Порівняльна характеристика платформ для реалізації програмного модуля фільтрації повідомлень.

Критерій	ПК-додатки	Мобільні пристрої	WEB-застосунки	Месенджери (Telegram)
Доступність	Обмежена робочим місцем	Висока	Висока	Висока

Продовження таблиці 1.1

Критерій	ПК-додатки	Мобільні пристрої	WEB-застосунки	Месенджери (Telegram)
Операційні системи	Windows, macOS, Linux	Android, iOS	Будь-яка, що підтримує браузер	Працює в межах Telegram на всіх ОС
Охоплення користувачів	Низьке	Високе	Високе	Дуже високе завдяки популярності Telegram
Потреба в установці	Потрібна установка	Потрібна установка	Не потребує установки	Не потребує – використовується вже наявний Telegram-додаток
Зручність використання	Висока функціональність, але низька мобільність	Висока мобільність	Універсальний доступ із різних пристроїв	Надзвичайна зручність
Екосистема та інструменти	.NET, Qt, Electron та ін.	Kotlin, Swift, Flutter тощо	Vue.js, React, Laravel, Django, Flask	Telegram Bot API, Node.js, Python-боти, Webhooks
Мови програмування	Java, C#, Python, C++	Kotlin, Swift, Dart	JavaScript, PHP, Python, TypeScript, інші	Python, JavaScript, Go, PHP та ін. через Bot API
Інтерактивність	Висока, залежить від GUI	Висока	Дуже висока через фронтенд-фреймворки	Обмежена форматом повідомлень, але висока доступність
Швидкість реалізації	Середня	Середня	Висока	Висока – завдяки готовим API та швидкій інтеграції

У сучасних умовах, коли користувачі прагнуть швидкого та зручного доступу до цифрових сервісів, Telegram-боти та веб-застосунки є доцільними середовищами для реалізації програмних рішень. Вони не потребують складної

установки, є доступними з будь-якого пристрою, що має підключення до Інтернету, та мають низький поріг входу для кінцевого користувача.

Telegram-бот дозволяє інтегрувати функціональність модуля у вже знайоме середовище месенджера. Оскільки Telegram широко використовується на мобільних пристроях, це забезпечує постійний доступ до сервісу в зручному форматі, без потреби встановлювати окремий додаток. Користувачі можуть отримувати повідомлення про виявлені образливі повідомлення прямо в чаті, що є особливо зручним у повсякденному користуванні.

Водночас веб-застосунок надає більше можливостей для візуального оформлення й інтерактивної взаємодії. Він дозволяє створити повноцінний інтерфейс із чатом або панеллю керування, де модуль фільтрування образливих повідомлень працює як частина окремого сайту. Це відкриває додаткові можливості для розширення функціоналу в майбутньому – наприклад, додавання системи звітності, статистики, налаштувань фільтрів або модерації контенту. У підсумку, використання обох середовищ – Telegram-бота та веб-застосунку – дозволяє досягти максимально широкого охоплення аудиторії, поєднуючи зручність, швидкість доступу та гнучкість у реалізації функціоналу.

Такий підхід дозволяє задовольнити потреби як мобільних, так і десктопних користувачів.

1.3 Огляд систем-аналогів

У сучасному цифровому середовищі існує багато інструментів, які реалізують автоматичне виявлення та блокування образливого, токсичного або небажаного контенту в чатах і соціальних мережах. Кожен із цих рішень має свої особливості, переваги та обмеження, що важливо враховувати під час розробки власного модуля.

Для порівняння розглянемо кілька популярних систем: Microsoft Content Moderator, Detoxify, HateSonar, DeepAI Toxic Comment Classification API, Cleanspeak, ToxiChat.

Сервіс Microsoft Content Moderator – хмарний компонент Azure AI, що дозволяє автоматично перевіряти текст, зображення і відео на неприйнятний контент. Система має широкий набір функцій: пошук нецензурних слів, образ (hate speech), перевірку на особисті дані, а також технології OCR. За даними документації, Azure Content Moderator претендує на підтримку «понад 100 мов» для виявлення ненормативної лексики. Однак офіційні таблиці показують, що українська мова зазначена лише як мова виявлення тексту (language detection), без позначки підтримки модулю перевірки лайливих слів [8].

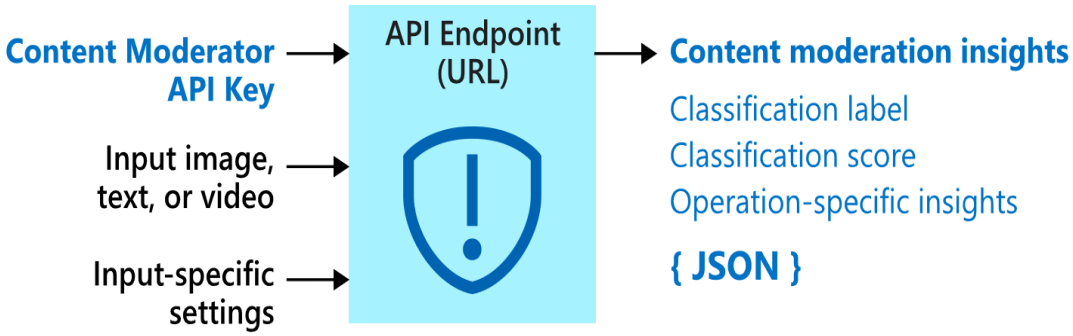


Рисунок 1.1 – Схема роботи Microsoft Content Moderator.

Переваги:

- Широке багатомовне охоплення
- Готові SDK і інтеграції в екосистему Azure (Power Apps, Logic Apps тощо), забезпечуючи зручність використання у корпоративних рішеннях.
- Сервіс надає як API, так і веб-інтерфейс для модераторів з рев'ю-панеллю (є UI для ручної оцінки помічених випадків).
- Можливість створювати власні «чорні списки» слів і виразів (custom block lists) для конкретних спільнот або тематики.

Недоліки:

- Погана підтримка української мови.
- Висока вартість

Detoxify — це відкрита Python-бібліотека від Unitary, що містить попередньо натреновані трансформерні моделі для класифікації токсичності

коментарів за категоріями (toxic, severe_toxic, obscene, threat, insult, identity_hate тощо). Система має декілька режимів моделей (оригінальна, unbiased, multilingual), оптимізованих для англійської та низки інших мов. Пакет не вимагає донавчання — користувач просто викликає функцію predict на тексті. При цьому Detoxify може ефективно обробляти вирази на тих мовах, на яких його навчили [9].

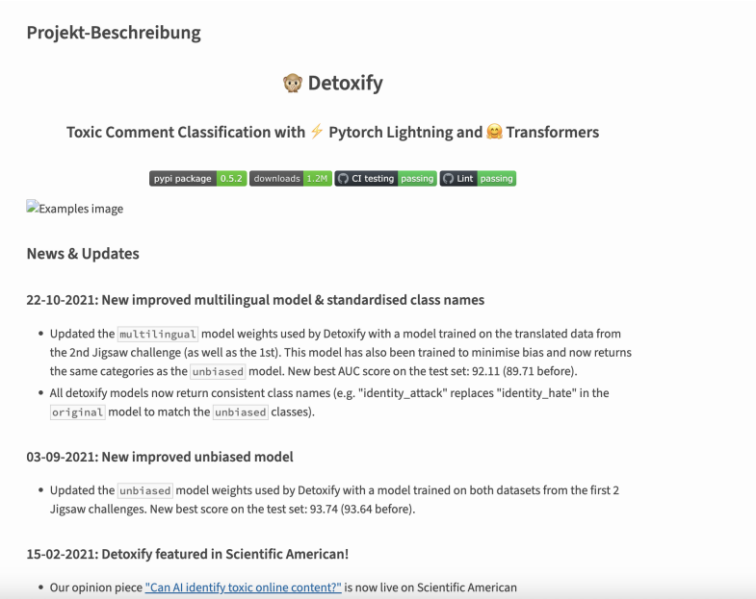


Рисунок 1.2 – Сторінка репозиторію Detoxify.

Переваги:

- Відкритий код і безкоштовність. Модель доступна на GitHub та PyPI, не потребує ліцензійних витрат.
- Легкість інтеграції як Python-бібліотеки (pip install, виклик API-функції в коді). Не потрібні зовнішні сервіси чи ключі.
- Наявність мультимовної моделі, тобто Detoxify містить «multilingual» версію, але вона натренована лише на 7 мовах (англійська, французька, іспанська, італійська, португальська, турецька, російська).

Недоліки:

- Відсутність підтримки української мови.
- Модель на базі BERT–RoBERTa велика (100+ млн параметрів), потребує значної пам’яті та часу на інференс.

- Тільки Python-оточення, тому у веб-додаток чи мобайл доведеться обгортати окремим мікросервісом. Немає готового REST-API чи SDK для інших платформ.

HateSonar — це проста Python-бібліотека для виявлення хейтспічу та образливої мови. Її автор – Hironsan (Naoto Nakayama). У HateSonar вбудовано BERT-based модель, яка класифікує текст на три категорії: hate_speech (ненависть), offensive_language (образа) або neither. Система не вимагає донавчання – просто викликаємо Sonar().ping(text=…) [10].

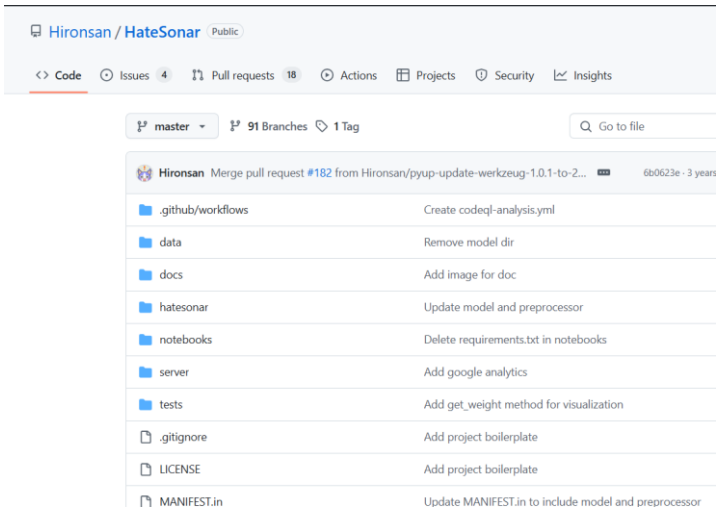


Рисунок 1.2 – HateSonar (бібліотека для Python).

Переваги:

- Легка інтеграція в Python-проєкт.
- Не потрібно тренувати модель
- Повертає докладний результат.
- Безкоштовна та open-source (ліцензія MIT, код на GitHub). Необхідні лише Python-залежності, без сторонніх API-ключів.

Недоліки:

- Тільки англійська. Бібліотека натренована на англомовних даних, жодних інших мов (включно з українською) не підтримує.
- Обмежена актуальність (Останній реліз припадає на 2019 р.).

- Немає веб-сервісу чи API. HateSonar — чисто Python-бібліотека, без REST API чи інтеграційних модулів.

DeepAI Toxic Comment Classification API пропонує кілька API для обробки тексту, зокрема для класифікації токсичних коментарів. Це хмарний сервіс: відправляється HTTP-запит з текстом, сервіс повертає чи є коментар токсичним. Цей API орієнтований передусім на англомовні тексти [11].

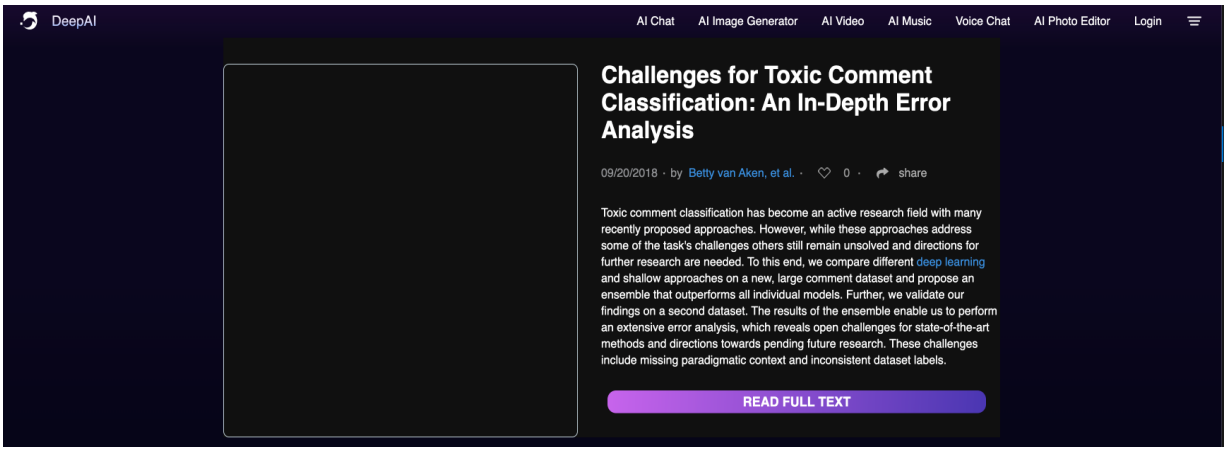


Рисунок 1.3 – DeepAI Toxic Comment Classification API Інтерфейс сайту з API ключем.

Переваги:

- Не треба розгортати модель власноруч; користувач отримує простий REST API.
- Достатньо отримати безкоштовний API-ключ DeepAI та зробити POST-запит; є документація англійською.
- Дозволяє негайно використовувати нейронну класифікацію без участі машинного навчання (достатньо навичок роботи з HTTP).
- Сервіс працює на потужностях DeepAI, можна робити паралельні запити, є платні тарифи для великої кількості запитів.

Недоліки:

- Лише англомовна модель.

- Потрібен постійний доступ до мережі та дійсний API-ключ. Немає автономного режиму.
- Безкоштовний тариф дозволяє небагато запитів, за великий трафік треба платити; можливі затримки в обробці (черги).

CleanSpeak (Inversoft) — комерційне рішення для модерації і фільтрації контенту. Це програмне забезпечення, яке можна встановити на сервер (або використовувати SaaS-версію), і має дуже гнучкі інструменти: чорні/білі списки, машинне навчання, масштабовані алгоритми та повний UI. CleanSpeak підтримує багато мов через кастомні налаштування, але базових словників (як для англійської) для української у стандартному вигляді немає [12].

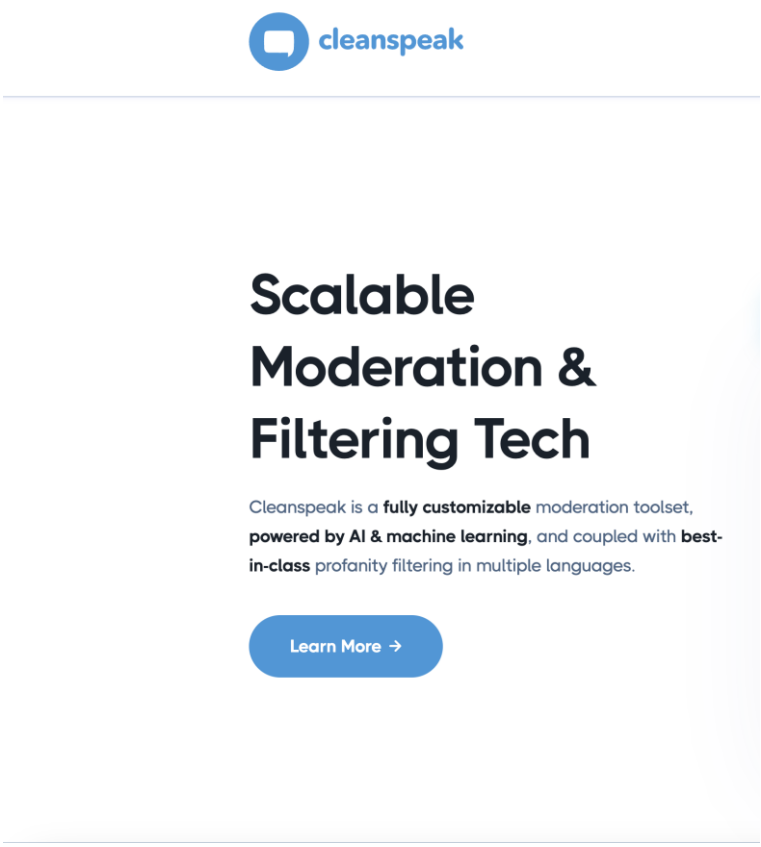


Рисунок 1.4 – CleanSpeak.

Переваги:

- Гнучкість та розширюваність.
- Підтримка будь-якої мови.

- Добре масштабована. Платформа розрахована на високі навантаження (ігрові чати, форуми), дозволяє класифікувати велику кількість повідомлень за секунду.

Недоліки:

- CleanSpeak — комерційний продукт (ліцензія, SaaS-підписка), що може бути недоступно для невеликого проєкту чи ПЗ з обмеженим бюджетом.
- Потрібно налаштовувати сервер, встановлювати ПО або підписуватись на хмарний сервіс; немає простого «безкоштовного хостингу».
- Out-of-box система не містить офіційної колекції українських нецензурних слів.

ToxiChat — не готова система, а датасет та дослідницький набір розмічених діалогів. Він містить реальні приклади спілкування користувачів та бота з маркуванням, чи є відповідь бота образливою та якою. Іншими словами, ToxiChat — це зібраний корпус діалогів для досліджень у сфері безпечного ведення ботів, а не готовий сервіс модерації [13].

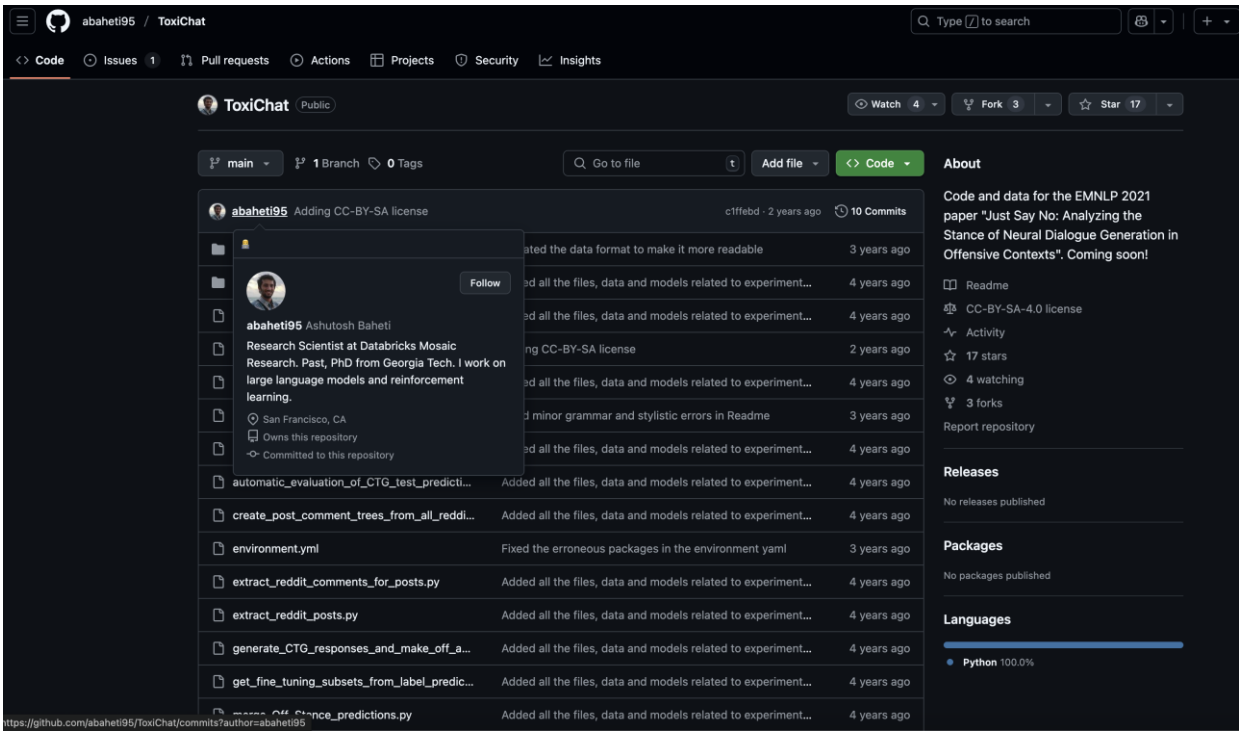


Рисунок 1.5 – Репозиторій ToxiChat.

Переваги:

- Набір сконструйований на реальних повідомленнях із Reddit, містить живі «розмови» та відповіді моделі, що робить його цінним для навчання і тестування моделей модерування.
- Позначені образи та стани.
- Кількість прикладів (загалом близько 2000 діалогових пар (1400 тренувальних, 300 валідаційних, 300 тестових)).

Недоліки:

- Це лише навчальний датасет, а не API чи бібліотека. Щоб його використати, потрібно писати свій класифікатор/модель.
- Лише англійська мова.
- Немає інтерфейсу.

Усі розглянуті інструменти та сервіси демонструють спільну проблему – слабку підтримку української мови у контексті виявлення токсичних висловлювань. Майже всі рішення орієнтовані на англійську або кілька поширених європейських мов і без доопрацювання пропускають українські лайки. Хоча деякі системи (наприклад, Content Moderator або CleanSpeak) заявляють багатомовність, на практиці українська присутня лише як мова розпізнавання тексту, а не як джерело фільтрації образ. Усі вони потребують ручного розширення словників для української або власного моделювання, щоб адекватно блокувати фрази на зневажливому штибу.

Отже, головний недолік усіх існуючих систем модерації – саме відсутність нормальної роботи з українською лексикою. У зв’язку з цим перевагою розроблюваної користувачем системи буде цілеспрямована підтримка української мови: власні моделі або словники, навчені/розроблені для виявлення національних та регіональних образ, забезпечать високу точність фільтрації тих виразів, що інші інструменти пропускають.

1.4 Формулювання вимог та постановка задачі

Зі зростанням популярності онлайн-комунікацій у чатах та месенджерах, значно збільшується і кількість токсичних, образливих або небажаних повідомлень. Це створює дискомфорт для користувачів, негативно впливає на атмосферу спілкування і може призводити до конфліктів у спільнотах. Тому автоматичне виявлення та фільтрація токсичного контенту стає необхідним інструментом для підтримки здорової та безпечної комунікації [14].

Майбутній програмний модуль спрямований на інтеграцію з чатами через ботів в месенджерах та вебзастосунок із функціоналом онлайн-чату. Основним завданням модуля є ефективна детекція образливих і токсичних повідомлень, а також застосування відповідних заходів для підтримки дисципліни у спільнотах.

Передбачуване програмне забезпечення буде мати три ступені покарань для повідомлень із токсичним вмістом:

- Попередження з рекомендацією відредагувати повідомлення — для помірно токсичних висловлювань.
- Попередження про можливий бан — у випадку повторних або більш виражених порушень.
- Миттєвий бан та кік з чату — для дуже токсичних або образливих повідомлень, що порушують правила спільноти.

Основні функції розроблюваного програмного модуля:

- Автоматична детекція токсичного контенту у текстах користувачів.
- Фільтрування повідомлень у реальному часі як у Telegram-боті, так і у вебчаті.
- Реалізація багаторівневої системи покарань залежно від ступеня токсичності.
- Можливість адміністрування та налаштувань модуля через вебінтерфейс.
- Інтуїтивно зрозумілий користувацький інтерфейс для спілкування та отримання повідомлень про порушення.

Для досягнення ефективної роботи модуля були поставлені такі завдання:

- Реалізувати систему автоматичної детекції образливого та токсичного контенту;

- Розробити механізм багаторівневої фільтрації з відповідними попередженнями та санкціями;
- Інтегрувати модуль у Telegram-бота та вебзастосунок із чатом;
- Забезпечити можливість адміністрування модуля через вебінтерфейс;
- Провести тестування на різних сценаріях використання для забезпечення надійності та точності роботи.

1.5 Висновок до розділу 1

У цьому розділі здійснено аналіз існуючих підходів до фільтрації токсичних повідомлень у чатах і месенджерах. Розглянуті системи-фільтри часто мають обмежені можливості або недостатньо гнучкі механізми покарань за токсичний контент, а також недостатню підтримку української мови або низьку точність її обробки. У результаті аналізу визначено, що більшість існуючих рішень орієнтовані на англомовне середовище, а фільтрація українських образ часто базується лише на простому переліку "заборонених слів", без контекстного аналізу або врахування стилістичних особливостей.

Передбачуване програмне забезпечення на етапі подальшої розробки включатиме модуль автоматичного виявлення токсичних повідомлень, який інтегруватиметься у месенджери та вебчат. Особливу увагу планується приділити підтримці української мови, точності обробки контекстуальних висловлювань, зручності взаємодії з користувачем і можливості гнучкої адаптації модераційних правил. Модуль реалізовуватиме триступеневу систему попереджень і санкцій, що має забезпечити ефективне регулювання поведінки в онлайн-спільнотах.

У цьому розділі також сформульовано основні завдання для подальшої реалізації та тестування програмного модуля, серед яких: удосконалення точності аналізу природної мови, адаптація до різних стилів висловлювань (зокрема сарказму, іронії, прихованої образи), забезпечення стабільної роботи в режимі реального часу, підтримка кількох платформ і мов.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ФІЛЬТРАЦІЇ ОБРАЗЛИВИХ ПОВІДОМЛЕНЬ

2.1 Розробка алгоритму роботи програмного модуля фільтрації образливих повідомлень

Алгоритм — це чітко визначена послідовність дій, спрямованих на досягнення певного результату. У контексті модуля фільтрації образливих повідомлень алгоритм забезпечує послідовну обробку повідомлень, їх аналіз на токсичність і прийняття рішення про подальшу обробку або блокування. Графічне подання алгоритму здійснюється за допомогою блок-схеми, де оператори зображуються у вигляді геометричних фігур, а стрілками вказується напрямок виконання [15].

Алгоритм також включає в себе обробку даних та прийняття рішень на основі певних правил або критеріїв. Він може бути покращений з часом шляхом аналізу результатів і коригування дій для досягнення кращої ефективності. Такий підхід робить алгоритм більш гнучким і здатним адаптуватися до змінних умов.

Випадок 1: Обробка повідомлення без Hate Speech

1. **Початок**

Ініціація сесії користувача в чаті, запуск необхідних серверних сервісів (WebSocket, фільтрація тощо). Система готова приймати повідомлення.

2. **Надсилання повідомлення користувачем**

Користувач вводить текст повідомлення у чаті та натискає кнопку надсилання. Повідомлення передається на сервер через WebSocket-з'єднання.

3. **Отримання повідомлення сервером**

Сервер отримує повідомлення і ініціює його перевірку. Повідомлення одразу не публікується, а спочатку направляється до модуля фільтрації.

4. **Аналіз повідомлення модулем фільтрації**

Модуль фільтрації аналізує текст повідомлення за допомогою двох компонентів:

- Викликається API, щоб отримати оцінку токсичності.
- Здійснюється перевірка на наявність заборонених слів через вбудований словник.

5. **Оцінка: немає порушень**

На основі результатів перевірки визначається, що повідомлення не містить токсичних або образливих висловлювань:

- Значення токсичності є нижчим за допустимий поріг.
- Заборонені слова в тексті відсутні.

6. **Публікація повідомлення**

Оскільки порушень не виявлено, модуль фільтрації передає повідомлення на публікацію у загальному чаті. Повідомлення відображається всім користувачам у кімнаті.

7. **Кінець**

Жодних попереджень чи санкцій не застосовується. Система повертається в очікування нових повідомлень. Алгоритм завершується до надходження наступного повідомлення.

Випадок 2: Виявлення Hate Speech в повідомлені

Загальний алгоритм модерації повідомлень має такі етапи:

1. **Початок**

Алгоритм розпочинається з моменту запуску вебзастосунку або переходу користувача на головну сторінку публічного чату.

2. **Перехід на головну сторінку**

Користувач відкриває головну сторінку, яка містить інтерфейс чату з можливістю надсилання повідомлень у реальному часі через WebSocket.

3. **Надсилання повідомлення**

Користувач вводить текст повідомлення та надсилає його, після чого повідомлення передається на сервер.

4. **Спрацьовує модуль фільтрації**

Повідомлення обробляється модулем машинного навчання, який виконує аналіз тексту та визначає наявність порушень (токсичних, образливих чи неприйнятних фраз).

5. **Перевірка наявності порушень**

На цьому етапі приймається рішення: якщо в повідомленні немає порушень - воно одразу публікується у чаті. Якщо є порушення - застосовується подальша логіка модерації.

6. **Збільшення лічильника порушень користувача**

У разі фіксації порушення лічильник порушень для даного користувача збільшується на 1.

7. **Оцінка кількості порушень**

Система перевіряє, скільки порушень було зафіксовано:

- При першому порушенні користувач отримує попередження.
- При другому порушенні надсилається ще одне попередження.
- При третьому порушенні система автоматично видає тимчасовий бан користувачеві.

8. **Публікація або блокування повідомлення**

Якщо повідомлення не містить порушень, воно публікується у чаті, і користувач бачить підтвердження отримання. Якщо було виявлено порушення - повідомлення блокується, а користувач отримує попередження або бан відповідно до кількості попередніх порушень.

9. **Кінець**

У разі накладення тимчасового бану користувач більше не може надсилати повідомлення. Якщо бану не було, користувач продовжує взаємодію з чатом, і цикл повторюється з кроку 3.

Алгоритм двох випадків зображено на рисунку 2.1.

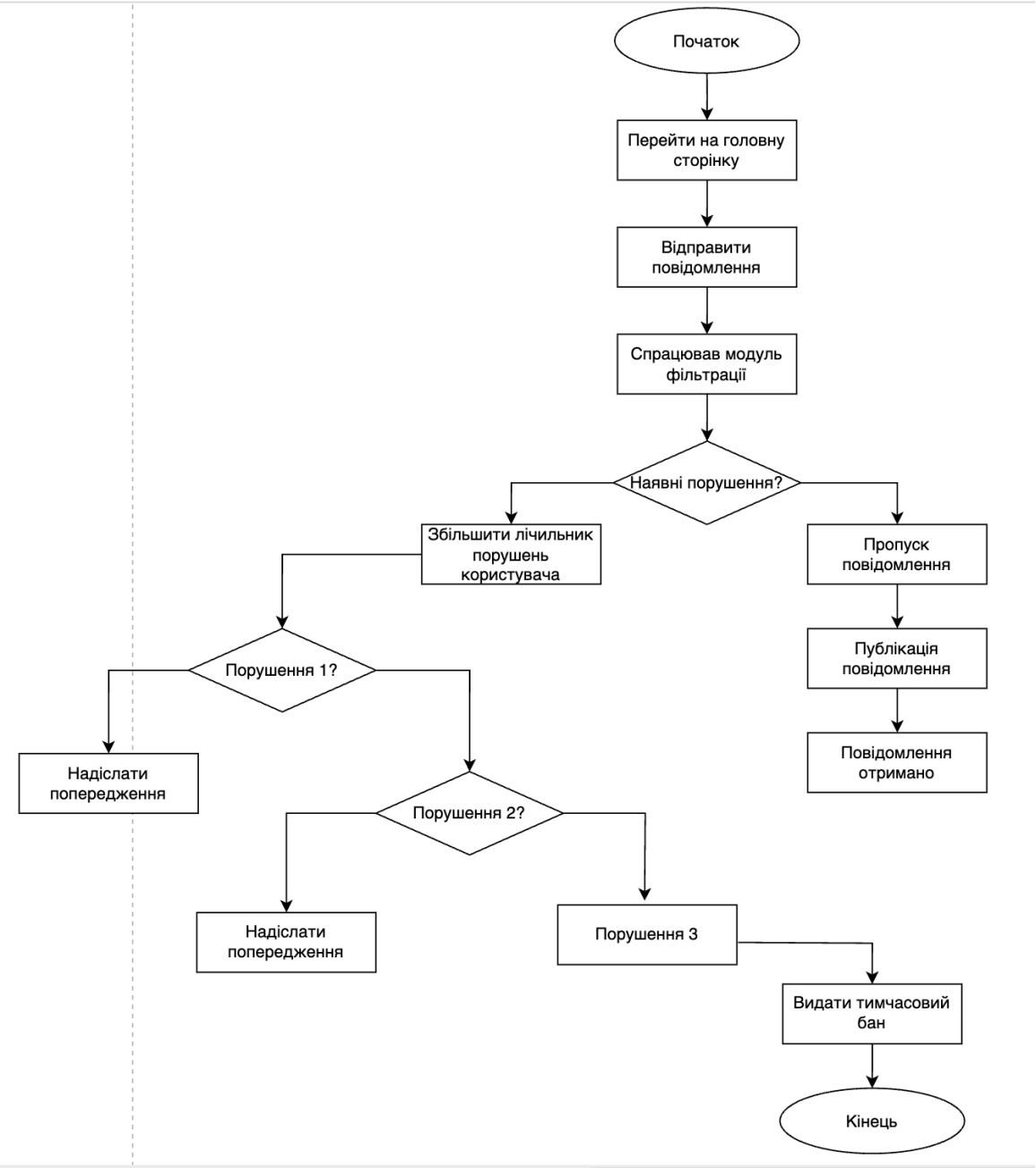


Рисунок 2.1 – Схема алгоритму роботи програмного модуля фільтрування образливих повідомлень.

2.2 Розробка структури програмного модуля автоматизованої фільтрації образливих повідомлень

Програмний модуль фільтрації токсичних повідомлень складається з двох основних частин: Telegram-бота, який інтегрується у групові чати, та вебзастосунку, що реалізує функціональність обміну повідомленнями між користувачами через браузер у реальному часі. Вебзастосунок в свою чергу можна поділити на 6 основних компонентів: реєстрація та авторизація, публічний чат, приватний чат, редагування профілю, фільтрування образливих повідомлень та машинне навчання фільтру.

Telegram-бот для фільтрації токсичності.

Telegram-бот написаний за допомогою бібліотеки node-telegram-bot-api та використовує Perspective API, що надається Google, для аналізу рівня токсичності повідомлень. Бот працює в режимі реального часу: він автоматично перевіряє кожне повідомлення, яке надсилається в груповому чаті, і відповідно до рівня токсичності виконує одне з кількох дій:

- Toxicity 0.5–0.6: повідомлення приховується та замінюється кнопкою для перегляду; при натисканні – відправляється в особисті повідомлення користувача.
- Toxicity 0.6–0.85: повідомлення видаляється, а користувач отримує попередження. При трьох попередженнях – тимчасова заборона на надсилання повідомлень.
- Toxicity 0.85+: користувач отримує тимчасове блокування на 2 хвилини, повідомлення видаляється.

Усі дії бот виконує автоматично: видалення, надсилання попереджень, блокування, відповіді на callback-кнопки. Такий підхід дозволяє модерувати чати без участі адміністратора.

Вебзастосунок із фільтрацією повідомлень.

Окрім бота, модуль фільтрації також інтегрований у вебзастосунок, який реалізує функціональність обміну повідомленнями між користувачами. Застосунок побудований на PHP та використовує WebSocket-сервер на основі бібліотеки Ratchet. Сервер дозволяє забезпечити миттєву доставку повідомлень між користувачами без необхідності постійного оновлення сторінки.

Сценарій роботи наступний:

- Користувач заходить на вебсторінку чату, яка під'єднується до WebSocket-сервера.
- При надсиланні повідомлення, воно:
  - перевіряється на токсичність (через інтеграцію з Perspective API або власний PHP-фільтр),
  - зберігається у реляційній базі даних, яка також використовується для зберігання історії чату.
- Сервер відправляє перевірене повідомлення до всіх під'єднаних клієнтів у режимі реального часу.

База даних зберігає інформацію про користувачів, повідомлення та рівень токсичності кожного повідомлення. Це дозволяє створити аналітичну систему або панель адміністратора для перегляду статистики токсичних повідомлень, часу активності, тощо [16].

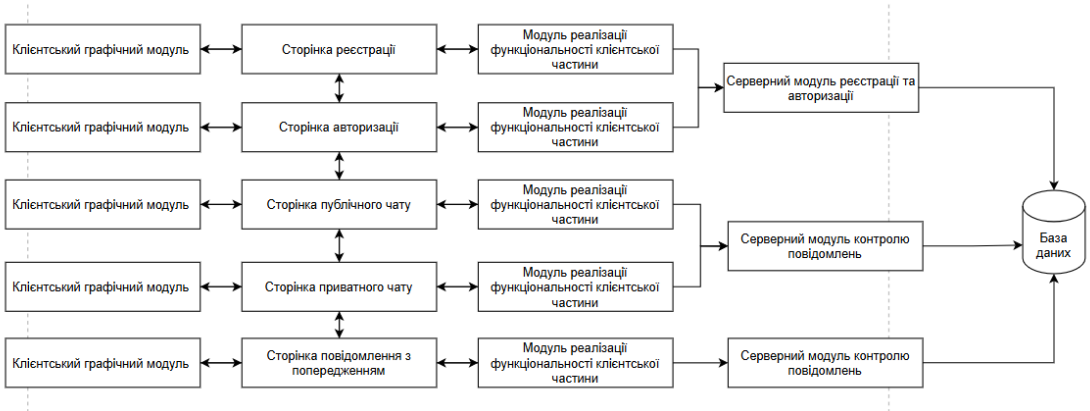


Рисунок 2.2 – Загальна структурна схема функціонування програмного модуля.

Компонентна структура вебзастосунку.

Вебзастосунок умовно поділений на такі модулі:

- Модуль автентифікації - сторінки реєстрації та входу, що працюють із відповідним PHP-обробником.
- Модуль публічного чату - головна сторінка чату, що використовує WebSocket-підключення для миттєвого обміну повідомленнями.
- Модуль модерації - фонові процеси або скрипти на сервері, що аналізують повідомлення та зберігають результат у базу.
- Модуль приватного чату – сторінка з чатом, що використовує WebSocket-підключення та має логіку приватного чату.
- Модуль машинного навчання автоматизованого фільтру – відповідає за процес створення та навчання моделі виявлення образливих повідомлень.

У вебзастосунку реалізовано кілька ключових компонентів, кожен із яких виконує окрему функцію.

Модуль автентифікації відповідає за реєстрацію та вхід користувачів через відповідні PHP-обробники. Публічний чат реалізований як головна сторінка із WebSocket-з'єднанням, що забезпечує миттєвий обмін повідомленнями в загальнодоступному просторі. Приватний чат має подібну структуру, але дозволяє здійснювати спілкування між окремими користувачами у закритому режимі. Модуль модерації включає серверні скрипти, які обробляють повідомлення в реальному часі, аналізуючи їхню токсичність і зберігаючи відповідну інформацію в базу даних. Модуль машинного навчання реалізує процес підготовки та навчання моделі, що автоматично виявляє образливі повідомлення. Уся система працює автономно на власному сервері, що гарантує повний контроль над даними та безпеку з'єднань.

Для розпізнавання токсичних повідомлень використовується підхід із застосуванням нейронних мереж, зокрема модель типу BERT (Bidirectional Encoder Representations from Transformers), яка добре зарекомендувала себе у задачах обробки природної мови (NLP). Модель навчається на великому корпусі

анотаційованих даних, де кожне повідомлення має відповідну оцінку токсичності. У процесі підготовки дані очищуються, нормалізуються, токенізуються та конвертуються у числовий формат, зручний для подачі на вхід моделі.

Після етапу навчання модель здатна передбачати індекс токсичності нового повідомлення в діапазоні від 0 до 1, де 0 – повністю нейтральне повідомлення, а 1 – відверто образливе. Для зручності модерації система дозволяє налаштовувати порогове значення токсичності, при перевищенні якого повідомлення автоматично блокується або передається на ручну перевірку.

Усі запити (HTTP-запити для реєстрації, авторизації, WebSocket-з'єднання) проходять через власний сервер, без використання сторонніх сервісів, що дозволяє забезпечити повну контрольованість і захищеність системи.

Сумісне використання Telegram-бота і вебчату

Завдяки однаковим принципам роботи, Telegram-бот та вебзастосунок можуть бути інтегровані в єдину систему моніторингу. Наприклад, збереження токсичних повідомлень як із Telegram, так і з вебчату в єдину базу даних дозволяє здійснювати централізований аналіз поведінки користувачів.

Весь проєкт підтримує контейнеризацію, що дозволяє запускати вебсервер, WebSocket-сервер, Telegram-бота та базу даних в ізольованих середовищах за допомогою Docker. Такий підхід дозволяє масштабувати проєкт, запускати нові інстанси WebSocket-сервера або Telegram-бота при підвищеному навантаженні.

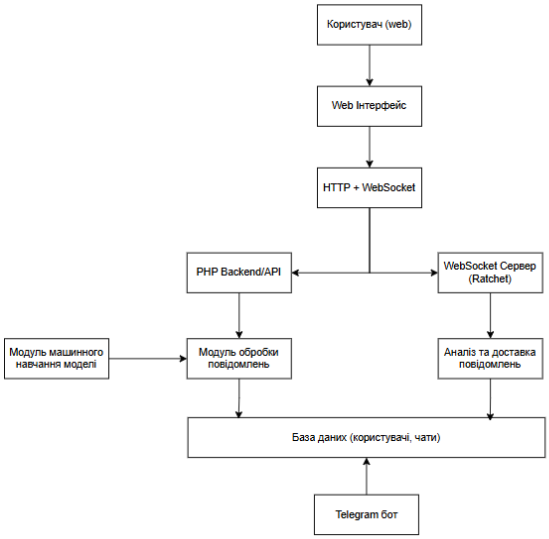


Рисунок 2.3 – Загальна структурна схема компонентів програмного модуля фільтрування образливого контенту.

2.3 Розробка UML-діаграми класів

Розробка сучасних вебзастосунків дедалі більше спирається на принципи об'єктно-орієнтованого програмування (ООП), що дозволяє забезпечити високу гнучкість, масштабованість і підтримуваність програмного коду. Особливо важливим це є у випадках розробки модулів, що потребують постійної взаємодії користувачів, складної логіки обробки повідомлень та безпечної авторизації. В контексті реалізації чат-системи з фільтрацією токсичних повідомлень, а також модулю авторизації з верифікацією через email, ООП виступає центральною парадигмою проєктування та реалізації програмного забезпечення.

Об'єктно-орієнтоване програмування ґрунтується на чотирьох основних принципах: інкапсуляція, наслідування, поліморфізм та абстракція.

Інкапсуляція дозволяє приховати внутрішні механізми роботи класів, відкриваючи лише необхідний для взаємодії інтерфейс. Це забезпечує безпечне та контрольоване використання об'єктів.

Наслідування дає можливість створювати нові класи на основі існуючих, що дозволяє повторно використовувати код.

Поліморфізм забезпечує здатність різних об'єктів взаємодіяти через спільний інтерфейс, що спрощує підтримку та розширення системи.

Абстракція дозволяє зосередитися лише на важливих характеристиках об'єкта, приховуючи деталі реалізації.

У серверній частині чат-системи з фільтрацією токсичних повідомлень реалізовано ключові класи: Chat, ChatRooms, ToxicityFilter, User, BanService та допоміжні компоненти.

Клас Chat відповідає за основну логіку WebSocket-з'єднання. Він отримує повідомлення, перевіряє їх на токсичність і надсилає очищені повідомлення учасникам кімнати. Основними методами є: onOpen, onMessage, onClose, onError, broadcast. Метод onMessage є ключовим, оскільки реалізує фільтрацію через ToxicityFilter перед трансляцією повідомлень.

Клас ChatRooms відповідає за зберігання активних кімнат, додавання/видалення користувачів, запис історії повідомлень та звернення до ToxicityFilter при додаванні нових повідомлень. Клас використовує методи addUserToRoom, removeUserFromRoom, getMessages, addMessage, getRoomUsers.

Клас ToxicityFilter взаємодіє з Perspective API та інкапсулює логіку визначення токсичності тексту. Він містить метод isToxic(message: string): boolean, який повертає булеве значення на основі оцінки повідомлення.

Клас User представляє зареєстрованого користувача системи та містить поля id, name, email, bannedUntil. Також є метод isBanned().

Клас BanService забезпечує логіку блокування користувачів на певний час у разі надсилання токсичних повідомлень. Методи: banUser(userId, duration), isUserBanned(userId).

На рисунку 1 зображено UML-діаграму класів серверної частини чат-системи:

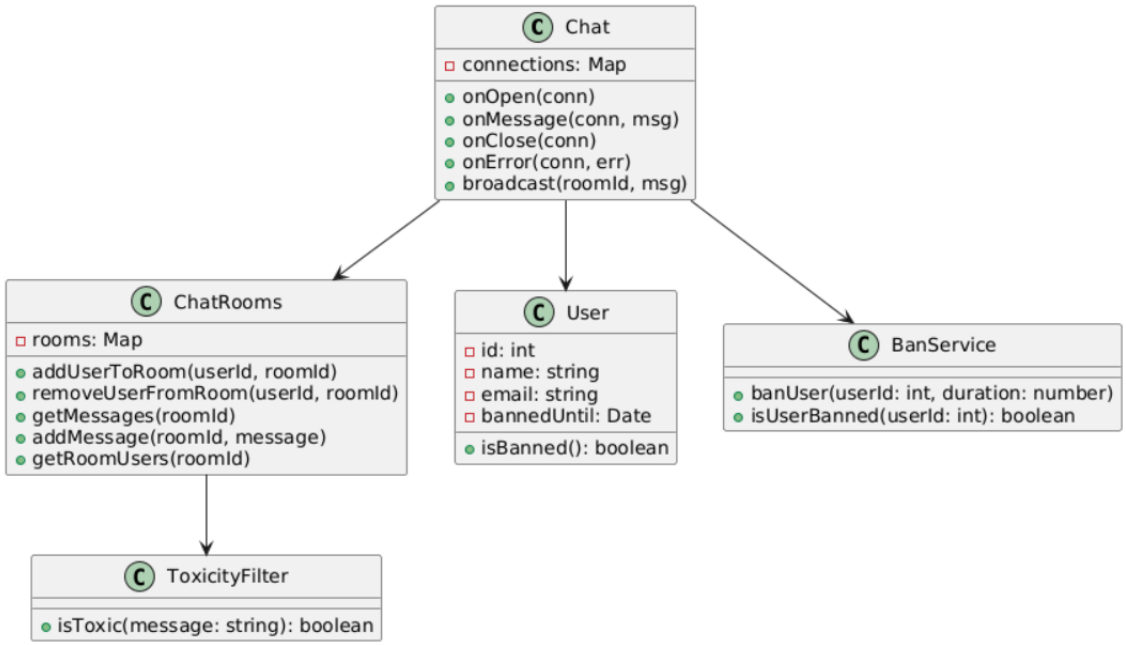


Рисунок 2.4 – UML-діаграма класів серверної частини модуля фільтрування образливих повідомлень.

У модулі авторизації реалізовані наступні ключові класи: AuthService, AuthController, UserService, TokenService, EmailService.

Клас AuthService обробляє логіку входу, реєстрації та оновлення токенів. Основні методи: signIn(email, password), signUp(email, password), refreshToken(refreshToken). Він використовує UserService для перевірки користувачів і TokenService для генерації токенів.

Клас AuthController — це HTTP-контролер, який приймає запити з клієнтської частини і викликає відповідні методи AuthService. Має методи: signIn(req), signUp(req), refreshTokens(req), getProfile(req).

Клас UserService містить методи для створення, пошуку та оновлення даних користувачів у базі.

Клас TokenService відповідає за створення та валідацію JWT токенів: generateTokens(user), verifyAccessToken(token), verifyRefreshToken(token).

Клас EmailService надсилає листи з підтвердженням реєстрації, відновленням пароля тощо.

На рисунку 2 зображено UML-діаграму класів модуля авторизації:

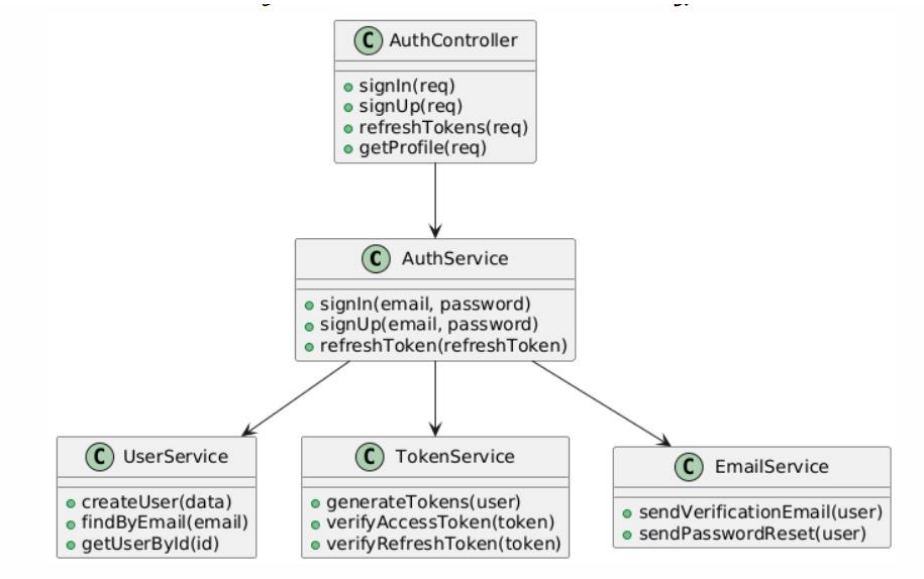


Рисунок 2.5 – UML-діаграма класів серверної частини модуля аутентифікації.

2.4 Висновок до розділу 2

У цьому розділі було розроблено детальну архітектуру програмного модуля для автоматизованої фільтрації образливих повідомлень у чаті. Описано ключові класи, що відповідають за прийом повідомлень, їх перевірку на токсичність, а також механізми авторизації користувачів і керування банами. Застосування об’єктно-орієнтованого підходу дозволило створити гнучку та зрозумілу структуру, що забезпечує легке розширення і підтримку системи.

Було створено UML-діаграми класів, які наочно демонструють взаємодію основних компонентів модуля - від обробки повідомлень до прийняття рішень щодо їх блокування або пропуску. Також розроблено покрокові алгоритми роботи з повідомленнями різного рівня порушень, що забезпечує послідовність і ефективність модерації.

Реалізований модуль суттєво підвищує якість комунікації, автоматизує процес модерації та створює безпечне середовище для користувачів. Така система є важливим внеском у розвиток сучасних чат-сервісів, де контроль контенту і захист від токсичних повідомлень мають першочергове значення. Обрана архітектура гарантує високу продуктивність, надійність та можливості для подальшого удосконалення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ФІЛЬТРУВАННЯ ОБРАЗЛИВИХ ПОВІДОМЛЕНЬ

3.1 Обґрунтування вибору мови та бібліотек програмування

Для розробки програмного модуля інтерактивної системи, що включає як вебчат із фільтрацією повідомлень, так і взаємодію з голосовим керуванням, особливо актуальним є правильний вибір мов програмування. Він має враховувати не лише технічні можливості конкретної мови, а й доступність бібліотек, підтримку сучасних технологій реального часу, зручність розгортання та масштабування. У цьому контексті розглянемо 4 мови програмування: Node.js (JavaScript), PHP та Python.

Вебчат на PHP із Ratchet та Python

PHP, як мова сценаріїв для розробки серверної логіки вебзастосунків, вже багато років залишається популярним завдяки простоті у використанні, широкій екосистемі та активній спільноті. Хоча PHP історично не асоціювався з розробкою у режимі реального часу, з появою бібліотеки Ratchet стало можливим створювати WebSocket-сервери на PHP. Ratchet є бібліотекою для асинхронної роботи з протоколом WebSocket, що дозволяє організовувати двостороннє спілкування між клієнтом і сервером без необхідності постійного опитування.

Для реалізації фільтрації токсичних повідомлень у чаті використано поєднання PHP та Python. Основна частина логіки чату побудована на PHP з Ratchet, тоді як модуль аналізу токсичності винесено окремо на Python. Це дозволило інтегрувати машинне навчання у вебчат: було створено та натреновано власну модель на основі корпусу українських коментарів, що дає змогу більш точно виявляти токсичні висловлювання в українськомовному середовищі. Python, завдяки своїм потужним ML-бібліотекам (зокрема scikit-learn, transformers, pytorch або tensorflow), забезпечує гнучкість і високу точність

класифікації повідомлень, тоді як PHP відповідає за обробку запитів і обслуговування клієнтів у режимі реального часу.

У проєкті Ratchet використано для забезпечення постійного з'єднання з клієнтами вебчату та обробки повідомлень у режимі реального часу. На відміну від традиційного PHP-застосунку, де кожен запит ініціює новий процес, Ratchet працює як постійно активний сервер, здатний обробляти десятки і навіть сотні з'єднань одночасно. Це дало змогу реалізувати швидке, інтерактивне середовище для обміну повідомленнями та одночасно здійснювати перевірку кожного повідомлення через вбудований модуль фільтрації токсичності на стороні сервера.

Ratchet використовує асинхронний цикл подій, побудований на основі ReactPHP — низькорівневої бібліотеки для асинхронного програмування. Це дозволило поєднати традиційну архітектуру PHP з можливостями обробки подій у реальному часі без значного відходу від мови, яка вже була в основі проєкту.

Telegram-бот на JavaScript (Node.js)

Для реалізації Telegram-бота було обрано мову JavaScript з використанням Node.js — середовища виконання, що дозволяє запускати JavaScript на сервері. На відміну від PHP, який традиційно орієнтований на запити HTTP, Node.js з самого початку розроблявся як високопродуктивне асинхронне середовище, ідеально підходить для задач реального часу та обробки великої кількості одночасних з'єднань.

Telegram-бот у цьому випадку не взаємодіє з браузером напряму, але активно обробляє потік повідомлень через Telegram API. Node.js надає потужні бібліотеки для роботи з Telegram-ботами та забезпечує швидку обробку запитів, що надходять від користувачів. Для реалізації фільтрації образливих повідомлень на боці бота було інтегровано API зовнішнього сервісу, який аналізує текст і повертає оцінку токсичності.

JavaScript було обрано не тільки через його продуктивність та підтримку асинхронних операцій через Promise і async/await, але й завдяки великій кількості готових модулів для роботи з Telegram, HTTP та сторонніми API. В результаті

реалізація Telegram-бота була простою та швидкою, із мінімальним накладом на ресурсну інфраструктуру.

Порівняння мов програмування PHP, JavaScript з python у цьому проєкті наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування PHP, JavaScript та python

Критерій	PHP (з Ratchet)	Node.js (JavaScript)	Python
Підтримка WebSocket	Через Ratchet + ReactPHP	Нативна (ws, socket.io)	Через aiohttp, websockets, менш стабільно
Асинхронність	Обмежено через ReactPHP	Нативна, потужна подієва модель	Наявна через asyncio, але складніша в налаштуванні
Telegram API	Вимагає сторонніх обгорток, складно	Добре підтримується через node-telegram-bot-api	Добре підтримується (python-telegram-bot, telebot)
Продуктивність для великої кількості з'єднань	Середня	Висока	Середня
Простота інтеграції з фронтендом	Висока (PHP традиційно веборієнтований)	Посередня	Посередня
Наявність бібліотек для фільтрації тексту	Обмежено, часто через API	Широкий вибір, легко підключати API	Також широкий вибір бібліотек і API
Стабільність і масштабованість	Висока (за умови правильного налаштування Ratchet)	Висока	Середня, залежить від обраного фреймворку
Висхідна архітектура в проєкті	Уже використовується в бекенді сайту	Найкраще підходить для клієнтських ботів і API	Додаткове середовище, що ускладнює підтримку

Зіставивши всі критерії, можна дійти до логічного висновку: для серверної частини вебчату оптимальним є використання PHP із Ratchet, оскільки PHP вже використовується як основа серверної архітектури проєкту, має добру інтеграцію з існуючим бекендом і дозволяє реалізувати WebSocket-сервер через Ratchet. Це дозволяє уникнути додаткових накладних витрат на підтримку ще одного середовища.

Для Telegram-бота найкращим вибором є JavaScript на Node.js, оскільки ця платформа забезпечує нативну асинхронну модель, має чудову підтримку Telegram API та дозволяє легко взаємодіяти зі сторонніми сервісами фільтрації текстів.

Python у межах цього проєкту використовується для реалізації модулю машинного навчання, який відповідає за класифікацію токсичних повідомлень на основі попередньо натренованої моделі. Таким чином, кожна технологія використовується у своїй сильній стороні: PHP — для серверної частини з WebSocket, Node.js — для інтеграції з Telegram, а Python — для обробки текстів за допомогою ML.

3.2 Обґрунтування вибору середовища для управління організованою базою даних

Системи керування базами даних (СКБД), що працюють із мовою SQL (Structured Query Language), становлять невід'ємну частину більшості сучасних інформаційних систем. Серед великої кількості доступних СКБД особливої уваги заслуговують MySQL та PostgreSQL — обидві є потужними, поширеними і відкритими рішеннями для зберігання, доступу та обробки структурованих даних. Проте кожна з них має свої особливості, які суттєво впливають на вибір у контексті конкретного проєкту.

MySQL — це одна з найпопулярніших у світі систем керування базами даних, яка набула широкого використання завдяки своїй простоті, надійності, високій швидкодії при виконанні стандартних SQL-запитів і широкій підтримці

в різноманітних інструментах і середовищах розробки. MySQL використовується в безлічі вебсервісів та застосунків, зокрема в поєднанні з PHP, Node.js, Python та іншими мовами. Завдяки широкій документації, стабільності версій, сумісності з популярними хостингами та чудовій інтеграції з фреймворками (Laravel, Symfony, Yii, Express.js тощо), MySQL став де-факто стандартом у веброзробці.

PostgreSQL, у свою чергу, позиціонується як «найпросунутіша об'єктно-реляційна СКБД з відкритим кодом». Вона підтримує багаті можливості, серед яких — робота з JSON, транзакційна цілісність, CTE-запити, віконні функції, вбудована реплікація, підтримка розширень і можливість створювати користувацькі типи даних. У багатьох випадках PostgreSQL пропонує більш строгий підхід до стандартів SQL і глибший контроль над транзакціями та обробкою складних запитів. Це робить її ідеальним вибором для складних систем із високими вимогами до консистентності даних, складної логіки або обробки великих обсягів нереляційної інформації, як-от графів чи документів.

Проте при порівнянні обох систем в умовах веборієнтованого проєкту, що фокусується на продуктивній обробці запитів до класичної реляційної структури даних, MySQL демонструє себе з практичнішого боку. У типових CRUD-операціях (створення, читання, оновлення, видалення), які виконуються в контексті користувацького чату, системи голосування, обробки повідомлень або автентифікації, MySQL діє значно швидше, особливо коли використовується рушій InnoDB, що забезпечує транзакції, зовнішні ключі та блокування на рівні рядків. Крім того, MySQL має нижчий поріг входу для початківців, зручніші інструменти адміністрування, такі як phpMyAdmin, і загалом краще підходить для швидкого розгортання невеликих та середніх проєктів.

Порівняно з SQLite, яка теж належить до СКБД, але є вбудованою і не працює у клієнт-серверному режимі, MySQL має значну перевагу в масштабованості, одночасному доступі кількох клієнтів і підтримці розподілених систем. SQLite, хоч і ідеально підходить для мобільних або локальних застосунків, не здатна задовольнити потреби повноцінної вебсистеми

з активною взаємодією користувачів, зокрема в чатах, форумах чи онлайн-іграх, де необхідно обробляти сотні запитів за секунду.

У таблиці 3.2 наведено порівняння основних характеристик MySQL, PostgreSQL, SQLite.

Таблиця 3.2 – Порівняння MySQL, PostgreSQL та SQLite.

Характеристика	MySQL	PostgreSQL	SQLite
Модель розгортання	Клієнт-сервер	Клієнт-сервер	Вбудована, файлова система
Тип бази даних	Реляційна	Об’єктно-реляційна	Реляційна
Продуктивність на простих запитах	Висока, особливо на SELECT	Висока, але трохи повільніша через строгі обмеження	Висока, але не масштабована на кількох клієнтів
Підтримка транзакцій	Так (через InnoDB)	Повна підтримка, з ACID-гарантіями	Обмежена, лише для простих транзакцій
Сумісність із SQL-стандартом	Часткова	Висока відповідність SQL:2011	Часткова
Підтримка JSON	Є (обмежена, як текстовий тип)	Повноцінна підтримка JSON, JSONB	Є, але базова
Масштабованість	Висока (кластеризація, реплікація)	Висока (власна реплікація, sharding, розширення)	Низька
Паралельні з’єднання	Висока кількість одночасних клієнтів	Дуже висока, підтримка багатьох потоків	Низька, не підходить для багатокористувацьких систем
Простота встановлення	Висока	Складніша конфігурація	Найпростіша — один файл
Зручність для початківців	Висока, простий синтаксис, велика спільнота	Складніший підхід, вища крива навчання	Найвища — підходить для навчання та простих проєктів

Продовження таблиці 3.2

Характеристика	MySQL	PostgreSQL	SQLite
Підтримка зовнішніх ключів	Так (у InnoDB)	Так (строгі правила цілісності)	Частково
Використання у веброзробці	Дуже поширене (особливо з PHP, Node.js)	Широке, але більше в складних або наукових системах	Рідко використовується у продакшн-веброзробці
Файл бази даних	Серверна система, зберігання на диску	Серверна система, зберігання на диску	Один файл .db

Таким чином, виходячи з конкретних вимог до системи — обробка повідомлень, фільтрація контенту, робота в реальному часі, підтримка багатьох з'єднань та надійне збереження інформації — найбільш доречним вибором є саме MySQL. Ця система чудово інтегрується як із PHP (для серверної частини), так і з Node.js (через відповідні драйвери, наприклад mysql2), дозволяючи створювати потужну, ефективну та масштабовану архітектуру. Завдяки зрозумілому синтаксису SQL-запитів, широкій підтримці спільноти та високій продуктивності в реальних умовах, MySQL є оптимальним рішенням для реалізації проєкту з вимогами до швидкої взаємодії з базою даних, а також забезпечення стабільності та надійності збереження користувацької інформації.

3.3 Обґрунтування вибору середовища розробки

Процес розробки програмного забезпечення нерозривно пов'язаний з використанням спеціалізованих інструментів, які допомагають ефективно писати, аналізувати та налагоджувати програмний код. Одним із ключових таких інструментів є інтегроване середовище розробки (IDE – Integrated Development Environment). IDE поєднує в собі редактор коду, засоби автодоповнення, підсвічування синтаксису, відлагоджувач, термінал, а також додаткові

інструменти для керування залежностями, версіями коду та взаємодії з базами даних. Використання якісного IDE дає змогу зменшити кількість помилок на ранніх етапах розробки, підвищити продуктивність і забезпечити комфортну роботу з кодом.

На сучасному ринку існує безліч інтегрованих середовищ, кожне з яких має свої сильні сторони, орієнтовані на конкретні мови або типи застосунків. Серед найпопулярніших редакторів і IDE, які були розглянуті в межах цього проєкту, — Visual Studio Code, PHPStorm та Atom. Кожне з них було проаналізовано за критеріями зручності, швидкодії, функціональності та відповідності до потреб конкретного програмного модуля та їх порівняльна характеристика була наведена в таблиці 3.3.

Таблиця 3.3 – Характеристика середовищ розробки

Характеристика	Visual Studio Code	PHPStorm	Atom
Вартість	Безкоштовний	Платний (преміум)	Безкоштовний
Продуктивність	Висока	Висока, оптимізована для PHP	Середня, залежить від розширень
Підтримка PHP	Через розширення	Повна нативна підтримка	Мінімальна через додаткові пакети
Підтримка JavaScript/TypeScript	Відмінна	Добра, але другорядна	Обмежена
Налагодження та тестування	Вбудований відлагоджувач	Повноцінне налагодження для PHP	Обмежене
Робота з базами даних	Через плагіни	Вбудований інтерфейс	Через розширення
Кросплатформеність	Так	Так	Так
Швидкість запуску	Дуже швидкий	Трохи повільніший	Швидкий
Гнучкість налаштувань	Висока	Висока	Висока

На основі цієї порівняльної характеристики було прийнято рішення застосовувати різні середовища розробки для окремих частин програмного проєкту. Для реалізації бекенд-модуля, який написаний мовою PHP та активно взаємодіє з базами даних, ідеальним вибором став PHPStorm. Ця IDE має глибоку інтеграцію з мовою PHP, інтелектуальний аналіз коду, налагоджувач, а також зручні інструменти для роботи з SQL-запитами. Незважаючи на те, що PHPStorm є платним програмним продуктом, для цього проєкту це не стало проблемою — автор має доступ до ліцензійної версії, що дозволяє повною мірою скористатися всіма функціональними перевагами IDE [18].

Натомість для фронтенд-частини застосунку, реалізованої на JavaScript/TypeScript, обрано Visual Studio Code. Цей редактор чудово підходить для роботи з вебтехнологіями, має величезну екосистему плагінів, швидкий запуск, інтегрований термінал та підтримку систем контролю версій. Завдяки своїй легкості та безкоштовному розповсюдженню, VS Code забезпечує комфортне середовище для роботи над інтерфейсом користувача та логікою клієнтської частини [17].

3.4 Розробка ER-моделі бази даних

Для забезпечення збереження та ефективного управління даними в чаті з фільтрацією образливих повідомлень було розроблено ER-модель бази даних. Основними сутностями, що формують структуру даних, є: користувачі, повідомлення, персональні повідомлення, а також попередження, які система видає за порушення правил спілкування.

Універсальне відношення включає такі атрибути:
R (id_користувача, ім’я, email, пароль, аватар, роль, id_повідомлення, текст, час, токсичність, id_персонального_повідомлення, id_отримувача, id_попередження, дата).

Загальна ступінь відношення — 14.

У таблиці 3.4 наведено основні типи зв’язків між сутностями:

Таблиця 3.4 – Характеристики відношень між сутностями програмного модуля фільтрування образливих повідомлень

Ім’я сутності 1	Ім’я сутності 2	Тип зв’язку	Клас належності
Користувачі	Повідомлення	1:Б	Обов’язковий
Користувачі	Персональні повідомлення	1:Б	Обов’язковий
Користувачі	Попередження	1:Б	Необов’язковий
Персональні повідомлення	Отримувач	1:Б	Обов’язковий

Представлення ER-моделі для бази даних програмного модуля фільтрування образливих повідомлень зображено на рисунку 3.1.

Рисунок 3.1 – ER-модель бази даних модуль фільтрування образливих повідомлень.

3.5 Розробка основних модулів програмного забезпечення

Для реалізації Telegram-бота на Node.js було використано бібліотеку node-telegram-bot-api, яка забезпечує простий інтерфейс для взаємодії з Telegram API. Бот ініціалізується через клас TelegramBot з параметром polling, що дозволяє йому постійно опитувати сервер на предмет нових повідомлень. Основна логіка обробки повідомлень реалізована у функції, що підписується на подію «message». Ця функція отримує об'єкт повідомлення, з якого витягуються

важливі дані: ідентифікатор чату, текст повідомлення, ідентифікатор користувача та інші.

Для визначення токсичності тексту використовується асинхронна функція checkToxicity, яка надсилає HTTP POST-запит до Google Perspective API з текстом повідомлення та отримує у відповідь числове значення токсичності. Це значення використовується для визначення подальших дій бота. Якщо токсичність незначна, повідомлення ігнорується або допускається, якщо середня — воно видаляється, а користувачу надсилається попередження з лічильником порушень. При високій токсичності повідомлення видаляється, а користувач блокується на певний час [19].

Крім того, для зручності модерації реалізована обробка callback-запитів — це дозволяє відправити користувачу кнопку для перегляду потенційно токсичного повідомлення у приватних повідомленнях, при цьому видаляючи його із загального чату. Логіка роботи з callback-запитами реалізована через підписку на подію «callback_query», яка отримує дані натиснутої кнопки, а потім надсилає відповідне приватне повідомлення та видаляє повідомлення в чаті.

Таким чином, весь процес обробки повідомлень у бота складається з ініціалізації класу бота, функції перевірки токсичності через зовнішній API, обробки отриманих повідомлень із відповідними реакціями залежно від рівня токсичності та керування попередженнями користувачів. Наведений нижче фрагмент коду демонструє створення бота та перевірку токсичності:

```
const TelegramBot = require("node-telegram-bot-api");
const axios = require("axios");

const bot = new TelegramBot(TELEGRAM_TOKEN, { polling: true });

async function checkToxicity(message) {
  const url = `https://commentanalyzer.googleapis.com/v1alpha1/comments:analyze?key=${PERSPECTIVE_API_KEY}`;
```

```
    const requestData = {
        comment: { text: message },
        languages: ["ru", "en"],
        requestedAttributes: { TOXICITY: {} }
    };

    try {
        const response = await axios.post(url, requestData);
        return response.data.attributeScores.TOXICITY.summaryScore.value;
    } catch (error) {
        console.error("Error checking toxicity:", error);
        return null;
    }
}

bot.on("message", async (msg) => {
    const chatId = msg.chat.id;
    const text = msg.text;
    const user = msg.from;

    const toxicityScore = await checkToxicity(text);

    if (toxicityScore !== null) {
        if (toxicityScore >= 0.6) {
            await bot.deleteMessage(chatId, msg.message_id);
            // Додаткові дії — попередження, блокування
        }
    }
});
```

Цей підхід забезпечує ефективне відокремлення логіки роботи з Telegram API, обробки голосових чи текстових повідомлень та модерації контенту.

Наступним етапом реалізації системи чату є створення WebSocket-сервера на PHP із застосуванням бібліотеки Ratchet, яка забезпечує асинхронну обробку з'єднань у реальному часі. Основний клас Chat реалізує інтерфейс MessageComponentInterface, що визначає п'ять ключових методів: onOpen, onMessage, onClose, onError та конструктор.

```
php
КопироватьРедактировать
class Chat implements MessageComponentInterface {
    protected $clients;

    public function __construct() {
        $this->clients = new \SplObjectStorage;
        echo 'Server Started';
    }
    ...
}
```

У конструкторі створюється контейнер для збереження активних з'єднань клієнтів, реалізований через об'єкт SplObjectStorage. Це дозволяє підтримувати список усіх підключених користувачів для подальшої взаємодії.

Метод onOpen виконується при встановленні нового WebSocket-з'єднання. На цьому етапі сервер отримує із запиту токен користувача, який використовується для ідентифікації та оновлення інформації про з'єднання в базі даних. Після підтвердження підключення, усім активним клієнтам розсилається повідомлення про зміну статусу користувача на "Online".

```
php
КопироватьРедактировать
public function onOpen(ConnectionInterface $conn) {
    $this->clients->attach($conn);
```

```
        $querystring = $conn->httpRequest->getUri()->getQuery();
        parse_str($querystring, $queryarray);

        if(isset($queryarray['token'])) {
            $user_object = new \ChatUser;
            $user_object->setUserToken($queryarray['token']);
            $user_object->setUserConnectionId($conn->resourceId);
            $user_object->update_user_connection_id();

            $user_data = $user_object->get_user_id_from_token();
            $user_id = $user_data['user_id'];

            $data['status_type'] = 'Online';
            $data['user_id_status'] = $user_id;

            foreach ($this->clients as $client) {
                $client->send(json_encode($data));
            }
        }

        echo "New connection! ({$conn->resourceId})\n";
    }
```

Метод onMessage відповідає за прийом і обробку повідомлень від клієнтів. В залежності від типу повідомлення (визначеного командою command у JSON-структурі) реалізуються два сценарії: приватний чат та загальний чат.

Для приватного чату створюється об’єкт класу PrivateChat, у який передається інформація про відправника, отримувача, текст повідомлення та час надсилання. Повідомлення зберігається у базі даних і надсилається виключно учасникам приватної бесіди.

php

КопироватьРедактировать

```
if($data['command'] == 'private') {
    $private_chat_object = new \PrivateChat;
    $private_chat_object->setToUserId($data['receiver_userid']);
    $private_chat_object->setFromUserId($data['userId']);
    $private_chat_object->setChatMessage($data['msg']);
    $timestamp = date('Y-m-d h:i:s');
    $private_chat_object->setTimestamp($timestamp);
    $private_chat_object->setStatus('Yes');
    $chat_message_id = $private_chat_object->save_chat();

    $user_object = new \ChatUser;
    $user_object->setUserId($data['userId']);
    $sender_user_data = $user_object->get_user_data_by_id();

    $user_object->setUserId($data['receiver_userid']);
    $receiver_user_data = $user_object->get_user_data_by_id();

    $sender_user_name = $sender_user_data['user_name'];
    $data['datetime'] = $timestamp;
    $receiver_user_connection_id = $receiver_user_data['user_connection_id'];

    foreach($this->clients as $client) {
        if($from == $client) {
            $data['from'] = 'Me';
        } else {
            $data['from'] = $sender_user_name;
        }
```

```
            if($client->resourceId == $receiver_user_connection_id || $from == $client) {
                $client->send(json_encode($data));
            } else {
                $private_chat_object->setStatus('No');
                $private_chat_object->setChatMessageId($chat_message_id);
                $private_chat_object->update_chat_status();
            }
        }
    }
```

У випадку загального чату створюється об’єкт ChatRooms, що зберігає повідомлення, а після — транслює його всім підключеним користувачам із відповідним позначенням автора.

```
php
КопироватьРедактировать
else {
    $chat_object = new \ChatRooms;
    $chat_object->setUserId($data['userId']);
    $chat_object->setMessage($data['msg']);
    $chat_object->setCreatedOn(date("Y-m-d h:i:s"));
    $chat_object->save_chat();

    $user_object = new \ChatUser;
    $user_object->setUserId($data['userId']);
    $user_data = $user_object->get_user_data_by_id();
    $user_name = $user_data['user_name'];

    $data['dt'] = date("d-m-Y h:i:s");

    foreach ($this->clients as $client) {
```

```
        if($from == $client) {
            $data['from'] = 'Me';
        } else {
            $data['from'] = $user_name;
        }
        $client->send(json_encode($data));
    }
}
```

Метод onClose активується при закритті WebSocket-з'єднання клієнта. На цьому кроці відбувається оновлення статусу користувача у базі на "Offline" і оповіщення всіх клієнтів про зміну статусу. З'єднання вилучається зі списку активних.

```
php
КопироватьРедактировать
public function onClose(ConnectionInterface $conn) {
    $querystring = $conn->httpRequest->getUri()->getQuery();
    parse_str($querystring, $queryarray);

    if(isset($queryarray['token'])) {
        $user_object = new \ChatUser;
        $user_object->setUserToken($queryarray['token']);
        $user_data = $user_object->get_user_id_from_token();
        $user_id = $user_data['user_id'];

        $data['status_type'] = 'Offline';
        $data['user_id_status'] = $user_id;

        foreach($this->clients as $client) {
            $client->send(json_encode($data));
        }
```

```
    }

    $this->clients->detach($conn);
    echo "Connection {$conn->resourceId} has disconnected\n";
}
```

Метод onError відповідає за обробку помилок під час роботи сервера, логуючи повідомлення про помилки та закриваючи проблемні з'єднання. [20]

```
php
КопироватьРедактировать
public function onError(ConnectionInterface $conn, \Exception $e) {
    echo "An error has occurred: {$e->getMessage()}\n";
    $conn->close();
}
```

Таким чином, наведений клас Chat реалізує повноцінний WebSocket-сервер, що забезпечує обробку підключень, прийом і надсилання повідомлень у реальному часі, підтримку приватних і групових чатів, а також синхронізацію статусів користувачів.

У цьому фрагменті коду реалізується перевірка повідомлення на токсичність та його відправка в публічний чат. Коли користувач вводить текст і натискає кнопку відправки, запускається обробник події форми. Він спочатку перевіряє, чи повідомлення відповідає валідації форми через Parsley. Потім перевіряється, чи не було у користувача раніше порушень, які призвели до блокування.

```
$('#chat_form').on('submit', async function(event){
    event.preventDefault();

    if ($('#chat_form').parsley().isValid()) {
        var message = $('#chat_message').val();

        if (violations[user_id] >= 3) {
```

```
        alert("You have been blocked due to repeated violations.");
        return;
    }
```

Після цього викликається функція checkToxicity, яка асинхронно надсилає текст повідомлення на API Perspective для оцінки токсичності. Вона формує запит із текстом повідомлення і отримує у відповідь числовий показник токсичності. Якщо значення токсичності перевищує встановлений поріг (0.7), повідомлення вважається образливим, і користувачу виводиться відповідне повідомлення з проханням відредагувати текст [21].

```
    const isToxic = await checkToxicity(message);
    if (isToxic) {
        alert("Your message could be considered offensive. Please edit it.");

        violations[user_id] = (violations[user_id] || 0) + 1;

        if (violations[user_id] >= 3) {
            alert("You have been temporarily blocked from sending messages.");
        }

        return;
    }
```

Якщо ж повідомлення не містить токсичних елементів, воно упаковується у формат JSON із вказанням ідентифікатора користувача та текстом повідомлення, після чого відправляється на сервер WebSocket для трансляції в усім учасникам чату.

```
    var data = { userId: user_id, msg: message };
    console.log("Message sent:", data);

    conn.send(JSON.stringify(data));
    $('#messages_area').scrollTop($('#messages_area')[0].scrollHeight);
```

```
    }
});
```

Функція checkToxicity відповідає за звернення до зовнішнього сервісу Perspective API, який аналізує текст і повертає числову оцінку токсичності. У разі помилки під час виклику API або якщо оцінка нижча за поріг, функція дозволяє надсилати повідомлення без блокування.

```
async function checkToxicity(message) {
    const apiKey = "AIzaSyCcvILtOkLlrx6lnhcS83NO2BGhzf5WzNM";
    const url = `https://commentanalyzer.googleapis.com/v1alpha1/comments:analyze?key=${apiKey}`;

    const requestBody = {
        comment: { text: message },
        languages: ["en"],
        requestedAttributes: { TOXICITY: {} }
    };

    try {
        const response = await fetch(url, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify(requestBody)
        });

        const data = await response.json();
        const toxicityScore = data.attributeScores.TOXICITY.summaryScore.value;

        console.log(`Toxicity: ${toxicityScore}`);
```

```
        return toxicityScore >= 0.7;
    } catch (error) {
        console.error("Perspective API error:", error);
        return false;
    }
}
```

Таким чином, цей код забезпечує автоматичну модерацію вхідних повідомлень у чаті, блокуючи надсилання образливого контенту і захищаючи учасників від небажаного негативного спілкування.

Також важливо зазначити як саме відбувається підключення до websocket серверу:

Спочатку підключаються необхідні класи з бібліотеки Ratchet, а також підключається клас Chat, який реалізує логіку чату:

```
php
КопироватьРедактировать
use Ratchet\Server\IoServer;
use Ratchet\Http\HttpServer;
use Ratchet\WebSocket\WsServer;
use MyApp\Chat;

require dirname(__DIR__) . '/vendor/autoload.php';
```

Далі створюється об’єкт сервера, який слухає порт 8080. Конструктор IoServer::factory приймає обробник WebSocket-запитів, який складається з декількох шарів. Перший шар — це HTTP-сервер (HttpServer), який приймає HTTP-запити, другий — це WebSocket сервер (WsServer), який управляє протоколом WebSocket, а всередині нього ініціалізується екземпляр класу Chat, де реалізовано обробку повідомлень, підключень і відключень користувачів:

```
php
КопироватьРедактировать
```

```
$server = IoServer::factory(
    new HttpServer(
        new WsServer(
            new Chat()
        )
    ),
    8080
);
```

Останній рядок запускає безперервний цикл сервера, який очікує на підключення клієнтів та обробляє події у реальному часі:

```
php
КопироватьРедактировать
$server->run();
```

Таким чином, цей код піднімає WebSocket-сервер, до якого клієнти можуть підключатися по адресу ws://localhost:8080.

3.6 Тестування роботи програми та аналіз результатів

Тестування реалізованих імплементацій є ключовим етапом для перевірки працездатності, стійкості та зручності використання розробленої системи в реальних умовах. У процесі тестування було проаналізовано роботу як повністю PHP-реалізації чату з фільтрацією токсичних повідомлень, так і варіанта з використанням розробленого телеграм бота.

Після запуску системи користувач потрапляє на сторінку входу в чат, де необхідно пройти авторизацію. У випадку відсутності облікового запису можна зареєструвати нового користувача через відповідну форму, після чого система перенаправляє на головну сторінку чату (Рисунок 3.2 та Рисунок 3.3).

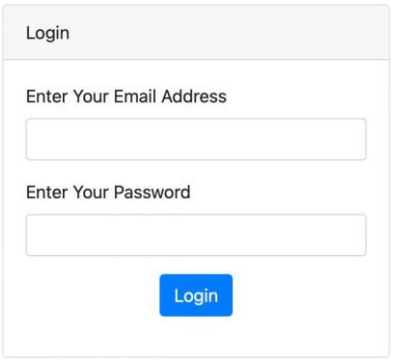


Рисунок 3.2 – Сторінка авторизації.

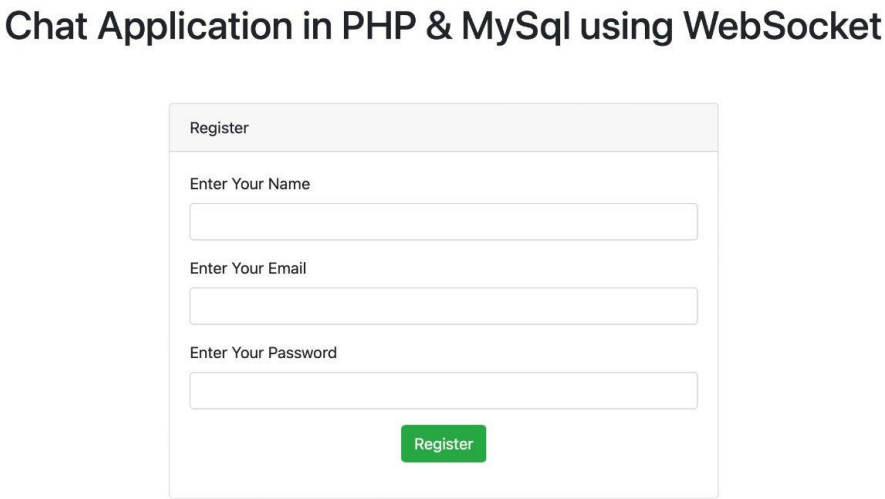


Рисунок 3.3 – Сторінка реєстрації.

Після авторизації користувач потрапляє в публічний чат, де має можливість спілкуватися з іншими зареєстрованими користувачами. Зовнішній вигляд чату адаптовано як під десктопні браузери, так і під мобільні пристрої. Для кожного повідомлення в чаті система перевіряє його зміст на наявність токсичних елементів (Рисунок 3.4 та Рисунок 3.5).

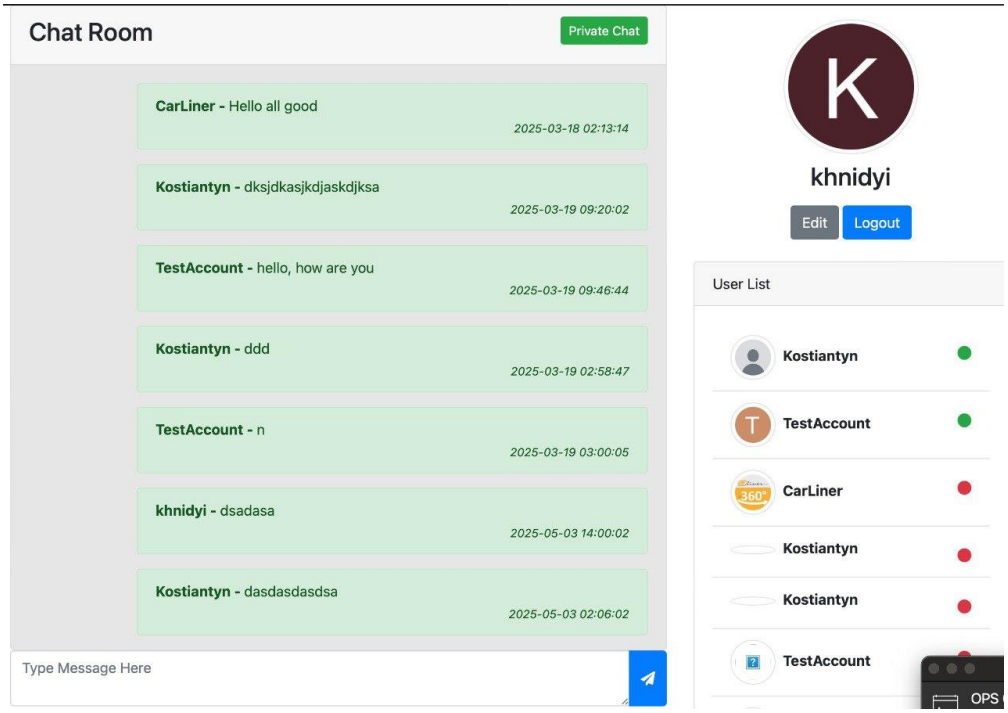
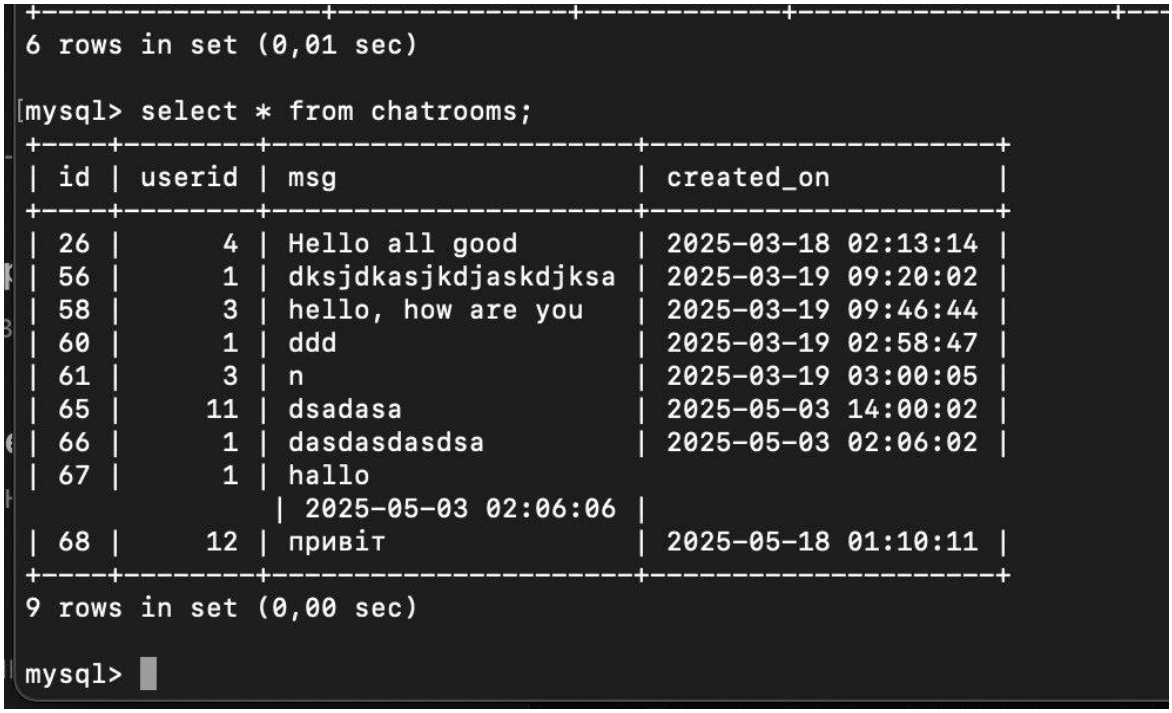


Рисунок 3.4 – Публічний чат.

Рисунок 3.5 – Результати тестування повідомлення на токсичний вміст, де Toxicity від 0 до 1.

У випадку, якщо повідомлення не містить порушень, воно зберігається в базу даних і транслюється в реальному часі всім підключеним користувачам. Якщо ж повідомлення містить образливі слова, його буде заблоковано і користувач отримає відповідне повідомлення про відхилення (Рисунок 3.6 та Рисунок 3.7).



```
+----------------+-------------+-----------+-----------------+---
6 rows in set (0,01 sec)

mysql> select * from chatrooms;
+----+--------+---------------------+---------------------+
| id | userid | msg                 | created_on          |
+----+--------+---------------------+---------------------+
| 26 |      4 | Hello all good      | 2025-03-18 02:13:14 |
| 56 |      1 | dksjdkasjkdjaskdjksa | 2025-03-19 09:20:02 |
| 58 |      3 | hello, how are you  | 2025-03-19 09:46:44 |
| 60 |      1 | ddd                 | 2025-03-19 02:58:47 |
| 61 |      3 | n                   | 2025-03-19 03:00:05 |
| 65 |     11 | dsadasa             | 2025-05-03 14:00:02 |
| 66 |      1 | dasdasdasdsa        | 2025-05-03 02:06:02 |
| 67 |      1 | hallo
                | 2025-05-03 02:06:06 |
| 68 |     12 | привіт              | 2025-05-18 01:10:11 |
+----+--------+---------------------+---------------------+
9 rows in set (0,00 sec)

mysql>
```

Рисунок 3.6 – збереження повідомлення у разі невиявлення порушень.

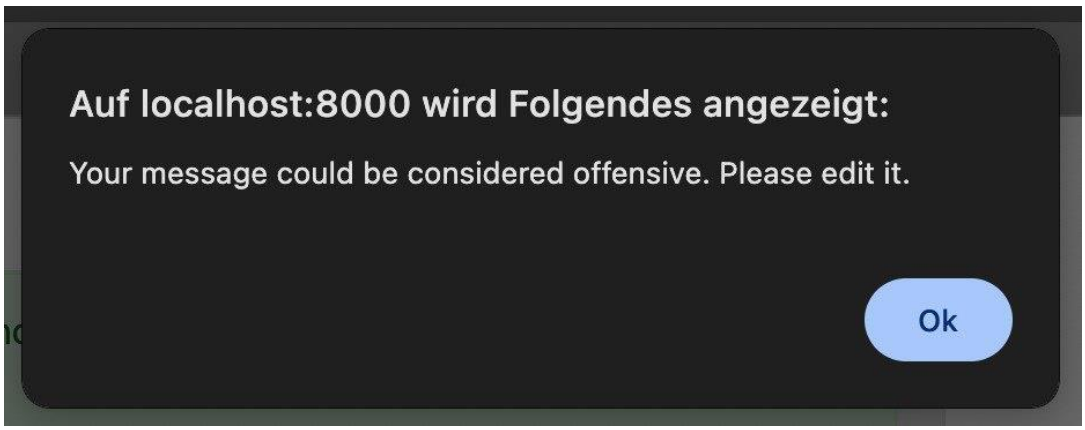


Рисунок 3.7 – блокування повідомлення у разі знаходження образливого вмісту (показник токсичності > 0.6).

Для реалізації з використанням Node.js (Телеграм бот) було протестовано коректність клієнтського з’єднання з Node-сервером, стабільність зв’язку при великій кількості клієнтів, а також правильність обміну повідомленнями між користувачами (Рисунок 3.8).

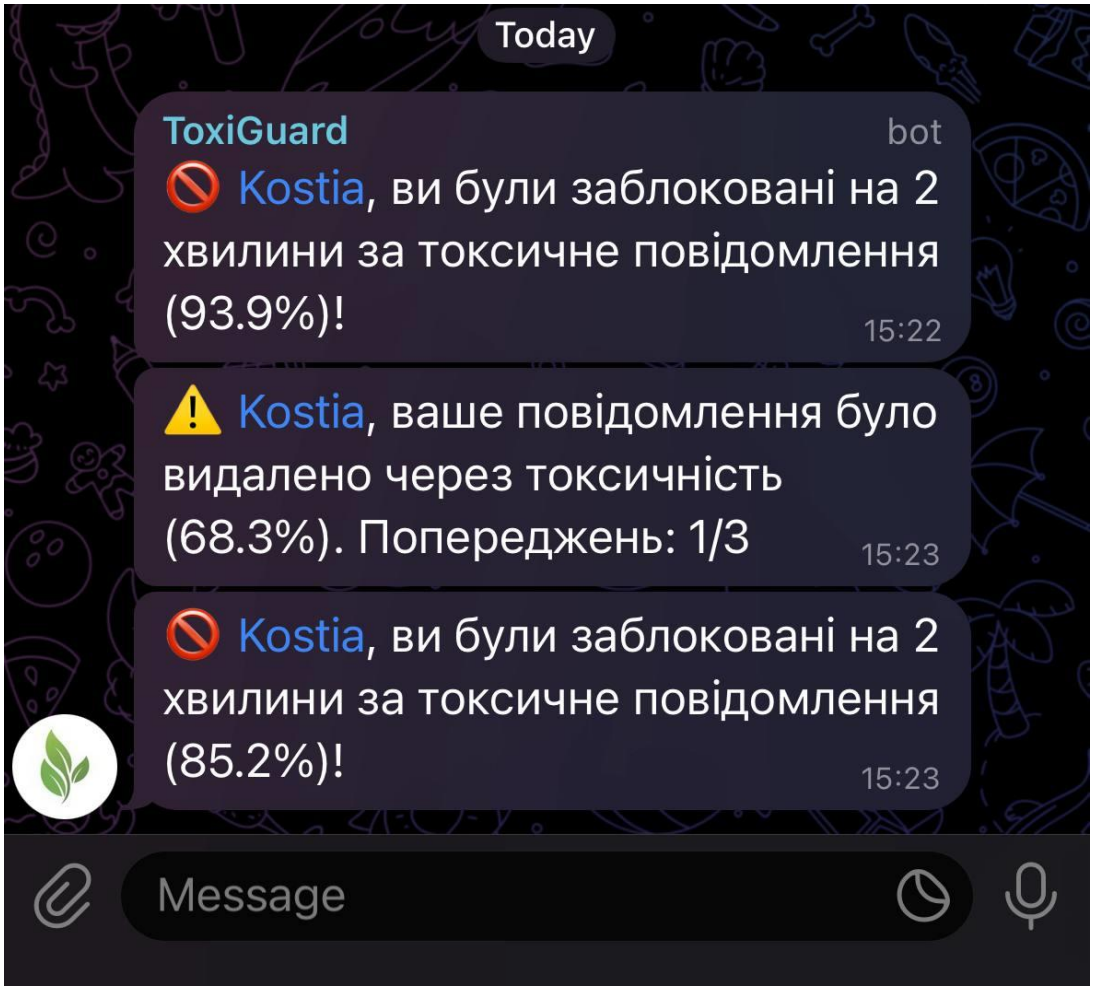


Рисунок 3.8 – Робота бота.

Для PHP-реалізації з Ratchet протестовано запуск сервера, прийом та обробку підключень, а також інтеграцію з фільтром повідомлень, що працює через Perspective API. Зокрема, перевірено, що повідомлення відправляються лише після проходження перевірки, і при повторному відкритті сторінки користувач бачить історію без порушень.

Розроблений програмний модуль фільтрації токсичних повідомлень був успішно протестований та відповідає всім поставленим вимогам. Було протестовано дві основні імплементації: повністю PHP-реалізацію з використанням Ratchet та WebSocket-сервер на Node.js у комбінації з клієнтськими застосунками. Система продемонструвала стабільну роботу, ефективну фільтрацію образливого контенту в режимі реального часу та високу точність виявлення токсичних повідомлень.

Крім того, для підвищення якості фільтрації ми розробили та навчили додаткову модель машинного навчання на Python, адаптовану спеціально для української мови.

Ця модель враховує особливості української лайки, а також розпізнає іронію та сарказм, що суттєво покращує якість модерації. Ми створили окремий модуль, який легко інтегрується з основним фільтром, розширюючи його функціональні можливості і перевершуючи деякі відомі аналоги.

Розширені можливості включають:

- Підтримку української лайки та образливих виразів, що не враховуються багатьма стандартними сервісами;
- Розпізнавання іронії та сарказму у повідомленнях, що дозволяє уникнути помилкових блокувань;
- Можливість інтеграції з існуючими системами як окремий сервіс;
- Покращену точність класифікації за рахунок комбінування правил та машинного навчання.

У результаті розробки вдалося реалізувати функціонал, який повністю відсутній у відомих аналогах. Зокрема, було впроваджено автоматичну модерацію чату з використанням Perspective API, адаптивний дизайн для мобільних пристроїв, підтримку української мови, а також розширювану систему правил для фільтрації повідомлень.

В таблиці 3.5 подано результати тестування розробленого програмного продукту та аналогів.

Таблиця 3.5 – результати тестування розробленого програмного продукту та аналогів

Критерій / Система	**Розроблений продукт**	**Microsoft Content Moderator**	**Detoxify**	**HateSonar**	**DeepAI Toxic Comment API**	**CleanSpeak**	**ToxiChat**
Платформа / Технологія	PHP (Ratchet), Python ML-модель	Azure AI	Python-бібліотека (трансформер)	Python-бібліотека (BERT-based)	Хмарний REST API	Локальний сервер/SaaS	Датасет (не сервіс)

Продовження таблиці 3.5

Критерій / Система	Розроблений продукт	Microsoft Content Moderator	Detoxify	HateSonar	DeepAI Toxic Comment API	CleanSpeak	ToxiChat
Підтримка української мови	Повна підтримка	Обмежена (тільки детекція мови)	Відсутня	Відсутня	Відсутня	Немає словників для української	Відсутня
Обробка токсичних повідомлень	Висока точність, підтримка іронії та сарказму	Базова (словники, OCR, AI)	Висока (англ. і декілька мов)	Середня (3 категорії: hate, offensive)	Середня (англ. текст)	Висока (кастомізована ML)	Немає (датасет)
Інтеграція	Модульна, з можливістю інтеграції в існуючі системи	Готові SDK, інтеграція в Azure екосистему	Локальна бібліотека Python	Локальна бібліотека Python	REST API	Локальний сервер або SaaS	Лише для навчання моделей
Анонімність модерації	Забезпечена, користувач не бачить заблокованих повідомлення	Часткова, UI для модераторів	Немає UI	Немає UI	Немає UI	Є UI	Немає UI
Підтримка мобільного/adaptive дизайну	Так	Частково	Ні	Ні	Ні	Залежить від інтеграції	Ні

Після аналізу та тестування можна зробити висновок, що запропоноване рішення значно розширює функціональність традиційних систем обміну повідомленнями, надаючи новий рівень безпеки та інклюзивності спілкування. Унікальна підтримка автоматичної фільтрації образливого контенту, кастомізації фільтрів та підтримки української мови дає підстави вважати систему інноваційним кроком у напрямку безпечного онлайн-спілкування.

Розроблений модуль може бути інтегрований у різноманітні онлайн-платформи: публічні чати, освітні середовища, корпоративні рішення чи

соціальні мережі. Завдяки своїй універсальності та технологічності він здатен адаптуватись під конкретні потреби кінцевих користувачів, забезпечуючи при цьому комфортну взаємодію без ризику зіткнення з небажаним контентом.

3.7 Висновок до розділу 3

Для реалізації програмного модуля автоматизованої фільтрації образливих повідомлень було обрано мову програмування PHP, що широко застосовується у веб-розробці та забезпечує стабільну роботу серверної частини. Для організації обміну повідомленнями в реальному часі використано WebSocket-сервер Ratchet, який дозволяє ефективно обробляти дані з мінімальною затримкою. Для клієнтської взаємодії застосовано сучасні веб-технології, що забезпечують зручність і швидкодію.

Обробка тексту та визначення його токсичності здійснюється через інтеграцію з API машинного навчання (Perspective API), а також розроблено і навчено власну модель для більш точного виявлення образливих висловлювань українською мовою. Такий підхід дозволяє впровадити адаптивну модерацію, що підвищує якість фільтрації та робить систему більш чутливою до контексту повідомлень.

Розроблений модуль має модульну структуру, що сприяє легкому масштабуванню та подальшому розвитку функціоналу, включаючи підтримку нових мов і складніших сценаріїв аналізу тексту. Проведене тестування підтвердило високу стабільність, точність виявлення токсичного контенту та ефективність роботи системи в реальному часі.

ВИСНОВОК

У дослідженні розглянуто проблему автоматизованої фільтрації образливих повідомлень у цифрових чатах, що є актуальним завданням у сучасних умовах значного зростання токсичного контенту в Інтернеті. Проведено аналіз існуючих методів і технологій фільтрації, серед яких особливу увагу приділено підходам із застосуванням машинного навчання, що дозволяють підвищити точність і адаптивність системи. З'ясовано, що існуючі рішення мають певні обмеження, зокрема у швидкодії або складності інтеграції, що вимагає розробки більш гнучких і ефективних модулів.

В роботі здійснено огляд відомих аналогів програмного забезпечення для автоматичної модерації, виділено їхні переваги й недоліки, що стало основою для формування вимог до власного модуля. Розроблено загальний алгоритм роботи, який інтегрує використання зовнішніх API на основі моделей машинного навчання, що дозволяє відмовитись від ресурсоємної розробки власної складної моделі і при цьому забезпечити високу якість фільтрації. Архітектура модуля спроєктована з урахуванням модульності та масштабованості, що полегшує подальше розширення функціоналу, зокрема, підтримку різних мов та розпізнавання складніших форм контенту, наприклад, іронії чи сарказму.

Реалізовано програмний модуль із використанням мови PHP та Ratchet WebSocket для забезпечення обміну повідомленнями в реальному часі, що підвищує ефективність роботи системи у динамічних середовищах. Проведене тестування підтвердило стабільність роботи, низьку затримку обробки повідомлень і високу точність виявлення образливого контенту, що робить розроблене рішення конкурентоспроможним щодо існуючих аналогів.

Таким чином, у роботі успішно поєднано сучасні підходи машинного навчання з практичною реалізацією модульної системи автоматичної фільтрації образливих повідомлень, що забезпечує ефективний захист користувачів чатів від токсичного контенту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Гнідий К. В., Арсенюк І. Р. Обґрунтування доцільності використання технології авто-матичного фільтрування образливих повідомлень у розробці веб-чатів. // Тези допові-дей LIV науково-технічної конференції факультету інтелектуальних інформаційних технологій та автоматизації (2025). – Вінниця: ВНТУ, 2025. Електронний ресурс (ре-жим доступу (https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23577/19502 (Дата звернення 27.05.2025)
2. Google’s AI to detect toxic comments can be easily fooled with ‘love’. URL: https://thenextweb.com/news/googles-hate-speech-ai-easily-fooled (Дата звернення 27.05.2025)
3. Різниця між іронія та сарказм. URL: https://dovidka.biz.ua/riznitsya-mizh-ironiya-ta-sarkazm/ (Дата звернення 27.05.2025)
4. Страшно смішно. Чому ми плутаємо іронію з сарказмом — і як її опанувати. URL: https://shorturl.at/322U9 (Дата звернення 27.05.2025)
5. ІМПЛІКАТУРИ Й ІРОНІЯ В КОНТЕКСТІ ШТУЧНОГО ІНТЕЛЕКТУ. URL: http://philologyjournal.lviv.ua/archives/16_2024/5.pdf (Дата звернення 27.05.2025)
6. A contextual-based approach for sarcasm detection. URL: https://www.nature.com/articles/s41598-024-65217-8 (Дата звернення 27.05.2025)
7. Adorjan M., Ricciardelli R. Relational aggression. Cyber-Risk and Youth. Abingdon, Oxon ; New York, NY : Routledge, 2019., 2018. P. 70–90. [Електронний ресурс] – Режим доступу: https://doi.org/10.4324/9781315158686-5 (date of access: 27.05.2025).
8. Microsoft Content Moderator [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/ai-services/content moderator/overview (дата звернення: 22.05.2025)

9. Detoxify [Електронний ресурс]. – Режим доступу: https://huggingface.co/unitary/toxic-bert (дата звернення: 27.05.2025)
10. HateSonar [Електронний ресурс]. – Режим доступу: https://github.com/Hironsan/HateSonar (дата звернення: 27.05.2025)
11. DeepAI Toxic Classification API [Електронний ресурс]. – Режим доступу: https://deepai.org/publication/toxicity-detection-with-generative-prompt-based-inference (дата звернення: 27.05.2025)
12. CleanSpeak [Електронний ресурс]. – Режим доступу: https://cleanspeak.com/ (дата звернення: 27.05.2025)
13. ToxiChat [Електронний ресурс]. – Режим доступу: https://github.com/abaheti95/ToxiChat (дата звернення: 27.05.2025)
14. Бондарчук, В.Ю., 2019. Порівняння методів аналізу тональності тексту (Doctoral dissertation, ВНТУ). (Дата звернення 27.05.2025)
15. UML для бізнес-моделювання: для чого потрібні діаграми процесів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (Дата звернення 27.05.2025).
16. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/ (Дата звернення 27.05.2025).
17. Visual Studio Code. URL: https://code.visualstudio.com/ (Дата звернення 27.05.2025).
18. PHPStorm: The PHP IDE by JetBrains. URL: https://www.jetbrains.com/phpstorm/ (Дата звернення 27.05.2025).
19. JavaScript Tutorial. URL: https://www.w3schools.com/js/DEFAULT.asp (Дата звернення 27.05.2025).
20. Ratchet: WebSockets for PHP. URL: http://socketo.me/ (Дата звернення 27.05.2025).
21. Perspective API. URL: https://perspectiveapi.com/ (Дата звернення 27.05.2025).
22. Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності А.А.Яровий, О.В.Сілагін, І.В.Богач.

ДОДАТКИ

ДОДАТОК А ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль автоматизованого фільтрування образливих повідомлень в чатах
Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)
Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)
Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,80 % .
Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

Яровий А.А., зав. каф. КН
(прізвище, ініціали, посада)
(підпис)

Колесницький О.К., проф. каф КН
(прізвище, ініціали, посада)
(підпис)

Особа, відповідальна за перевірку Озеранський В.С.
(підпис)
(прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник Арсенюк І.Р.
(підпис) (прізвище, ініціали, посада)

Здобувач Гнідий К.В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б ЛІСТИНГ ПРОГРАМИ

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from transformers import pipeline, AutoTokenizer,
AutoModelForSequenceClassification, Trainer, TrainingArguments
from datasets import load_dataset
import evaluate
import torch

app = Flask(__name__)
CORS(app)

# Білий список фраз
whitelist = [
    "дурень ти мій",
    "ну ти і дід"
]

def is_whitelisted(text):
    text_lower = text.lower().strip()
    for phrase in whitelist:
        if phrase in text_lower:
            return True
    return False

def train_model():
    print("Завантаження датасету...")
    ds = load_dataset("ukr-detect/ukr-toxicity-dataset-seminatural")
```

```
# Фільтрація None у тегах
ds = ds.filter(lambda example: example["tags"] is not None)

print("Приклад даних:", ds["train"][0])

model_name = "unitary/toxic-bert"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Binary classification (1 вихід), sigmoid буде використовуватись
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=1,
    ignore_mismatched_sizes=True
)

def preprocess_function(examples):
    tokenized = tokenizer(examples["text"], truncation=True,
padding="max_length", max_length=128)
    tokenized["label"] = [[float(tag)] for tag in examples["tags"]]  # <==
додано подвійні дужки
    return tokenized

tokenized_ds = ds.map(preprocess_function, batched=True)
tokenized_ds.set_format(type="torch", columns=["input_ids",
"attention_mask", "token_type_ids", "label"])

accuracy = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
```

```
        probs = torch.sigmoid(torch.tensor(logits)).numpy()
        preds = (probs > 0.5).astype(int)
        return accuracy.compute(predictions=preds, references=labels)

    training_args = TrainingArguments(
        output_dir="./model_output",
        evaluation_strategy="epoch",
        save_strategy="epoch",
        num_train_epochs=3,
        per_device_train_batch_size=16,
        save_total_limit=2,
        load_best_model_at_end=True,
        metric_for_best_model="accuracy",
        logging_dir='./logs',
        logging_steps=10,
        no_cuda=True,  # якщо без GPU
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_ds["train"],
        eval_dataset=tokenized_ds["validation"] if "validation" in tokenized_ds
else None,
        tokenizer=tokenizer,
        compute_metrics=compute_metrics
    )

    print("Початок тренування...")
```

```
        trainer.train()
        print("Збереження моделі...")
        trainer.save_model("./my_finetuned_model")
        tokenizer.save_pretrained("./my_finetuned_model")
        print("Тренування завершено.")
        return "./my_finetuned_model"

    # Тренуємо модель та створюємо pipeline
    model_path = "./my_finetuned_model"
    classifier = pipeline(
        "text-classification",
        model=model_path,
        tokenizer=model_path,
        return_all_scores=True,
        function_to_apply="sigmoid",  # обов'язково при num_labels=1
        device=0 if torch.cuda.is_available() else -1
    )

    @app.route('/check', methods=['POST'])
    def check_toxicity():
        data = request.get_json()
        text = data.get("text", "")

        if is_whitelisted(text):
            return jsonify({"toxic": False, "reason": "whitelist"})

        results = classifier(text)[0]  # список словників [{'label': 'LABEL_0',
'score': 0.834}, ...]
```

```
    # У binary model з sigmoid повертається один score
    score = results[0]['score']
    is_toxic = score > 0.5

    return jsonify({"toxic": bool(is_toxic), "score": float(score)})

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5005)
```

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ АВТОМАТИЗОВАНОГО ФІЛЬТРУВАННЯ ОБРАЗЛИВИХ ПОВІДОМЛЕНЬ У ЧАТАХ»

Виконав: студент 4 курсу, групи ЗКН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Гнідий К.В.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)

3 червня 2025 рік

Вінниця ВНТУ – 2025 рік

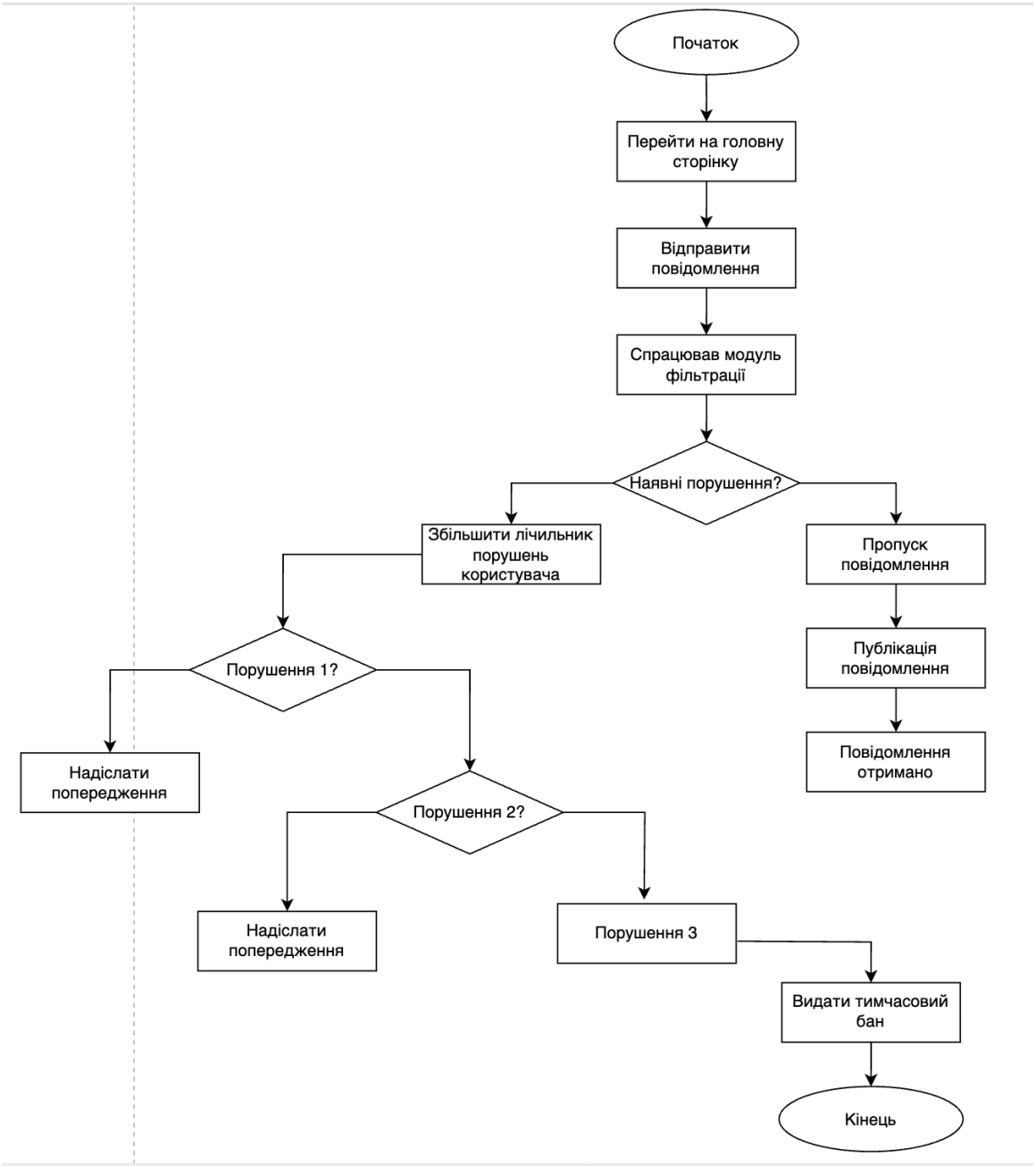


Рисунок В.1 – Схема алгоритму роботи програмного модуля фільтрування образливих повідомлень.

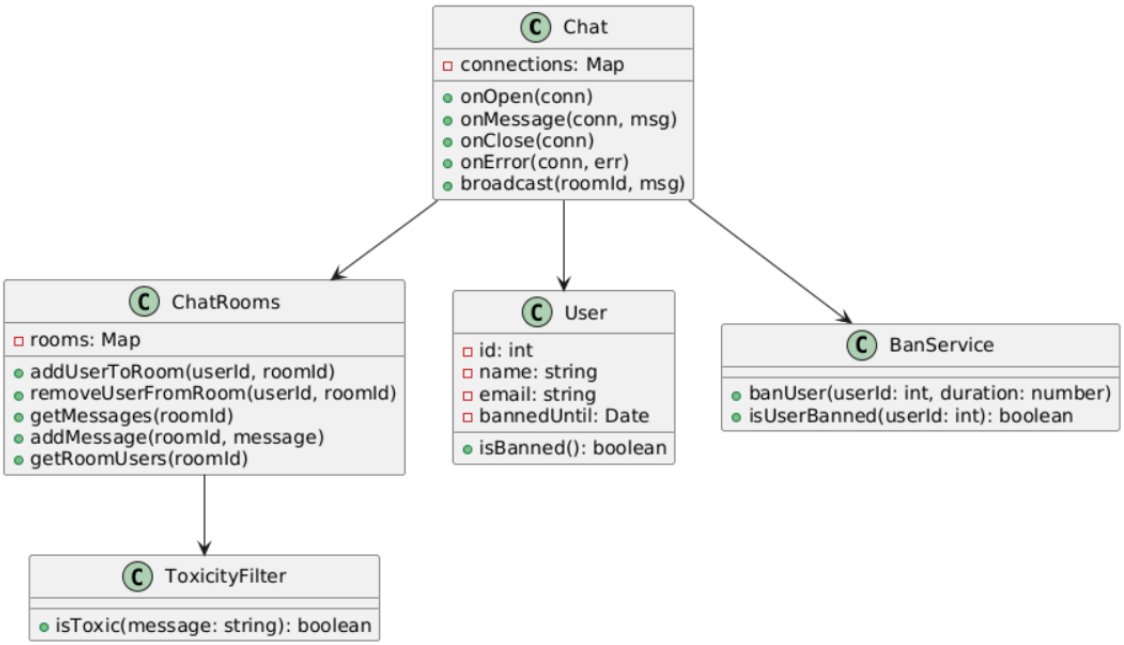


Рисунок В.2 – UML-діаграма класів серверної частини модуля фільтрування образливих повідомлень.

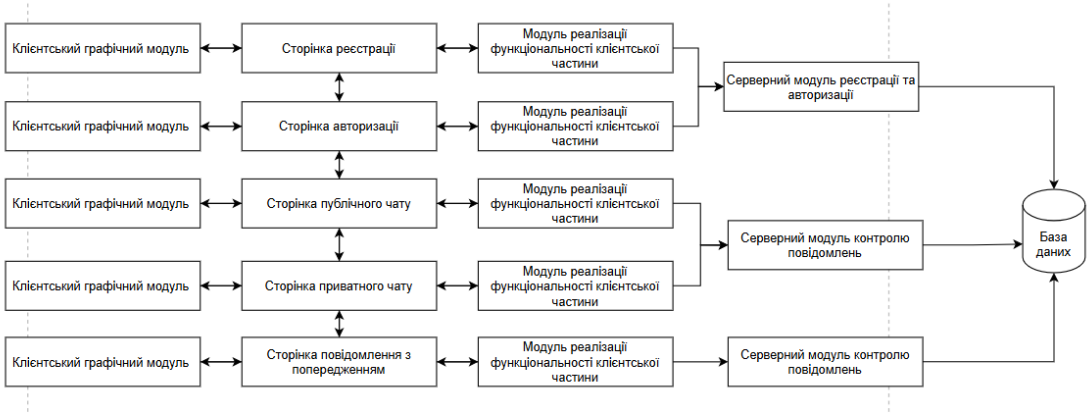


Рисунок В.3 – Загальна структурна схема функціонування програмного модуля.

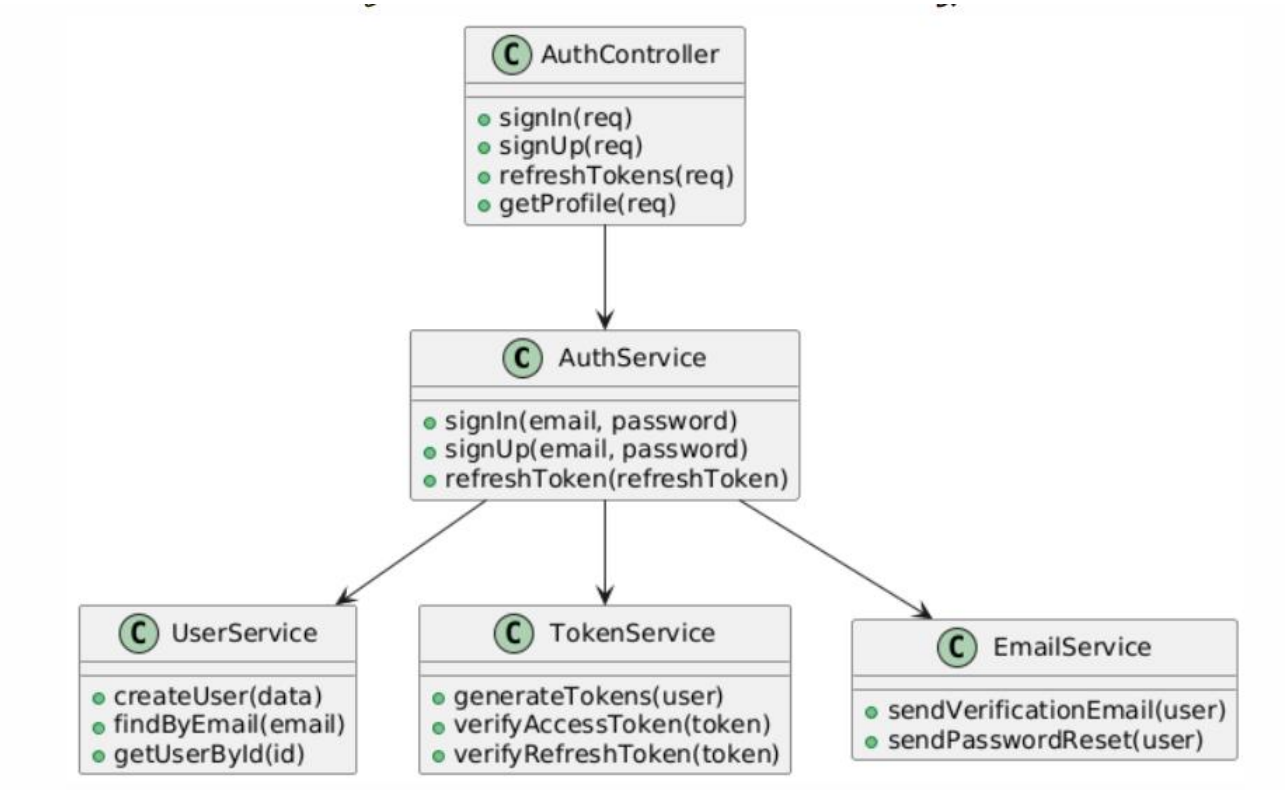


Рисунок В.4 – UML-діаграма класів серверної частини модуля аутентифікації.

Рисунок В.5 – ER-модель бази даних модуль фільтрування образливих повідомлень.

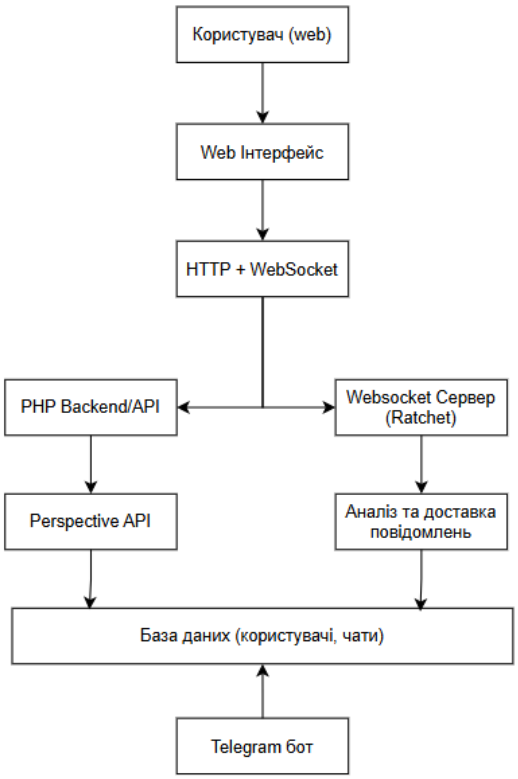


Рисунок В.6 – Загальна структурна схема компонентів програмного модуля фільтрування образливого контенту

ДОДАТОК Г ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗОВАНОГО ФІЛЬТРУВАННЯ ОБРАЗЛИВИХ ПОВІДОМЛЕНЬ

Для початку роботи необхідно зареєструватись. Щоб це зробити необхідно натиснути на кнопку «Реєстрація» в меню навігації і ввести свої дані у вікно реєстрації. Це вікно зображене на рисунку Г.1. Розпочати варто з тестування реєстрації користувача. На рисунку Г.2 зображено успішну реєстрацію користувача.

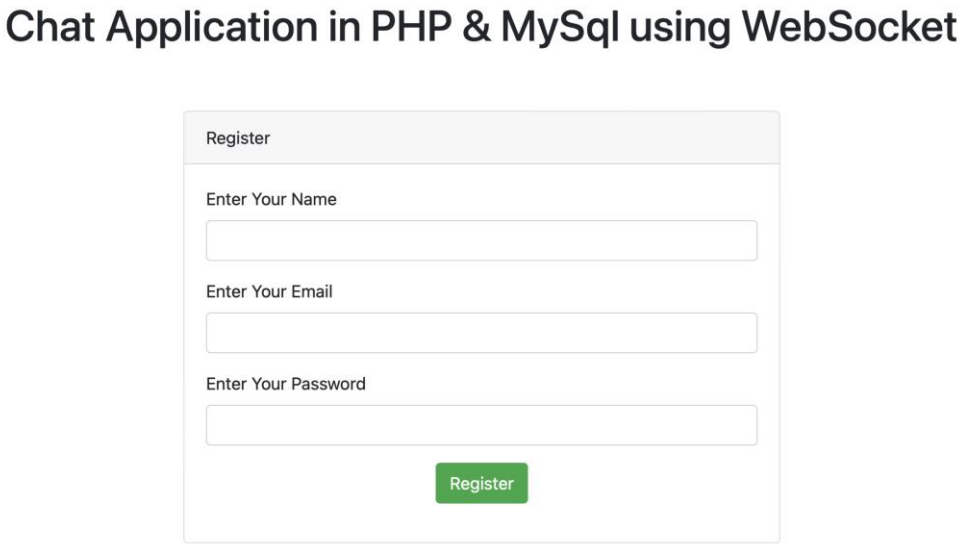


Рисунок Г.1 – Вікно реєстрації користувача.

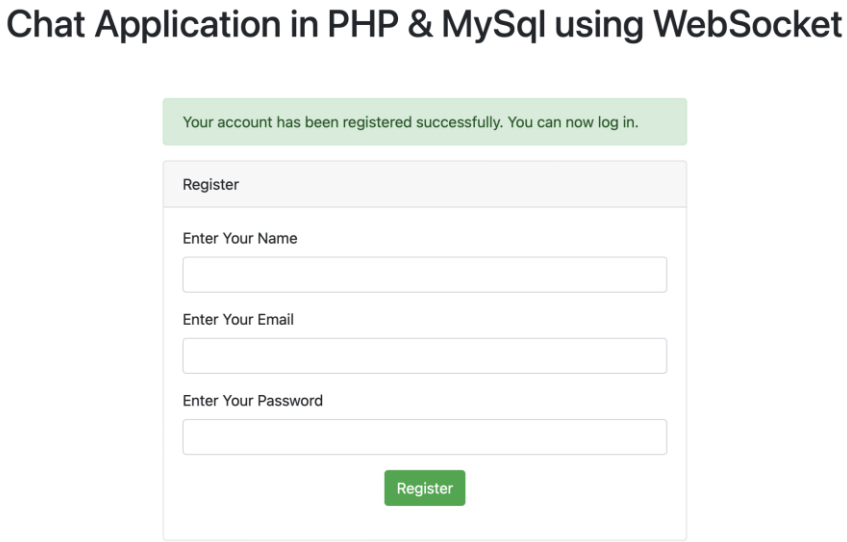


Рисунок Г.2 – Вікно успішної реєстрації.

Після цього слід відкрити меню «Увійти» та ввести свої облікові дані, це представлено на рисунку Г.3. У разі успішної авторизації відбудеться перенаправлення на головну сторінку програми, представлену на рисунку Г.4.

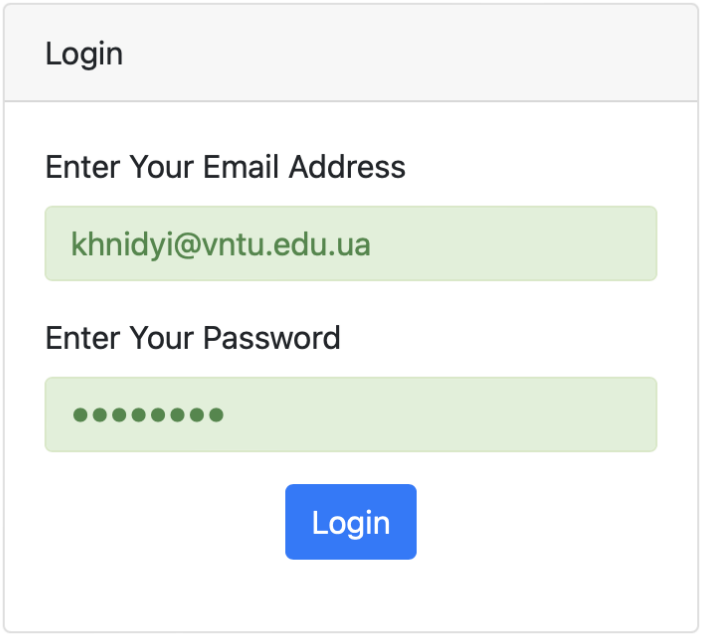


Рисунок Г.3 – Меню «Увійти».

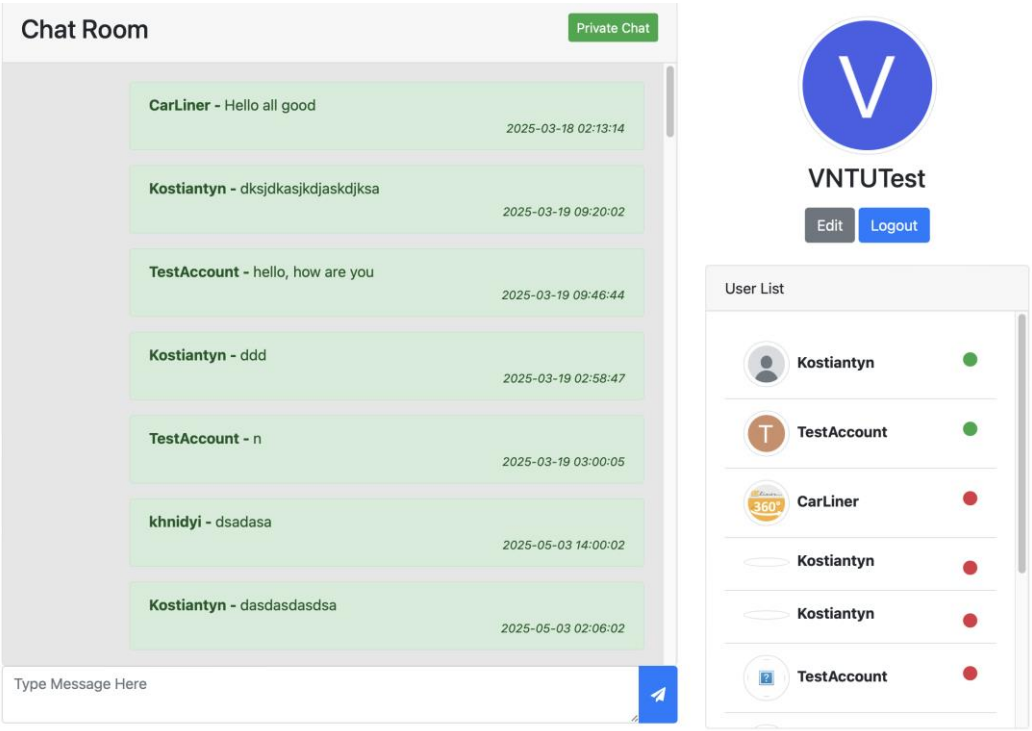


Рисунок Г.4 – Вікно успішного входу.

Після успішного входу та перенаправлення користувача на інтерфейс основного додатку можна побачити роботу WebSockets та відправити повідомлення, яке буде отримано вживу всіма користувачами. Це продемонстровано на рисунку Г.5.

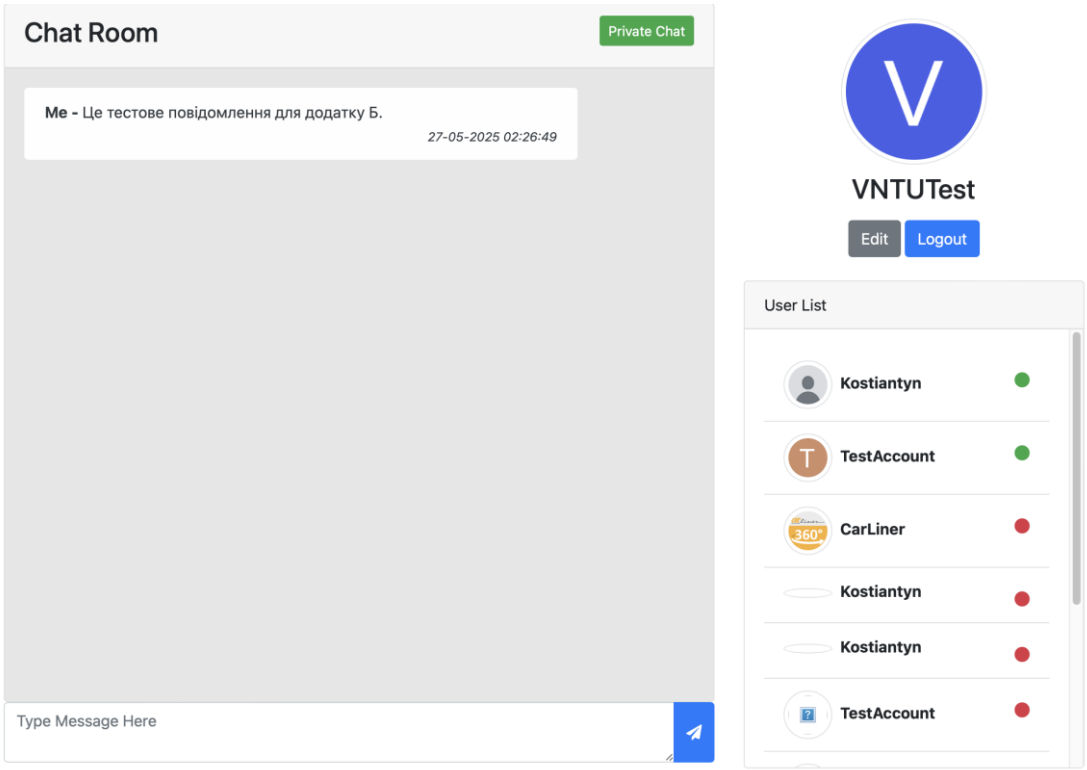


Рисунок Г.5 – Робота чату.

Також звідси крім можливості вийти з аккаунту можна перейти на приватні чати. Показано це на рисунку Г.6.

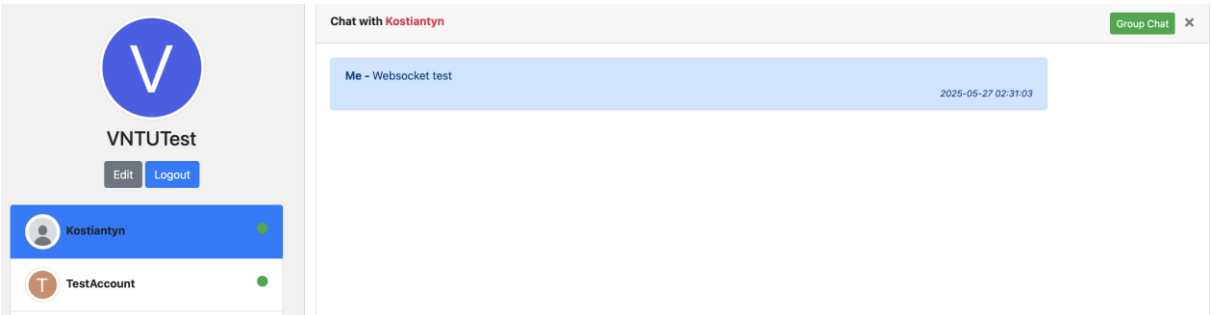


Рисунок Г.6 – Приватний чат.

Для демонстрування роботи самого модуля запустимо обидві версії та спробуємо ввести образливе повідомлення українською мовою. На рисунку Г.7 зображено роботу веб додатку, на рисунку Г.8 – роботу телеграм бота.

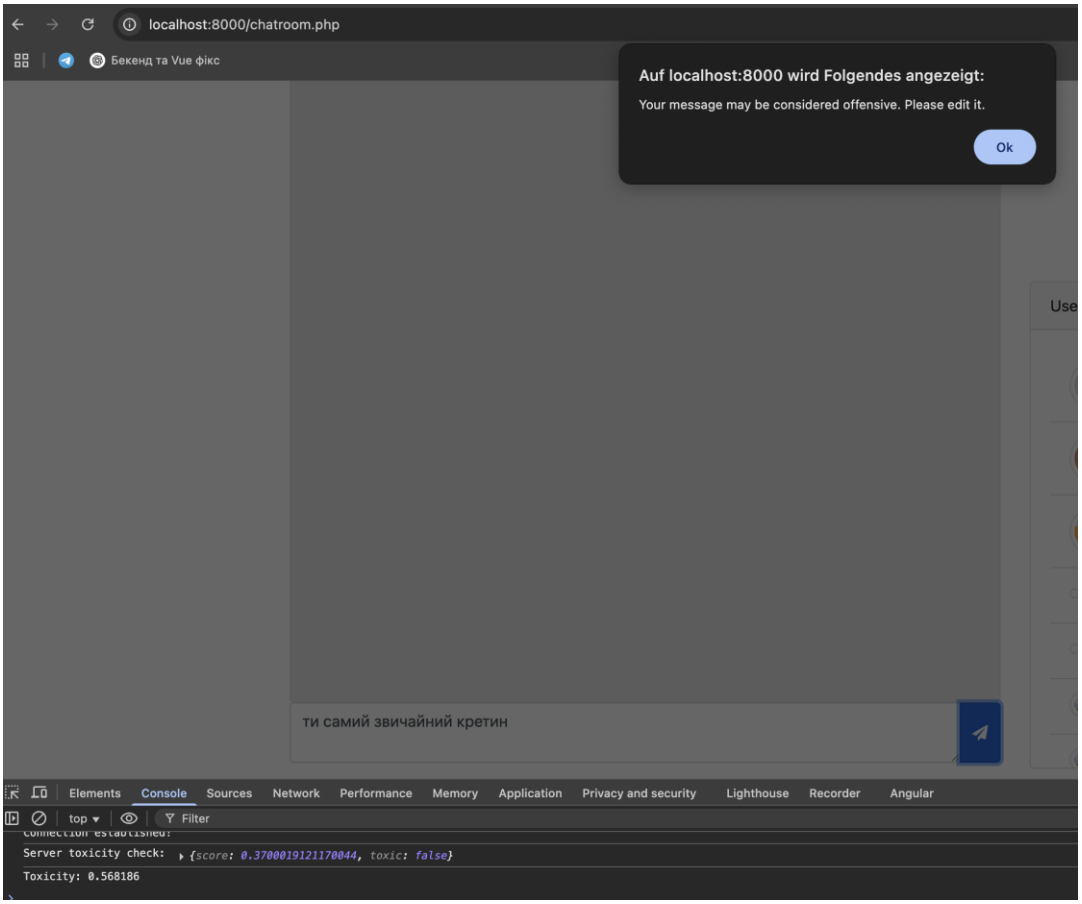


Рисунок Г.7 – Приклад роботи програмного модуля автоматизованого фільтрування образливих повідомлень у веб-додатку.

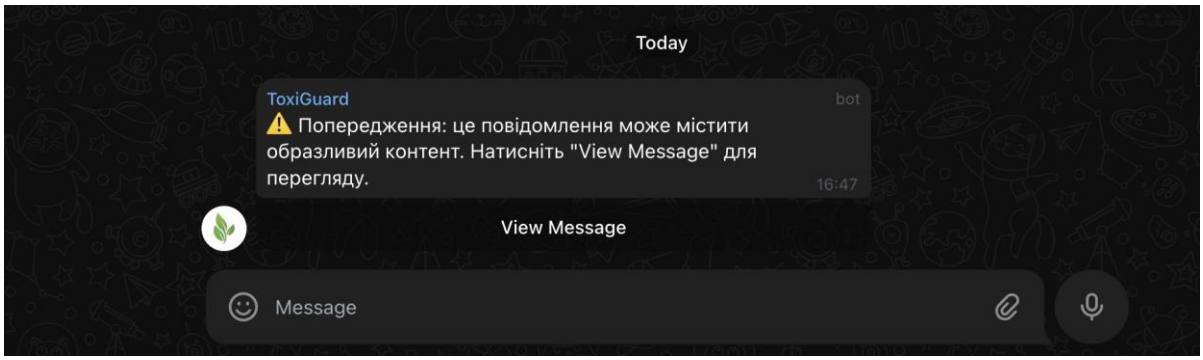


Рисунок Г.8 – Приклад роботи програмного модуля автоматизованого фільтрування образливих повідомлень як телеграм бота.