Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

за спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка програмного модуля виявлення та запобігання поширенню спаму, небезпечного та неприйнятного контенту в публічних чатах

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-20-2
_____/Д.Ю.Сидоренко/

Керівник
кваліфікаційної роботи _____ /Н.Н.Шаповалова/

Завідувач кафедри _____ /А.М.Стрюк/

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М.Стрюк

«__» _____ 20__р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-20-2 Сидоренко Денису Юрійовичу

1. Тема:. Розробка програмного модуля виявлення та запобігання поширенню спаму, небезпечного та неприйнятного контенту в публічних чатах Затверджено наказом по КНУ №__ від «_» _____ 2024р.
2. Термін подання студентом закінченої роботи : «_» _____ 2024р.
3. Вихідні дані по роботі: розроблювана система повина працювати з текстами (статтями, коментарями, відгуками) англійської мовою.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз вже існуючих систем інтелектуального аналізу тексту, розглянути методи та технології які можна використати під час розробки, вибрати найефективніші. Спроектувати систему інтелектуального аналізу тексту, програмно реалізувати розроблену систему, провести фінальне тестування для перевірки правильності роботи функціоналу системи.
5. Перелік ілюстративного матеріалу: блок-схеми розроблюваних алгоритмів системи, знімки екранних форм, графи нейронних мереж, схеми взаємодії програмних модулів між собою.

РЕФЕРАТ

Фільтрація спаму,виявлення небезпечного контенту,блокування неприйнятного вмісту, аналіз текстових повідомлень , машинне навчання для виявлення шкідливого вмісту,автоматичне видалення небажаного контенту

Пояснювальна записка: 1 с., 1 табл., 1 рис., 1 дод., 1 джерел.

Проблема спаму, небезпечного та неприйнятного контенту в публічних чатах у наші часи є дуже актуальною темою у сучасному світі. Зараз існує безліч онлайн-платформ, таких як соціальні мережі, месенджери, форуми тощо, де користувачі можуть взаємодіяти між собою та відправляти неприйнятний контент для них. Це створює ідеальне середовище для розповсюдження спаму, шкідливих посилань та непристойного вмісту між людьми. Такі матеріали можуть не лише обурювати користувачів непотрібною інформацією, а й становити загрозу їхній безпеці, включаючи можливість вірусів, фішингових атак і т. д. Тому розробка та вдосконалення програмних засобів для виявлення та запобігання цьому типу вмісту є надзвичайно важливим завданням для багатьох компаній і організацій.

Метою кваліфікаційної роботи є розробка програмного модуля для виявлення та запобігання поширенню спаму, небезпечного та неприйнятного контенту в публічних чатах. Ця робота спрямована на розробку ефективного інструменту, з боротьбою проти спаму, який допоможе забезпечити безпеку та комфорт користувачів у віртуальних спільнотах та онлайн-спілкуванні. Буде включати в себе дослідження сучасних методів аналізу текстового контенту, розробку алгоритмів фільтрації та класифікації повідомлень, а також впровадження і тестування програмного модуля з метою підвищення ефективності і точності виявлення шкідливого вмісту. У результаті роботи буде створено програму, яка може бути використана для захисту користувачів від небажаних впливів у віртуальному середовищі та підвищення загальної якості спілкуванні у мережі.

ABSTRACT

Spam filtering, detection of dangerous content, blocking of unacceptable content, analysis of text messages, machine learning to detect malicious content, automatic removal of unwanted content

Thesis in 1 p., 1 table, 1 figure, 1 appendix, 1 source.

The problem of spam, dangerous and unacceptable content in public chats is a very hot topic in today's world. Nowadays, there are many online platforms such as social networks, messengers, forums, etc., where users can interact with each other and post content that is not acceptable to them. This creates the perfect environment for spam, malicious links, and obscene content to spread between people. Such materials can not only annoy users with unnecessary information, but also pose a threat to their security, including the possibility of viruses, phishing attacks, etc. Therefore, the development and improvement of software tools to detect and prevent this type of content is an extremely important task for many companies and organizations .

The purpose of the qualification work is to develop a software module for detecting and preventing the spread of spam, dangerous and unacceptable content in public chats. This work is aimed at developing an effective anti-spam tool that will help ensure the safety and comfort of users in virtual communities and online communication. It will include the study of modern methods of textual content analysis, the development of message filtering and classification algorithms, as well as the implementation and testing of a software module in order to increase the efficiency and accuracy of detecting malicious content. As a result of the work, a program will be created that can be used to protect users from unwanted influences in the virtual environment and improve the overall quality of communication in the network.

2. РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1.Розробка та докладний опис функціональної схеми програми

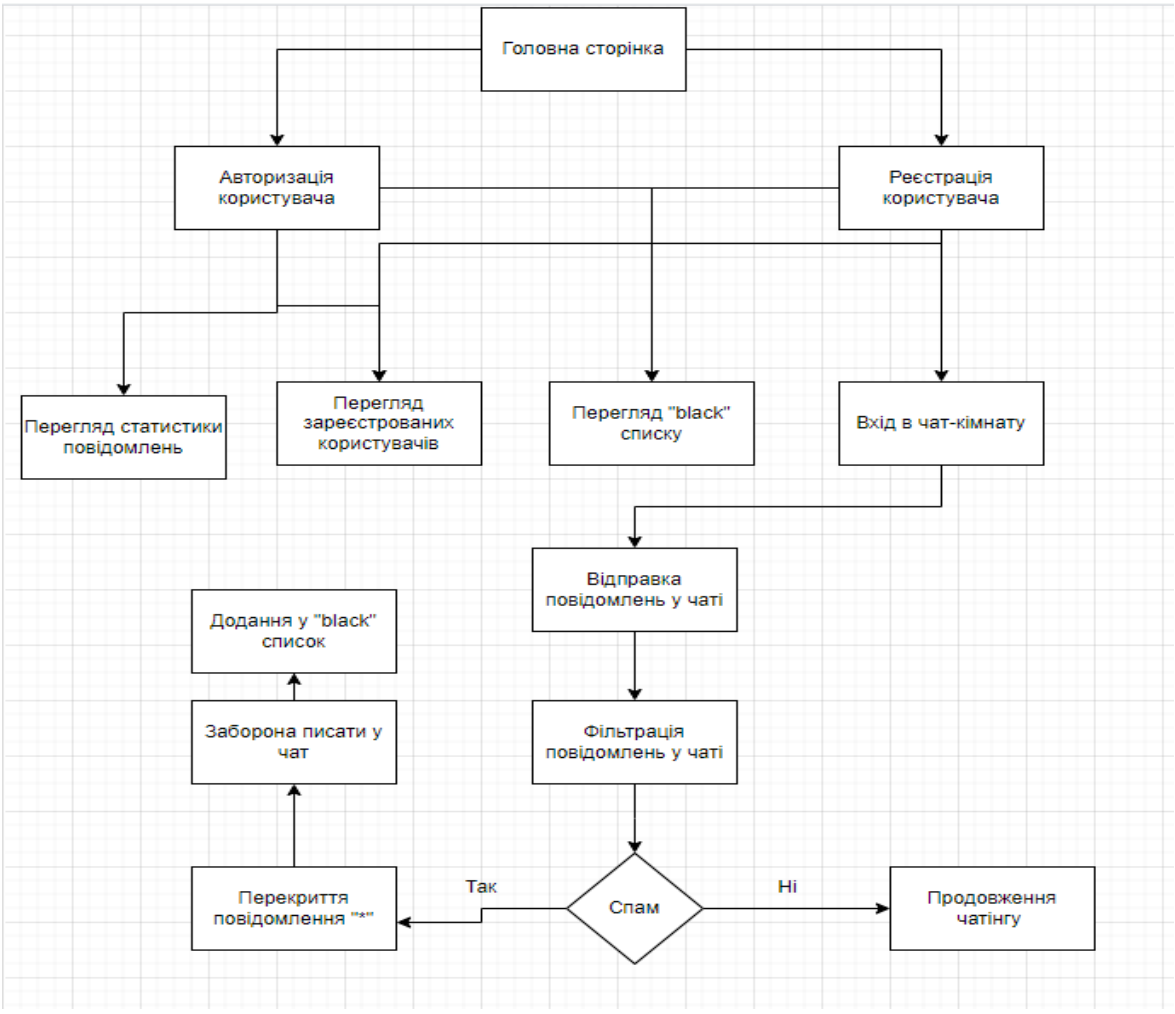
Основні компоненти програми включають:

Користувачі: вхідні дані для програми, які можуть включати інформацію про користувачів чату, їхні повідомлення та інші деталі.

Повідомлення: інформація, надіслана користувачами чату. Ці дані будуть відфільтровані для виявлення спаму.

Фільтр спаму: основна частина програми, яка використовує різні алгоритми та методи для виявлення спаму в повідомленнях.

Аналіз повідомлень: цей компонент відповідає за аналіз повідомлень на наявність спаму та виконання відповідних дій, наприклад як видалення або позначення їх як небажаного вмісту.



Малюнок 2.1 – Структурна схема програми

1. Користувачі:
    a. Ця структура представляє дані про користувачів чату, такі як ідентифікатор користувача, ім'я, електронна пошта тощо.
    b. Може містити також метадані про активність користувачів, наприклад, кількість повідомлень, час останньої активності тощо.
    c. Дані про користувачів можуть бути використані для аналізу їхньої поведінки та ідентифікації потенційних спамерів.
2. Повідомлення:
    a. Ця структура містить текстові дані повідомлень, їхні ідентифікатори, інформацію про час надходження та відправника.
    b. Може також містити метадані, такі як тип повідомлення (текст, зображення, відео), важливість тощо.
    c. Дані про повідомлення використовуються для подальшого аналізу та фільтрації спаму.
3. Фільтр спаму:
    a. Цей компонент виконує основну роботу з виявлення та фільтрації спаму.
    b. Використовує різні методи, такі як аналіз тексту, машинне навчання, регулярні вирази тощо, для визначення спамового вмісту.
    c. Може мати модульну структуру, що дозволяє легко додавати та вдосконалювати методи фільтрації.
4. Аналіз повідомлень:
    a. Цей компонент відповідає за аналіз та обробку повідомлень після фільтрації спаму.
    b. Виконує дії, такі як видалення спамових повідомлень, позначення їх як небажаний вміст, або пересилання модераторам для подальшого розгляду.
    c. Може також генерувати звіти або статистику про ефективність фільтрації спаму для подальшого аналізу та вдосконалення системи.

Контекстна діаграма - це ілюстрація, яка показує, як система (розроблюваний об'єкт) та інші сутності (зазвичай зовнішні системи або користувачі) взаємодіють у межах середовища, в якому вона функціонує. Вона допомагає зрозуміти, як система взаємодіє зі своїм оточенням.

Наприклад, контекстна діаграма для програми фільтрації спаму у чаті продемонструє, як ця програма взаємодіє з іншими елементами у середовищі чату.

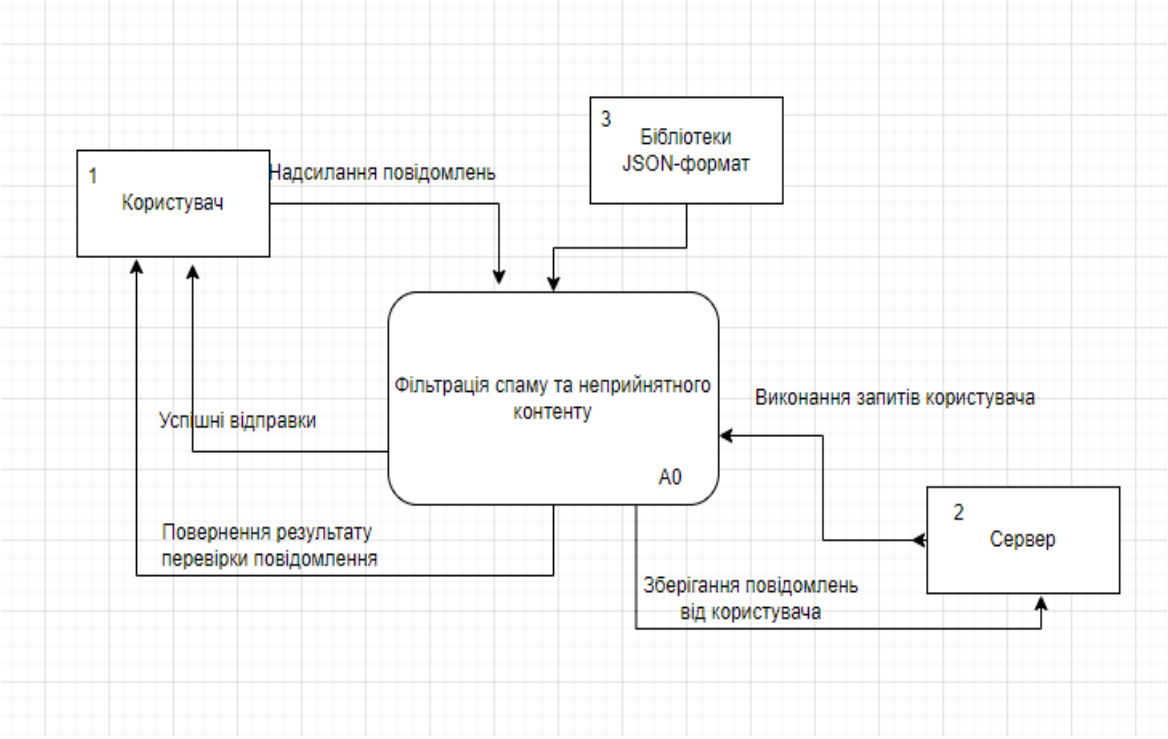
Вона може включати такі компоненти:

Користувачі чату: Люди, які використовують чат для спілкування. Вони можуть надсилати та отримувати повідомлення, а також взаємодіяти з системою фільтрації спаму.

Система фільтрації спаму:

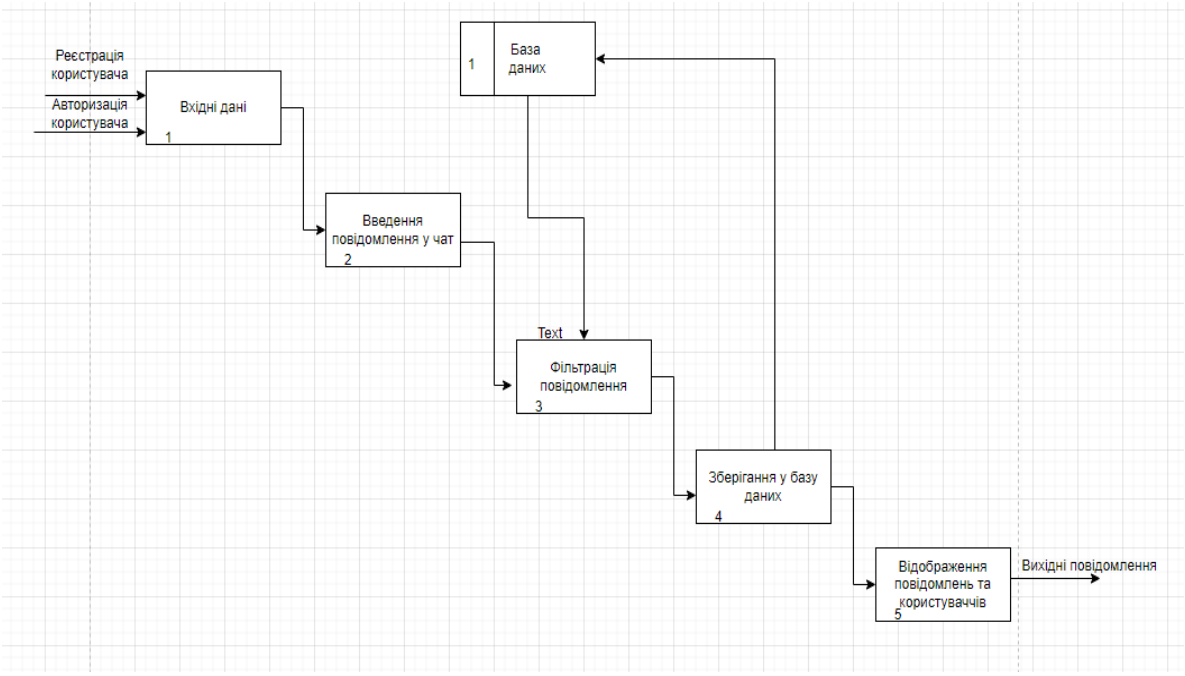
Програма, яка аналізує повідомлення в чаті та виявляє спам. Вона може блокувати або видаляти спамові повідомлення, а також повідомляти користувачів про виявлені дії.Інші системи чату:

Будь-які інші компоненти чатової системи, які можуть взаємодіяти з системою фільтрації спаму, наприклад, сервер чату.



Малюнок 2.2 – Контекстна діаграма

Загальна діаграма потоків даних (DFD) - це графічне зображення потоків даних та їхньої обробки в системі. Діаграма потоків даних допомагає узагальнити та візуалізувати потоки даних в системі, відображаючи, як вони обробляються та взаємодіють між собою. Нижче наведено детальную діаграму потоків даних



Малюнок 2.3 – Загальна діаграма потоків даних

Деталізована блок-схема обробки даних покаже кожен етап обробки даних в системі фільтрації спаму у чаті, включаючи всі кроки та умови, які виконуються на кожному етапі.

1. Вхідні повідомлення:

   Це вихідні дані, які надходять в систему для обробки.

2. Перевірка спаму:

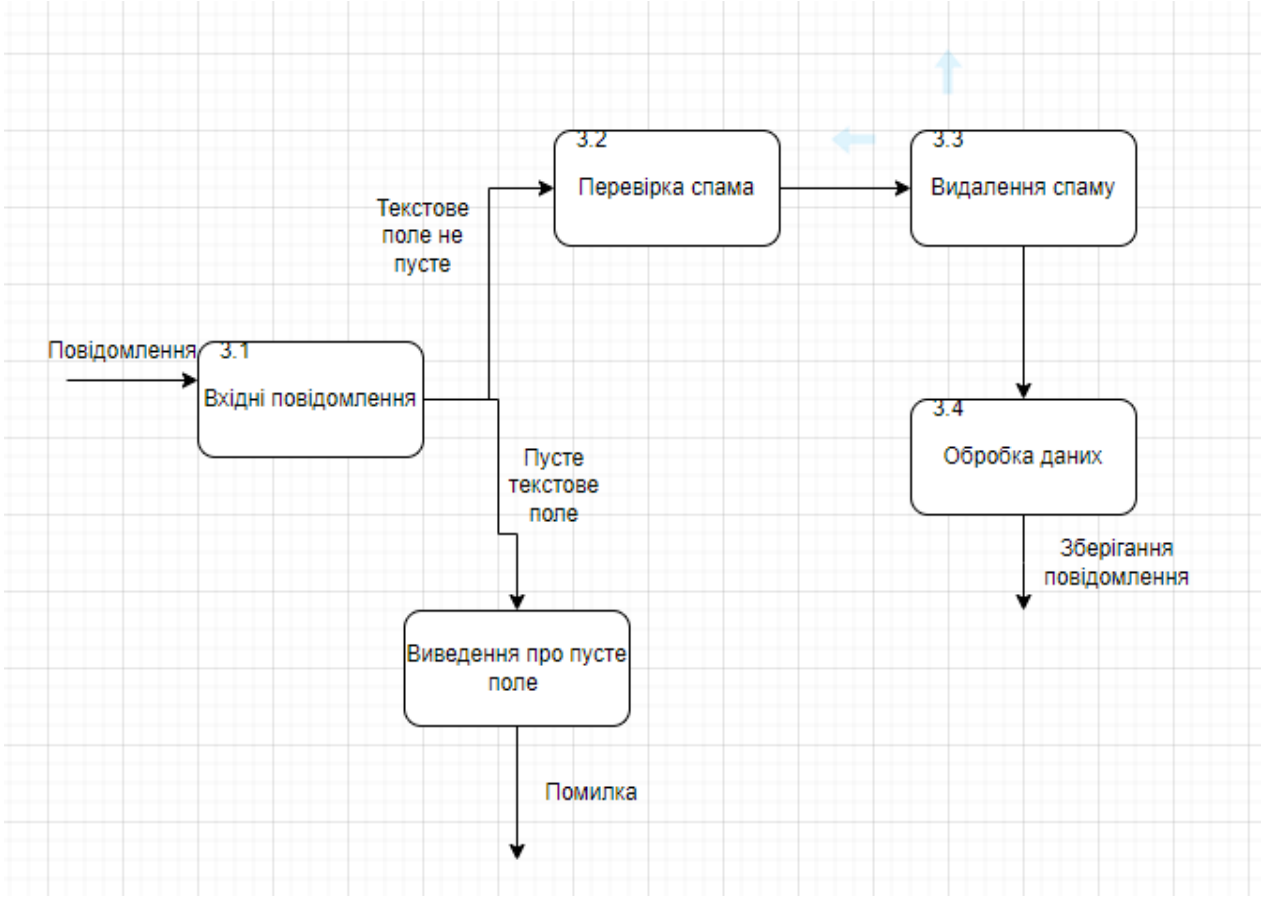
   На цьому етапі вхідні повідомлення перевіряються на наявність спаму. Можна використовувати різні методи, такі як аналіз тексту, перевірка ключових слів тощо.

Чи знайдено спам?: це рішення, засноване на результатах перевірки, чи виявлено спам у повідомленні.

3. Видалення спаму: якщо спам знайдено, цей блок виконує дії для видалення спаму.
4. Вихідні результати:

Остання стадія обробки, на яку доставляються результати фільтрації, які можуть містити як спам, так і звичайні повідомлення.

і



Малюнок 2.4 – Деталізована діаграма потоків даних

Перейдемо до функціональної схеми ПЗ.

Кожен елемент схеми має своє призначення:

1. Вхідні повідомлення:

Це дані, які надходять від користувачів у чат.

2. Фільтрація та аналіз спаму:

На цьому етапі перевіряється кожне повідомлення на наявність спаму.

Використовуються різні методи, такі як аналіз тексту, машинне навчання та правила фільтрації.

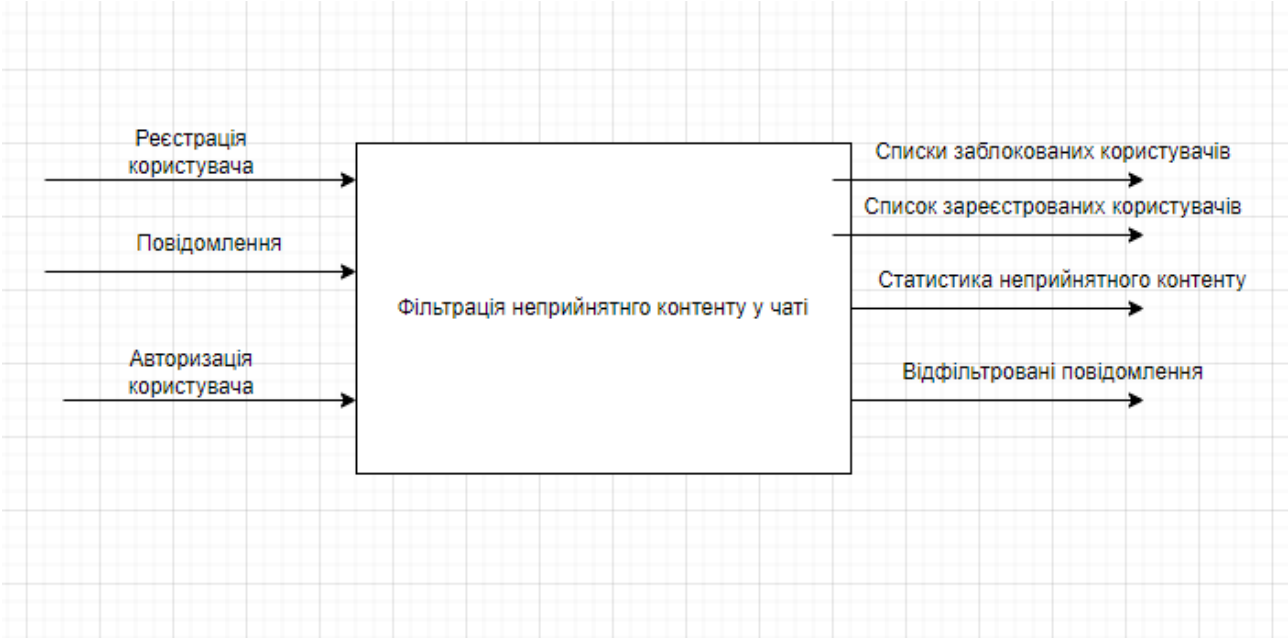
3. Видалення спаму з чату:

Якщо повідомлення виявляється спамом, його видаляють з чату.

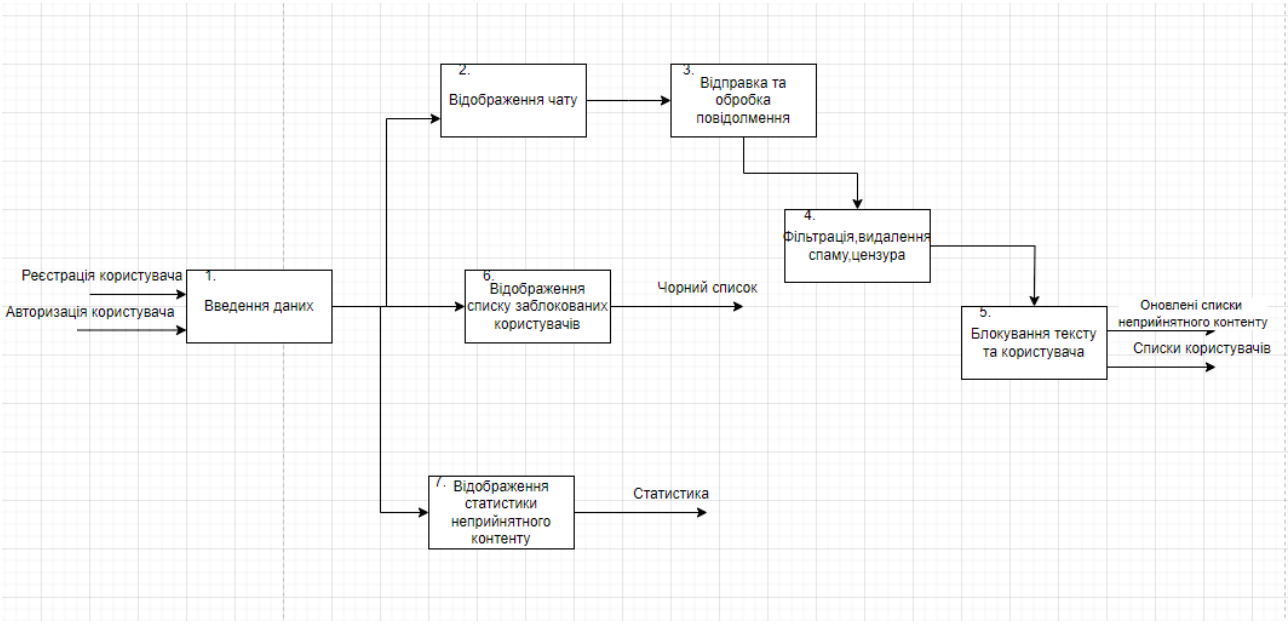
4. Вихідні повідомлення (очищені):

Це оброблені повідомлення, які пройшли процес фільтрації та не містять спаму.

Такі повідомлення виводяться на екран для відображення в чаті.



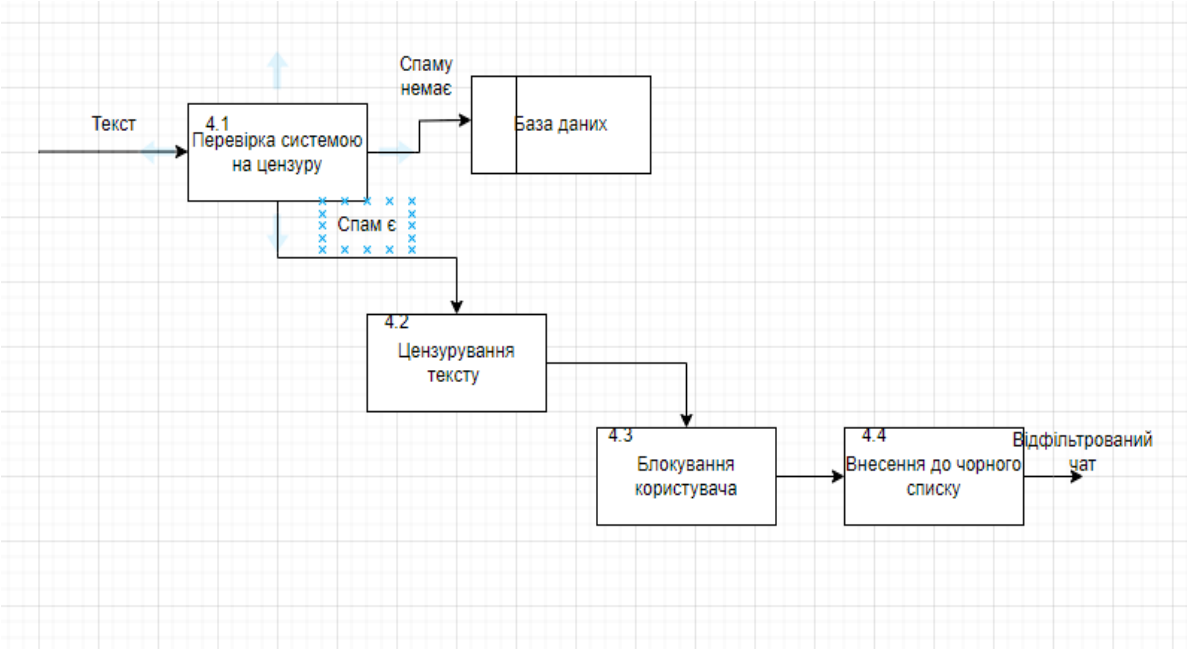
Малюнок 2.5 - Загальна функціональна схема ПЗ



Малюнок 2.6 - Фунціональна схема ПЗ

Програма отримує повідомлення від користувачів і відображає їх у чаті. Отримавши кожне повідомлення, програма аналізує його, щоб визначити, чи містить воно спам або інший небажаний вміст.

Можуть використовуватися різні методи фільтрації, перевірка ключових слів або машинне навчання. Якщо повідомлення визначено як спам або містить небажаний вміст, воно видаляється з чату. В іншому випадку програма може обробити повідомлення, наприклад, попередивши користувачів про потенційно небажаний вміст або переславши його модераторам для подальшого аналізу.



Малюнок 2.7 – Деталізація блоку «Фільтрація»

Приймання повідомлень: Система отримує повідомлення, надіслані користувачами чату.

1. Перевірка повідомлень:

   Текст повідомлень перевіряється різними способами:

2. Пошук ключових слів:

Порівняння тексту з переліком слів або фраз, які можуть вказувати на спам чи небажаний вміст.

3. Застосування правил:

   Використання правил фільтрації, що визначають типові ознаки спаму або небажаного вмісту.

4. Машинне навчання:

   Застосування моделей машинного навчання для автоматичного виявлення спаму на основі попереднього досвіду.

5. Визначення спаму:

   Якщо перевірка виявляє, що повідомлення містить спам чи небажаний вміст, воно позначається як спам.

6. Результат фільтрації:

Залежно від результатів перевірки:

Якщо повідомлення визначено як спам, воно видаляється з чату.

Якщо спам не виявлено, повідомлення передається для відображення в чаті.

Повідомлення результатів:

Результати фільтрації можуть бути показані користувачам або використані у подальшій роботі системи.

2.2. Розробка та докладний опис алгоритму роботи програми

2.2.1. Опис функціональних та нефункціональних вимог до систем

Функціональні вимоги — це специфікації функцій і операцій, які має виконувати система або програмне забезпечення. Вони визначають, як система має реагувати на певні вхідні дані або події та які вихідні результати мають бути згенеровані. Функціональні вимоги зосереджені на функціональності системи та визначають, що система повинна робити. Ключові характеристики функціональних вимог:

1. Опис операцій і функцій:

   Функціональні вимоги описують конкретні операції та функції, які повинна виконувати система.

2. Вхідні дані:

   вони визначають вхідні дані або події які активують певну функцію чи операцію.

3. Вихідні результати:

   функціональні вимоги вказують очікувані вихідні результати, які система має генерувати після виконання операцій.

4. Умови виконання:

   вони описують умови, за яких функція чи операція можуть бути виконані.

Таб. 1 – Функціональні вимоги

№	**Вимога**
1	Програма повинна мати можливість отримувати повідомлення від користувачів чату.
2	Він повинен аналізувати кожне повідомлення, щоб виявити спам і небажаний вміст.
3	Якщо повідомлення визначено як спам, його слід видалити з чату, перш ніж відобразити.
4	Користувачі мають отримати сповіщення, якщо їхнє повідомлення буде видалено через спам, разом із поясненням.
5	Модератори чату повинні мати можливість переглядати та аналізувати видалені повідомлення для подальших дій.
6	Цензурування небажанаого на несприйнятного контенту у чаті
7	

Нефункціональні вимоги - це характеристики системи або програмного забезпечення, які не стосуються безпосередньо функціональності, але визначають важливі якісні аспекти. Вони описують різні аспекти програми, які впливають на її ефективність, надійність, безпеку та інші важливі характеристики.

Основні особливості нефункціональних вимог

1. Якісні аспекти:

   Вони описують важливі якості програми чи системи, але не визначають її конкретні функції.

2. Продуктивність:

   Включають швидкодію, потужність та масштабованість системи.

3. Безпека:

   Визначають вимоги до захисту даних та конфіденційності.

4. Надійність та стабільність:

Описують стійкість до збоїв та вимоги до резервного копіювання.-

5. Зручність використання:

Встановлюють вимоги до інтерфейсу та легкості використання.-

6. Управління та підтримка:

Визначають вимоги до підтримки, управління та адміністрування системи.

7. Інтеграція та сумісність:

Описують вимоги до інтеграції з іншими системами.

Нефункціональні вимоги :

Таб. 2 – Нефункціональні вимоги

№	**Вимога**
1	Система повинна мати високу точність виявлення спаму, щоб уникнути видалення корисного вмісту.
2	Він має ефективно та швидко фільтрувати та видаляти спам-повідомлення.
3	Система повинна мати можливість обробляти велику кількість користувачів і величезний обсяг повідомлень без шкоди для продуктивності.
4	Дані користувача повинні бути захищені від несанкціонованого доступу та надійно зберігатися.
5	Дані користувача повинні бути захищені від несанкціонованого доступу та надійно зберігатися.
6	Масштабованість: програма повинна бути здатною працювати з великою кількістю користувачів та обсягом даних без втрати продуктивності.
7	Інтерфейс: програма повинна мати зручний та інтуїтивно зрозумілий інтерфейс користувача.

2.2.2. Технології та інструменти фільтрування тексту у чаті

Технології та інструменти для фільтрації тексту в чаті:

"bad-words" - це бібліотека, яка допомагає виявляти та видаляти нецензурну лексику та образливий вміст у тексті.

"bcrypt" - використовується для захищеного зберігання та перевірки паролів користувачів. Це може допомогти ідентифікувати та заблокувати користувачів, які поширюють спам або небажаний вміст.

"expre-ss" та "body-parser" - ці інструменти допомагають обробляти та аналізувати вхідні повідомлення від користувачів чату

."socket.io" - це бібліотека для реалізації двостороннього зв'язку в реальному часі між користувачами чату. Це може полегшити фільтрацію та обробку повідомлень.

"jsonwebtoke-n" - використовується для створення та перевірки токенів автентифікації, що важливо для забезпечення безпеки та ідентифікації користувачів чату."mongodb" та "mongoose" - ці інструменти дозволяють зберігати дані про користувачів, повідомлення та інші деталі чату в базі даних.

Технологія	**Опис**	**Особливості**
bad-words	Бібліотека для фільтрації нецензурних слів у тексті.	Надає простий інтерфейс для виявлення та фільтрації нецензурного вмісту.
bcrypt	Хешування та перевірка паролів користувачів.	Використовується для зберігання паролів у захешованому вигляді, що підвищує безпеку даних.
body-parser	Middleware для обробки вхідних даних у форматі JSON.	Дозволяє отримувати та обробляти дані в форматі JSON з вхідних запитів.

cors	Middleware для управління політикою CORS.	Дозволяє обмежувати доступ до ресурсів на сервері з інших доменів для безпеки та конфіденційності.
dotenv	Завантаження змінних середовища з файлу **.env**.	Забезпечує зручний спосіб керування конфіденційними даними та параметрами конфігурації.
express	Фреймворк для створення веб-серверів на Node.js.	Надає потужні інструменти для розробки веб-додатків та API з використанням JavaScript.
jsonwebtoken	Створення та перевірка токенів автентифікації.	Використовується для створення та перевірки токенів автентифікації користувачів.
mongodb	База даних NoSQL, яка зберігає дані у документах.	Дозволяє зберігати структуровані дані та виконувати потужні операції з базою даних.
mongoose	Об'єктно-орієнтований драйвер для MongoDB.	Надає зручний інтерфейс для взаємодії з базою даних MongoDB з використанням Node.js.

2.2.3. Алгоритми фільтрування тексту

Тут кілька методів, які можна використовувати для фільтрації небажаного контенту в чаті:

Фільтрація за ключовими словами:

Цей метод використовує заздалегідь визначені списки ключових слів, які вказують на спам або небажаний вміст. Коли надходить повідомлення, воно перевіряється на наявність цих ключових слів, і потім повідомлення фільтрується або відхиляється.

Машинне навчання:

Методи машинного навчання, такі як класифікація тексту або нейронні мережі, можуть автоматично виявляти спам та небажаний вміст у текстових повідомленнях. Вони аналізують зміст повідомлень та визначають, наскільки ймовірно, що вони є спамом.

Правила на основі випадкових лісів:

Цей метод використовує алгоритм машинного навчання під назвою "випадковий ліс" для створення правил фільтрації тексту.

Він аналізує набір даних, щоб визначити, які ознаки (слова, фрази тощо) вказують на спам, а потім застосовує ці правила до нових повідомлень для фільтрації.

Таблиця – алгоритми фільтрування тексту

Алгоритм	**Опис**
Відфільтрування за допомогою ключових слів	Цей алгоритм використовує попередньо визначені списки ключових слів, що характеризують спам або небажаний вміст, для фільтрації тексту.
Машинне навчання	Використання методів машинного навчання, таких як класифікація тексту або нейронні мережі, для

	автоматичного виявлення спаму та небажаного вмісту.
Правила на основі випадкових лісів (Random Forest Rules)	Використання алгоритму машинного навчання Random Forest для створення правил для фільтрації тексту, що базуються на аналізі набору даних.
Аналіз тональності	Аналіз тону та емоційного забарвлення тексту для визначення його спамовості або небажаного характеру.
Стеммінг та лемматизація	Методи морфологічного аналізу, які допомагають зменшити слова до їх базової форми для подальшої фільтрації та аналізу.

Ліміт повідомлень від спаму.

Ведеться мова про повідомлення в чаті для користувача, за певний період часу — перевірка ліміту повідомлень. А нижче — опис, як працює ця функція:

Перевірка кількості повідомлень: Тут функція проводить перевірку, чи користувач вже відправив 3 чи більше повідомлень. Це контролюється за допомогою парамета messageCount.

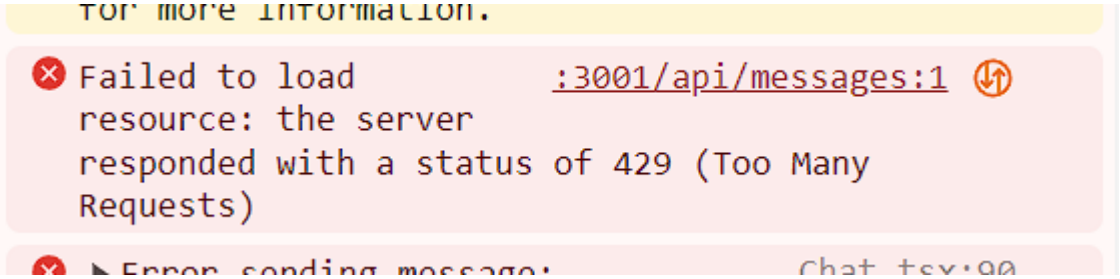
Перевірка часу останнього повідомлення: В разі того, що користувач здесь вже відправив достатню кількість повідомлень, функція перевіряє час, який пройшов з моменту надходження останнього повідомлення. Це робиться за допомогою розрахунку різниці між поточним часом і часом останнього повідомлення.

Повернення відповіді при перевищенні ліміту: В разі такої ситуації, коли користувач здесь вже відправив достатню кількість повідомлень і час між останнім і поточним повідомленнями менше 10 секунд, функція повертає

відповідь з HTTP-кодом 429 (Too Many Requests) та повідомленням, що користувач повинен зачекати певний час перед відправкою наступного повідомлення.

```
const checkMessageLimit = (username, messageCount, lastMessageTime, currentTime, res) => {
    if (messageCount && messageCount >= 3) {
        if (lastMessageTime) {
            const timeDifference = currentTime - lastMessageTime;
            if (timeDifference < 10000) {
                const timeLeft = Math.ceil( x: (10000 - timeDifference) / 1000);
                res.status(429).json({ message: `Please wait ${timeLeft} seconds before sending another message` });
                return true;
            }
        } else {
            lastMessageTime[username] = currentTime;
            messageCount++;
        }
    }
    return false;
};
```

Малюнок 2.8 – Функція затримки відправки повідомлень



Малюнок 2.9 – Приклад використання функції

checkUserProfanity — перевіряє, чи використовував користувач нецензурну лексику в чаті. Якщо користувач зловживає нецензурною лексикою більше певного часу, його додають до чорного списку. Як працює функція:

Перевірка, скільки нецензурних слів використав користувач:

функція з’ясовує, чи вживав користувач нецензурні слова в чаті. Параметр userBadWordCount містить підрахунок того, скільки разів користувач використав вульгарне слово.

Додавання користувача в чорний список:

Додаток перевіряє кількість непідтверджених слів, використаних користувачем. Якщо кількість спроб введення слова від користувача більше 2,

функція addToBlackList викликає метод, у якому створюється запис у чорному списку користувача для блокування – «нецензурна лексика».

Повернення відповіді на блокування користувача: якщо користувача заблоковано, функція повертає відповідь із HTTP-кодом 403 Forbidden із повідомленням про блокування через заборонену поведінку.

Обробка помилок: після успішного додавання користувача до чорного списку функція повертає код статусу HTTP 200, інакше повертає код статусу HTTP 500 – Внутрішня помилка сервера під час оновлення повідомлення про помилку з описом помилки.

Ця функція регулює неприйнятні слова в чаті та забороняє користувачам її використовувати.

```
export const checkUserProfanity = async (username, userBadWordCount, addToBlackList, res) => {
    if (userBadWordCount[username] && userBadWordCount[username] >= 2) {
        try {
            await addToBlackList(username, 'inappropriate language');
            return res.status(403).json({ message: 'You are blocked for inappropriate behavior' });
        } catch (error) {
            console.error('Error adding user to blacklist:', error);
            return res.status(500).json({ message: 'Error adding user to blacklist' });
        }
    }
};
```

Малюнок 2.10 – Функція перевірки на неприйнятний контент

```
est, …}

User op added to blacklist for reason: inappropriate words      Chat.tsx:61
>
```


Малюнок 2.11 – Приклад використання фільтрації та блокування користувачів

2.3.Опис та порівняння алгоритмів фільтрування тексту

2.3.1. Алгоритм пошуку за ключовими словами:

Опис: цей алгоритм шукає певні ключові слова або фрази в тексті, які часто асоціюються зі спамом або небажаним вмістом. Якщо текст містить одне з таких ключових слів, він позначається як потенційно небажаний.

Переваги:

Легко реалізувати.

Швидка обробка тексту.

Недоліки:

Може мати багато помилкових спрацьовувань.

Неефективний при спробі виявити нові форми спаму.

2.3.2. 2. Тональний аналіз тексту

Опис: цей алгоритм аналізує емоційний тон тексту, щоб визначити, чи є в тексті негативний або агресивний тон. Він використовується для виявлення тексту з образливим змістом.

Переваги:

Це дозволяє виявити образливий вміст.

Ефективний при моніторингу спілкування в соціальних мережах.

Недоліки:

Він може пропустити деякі вирази образливого змісту.

Для вивчення точних моделей потрібна велика кількість даних.

2.3.3. 3. Моделі машинного навчання:

Ці алгоритми використовуються для класифікації тексту на основі великої кількості ознак і шаблонів, які були вивчені на історичних даних. Вони можуть використовувати різні алгоритми, такі як Naive Bayes, SVM, нейронні мережі тощо.

Переваги:

Висока точність при правильному навчанні.

Можливість пошуку нових форм спаму.

Недоліки:

Для навчання потрібна величезна кількість даних.

Ускладнення при реалізації та обслуговуванні моделей.

Порівняння:

Простота реалізації: Алгоритм на основі ключового слова є найпростішим з точки зору реалізації, тоді як моделі машинного навчання потребують складного процесу навчання та реалізації.

Ефективність: моделі машинного навчання зазвичай є найефективнішими за умови належного навчання, оскільки вони мають можливість знаходити нові форми спаму. Тим не менш, алгоритми, засновані на ключових словах, можуть бути швидшими в плані обчислень.

Точність: моделі машинного навчання зазвичай мають найвищу точність за умови належного навчання. Тоді як алгоритми, засновані на ключових словах, можуть мати величезну кількість поганих обчислень.

Складність: моделі машинного навчання мають більш складний процес навчання та реалізації.

3. РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Аналіз обраної середи програмування

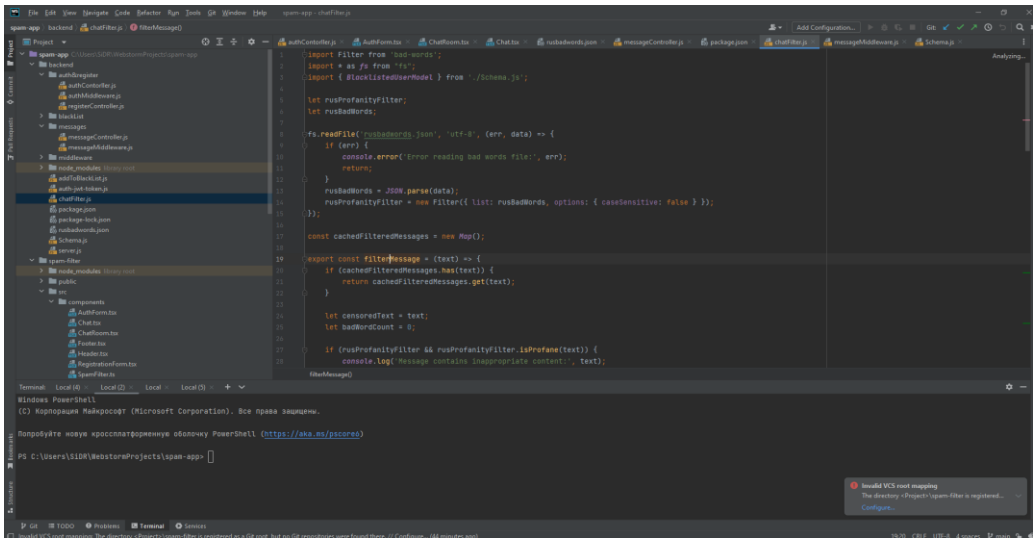
Почнемо розбирати, які ж найголовніші критерії та підходи у середі програмування найважливіші.

По-перше,підтримка операцій із рядками та регулярними виразами: оскільки фільтрація тексту є важливим будівельним блоком будь-якої мови програмування, повинні бути потужні бібліотеки для роботи з рядками та регулярними виразами.

По-друге, масштабованість і продуктивність: якщо він повинен справлятися з навантаженням великої кількості тексту або працювати в режимі реального часу, то вам слід вибрати платформу з хорошою масштабованістю та продуктивністю.

По-третє,підтримка асинхронного програмування: якщо ви розробляєте веб-систему або систему, яка має багато паралельних запитів, тоді мова, з якою ви працюєте, повинна підтримувати асинхронне програмування, щоб обробляти кілька операцій найбільш ефективним способом.

Розробка на базі WebStorm, це чудовий інструмент для розробки веб-додатків за допомогою React і Node.js. WebStorm — це інтегроване середовище розробки з набором функцій та інструментів для полегшення роботи з кодування. Він забезпечує легке кодування для розробника, оскільки підтримує як React, так і Node.js. Його інтерфейс інтуїтивно зрозумілий, що підвищує швидкість навчання та продуктивну роботу.



Малюнок 3.1 – Інтерфейс середовища WebStorm

3.2. Розробка бази даних

Наведений нижче код визначає різні типи даних у MongoDB за допомогою Mongoose у середовищі Node.js.

Наведений нижче код визначає структуру даних для користувачів, які були додані до чорного списку. Ця схема містить поля з ім'ям користувача для імені користувача та причиною причини додавання до чорного списку.

blacklistedReceiverSchema: визначає структуру даних для пари користувачів, у якій один із користувачів блокує повідомлення від іншого. Дані такої пари містять два поля: senderId, що ідентифікує відправника, і receiverId, що ідентифікує одержувача.

userSchema: визначає модель бази даних для користувачів, зареєстрованих у системі. Схема користувача містить такі поля, як ім'я користувача для імені користувача, адреса електронної пошти для електронної пошти та пароль для пароля.

messageSchema: визначає модель даних для повідомлень у чаті. Він має такі властивості, як id для ідентифікатора повідомлення, text для фактичного тексту повідомлення, ім'я користувача відправника та userId відправника.

Схема покриття дозволяє визначити структуру даних і зв'язки між цими даними. Після визначення схеми вони експортуються як моделі в mongoose.model, що дозволяє взаємодіяти з цими даними у вашій програмі.

```
import mongoose from 'mongoose';

const blacklistedUserSchema = new mongoose.Schema({
    username: String,
    reason: String,
});

const blacklistedReceiverSchema = new mongoose.Schema({
    senderId: String,
    receiverId: String,
});

const userSchema = new mongoose.Schema({
    username: String,
    email: String,
    password: String,
});

const messageSchema = new mongoose.Schema({
```

```
    id: String,
    text: String,
    username: String,
    userId: String,
});

export const BlacklistedUserModel = mongoose.model('BlacklistedUser',
blacklistedUserSchema);
export const BlacklistedReceiverModel = mongoose.model('BlacklistedReceiver',
blacklistedReceiverSchema);
export const UserModel = mongoose.model('User', userSchema);
export const MessageModel = mongoose.model('Message', messageSchema);
```

З Mongoose ми можемо:

• Визначати структуру даних, які ми використовуємо в JavaScript, щоб зробити їх більш читабельними та редагованими

• Застосувати перевірку даних, щоб переконатися, що введені дані дійсні перед їх збереженням

• Моделювати просту взаємодію з основним сховищем даних, що полегшує створення, читання, оновлення та видалення записів.

• Легко розширювати функціональність за допомогою плагінів

• Працювати з MongoDB у середовищі Node.js, використовуючи знайомий синтаксис JavaScript.

Малюнок 3.2 – Логотип бази даних mongoDb

3.3. Програмна реалізація основних функцій

Як тільки повідомлення отримано від користувача:

наш сервер отримує повідомлення від клієнта, яке слід відфільтрувати або вважати спамом або небажаним.

Просіювання новин:

за допомогою аналізу тексту, пошуку за ключовими словами або алгоритмів машинного навчання наша програма може фільтрувати текст новин, щоб визначити, чи є він спамом чи ні.

Обробка результату фільтрації:

у такій ситуації його буде видалено з чату або позначено як небажане. В інших випадках він додається до чату з метою відображення іншим користувачам.

Звіт про спам або небажаний вміст, створений користувачами. Якщо отриманий вміст був чимось на кшталт спаму або небажаного вмісту, користувачі можуть повідомити про це через центр попереджень або сповіщення від модератора для подальшого аналізу.

Мета полягає в тому, щоб зберігати дані, щоб відповідати на питання щодо відбору записів у базах даних: база даних, тобто інформація про спам і небажаний вміст може бути записана в базу даних для подальшого аналізу або застосування заходів проти таких ситуацій в майбутньому.

Програма реалізує різноманітні функції для управління чатом та забезпечення безпеки користувачів. Це включає в себе можливість додавання користувачів до чорного списку, блокування отримання повідомлень від певних користувачів, реєстрацію нових користувачів у системі, надсилання повідомлень у чаті, перевірку повідомлень на наявність спаму та небажаного контенту, видалення користувачів з чорного списку, а також отримання повідомлень з бази даних для відображення в чаті.

Ці функції допомагають забезпечити безпечне та зручне спілкування користувачів у чаті, мінімізуючи вплив спаму та небажаного контенту.

```
export const addToBlackList = async (username, reason) => {
    try {
        const existingBlacklistedUser = await BlacklistedUserModel.findOne({ username });
        if (existingBlacklistedUser) {
            console.log('User already blacklisted:', username);
            return;
        }
        const blacklistedUser = new BlacklistedUserModel( doc: { username, reason });
        await blacklistedUser.save();

        console.log('User added to blacklist:', username);
    } catch (error) {
        console.error('Error adding user to blacklist:', error);
        throw error;
    }
};
```

Малюнок 3.3 – Функція додавання користувачів у «чорний» список

```
export const loginUser = async (req, res) => {
    const { username, password } = req.body;

    try {
        const user = await UserModel.findOne({ username });

        if (!user || !(await bcrypt.compare(password, user.password))) {
            return res.status(401).json({ message: 'Invalid credentials' });
        }

        const authToken = jwt.sign( payload: { userId: user._id }, secretOrPrivateKey: 'secretKey', options: { expiresIn: '1h' });

        res.json({ authToken });
    } catch (error) {
        console.error('Error during login:', error);
        res.status(500).json({ message: 'Error during login' });
    }
};
```

Малюнок 3.4 - Функція входу користувача у систему

```
export const registerUser = async (req, res) => {
    const { username, email, password } = req.body;

    try {
        if (!password) {
            return res.status(400).json({ message: 'Password is required' });
        }

        const hashedPassword = await bcrypt.hash(password, salt: 10);
        const newUser = new UserModel( doc: { username, email, password: hashedPassword });

        await newUser.save();

        res.status(201).json({ message: 'User registered successfully' });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Error registering user' });
    }
};
```

Малюнок 3.5 – Функція реєстрації користувача

3.4. Методика роботи користувача

Користувач спочатку увійде в систему, використовуючи свій обліковий запис або зареєструється як новий користувач. Потім він матиме можливість переглядати та взаємодіяти з чатом, надсилаючи та отримуючи повідомлення у спільному чаті. Над користувачем також можуть виконуватися додаткові дії, такі як блокування його та отримання повідомлень від усіх користувачів або додавання інших користувачів до чорного списку. Він може також переглядати свою власну інформацію та відстежувати активність у чаті.

Висновки

У результаті розробки було створено чатову систему з функціоналом фільтрації спаму та небажаного контенту. Це дозволяє користувачам спілкуватися без перешкод від небажаних повідомлень. Реалізація бази даних за допомогою Mongoose дозволяє зберігати дані користувачів, повідомлень та інших параметрів ефективно та зручно. Використання Node.js та React дозволяє створювати швидкі та реактивні веб-додатки з високою продуктивністю. Технології фільтрації тексту та алгоритми, використані в системі, дозволяють ефективно виявляти спам та небажаний контент, забезпечуючи безпеку та комфорт користувачів. Під час розробки було також звернуто увагу на зручність та ергономіку інтерфейсу користувача, щоб забезпечити приємний досвід взаємодії з додатком. Функції реєстрації та входу були реалізовані з мінімальним навантаженням для користувача, щоб спростити процес отримання доступу до чату. Крім того, велика увага приділялася безпеці даних та конфіденційності, забезпечуючи надійний захист особистої інформації користувачів.

Додатково, в процесі розробки була здійснена оптимізація продуктивності системи, щоб забезпечити швидку відповідь на запити користувачів та знизити час завантаження сторінок. Розроблені функціональні та нефункціональні вимоги були успішно виконані, що дозволило створити високоякісний та функціональний продукт, задовольняючи потреби користувачів у спілкуванні в онлайн-середовищі.

В цілому, розробка була успішною і відповідає вимогам сучасного чатового додатку.

Перелік посилань

1. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf
2. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media. https://tjzhifei.github.io/resources/NLTK.pdf
3. Jurafsky, D., & Martin, J. H. (2019). Speech and Language Processing (3rd ed.). Pearson. https://web.stanford.edu/~jurafsky/slp3/
4. Segaran, T. (2007). Programming Collective Intelligence: Building Smart Web 2.0 Applications. O'Reilly Media. https://www.oreilly.com/library/view/programming-collective-intelligence/9780596529321/

Додаток А

ChatRoom.tsx

```
import React, { useState, useEffect, useCallback } from 'react';
import axios from 'axios';
import RegistrationForm from "./RegistrationForm";
import { Button, Paper, TextField, Typography } from "@mui/material";
import styles from '../styles/Home.module.css'
import LoginForm from "./AuthForm";
import Chat from "./Chat";

interface Message {
    id: number;
    text: string;
    username: string;
}

interface User {
    id: number;
    username: string;
    nickname: string;
}

const ChatRoom: React.FC = () => {
    const classes = styles;

    const [currentUser, setCurrentUser] = useState<User | null>(() => {
        const storedUser = localStorage.getItem('currentUser');
        try {
            return storedUser ? JSON.parse(storedUser) : null;
        } catch (error) {
            console.error('Error parsing stored user:', error);
            return null;
        }
    });

    const [name, setName] = useState<string>('');

    const [showRegistrationForm, setShowRegistrationForm] =
useState<boolean>(false);
    const [showLoginForm, setShowLoginForm] = useState<boolean>(false);

    useEffect(() => {
        const authToken = localStorage.getItem('authToken');
        if (authToken && !currentUser) {
            axios.get('http://localhost:3001/api/users', {
                headers: {
                    Authorization: `Bearer ${authToken}`
                }
            }).then(response => {
                setCurrentUser(response.data);
                setName(response.data.name);
            }).catch(error => {
                console.error('Error fetching user data:', error);
            });
        }
    }, [currentUser]);

    const handleRegister = useCallback((userData: { username: string; email:
string; password: string }) => {
        setShowRegistrationForm(false);
        axios.post('http://localhost:3001/api/register', userData)
            .then((response) => {
                console.log(response.data); // Log the response data to check
its structure
```

```
                // Proceed with handling the response data as needed
                const { authToken, user } = response.data; // Adjust this line
based on the actual response structure
                localStorage.setItem('authToken', authToken);
                setCurrentUser(user);
                localStorage.setItem('currentUser', JSON.stringify(user)); //
Save user data to local storage
                setName(user.username);
            })
            .catch((error) => {
                console.error('Error creating user:', error);
            });
    }, [setCurrentUser, setName]);

    const handleLoginSuccess = useCallback((authToken: string, username: string)
=> {
        setShowLoginForm(false);
        localStorage.setItem('authToken', authToken);
        axios.get(`http://localhost:3001/api/user/${username}`, {
            headers: {
                Authorization: `Bearer ${authToken}`
            }
        }).then(response => {
            const newUser: User = response.data;
            setCurrentUser(newUser)
            localStorage.setItem('currentUser', JSON.stringify(newUser));
            setName(response.data.username);
            localStorage.setItem('username', name);
        }).catch(error => {
            console.error('Error fetching user data:', error);
        });
    }, [setCurrentUser, setName]);

    const handleShowRegistrationForm = () => {
        setShowRegistrationForm(true);
        setShowLoginForm(false);
    };

    const handleShowLoginForm = () => {
        setShowLoginForm(true);
        setShowRegistrationForm(false);
    };

    const handleLogout = () => {
        localStorage.removeItem('currentUser');
        localStorage.removeItem('authToken');
        localStorage.removeItem('username');
        setCurrentUser(null);
        setName('');
    };

    return (
        <Paper className={classes.root} elevation={3}>
            <Typography variant="h5" className={classes.welcomeMessage}>
                Chat Room
            </Typography>
            {!currentUser && !showRegistrationForm && (
                <div>
                    <Button onClick={handleShowRegistrationForm}>Sign
up</Button>
                    <Button onClick={handleShowLoginForm}>Sign in</Button>
                </div>
            )}

            {showRegistrationForm && (
```

```
                <RegistrationForm
                    onRegister={(userData) => handleRegister({
                        username: userData.name,
                        email: userData.nickname,
                        password: userData.password
                    })}
                />
            )}
            {showLoginForm && (
                <LoginForm onLogin={handleLoginSuccess} />
            )}

            {currentUser && (
                <div>
                    <Chat currentUser={currentUser} username={name} />
                    <Button onClick={handleLogout}>Log out</Button>
                </div>
            )}
        </Paper>
    );
};

export default ChatRoom;
```

Chat.tsx

```
import React, { useEffect, useState } from 'react';
import { Button, TextField, Typography } from '@mui/material';
import styles from '../styles/Home.module.css';
import axios from 'axios';
import { v4 as uuidv4 } from 'uuid';
import Filter from 'bad-words';
import rusBadWords from '../rusbadwords.json';

interface Message {
    _id: string;
    text: string;
    username: string;
}

interface User {
    id: number;
    username: string;
    nickname: string;
}

interface ChatProps {
    currentUser: User;
    username: string;
}

const Chat: React.FC<ChatProps> = ({ currentUser, username }) => {
    const classes = styles;
    const [messages, setMessages] = useState<Message[]>([]);
    const [newMessage, setNewMessage] = useState<string>('');
    const filter = new Filter();
    const rusFilter = new Set(rusBadWords);
    useEffect(() => {
        const fetchMessages = async () => {
            try {
                const response = await
axios.get('http://localhost:3001/api/messages');
                const storedMessages = response.data;
                setMessages(storedMessages);
            } catch (error) {
```

```
                console.error('Error fetching messages:', error);
            }
        };

        fetchMessages();
    }, [messages]);

    useEffect(() => {
        return () => {
                localStorage.setItem('chatMessages', JSON.stringify(messages));
        };
    }, [messages]);


    const addToBlacklist = async (username: string, reason: string) => {
        try {
            const authToken = localStorage.getItem('authToken');
            await axios.post('http://localhost:3001/api/blacklist', { username,
reason }, {
                headers: {
                    Authorization: `Bearer ${authToken}`
                }
            });
            console.log(`User ${username} added to blacklist for reason:
${reason}`);
        } catch (error) {
            console.error('Error adding user to blacklist:', error);
        }
    };

    const handleSendMessage = () => {
        if (!newMessage.trim()) {
            console.warn('Message text is empty');
            return;
        }

        if (currentUser) {
            const profaneWordCount = countProfaneWords(newMessage);
            if (profaneWordCount > 2) {
                addToBlacklist(currentUser.username, 'Excessive profanity');
                alert('Excessive profanity detected. Your message cannot be
sent.');
                return;
            }

            const censoredMessage = censorMessage(newMessage);

            axios.post('http://localhost:3001/api/messages', { text:
censoredMessage, userId: currentUser.id, username })
                .then(response => {
                    const newMessageObj: Message = response.data;
                    setMessages(prevMessages => [...prevMessages,
newMessageObj]);
                    setNewMessage('');
                })
                .catch(error => {
                    console.error('Error sending message:', error);
                });
        } else {
            alert('You need to log in first.');
        }
    };

    let profaneCount = 0;
    const countProfaneWords = (message: string): number => {
        const words = message.split(/\s+/);
```

```
        let profaneCount = 0;

        words.forEach(word => {
            if (filter.isProfane(word) || rusFilter.has(word.toLowerCase())) {
                console.log(`Processing word: ${word}`);
                console.log(`Profane word detected: ${word}`);
                profaneCount++;
                console.log(`Total profane word count: ${profaneCount}`);
            }
        });

        return profaneCount;
    };


    const handleDeleteMessage = async (messageId: string) => {
        console.log("Deleting message with ID:", messageId);
        const authToken = localStorage.getItem('authToken');

        try {
            await
axios.delete(`http://localhost:3001/api/messages/${messageId}`, {
                headers: {
                    Authorization: `Bearer ${authToken}`
                }
            });

            setMessages(prevMessages => prevMessages.filter(message =>
message._id !== messageId));
        } catch (error) {
            console.error('Error deleting message:', error);
            alert('Failed to delete message. Please try again later.');
        }
    };

    const censorMessage = (message: string) => {
        return message
            .split(/\s+/)
            .map(word => {
                if (filter.isProfane(word) || rusFilter.has(word.toLowerCase()))
{
                    addToBlacklist(currentUser.username, 'inappropriate words');
                    return '*'.repeat(word.length);
                } else {
                    return word;
                }
            })
            .join(' ');
    };

    return (
        <div className={classes.chatContainer}>
            <Typography variant="body1" paragraph
className={classes.welcomeMessage}>
                Welcome, {currentUser.username} ({currentUser.id})!
            </Typography>

            <ul className={classes.messageList}>
                {messages.map(message => (
                    <li key={message._id} className={classes.messageItem}>
                        <div className={classes.messageContent}>
                            <strong>{message.username}: </strong>
                            {message.text}
                        </div>
                        {message.username === username && (
                            <Button variant="outlined" color="error" onClick={()
```

```
=> handleDeleteMessage(message._id)}>
                                Delete
                            </Button>
                        )}
                    </li>
                ))}
            </ul>

            <div className={classes.inputContainer}>
                <TextField
                    className={classes.input}
                    type="text"
                    placeholder="Type your message..."
                    value={newMessage}
                    onChange={e => setNewMessage(e.target.value)}
                />
                <Button variant="contained" color="primary"
onClick={handleSendMessage} className={classes.sendButton}>
                    Send
                </Button>
            </div>
        </div>
    );
};

export default Chat;
```