Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія аналізу текстів на наявність образливих висловлювань»

Виконав: студент 2-го курсу, групи 1КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Шинкаренко О.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: к.т.н., доцент каф. АІІТ
Іванов Ю.Ю.
(прізвище та ініціали)
«15» 12 2022 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.

14.09 **2022 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шинкаренко Олег Олександрович
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія аналізу текстів на наявність образливих висловлювань
керівник роботи к.т.н., доцент кафедри КН Сілагін О.В.
затверджені наказом вищого навчального закладу від "14" 09 2022 року №203
2. Строк подання студентом роботи 18 листопада 2022 року

3. Вихідні дані до роботи:
Мова об'єктно-орієнтованого програмування, браузерне середовище роботи додатку, кількість критеріїв класифікації – 7 од., мова текстів, що класифікуються – англійська, легкодоступність додатку за посиланням, відповідність стандарту ODBS, зашифровані API-ключі.
4. Зміст текстової частини:
Вступ, аналіз сучасного стану розвитку систем аналізу текстів на наявність образливих висловлювань, розробка інформаційної технології аналізу текстів на наявність образливих висловлювань, програмна реалізація інформаційної технології аналізу текстів на наявність образливих висловлювань, економічна частина, висновки, список використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Загальна структурна схема інформаційної технології, схема алгоритму роботи модуля збереження та аналізу тексту, схема алгоритму підпроцесу валідації API-ключів, діаграма послідовності, діаграма розгортування, загальний вигляд інтерфейсних вікон, результати тестування розробленої програми та програм-аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціалита посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-3	Сілагін О.В., к.т.н., доц. каф. КН	14.09.2022	14.12.22
4	Нікіфорова Л. О., к. е. н., доц. каф. ЕПВМ	19.09.2022	12.12.22

7. Дата видачі завдання 14.09 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ етапу	Назва етапу	Термін виконання		Очікувані результа
		початок	кінець	
1	Аналіз сучасного стану розвитку систем аналізу текстів на наявність образливих висловлювань	14.09.2022р.	1.10.2022р.	Аналітичний огляд літературних джер задачі досліджень, розділ 1
2	Обґрунтування методу розв'язання задачі, обґрунтування вибору інструментів технічної реалізації	2.10.2022р.	16.10.2022р.	Моделі, розділ 2
3	Проектування інформаційної технології. Програмна реалізація інформаційної технології. Аналіз результатів тестування	17.10.2022р.	07.11.2022р.	Розділ 2, 3
4	Підготовка економічної частини	08.11.2022р.	21.11.2022р.	Розділ 4
5	Апробація та/або впровадження результатів дослідження	23.11.2022р.	01.12.2022р.	Тези доповідей/ак впровадження
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.12.2022р.	14.12.2022р.	Пояснювальна записка, графічний матеріал, презента

Студент _____ (підпис) Шинкаренко О.О.

Керівник роботи _____ (підпис) Сілагін О.В.

АНОТАЦІЯ

УДК 004.8

Шинкаренко О.О. Інформаційна технологія аналізу текстів на наявність образливих висловлювань. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма - комп'ютерні науки. Вінниця: ВНТУ, 2022. 111 с.

На укр. мові. Бібліогр.: 32 назв; рис.: 24; табл. 9.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для аналізу текстів на наявність образливих висловлювань. Були розглянуті та проаналізовані існуючі програмні рішення і їх функціональні можливості, та обрано гібридний режим функціонування. Було проаналізовано різні підходи до вирішення задачі аналізу текстів шляхом класифікації та обрано універсальний кодувальник речень на основі трансформера через його високу точність класифікації. Було проаналізовано різні алгоритми шифрування API-ключів та обрано алгоритм SHA-2. Було спроектовано програму аналізу текстів на наявність образливих висловлювань, написану мовою програмування JavaScript з використанням бібліотеки React для клієнтської частини, Node.js і Express для серверної частини та СУБД MongoDB для управління базами даних.

Графічна частина складається з 7 плакатів.

У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення, яка складає 909295 гривень, спрогнозовано орієнтовану величину витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 0.92 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: класифікація тексту, універсальний кодувальник речень, веб-клієнт, API-ключ, клієнт-серверна архітектура.

ABSTRACT

Shynkarenko O.O. Information technology of text analysis for the presence of offensive statements. Master's thesis in the specialty 122 - computer sciences, educational program - computer science. Vinnytsia: VNTU, 2022. 111 p.

In Ukrainian language. Bibliographer: 32 titles; fig.: 24; table 9.

This master's thesis is devoted to the development of software for text analysis for the presence of offensive statements. The existing software solutions for text analysis, the hybrid method of lifecycle process was chosen. Different approaches of text classification were analyzed, and the universal sentence encoder algorithm based on the transformer architecture, was substantiated. Different method of API-keys encryption were analysed and SHA-2 algorithm was chosen. A program for text analysis for the presence of offensive statements, written in the JavaScript programming language using the React library for the client part, Node.js and Express for the server part and MongoDB for managing the database.

The graphic part consists of 7 posters.

The economic section calculates the amount of costs for the development and manufacture of a new technical solution, which is 909295 hryvnia, predicted the estimated cost of each of the cost items, calculated net profit, payback period for the manufacturer 0.92 years and the economic effect for consumers using this development.

Keywords: text classification, universal sentence encoder, web-client, API-key, client-server architecture.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сьогодні більшість онлайн-ресурсів, таких як платформи розповсюдження новин, платформи відповідей на запитання та обміну досвідом, соціальні мережі, форуми, а також онлайн-ігри включають в себе простір для спілкування, висловлення думок. З ціллю збереження поважного ставлення до співрозмовника та плідного обговорення тем, компанії наймають контент-модераторів, які слідкують за тим, аби правила обговорення платформи виконувались. У разі невиконання таких правил, модератори видаляють дописи користувачів та, за необхідності, блокують і самого користувача.

Відповідно, модератори повинні правильно класифікувати та визначити причину видалення коментаря та повідомити про неї автора. Така робота є рутинною та вимагає великих часових затрат на прийняття рішень, а зі збільшенням кількості коментарів для розгляду виникає потреба у наймі більшої кількості працівників [1–5]. Процес можливо пришвидшити та спростити, якщо відсортовувати коментарі по ступеню ймовірності порушення правил спілкування платформи автоматизовано та пропонувати на розгляд модераторів у пріоритеті більш спірні випадки порушення правил. Для пошуку ймовірностей оптимальним рішенням є використання штучного інтелекту для аналізу тексту. З вищенаведеного можна зробити висновок, що проблема аналізу текстів для подальшої модерації є актуальною та потребує подальшого дослідження. Стоїть проблема створення інформаційної технології із веб-інтерфейсом та механізмом децентралізованого та ізольованого доступу до даних клієнта для автоматизованого аналізу тексту на елементів, що порушують правила платформи, та пріоритетного пропонування на розгляд працівникам-модераторам на розгляд спірних, неоднозначних випадків порушення. Тому створення інформаційної технології аналізу текстів на наявність образливих висловлювань є актуальним.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є розширення функціональних можливостей в аналізі текстів на наявність образливих висловлювань та покращення метрик продуктивності програмного продукту за рахунок використання механізму ідентифікації програмного інтерфейсу.

Для досягнення поставленої мети необхідно виконати такі задачі:

- аналіз сучасного стану розвитку систем аналізу текстів на наявність образливих висловлювань;
- обґрунтування методу розв’язання задачі;
- проектування інформаційної технології аналізу текстів на наявність образливих висловлювань;
- програмна реалізація інформаційної технології аналізу текстів на наявність образливих висловлювань;
- тестування інформаційної технології аналізу текстів на наявність образливих висловлювань та аналіз результатів тестування;
- розрахунок витрат на розробку, розрахунок чистого прибутку, визначення конкурентоспроможності розробки.

Об’єктом дослідження є процеси аналізу тексту на наявність образливих висловів та пріоритезації.

Предметом дослідження є алгоритми та програмне забезпечення, що організовує процес аналізу текстів на наявність образливих висловлювань.

Методи дослідження - методи аналізу тексту, методи масштабування та інтеграції програмного забезпечення, методи та підходи до розробки систем

штучного інтелекту, методи та підходи до розробки веб-орієнтованих програмних додатків, методи об’єктно-орієнтованого програмування.

Наукова новизна одержаних результатів полягає в наступному:

Розроблено інформаційну технологію аналізу текстів, що відрізняється від існуючих розширенням функціональних можливостей за допомогою моделі поєднання інтелектуальної системи із механізмом доступу до даних за механізмом ідентифікації програмного інтерфейсу, що забезпечило покращення процесу поширення та інтеграції системи в існуючі програмні рішення.

Практичне значення одержаних результатів полягає у наступному:

1. Розроблено алгоритм роботи інформаційної технології аналізу текстів на наявність образливих висловлювань.

2. Розроблено структуру програмного забезпечення на основі ідентифікації програмного інтерфейсу для спрощеного процесу інтеграції та розповсюдження системи, здійснено програмну реалізацію системи згідно розробленій структурі.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Апробація роботи. Робота апробувалась на наступних конференціях:

– Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації 2022 року, доповідь «Розробка інформаційної технології аналізу текстів на наявність образливих висловлювань» [1];

Публікації: електронні тези науково-технічної конференції факультету інтелектуальних інформаційних технологій та автоматизації 2022 року [1]; стаття III Міжнародної науково-практичної конференції 2022 року [2].

1 АНАЛІЗ СУЧАСНОГО СТАНУ РОЗВИТКУ СИСТЕМ АНАЛІЗУ ТЕКСТІВ НА НАЯВНІСТЬ ОБРАЗЛИВИХ ВИСЛОВЛЮВАНЬ

1.1 Постановка задачі і формулювання вимог до інформаційної технології аналізу текстів на наявність образливих висловлювань

Обмін інформацією в Інтернеті є одним з найважливіших способів поширення інформації і комунікації, у той же час токсичні та образливі висловлювання, що можуть становити загрозу, нести в собі образу, є однією з найбільших проблем у створенні інформативного середовища.

Задача модерації повідомлень і текстів користувачів на різних онлайн-платформах, таких як сайти новин, тематичні блоги та форуми, соціальні складові комп'ютерних ігор включає в себе багато факторів та має різні підходи до вирішення в залежності від цілей, що ставить перед собою компанія. Так, модерацію контенту проводять на основі політики компанії та користувацької згоди на використання продуктів [2]. Відповідно, контент можуть модерувати на наявність:

- висловлювань, що несуть в собі образи на основі віросповідання;
- нецензурної лексики;
- висловлювань, що несуть негатив на основі гендерних стереотипів;
- висловлювань, що несуть в собі образи на основі расової дискримінації;
- висловлювань, що несуть образи на основі національної дискримінації;
- висловлювань, що спонукають до насильства;
- повідомлень, що містять інформацію про продаж сумнівних або заборонених товарів або послуг;
- повідомлень, що несуть завідома провокативний характер та розпалюють ворожнечу;
- повідомлень що містять неприйнятну інформацію сексуального характеру.

Вищенаведений список є далеко неповним, але вже відкриває широкий спектр проблем, які можуть викликати повідомлення наведеного характеру. Компанії застосовують різні моделі санкцій в залежності від важкості порушення. Так, подібні повідомлення можуть блокуватися, видалятися, помічатися для інших користувачів, як неприйнятні, приводити до тимчасових або постійних блокувань користувачів, що розмістили дані повідомлення. Процес блокування може стосуватися як облікового запису, так і IP-адреси, та, навіть, MAC – адреси користувача, що порушив політику користування платформою.

Тема токсичності може розглядатися в трьох різних напрямках. Перше — токсичність як напрямок національної безпеки. В Європі майже в усіх країнах створені організації, які займаються стратегічною національною безпекою. Друге — це ставлення аудиторії до експертів. В умовах доступу до інформації та свободи висловлення думок дуже важливою є медіаграмотність.

Третє — це сегментація простору. Формуються бульбашки, зростає роль емоцій. Завдяки певним мовним формам і зворотам можна створювати віртуальні емоції, які використовуються з токсичною метою. Таким чином, ціллю токсичного коментаря може бути не лише прямий зміст сказаного, але й емоції, що передає даний текст і подальша ретрансляція її на інші сфери життя [3].

Процес модерації неможливо повністю автоматизувати, так як використання інформаційних технологій може призвести до неочікуваних результатів, таких як блокування дописів і повідомлень, що насправді не несли в собі ніяких порушень, але по тій чи іншій причині були сприйняті системою як такі, що порушили правила. Це відповідно викликає велику кількість звернень користувачів до модераторів та адміністраторів платформ з ціллю зміни рішення, прийнятого автоматизованою системою. На кожну таку заявку виділяється багато людського часу та, відповідно, додаткові грошові витрати. З іншої сторони, повністю людська праця також не є ефективною та призведе до ще більших фінансових збитків. У разі людської модерації набір дописів і

коментарів, як правило, звужується до тих, на які поскаржились самі користувачі. У такому випадку можлива необ’єктивна оцінка вмісту того чи іншого коментаря та безпідставне блокування повідомлень, що не порушували політику. Тому, необхідно знаходити компроміс у вигляді часткової автоматизації процесу аналізу коментарів, але при цьому рекомендовано залишити прийняття остаточного рішення щодо блокування за людиною-фахівцем. Необхідно зазначити також складність процесу автоматизації через відсутність універсальних критеріїв оцінки тексту, так як кожна платформа має власну інформаційну політику. Це призводить до необхідності створення унікальних програмних рішень для кожного окремого випадку [4].

Вирішити таку проблему може багатокритеріальний аналіз тексту на основні види порушень. Такий підхід не покриває найбільш точно потреби користувачів, але забезпечує достатній рівень універсальності для основного аналізу.

Для автоматизованого процесу аналізу тексту можна використати методології аналізу природної мови (Natural Language Processing) та машинного навчання на основі нейронних мереж. Методологія машинного навчання. Машинне навчання є легшим у застосуванні через велику існуючу базу класифікованих за різними критеріями текстів, в тому числі за критерієм загального порушення правил платформи. Методи класифікації також дозволяють застосовувати різні ступені мір, що необхідно прийняти по відношенню до допису чи користувача. Так, до прикладу, текст можна приховати для користувачів, що не досягли певного віку [5].

Визначимо задачі і вимоги до інформаційної технології аналізу текстів на наявність образливих висловлювань. Задача: розширення функціоналу аналізу текстів на наявність образливих висловлювань шляхом створення системи адміністрування дописів із пріоритезацією та візуалізацією даних для адміністратора та механізмом інтеграції в існуючі програмні рішення за допомогою API-ключів.

Інформаційна технологія має реалізовувати такі функції:

- авторизація користувачів;
- отримання дописів користувачів;
- публікацію дописів користувачів;
- аналіз та збереження дописів користувачів у базі даних;
- адміністрування дописів через панель керування;
- надання детальної інформації про допис та його критерії загрози;
- блокування та розблокування дописів користувачів;
- доступ до фрагменту бази даних за API-ключем;
- доступ до декількох клієнтських рішень за закріпленими за користувачем API-ключами;
- шифрування та перевірку наявності відповідного API-ключа в базі даних;
- можливість переключення між панелями адміністрування декількох клієнтських рішень у реальному часі;
- трансляція змін інформації про допис для спілкування із клієнтом, в який було інтегровано інформаційну технологію;

Також необхідно провести тестування розробленої інформаційної технології на наявність помилок, безпечність та коректність роботи. Розроблене програмне забезпечення має функціонувати у всіх браузерах. Усі наявні елементи повинні чітко працювати без будь-яких помилок.

Початком розробки інформаційної технології аналізу текстів на наявність образливих висловлювань є вибір методів аналізу тексту для виявлення загрози у дописах. Далі – розробка загальної структурної схеми системи, її модифікація механізмом валідації API-ключів, розробка загальної схеми алгоритму роботи системи та алгоритму валідації API-ключів, створення UML-діаграм розгортування та послідовності для деталізації складових подальшої розробки. Наступним етапом є вибір технологій реалізації та безпосередня розробка інформаційної технології. Заключними етапами є тестування розробленої інформаційної технології і розробка інструкції користувача.

1.2 Аналітичний огляд відомих програмних рішень аналізу текстів на наявність образливих висловлювань

Проблему модерації вирішують велика кількість компаній як сервісних, так і таких, що розробляють програмне забезпечення. Розглянемо деякі з існуючих рішень вирішення проблеми модерації тексту.

Першим прикладом є система BattlEye, що спеціалізується на модерації спілкування в комп'ютерних іграх та слідкує за використанням нелегального програмного забезпечення з ціллю отримання переваги у грі. Система працює у реальному часі та є автоматизованою.

BattlEye не вимагає багато ресурсів процесора та оперативної пам'яті та працює у фоновому режимі під час виконання програми, в яку була інтегрована система. Інтеграція системи відбувається як на серверній, так і на клієнтській стороні. У своїй основі, BattlEye є проактивною системою захисту

Функціями і перевагами BattlEye є:

- Швидке динамічне та постійне сканування системи гравця в режимі користувача та режимі ядра з використанням інноваційних, складних специфічних і евристичних/загальних процедур виявлення та аналізу читів для максимальної ефективності;
- Прийняття рішення і мір проходить у реальному часі
- Комунікація відбувається по каналах комунікації гри і не потребує додаткових;
- Система автоматичного оновлення;

Перевагою такої системи є велика швидкість прийняття рішень та відносна універсальність, так як компанії, що використовують послуги BattlEye погоджуються з правилами модерації системи [6]. У головній перевазі системи криється і найвагоміший недолік. Ним є автоматичне блокування підозрілих дій та сумнівних коментарів, що не завжди відпрацьовує правильно, на основі семантичної схожості коментаря, чи наприклад, різне тлумачення слів на двох

різних мовах. Тому для оскарження такого рішення необхідно писати звернення у службу підтримки, де кожен з випадків розглядається окремо. В окремих випадках оскарження рішення є неможливим та незмінним. Загальний вигляд вікна BattlEye наведено на рисунку 1.1.

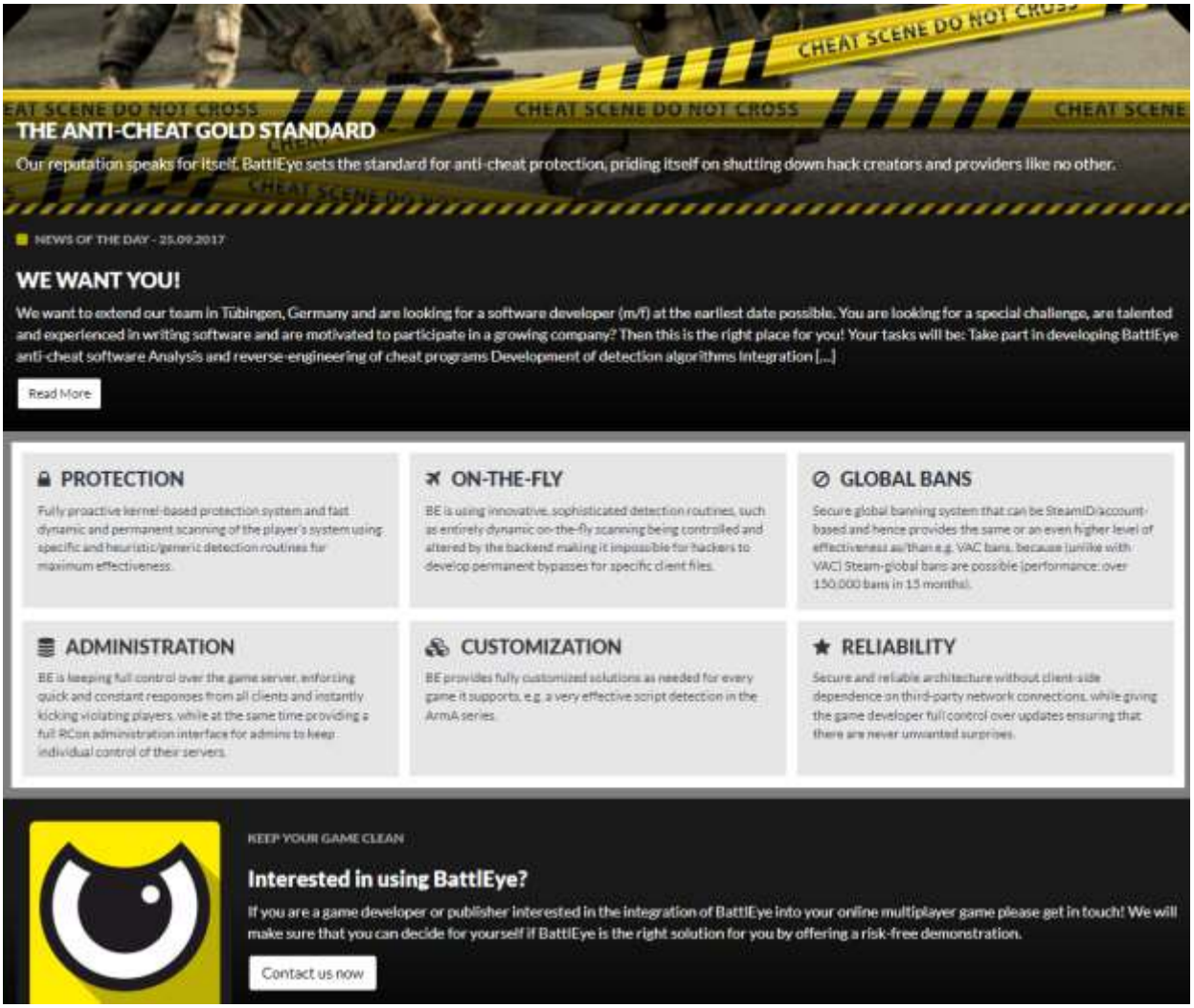


Рисунок 1.1 – Загальний вигляд вікна платформи BattlEye

Іншим аналогом рішення, що розглядається, є Perspective API – програмний інтерфейс аналізу текстів за встановленими критеріями. Даний інтерфейс легко інтегрувати в існуючі програмні рішення та використовувати отримані дані для власних цілей. Основною перевагою інтерфейсу є інтеграція через Google Cloud. Крім того, інтерфейс в експериментальному режимі надає

можливість класифікувати коментарі за такими параметрами, як токсичність, «профанство», флірт. Інтерфейс надає лише інформацію про класифікацію коментаря [7]. Головним недоліком даної системи є відсутність власного функціоналу і візуалізації даних, отриманих в результаті аналізу, тому у випадках подальшої обробки отриманої інформації необхідно додатково реалізовувати власний функціонал, що несе за собою додаткові грошові та часові витрати, тому Perspective API можна розглядати як допоміжний інтерфейс для створення більш комплексної інформаційної технології, що використовуватиме API для класифікації. Загальний вигляд вікна платформи розповсюдження Perspective API наведено на рисунку 1.2.

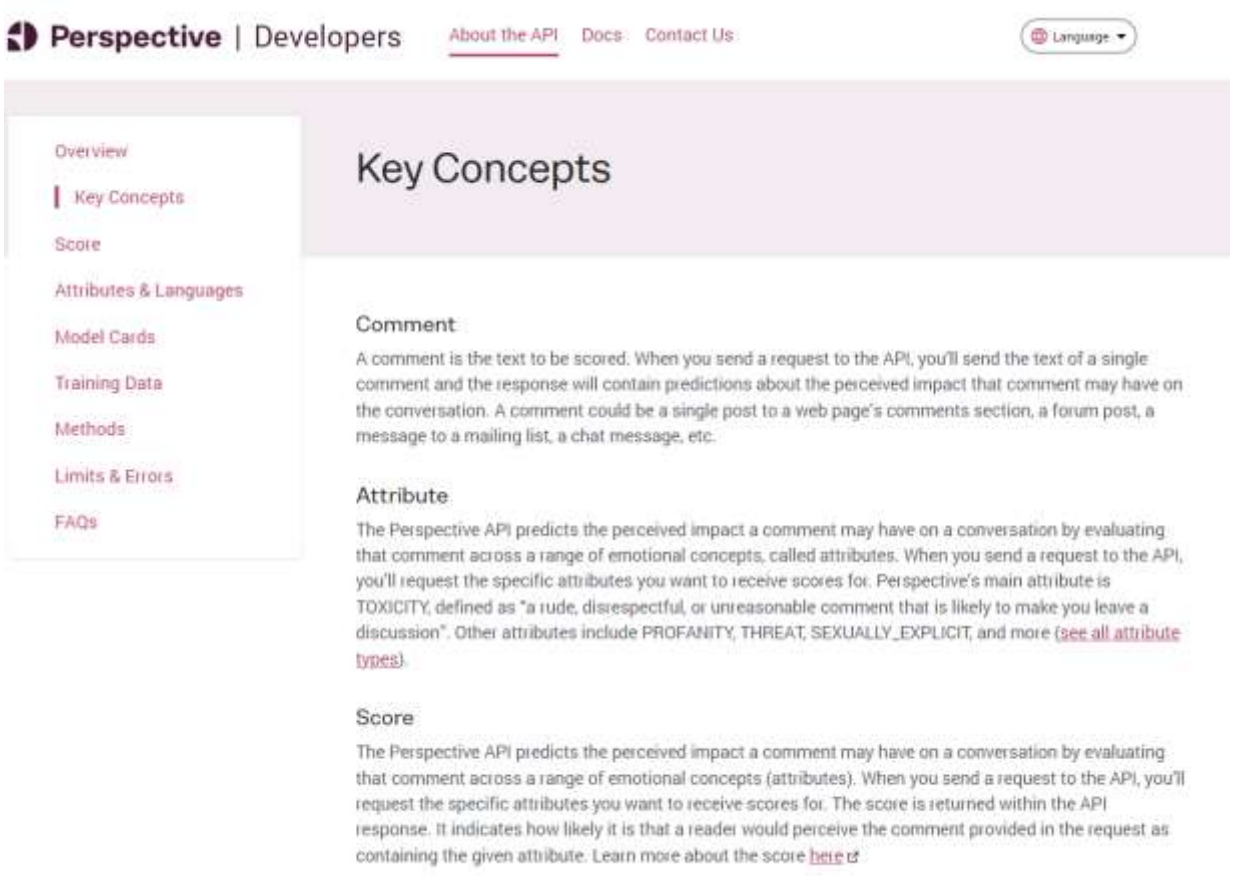


Рисунок 1.2 – Загальний вигляд вікна платформи розповсюдження Perspective API

PLNia – провайдер програмних інтерфейсів для аналізу текстів. На відміну від аналогів, що використовують технології на основі машинного навчання, PLNia використовує в основі деяких зі своїх інтерфейсів методи аналізу природної мови, що належить до класу методів штучного інтелекту.

PLNia надає послуги за наступними напрямками:

- аналізу настрою в тексті;
- знаходження ключових слів у тексті;
- формування висновку по змісту тексту;
- інтерфейс для аналізу на образливі висловлювання у тексті;
- визначення мови тексту;

Програмний інтерфейс аналізу на образливі висовлювання розповсюджується на основі механізму API-ключів, тому є таким, що легко масштабується. Вхідними даними інтерфейсу є текст, результатом роботи – прапорець істинності на належність тексту до образливих висловлювань.

В основі інтерфейсу для аналізу текстів на образливі висловлювання PLNia використовує поєднання методів аналізу природної мови та машинного навчання. Інтерфейс поширюється як RESTful API [8]. Головним недоліком даної системи є відсутність власного функціоналу і візуалізації даних, отриманих в результаті аналізу, тому у випадках подальшої обробки отриманої інформації необхідно додатково реалізовувати власний функціонал, що несе за собою додаткові грошові та часові витрати. Крім того, програмний інтерфейс не визначає клас образливого висловлювання, що ускладнює процес модерації за різного ступеня серйозності порушення різних класів образ у тексті. За використання PLNia, задача класифікації порушення все ще буде покладена на людину-модератора.

Загальний вигляд вікна платформи розповсюдження PLNia наведено на рисунку 1.3.

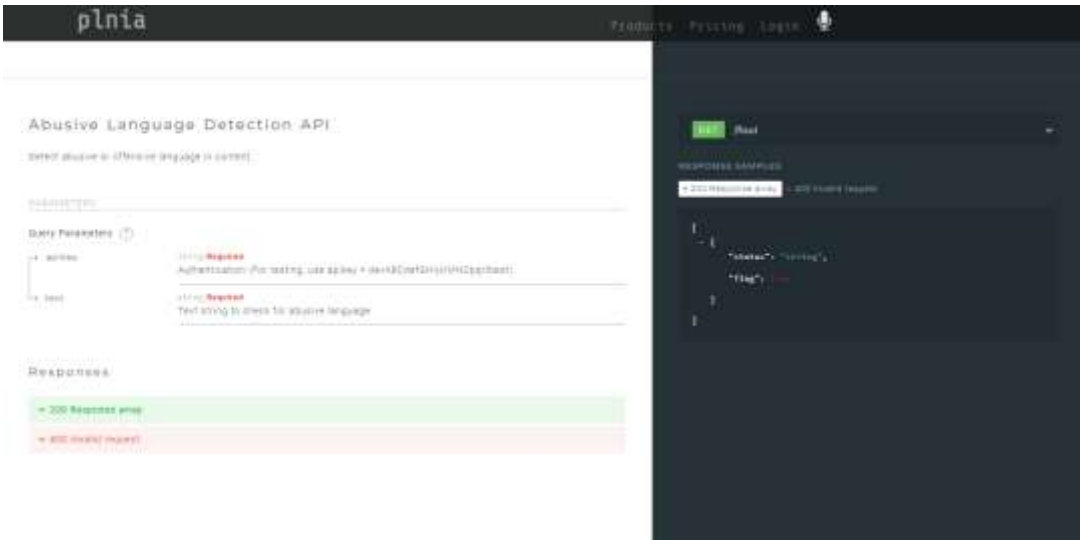


Рисунок 1.3 – Загальний вигляд вікна платформи розповсюдження PLNia

Деякі з вищенаведених комерційних рішень у тій чи іншій формі стикаються із проблемою масштабування. Із масштабуванням програмного забезпечення на більшу кількість користувачів також виникає проблема забезпечення децентралізації сховищ даних та доступу до них. Рішення «під ключ» із збільшенням кількості клієнтів зустрічають проблему росту необхідних ресурсів, як матеріальних, так і часових, на створення нового рішення. В той же час, універсальні рішення не надають індивідуальних можливостей при роботі із проаналізованими даними, в тому числі і взагалі не забезпечують зберігання та модерацію проаналізованого тексту. З цього виникає проблема децентралізованого і ізольованого доступу до даних, для вирішення якої необхідно враховувати функціональну повноту системи та економічну доцільність впровадження, що має поєднуватись із універсальністю та можливістю масштабування на більшу кількість клієнтів та клієнтських рішень, враховуючи факт того, що один клієнт може мати декілька продуктів і постійний доступ до них.

Тому стоїть проблема створення інформаційної технології із веб-інтерфейсом та механізмом децентралізованого та ізольованого доступу до даних клієнта для автоматизованого аналізу тексту на елементів, що порушують

правила платформи, та пріоритетного пропонування на розгляд працівникам-модераторам на розгляд спірних, неоднозначних випадків порушення, із забезпеченням доступу до декількох клієнтських рішень за наявності таких.

Порівняльна характеристика наведених аналогів приведена у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика існуючих аналогів інформаційної технології аналізу текстів на наявність образливих висловлювань

Характеристика	BattlEye	Perspective API	PLNia
Автоматизована обробка з використанням методів штучного інтелекту	+	+	+
Остаточне прийняття рішення людиною	-	+	+
Наявність функціоналу для модерації проаналізованого тексту	+	-	-
Універсальність рішення	-	+	+
Візуалізація даних	-	-	-
Можливість повного забезпечення вимог замовника	+	-	-
Швидка інтеграція в існуючі програмні рішення	+	-	+
Класифікація загроз	+	+	-

Можна побачити, що жодна з існуючих систем не пристосована до задачі інтеграції в існуючі програмні рішення, деякі потребують додаткових витрат на

розробку та адаптацію, жодне з рішень не може бути використане для одночасно легкої інтеграції в існуючі продукти та надання функціоналу, достатнього для забезпечення модерації. Тому створення компромісної системи із моделлю простої інтеграції в існуючі програмні рішення може вирішити вищенаведені проблеми існуючих програмних рішень.. З цього можна сформувати основну задачу даної роботи – розробку компромісного рішення, що проводить автоматизовану перевірку контенту за допомогою штучного інтелекту та залишає фінальне рішення за фахівцем-модератором, пропонуючи йому ймовірності порушення правил за декількома критеріями та пріоритезацією коментарів на розгляд, що легко інтеграції системи в існуючі програмні продукти для швидшого впровадження і використання програмних продуктів, що збільшить обсяг зацікавлених споживачів та можливу фінансову вигоду від розповсюдження.

1.3 Висновок до розділу 1

В даному розділі було розглянуто проблему аналізу текстів на наявність образливих висловлювань, визначено основні труднощі даного процесу; проаналізовано існуючі рішення проблеми аналізу текстів та подальшої їх модерації, показано недоліки даних рішень та запропоновано компромісне рішення, що полягає у використанні штучного інтелекту для аналізу тексту і подальшого прийняття рішення людиною фахівцем, а також механізмом інтеграції через API – ключі; визначено задачі та вимоги до інформаційної технології аналізу текстів на наявність образливих висловлювань, що дозволяє перейти до етапу вибору методів аналізу та перетворення текстів для виконання інтелектуальної частини системи.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АНАЛІЗУ ТЕКСТІВ НА НАЯВНІСТЬ ОБРАЗЛИВИХ ВИСЛОВЛЮВАНЬ

2.1 Обґрунтування вибору методів аналізу та перетворення тексту

Основною задачею в автоматизованому процесі перевірки коментарів є аналіз та класифікація текстів. Опишемо сутність задачі класифікації.

Класифікація – процес групування та організації інформацію змістовно та систематично у стандартному форматі, що використовується для виявлення схожості ідей, подій, об’єктів, осіб, явищ [14]. Особливістю класифікації є те, що групи, на які мають бути поділені дані(класи) заздалегідь відомі та описані, тобто представляють собою скінчену дискретну множину категорій, до яких можна віднести об’єкти аналізу. При перенесенні задачі класифікації на аналіз тексту, можна виділити декілька основних підходів:

- класифікація окремо взятих слів або висловлювань у тексті;
- класифікація синтаксично зв’язаного тексту.

Серед вищезазначених підходів, метод аналізу окремо взятих слів або висловлювань є простішим у реалізації, тому є пріоритетним у разі відповідності вимогам аналізу. Подібний аналіз може забезпечити наївний Баєсовий класифікатор. Наївний Баєсовий класифікатор є одним з найпростіших алгоритмів класифікації. Формула для знаходження класу має вигляд:

$$c = \arg\max_{c \in C} P(c|o_1 o_2 \dots o_n) \prod P(o_i|c), \qquad (2.1)$$

де P – ймовірність належності, c – клас, C – множина класів, o_1, o_2 … o_n – ознаки строки(features).

Такий класифікатор надає достатню точність для задачі класифікації незалежних даних. Звідси і головний недолік – неможливість врахувати взаємодії

функцій, а, відповідно, якщо перенести на задачі аналізу тексту, неможливість аналізу синтаксично зв’язного тексту.

Друга група методів, а саме класифікація зв’язного тексту, включають декілька етапів. Першим етапом є перетворення вхідних даних у такі, що можуть використовуватись для подальшого аналізу нейронними мережами. У задачах аналізу тексту використовують векторне представлення слів(word embeddings). Методи векторного представлення слів дозволяють перетворити вхідний текст в малорозмірні вектори, що і називається векторним представленням [10, 18]. Такий метод дозволяє представити дані у більш компактному вигляді та представити зв’язки слів між собою(рис. 2.1).

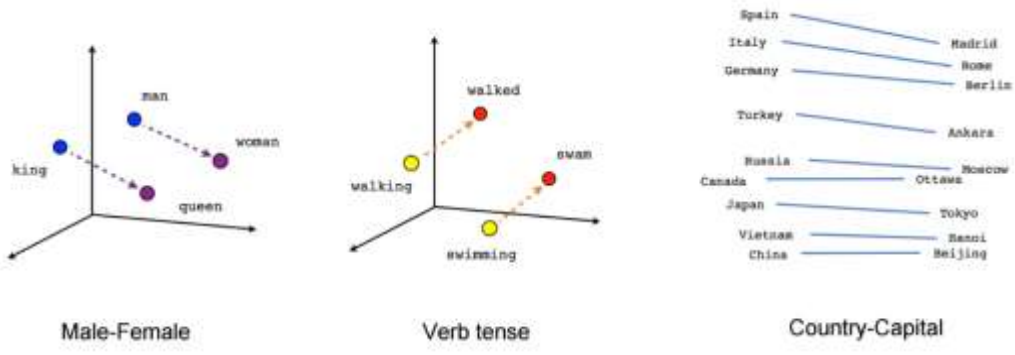


Рисунок 2.1 – Діаграми векторного представлення слів

Для векторного представлення слів використовуються різні моделі кодування. Розглянемо дві з них, а саме модель кодування речень на основі трансформера (transformer based encoding) та модель кодування на основі глибокої усереднювальної мережі (Deep Averaging Network) [17].

Трансформер використовує механізм уваги – підхід у машинному навчанні, що полягає у виділенні частини вхідних даних(регіонів зображень, фрагментів тексту), для більш детальної обробки. Це пов’язано з тим, що у більшості випадків для вирішення задачі класифікації не потрібно аналізувати усі пікселі зображення або увесь текст. Тим не менш, згорткові нейронні мережі витрачатимуть однакову кількість обчислювальних ресурсів для аналізу усіх

фрагментів. Відповідно у задачі класифікації тексту підграф на основі архітектури трансформатора використовує механізм уваги для розрахунку представлення слів, враховуючи як порядок так і зв’язок усіх інших слів. Дане представлення конвертується у кодувальний вектор фіксованої довжини шляхом обчислення поелементної суми представлення позиції кожного слова. На вхід кодувальнику подається токенізована строка на основі деревовидного корпусу Пенна (Penn Treebank tokenization). На виході отримується 512-вимірний вектор представлення. Така модель спроектована для загального використання. Недоліком даної моделі є збільшення часу розрахунків та використання пам’яті зі збільшенням довжини речення [12, 14, 15]. Друга модель використовує глибоку усереднювальну мережу (Deep Averaging Network, DAN), у якій вхідні представлення для слів спочатку асоціюються разом і потім проходять через глибоку нейронну мережу для створення векторних представлень. Процес асоціації необхідний для отримання просторового вектору речення, а не окремого слова. Відповідно, для вирішення цієї задачі використовуються функції композиції. Функції композиції діляться на два типи:

- невпорядковані;
- синтаксичні – враховуються порядок слів.

Синтаксичні функції композиції показують набагато кращий результат у багатьох задачах, але програють у витратах на розрахунок і часі навчання. Приклад роботи DAN наведено на рисунку 2.2. Основною перевагою DAN є те, що час на розрахунки лінійно залежний від довжини вхідної послідовності. Обидва підходи дозволяють отримати вищу точність ціною більшої складності моделі і витрати ресурсів. DAN демонструє дещо нижчу точність, ніж підхід із використанням трансформера, але є швидшою та менш ресурсноємкою [15-17].

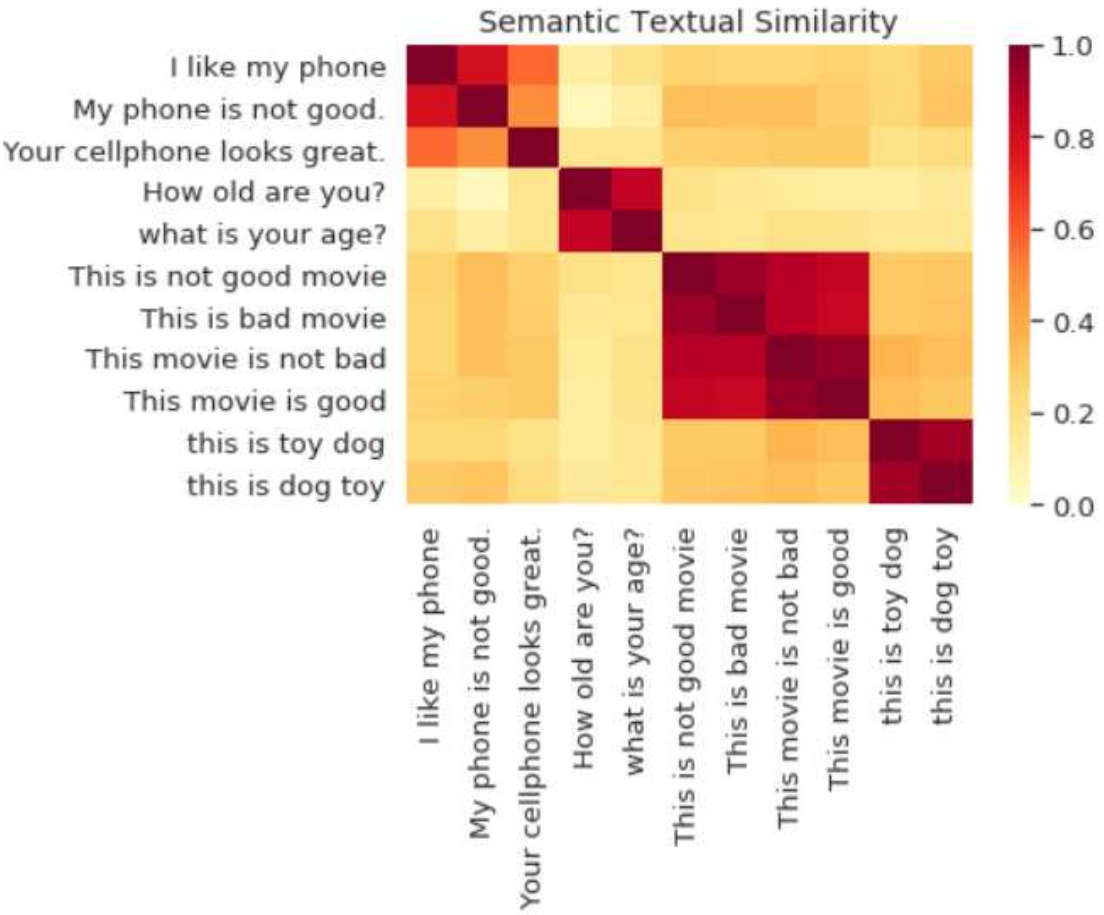


Рисунок 2.2 – Схема схожості векторів речень у результаті роботи DAN

Після формування векторного представлення речень проводиться аналіз отриманих даних та їх класифікація. Для цього використовуються згорткові нейронні мережі(Convolutional Neural Networks, CNN). Такі мережі дозволяють виділити характерні риси речень, отриманих після формування векторного представлення.

Головний принцип згорткової нейронної мережі – формування матриці ознак з вхідного вектора за допомогою матриці згортки. У випадку задачі класифікації тексту, достатньо одного шару згортки для отримання задовільного результату [12].

Для отримання кращого кінцевого результату можливим є поєднання рівня слів і рівня речень. Рівень схожості речень можна отримати за формулою

$$sim(u,v) = (1 - \arccos\left(\frac{\frac{u * v}{\|u\|\|v\|}}{\pi}\right). \quad (2.2)$$

Описаний підхід із використанням векторного представлення тексту та подальшої обробки за допомогою згорткової нейронної мережі лежить в основі універсального кодувальника речень (Universal Sentence Encoder), тому є доцільним вибір та подальша реалізація саме цієї моделі.

2.2 Архітектура універсального кодувальника речень

Універсальний кодувальник що дозволяє класифікувати синтаксично зв'язні висловлювання та речення. Кодувальник має дві імплементації та побудований на основі трансформера та глибокої усереднювальної мережі, що були розглянуті у попередньому пункті даного розділу.

Після завершення процедури векторного представлення тексту є можливим застосування відповідних нейронних мереж для подальшого аналізу. Приклад роботи універсального кодувальника наведено на рисунку 2.3.

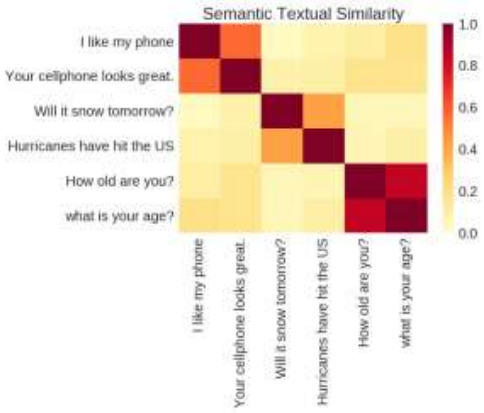


Рисунок 2.3 – Схема універсального кодувальника речень

У залежності від поставлених задач та вимог можливо використання обох варіацій кодувальника. У разі наявності більших технічних ресурсів та вимог

більш точного результату, доцільним є використання кодера на основі трансформатора. У разі допущення меншої точності обчислень, доцільним є використання кодера на основі DAN [11].

За допомогою згорткової нейронної мережі можливо обрахувати основні необхідні критерії оцінки речень. У випадку аналізу тексту на наявність образливих висловлювань, можна виділити такі типи образ:

- атака на особистість;
- образа;
- нецензурна лексика;
- критична токсичність;
- сексуальна неприйнятність;
- погроза;
- токсичність.

Відповідно, після проведення аналізу отримаємо набір коефіцієнтів відповідності одному з вищенаведених критеріїв, у разі отримання високого показника ймовірності(c >= 85%) висловлювання буде вважатись таким, що відноситься до відповідного класу образ. У разі отримання середнього коефіцієнта схожості (40% <= c < 85%) заданому класу повідомлення можна помітити як таке, що потребує додаткового дослідження. У разі отримання низького коефіцієнта(c< 40%) повідомлення вважатиметься таким, що не містить вищезазначений клас порушень, а отже є прийнятним і не потребує додаткової перевірки.

Крім того, є можливим установлення різних граничних значень ймовірності для різних класів повідомлень для отримання більш гнучкої оцінки загрози коментаря. Для отримання результуючого значення рівня загрози коментаря можна скористатися формулою:

$$k = \sum_{n}^{i} c_i * w_i; \ \sum_{n}^{i} w_i = 1, \qquad (2.3)$$

де c_i – коефіцієнт схожості, w_i – ваговий коефіцієнт критерію, n – кількість критеріїв.

Таким чином, розглянувши переваги і недоліки зазначених методів аналізу тексту, оберемо для проектування та реалізації універсальний кодувальник речень з векторним перетворенням речень на основі трансформатора для досягнення вищого показника точності, а також через необхідність аналізу синтаксично зв'язного тексту. Крім того, введемо граничні значення ймовірностей для кожного з критеріїв(класів) оцінки тексту на вміст неприйнятних висловлювань через різний ступінь загрози поширення відповідних повідомлень.

У даному розділі було розглянуто відомі методи аналізу текстів, а саме методи аналізу зв'язного та незв'язного тексту, було визначено переваги і недоліки підходів; було обрано універсальний кодувальник речень як методику, що забезпечує аналіз зв'язного тексту; запропоновано коефіцієнт загрози, що дозволяє отримати загальний рівень загрози коментаря в залежності від важливості критеріїв оцінки. Отримані результати дослідження дозволяють розробити інтелектуальну частину інформаційної технології та перейти до проектування інформаційної технології загалом.

2.3. Аналіз принципу роботи API-ключів та обґрунтування вибору методу шифрування API-ключів

Ключем програмного інтерфейсу, або API-ключем, називають унікальний ідентифікатор, що використовується для аутентифікації користувача, розробника, або виклику програми до програмного інтерфейсу (API). Найчастіше ключ програмного інтерфейсу використовується як ідентифікатор проекту, чи клієнтського рішення, що працює з API. Різні платформи можуть використовувати та впроваджувати API-ключі різними способами. Розглянемо способи використання API-ключів у web-середовищі.

До них відносяться наступні методи передачі за принципами HTTP:

- як параметр запиту, наприклад «/something?api_key=abcdef12345», даний метод є найбільш вразливим та легким до виявлення та використання API-ключа;
- в заголовку запиту;
- як cookie-запис.

Ключі не вважаються безпечними та захищеними, так як є постійно діючими. Якщо зловмисник отримав доступ до ключа, то єдиним способом убезпечення доступу до даних є відкликання API-ключа. Для забезпечення захисту API-ключів, що передаються, використовуються загальні безпекові механізми, такі як HTTPS/SSL.

Ключі програмного інтерфейсу не повинні зберігатись в одній базі даних із іншими даними, та доступ до них має мати лише програмний інтерфейс. Усі ключі для забезпечення неможливості витоку даних з бази даних повинні зберігатися у зашифрованому вигляді, бажано у вигляді гешу – результату виконання геш-функції що не може бути дешифрованим. Єдиним способом валідації загешованого API-ключа є порівняння результату виконання геш-функції над надісланим у запиті клієнта API-ключем із збереженими у базі даних зашифрованими ключами. У разі невідповідності жодному API-ключу, програмний інтерфейс не повинен надавати доступу до жодних даних.

Розглянемо деякі із найпоширеніших функцій шифрування за допомогою гешування. Для порівняння оберемо функції MD5, SHA-1, SHA-2, SHA-3.

MD5 – один із найстаріших алгоритмів, що досі широко поширений у використанні. Алгоритм приймає на вхід повідомлення будь-якої довжини та повертає як результат роботи строку довжиною в 32 символи шляхом додавання бітів та проведення декількох раундів шифрування. Таким чином, вхідне повідомлення розбивається на блоки фіксованої довжини, кожен з яких проходить через функції стиснення. На даний момент MD5 не вважається безпечної відповідно до стандартів безпеки.

SHA-1 (Secure Hash Algorithm 1) – алгоритм, що приймає на вхід повідомлення будь-якої довжини та конвертує їх в 160-бітний геш. За структурою є схожим до MD5, проте процес отримання блоків повідомлення є більш комплексним. Так само як MD5, SHA-1 більше не вважається безпечним та вартим використання для цифрових підписів.

SHA-2 (Secure Hash Algorithm 2) – алгоритм, що має найбільш поширене використання у сервісах правління Сполучених Штатів та є підтвердженим методом шифрування за NIST. SHA-2 складає сім'ю з 6 різних алгоритмів:

- SHA-224 (обрізана версія SHA-256),
- SHA-256;
- SHA-384 (обрізана версія SHA-512),
- SHA-512,
- SHA-512/224 (обрізана версія SHA-512),
- SHA-512/256 (обрізана версія SHA-512).

Процес гешування є схожим на попередні алгоритми, проте знову має більшу складність – так, виходом алгоритму SHA-256 є 256-бітний геш. Дані алгоритми є стійкими до колізій та широко підтримані більшістю браузерів та операційних систем.

SHA-3 (Secure Hash Algorithm 3) – найновіше сімейство алгоритмів, розроблене за стандартами попередніх поколінь алгоритмів. Дані алгоритми надають більше гнучкості у підходах та цілях шифрування (так, до прикладу, за допомогою SHA-3 можна шифрувати MAC-адреси). Проте, дані алгоритми є набагато більш комплексними та повільнішими в обчисленнях та ще не мають належної підтримки операційних систем, браузерів, платформ. Крім того, за певних умов дані алгоритми можуть зазнавати колізій [19].

Порівняльна характеристика вище описаних алгоритмів гешування наведена в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика алгоритмів гешування

	MD5	SHA-1	SHA-2 (224 & 256/384 & 512)	SHA-3 (224/256/384/512)
Доступні з	1992	1995	2002	2008
Розмір блоку	512 bits	512 bits	512/1024 bits	1152/1088/832/576
Розмір вихідного гешу	128 bits	160 bits	256 bits /512 bits	224/256/384/512 bits
Вразливість до колізіій	Висока	Висока	Низька	Низька
Наявність успішно проведених атак	+	+	-	+
Застарілі	+	+	-	-

Після проведеного аналізу існуючих методів гешування, було вирішено використовувати алгоритм SHA-2, зокрема його модифікацію SHA-256, через помірну складність алгоритму, помірний розмір вихідного гешу, відсутність зафіксованих успішних атак, а також найбільшу підтримку операційних систем, браузерів, платформ серед аналогів.

Таким чином, в даному розділі було розглянуто принцип роботи API-ключів, методи їх захисту та обрано алгоритм шифрування API-ключів, що дозволяє перейти до етапу проектування інформаційної технології на загальному структурному рівні.

2.4 Розробка загальної структурної схеми функціонування інформаційної технології аналізу текстів на наявність образливих висловлювань

У ході попередніх досліджень було створено загальну структурну схему модуля аналізу текстів на наявність образливих висловлювань. Розроблений модуль складається з 5 основних компонентів:

- веб-клієнт;
- програмний інтерфейс додатку(API);
- універсальний кодувальник речень;
- веб-панель адміністрування;
- база даних.

Визначимо призначення кожного із компонентів.

Веб-клієнт представляє собою спрощену платформу для спілкування, де користувачі, як зареєстровані, так і анонімні, зможуть публікувати власні повідомлення. Клієнт отримуватиме дані з сервера та надсилатиме опубліковані дописи на аналіз.

Програмний інтерфейс представляє собою серверну частину, включає обробку даних, ідентифікацію користувачів, валідацію запитів, збереження, редагування та видалення записів у базі даних.

Універсальний кодувальник речень виконує функцію аналізу тексту на наявність образливих висловлювань та формує висновки щодо належності тексту до одного з класів образ, що мають модеруватися, таких як:

- атака на особистість;
- образа;
- нецензурна лексика;
- критична токсичність;
- сексуальна неприйнятність;
- погроза;

– токсичність.

Веб панель адміністрування представляє собою спрощену систему керування дописами користувачів. Ідентифіковані адміністратори отримують інформацію про дописи на основі інтелектуального аналізу та можуть приймати відповідні рішення щодо користувачів: видаляти дописи, блокувати користувачів.

База даних виконує функцію збереження даних про користувачів, адміністраторів та опубліковані дописи, а також інформацію про порушення правил дописами після інтелектуального аналізу.

Основна структурна схема попередньо розробленого модуля показана на рисунку 2.4

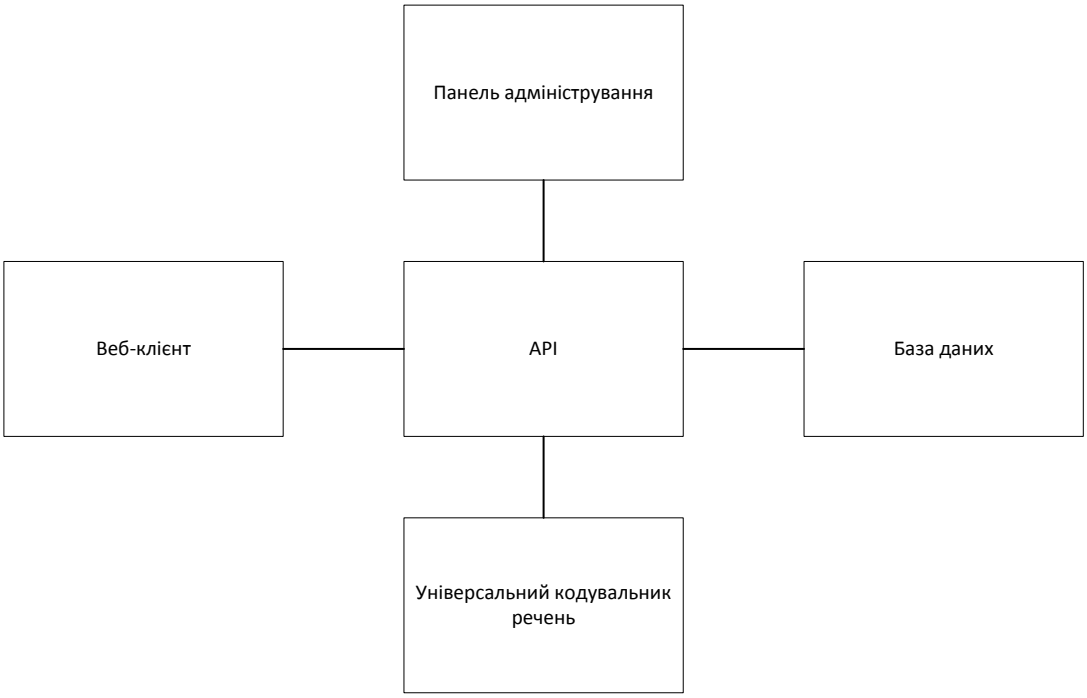


Рисунок 2.4 – Загальна структурна схема інтелектуального модуля аналізу текстів на наявність образливих висловлювань

Основними недоліками розробленого модуля є:

– відсутність механізму аутентифікації декількох користувачів
– відсутність механізму валідації та використання API-ключів;

- відсутність бази даних зберігання ключів;
- неможливість доступу до декількох клієнтських рішень в ізольованому доступі;
- відсутність захисту доступу користувачів без відповідних прав організації.

Для усунення вищезазначених недоліків доповнимо структурну схему наступними компонентами:

Валідатор API-ключів – компонент, що представляє собою механізм валідації ключів через програмний інтерфейс та який має прямий зв’язок із панеллю адміністрування для контролю за поточною сутністю клієнтського рішення, з якою працює користувач.

База даних API-ключів – компонент, задачею якого є зберігання зашифрованих API-ключів, доступ до них здійснюється через програмний інтерфейс. Крім того, програмний інтерфейс доповнимо функцією шифрування API-ключів із запиту користувача для порівняння зі збереженими в базі даних API-ключів.

Описані вище компоненти забезпечують механізм доступу до сутностей клієнтських рішень за допомогою механізму API-ключів, а також забезпечують захист даних шляхом відділення сховища API-ключів від основного, що зменшує шанс витоку даних у випадку атаки на основне сховище даних. Крім того, дані компоненти дозволяють працювати із інформаційною технологією декільком клієнтам одночасно, кожен з яких матиме доступ лише до тих даних, до яких йому надано доступ організацією за допомогою поширеного йому API-ключа. Розроблена структурна схема інформаційної технології аналізу текстів на наявність образливих висловлювань наведена на рисунку 2.5.

Рисунок 2.5 – Загальна структурна схема інформаційної технології аналізу текстів на наявність образливих висловлювань

Розроблена загальна структурна схема формує загальне уявлення про архітектуру інформаційної технології, виділяє її основні компоненти та їх призначення, що дозволяє перейти до розробки алгоритмів роботи інформаційної технології аналізу текстів на наявність образливих висловлювань.

2.5 Розробка основного алгоритму роботи інформаційної технології аналізу текстів на наявність образливих висловлювань

Для відображення циклу роботи програмного забезпечення використаємо текстовий та графічний опис основного алгоритму роботи інформаційної технології аналізу текстів на наявність образливих висловлювань.

Кроки алгоритму:

1. Ідентифікація користувача.
2. Перевірка статусу користувача, якщо користувач є адміністратором, перейти на крок 9, якщо ні, перейти на крок 3.
3. Отримання дописів з сервера.
4. Введення нового допису.

5. Відправка допису на сервер.
6. Аналіз вмісту допису універсальним кодувальником речень.
7. Розрахунок критерію загрози.
8. Збереження даних у базі даних. Перейти на крок 17.
9. Отримання API-ключа.
10. Хешування ключа.
11. Підключення до БД API-ключів.
12. Пошук API-ключа.
13. Пошук, якщо співпадіння знайдене, перейти на крок 16, якщо ні – перейти на крок 14.
14. Відправка помилки.
15. Відправка пустої відповіді.
16. Фільтрація дописів по API-ключу.
17. Формування відповіді сервера.
18. Отримання дописів із інформацією про модерацію.
19. Отримання детальних даних про допис.
20. Візуалізація отриманих даних.
21. Вибір дії, якщо вибрано видалення, перейти на крок 14, якщо ні – перейти на крок 13.
22. Вибір дії, якщо вибрано помітити допис як безпечний, перейти на крок 15, якщо ні, перейти на крок 18.
23. Видалення допису.
24. Відмітка допису як перевіреного.
25. Оновлення даних у базі даних.
26. Оновлення даних на клієнті.
27. Завершення роботи

Графічна інтерпретація описаного алгоритму наведена на рисунку 2.6, графічна інтерпретація підпроцесу валідації API-ключа наведена на рисунку 2.7.

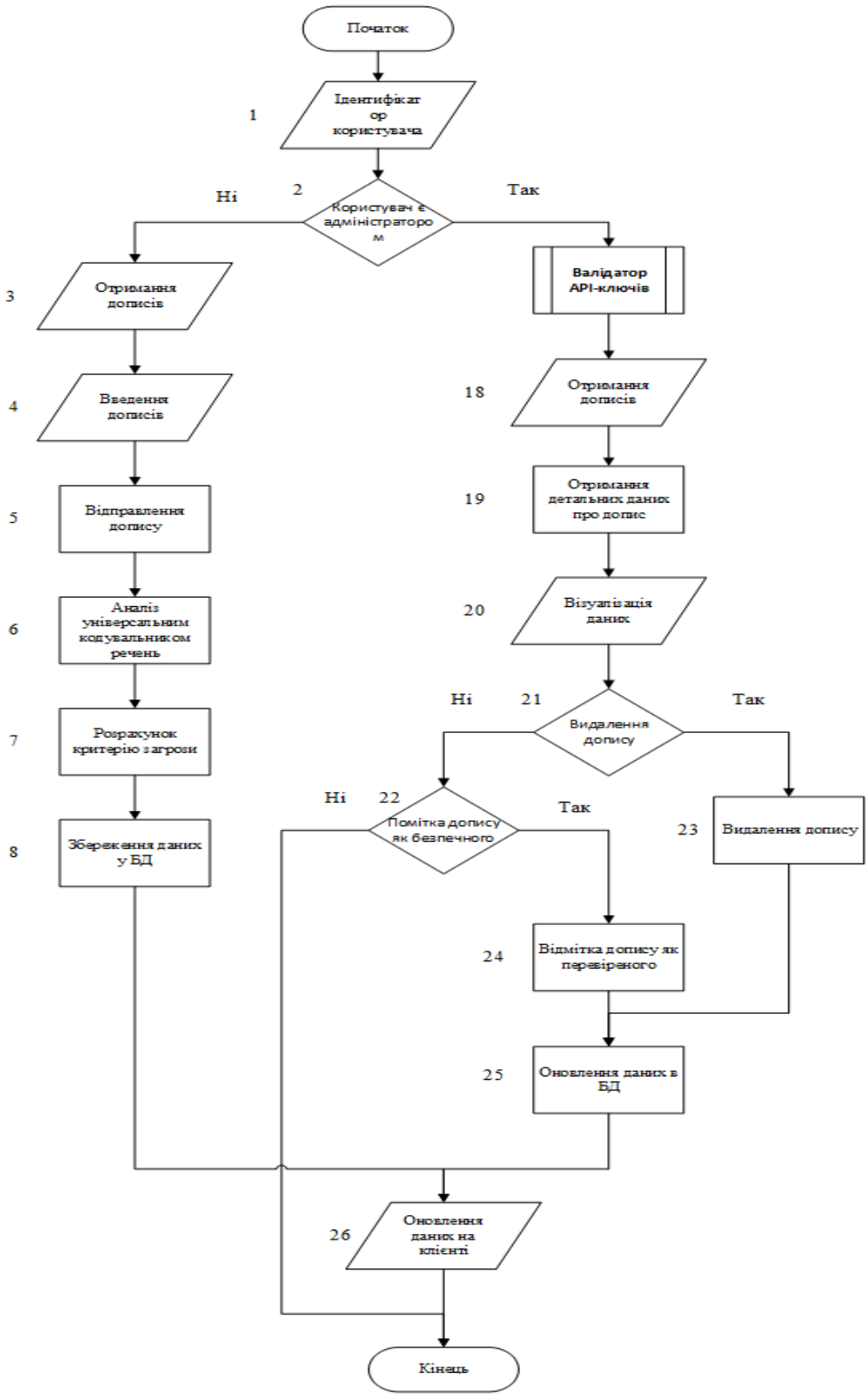


Рисунок 2.6 – Схема основного алгоритму роботи інформаційної технології аналізу текстів на наявність образливих висловлювань

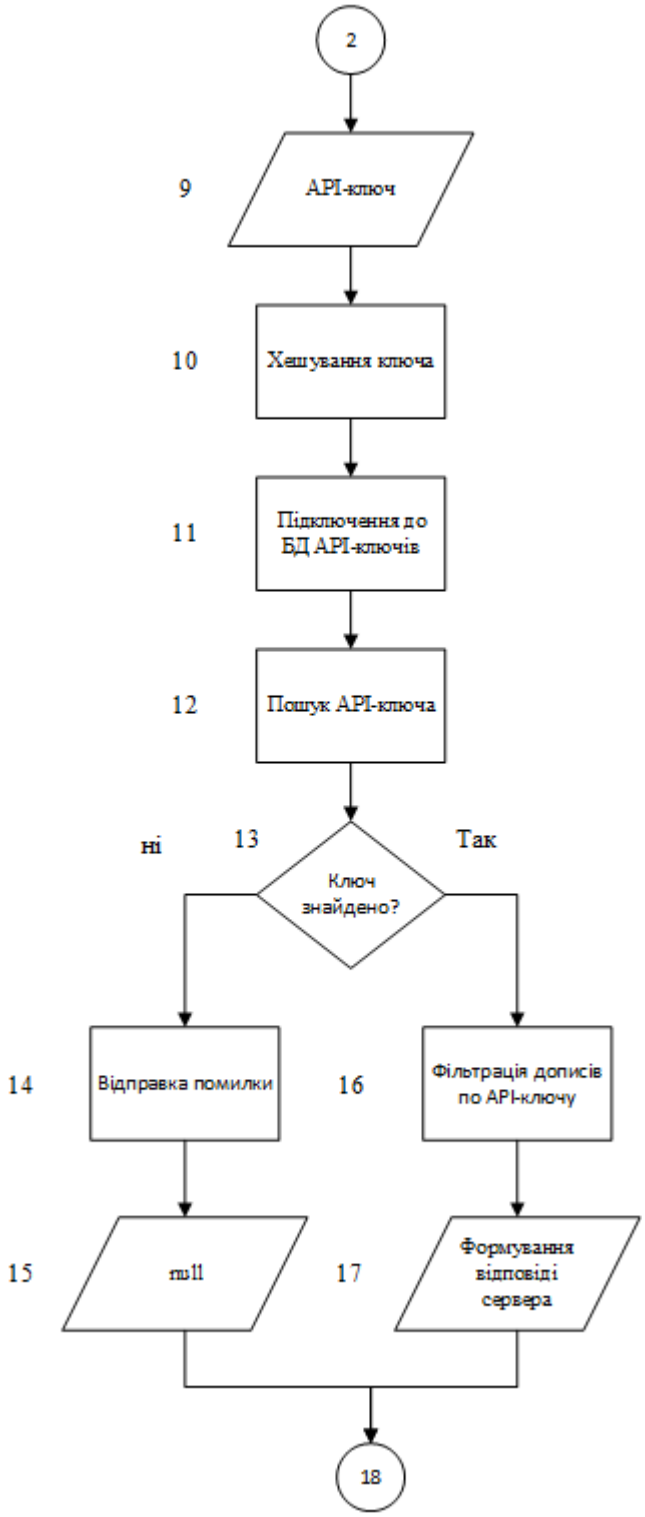


Рисунок 2.7 – Схема алгоритму підпроцесу валідації API-ключів інформаційної технології аналізу текстів на наявність образливих висловлювань

Розроблені в даному розділі алгоритми роботи інформаційної технології, а також розроблена у попередньому розділі загальна структурна схема інформаційної технології надають достатньо інформації про компоненти технології та принцип їх роботи, щоб перейти до етапу моделювання інформаційної технології із використанням мови UML.

2.6 Моделювання інформаційної технології аналізу текстів на наявність образливих висловлювань із використанням мови UML

Модель – це абстракція, яка створюється з метою осмислення чого завгодно перед тим, як його створювати.

Абстрагування – це вибіркове вивчення деяких аспектів проблеми. Основна мета абстрагування полягає в тому, щоб ізолювати аспекти, важливі для деякої цілі, і відкинути всі інші [20].

UML (англ. Unified Modeling Language) — уніфікована мова моделювання. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю.

Для опису основного циклу роботи програми можна використати різні діаграми UML, зокрема, діаграму активності та діаграму послідовностей.

Діаграма активності є аналогом звичної схеми алгоритму програми, тобто відображає, як поток управління переходить з одної діяльності в іншу.

Діаграма послідовностей також показує життєвий цикл певної сутності, але при цьому зображає їх на єдиній часовій осі, взаємодію деяких сутностей(акторів) та відносну тривалість процесів. Тому використання діаграми послідовностей дає більш повне представлення про цикл роботи програмного забезпечення.

До об’єктів діаграми послідовності можемо віднести зазначені у попередньому розділі компоненти інформаційної технології, а також додамо два об’єкти – користувача та адміністратора для відображення їх взаємодії. Таким чином, отримаємо 9 об’єктів: веб-клієнт, програмний інтерфейс, валідатор API-ключів, панель адміністратора, універсальний кодувальник речень, база даних, база даних API-ключів, користувач, адміністратор. Розроблена діаграма послідовностей зображена на рисунку 3.2.

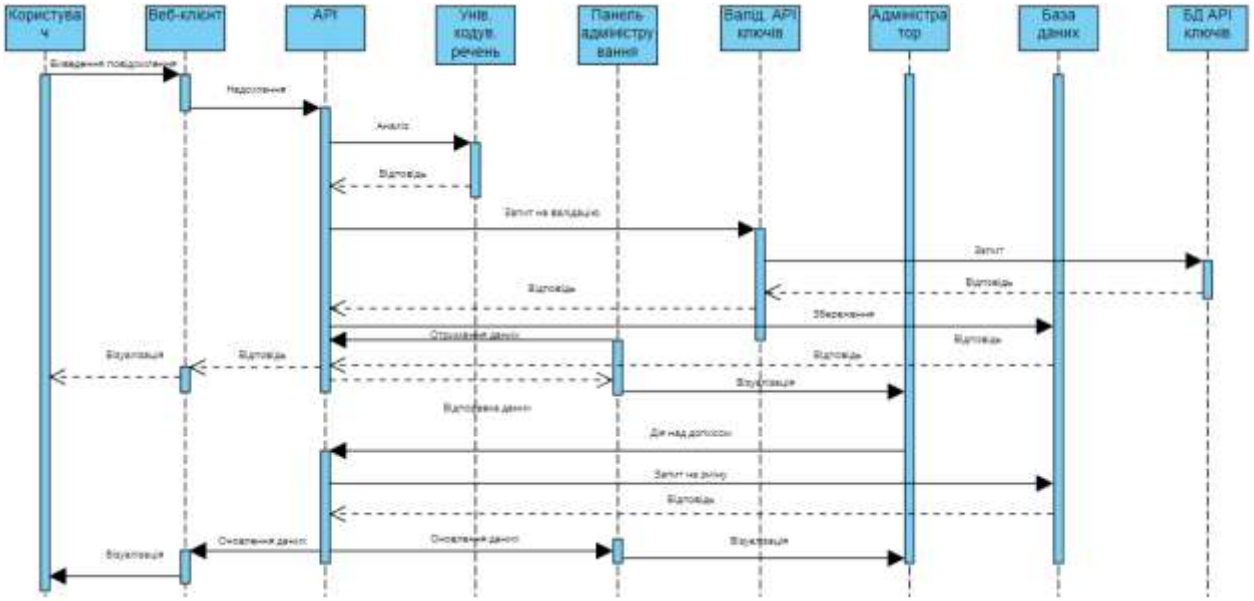


Рисунок 2.8 – Діаграма послідовності взаємодії об’єктів в інформаційній технології аналізу текстів на наявність образливих висловлювань

Для проектування фізичної та логічної організації компонентів інформаційної технології аналізу текстів використаємо діаграму розгортування. Діаграма розгортування – вид UML-діаграми, яка демонструє архітектуру виконання системи, включає в себе такі вузли, як апаратні або програмні середовища виконання, а також проміжне програмне забезпечення, що їх сполучує. Діаграма розгортування для інтелектуального модуля аналізу текстів включає в себе наступні вузли:

– веб-клієнт;
– сервер;
– хостинг бази даних;
– база даних;
– база даних API-ключів;
– СУБД;
– програмний інтерфейс;
– валідатор API-ключів;
– універсальний кодувальник речень.

Усі компоненти пов’язані між собою мережею, тобто зв’язком на основі протоколів TCP/IP. Розроблена діаграма розгортування представлена на рисунку 2.9.

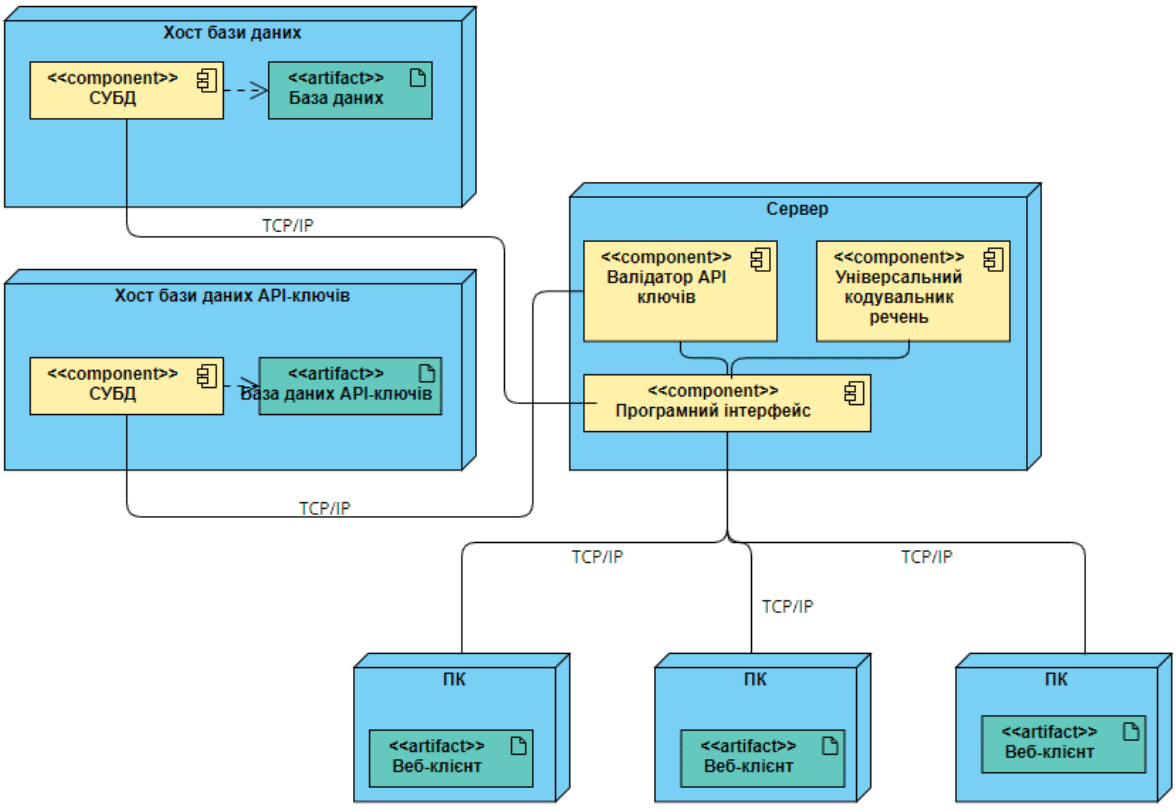


Рисунок 2.9 – Діаграма розгортування інформаційної технології аналізу текстів на наявність образливих висловлювань

2.7 Висновок до розділу 2

Таким чином, в даному розділі було розглянуто відомі методи аналізу текстів, а саме методи аналізу зв’язного та незв’язного тексту, було визначено переваги і недоліки підходів; було обрано універсальний кодувальник речень як методику, що забезпечує аналіз зв’язного тексту; запропоновано коефіцієнт загрози, що дозволяє отримати загальний рівень загрози коментаря в залежності від важливості критеріїв оцінки. Було розглянута область застосування API-ключів, вказані недоліки підходу з точки зору безпеки, та запропоновано методи захисту інформації, зокрема розглянуто різні алгоритми гешування інформації та обрано SHA-2(SHA-256) для розробки валідатора API-ключів інформаційної технології. Було виділено основні компоненти інформаційної технології аналізу текстів на вміст образливих висловлювань, розроблено загальну структурну схему інформаційної технології, розроблено основний алгоритм роботи інформаційної технології, а також підпроцесу валідації API-ключів, розроблено UML-діаграми розгортування та послідовності для відображення послідовності процесів в ході використання програмного забезпечення, а також реальні архітектурні складові інформаційної технології. Обрані алгоритм аналізу текстів, алгоритм шифрування API-ключів, розроблені схеми і діаграми детально описують структуру та принцип роботи програмного забезпечення, що дозволяє перейти до програмної реалізації інформаційної технології аналізу текстів на наявність образливих висловлювань.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АНАЛІЗУ ТЕКСТІВ НА НАЯВНІСТЬ ОБРАЗЛИВИХ ВИСЛОВЛЮВАНЬ

3.1 Обґрунтування вибору мови програмування та системи управління базами даних, бібліотек та фреймворків для програмної реалізації інформаційної технології аналізу текстів на наявність образливих висловлювань

Визначимо інструменти, за допомогою яких буде створюватись програмна реалізація інформаційної технології аналізу текстів на наявність образливих висловлювань. Зокрема, необхідно визначити мову програмування для створення веб-клієнта, панелі адміністрування, програмного інтерфейсу, універсального кодувальника речень, а також системи керування базами даних(СКБД) для збереження даних.

Для аналізу і порівняння оберемо 3 мови програмування: JavaScript, C# Python.

JavaScript – мова програмування, що розвивається найбільш динамічно. Спочатку була орієнтована лише на веб-розробку, а саме на її клієнтську частину. Проте можливості мови розширюються з прийняттям нових стандартів EcmaScript, в тому числі і нові можливості об’єктно-орієнтованого програмування, що були недоступні раніше, наприклад, модифікатори доступу полів класів, можливість створення інтерфейсів, використання принципів поліморфізму та перевантаження методів. Так, на даний момент за допомогою JavaScript без труднощів розробляється і серверна частина веб-сайтів і додатків, а створення платформи Node.js дозволило створювати звичні програмні додатки [21].

C# - це мова програмування, що відноситься до мов із C-подібним синтаксисом. Ця мова ввібрала в себе елементи C++ і Java та була створена для

розробки додатків на платформі Microsoft .NET Framework. Використовується для створення клієнтських та веб-додатків. Крім того, в Visual Studio вбудований garbage-collector, який сам контролює пам’ять. C# використовується через свою універсальність при виконанні на платформі Windows [22].

Python – інша мова програмування, що швидко розвивається. Мова має простий і зрозумілий синтаксис. Це значно спрощує процес розробки. Має динамічну типізацію та автоматичний контроль пам’яті та механізм обробки виключень. Має велику кількість готових пакетів для машинного навчання, створення нейронних мереж, аналізу даних, тому часто використовується у даних напрямках [23].

У таблиці 3.1 наведена порівняльна характеристика мов програмування.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Критерій/мова	C#	JavaScript	Python
Ручне управління пам’яттю	+	-	-
Необхідність компіляції	+	-	-
Необхідність у роботі з браузером	-	+	-
Виконання на клієнтській частині браузера	-	+	-
Пряме підключення скриптів	-	+	-
Використання на сервері	+	+	+
Наявність пакетів для машинного навчання	-	+	+
Підтримка на всіх платформах	-	+	-

Виходячи з необхідності розробки веб-додатку, для створення програмного забезпечення було обрано мову програмування JavaScript, тому що вона є незамінною при роботі на клієнтській частині браузера. Крім того,

JavaScript може використовуватись також на сервері, що зменшує ресурси і час розробки та спрощує подальшу підтримку.

Розглянемо три системи управління базами даних: MongoDB, MySQL та Redis.

MongoDB – документоорієнтована система управління базами даних. Особливістю даної СУБД є те, що дані зберігаються у вигляді документів формату JSON (JavaScript Object Notation), що спрощує процес обробки даних, отриманих в результаті виконання запитів до бази даних. Це означає, при роботі з MongoDB немає необхідності опису схем таблиць, тобто MongoDB є представником NoSQL-систем. Головною перевагою при роботі з даною СУБД є її гнучка структура, яку можна легко модифікувати та розширювати в процесі експлуатації бази даних без необхідності модифікації існуючих даних. Крім того, MongoDB підтримує JavaScript як мову написання запитів та функцій агрегації, також можливе використання регулярних виразів. Основним недоліком у деяких ситуаціях роботи з MongoDB є відсутність строгої схеми даних у базі даних, що може призвести до неочікуваних наслідків при роботі з отриманими даними [24].

MySQL – реляційна система управління базами даних. Відповідно, MySQL реалізовує реляційну модель бази даних. Реляційною називається БД, у якій всі дані, доступні користувачу, організовані у вигляді таблиць, а всі операції над даними зводяться до операції над цими таблицями. До переваг реляційної бази можна віднести: простоту і доступність для розуміння користувачем; єдиною використовуваною інформаційною конструкцією є «таблиця»; строгі правила проектування, які базуються на математичному апараті; повну незалежність даних. Недоліком реляційної моделі бази даних є велике використання зовнішньої пам’яті, а також можливість виникнення аномалій, якщо не проводиться нормалізація. Іншим недоліком можна вважати необхідність приведення даних до необхідного формату, у випадку задачі розробки на мові JavaScript – у формат JSON, що потребує використання додаткових ORM(Object-Relational Mapping) для перетворення даних [25].

Redis – резидентна система управління базами даних, відноситься до NoSQL-систем. Резидентною базою даних називають таку базу даних, що розміщається в оперативній пам’яті. Дані зберігаються у вигляді «ключ – значення». Основною перевагою резидентних баз даних є швидкість відповіді на запити, що може грати ключову роль у таких операціях, як торги на біржі у реальному часі, в управлінні телекомунікаційними приладами. Redis може використовуватись не тільки як база даних, але й як механізм кешування. Redis не є традиційною СУБД та використовується для специфічних задач, наведених вище [26].

Після аналізу та порівняння вищенаведених систем управління базами даних, для розробки інтелектуального модуля було обрано СУБД MongoDB, яка забезпечує простоту розробки, підтримки та використання при роботі з мовою JavaScript, так як не потребує додаткових інструментів для приведення даних у необхідний формат, легко масштабується та має гнучку схему даних, що дозволяє спростити подальшу підтримку розробленого модуля.

Для реалізації клієнтської частини інтелектуального модуля оберемо SPA-модель веб-додатку. SPA(Single Page Application) – вид веб-додатків, у яких при переході між сторінками не відбувається перезавантаження додатку, що забезпечує кращий досвід користування. Для вибору інструменту реалізації SPA розглянемо бібліотеку React та фреймворки Vue.js та Angular.

React – бібліотека, що розроблена компанією Facebook. React використовує мову шаблонів JSX, що спрощує розробку об’єктної моделі документа (DOM). React має механізм Virtual DOM, що вирішує, чи необхідно виконувати рендер наступного екрана до реального відмалювання, так як зміна реального DOM-дерева є однією із найбільш ресурсомістких задач при роботі веб-додатку [27]. React забезпечує одностороннє прив’язування даних, що в свою чергу забезпечує модулярність та швидкість. Односпрямований потік даних означає, що коли розробник розробляє додаток React, він часто вкладає дочірні компоненти в батьківські компоненти. Таким чином, розробник знає, де і коли виникає

помилка, надаючи їм кращий контроль над усією веб-програмою. React не має чіткої методології розробки, так як не є фреймворком, що дозволяє підлаштовувати процес розробки під індивідуальні потреби. Недоліком React є відсутність механізму навігації та централізованого сховища, через що необхідно встановлювати додаткові бібліотеки для створення повноцінного односторінкового додатку.

Vue.js є прогресивним фреймворком для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро у першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків, якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками [28]. До переваг Vue можна віднести невеликий розмір пакета, використання віртуального DOM-дерева, реактивну двосторонню прив’язку даних. Двостороння прив'язка даних - це зв'язок між оновленням даних моделі та поданням (UI). Зв’язані компоненти містять дані, які можна час від часу оновлювати. За допомогою двостороннього прив’язки даних простіше оновлювати пов’язані компоненти та відстежувати оновлення даних. У Vue пов'язані дані оновлюються реактивно, як і об'єкти DOM, що чудово підходить для будь-якої програми, яка вимагає оновлення в режимі реального часу. До недоліків Vue можна віднести відсутність підтримки масштабних проектів, дещо більш повільну підтримку за рахунок меншої команди розробників, відсутність великої кількості матеріалів через новизну фреймворка.

Angular – фреймворк, що розробляється компанією Google, використовується для клієнтських SPA-додатків. Angular використовує вищенаведений механізм двостороннього зв’язування, що дозволяє динамічно змінювати дані в інтерфейсі. Ключовою відміннстю Angular є використання TypeScript, що дозволяє спростити процес розробки та виправляти помилки на

етапі компіляції, а не під час реальної експлуатації. Проте, TypeScript можна віднести і до недоліків, так як його вивчення займає багато часу. Angular є найбільш комплексним із наведених фреймворків та надає інструменти для повного циклу розробки веб-додатку, як управління сховищем даних, навігацію тощо [29]. Комплексність фреймворку також може бути його недоліком, так як для повного засвоєння принципів роботи Angular необхідно витратити значно більше часу, ніж із іншими фреймворками.

Розглянувши переваги і недоліки наведених інструментів, для програмної реалізації клієнтської частини інтелектуального модуля аналізу текстів на наявність образливих висловлювань оберемо бібліотеку React, так як вона має нечітку архітектуру та методологію розробки, забезпечує оптимальну роботу з DOM-деревом та має низький рівень входу та простий процес подальшої підтримки розробленого додатку.

Розглянемо інструменти для спрощення написання серверної частини інтелектуального модуля: бібліотеки Express.js, Koa.js, GraphQL.

Express.js – бібліотека для Node.js, що забезпечує створення класичних REST-програмних інтерфейсів. REST (Representation state transfer) – вид програмних інтерфейсів, що реалізовує взаємодію клієнт-серверну модель взаємодії. Така модель використовує основні чотири види запитів: на отримання даних(GET), відправку даних(POST), редагування даних(PUT), та видалення даних(DELETE). Наведені моделі запитів покривають більшу частину задач, що потрібно вирішити при обміні інформацією між клієнтом і сервером, спрощують процес обробки запитів [30]. Express є найпопулярнішою бібліотекою для створення програмних інтерфейсів на JavaScript, має обширну документацію та повний пакет необхідного функціоналу із збереженням малого розміру пакета.

Koa.js – мінімалістична бібліотека для створення REST-інтерфейсів. Особливістю Koa є найменший розмір пакета та модульність, що дозволяє встановлювати та використовувати лише необхідні пакети. Koa.js розробляється творцями Express, тому сповідує ті ж принципи при розробці програмних

інтерфейсів. Koa дозволяє значно розширити та спростити процес відловлення та опрацювання помилок при роботі сервера. Основним недоліком Koa є його новизна, відсутність обширної документації та великої спільноти розробників, що ускладнює процес розробки.

GraphQL – бібліотека, що забезпечує параметричну взаємодію між клієнтом та сервером. Особливістю GraphQL є те, що бібліотека використовує лише одну точку входження(endpoint) для взаємодії, на якому реалізовується схема додатку – загальна модель усієї структури даних, необхідної для роботи клієнта. GraphQL підтримує потужні механізми кешування даних та віддає на клієнт лише ту частину даних, що змінилося, значно знижуючи використання трафіку та використання обчислювальних ресурсів для обробки даних на клієнті [31]. Недоліком GraphQL можна визначити нестандартну модель взаємодії клієнта та сервера, що вимагає додаткового вивчення та досвіду розробки. Крім того, механізм взаємодії між клієнтом та сервером не є простим та вимагає встановлення додаткових пакеті на клієнті, таких як Apollo.

Проаналізувавши переваги і недоліки інструментів реалізації серверної частини, для створення інтелектуального модуля аналізу текстів було обрано бібліотеку Express.js через простоту архітектури клієнт-серверної взаємодії, велику документну базу та велику спільноту розробників, що пришвидшує та спрощує процес розробки серверної частини.

3.2 Програмна реалізація компонентів інформаційної технології аналізу текстів на наявність образливих висловлювань

Розглянемо основні моменти програмної реалізації інформаційної технології аналізу текстів на наявність образливих висловлювань.

Головним компонентом інформаційної технології є серверна частина, яка забезпечує взаємодію між клієнтом і базою даних, а також виконує інтелектуальний аналіз тексту дописів користувачів.

При ініціалізації програмного інтерфейсу відбувається підключення до бази даних та ініціалізація слухачів запитів (routes). Після цього усі запити, що потребують авторизації користувача, проходитимуть перевірку на актуальність JWT-токена, що ідентифікує сесію та користувача. Фрагмент коду, що реалізує перевірку JWT-токена:

```
const jwt = require('jsonwebtoken');
const config = require('config');

module.exports = function(req, res, next) {
  const token = req.header('x-auth-token');

  if (!token) {
    return res.status(401).json({ msg: 'No token, authorization denied' }
);
  }

  try {
    const decoded = jwt.verify(token, config.get('jwtSecret'));

    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({ msg: 'Token is not valid' });
  }
};
```

Для самої авторизації використовуватимемо відповідний запит «/auth», що потребуватиме логіна і пароля користувача. Пароль користувача зберігається у хешованому вигляді та дешифрується на сервері. У разі відповідності даних користувачу повертається JWT-токен сесії, необхідний для подальшої роботи із інтелектуальним модулем.

Фрагмент коду, що реалізує авторизацію користувачів:

```
const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    const { email, password } = req.body;
    try {
      let user = await User.findOne({ email });
      if (!user) {
        return res
          .status(400)
          .json({ errors: [{ msg: 'Invalid credentials' }] });
      }
      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return res
          .status(400)
          .json({ errors: [{ msg: 'Invalid credentials' }] });
      }
      const payload = {
        user: {
          id: user.id
        }
      };
      jwt.sign(
        payload,
        config.get('jwtSecret'),
        { expiresIn: 360000 },
        (err, token) => {
          if (err) throw err;
          res.json({ token });
        }
```

```
      );
    } catch (err) {
      console.error(err.message);
      res.status(500).send('Server error');
    }
```

При публікації допису на сервері виконується автоматична перевірка вмісту на образливість за допомогою універсального кодувальника речень. Модель була натренована на наборі дописів користувачів, опублікованого центром Civil Research Data [25]. Крім того, обчислюється загальний коефіцієнт загрози. Уся отримана інформація разом із інформацією про допис записується у базу даних, після чого іде оновлення актуальних даних на клієнті.

Фрагмент коду, що реалізує інтелектуальний аналіз тексту:

```
const model = await toxicity.load(threshold);
    const analysisResult = await model.classify([ content ]);
        const severityCoefficient = analysisResult.reduce((acc, criteria)
 => {
            const criteriaProbability = criteria.results[0].match ? 1 : c
riteria.results[0].probabilities['1'];

            return acc + criteriaSeverities[criteria.label] * criteriaPro
bability;}, 0);

const post = new Post({
     userId: id,
     content,
     moderation: {
         severityCoefficient: severityCoefficient % 1,
         analysisResult,
         moderated: false
} });
```

Розглянемо ключові моменти розробки клієнтської частини інтелектуального модуля аналізу текстів на наявність образливих висловлювань. Для розробки користувацького інтерфейсу оберемо бібліотеку Material UI, що надає доступ до великої кількості компонентів інтерфейсу для React, що спрощує процес розробки.

Для авторизації користувачів є необхідність перевірки відповідності введених даних вимогам, зокрема, формату електронної пошти та мінімальної кількості символів у паролі. Фрагмент коду, що виконує перевірку валідності полів авторизації:

```
const [ email, setEmail ] = useState('');
    const [ emailError, setEmailError ] = useState();
    const [ password, setPassword] = useState('');
    const [ passwordError, setPasswordError ] = useState();
    const [ emailManuallyChanged, setEmailManuallyChanged ] = useState(fa
lse);
    const [ passwordManuallyChanged, setPasswordManuallyChanged ] = useSt
ate(false);
useEffect(() => {
        if(passwordManuallyChanged && password.length < 6){
            !passwordError && setPasswordError('minimum length is 6')
        } else {
            passwordError && setPasswordError(false)
        }

        if(emailManuallyChanged && !email.match(re)){
            !emailError && setEmailError('invalid email')
        } else {
            emailError && setEmailError(false)
        }
    }, [email, password, passwordManuallyChanged,emailManuallyChanged, em
ailError, passwordError]);
```

Приклад роботи перевірки полів наведено на рисунку 3.1.

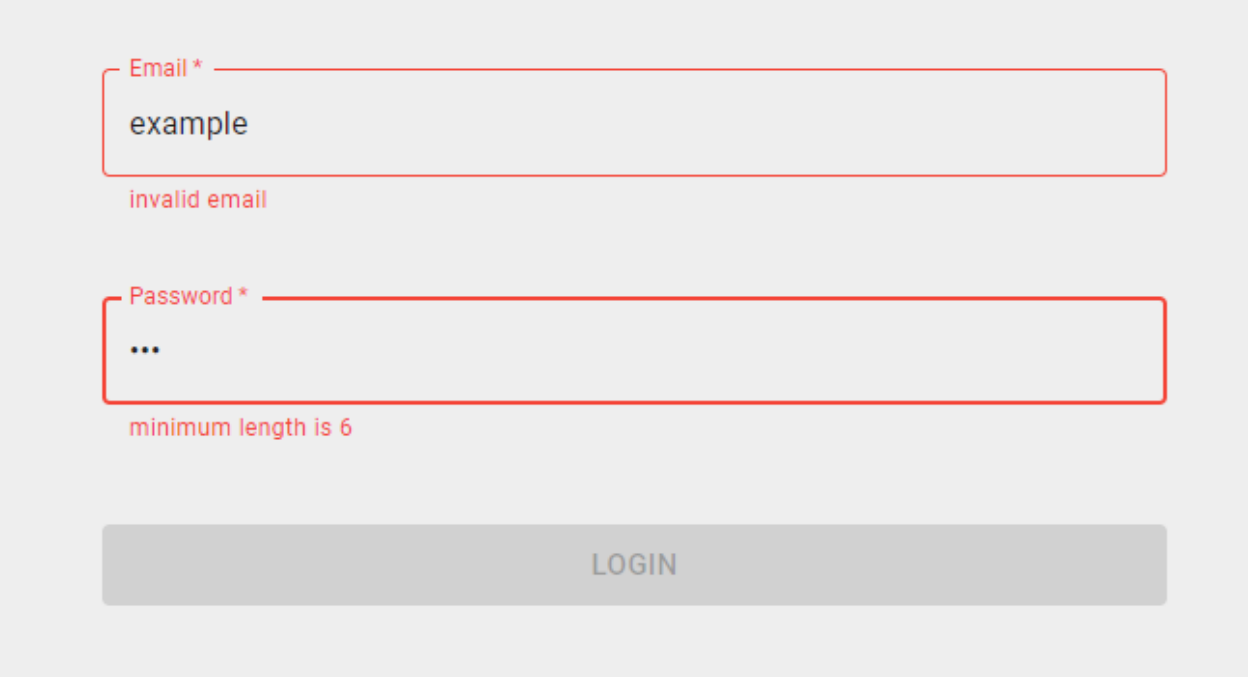


Рисунок 3.1 – Загальний вигляд інтерфейсного вікна авторизації на відповідність встановленим правилам

Для візуалізації отриманих даних належності допису до певного класу образ можна використати діаграми, що спрощують сприйняття чисельної інформації. Для візуалізації даних по критеріях загрози коментаря інтелектуального модуля аналізу текстів на наявність образливих висловлювань використаємо діаграму виду «радар» – тип діаграм, в якому променями є відповідні характеристики об’єкту, полігони – відповідають за візуалізацію степені критерія. Приклад діаграми виду «радар» наведено на рисунку 3.2.

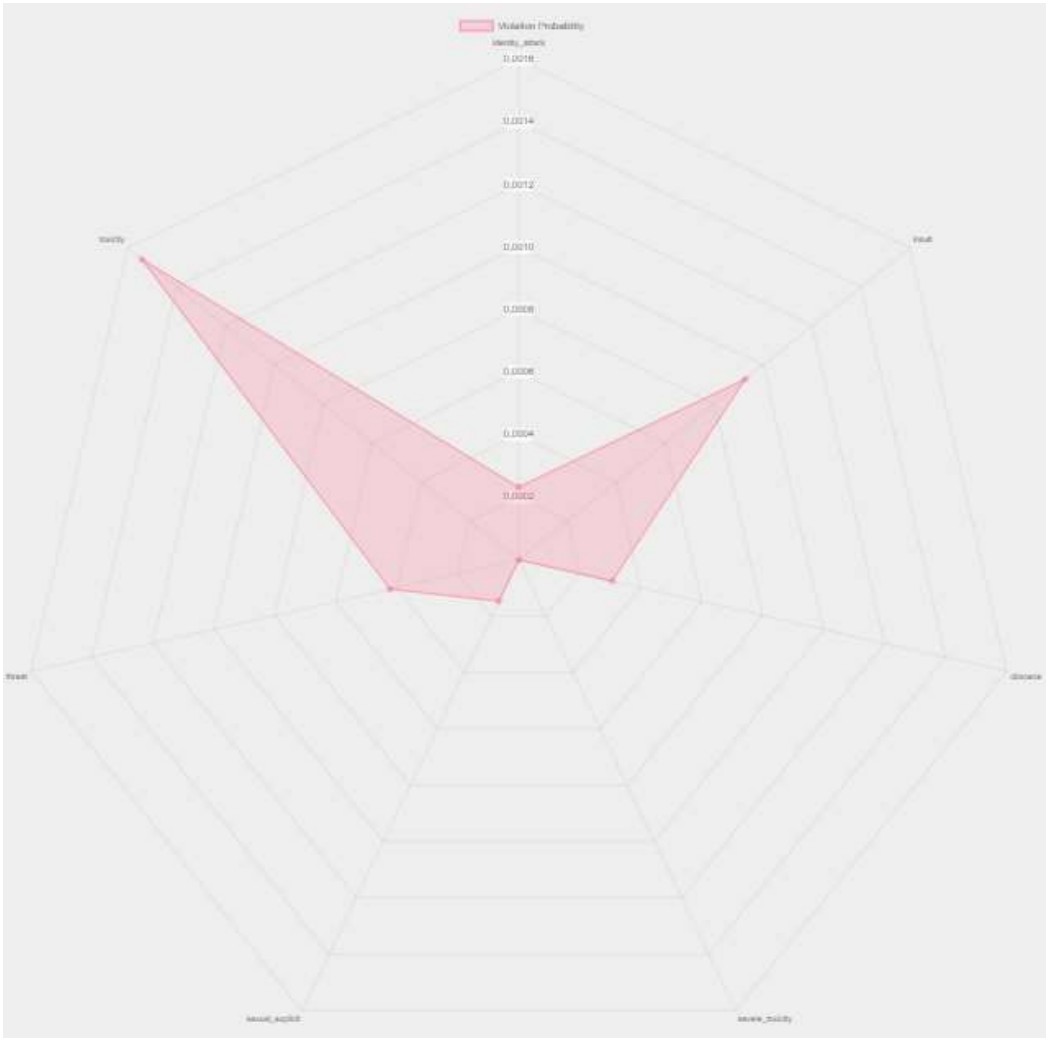


Рисунок 3.2 – Загальний вигляд інтерфейсного вікна діаграми «Радар»

Для забезпечення більш зручного пошуку дописів за різними критеріями, таким як дата публікації або рівень загрози коментаря, необхідно виконувати сортування даних за відповідним показником. Фрагмент коду, що виконує сортування за показником:

```
if(sortingRule !== fieldName){
    setSortingRule('date');
    setSortedPosts(posts.sort((a, b) => {
        return Date.parse(a.date) < Date.parse(b.date);
    }));
    changeOrder && setOrder('asc');
```

```
} else {
   setSortingRule('date');
   changeOrder && setOrder(order === 'asc' ? 'desc' : 'asc');
   setSortedPosts(sortedPosts.reverse())
}
```

Обмін даними з сервером виконується за допомогою асинхронних запитів, реалізованих за допомогою бібліотеки redux-api-middleware, що дозволяє спростити процес відправки, обробки та збереження даних, отриманих з сервера. Усі дані на клієнті зберігаються у централізованому сховищі. При виході зі свого акаунту усі дані про сесію та дописи видаляються.

3.3 Тестування та аналіз результатів роботи програми аналізу текстів на наявність образливих висловлювань

Визначимо критерії, за якими будемо проводити тестування розробленої програми. Так як метою дослідження є розширення функціональних можливостей в області аналізу текстів на наявність образливих висловлювань, визначимо критерії відповідності програми поставленій меті згідно поставленим задачам дослідження:

- веб-клієнт інтелектуального модуля аналізу текстів доступний за відповідним посиланням;
- на відкриття сторінки авторизації відображаються відповідні поля авторизації, відбувається перевірка введених даних та виконується процес авторизації;
- дані про дописи доступні на основній сторінці веб-клієнта;
- дані про дописи для проведення процесу модерації наведені на адміністративній панелі;
- можливе сортування за різними показниками;
- при виборі допису відображається детальна інформація;

– на сторінці детальної інформації є можливість видалити допис або позначити його як безпечний;
– при завершенні роботи видаляються дані про сесію та деталі дописів.

Проведемо тестування відповідно до визначених критеріїв.

Відкриємо веб-клієнт інформаційної технології аналізу текстів. Отримаємо інформацію про опубліковані дописи (рис. 3.4).

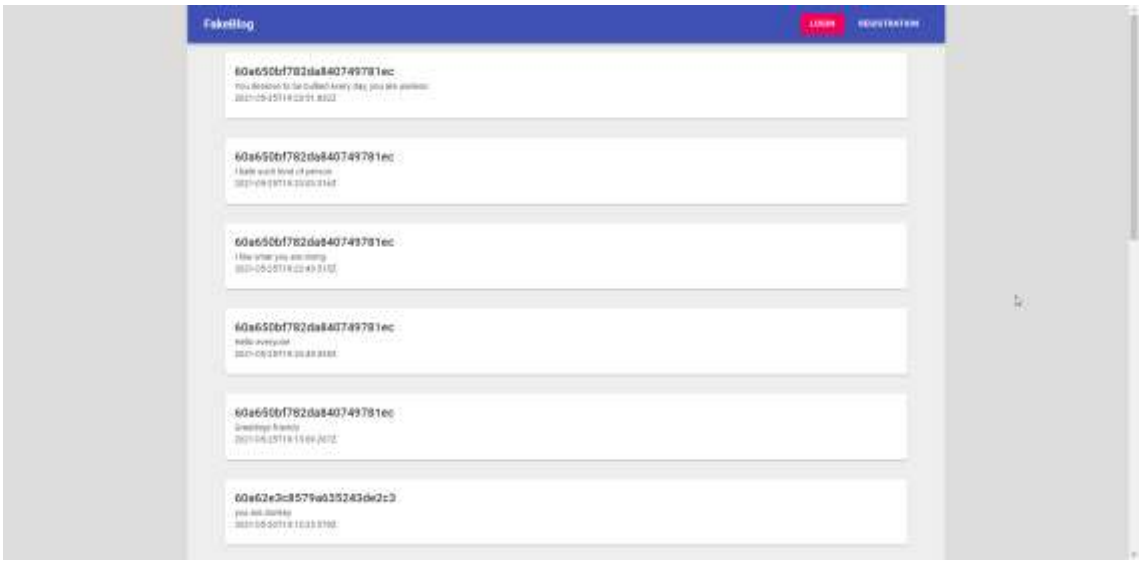


Рисунок 3.4 – Загальний вигляд інтерфейсного вікна інформації про опубліковані дописи

Відкриємо сторінку авторизації. Заповнимо поля невідповідними даними, отримуємо повідомлення про помилку, завершити процес авторизації не є можливим (рис. 3.5). Введемо вірні дані та завершимо процес авторизації. Перевіримо запис даних про сесію користувача (рис. 3.6). Відкриємо панель адміністратора, отримуємо таблицю даних про опубліковані дописи (рис. 3.7).

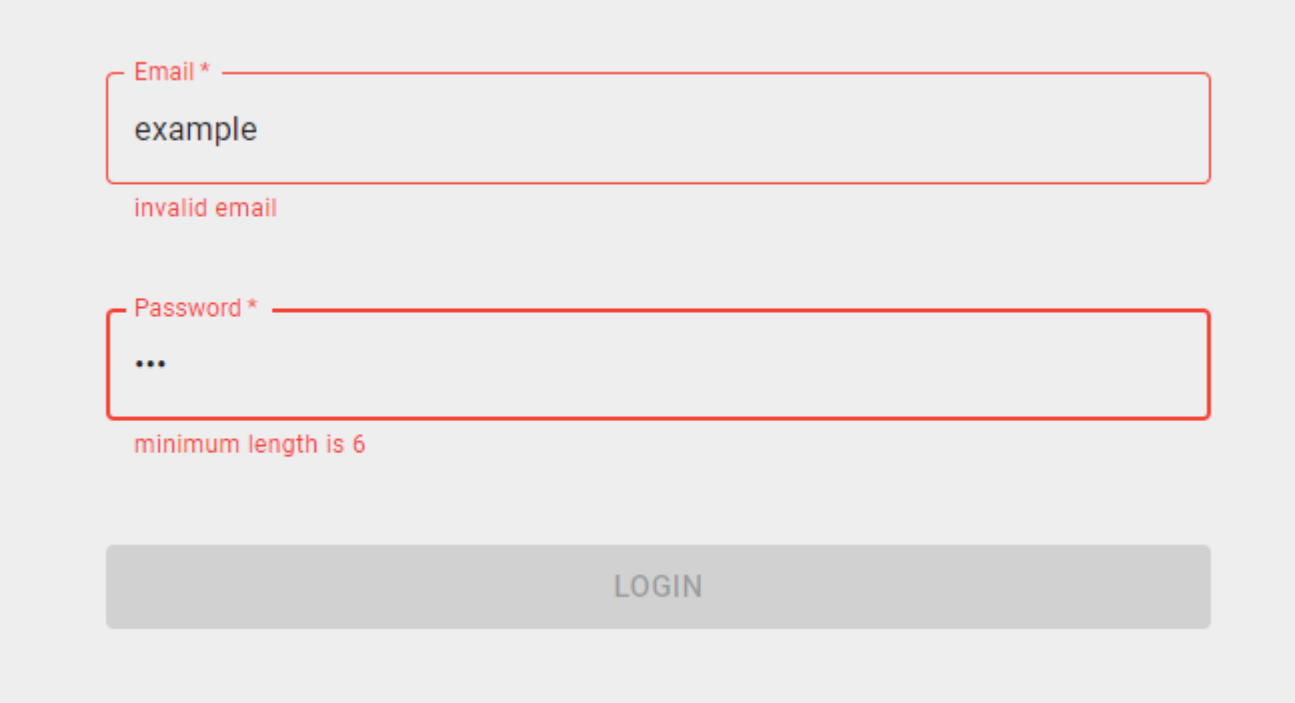


Рисунок 3.5 – Загальний вигляд інтерфейсного вікна авторизації з виведенням помилок про невірні дані

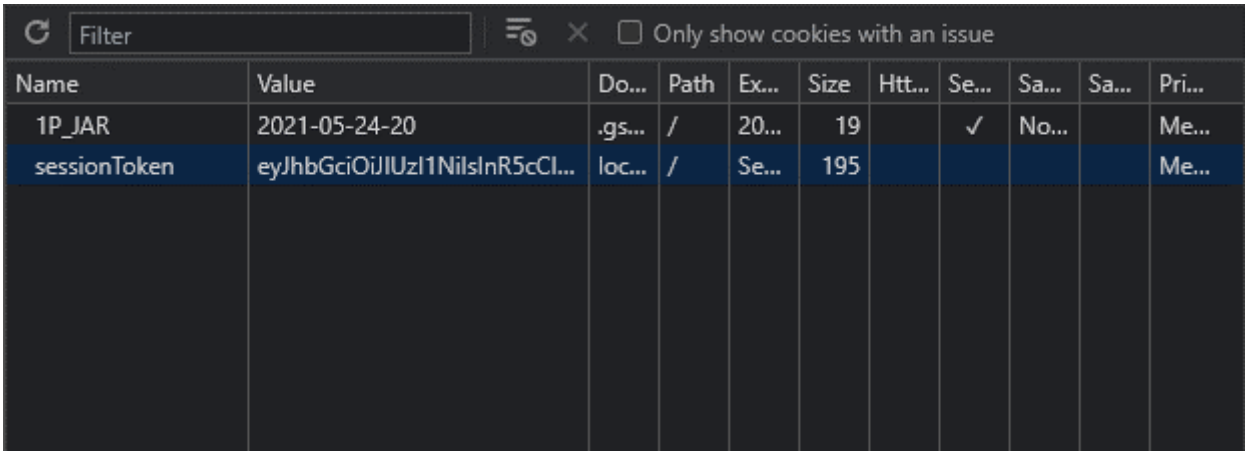


Рисунок 3.6 – Загальний вигляд вікна даних про сесію користувача

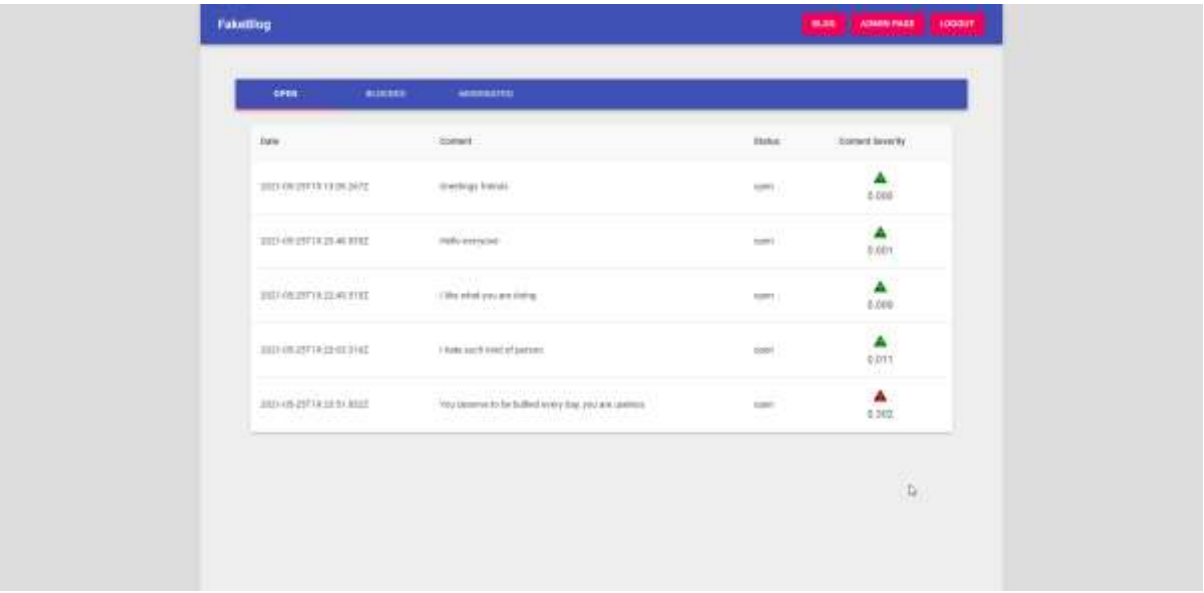


Рисунок 3.7 – Загальний вигляд інтерфейсного вікна панелі адміністратора

Відсортуємо дані за загрозою коментаря, для цього натиснемо відповідну кнопку на таблиці із дописами. Результати сортування відображено на рисунку 3.8.

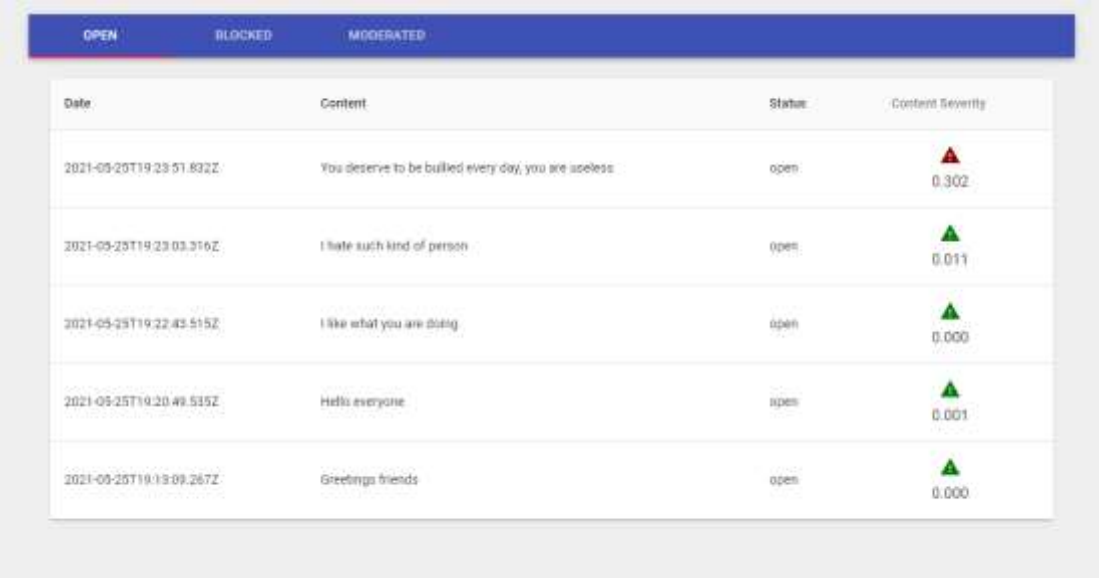


Рисунок 3.8 – Загальний вигляд інтрефейсного вікна з даними про дописи після сортування за критерієм загрози

Натиснемо на допис, чим відкриємо сторінку із детальною інформацією про нього. На відкритій сторінці можна отримати візуалізацію даних у вигляді діаграми, дані про допис і користувача, хто є автором, а також кнопки керування дописом (рис. 3.9).

Видалимо допис, натиснувши відповідну кнопку керування. Допис видаляється з не модерованих, змінює статус на блокований та переміщається у вкладку заблокованих дописів (рис. 3.10).

Повернемося на сторінку дописів, можна помітити, що допис більше не доступний для перегляду на даній сторінці. Перейдемо на вкладку заблокованих дописів. Знову відкриємо відмодерований допис. Натиснемо кнопку помітки допису як безпечного, чим перемістимо його на вкладку відмодерованих дописів (рис. 3.11).

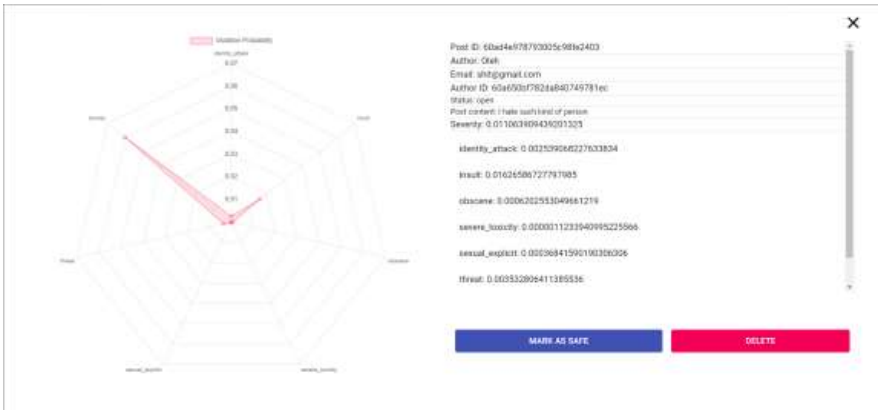


Рисунок 3.9 – Загальний вигляд інтерфейсного вікна детальної інформації про допис

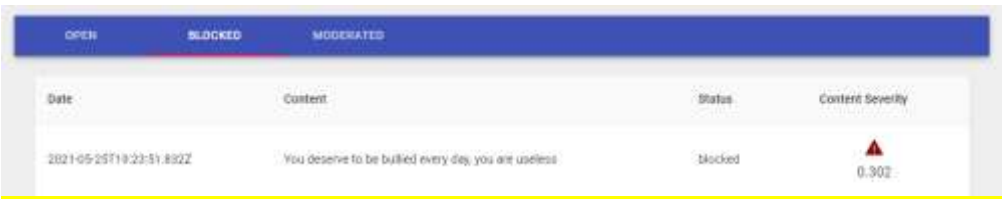


Рисунок 3.10 – загальний вигляд інтерфейсного вікна панелі заблоковних дописів

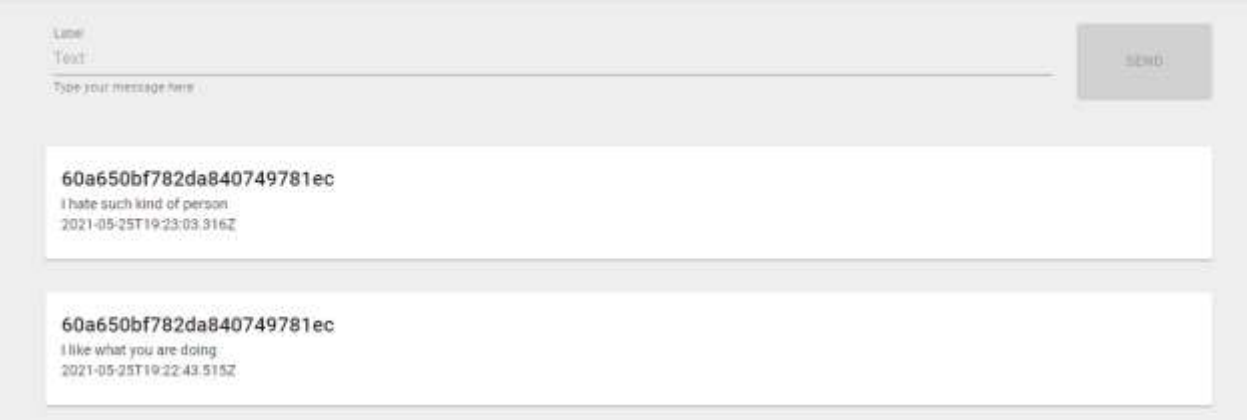


Рисунок 3.11 – Загальний вигляд інтерфейсного вікна дописів після видалення відмодерованого допису

Вийдемо зі свого облікового запису, усі дані про сесію видалено (рис. 3.12).

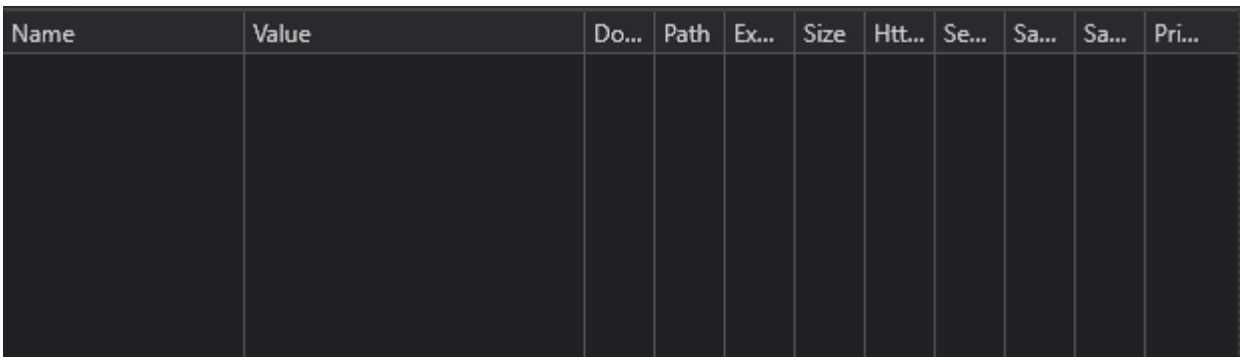


Рисунок 3.12 – Загальний вигляд вікна даних про сесію після завершення роботи із обліковим записом

Отже, в ході тестування створеної програми було перевірено її роботу згідно встановлених критеріїв, що забезпечило розширення функціональних можливостей в області аналізу текстів на наявність образливих висловлювань.

Деякий ілюстративний матеріал до програми (у т.ч. скріншоти) подано в додатку В. Інструкцію користування розробленою програмою наведено у додатку Г.

Модель універсального кодувальника речень була протестована на фрагменті даних, опублікованих центром Civil Research Data [25] та показала точність класифікації 95.32%, що є достатнім для впевненості в істинності аналізу кодувальника речень.

Так як метою роботи є покращення метрик продуктивності програмного продукту, проведемо порівняльний аналіз продуктивності розробленого програмного продукту і аналогів за допомогою інструментів Chrome DevTools Lighthouse [32]. Інструменти Lighthouse дозволяють виміряти метрики таких параметрів як:

- продуктивність;
- доступність;
- оптимальність методів;
- оптимізація для пошукових систем.

Приклад оцінки метрик розробленого програмного продукту наведено на рисунку 3.13.

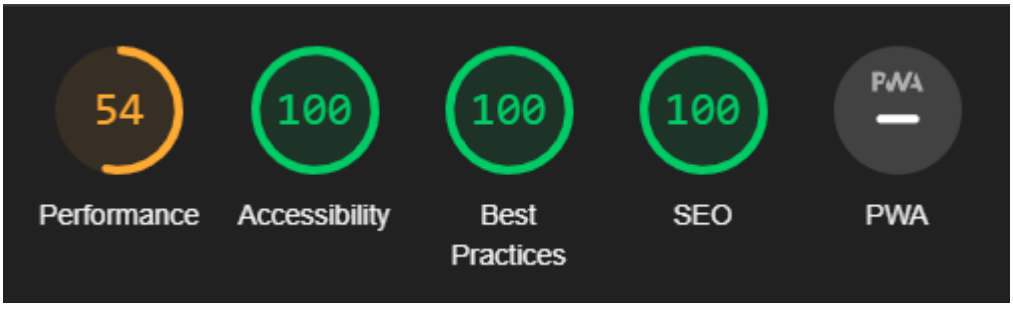


Рисунок 3.13 – Загальний вигляд інтерфейсного вікна Lighthouse із вимірами оптимізаційних метрик

Дані метрики відображають те, наскільки оптимізованим є програмний продукт, наскільки доступним є програмний продукт для людей із фізичними вадами, наскільки актуальними є методи і принципи, за якими створено програмний продукт, та наскільки оптимізованими є метадані програмного

продукту для індексації пошуковими двигунами, що допоможе спростити пошук програмного продукту користувачами.

Зведені дані аналізу метрик програмної реалізації інформаційної технології та аналогів наведено у таблиці 3.2.

Таблиця 3.2 – зведені результати аналізу метрик розробленого програмного продукту та аналогів

	Програмна реалізація інформаційної технології	BattlEye	Perspective API	PLNia
Продуктивність	54	34	30	22
Доступність	100	52	92	90
Оптимальність методів	100	83	92	92
Оптимізація для пошукових систем	100	79	83	93

Таким чином, програмна реалізація інформаційної технології аналізу текстів на наявність образливих висловлювань краща за аналоги за параметром продуктивності на 59%, за параметром доступності на 9%, за параметром оптимальності методів на 9%, за параметром оптимізації для пошукових систем на 7.5%.

3.4 Висновок до розділу 3

Таким чином, в даному розділі було розглянуто, проведено порівняльний аналіз мов програмування та технологій для програмної реалізації інформаційної технології аналізу текстів на наявність образливих висловлювань, обрано мову програмування JavaScript, бібліотеки React.js, Express, середовище Node.js, систему управління базами даних MongoDB та бібліотеку Tensorflow для реалізації універсального кодувальника речень. Використовуючи обрані технології, було розроблено програму, що реалізує функції та виконує задачі, описані у попередніх розділах. Проведено тестування розробленої програми за критеріями, що базуються на задачах дослідження та досягненні мети дослідження, а саме розширення функціональних можливостей в області аналізу текстів на наявність образливих висловлювань та покращення метрик продуктивності програмного продукту. Проведено перевірку достовірності аналізу тексту універсальним кодувальником речень. Проведено перевірку продуктивності програмного продукту та встановлено, що програмна реалізація інформаційної технології аналізу текстів на наявність образливих висловлювань краща за аналоги за параметром продуктивності на 59%, за параметром доступності на 9%, за параметром оптимальності методів на 9%, за параметром оптимізації для пошукових систем на 7.5%. Наступним етапом є розрахунок витрат на розробку, розрахунок чистого прибутку, визначення конкурентоспроможності розробки.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного продукту інформаційної технології аналізу текстів на наявність образливих висловлювань.

Особливістю програми є те, що дана технологія надає змогу розширення функціональних можливостей в області аналізу текстів з ціллю модерації контенту на веб-платформах та прийняття рішення щодо мір, які необхідно прийняти до коментарів, що порушують правила веб-платформи.

Аналогом може бути Perspective API – 10000 грн, BattlEye – від 40000 грн до 200000 грн, PLNia – 30000 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність проду-

Продовження табл. 4.1

Ринкові переваги					
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження табл. 4.1

Ринкові перспективи					
6	Ринок малий і не має	Ринок малий, але має пози-	Середній ринок з позитивною	Великий стабільний	Великий ринок з
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкуренті в немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження табл. 4.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовують	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	4	4
Наявність аналогів на ринку	4	4	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	3	4	4
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	4
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	4	4	3

Продовження табл. 4.2

Термін реалізації ідеї	4	4	3
Супровідна документація	4	3	4
Сума	46	44	45
Середньоарифметична сума балів	(46+44+45) / 3 = 45		

За даними таблиці 4.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.3.

Таблиця 4.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що інформаційна технологія відрізняється від існуючих розширенням функціональних можливостей, що забезпечило покращення процесу поширення та інтеграції системи в існуючі програмні рішення. Слід зауважити, що інформаційна технологія створюється із метою вирішення проблеми автоматизованого аналізу тексту на наявність елементів, що порушують правила веб-платформи, та пріорітетного пропонування на розгляд працівникам-модераторам на розгляд спірних випадків порушення, із забезпеченням доступу до декількох клієнтських рішень за допомогою механізму децентралізованого доступу.

4.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

4.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \tag{4.1}$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ – число робочих днів в місяці, 20 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 4.4.

Таблиця 4.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	30000	1500,00	40	60000,000
Інженер	27000	1350,00	40	54000,000
Всього				114000,00

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

4.2.2 Додаткова заробітна плата розробників, які приймали участь в розробці обладнання.

Додаткова заробітна плата прийнято розраховувати як 10 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 10\ \% / 100\ \% \qquad (4.2)$$

$З_д = (114000{,}00 \cdot 10\ \% / 100\ \%) = 11400{,}00$ (грн.)

4.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (4.3)$$

$Н_з = (114000{,}00 + 11400{,}00) \cdot 22\ \% / 100\ \% = 27588{,}00$ (грн.)

4.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

4.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12}\ \text{[грн.]}. \qquad (4.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 93000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 2,00 міс.

$$A_{обл} = \frac{93000}{2} \times \frac{2}{12} = 7750 \ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.5. Для розрахунку амортизації нематеріальних ресурсів використовується формула:

$$A_{н.р.} = Ц_{н.р.} * Н_а * \frac{t_{вик.}}{12} \qquad (4.5)$$

Але, так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $B_{нем.ак.}$ = 5300 грн.

Таблиця 4.5 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна переферія	93000	2	2,00	7750,000
Офісне обладнення	22000	4	2,00	916,667
Приміщення	900000	20	2,00	7500,000
Всього				16166,67

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних

постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (4.6)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,5 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$В_е = 0,9 \cdot 0,5 \cdot 8 \cdot 40 \cdot 6,2 = 892,8$ (грн.)

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.7)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$I_в$ = 114000,00 * 125% / 100% = 142500 (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (4.8)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$H_{нзв}$ = 114000,00 * 120 % / 100 % = 136800 (грн.)

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$B_{заг}$ = 114000,00+11400,00+27588,00+16166,67+5300+892,80+142500+
+136800 = 454647,47 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (4.9)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$ЗВ$ = 454647,47 / 0,5 = 909295 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації наринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);
- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

4.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.10)$$

де $\pm\Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

Цб – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 2000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *500* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 5000 шт., протягом другого року – на 10000 шт., протягом третього року на 15000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1$ = (0*500 + (2000 + 500)*5000)* 0,8333* 0,25) * (1 - 0,18) = 1708333,265 грн.

$\Delta П_2$ = (0*500 + (2000 + 500)*(5000+10000)* 0,8333* 0,25) * (1 - 0,18) = 6406249,744 грн.

$\Delta П_3$ = (0*500 + (2000 + 500)*(5000+10000+15000)* 0,8333* 0,25) * (1 - 0,18) = 12812499,488 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 20927082,50 грн.

4.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

ПП= (1708333,265/(1+0,1)^1)+(6406249,744/(1+0,1)^2)+(12812499,488/(1+0,1)^3) = 1553030,24 + 5294421,276 + 9626220,502 = 16473672,02 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (4.12)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$PV = 2 * 909295 = 1818589{,}87$ *грн.*

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП – PV, \quad (4.13)$$

$E_{абс} = 16473672{,}02 – 1818589{,}87 = 14655082{,}15$ грн.

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар’єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.14}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 14655082{,}15/1818589{,}87} - 1 = 1{,}085$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{4.15}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{4.16}$$

$$T_{ок} = 1 / 1{,}085 = 0{,}92 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,92 роки, то фінансування даної наукової розробки є доцільним.

4.4 Висновок до розділу 4

Економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 909295 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,92 роки.

ВИСНОВКИ

При виконанні магістерської кваліфікаційної роботи розв’язано задачу розробки інформаційної технології та програмного забезпеченя аналізу текстів на наявність образливих висловлювань.

1. Проаналізовано сучасний стан розвитку систем аналізу текстів на наявність образливих висловлювань.
2. Обґрунтовано вибір методу розв’язання задачі;
3. Спроектовано інформаційну технологію аналізу текстів на наявність образливих висловлювань.
4. Програмно реалізовано інформаційну технологію аналізу текстів на наявність образливих висловлювань.
5. Проведено тестування інформаційної технології аналізу текстів на наявність образливих висловлювань та аналіз результатів тестування.
6. Розраховано витрати на розробку, розраховано чистий прибуток, визначено конкурентоспроможність розробки.

Мета дослідження досягнута за рахунок розширення функціональних можливостей в аналізі текстів та на наявність образливих висловлювань покращення метрик продуктивності у web-середовищі за допомогою класифікації нейронною мережею та використанням механізму API-ключів.

Проаналізовано архітектуру універсального кодувальника речень. Для підвищення безпеки було обрано метод шифрування на основі геш-функції, розглянуто алгоритму гешування та обрано алгоритм SHA-2 як найбільш відповідний задачі збереження API-ключів. Під час програмної реалізації розглянуто плюси та мінуси мови програмування JavaScript та обґрунтовано її вибір, розроблено програму аналізу текстів на наявність образливих висловлювань, створену мовою програмування JavaScript із застосуванням бібліотек React.js, Express, Tensorflow, середовища Node.js і СУБД MongoDB. Було проведено тестування програми аналізу текстів на наявність образливих

висловлювань. Аналіз результатів тестування показує, що програма виконує усі поставлені задачі, що забезпечують розширення функціональних можливостей аналізу текстів на наявність образливих висловлювань. Проведено перевірку достовірності аналізу тексту універсальним кодувальником речень. Проведено перевірку продуктивності програмного продукту та встановлено, що програмна реалізація інформаційної технології аналізу текстів на наявність образливих висловлювань краща за аналоги за параметром продуктивності на 59%, за параметром доступності на 9%, за параметром оптимальності методів на 9%, за параметром оптимізації для пошукових систем на 7.5%.

Розроблений програмний продукт можливо покращити шляхом розширення функціональних можливостей, а саме імплементацією можливості взаємодії користувача з декількома програмними інтеграціями за допомогою асоціації декількох API-ключів із одним обліковим записом, що дозволить пришвидшити процес взаємодії із програмою та зменшити використання пам’яті для збереження окремих облікових записів для кожного API-ключа.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О. Шинкаренко. Розробка інформаційної технології аналізу текстів на наявність образливих висловлювань [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15948/13378
2. О. Шинкаренко. Інформаційна технологія аналізу текстів на наявність образливих висловлювань, III International Scientific and Practical Conference «Science and Technology. Problems, Prospects and Innovations», Osaka, 2022.
3. О. Шинкаренко. Розробка інтелектуальної системи аналізу тексту на наявність образливих висловлювань [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2021/paper/view/11738
4. О. Шинкаренко. Розробка додатку інтелектуального аналізу тексту на наявність образливих висловлювань [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/12600
5. J. Risch, R. Ruff, R. Krestel. Offensive Language Detection Explained, 2020. — 137c.
6. BattlEye documentation [Електронний ресурс] – Режим доступу: https://www.battleye.com/support/documentation/
7. Perspective API docs [Електронний ресурс] – Режим доступу: https://developers.perspectiveapi.com/s/docs?language=en_US
8. PLNia docs [Електронний ресурс] – Режим доступу: https://www.plnia.com/docs/
9. R. Pradhan, A. Chaturvedi, A. Tripathi, D.K. Sharma. A Review on Offensive Language Detection. [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/338355806_A_Review_on_Offensive_Language_Detection.
10. K. Pykes. Vector Space Models [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/vector-space-models-48b42a15d86d.

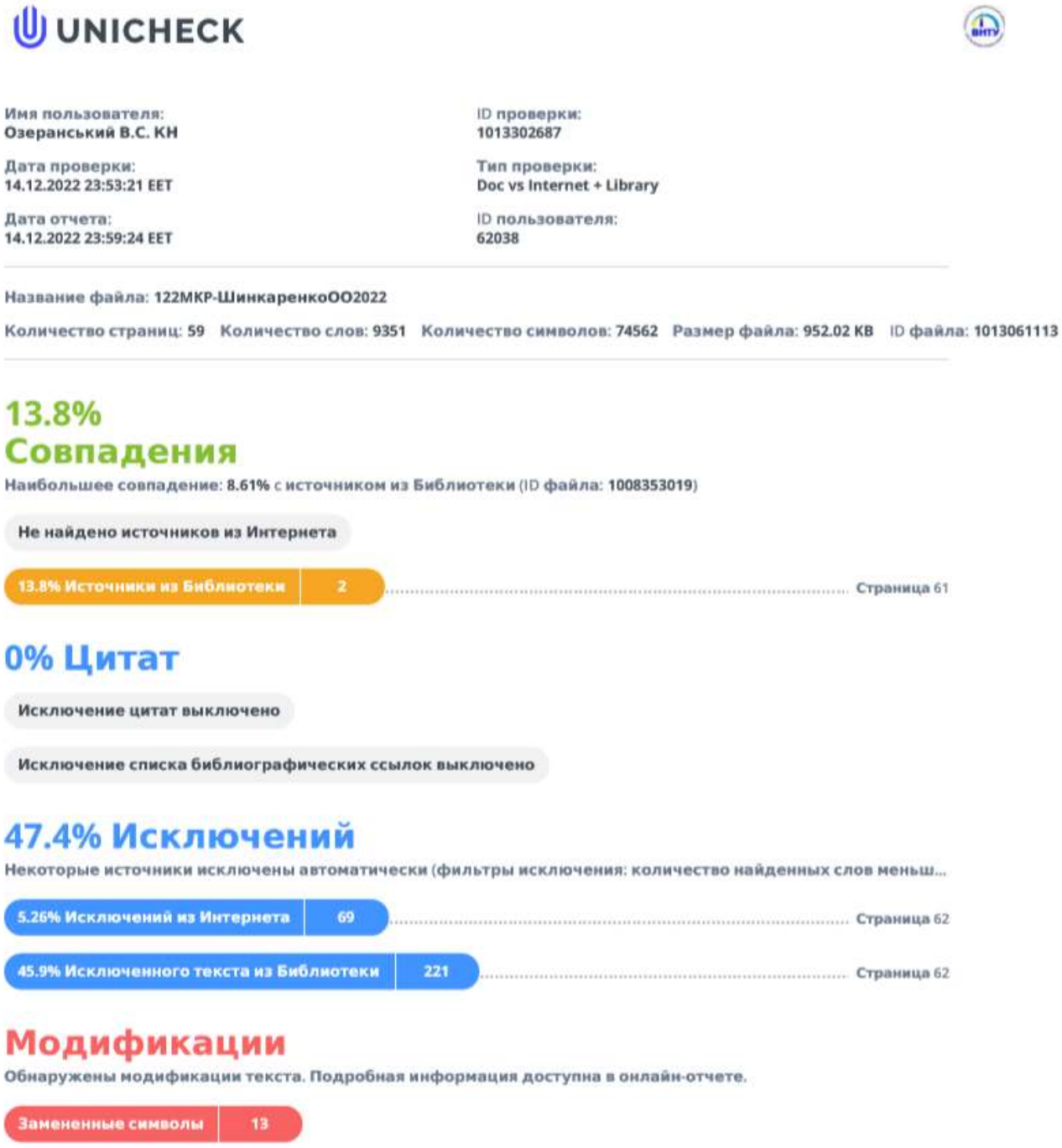
11. D.Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y. Sung, B. Strope, R. Kurzweil – Universal Sentence Encoder.
12. Y. Kim. Convolutional Neural Networks for Sentence Classification.
13. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin. Attention Is All You Need.
14. Д. Поликовський. Механизмы внимания в нейронных сетях.
15. E. Hoffmann. Standard Statistical Classifications: Basic Principles.
16. A. Kumar. Deep averaging network in Universal sentence encoder [Електронний ресурс] – Режим доступу: https://medium.com/tech-that-works/deep-averaging-network-in-universal-sentence-encoder-465655874a04
17. M. Iyyer, V. Manjuhatha, J. Boyd-Graber, Hal Daume III. Deep Unordered Composition Rivals Syntactic Methods for Text Classification.
18. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. Distributed Representations of Words and Phrases and their Compositionality
19. Hash Algorithm Comparison: MD5, SHA-1, SHA-2 & SHA-3 [Електронний ресурс] – режим доступу: https://codesigningstore.com/hash-algorithm-comparison
20. Загальна характеристика UML [Електронний ресурс] – Режим доступу: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=2
21. Д. Фленаган. JavaScript. Подробное руководство.
22. Документація по C# [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/ru-ru/dotnet/csharp/
23. Python 3.11.1 documentation [Електронний ресурс] – Режим доступу: https://docs.python.org/3/
24. MongoDB [Електронний ресурс] – Режим доступу: https://www.mongodb.com/
25. MySQL – Documentation [Електронний ресурс] – Режим доступу: https://dev.mysql.com/doc/

26. Redis – Documentation [Електронний ресурс] – Режим доступу: https://redis.io/documentation
27. ReactJS - Docs [Електронний ресурс] – Режим доступу: https://reactjs.org/docs/
28. Vue.js - Guide [Електронний ресурс] – Режим доступу: https://vuejs.org/v2/guide/
29. Angular - Docs [Електронний ресурс] – Режим доступу: https://angular.io/docs
30. Express.js - guide [Електронний ресурс] – Режим доступу: https://expressjs.com/en/guide
31. Introduction to GraphQL [Електронний ресурс] – Режим доступу: https://graphql.org/learn.
32. Lighthouse extension [Електронний ресурс] – Режим доступу: https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=en

ДОДАТКИ

Додаток А

Результат перевірки на плагіат в онлайн-системі UNICHECK



UNICHECK

Имя пользователя:
Озеранський В.С. КН

ID проверки:
1013302687

Дата проверки:
14.12.2022 23:53:21 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
14.12.2022 23:59:24 EET

ID пользователя:
62038

Название файла: 122МКР-ШинкаренкоОО2022

Количество страниц: 59 Количество слов: 9351 Количество символов: 74562 Размер файла: 952.02 KB ID файла: 1013061113

13.8%
Совпадения

Наибольшее совпадение: 8.61% с источником из Библиотеки (ID файла: 1008353019)

Не найдено источников из Интернета

13.8% Источники из Библиотеки 2 ..... Страница 61

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

47.4% Исключений

Некоторые источники исключены автоматически (фильтры исключения: количество найденных слов меньш...

5.26% Исключений из Интернета 69 ..... Страница 62

45.9% Исключенного текста из Библиотеки 221 ..... Страница 62

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы 13

Додаток Б

Лістинг програми

```
// Точка ініціалізації сервера

const express = require('express');
const cors = require('cors')
const connectDB = require('./config/db');
const app = express();


connectDB();

app.get('/', (req, res) => {
  res.send('API Running');
});

var whitelist = ['http://localhost:3000', 'http://localhost:3001']
var corsOptions = {
  origin: function (origin, callback) {
    if (whitelist.indexOf(origin) !== -1) {
      callback(null, true)
    } else {
      callback(new Error('Not allowed by CORS'))
    }
  }
}

//Init Middleware

app.use(express.json({ extended: false }));
app.use(cors(corsOptions));

//Define routes

app.use('/api/posts', require('./routes/api/posts'));
app.use('/api/users', require('./routes/api/users'));
app.use('/api/auth', require('./routes/api/auth'));
// app.use('/api/profile', require('./routes/api/profile'));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
  console.log(`Server started on port ${PORT}`);
});
const jwt = require('jsonwebtoken');
const config = require('config');
```

```
module.exports = function(req, res, next) {
  const token = req.header('x-auth-token');

  if (!token) {
    return res.status(401).json({ msg: 'No token, authorization denied' });
  }

  try {
    const decoded = jwt.verify(token, config.get('jwtSecret'));

    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({ msg: 'Token is not valid' });
  }
};


const mongoose = require('mongoose');
const config = require('config');
const db = config.get('mongoURI');

//Точка підключення БД
const connectDB = async () => {
  try {
    await mongoose.connect(db, {
      useNewUrlParser: true,
      useCreateIndex: true,
      useFindAndModify: false,
      useUnifiedTopology: true
    });
    console.log('MongoDB Connected...');
  } catch (err) {
    console.error(err.message);
    process.exit(1);
  }
};

module.exports = connectDB;

const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const auth = require('../../middleware/auth');
const jwt = require('jsonwebtoken');
const config = require('config');
const { check, validationResult } = require('express-validator/check');

const User = require('../../models/User');
```

```
// обробник авторизації

//@route  GET api/auth
//@desc   Test route
//@access Public
router.get('/', auth, async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select('-password');
    res.json(user);
  } catch (err) {
    console.error(err.message);
    res.status(500).send('Server Error');
  }
});

router.post(
  '/',
  [
    check('email', 'Please type a vaild email').isEmail(),
    check('password', 'Password is required').exists()
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email, password } = req.body;

    try {
      let user = await User.findOne({ email });
      if (!user) {
        return res
          .status(400)
          .json({ errors: [{ msg: 'Invalid credentials' }] });
      }

      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return res
          .status(400)
          .json({ errors: [{ msg: 'Invalid credentials' }] });
      }
      const payload = {
        user: {
          id: user.id
        }
      };

      jwt.sign(
```

```
        payload,
        config.get('jwtSecret'),
        { expiresIn: 360000 },
        (err, token) => {
          if (err) throw err;
          res.json({ token });
        }
      );
    } catch (err) {
      console.error(err.message);
      res.status(500).send('Server error');
    }
  }
);

module.exports = router;

const User = require('../../models/User');
const Post = require('../../models/Post');
const e = require('express');

require('@tensorflow/tfjs');
const toxicity = require('@tensorflow-models/toxicity');

// Критерій точності аналізу

const threshold = 0.8;

// Показники важливості критеріїв

const criteriaSeverities = {
    'identity_attack': 0.2,
    'insult': 0.2,
    'obscene': 0.2,
    'severe_toxicity': 0.3,
    'sexual_explicit': 0.3,
    'threat': 0.3,
    'toxicity': 0.1
};

// @route  POST api/users
// @desc   register user
// @access Public
router.get(
    '/',
    async (req, res) => {
        const errors = validationResult(req);

        if (!errors.isEmpty()) {
            return res.status(400).json({ errors: errors.array() });
```

```
        }

        const token = req.header('x-auth-token');
        let userFromToken = {};

        if (token) {
            const decoded = jwt.verify(token, config.get('jwtSecret'));

            userFromToken = decoded.user;
        }

        const { deleted, moderated, open } = req.query;
        const { id } = userFromToken;

        try {
            let posts;
            let user;
            }

            const payload = {
                posts
            };

      res.json(payload);

        } catch (err) {
            console.error(err.message);
            res.status(500).send('Server error');
        }
    }
);


router.post('/post', auth, [
    check('content', 'message text is required').not().isEmpty()
],
async (req, res) => {
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
    }

    const token = req.header('x-auth-token');
    let userFromToken = {};

    if (token) {
        const decoded = jwt.verify(token, config.get('jwtSecret'));

        userFromToken = decoded.user;
```

```
    } else {
        res.status(401).send('Forbidden');
    }

    const { id } = userFromToken;

    const { content } = req.body;

    try {
        const user = await User.findById(id).select('-password');

        if (user.banned) {
            res.status(400).json({ error: 'user is banned from posting' });
        } else {

            // аналіз та класифікація тексту допису

            const model = await toxicity.load(threshold);
            const analysisResult = await model.classify([ content ]);
            const severityCoefficient = analysisResult.reduce((acc, criteria) => {
                const criteriaProbability = criteria.results[0].match ? 1 : criteri
a.results[0].probabilities['1'];

                return acc + criteriaSeverities[criteria.label] * criteriaProbabili
ty;
            }, 0);

            const post = new Post({
                userId: id,
                content,
                moderation: {
                    severityCoefficient: severityCoefficient % 1,
                    analysisResult,
                    moderated: false
                } });

            await post.save();
            const payload = {
                user: {
                    id: post.id,
                    userId: post.userId,
                    content: post.content,
                    date: post.date
                }
            };

            res.json(payload);
        }
    } catch (err) {
        console.error(err.message);
```

```
        res.status(500).send('Server error');
    }
}
);

router.post('/moderate', auth,
    check('postId', 'postId is required').not().isEmpty(),
    check('status', 'updated status is required').not().isEmpty(),
    async (req, res) => {
        const errors = validationResult(req);

        if (!errors.isEmpty()) {
            return res.status(400).json({ errors: errors.array() });
        }

        const token = req.header('x-auth-token');
        let userFromToken = {};

        if (token) {
            const decoded = jwt.verify(token, config.get('jwtSecret'));

            userFromToken = decoded.user;
        } else {
            res.status(401).send('Forbidden');
        }

        const { id } = userFromToken;

        const { postId, status } = req.body;

        try {
            let user;

            user = await User.findById(id).select('-password');

            if (user.role !== 'admin') {
                res.status(402).send('Forbidden');
            } else {
                const post = await Post.findByIdAndUpdate(postId, {
                    'moderation.moderated': true,
                    'moderation.moderationDate': Date.now(),
                    status
                });
                const payload = {
                    post
                };

                res.json(payload);
            }
        } catch (err) {
```

```
            console.error(err.message);
            res.status(500).send('Server error');
        }
    }
);

module.exports = router;

import { useMemo } from 'react';
import { useDispatch } from 'react-redux';

const useActionCreators = (creators) => {
    const dispatch = useDispatch();

    return useMemo(() => creators.map(creator => (...params) => dispatch(creator(..
.params))), [ ...creators ]);
};

export const fetchPost = (postId) => (dispatch, getState) => {
    const sessionToken = getState().getIn(['user', 'sessionToken']);

    return dispatch({
        [RSAA]: {
            endpoint: getEndpointUrl('post', {postId}),
            method: 'GET',
            headers: {
                'Content-Type': 'application/json',
                'x-auth-token': sessionToken
            },
            types: [
                POST_FETCH_REQUEST,
                POST_FETCH_SUCCESS,
                POST_FETCH_FAILURE
            ]
        }
    });
};

export const moderatePost = (postId, status) => (dispatch, getState) => {
    const sessionToken = getState().getIn(['user', 'sessionToken']);

    return dispatch({
        [RSAA]: {
            endpoint: getEndpointUrl('moderate'),
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'x-auth-token': sessionToken
            },
            body: JSON.stringify({
```

```
                postId,
                status
            }),
            types: [
                POST_MODERATE_REQUEST,
                POST_MODERATE_SUCCESS,
                POST_MODERATE_FAILURE
            ]
        }
    });
};

export const sendPost = (content) => (dispatch, getState) => {
    const sessionToken = getState().getIn(['user', 'sessionToken']);

    return dispatch({
        [RSAA]: {
            endpoint: getEndpointUrl('sendPost'),
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'x-auth-token': sessionToken
            },
            body: JSON.stringify({
                content
            }),
            types: [
                POSTS_SEND_REQUEST,
                POSTS_SEND_SUCCESS,
                POSTS_SEND_FAILURE
            ]
        }
    });
};
const postReducer = (state = Map(), action = {}) => {
    switch(action.type){
        case POST_FETCH_SUCCESS:
            if(action.payload.post){
                return state.set('postView', action.payload.post);
            }
            return state;
        case CLEAR_POST:
            return state.clear();
        default:
            return state;
    }
}

export default postReducer;
```

```
import { getEndpointUrl } from '../utils/fetchHelpers';

import { LOGOUT } from './user';

export const fetchPosts = (tab) => (dispatch, getState) => {
    const sessionToken = getState().getIn(['user', 'sessionToken']);

    const tabs = [
        'open',
        'deleted',
        'moderated'
    ];

    let params = {};

    if(tab !== undefined){
        params = {
            [`${tabs[tab]}`]: true
        }
    }

    const headers = {
        'Content-Type': 'application/json',
    };

    if(sessionToken){
        headers['x-auth-token'] = sessionToken;
    }

    return dispatch({
        [RSAA]: {
            endpoint: getEndpointUrl('posts', {...params}),
            method: 'GET',
            headers,
            types: [
                POSTS_FETCH_REQUEST,
                POSTS_FETCH_SUCCESS,
                POSTS_FETCH_FAILURE
            ]
        }
    });
};

export const clearPosts = () => ({
    type: CLEAR_POSTS
})

const postsReducer = (state = OrderedMap(), action = {}) => {
    switch(action.type){
        case POSTS_FETCH_SUCCESS:
```

```
            if(action.payload.posts){
                return action.payload.posts.reduce((acc, post) => {
                    return acc.set(post._id, post);
                }, OrderedMap());
            }

            return state;
        case LOGOUT:
        case CLEAR_POSTS:
            return state.clear()
        default:
            return state;
    }
}

export default postsReducer;

export const auth = (email, password) => (dispatch, getState) => {
    return dispatch({
        [RSAA]: {
            endpoint: getEndpointUrl('auth'),
            body: JSON.stringify({
                email,
                password
            }),
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            types: [
                USER_AUTH_REQUEST,
                USER_AUTH_SUCCESS,
                USER_AUTH_FAILURE
            ]
        }
    });
};

export const getUserProfile = () => (dispatch, getState) => {
    const sessionToken = getState().getIn(['user', 'sessionToken']);

    return dispatch({
        [RSAA]: {
            endpoint: getEndpointUrl('profile'),
            method: 'GET',
            headers: {
                'Content-Type': 'application/json',
                'x-auth-token': sessionToken
            },
            types: [
```

```
                USER_PROFILE_REQUEST,
                USER_PROFILE_SUCCESS,
                USER_PROFILE_FAILURE
            ]
        }
    });
};

export const logout = () => ({
    type: LOGOUT
})

const userReducer = (state = Map(), action = {}) => {
    switch(action.type){
        case USER_AUTH_SUCCESS:
            Cookies.set('sessionToken', action.payload.token);

            return state.set('sessionToken', action.payload.token);
        case USER_PROFILE_SUCCESS:
            return state.merge(fromJS(action.payload.user));
        case LOGOUT:
            Cookies.remove('sessionToken');

            return state.clear();
        default:
            return state;
    }
}

export default userReducer;

import { userActions } from '../../ducks';
import useActionCreators from '../../hooks/useActionCreators';


const re = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/
const Login = () => {
    const classes = useStyles();
    const [ email, setEmail ] = useState('');
    const [ emailError, setEmailError ] = useState();
    const [ password, setPassword] = useState('');
    const [ passwordError, setPasswordError ] = useState();
    const [ emailManuallyChanged, setEmailManuallyChanged ] = useState(false);
    const [ passwordManuallyChanged, setPasswordManuallyChanged ] = useState(false)
;
    const [ loading, setLoading ] = useState(false);
    const [ auth ] = useActionCreators([userActions.auth]);
    const loggedIn = useSelector(state => !!state.getIn(['user', 'sessionToken']));
    const location = useLocation();
    const history = useHistory();
```

```
    useEffect(() => {
        if(loggedIn && location.pathname === '/login'){
            history.replace('/');
        }
    }, [loggedIn, location, history]);

    useEffect(() => {
        if(passwordManuallyChanged && password.length < 6){
            !passwordError && setPasswordError('minimum length is 6')
        } else {
            passwordError && setPasswordError(false)
        }

        if(emailManuallyChanged && !email.match(re)){
            !emailError && setEmailError('invalid email')
        } else {
            emailError && setEmailError(false)
        }
    }, [email, password, passwordManuallyChanged, emailManuallyChanged, emailError,
 passwordError]);

    const onEmailChange = (e) => {
        !emailManuallyChanged && setEmailManuallyChanged(true)
        setEmail(e.target.value);
    }

    const onPasswordChange = (e) => {
        !passwordManuallyChanged && setPasswordManuallyChanged(true);
        setPassword(e.target.value);
    }

    const login = () => {
        setLoading(true);
        auth(email, password).then(() => {
            setLoading(false);
        });
    };

export default Login;

import { postActions, postsActions } from '../../ducks';
import useActionCreators from '../../hooks/useActionCreators';

export default memo(PostDetailsCard);
import PostDetailsCard from '../PostDetailsCard';

import { postsActions, postActions } from '../../ducks';

import useActionCreators from '../../hooks/useActionCreators';
```

```
const getColor = (value) => {
    if(value <= 0.1){
        return '#007700';
    } else if (value >= 0.3){
        return '#880000';
    } else {
        return '#FCE205'
    }
}

let timer = null;

const AdminTable = ({posts, tab}) => {
    const classes = useStyles();
    const [sortedPosts, setSortedPosts] = useState(posts);
    const [sortingRule, setSortingRule] = useState();
    const [order, setOrder] = useState();
    const [openedPost, setOpenedPost] = useState();

    const applySorting = useCallback((fieldName, changeOrder = true) => () => {
        if(fieldName === 'date'){
            if(sortingRule !== fieldName){
                setSortingRule('date');
                setSortedPosts(posts.sort((a, b) => {
                    return Date.parse(a.date) < Date.parse(b.date);
                }));
                changeOrder && setOrder('asc');
            } else {
                setSortingRule('date');
                changeOrder && setOrder(order === 'asc' ? 'desc' : 'asc');
                setSortedPosts(sortedPosts.reverse())
            }
        } else if(fieldName === 'severityCoefficient'){
            if(sortingRule !== fieldName){
                setSortingRule('severityCoefficient');
                setSortedPosts(posts.sort((a, b) => {
                    return a.moderation.severityCoefficient < b.moderation.severity
Coefficient;
                }));
                changeOrder && setOrder('asc');
            } else {
                setSortingRule('severityCoefficient');
                changeOrder && setOrder(order === 'asc' ? 'desc' : 'asc');
                setSortedPosts(sortedPosts.reverse());
            }
        }
    });

    useEffect(() => {
```

```
        if(
            posts.size !== sortedPosts.size || !posts.every(post => sortedPosts.get
(post._id) && post.status === sortedPosts.get(post._id).status)
        ){
            setSortedPosts(posts);
            sortingRule && applySorting(sortingRule, false)
        };
    }, [sortingRule, sortedPosts, posts, applySorting]);

    const openPost = (postId) => () => {
        setOpenedPost(null);
        setOpenedPost(postId);
    }

    const closePost = useCallback(() => {
        setOpenedPost(null);
    }, []);

    if(!posts.size){
        return (
            <Typography variant='h5'>No posts available in this category, try chang
ing the tab</Typography>
        )
    }

    return(
        <Fragment>
            {openedPost && <PostDetailsModal tab = { tab } postId={openedPost} clos
ePost = { closePost }/>}
            <TableContainer component = { Paper }>
            <Table stickyHeader size="medium">
                <TableHead>
                    <TableRow>
                        <TableCell onClick={applySorting('date')} className={classe
s.rowPointer}>
                            <TableSortLabel direction={sortingRule === 'date' ? ord
er : 'asc' }>
                                Date
                            </TableSortLabel>
                        </TableCell>
                        <TableCell align="left">Content</TableCell>
                        <TableCell align="left">Status</TableCell>
                        <TableCell align="center" onClick={applySorting('severityCo
efficient')}>
                            <TableSortLabel direction={sortingRule === 'severityCoe
fficient' ? order : 'asc'}>
                                Content Severity
                            </TableSortLabel>
                        </TableCell>
                    </TableRow>
```

```
                </TableHead>
                    <TableBody>
                        {sortedPosts.map((post, index) =>
                            post.moderation && <TableRow hover onClick = { openPost
(post._id) } key={index} classNameName={classes.rowPointer}>
                                <TableCell component="th" scope="row">
                                    {post.date}
                                </TableCell>
                                <TableCell align="left">
                                    <span classNameName = {classes.ellipsisText}>
                                        {post.content}
                                    </span>
                                </TableCell>
                                <TableCell align="left">{post.status}</TableCell>
                                <TableCell align="center">
                                    <WarningIcon style={{color: getColor(post.moder
ation.severityCoefficient)}} />
                                    <Typography>{post.moderation.severityCoefficien
t.toFixed(3)}</Typography>
                                </TableCell>
                            </TableRow>
                        ).toList().toArray()}
                    </TableBody>
            </Table>
        </TableContainer>
        </Fragment>)
};

const PostDetailsModal = memo(({postId, closePost, tab}) => {
    const classes = useStyles();
    const [ fetchPost, clearPost ] = useActionCreators([postActions.fetchPost, post
Actions.clearPost]);
    const [ loading, setLoading ] = useState(true);
    const [ open, setOpen ] = useState(true);
    const postInfo = useSelector(state => state.getIn(['post', 'postView' ]));

    useEffect(() => {
        fetchPost(postId).then(() => setLoading(false));
    }, [])

    const closeHandler = () => {
        setOpen(false);
        clearPost();
        setTimeout(() => closePost(), 500);
    }

    return (
        <Modal
            classNameName={classes.modal}
            open={open}
```

```
            onClose={closeHandler}
            closeAfterTransition
            BackdropComponent={Backdrop}
            BackdropProps={{
            timeout: 500,
          }}
        >

          <Fade in={open}>
              <Fragment>
                  <div className={classes.paper}>
                  {loading && <Box className={classes.spinnerHolder}>
                          <CircularProgress />
                  </Box> }
                  {!loading && postInfo &&
                      <div style={{height: '100%', width: '100%',}}>
                          <div className={classes.close}>
                              <CloseIcon className={classes.rowPointer} onClick = {cl
oseHandler} fontSize="large"/>
                          </div>
                          <PostDetailsCard tab = { tab } post = { postInfo } closeHan
dler ={ closeHandler } />
                      </div>}
                  </div>
              </Fragment>
          </Fade>
        </Modal>
    );
});

function TabPanel(props) {
    const { children, value, index, ...other } = props;

    return (
      <div
        role="tabpanel"
        hidden={value !== index}
        id={`simple-tabpanel-${index}`}
        aria-labelledby={`simple-tab-${index}`}
        {...other}
      >
        {value === index && (
          <Box p={3}>
            {children}
          </Box>
        )}
      </div>
    );
```

Додаток В

105

Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АНАЛІЗУ ТЕКСТІВ НА НАЯВНІСТЬ ОБРАЗЛИВИХ ВИСЛОВЛЮВАНЬ

Виконав: студент 2-го курсу,
групи 1КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Шинкаренко О.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)
«14» 12 2022 р.

Рисунок В.1 – Загальна структурна схема інформаційної технології аналізу текстів на наявність образливих висловлювань

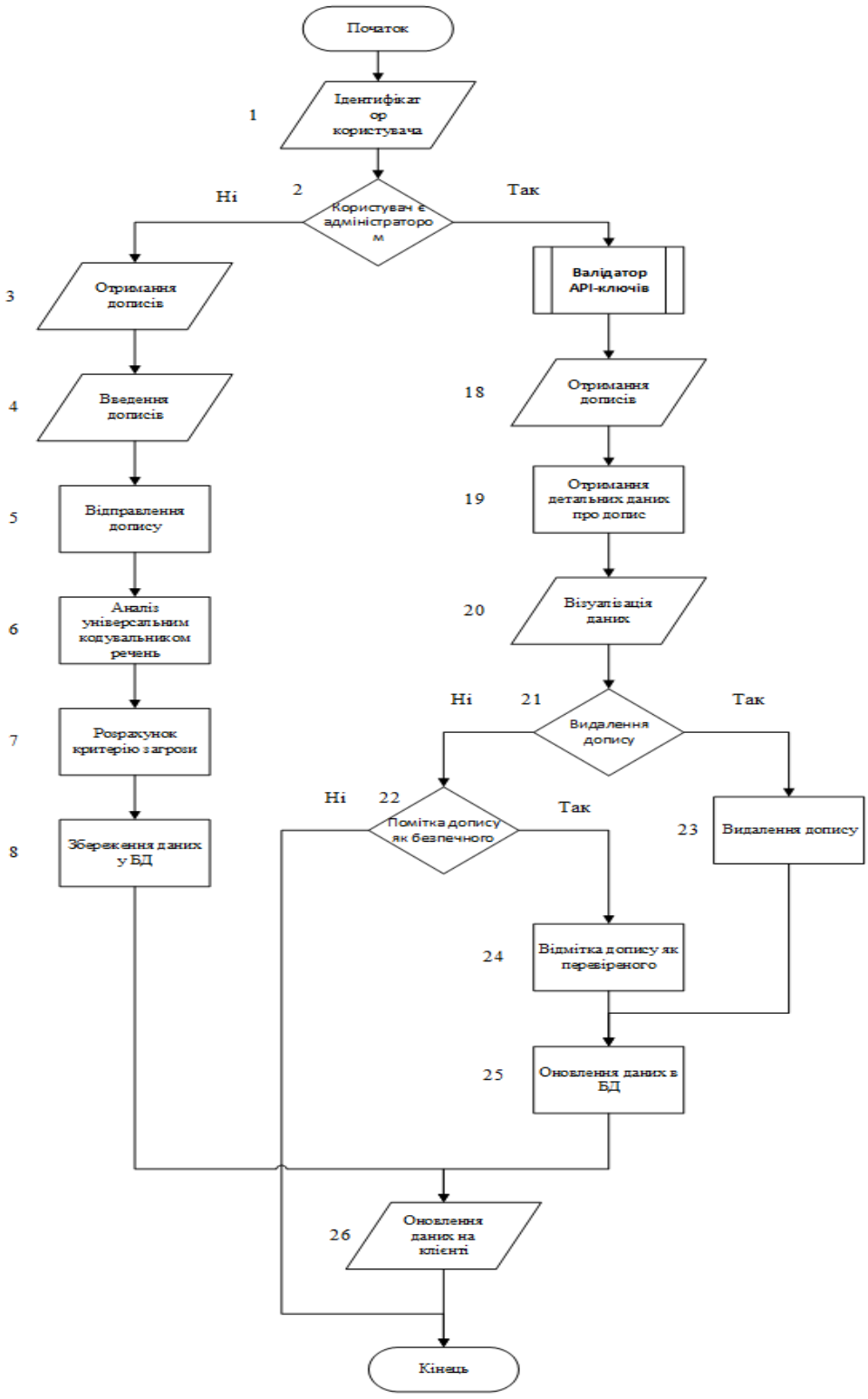


Рисунок В.2 – Схема основного алгоритму роботи інформаційної технології аналізу текстів на наявність образливих висловлювань

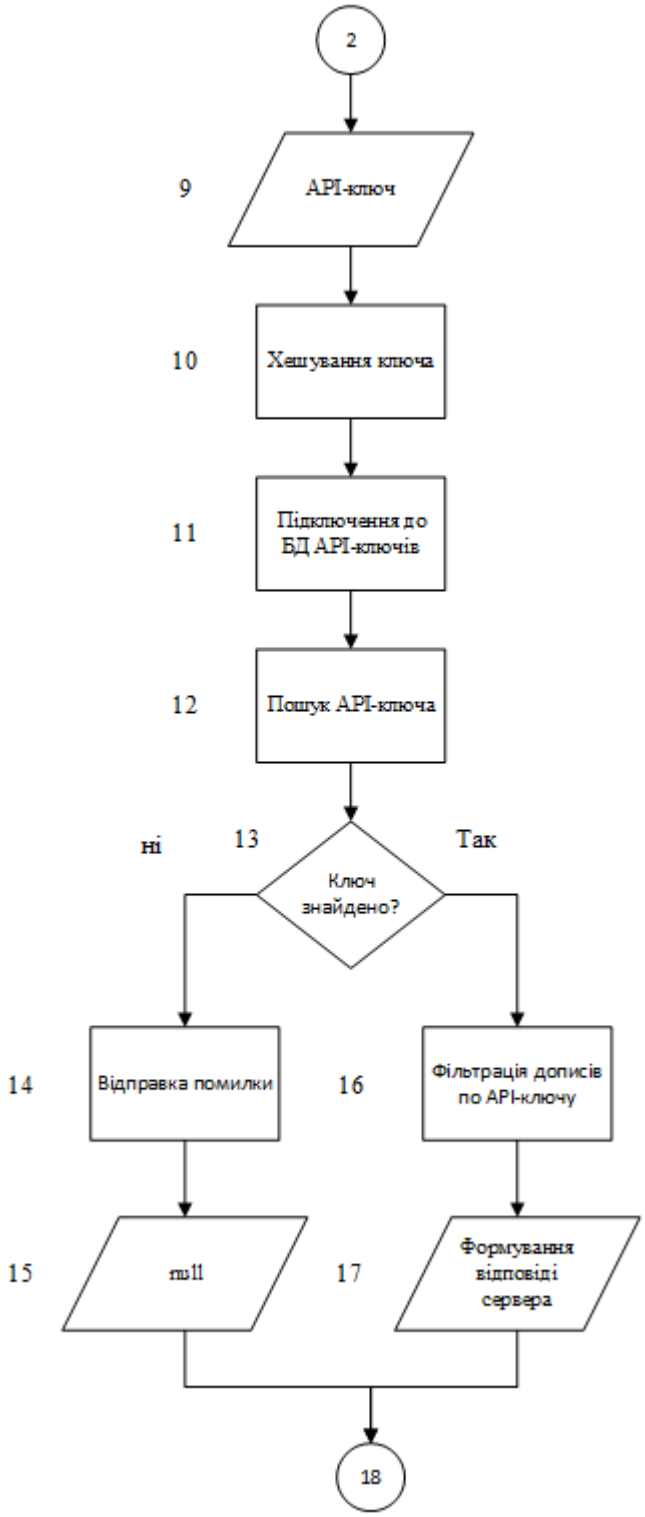


Рисунок В.3 – Схема алгоритму підпроцесу валідації API-ключів інформаційної технології аналізу текстів на наявність образливих висловлювань

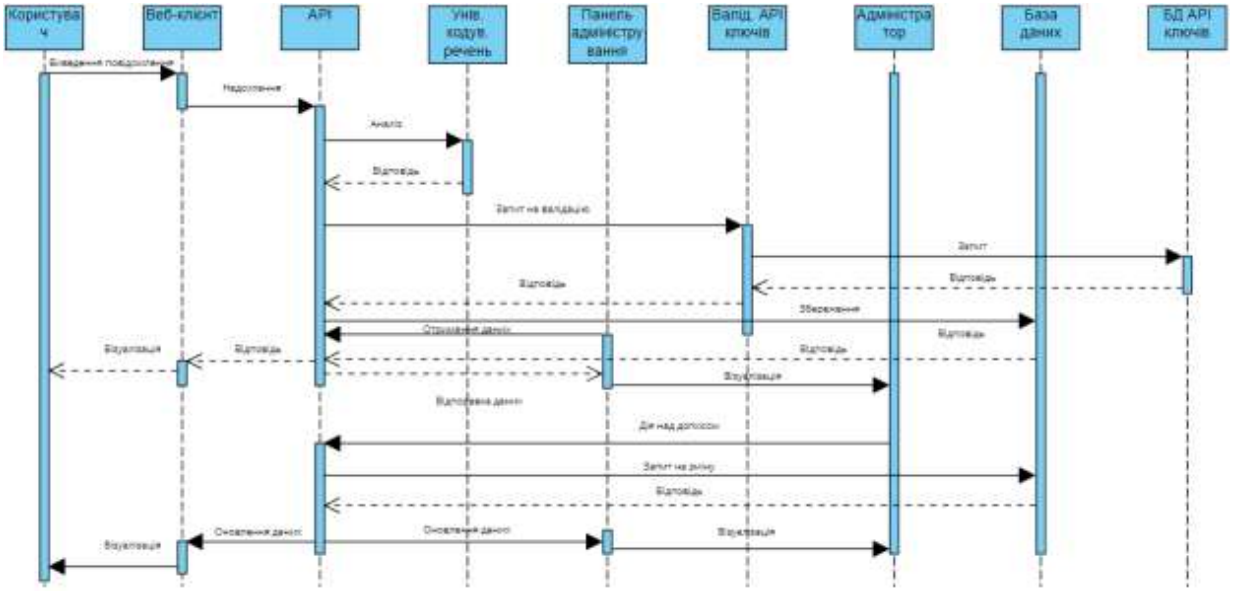


Рисунок В.4 – Діаграма послідовності взаємодії об’єктів в інформаційній технології аналізу текстів на наявність образливих висловлювань

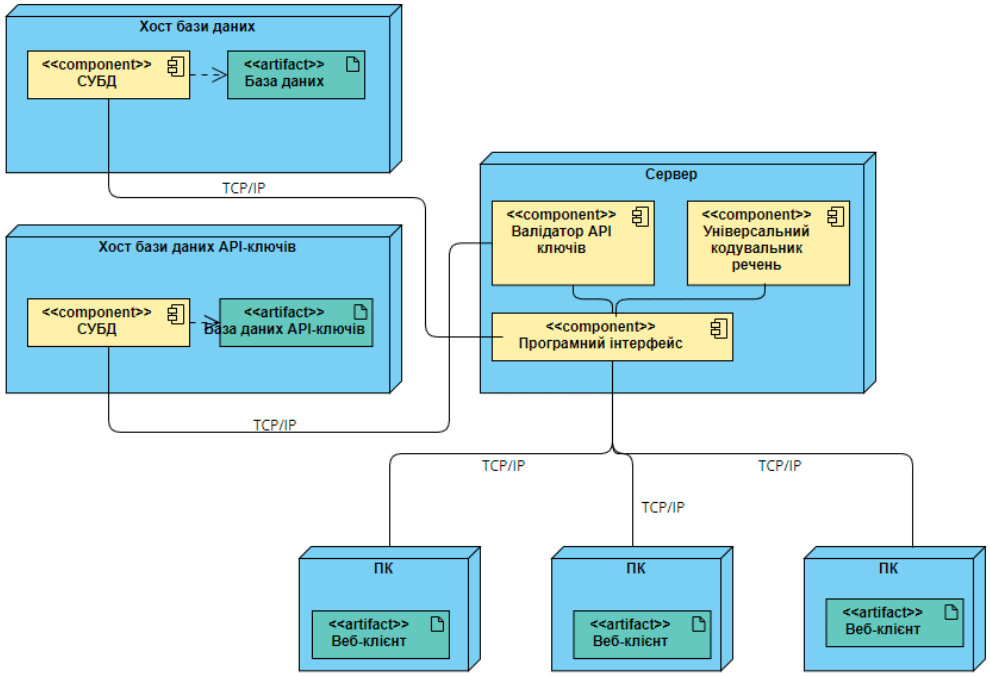
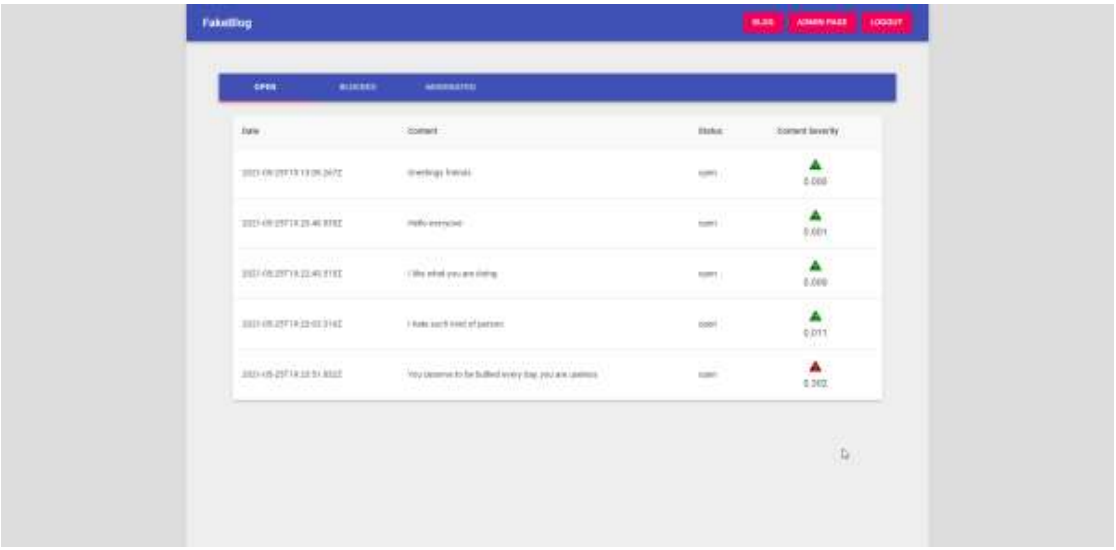


Рисунок В.5 – Діаграма розгортування інформаційної технології аналізу текстів на наявність образливих висловлювань



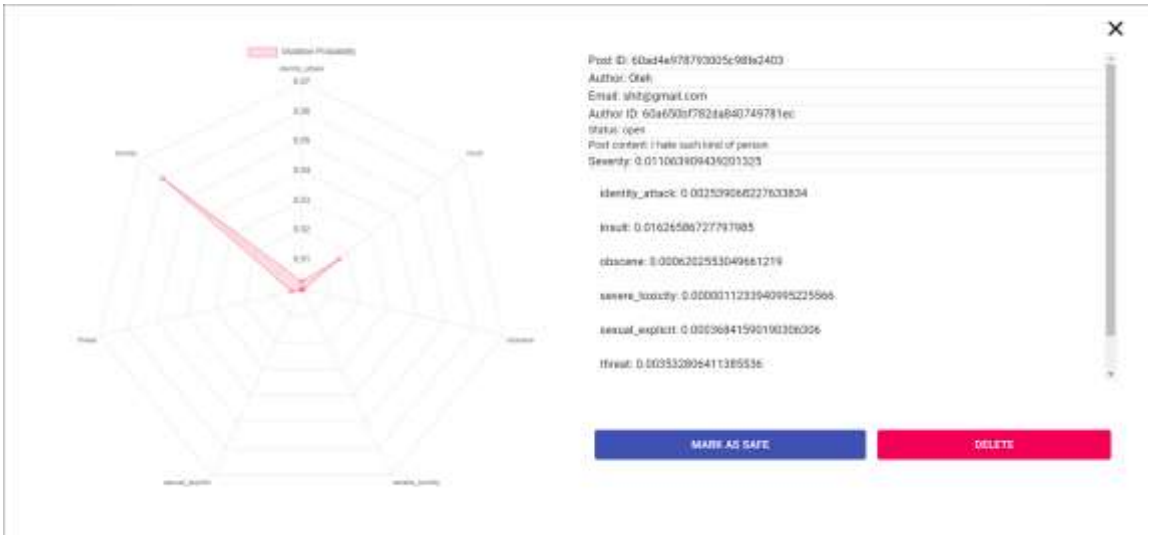
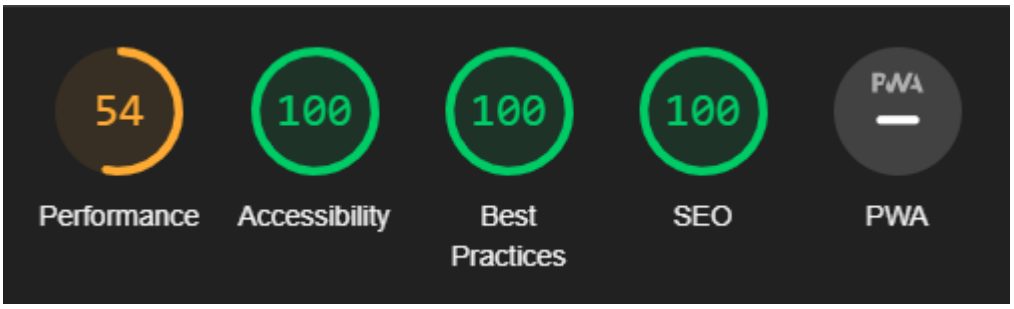


Рисунок В.6 – Загальний вигляд інтерфейсних вікон інформаційної технології аналізу текстів на наявність образливих висловлювань



	Програмна реалізація інформаційної технології	BattlEye	Perspective API	PLNia
Продуктивність	54	34	30	22
Доступність	100	52	92	90
Оптимальність методів	100	83	92	92
Оптимізація для пошукових систем	100	79	83	93

Рисунок В.7 – Результати тестування розробленої програми та програм-аналогів