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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка веб чату з використанням моделей аналізу тексту

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-21-2
Губарєв Р.В.

Керівник кваліфікаційної роботи Шаповалова Н.Н.

Завідувач кафедри _____

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Освітньо-кваліфікаційний рівень: бакалавр

Спеціальність: 121 - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____

«___» _____ 20__ р.

З А В Д А Н Н Я

на кваліфікаційну роботу

студенту групи ІПЗ-21-2 Губарєву Ростиславу Вадимовичу

1. Тема: Розробка веб чату з використанням моделей аналізу тексту

затверджено наказом по КНУ №200с від «10» 04 2025 р.

2. Термін подання студентом закінченої роботи « » 2025 р.

3. Вихідні дані по роботі: розроблюваний додаток повинен працювати без помилок, модель повинна розпізнавати токсичні слова в повідомленнях.

4. Зміст пояснювальної записки: ознайомитися з програмними аналогами, розробити алгоритм роботи додатку, написати код клієнтської та серверної частини програми, дослідити моделі аналізу тексту, створити та протестувати модель аналізу тексту на токсичність, інтегрувати модель в додаток, протестувати та налагодити додаток.

5. Перелік ілюстративного матеріалу: функціональні схеми програми, діаграма діяльності, ER-діаграма, зображення з фрагментами коду програми, зображення з інтерфейсом додатку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Ознайомлення з програмними аналогами, визначення їх переваг та недоліків.	24.02.2025 – 01.03.2025
2	Ознайомлення з документацією на тему проекту	01.03.2025 – 02.03.2025
3	Розробка алгоритму роботи та логіки додатку	03.03.2025 – 10.03.2025
4	Написання коду серверної частини програмного забезпечення	11.03.2025 – 20.03.2025
5	Написання коду клієнтської частини програмного забезпечення	21.03.2025 – 02.04.2025
6	Дослідження моделей аналізу тексту	03.04.2025 – 05.04.2025
7	Пошук чи створення набору даних для навчання моделі	06.04.2025
8	Обробка даних та навчання моделі аналізу тексту на визначення токсичності	07.04.2025 – 10.04.2025
9	Тестування та порівняння декількох навчених моделей	11.04.2025 – 18.04.2025
10	Інтеграція моделі в додаток	19.04.2025 – 23.04.2025
11	Тестування та налагодження додатку	24.04.2025 – 10.05.2025
12	Оформлення пояснювальної записки	01.05.2025 – 25.05.2025

Дата видачі завдання: «28» січня 2025 р.

Студент Губарєв Р. В.

Керівник роботи Шаповалова Н.Н.

РЕФЕРАТ

ВЕБЧАТ В РЕАЛЬНОМУ ЧАСІ, МОДЕЛЬ АНАЛІЗУ ТЕКСТУ, РОЗПІЗНАВАННЯ ТОКСИЧНОСТІ, НЕЙРОМЕРЕЖЕВІ ТЕХНОЛОГІЇ, ПОСЛІДОВНИЙ АЛГОРИТМ НАВЧАННЯ.

Пояснювальна записка: 40 с., 0 табл., 36 рис., 10 джерел, 0 додатків.

Метою кваліфікаційної роботи є створення веб-чату з використанням моделі аналізу тексту для спілкування в реальному часі.

У роботі проведено аналіз існуючих аналогів указаної задачі – наявних веб чатів, месенджерів та моделей аналізу тексту. Виконано їх порівняння з погляду дизайну, функціоналу та зручності використання. Проведено порівняння моделей аналізу тексту на український наборах даних. Для розв’язання задачі було обрано MERN (MongoDB, Express.js, React, Node.js) стек для створення додатку, та мову програмування Python для навчання моделі.

Результатом виконання курсового проекту стало створення вебчату в реальному часі з інтеграцією моделі аналізу тексту на токсичність.

ABSTRACT

REAL-TIME WEB CHAT, TEXT ANALYSIS MODEL, TOXICITY DETECTION, NEURAL NETWORK TECHNOLOGIES, SEQUENTIAL LEARNING ALGORITHM.

Thesis in 40 pages, 0 tables, 36 figures, 10 references, 0 app.

The goal of this qualification work is to develop a real-time web chat application that integrates a text analysis model for detecting toxic content during communication.

The project includes an analysis of existing solutions, such as web-based chat applications, messengers, and text analysis models. These alternatives were compared based on design, functionality, and user experience. A comparative study of various text classification models trained on Ukrainian language datasets was also carried out.

To implement the solution, the MERN stack (MongoDB, Express.js, React, Node.js) was chosen for building the web application, while Python was used to train the machine learning model.

As a result, a real-time web chat was successfully developed, featuring integrated toxicity detection powered by a neural network-based text analysis model.

ЗМІСТ

ВСТУП

У сучасну епоху цифрової трансформації обмін інформацією в режимі реального часу став невід'ємною частиною повсякденного життя. Вебчати є важливим інструментом комунікації в найрізноманітніших сферах: від технічної підтримки клієнтів до освітніх та соціальних платформ. Проте з ростом обсягів повідомлень та активності користувачів виникає потреба в інтелектуальному аналізі текстового контенту, зокрема для автоматичної фільтрації небажаних повідомлень, виявлення токсичності, спаму, маніпуляцій або емоційного забарвлення діалогів.

Актуальність теми полягає в необхідності поєднання технологій веброзробки з інструментами штучного інтелекту, зокрема моделями обробки природної мови (NLP), для створення більш безпечних, ефективних та інтерактивних чат-систем. Використання моделей аналізу тексту дозволяє в режимі реального часу здійснювати модерацію, класифікацію повідомлень та персоналізувати взаємодію з користувачем. Це особливо важливо у контексті підвищених вимог до інформаційної безпеки, захисту від мовленнєвої агресії та покращення користувацького досвіду.

Метою кваліфікаційної роботи є розробка веб-чату, що інтегрує модель аналізу тексту для автоматичної обробки повідомлень. Основна увага приділяється виявленню токсичності в комунікації, що дозволяє забезпечити більш безпечне середовище для користувачів. У межах реалізації поставленої мети планується створити інтерфейс веб-чату, розробити серверну логіку для обробки повідомлень, навчити модель аналізу тексту та інтегрувати її в систему в режимі реального часу.

Додано примітку [H1]: Кваліфікаційної роботи – тут і всюди замінити

Запропоноване рішення може бути корисним для платформ, що потребують інтелектуальної модерації контенту, таких як освітні чати, корпоративні месенджери або онлайн-сервіси підтримки клієнтів.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Критичний аналіз джерел за темою

В процесі дослідження було опрацьовано наукові публікації, статті, документації та технічні звіти, що стосуються тематики вебчатів, обробки природної мови (NLP) та інтеграції машинного навчання в системи онлайн-комунікації.

У роботах [1] та [2] запропоновано архітектури трансформерів, які поклали початок широкому застосуванню моделей типу BERT, GPT та їх похідних у задачах аналізу тексту. Ці підходи демонструють високу точність у виявленні токсичності, тональності та інших характеристик тексту. Проте їх використання у вебдодатках часто пов'язане з високими обчислювальними витратами, що потребує оптимізації або застосування легших моделей.

Додано примітку [H2]: У роботах [1]та [2]... тут і всюди поміняти на таких формат посилань на джерела

З огляду на сучасні практики веброзробки, важливою є документація фреймворків (React, Express.js, Socket.io), а також інструкції щодо інтеграції моделей машинного навчання через API або TensorFlow. Однак більшість таких джерел зосереджені на технічній реалізації і не розглядають аспекти безпеки, етики або соціальних наслідків застосування моделей аналізу тексту в чатах.

Щодо виявлення токсичності, найчастіше використовуються англомовні датасети, такі як Jigsaw (від Google), що демонструє проблему домінування англомовних підходів у дослідженнях. Це створює виклики, якщо проєкти орієнтовані на україномовну аудиторію.

З цього виходить, що наявні джерела надають теоретичну та технічну базу для реалізації вебчату з NLP-моделлю, однак спостерігається розрив між західними дослідженнями та українським контекстом. Це підтверджує доцільність створення власного рішення з адаптованою моделлю для української мови.

1.2 Актуальність теми кваліфікаційної роботи

У сучасному цифровому світі вебкомунікація стала основним способом взаємодії між користувачами в різних сферах - освіті, бізнесі, технічній підтримці, соціальних мережах тощо. Вебчати забезпечують зручний та швидкий обмін повідомленнями, однак із зростанням кількості користувачів постає проблема якості комунікації, зокрема поширення токсичних, агресивних або спам-повідомлень. Ці явища негативно впливають на безпеку, емоційний комфорт та ефективність онлайн-взаємодії.

Актуальність розробки вебчату з інтегрованою моделлю аналізу тексту полягає в необхідності автоматизованого контролю змісту повідомлень. Завдяки методам обробки природної мови (NLP) та машинного навчання з'являється можливість виявляти токсичні висловлювання, мовлення ворожнечі, образи, а також інші небажані елементи в режимі реального часу. Це дозволяє створити безпечніше й комфортніше середовище для спілкування.

Крім того, значна частина сучасних рішень у сфері аналізу тексту орієнтована на англомовний сегмент. Натомість існує обмежена кількість ефективних україномовних рішень, здатних здійснювати семантичний аналіз, класифікацію чи фільтрацію повідомлень. Тому розробка вебчату з підтримкою української мови та інтелектуальними функціями текстової модерації має як практичне, так і соціальне значення.

Загалом, актуальність обраної теми зумовлена потребою у сучасних вебзасобах комунікації, здатних забезпечувати не лише технічну функціональність, а й автоматичну модерацію контенту на основі технологій штучного інтелекту.

1.3 Цілі та завдання кваліфікаційної роботи

Мета роботи – розробка функціонального вебчату з інтеграцією моделі аналізу тексту, що виявляє токсичні повідомлення. Такий підхід спрямований на забезпечення безпечного середовища для користувачів та демонстрацію

можливостей застосування технологій обробки природньої мови в вебсередовищі.

Для досягнення поставленої мети треба виконати такі завдання:

1) провести дослідження сучасних підходів реалізації вебчатів, та методів аналізу тексту з використанням нейромереж;
2) знайти або створити датасет україномовних текстів з токсічними та нейтральними контекстами для навчання та тестування моделі;
3) розробити модель аналізу тексту, яка здатна класифікувати повідомлення за ознаками токсичності;
4) реалізувати клієнтську частину вебчату з використанням сучасних фреймворків (React, Node.js, Socket.io);
5) розробити серверну частину вебчату, яка буде забезпечувати обробку повідомлень та взаємодію з моделлю;
6) інтегрувати модель аналізу тексту в чат для перевірки змісту повідомлень в реальному часі;
7) протестувати розроблений додаток та оцінити його ефективність;
8) оформити результати розробки у вигляді текстової доповіді кваліфікаційної роботи та підготувати матеріали до захисту.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Розробка та докладний опис функціональної схеми програми

Розробка такого проекту, як вебчат, потребує ретельної підготовки. Треба скласти план розробки, а також в деталях продумати архітектуру системи для забезпечення повноцінної роботи застосунку. Для цього була зроблена функціональна схема, яка відображає архітектуру клієнтської та серверної частини (рисунки 2.1 – 2.5) [4].

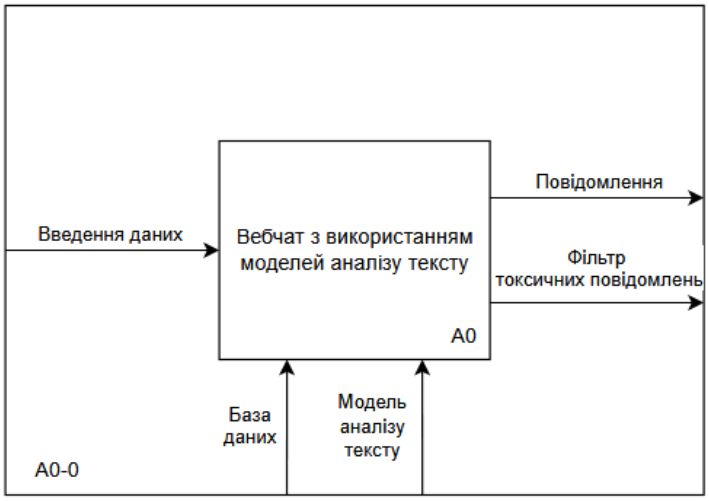

Рисунок 2.1 – Нульовий рівень функціональної схеми програми

Додано примітку [H3]: Краще написати слово Нульовий рівень
Ту т і всюди в рисунках – довге тире

На першому рівні показана загальна структура системи, а саме збір та обробка даних, фільтрація даних та виведення даних. Показано на яких етапах використовується база даних, а на яких – модель аналізу тексту. Варто зазначити, що "обробка інформації" в етапі A1 означає обробку даних про користувача при реєстрації чи авторизації в системі, а "обробка інформації" в A2 – обробка повідомлення перед його виведенням.

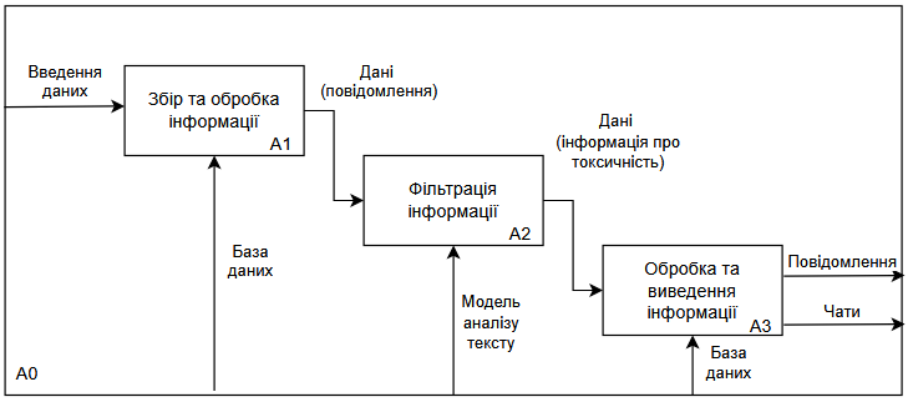

Рисунок 2.2 – Перший рівень функціональної діаграми

На рівні A1 зображені етапи збору та обробки даних. Показані реєстрація, авторизація користувача, а також відправлення повідомлення. На цьому рівні база даних використовується на кожному етапі.

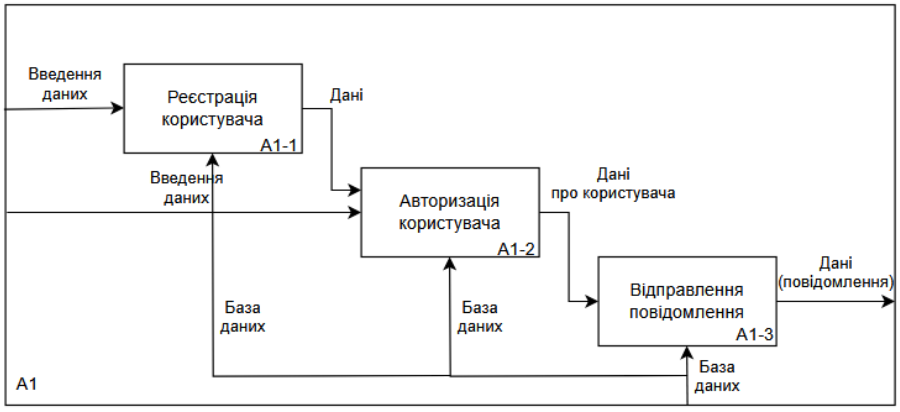

Рисунок 2.3 – Етап A1. Збір та обробка даних

На наступному етапі демонструються кроки фільтрації інформації. Відправлене користувачем повідомлення проходить через модель аналізу тексту. Натомість, модель відправляє інформацію про токсичність тексту.

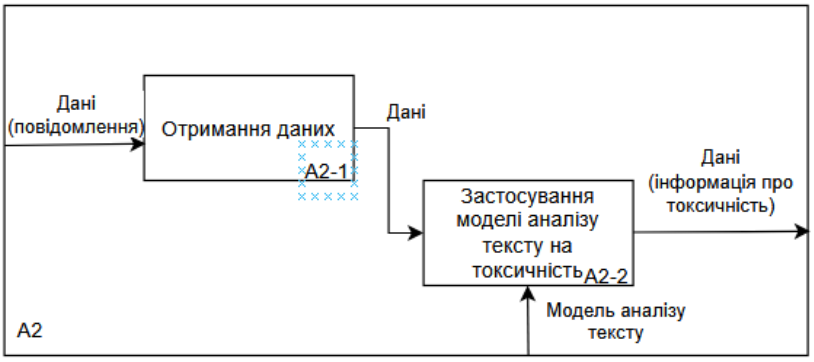

Рисунок 2.4 – Етап А2. Застосування моделі аналізу тексту

На заключному кроці система приймає дані від моделі, і в залежності від інформації про токсичність повідомлення, виводить зацензурене чи незацензурене речення.

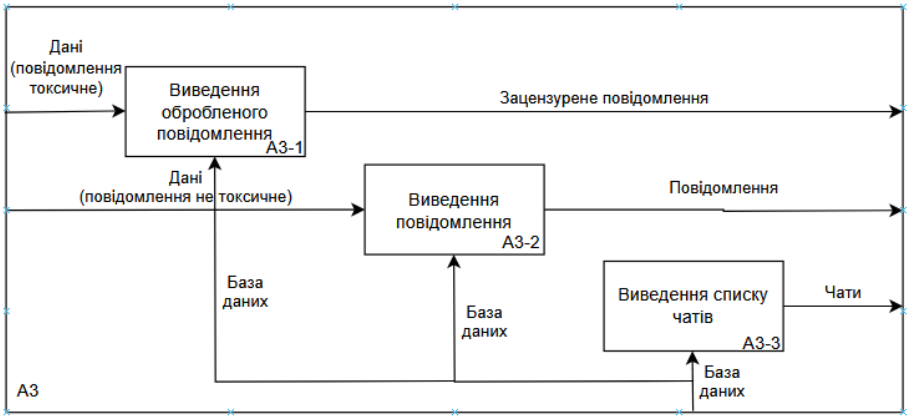

Рисунок 2.5 – Етап А3. Обробка та виведення інформації

2.2 Розробка та докладний опис алгоритму роботи програми

Для моделювання процесів і роботи програми використовуються діаграми діяльності. Діаграма діяльності – це блок-схема, що відображає перехід від однієї діяльності до іншої [5].

В цьому розділі розроблено та представлено діаграму діяльності, яка показує логіку функціонування вебчату з інтегрованою моделлю аналізу тексту (рисунок 2.6). Також наведено детальний опис алгоритму роботи програми.

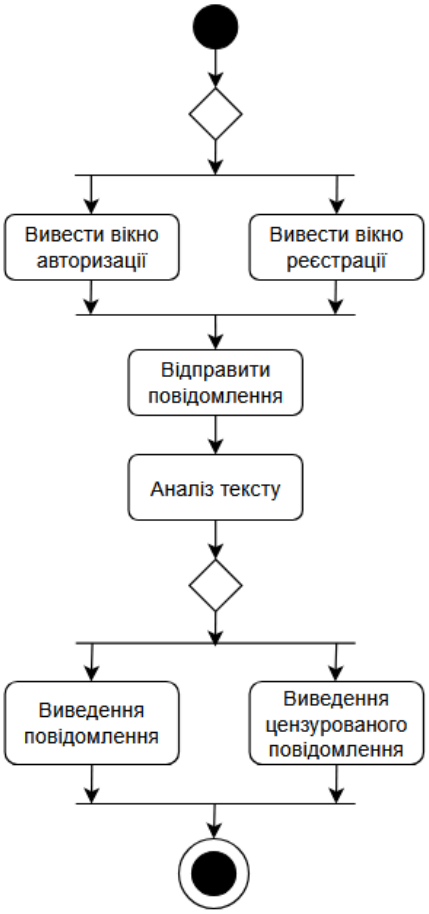

Рисунок 2.6 – Діаграма діяльності

Система побудована за клієнт-серверною архітектурою. Ця архітектура працює наступним чином: клієнт надсилає запит на сервер, сервер робить

запит до бази даних, база даних надає доступну інформацію, і сервер повертає цю інформацію клієнту [6].

Алгоритм роботи базується на обробці тексту в реальному часі та виявленні токсичності в повідомленнях користувачів перед їх виведенням.

Алгоритм взаємодії між компонентами:

1) користувач вводить повідомлення;
2) клієнтська частина надсилає повідомлення на сервер за допомогою HTTP-запиту (POST);
3) сервер приймає повідомлення та передає його до моделі аналізу тексту, де виконується його попередня обробка;
4) модель аналізує текст та визначає повідомлення токсичне чи ні;
5) модель генерує відповідь у вигляді мітки (1 – токсичне, 0 – ні) та відправляє назад на сервер;
6) сервер отримує результат аналізу;
7) відповідь надсилається клієнту, де повідомлення відображається відповідно результату аналізу. Якщо повідомлення токсичне, то його не видно, доки користувач не натисне, щоб розблокувати.

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Додано примітку [H4]: Новий розділ з нової сторінки

3.1 Аналіз обраної середи програмування

В цьому розділі обґрунтовано вибір інструментів, мов програмування, бібліотек, фреймворків та технологій, які будуть використані для реалізації вебчату з інтеграцією моделі аналізу тексту.

1) Клієнтська частина:

Для реалізації клієнтської частини було обрано React через його гнучкість та компонентну структуру. Цей JavaScript фреймворк дозволяє створювати інтерфейс вебчату, обробляти події користувача та оновлювати вміст сторінки без перезавантаження, а ще він швидкий та оптимізований [8].

Для стилів використовувався фреймворк Tailwind CSS. Він надає великий набір класів, які можна використовувати безпосередньо в HTML-розмітці. Завдяки своїй високій модульності, ці класи дозволяють створювати гнучкі дизайни без потреби писати власний CSS код. Це пришвидшує розробку, покращує продуктивність та облегшує обслуговування [9].

Також я використав daisyUI для таких компонентів, як кнопки, поля для введення, аватари та багато іншого. Це плагін для Tailwind CSS, в якому зберігається безліч компонентів з гарним сучасним дизайном. Завдяки ньому не треба писати власні компоненти, тому що всі вони вже вбудовані в плагін, і це пришвидшує розробку в рази.

2) Серверна частина:

Для серверної частини було використано Node.js та Express.js.

Node.js – це платформа для створення серверних застосунків, які працюють в режимі "реалтайм". Express.js дозволяє швидко створювати REST API та налаштовувати маршрутизацію між клієнтом, сервером та моделлю аналізу тексту.

Також було використано сервіс Postman для тестування HTTP запитів.

Для реалізації обміну повідомленнями в реальному часі було використано Socket.io. Це JS бібліотека, яка використовується для забезпечення миттєвої комунікації між клієнтом і сервером, що є ключовим для реалізації реалтайм вебчата.

3) Модель аналізу тексту

Для створення моделі аналізу тексту було використано Python з бібліотеками TensorFlow та Keras. Python – це основна мова машинного навчання.

Фреймворк Keras забезпечує просту реалізацію нейромереж, а також має потужні інструменти для попередньої обробки тексту, що дуже корисно в моделі аналізу тексту на токсичність, адже ця модель потребує контекст, в якому були вжиті ті, чи інші слова.

У поєднанні з TensorFlow модель може бути інтегрована в вебчат.

4) База даних

Для зберігання історії повідомлень, даних користувачів та чатів використовується документо-орієнтована база даних MongoDB. Ця база даних використовує колекції та документи замість стандартних таблиць і рядків, що краще відповідає об’єктоорієнтованому підходу, який зазвичай застосовують розробники при створенні класів та об’єктів [10].

3.2 Розробка бази даних

Основними об’єктами в базі даних є користувачі, повідомлення та чати. Для реалізації бази даних обрано документо-орієнтовану СУБД MongoDB, яка ідеально підходить для сучасних вебзастосунків завдяки своїй гнучкості та масштабованості.

Основні колекції бази даних:

1) users – користувачі.

Ця колекція містить інформацію про зареєстрованих користувачів, де:

_id – ідентифікатор користувача;

fullName – повне ім’я;

username – ім’я користувача;

password – хешований пароль;

gender – стать;

profilePicture – фото профілю.

Приклад документа:

Рисунок 3.1 – Документ з користувачем

2) messages – повідомлення.

Ця колекція містить текстові повідомлення з індексом відправника та отримувача, а також часом створення повідомлення, де:

_id – ідентифікатор повідомлення;

senderId – ідентифікатор відправника;

receiverId – ідентифікатор отримувача;

message – текст повідомлення;

createdAt – час створення.

Приклад документа:

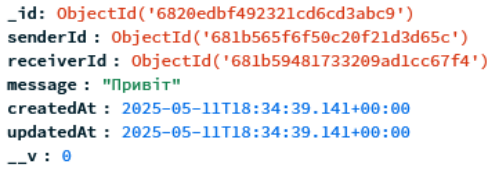

Рисунок 3.2 – Документ з повідомленням

3) conversations – чати.

Ця колекція містить чати, де

_id – ідентифікатор чата;

participants – учасники;

messages – список повідомлень;

Приклад документа:

Рисунок 3.3 – Документ з чатом

Для опису зв’язків між класами було зроблено ER-діаграму. ER-діаграма (діаграма "сутність-зв'язок") допомогає пояснити логічну структуру баз даних. В її основі лежать ключові елементи: сутності, їх атрибути та взаємозв'язки між ними (рисунок 3.4).

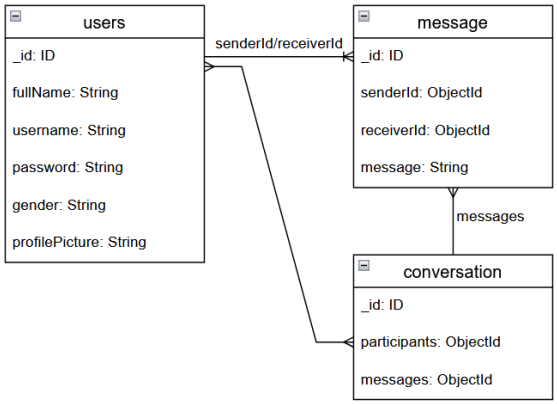

Рисунок 3.4 – ER-діаграма вебчату

Зв’язки між класами:

1) user – message (один до багатьох)

Один користувач може як надсилати, так і отримувати багато повідомлень.

Зв’язок:

user._id – message.senderId

user._id – message.receiverId

2) conversation – user (багато до багатьох)

Один чат може містити кількох учасників, і один користувач може брати участь у багатьох чатах.

Зв’язок:

conversation.participants[] – user._id

3) conversation – message (один до багатьох)

Кожен чат містить багато повідомлень.

Зв’язок:

conversation.messages[] – message._id

3.3 Програмна реалізація основних функцій

В цьому розділі описані ключові функціональні можливості вебчату. Детально розглянуто, як реалізовано обробку повідомлень, реєстрацію та авторизацію користувачів. Також показаний шлях реалізації та інтеграції моделі аналізу тексту. Кожна функція проілюстрована відповідним фрагментом коду.

3.3.1 Розробка функцій додатку

Розробка додатку почалася з бекенд-частини. На цьому етапі було підключено базу даних MongoDB, яка забезпечує збереження всієї необхідної інформації.

Для обробки запитів, пов’язаних із реєстрацією, входом та виходом користувача було створено маршрут auth.routes.js, який використовує Express

Router (рисунок 3.5). Цей маршрут організовує шляхи до відповідних функцій у контролері auth.controller.js.

```
You, 2 months ago | 1 author (You)
1  import express from "express";
2  import { login, logout, signup } from "../controllers/auth.
   controller.js";
3
4  const router = express.Router();
5
6  router.post("/signup", signup);
7
8  router.post("/login", login);
9
10 router.post("/logout", logout);
11
12 export default router;        You, 2 months ago • Initial com
```

Рисунок 3.5 – auth.routes.js

Файл auth.controller.js містить основну логіку для обробки процесів реєстрації, входу та виходу користувачів. Контролер виконує роль між вхідними HTTP-запитами та базою даних, забезпечуючи виконання необхідних перевірок, генерацію токенів та відповідей клієнту.

Основні функції:

- signup(req, res) – обробляє запит на реєстрацію нового користувача.
  - перевіряє, чи користувач існує;
  - хешує пароль за допомогою бібліотеки bcrypt (рисунок 3.6);
  - створює нового користувача в базі даних;
  - повертає відповідь про успішну реєстрацію або помилку.
- login(req, res) – обробляє запит на авторизацію користувача.
  - знаходить користувача за введеним логіном;
  - перевіряє правильність паролю;
  - генерує JWT-токен;
  - повертає токен та інформацію про користувача.
- logout(req, res) – обробляє запит на вихід користувача.

- видаляє токен;
- виводить повідомлення про успішний вихід або помилку.

```
// HASH PASSWORD
const salt = await bcrypt.genSalt(12);
const hashedPassword = await bcrypt.hash(password, salt);
```

Рисунок 3.6 – Хешування паролю

Робота HTTP-запитів тестувалася в сервісі Postman.

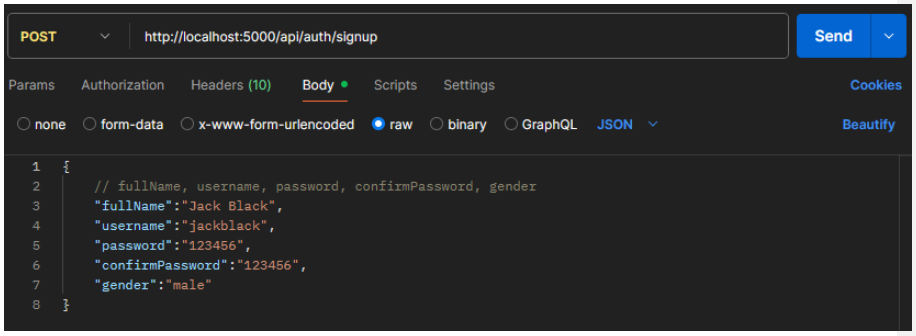

Рисунок 3.7 – Інтерфейс сервісу Postman. Тестування реєстрації

Також, було створено message.controller.js, в якому описано логіку для відправлення та отримання повідомлень. Файл містить два основні методи для роботи з повідомленнями в вебчаті, відправлення повідомлення та отримання списку повідомлень з іншим користувачем. Контролер також інтегрується з Socket.io для забезпечення відправки та отримання повідомлень в реальному часі.

Основні функції:

- sendMessage(req, res) – функція, що відповідає за створення нового повідомлення.

- отримується текст повідомлення з тіла запиту, а також ID отримувача з параметрів запиту. ID відправника береться з авторизованого користувача;

```
const { message } = req.body;
const { id: receiverId } = req.params;
const senderId = req.user._id;
```

Рисунок 3.8 – Отримання даних в функції sendMessage

- перевіряється, чи існує чат між учасниками. Якщо ні – створюється новий;

```
const conversation = await Conversation.findOne({
    participants: {
        $all: [senderId, receiverId]
    }
})

if (!conversation) {
    conversation = await Conversation.create({
        participants: [senderId, receiverId]
    })
}
```

Рисунок 3.9 – sendMessage. Перевірка існування та створення чату

- створюється об’єкт повідомлення з ID відправника, отримувача та текстом;
- повідомлення додається до масиву повідомлень у чаті, після чого обидва об’єкти зберігаються до бази даних;

```
const newMessage = new Message({
    senderId,
    receiverId,
    message
});

if (newMessage) {
    conversation.messages.push(newMessage._id);
}

await Promise.all([conversation.save(), newMessage.save
()]);
```

Рисунок 3.10 – Створення повідомлення та збереження в базі даних

- визначається socket ID отримувача (якщо він онлайн), і через Socket.io повідомлення надсилається в реальному часі подією newMessage.

- getMessages(req, res) – функція, що відповідає за отримання історії повідомлень між двома користувачами.
  - отримується userToChatId з параметрів запиту та senderId з авторизованого користувача;
  - здійснюється пошук розмови між двома користувачами. Якщо чат не знайдено – повертається порожній масив;
  - використовується populate(“messages”), щоб отримати повні дані всіх повідомлень із MongoDB

```
export const getMessages = async (req, res) => {
    try {
        const { id: userToChatId } = req.params;
        const senderId = req.user._id;

        const conversation = await Conversation.findOne({
            participants: {
                $all: [senderId, userToChatId]
            }
        }).populate("messages");

        if (!conversation) { return res.status(200).json([]); }

        const messages = conversation.messages;

        res.status(200).json(messages);

    } catch (error) {
        console.log("Error in getting messages ", error.message);
        res.status(500).json({
            message: error.message
        });
    }
};
```

Рисунок 3.11 – Функція getMessages

Файл user.controller.js відповідає за формування списку користувачів, який відображається на боковій панелі чату. Основна мета цього контролеру – показати всіх доступних користувачів для спілкування.

Логіка роботи:

- отримується ID авторизованого користувача;
- за допомогою запиту до MongoDB вибирається всі користувачі, ID яких не дорівнює ID авторизованого користувача;
- сформований масив користувачів повертається у JSON-форматі з HTTP-статусом "200 OK".

```
export const getUsersForSidebar = async (req, res) => {
    try {

        const loggedInUserId = req.user._id;

        const filteredUsers = await User.find({ _id: { $ne:
        loggedInUserId } }).select("-password");

        res.status(200).json({ users: filteredUsers });

    } catch (error) {
        console.error("Error in getUsersForSidebar: ", error.
        message);
        res.status(500).json({ message: error.message });
    }
};      You, 2 months ago • Initial commit
```

Рисунок 3.12 – Код контролеру user.controller.js

Файл protectRoute.js містить middleware функцію, що використовується для захисту приватних маршрутів, доступ до яких мають лише авторизовані користувачі. Вона перевіряє наявність та дійсність JWT-токена, що зберігається в cookie браузера користувача.

Логіка роботи:

- зчитується JWT-токен із cookie (req.cookies.jwt). Якщо токен відсутній, сервер повертає статус “401 Unauthorized”;
- токен перевіряється за допомогою методу jwt.verify, використовуючи секретний ключ process.env.JWT_SECRET. У разі помилки повертається повідомлення про недійсний токен;
- після успішної перевірки токена функція декодує ідентифікатор користувача та виконує пошук в базі даних.
- якщо користувача знайдено, його об’єкт додається до запиту req.user, що дозволяє іншим контролерам працювати з даними авторизованого користувача;
- якщо всі перевірки пройдені, викликається next(), і обробка переходить до наступного middleware або контролера.

```
const protectRoute = async (req, res, next) => {
    try {
        const token = req.cookies.jwt;

        if (!token) {
            return res.status(401).json({
                message: "You need to be logged in to access this
                route"
            });
        }

        const decoded = jwt.verify(token, process.env.JWT_SECRET);

        if (!decoded) {
            return res.status(401).json({
                message: "Invalid token"
            });
        }

        const user = await User.findById(decoded.userId).select
        ("-password");

        if (!user) {
            return res.status(404).json({
                message: "User not found"
            });
        }

        req.user = user;

        next();
    } catch (error) {
        console.log("Error in protectRoute middleware ", error.
        message);
        res.status(500).json({
            message: error.message
        });
    }
}
```

Рисунок 3.13 – protectRoute.js

Для підключення та взаємодії клієнтської частини програми з серверною було реалізовано хуки. Хуки (hooks) – це спеціальні функції, які створюються для обробки логіки роботи програми.

Файл useLogin.js – це React-хук, який реалізує логіку входу користувача в систему. Він відповідає за валідацію введених даних, відправку запиту на сервер для авторизації, обробку відповіді та оновлення глобального стану аутентифікації. Цей хук використовується в компоненті Login, який має інтерфейс введення даних для авторизації.

Основні функції:

- валідація вхідних даних, перевірка, чи заповнені поля username та password. Якщо одне з полів пусте, користувач отримує повідомлення помилки через toast (рисунок 3.15);

- відправка POST-запиту на API-ендпоінт /api/auth/login з логіном і паролем в тілі запиту;
- обробка відповіді. Якщо сервер повертає помилку, вона відображається через toast повідомлення. У разі успішної авторизації отримані дані зберігаються в localStorage під ключем “webchat-user” (рисунок 3.14), а також викликається функція setAuthUser для оновлення стану користувача в додатку;
- під чат запиту loading встановлюється у true, щоб під час обробки запиту показувався індикатор завантаження. Після завершення процесу – встановлюється false.

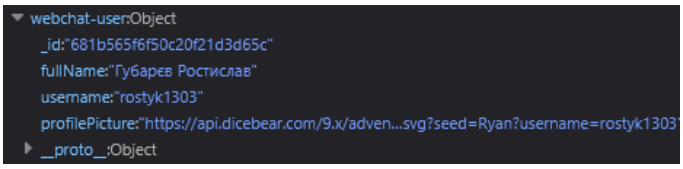

Рисунок 3.14 – збереження даних про авторизованого користувача в localStorage

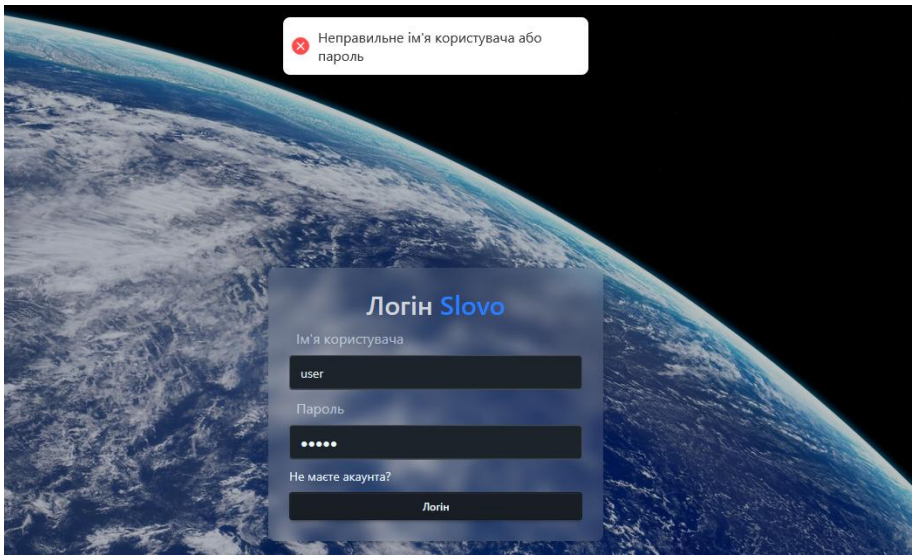

Рисунок 3.15 – toast-повідомлення

Хук useGetConversations відповідає за отримання списку чатів користувача. Цей хук використовується в компоненті Conversations, який дозволяє користувачу бачити всі доступні чати, при цьому забезпечуючи плавне завантаження та оновлення списку.

Основні функції:

- використовує useEffect для автоматичного запуску функції getConversations під час першого рендеру компонента. Ця функція відправляє GET-запит на API /api/users для отримання списку користувачів, з якими можна вести розмову (рисунок 3.16);
- якщо сервер повертає помилку, вона показується через toast.error. Якщо все успішно, список користувачів зберігаються в conversations;
- поки дані завантажуються, стан loading встановлений у true, після завершення loading стає false.

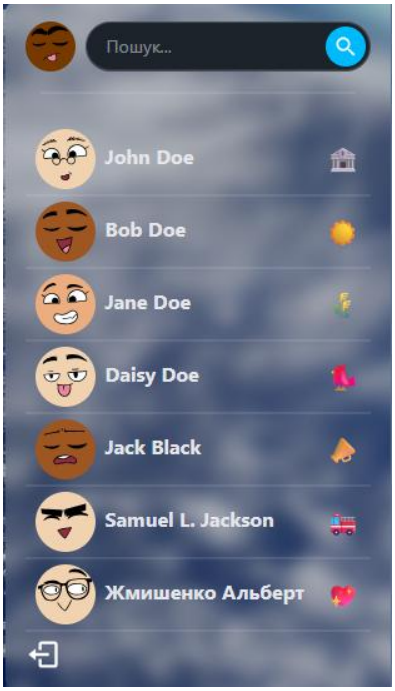

Рисунок 3.16 – Отриманий список користувачів

Хук useGetMessages відповідає за отримання повідомлень для обраної розмови в чаті. Він автоматично викликається кожного разу, коли користувач обирає чат, і завантажує пов'язані з ним повідомлення (рисунок 3.17). Цей хук використовується в компоненті Messages.

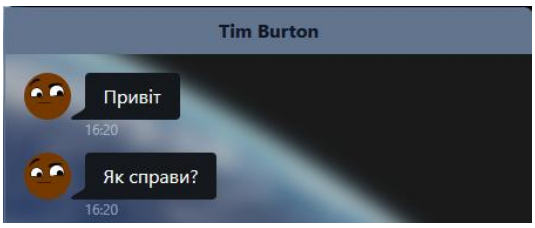

Рисунок 3.17 - Отримані повідомлення

Для надсилання повідомлення в поточний чат був створений хук useSendMessage. Він використовується в компоненті MessageInput (рисунок 3.18).

Основні функції:

- надсилає POST-запит на /api/messages/send/:conversationId, передаючи текст повідомлення в форматі JSON;
- якщо запит успішний, функція додає це повідомлення до списку повідомлень;
- якщо сталася помилка, користувач отримує текст помилки в toast-повідомленні.

Рисунок 3.18 – Компонент MessageInput, який використовується для відправки повідомлення

Для збереження та використання інформації про авторизованого користувача (наприклад, після входу або реєстрації) було створено AuthContext – контекст авторизації (рисунок 3.19).

Він використовується в кожному компоненті, де потребуються дані про авторизованого користувача.

Основні функції:

- зберігає стан користувача після перезавантаження сторінки;
- надає доступ до авторизованого користувача у будь-якому компоненті через useAuthContext;
- дозволяє оновлювати інформацію користувача через setAuthUser.

```
import { createContext, useContext, useState } from "react";

// eslint-disable-next-line react-refresh/only-export-components
export const AuthContext = createContext();

// eslint-disable-next-line react-refresh/only-export-components
export const useAuthContext = () => {
    return useContext(AuthContext);
};

export const AuthContextProvider = ({ children }) => {
  const [ authUser, setAuthUser ] = useState(
    JSON.parse(localStorage.getItem("webchat-user")) || null
  );

  return (
    <AuthContext.Provider value={{ authUser, setAuthUser }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Рисунок 3.19 – код файлу AuthContext

3.3.2 Розробка моделі аналізу тексту на токсичність

В цьому розділі описано повний цикл побудови моделі виявлення токсичності в текстах українською мовою.

Розробка моделі починається з завантаження та попередньої обробки даних. Було завантажено 2 датасети українських токсичних та нейтральних текстів в 3 файлах. Всі дані було об’єднано в один датафрейм. Пропущені значення видалились, а стовпець text очищається за допомогою функції clean_text. В тексті всі символи були переведені в нижній регістр, прибрались розділові знаки, цифри та стоп-слова (типу «і», «на», «це» тощо) та виконалась токенизація (розбиття тексту на слова).

Далі дані було розділено на навчальну та тестову вибірки у співвідношенні 80/20.

Для перетворення тексту в послідовності чисел використовується Tokenizer з лімітом у 20000 слів. Текстові дані конвертуються у числові послідовності та доповнюються до фіксованої довжини (100 слів).

Сама модель є послідовною нейронною мережею, яка включає:

- Embedding шар, який перетворює індекси слів у векторні представлення;
- Bidirectional(LSTM) – це двосторонній рекурентний шар LSTM, що аналізує текст у двох напрямках;
- два Dropout-шари видаляють частину нейронів для боротьби з перенавчаннями;
- фінальний Dense(1) з функцією активації sigmoid дозволяє здійснювати бінарну класифікацію (токсичне/нетоксичне).

Модель компілюється з функцією втрат binary_crossentropy і оптимізатором adam. Навчання відбувається протягом максимум 5 епох з використанням ранньої зупинки EarlyStopping, який зупиняє тренування, якщо валідаційна втрата не покращується протягом 2 епох.

Після навчання модель оцінюється на тестовій виборці, точність виводиться у відсотках.

Модель зберігається в форматі .keras, а токенізатор - у форматі .plk за допомогою бібліотеки pickle.

Для перевірки моделі реалізовано функцію predict_toxicity, яка приймає введений користувачем текст, очищає його, перетворює у послідовність чисел, подає модель для передбачення та в кінці повертає результат – «токсичне» чи «нетоксичне».

```
# Побудова моделі
model = Sequential()
model.add(Embedding(input_dim=20000, output_dim=128, input_length=max_length))
model.add(Bidirectional(LSTM(64, return_sequences=False)))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

Рисунок 3.10 – Параметри моделі

3.4 Методика роботи користувача

В цьому розділі описано методику роботи користувача в додатку.

При заході на сайт, якщо користувач не авторизований, його перенаправить на сторінку логіну, де буде запропоновано ввести дані для входу в акаунт (рисунок 3.21).

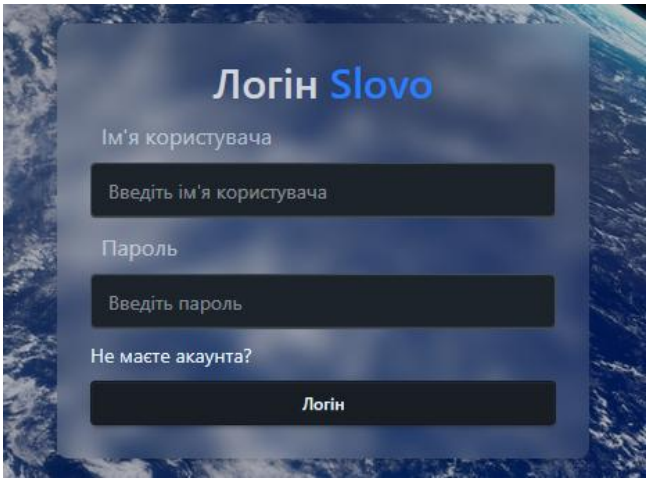

Рисунок 3.11 – Сторінка логіну

Якщо користувач не зареєстрований, він може натиснути на напис «Не маєте акаунта?». Після натискання відкриється сторінка реєстрації (рисунок 3.22). На цій сторінці користувач повинен ввести ПІБ, ім’я користувача, пароль та обрати стать. Якщо пароль буде містити менше ніж 6 символів, або паролі не будуть співпадати, то користувачу виведе помилку.

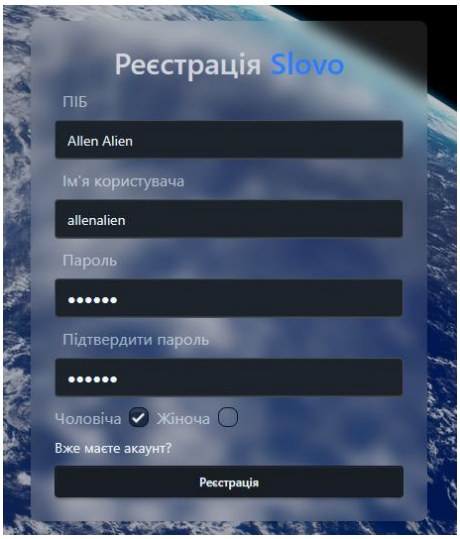

Рисунок 3.12 – Процес реєстрації

При успішній реєстрації користувача автоматично авторизує в систему. Перед ним з’явиться головна сторінка додатку (рисунок 3.23). В лівій частині додатку розташовується список чатів та пошук користувача. Поки юзер не обрав чат для спілкування, в правій частині додатку буде вікно привітання.

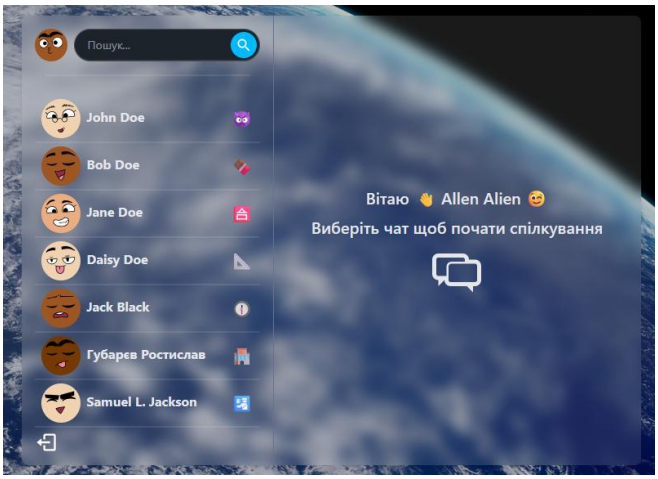

Рисунок 3.13 – Головна сторінка додатку

При натисканні на користувача в списку, в правій частині додатку відобразиться чат з ним. Також при введенні імені користувача в пошуку додаток автоматично відкриє чат з ним.

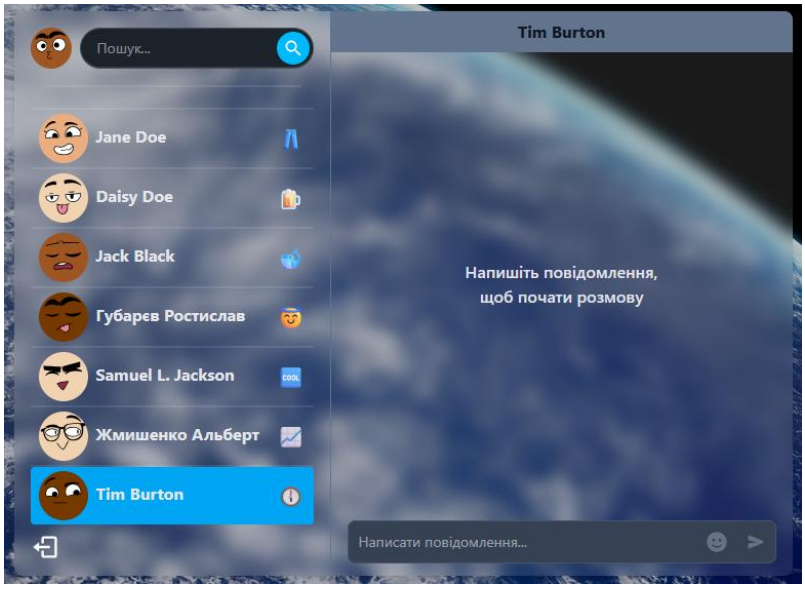

Рисунок 3.14 – Чат з користувачем

Якщо повідомлень між користувачами ще немає, юзеру буде запропоновано написати перше повідомлення. Для написання повідомлення користувачу треба натиснути на поле для введення внизу чату (рисунок 3.25). Також, щоб вставити емодзі в повідомлення користувач може натиснути на іконку обличчя біла кнопки відправлення (рисунок 3.26).

Рисунок 3.15 – Поле для введення та відправки повідомлення

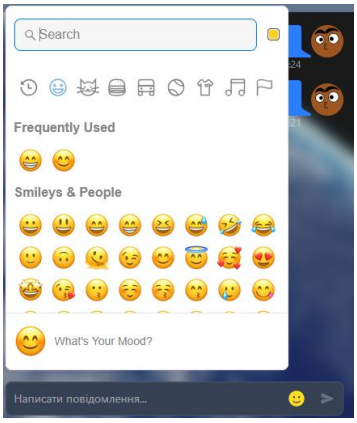

Рисунок 3.16 – Меню для вибору емодзі

Після відправки повідомлення воно миттєво з’явиться в чаті (рисунок 3.27).

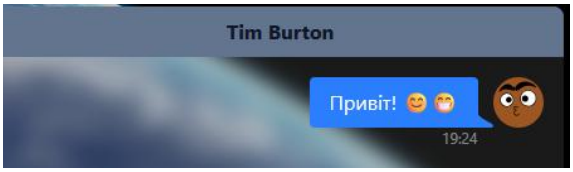

Рисунок 3.17 – Відправлене повідомлення

Також, якщо інтегрована модель розпізнає повідомлення як токсичне, то воно буде зацензурене.

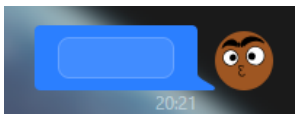

Рисунок 3.18 – Цензуроване повідомлення

В лівій нижній частині додатку є кнопка виходу з акаунту, у випадку, якщо користувач захоче вийти з системи.

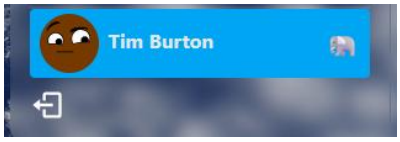

Рисунок 3.19 – Кнопка виходу з системи

З додаткових функцій, зліва вгорі сторінки є кнопка зміни фону. При натисканні з’явиться випадаюче меню, де користувач може обрати фон з запропонованих.

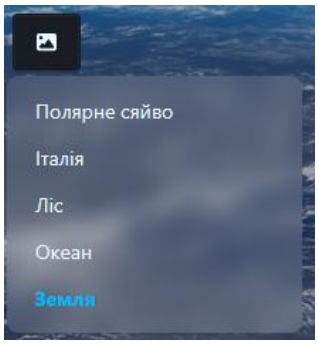

Рисунок 3.20 – Меню вибору фону додатку

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено повноцінний вебчат з відправкою та отриманням повідомлень в режимі реального часу, який об’єднує сучасні вебтехнології з інтелектуальними можливостями аналізу тексту на основі машинного навчання.

Для розробки вебчату було використано стек MERN (MongoDB, Express.js, React, Node.js), що забезпечує зручний інтуїтивно зрозумілий інтерфейс обміну повідомленнями в реальному часі з підтримкою Socket.io.

В додаток було інтегровано нейромережеву модель для визначення токсичності тексту, натреновану на українському наборі даних. Для розробки моделі було реалізовано автоматичну обробку тексту: очищення, видалення стоп-слів, токенизацію та перетворення в числові послідовності. Система вебчату в реальному часі реагує на небажані слова, та цензурить токсичні повідомлення.

Вебчат має великі перспективи подальшого розвитку. В сам додаток можна додати адаптацію під мобільні пристрої. Також відправлення медіа контенту, тобто відео та фото, та інтегрувати голосові повідомлення з можливістю транскрипції та подальшого аналізу тексту.

Для розвитку моделі в майбутньому можна додати можливість визначення типів токсичності (наприклад, образи, дискримінація, погрози), або перейти до трансформерних моделей (типу BERT), які краще враховують контекст та мають вищу точність у розумінні української мови.

Таким чином, результати цієї роботи демонструють, як за допомогою сучасних вебтехнологій та методів штучного інтелекту можна створити інтелектуальну платформу спілкування, що робить комунікацію більш безпечною та зручною.

ПЕРЕЛІК ПОСИЛАНЬ

1. Attention Is All You Need / A. Vaswani та ін. – 2017. – Режим доступу: https://arxiv.org/abs/1706.03762.

2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Електронний ресурс] / J. Devlin та ін. – 2019. – Режим доступу: https://aclanthology.org/N19-1423/.

3. Hugging Face. Transformers Documentation [Електронний ресурс]. Режим доступу: https://huggingface.co/docs/transformers/index.

4. Studlife. Побудова функціональної моделі системи [Електронний ресурс]. Режим доступу: https://studfile.net/preview/381095/page:14/

5. Guru99. Діаграма діяльності в UML: символ, компоненти та приклад [Електронний ресурс]. Режим доступу: https://www.guru99.com/uk/uml-activity-diagram.html.

6. Dou. Розуміння Клієнт-Серверної Архітектури на прикладах [Електронний ресурс]. Режим доступу: https://dou.ua/forums/topic/44636/.

7. Hugging Face. multilingual_toxicity_dataset [Електронний ресурс]. Режим доступу: https://huggingface.co/datasets/textdetox/multilingual_toxicity_dataset/viewer/default/ .

8. InfoWorld. React: Making faster, smoother UIs for data-driven Web apps [Електронний ресурс]. Режим доступу: https://www.infoworld.com/article/2179365/react-making-faster-smoother-uis-for-data-driven-web-apps.html .

9. DreamHost. Ваше повне керівництво по Tailwind CSS [Електронний ресурс]. Режим доступу: https://www.dreamhost.com/blog/uk/tailwind-css/ .

10. Guru99. Що таке MongoDB? Вступ, Архітектура, функції та приклад [Електронний ресурс]. Режим доступу: https://www.guru99.com/uk/what-is-mongodb.html .

Додано примітку [H5]: На вікіпедію не треба посилатись. У статті вікіпедії в кінці є посилання на першоджерела, візьми звідти

ДОДАТКИ