Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту

Виконав: студент 2 курсу, групи 3АКІТ-22М
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології
Михайло ЯКИМЧУК

Керівник: к.т.н., доцент кафедри АІІТ
Володимир КОЦЮБИНСЬКИЙ
«4» грудня 2023 р.

Опонент: доцент. каф. АІІТ
Микола БИКОВ
«8» грудня 2023 р.

Допущено до захисту

Зав. кафедри АІІТ
д.т.н проф.
Олег БІСІКАЛО
«11» грудня 2023 р.

Вінниця ВНТУ – 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти II-й (магістерський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня - програма – Інформаційні системи і Інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н проф. Олег БІСІКАЛО
"20" вересня 2023 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Якимчуку Михайлу Михайловичу
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту»

Керівник роботи: к.т.н., доцент кафедри АІІТ Коцюбинський В. Ю.

затверджені наказом ВНТУ від "18" вересня 2023р №247

2. Термін подання студентом роботи "05" грудня 2023р

3. Вихідні дані до магістерської кваліфікаційної роботи максимальний рівень шуму у приміщенні – 70 дБ; підтримка ОС – Windows; максимальний час завантаження – 5 с; авторизація користувачів – так; максимальна кількість запитів до системи – до 1000 шт/с; мови графічного та голосового інтерфейсів – українська, англійська, російська; достовірність розпізнавання мови – 95%; використання кодування під час передачі даних – так.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) вступ, історія та розвиток месенджерів, аналіз клієнт-серверної архітектури, аналіз концепції автоматичної фільтрації контенту, огляд існуючих рішень, загальні вимоги до програмного забезпечення, визначення загальних вимог до архітектури системи, характеристики оцінки успішності архітектури продукту, вибір оптимальних технологій для розроблення проекту, вибір СКБД для реалізації проекту, вибір середовища розробки проекту, розроблення модулю фільтрації контенту з використанням машинного навчання, загальний огляд функціональних модулів системи, моделювання та реалізація основних модулів функціонування, розробка структури системи, інструкція розгортання додатку для розробників, інструкція користувачеві.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) діаграма класів системи, UML-діаграма діяльності, UML-діаграма варіантів використання автоматизованої системи, вигляд екрану розробленого web-додатку – 8 шт.

6. Консультанти розділів роботи

Розділ змістової частини роботи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н., доцент кафедри АІІТ Володимир КОЦЮБИНСЬКИЙ	20.09.2023	05.12.2023

7. Дата видачі завдання "21" вересня 2023р

Календарний план

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз області, об'єкту та предмету досліджень	21.09.2023р.	виконав
2	Архітектура системи що розробляється	28.09.2023р.	виконав
3	Проектування модулів функціонування веб-системи	05.10.2023р.	виконав
4	Розроблення інструкцій для програмного забезпечення	12.10.2023р.	виконав
5	Апробація результатів дослідження	21.10.2023р.	виконав
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	29.10.2023р.	виконав
7	Графічні матеріали: Use-case UML-діаграма UML-діаграма класів UML-діграма діяльності	05.11.2023р. 14.11.2023р. 20.11.2023р.	виконав
8	Захист МКР	18.12. 2023р.	виконав

Студент Михайло ЯКИМЧУК

Керівник роботи Володимир КОЦЮБИНСЬКИЙ

АНОТАЦІЯ

УДК 004.738.5:004.9

Якимчук М.М. Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту. Магістерська кваліфікаційна робота зі спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології, освітня програма – Інформаційні системи і Інтернет речей. Вінниця: ВНТУ, 2023. ___ с.

На укр. мові. Бібліогр.: ___ назв; рисунків.: ____;.

У магістерській роботі розглянуто розробку клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту. В ході дослідження об'єкта вивчені основні аспекти функціонування клієнт-серверних веб-месенджерів, досліджено взаємодію модуля фільтрації даних з іншими модулями чат-системи. Проведено порівняльний аналіз існуючих веб-месенджерів. Розроблено технічні та програмні засоби системи, включаючи UML-діаграми та функціональні можливості для користувачів. Результати тестування свідчать про правильність роботи системи та зручність її використання.

Ключові слова: клієнт-серверна архітектура, веб-месенджери, фільтрація контенту, машинне навчання, нейронні мережі.

ANNOTATION

Yakymchuk M.M. Development of the client-server system based on usage of a WEB messenger with automatic content filtering. Master's thesis on specialty 151 - Automation and computer-integrated technologies, educational program - Information systems and the Internet of Things. Vinnytsia: VNTU, 2023. 105 p.

In Ukrainian speech Bibliography: ____ titles; Fig.: ___.

Development of the client-server system based on usage of a WEB messenger with automatic content filtering is considered in the master's work. During the study of the object, the main aspects of the functioning of client-server web messengers were studied, the interaction of the data filtering module with other modules of the chat system was studied. A comparative analysis of existing web messengers was carried out. Technical and software tools of the system, including UML diagrams and functionality for users were developed. The test results indicate the correct operation of the system and its ease of use.

Keywords: client-server architecture, web messengers, content filtering, machine learning, neural networks.

ЗМІСТ

ВСТУП

Актуальність магістерської кваліфікаційної роботи пов’язана із зростанням популярності веб-месенджерів, які стали необхідною складовою життя користувачів Інтернету. Вони використовуються для спілкування, обміну інформацією та бізнесу, але це також призводить до загроз безпеці та приватності, таких як небажаний контент і спам. Застосування методів машинного навчання для автоматичної фільтрації контенту може вирішити ці проблеми.

Мета роботи – створення клієнт-серверної веб-системи за принципом веб-месенджеру, який би забезпечував автоматичну фільтрацію контенту з використанням нейро-мережових технологій.

Задачі досліджень магістерської кваліфікаційної роботи:

- Огляд основних принципів функціонування веб-месенджерів;
- Дослідження наявних реалізацій веб-месенджерів;
- Дослідження існуючих рішень фільтрації контенту;
- Аналіз технологій та підходів для створення клієнт-серверної інфраструктури веб-месенджера;
- Розробка алгоритмічного та програмного забезпечення клієнт-серверної частини веб-месенджера;
- Тестування розробленого програмного забезпечення.

Об’єктом дослідження є процеси передавання, оброблення та фільтрації інформації, які реалізуються системою, побудованою за принципом веб-месенджера.

Предметом дослідження є структура комунікації між користувачами, засобами веб-системи, що базується на принципах роботи веб-месенджеру з підтримкою фільтрації контенту.

Методи досліджень: у роботі використано методи ідентифікації, збору і обробки даних, які отримуються від користувачів, та використання цих даних для покращення взаємодії між користувачами та підвищення достовірності інформації, яка публікується.

Новизна отриманих результатів дослідження полягає в удосконаленні моделей фільтрації контенту з використанням штучного інтелекту та технології нейронних мереж.

Практичне значення одержаних результатів полягає в тому, що в роботі було удосконалено методи передавання, оброблення та фільтрації інформації, які реалізуються системою, побудованою за принципом веб-месенджеру.

Апробація результатів роботи. Дана робота була апробована на науково-технічній конференції “XLIX Науково-технічна конференція факультету комп'ютерних систем і автоматики (2023)” Вінницького національного технічного університету у 2023 році.

Публікації. Результати теоретичних і експериментальних досліджень викладені у тезах доповіді [1].

1 АНАЛІЗ ОБЛАСТІ, ОБ'ЄКТУ ТА ПРЕДМЕТУ ДОСЛІДЖЕНЬ

1.1 Історія та розвиток месенджерів

З розвитком засобів комунікації, можна відзначити важливі інновації та тенденції. У сучасний період інтернет-спілкування стало важливою складовою повсякденного життя. Чат-додатки не лише забезпечують швидке обмін інформацією, але і виконують різноманітні функції, від роботи в команді до віртуальних зустрічей та розваг.

Поява додатків, таких як Telegram та Signal, з фокусом на конфіденційності та шифруванні, відображає зростання обізнаності щодо приватності в онлайн-спільноті. Користувачі стають більш освіченими та обережними у виборі платформ для своїх комунікаційних потреб.

Поява інтеграції штучного інтелекту у чат-додатках, які пропонують автоматизовані відповіді та персоналізовані поради, також свідчить про поступовий розвиток технологій у цьому сегменті. Це сприяє ефективнішому використанню часу та ресурсів під час комунікаційних процесів [2].

Зростання популярності віртуальних асистентів та голосового інтерфейсу також відображає сучасні тенденції в комунікації. Користувачі можуть використовувати голосові команди для виконання різних завдань, відправлення повідомлень до контролю додатків та послуг.

Усе це свідчить про постійний розвиток сфери комунікаційних технологій, який віддзеркалює сучасні потреби та вимоги користувачів у великому світі онлайн-спілкування.

Наприкінці 2023 року можна визначити ще кілька важливих тенденцій у світі чат-додатків. Однією з них є зростання популярності мультимедійних та інтерактивних елементів у чат-платформах. Користувачі все більше цінують можливість обміну не лише текстовими повідомленнями, а й зображеннями, відео та аудіозаписами, роблячи спілкування більш насиченим та емоційним.

Також важливою стає забезпечення високого рівня кібербезпеки та захисту приватності в чат-системах, враховуючи зростання кількості онлайн-загроз та ризиків порушення конфіденційності даних користувачів.

Наприкінці року важливим елементом стає інтеграція штучного інтелекту для покращення користувацького досвіду. Алгоритми машинного навчання в чат-додатках можуть аналізувати поведінку користувачів, надавати персоналізовані рекомендації та автоматизовані функції для полегшення взаємодії.

В цілому, чат-додатки залишаються динамічним інструментом комунікації, який продовжує еволюціонувати, пристосовуючись до змін потреб користувачів та передових технологічних трендів [3].

Зазначається також зростання популярності "чат-ботів", які використовують штучний інтелект для автоматизованого взаємодії з користувачами. Вони можуть виконувати різноманітні завдання, від відповіді на питання до надання інформації та виконання операцій безпосередньо в чаті.

Цей тренд у розвитку технологій чат-додатків відображає загальний рух до створення більш залучальних та іммерсивних взаємодій між користувачами. Здатність використовувати віртуальні та розширені реальності дозволяє створювати унікальні, інтерактивні досвіди для користувачів, які виходять за межі простого текстового обміну.

Ігрові елементи в чат-додатках надають можливість користувачам не тільки обговорювати події, але і долучати до свого спілкування веселі та захопливі елементи. Це сприяє збільшенню часу, проведеного в додатках, і залучає аудиторію різного віку та інтересів.

Чат-боти з штучним інтелектом також грають важливу роль у розвитку чат-додатків. Вони дозволяють автоматизувати багато рутинних завдань, забезпечуючи ефективний та оперативний обмін інформацією між користувачем і додатком. Це може включати в себе також фільтрацію небажаного контенту.

Розвиток технологій в чат-додатках також стимулює зростання колективної та соціальної активності. Завдяки ігровим елементам та віртуальним

просторам, користувачі можуть спільно розважатися та взаємодіяти в цифровому середовищі. Це може бути особливо важливим у контексті віддаленої роботи та навчання, де важко забезпечити відчуття комунікації та спільності.

Зокрема, віртуальні та розширені реальності відкривають нові горизонти для навчання, торгівлі, та інших сфер. Можливість взаємодії з інформацією у вигляді 3D-моделей чи віртуальних об'єктів може поліпшити процес розуміння складних концепцій та покращити засвоєння нового матеріалу.

У кінці кінців, інновації в галузі чат-додатків допомагають зробити взаємодію в інтернеті більш захоплюючою, зручною та особистою. Ці тенденції об'єднують технології та людей, створюючи новий рівень комунікації та спілкування в цифровому світі.

Також варто відзначити роль розумних асистентів у чат-додатках, які, базуючись на штучному інтелекті, можуть аналізувати інформацію з чатів та надавати корисні рекомендації [4].

В цілому, еволюція чат-додатків чат-ботів відображає постійний розвиток технологій та зміну в способах взаємодії людей в цифровому просторі.

Зараз деякі чат-додатки також експериментують із забезпеченням можливості користувачам створювати та вдосконалювати свої власні розумні асистенти, що дозволяє персоналізувати їхню функціональність та адаптувати їх до індивідуальних потреб.

1.2 Аналіз клієнт-серверної архітектури

Клієнт-серверна система - це архітектурний підхід у розробці програмного забезпечення, де функції розподіляються між клієнтами і серверами. Клієнти та сервери взаємодіють через мережу, таку як Інтернет або локальну мережу. Клієнти надають інтерфейс для користувача і відправляють запити, а сервери відповідають на ці запити, надаючи потрібні ресурси чи послуги. Цей підхід

сприяє розділенню завдань, полегшує роботу системи і дозволяє використовувати різні платформи для клієнтів і серверів.

Крім того, ця архітектура полегшує обслуговування та оновлення системи, оскільки зміни можна вносити на рівні клієнтів або серверів без значного впливу на іншу сторону.

В клієнт-серверних системах клієнти і сервери можуть виконуватися на різних пристроях або комп'ютерах, що дозволяє розділити завдання обробки даних і обчислень між різними ресурсами. Це полегшує розподілення ресурсів та забезпечує більш ефективне використання обчислювальної потужності[5].

Також важливо відзначити, що клієнтські програми і сервери можуть використовувати різні мови програмування та операційні системи, що дозволяє їм взаємодіяти, незважаючи на їхню різноманітність. Ця гнучкість у виборі технологій є важливою перевагою клієнт-серверних архітектур.

Застосування клієнт-серверних систем розповсюджене в багатьох галузях, таких як веб-розробка, бази даних, обмін повідомленнями, ігри та багато інших. Ця архітектура дозволяє створювати розширювані та ефективні системи, які можуть відповідати різноманітним потребам користувачів.

Крім того, клієнт-серверна архітектура дозволяє покращити безпеку системи. Сервер може контролювати доступ до ресурсів і визначати права користувачів, забезпечуючи централізований управління безпекою даних і функціональністю. Це важливо в областях, де важлива конфіденційність даних та запобігання несанкціонованому доступу.

Крім того, засновані на клієнт-серверній архітектурі додатки можуть бути оновлені централізовано, що спрощує управління версіями та розгортанням нового функціоналу. Це робить такі системи більш гнучкими і зручними для адміністрування.

На рисунок 1.1 показано загальну структуру клієнт-серверної архітектури.

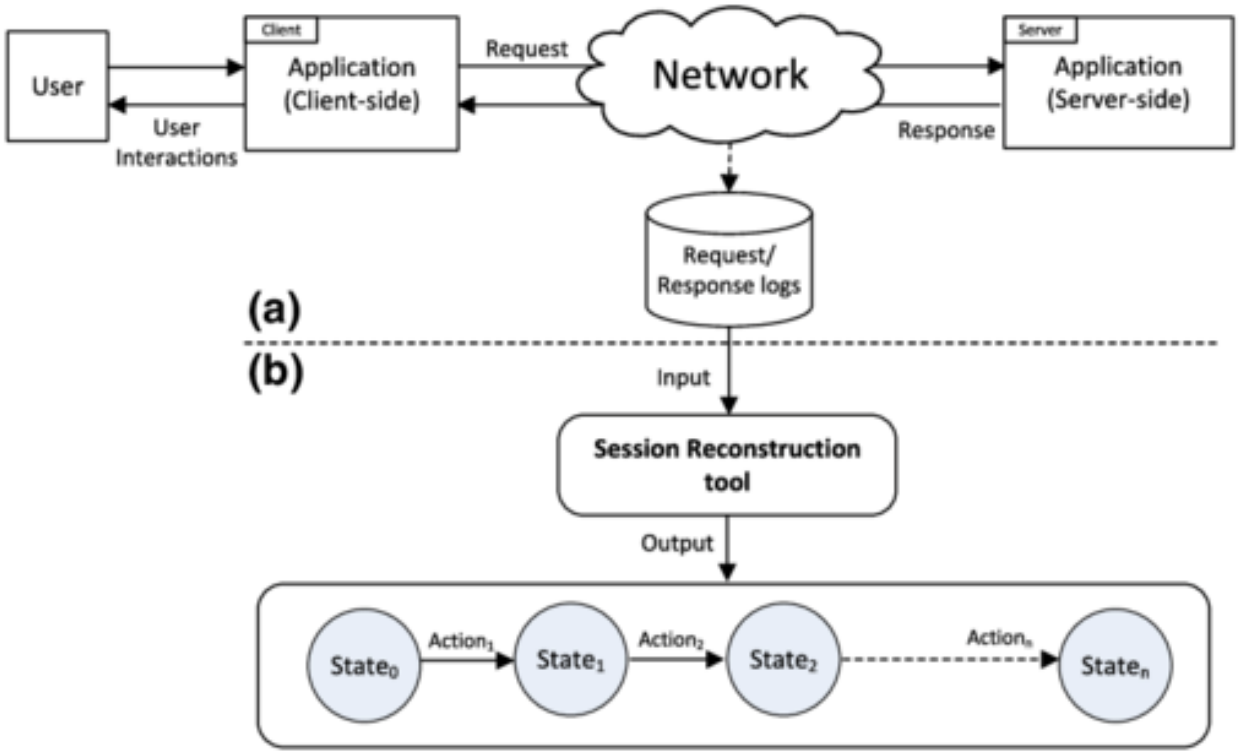

Рисунок 1.1 – Діаграма структури клієнт-серверної архітектури.

Необхідно також відзначити, що розвиток технологій робить клієнт-серверні системи все більш складними та потужними. Наприклад, з розвитком хмарних технологій, сервери можуть надавати послуги через інтернет, що робить доступ до ресурсів ще більш глобальним і мобільним.

Однією з ключових переваг клієнт-серверних систем є можливість розподілу завдань і обов'язків між клієнтами і серверами в залежності від їхніх можливостей та функціональності. Це розділення дозволяє краще використовувати ресурси, сприяє ефективності системи і дозволяє масштабувати інфраструктуру в залежності від зростання обсягів роботи[6].

Важливим аспектом є також можливість використання різних типів клієнтів (наприклад, веб-браузери, мобільні додатки, десктопні програми) для взаємодії з серверами. Це робить клієнт-серверні системи універсальними і здатними працювати на різноманітних пристроях та платформах.

Крім того, застосування клієнт-серверних архітектур стає дедалі актуальнішим у світі Інтернету речей (IoT), де багато пристроїв діють як клієнти,

взаємодіючи з хмарними серверами для обробки та аналізу великих обсягів даних.

В клієнт-серверній архітектурі для чат-додатків, клієнти і сервери обмінюються інформацією за допомогою мережі, такої як Інтернет. Клієнти можуть бути реалізовані на різних платформах, таких як веб-браузери, мобільні додатки або десктопні програми. Сервер, у свою чергу, може бути централізованим, розташованим на віддаленому сервері, і обробляти запити від багатьох клієнтів одночасно.

Типовий сценарій взаємодії включає в себе те, що клієнти відправляють запити на сервер для відправки повідомлень, отримання нових повідомлень, а також для виконання різних операцій, таких як створення групових чатів чи видалення повідомлень. Сервер обробляє ці запити, зберігає дані чату, ініціює розсилання повідомлень між клієнтами та виконує інші завдання, необхідні для правильної роботи системи [7].

Ця архітектура дозволяє забезпечити ефективний обмін даними та ресурсами між клієнтами та сервером, забезпечуючи при цьому централізований контроль над логікою додатка і забезпечуючи однорідний інтерфейс для користувачів незалежно від платформи.

1.3 Аналіз концепції автоматичної фільтрації контенту

Концепція автоматичної фільтрації контенту - це процес використання різноманітних алгоритмів та технік для визначення, класифікації та обробки вмісту, який обробляється на веб-сайтах, платформах соціальних мереж, форумах чи інших онлайн-ресурсах. Головною метою такої фільтрації є контроль над розповсюдженням небажаного або шкідливого контенту, такого як спам, образливі матеріали, фейкові новини, порнографія та інші вміст, який може порушувати правила використання платформи або навіть закон [8].

Аналіз концепції автоматичної фільтрації контенту включає розгляд принципів та методів, які використовуються для розпізнавання та обробки вмісту. Цей аналіз може включати в себе розгляд алгоритмів машинного навчання, які навчаються розпізнавати певні категорії контенту, інтеграцію евристичних правил для виявлення підозрілих патернів, а також використання технологій штучного інтелекту для покращення точності фільтрації.

Одним із ключових викликів у цьому процесі є забезпечення ефективності та точності фільтрації, а також уникнення неправомірних втручань у свободу вираження. Важливо збалансувати систему так, щоб вона була достатньо чутливою до розпізнавання небажаного контенту[9].

Також, концепція включає в себе постійне оновлення та адаптацію фільтрів до змін в структурі та характері контенту в Інтернеті, оскільки нові види шкідливого вмісту можуть з'являтися, і необхідно постійно вдосконалювати систему для їх ефективного виявлення.

На рисунку 1.2 показано діаграму процесу фільтрації тексту.

Рисунок 1.2 – Діаграма процесу фільтрації тексту.

Автоматична фільтрація контенту є важливим аспектом для забезпечення безпеки та комфорту користувачів в онлайн-середовищі. Зазвичай цей процес включає в себе аналіз тексту, зображень, відео та інших медійних форматів з метою визначення його відповідності встановленим стандартам та правилам. Методи фільтрації можуть використовувати ключові слова, семантичний аналіз, машинне навчання та інші техніки для ефективного виявлення небажаного вмісту.

Проте, існує велика складність у розпізнаванні контенту в контексті, оскільки суттєвий вплив має його семантика, мовний вираз, а також культурні

відмінності. Також, алгоритми фільтрації можуть стикатися з проблемою "фальшивих позитивів" та "фальшивих негативів", коли вони помилково класифікують безшкідливий контент як небажаний або навпаки.

Забезпечення прозорості та можливостей ручного втручання для користувачів є ще одним важливим аспектом в розробці систем автоматичної фільтрації. Це дозволяє виправляти помилки, які можуть виникнути в результаті роботи алгоритмів, і надає користувачам можливість керувати тим, який контент вони бажають бачити чи приховати [10].

У світі постійно зростає обумовлена технологічними та культурними змінами складність завдань автоматичної фільтрації контенту. Вирішення цих викликів вимагає поєднання різних підходів, включаючи технічні інновації, правові регуляції та участь спільноти в процесі розробки та вдосконалення фільтраційних систем.

1.4 Огляд існуючих рішень

Facebook Messenger for Web - це веб-версія популярного месенджера Facebook, яка дозволяє користувачам обмінюватися повідомленнями, здійснювати аудіо та відео-виклики, а також використовувати різноманітні функції, доступні в веб-версії. Веб-версія синхронізується з мобільним додатком, надаючи зручний спосіб спілкування через веб-браузер, без необхідності встановлення додаткових програм.

На рисунку 1.3 показано інтерфейс Facebook Messenger for Web.

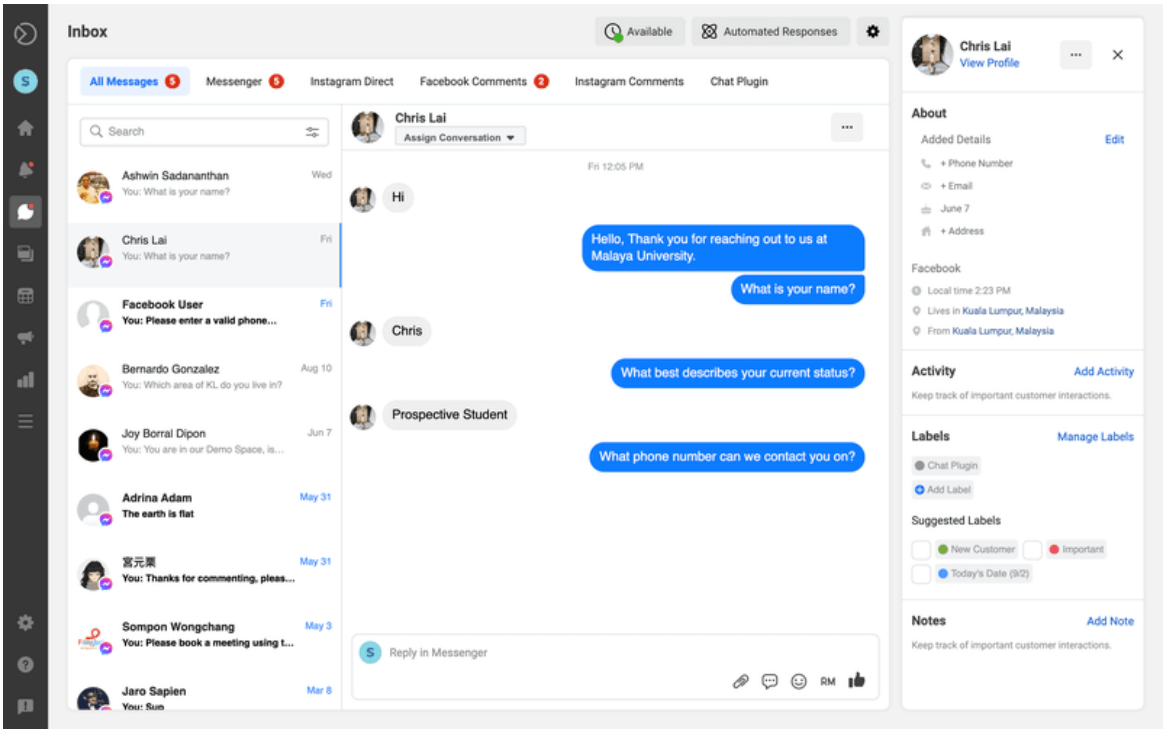

Рисунок 1.3 – Інтерфейс «Facebook Messenger for Web»

Facebook Messenger for Web використовує веб-технології, такі як HTML, CSS, та JavaScript для створення інтерфейсу користувача. Щодо комунікації між клієнтом і сервером, використовується технологія WebSocket для обміну повідомленнями в реальному часі. Клієнтська частина використовує браузер користувача для відображення інтерфейсу, тоді як серверна частина реалізована на серверах Facebook.

Плюси:

– зручний доступ до повідомлень через веб-браузер без необхідності встановлення додаткового програмного забезпечення;

– синхронізація з мобільним додатком дозволяє зручно перемикатися між пристроями;

– реалізація в реальному часі завдяки технології WebSocket надає швидку доставку повідомлень.

Мінуси:

– залежність від Інтернет-з'єднання для коректної роботи;

– деякі функції можуть бути обмежені у порівнянні з мобільною версією;

– приватність та безпека можуть бути проблемою в зв'язку з обробкою особистої інформації через веб-версію.

Telegram Web - це веб-версія популярного месенджера Telegram, яка дозволяє користувачам обмінюватися повідомленнями, мультимедійним вмістом та використовувати інші функції, доступні в мобільній версії. Додаток побудований з використанням таких технологій, як HTML, CSS, JavaScript та технології WebSocket для забезпечення обміну повідомленнями між клієнтом та сервером [11].

На рисунку 1.4 показано загальний вигляд інтерфейсу Telegram Web.

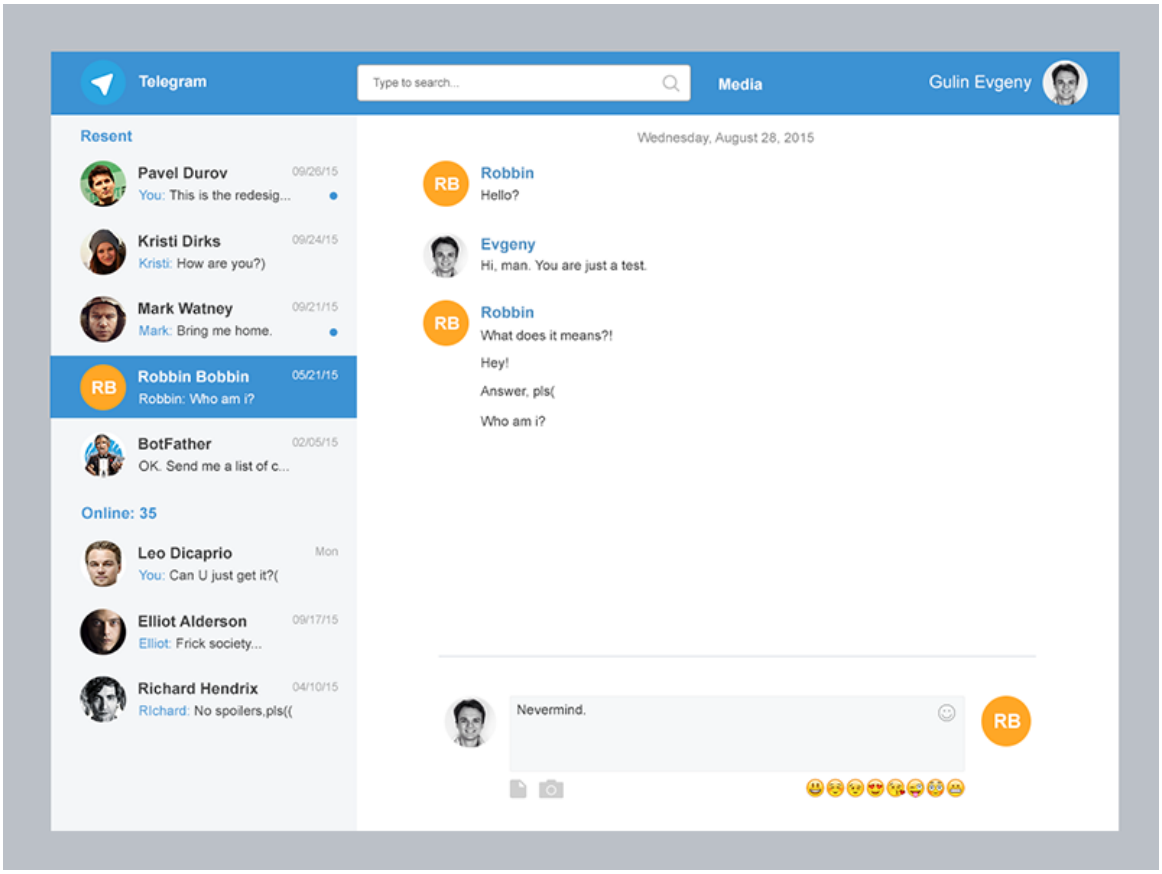

Рисунок 1.4 – Загальний вигляд інтерфейсу «Telegram Web».

Slack - це комунікаційний інструмент, спроектований для співпраці та обміну інформацією в робочих командах. Slack включає в себе клієнт-серверну систему, де користувачі можуть обмінюватися повідомленнями, файлами, створювати канали для різних проектів та вести обговорення.

На рисунку 1.5 показано загальний вигляд інтерфейсу Slack.

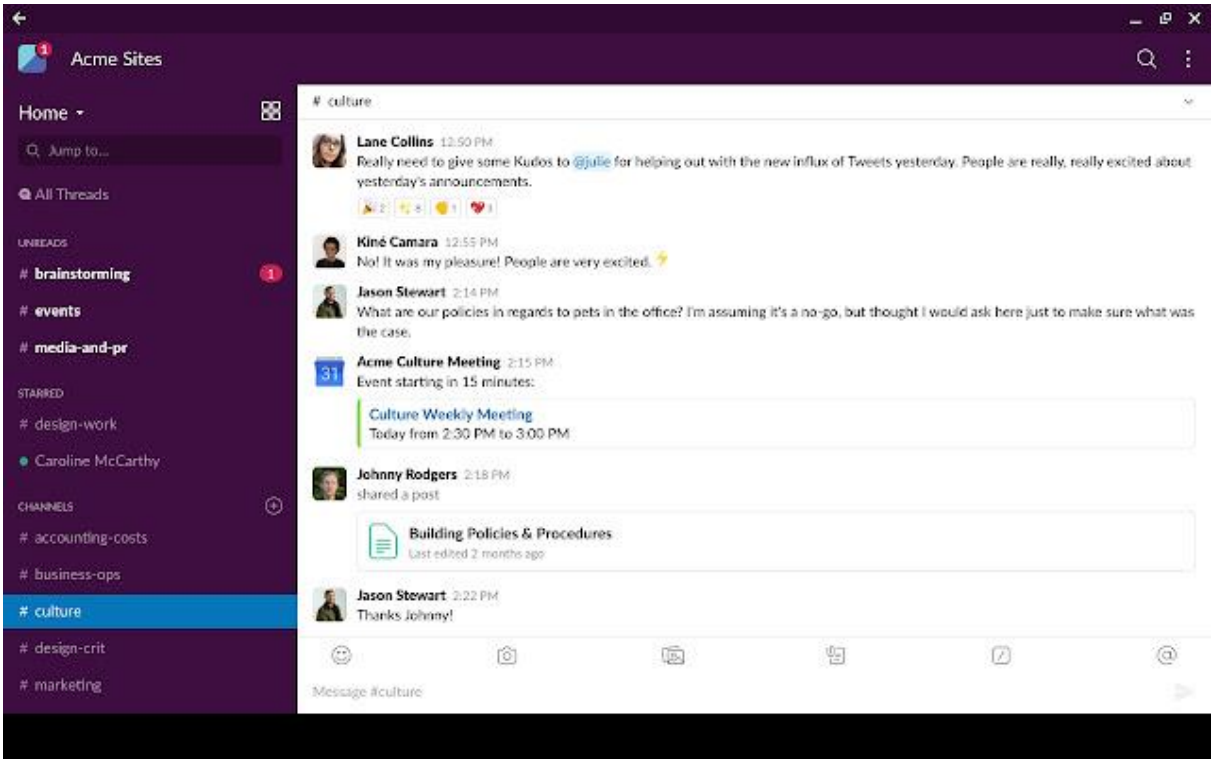

Рисунок 1.5 – Загальний вигляд інтерфейсу «Slack».

Slack використовує веб-технології, такі як HTML, CSS, та JavaScript для розробки клієнтської частини, а також має серверну інфраструктуру для забезпечення обміну даними та зберігання інформації.

Плюси:

– зручний інтерфейс користувача для легкої навігації та спілкування;

– можливість створення каналів для організації робочих обговорень та спільної роботи.

– обмін повідомленнями в реальному часі сприяє ефективній комунікації в робочому оточенні;

Мінуси:

– платна модель використання для розширених функцій та збільшення обсягу збереженої історії повідомлень;

– може вимагати часу для засвоєння та адаптації для нових користувачів.

Microsoft Teams - це корпоративний інструмент для комунікації та співпраці в робочому оточенні. Він використовує клієнт-серверну архітектуру з використанням різних технологій для забезпечення ефективної спільної роботи та обміну інформацією в командах.

На рисунку 1.6 показано інтерфейс програми Microsoft Teams.

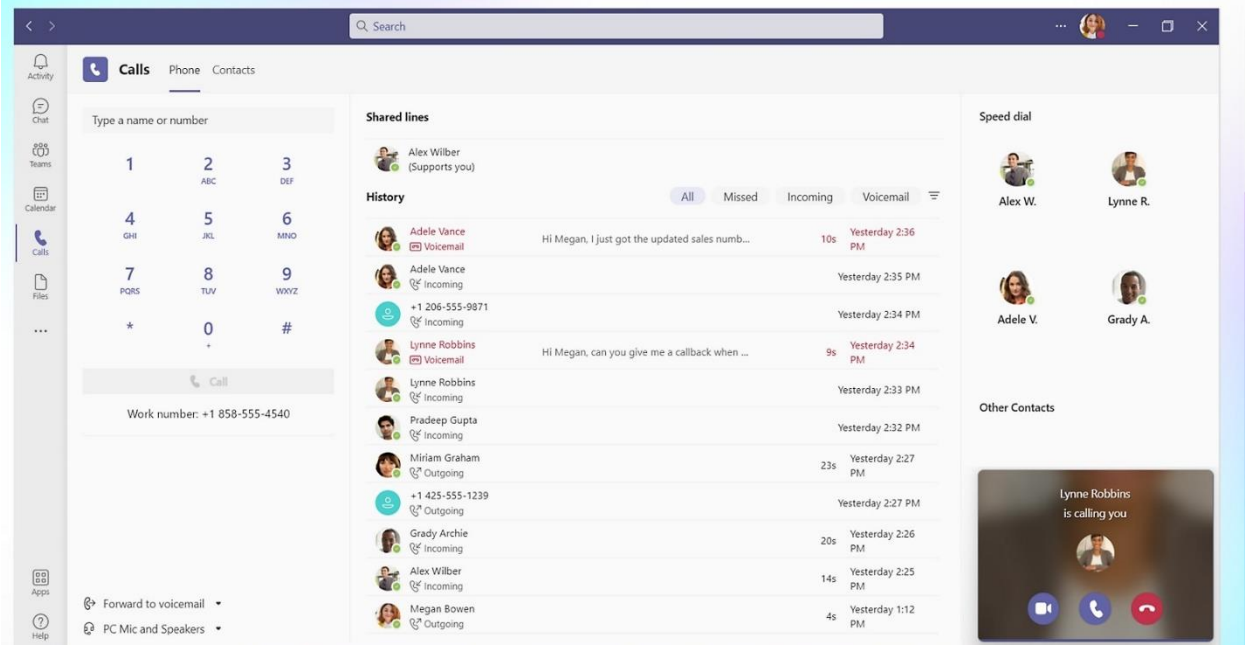

Рисунок 1.6 – «Microsoft Teams».

Microsoft Teams використовує веб-технології, а також має нативні додатки для різних платформ, таких як Windows, macOS, Android і iOS. Серверна інфраструктура Microsoft Teams побудована на хмарних технологіях Microsoft 365, що забезпечує обмін даними та зберігання інформації.

Плюси:

– інтеграція з іншими продуктами Microsoft, такими як Word, Excel, SharePoint, що полегшує спільну роботу над документами;

– можливість проведення відеоконференцій та онлайн-зустрічей прямо в інтерфейсі програми.

– розширені можливості для організації та керування завданнями та проектами;

Мінуси:

– великий функціонал може потребувати часу для освоєння новими користувачами;

– є платні плани, що може викликати обмеження для невеликих бізнесів або користувачів з обмеженим бюджетом.

Threema - це месенджер, який відомий своєю великою увагою до приватності та безпеки користувачів. Він використовує енд-ту-енд шифрування для захисту конфіденційності повідомлень. Особливість Threema - анонімна реєстрація без необхідності надавати особистий номер телефону чи ім'я. Додаток використовує QR-коди для перевірки ідентифікації. В Threema можна створювати групові чати та обмінюватися мультимедійним контентом. Його функціонал включає стікери, емодзі та можливість використання на смарт-годинниках.

Також додаток підтримує Threema Pay для надсилання грошей в чатах. Користувачі можуть керувати своєю приватністю, видаляючи повідомлення із чатів. Threema вигідно виділяється своїм підходом до захисту особистої інформації та забезпечення безпеки користувачів.

Інтерфейс месенджеру показано на рисунку 1.7.

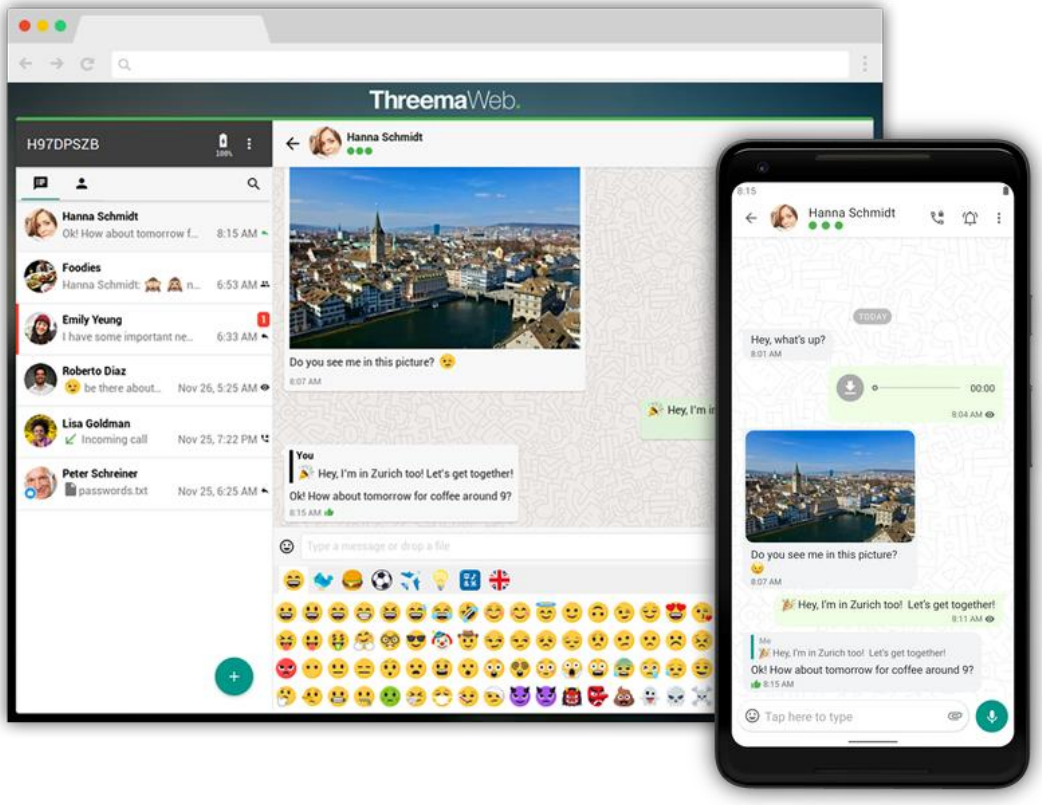

Рисунок 1.7 – Зовнішній вигляд месенджеру «Threema».

Threema визначається не лише високим рівнем безпеки, а й особливим підходом до анонімності користувачів. Можливість реєстрації без використання особистого номера телефону вказує на зобов'язання додатку зберігати конфіденційність і особисту інформацію своїх користувачів.

Також варто зазначити, що Threema пропонує платформи для безпечного обміну грошима через Threema Pay, додатково розширюючи можливості користувачів за межі звичайного обміну повідомленнями.

Цей месенджер, враховуючи свою фокусованість на приватності та безпеці, виявляється привабливим вибором для тих, хто ставить на перше місце захист своїх особистих даних в онлайн-спілкуванні.

Плюси Threema:

– сильне шифрування та анонімна реєстрація забезпечують високий рівень конфіденційності;

– можливість видалення повідомлень із чатів для більшого контролю;

– інтеграція з системою платежів у чатах.

Мінуси Threema:

– вартість використання може бути бар'єром для користувачів;

– менше користувачів порівняно з іншими популярними месенджерами;

– відсутність деяких функцій, що доступні в інших месенджерах.

WeChat - це китайський месенджер, який поєднує в собі функції миттєвого обміну повідомленнями, соціальної мережі та платіжної системи. Додаток відомий своєю універсальністю, забезпечуючи зручність спілкування, проведення фінансових операцій та взаємодії з іншими сервісами. WeChat Pay дозволяє користувачам здійснювати різні фінансові транзакції, що робить додаток популярним у Китаї. Однак слід враховувати, що WeChat піддається цензурі та контролю з боку китайської влади, що може впливати на приватність користувачів. Також, за межами Китаю, деякі функції можуть бути обмежені.

Графічний інтерфейс WeChat показана на рисунку 1.8.

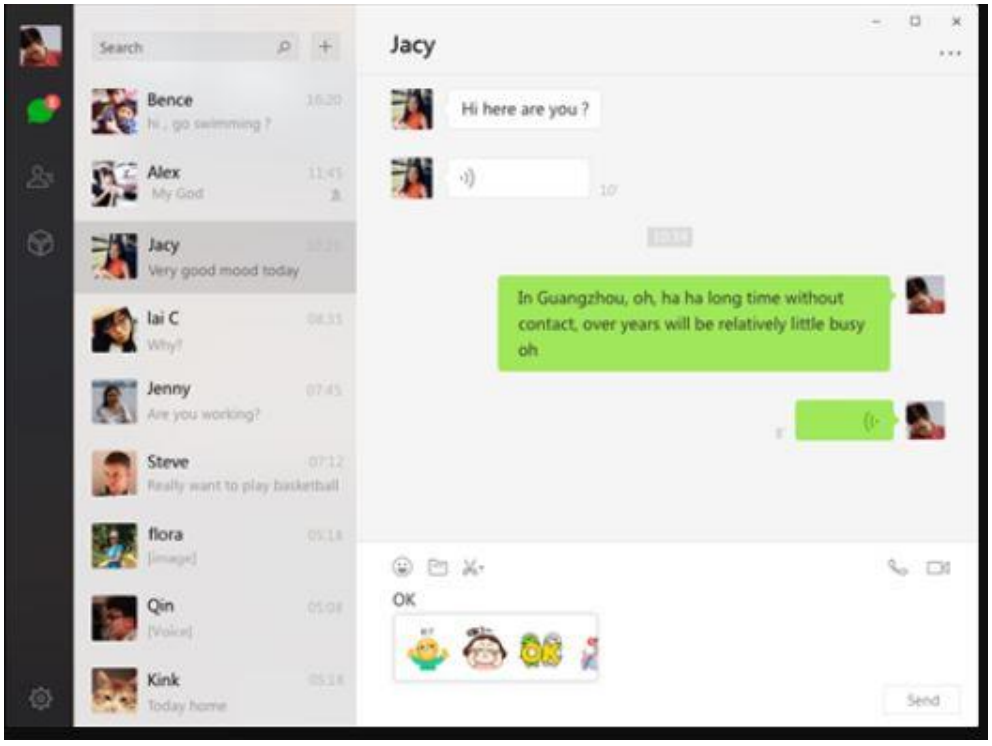

Рисунок 1.8 – Графічний інтерфейс веб-системи WeChat.

WeChat, розроблений компанією Tencent, став ключовим засобом спілкування в Китаї та заробив популярність за межами країни. Додаток пропонує різноманітні можливості, такі як обмін текстовими повідомленнями, відеодзвінки, групові чати та інші соціальні функції [12].

Унікальною особливістю WeChat є вбудована платіжна система, WeChat Pay, яка дозволяє користувачам здійснювати різні фінансові операції, включаючи платежі за товари та послуги, оплату комунальних послуг, а також навіть бронювання та купівлю квитків.

За все це WeChat визначається як екосистема, яка забезпечує не лише спілкування, але й широкі можливості для повсякденного життя та комунікації в цифровій ері. Однак, разом із зручністю, слід враховувати потенційні обмеження приватності та цензуру, що пов'язані з діяльністю компанії в контексті китайського законодавства [13].

1.5 Загальні вимоги до програмного забезпечення.

Враховуючи результати попередніх досліджень, технології які використовувались в розробці аналогів а також їхні плюси і мінуси, було визначено що програма повинна складатись з двох основних частин: клієнтської і серверної, які повинні взаємодіяти між собою.

Серверна частина відповідає за обробку запитів від клієнта, включаючи перевірку ідентифікації клієнта, обробку запитів та вірне виявлення та повідомлення про помилки, такі як непрацююча база даних чи невірно сформований запит [14].

Клієнтська частина реалізована у вигляді веб-додатку, який запускається в браузері і дозволяє користувачеві взаємодіяти з сервером через інтерфейс, включаючи введення запитів, обмін інформацією та отримання повідомлень про помилки.

Програма має бути реалізована за допомогою мов програмування TypeScript, HTML та CSS. З метою спрощення розробки слід використовувати бібліотеки React для клієнтської частини і NestJS для серверної частини.

Для шифрування паролів слід використовувати бібліотеку bcrypt, а для вирішення завдань аутентифікації та авторизації користувачів – бібліотеку Passport.js із застосуванням JWT (JSON Web Tokens) [15].

В результаті розроблення буде отримано програмне забезпечення, означене як "Програма для спілкування між користувачами", що має на меті полегшити взаємодію між користувачами, створюючи зручні умови для знаходження і спілкування з іншими користувачами за допомогою ряду функцій.

Основні функції включатимуть можливість здійснювати пошук інших користувачів за певними параметрами, обмін повідомленнями та перегляд інформації про користувачів. Завдяки цьому веб-система буде сприяти зручній взаємодії між користувачами та робити спілкування більш доступним та ефективним [16].

1.6 Висновки

У даному розділі було досліджено сучасні технології та реалізації веб-месенджерів, клієнт-серверної архітектури, концепції автоматичної фільтрації контенту та існуючих рішень у контексті вимог до програмного забезпечення. Висвітлено ключові аспекти області досліджень, що визначать основні напрямки які було реалізовано в наступних розділах роботи.

2 АРХІТЕКТУРА СИСТЕМИ, ЩО РОЗРОБЛЯЄТЬСЯ

В результаті досліджень, проведених в першому розділі, було з’ясовано, що наступними напрямками дослідження є визначення вимог до архітектури системи, що розробляється, визначення характеристик для подальшої оцінки успішності архітектури системи, вибір оптимальних технологій, що будуть використані під час розроблення проекту. Також, важливим є приділення окремої уваги розробленню модулю фільтрації контенту.

2.1 Визначення загальних вимог до архітектури системи

Архітектура програмного забезпечення визначається, як структура програми чи обчислювальної системи, що охоплює програмні компоненти, їх зовнішні властивості та взаємозв'язки між ними. Цей термін також може застосовуватися до документування програмної архітектури, що полегшує комунікацію між зацікавленими сторонами та фіксує прийняті рішення щодо високорівневого дизайну системи на ранніх етапах проектування. Крім того, це дозволяє використовувати компоненти та шаблони цього дизайну в інших проектах [17].

Галузь комп'ютерних наук, з моменту свого виникнення, вирішувала проблеми, пов'язані зі складністю програмних систем. Раніше ці проблеми вирішувалися розробниками через правильний вибір структур даних, розробку алгоритмів та застосування концепції розмежування повноважень.

Термін "архітектура програмного забезпечення" став популярним у середовищі ПЗ з 1980-х років, але його фундаментальні принципи використовувалися ще раніше. Спроби усвідомити та пояснити програмну архітектуру системи були початково неточними, але з часом були розвинуті шаблони проектування, стилі дизайну та інші концепції.

Основною ідеєю програмної архітектури є зниження складності системи за допомогою абстракції та розмежування повноважень, хоча досі немає єдиної думки щодо чіткого визначення цього терміну.

Початки концепції архітектури програмного забезпечення були закладені в роботах Едсгера Дейкстри та Девіда Парнаса у 1968-1970 роках, які визначили важливість структури системи ПЗ та правильної побудови її структури.

У розвитку цієї дисципліни велику роль відіграють науково-дослідні установи, такі як університет Карнегі-Меллон та Інститут ім. Ірвайна. Стандарт IEEE 1471, прийнятий у 2007 році як ISO/IEC 42010:2007, визначає переважно програмні системи та відіграє ключову роль у цій галузі.

Діяльність, яка включає в себе визначення глобальних обмежень для проектування системи, таких як вибір парадигми програмування, архітектурних стилів, стандартів розробки програмного забезпечення, що базуються на використанні компонентів, принципів проектування та обмежень, що накладаються законодавством, може бути представлено як архітектура програмного забезпечення чи стратегія його розробки. Ця діяльність включає в себе детальне проектування та розробку тактики, які пов'язані із визначенням локальних обмежень проекту, таких як шаблони проектування, архітектурні моделі, ідіоми програмування та рефакторинг [18].

Згідно з гіпотезою "напруги/околу", різниця між архітектурним та детальним проектуванням визначається критерієм околу. Твердження, що дизайн ПЗ є архітектурним, є істинним лише тоді, коли програма, яка відповідає цьому критерію, може бути розширена в програму, яка йому не відповідає. Наприклад, стиль додатку клієнт-сервер вважається архітектурним, оскільки програма, побудована на цьому принципі, може бути розширена, додаючи вузли peer-to-peer.

Хоча архітектура є формою проектування, не кожен дизайн може вважатися архітектурним. На практиці архітектор визначає межу між архітектурою програмного забезпечення та детальним дизайном. Відсутні універсальні правила чи інструкції для створення архітектури ПЗ, але існує

багато поширених методів розробки програмних модулів та їх зв'язків, таких як "Blackboard", "клієнт-серверна модель", "архітектури, побудовані навколо бази даних", "розподілені обчислення", "подієва архітектура", "Front end and back end", "неявні виклики", "монолітний додаток", "Peer-to-peer", "пайпи та фільтри", "плагіни", "Rule evaluation", "пошуко-орієнтовані архітектури", "сервіс-орієнтовані архітектури", "Shared nothing architecture", "Software componentry", "структуровані" та "трирівневі" [19].

2.2 Характеристики оцінки успішності архітектури продукту

У сучасний період, коли відсутні чіткі правила для "правильного" шляху розробки систем, архітектура програмного забезпечення розглядається як поєднання науки і мистецтва. Аспект мистецтва полягає в тому, що будь-яка комерційна система має враховувати свою придатність і місію. Основні цілі системи визначаються за допомогою сценаріїв та нефункціональних вимог. Атрибути якості включають в себе відмовостійкість, збереження зворотної сумісності, розширюваність, надійність, обслуговуваність, доступність, безпечність, зручність користування та інші характеристики.

З погляду користувача програмної архітектури, вона визначає напрямок руху і розв'язання завдань для команд розробників, тестування, підтримки і кінцевих користувачів. У цьому контексті архітектура програмного забезпечення справді об'єднує різні погляди на систему. Це визначає необхідність створення архітектури до етапу реалізації програмного забезпечення [20].

Оцінка кінцевої архітектури системи визначається декількома ключовими характеристиками. Ці параметри включають час відгуку, що представляє собою час, необхідний системі для обробки зовнішнього запиту, і швидкість реагування, яка визначає швидкість підтвердження запиту. Швидкість реагування особливо важлива для інтерактивних систем, оскільки вона впливає на сприйняття користувачами ефективності системи.

Ще однією важливою характеристикою є час затримки, який визначає мінімальний інтервал часу до отримання будь-якого відгуку від системи. У розподілених системах цей параметр стає ключовим, оскільки він визначає час, необхідний для передачі запиту і відповіді через мережу. Зниження часу затримки стає завданням для розробників, оскільки це сприяє покращенню продуктивності і забезпеченню більш ефективної роботи системи.

Пропускна здатність, яка визначає кількість даних чи операцій, що можуть бути передані чи виконані протягом одиниці часу, є ключовим показником продуктивності. Наприклад, при тестуванні процедури копіювання файлу пропускна здатність вимірюється кількістю переданих байтів за секунду. У світі корпоративних додатків часто використовується показник "транзакцій на секунду" (tps), але його обмеженість полягає в тому, що різні транзакції можуть мати різну складність. Таким чином, для повноцінного вимірювання продуктивності важливо враховувати суміш "типових" транзакцій [21].

Поняття завантаження вказує на ступінь "тиску" на систему, що визначається кількістю одночасно підключених користувачів. Це може суттєво впливати на інші характеристики, зокрема на час відгуку. Нерідко можна почути висловлення, де час відгуку на запит зазначається як "0,5 секунди для 10 користувачів та 2 секунди для 20 користувачів".

Чутливість до завантаження визначає ступінь залежності часу відгуку від рівня завантаження. Наприклад, система А може мати час відгуку 0,5 секунди як для 10, так і для 20 користувачів, тоді як система В показує час відгуку 0,2 секунди для 10 користувачів і 2 секунди для 20 користувачів. У цьому випадку можна стверджувати, що система А має меншу чутливість до завантаження, що робить її менш вразливою до коливань навантаження. Важливо враховувати ці параметри при оцінці продуктивності системи, оскільки поліпшення одного показника може вплинути на інші аспекти її ефективності.

Ефективність визначається як питома продуктивність в перерахунку на одиницю ресурсу. Наприклад, система з двома процесорами, яка здатна виконати 30 транзакцій на секунду (tps), може бути більш ефективною порівняно з системою, що має чотири аналогічні процесори і досягає продуктивності в 40 tps.

Потужність визначається як максимальне значення пропускної здатності чи завантаження системи. Це може бути абсолютним максимумом або значенням, при якому продуктивність все ще перевершує прийнятний поріг.

Здатність до масштабування характеризує поведінку системи при додаванні ресурсів, зазвичай апаратних. Масштабованою вважається система, чиє зростання продуктивності пропорційне збільшенню ресурсів, наприклад, удвічі при подвоєнні кількості серверів. Вертикальне масштабування - це збільшення потужності окремого сервера, наприклад, за рахунок збільшення обсягу оперативної пам'яті. Горизонтальне масштабування - це нарощування потенціалу системи шляхом додавання нових серверів [22].

Не можна недооцінювати роль архітектури в кінцевому продукті, оскільки цей аспект є важливим для розвитку та підтримки, незалежно від функціональності, яку продукт надає. Є випадки на ринку програмного забезпечення, коли нескошений продукт архітектурно стає дуже популярним, тоді як інший, хоч і довершений внутрішньо, залишається непоміченим для користувачів.

2.3 Вибір оптимальних технологій для розроблення проекту

Для розробки клієнт-серверного додатку веб-месенджера було вибрано набір технологій, що найкраще відповідає вимогам проекту. На рівні клієнтської сторони використовується TypeScript, HTML, та CSS, з фронтенд-розробкою, що базується на потужному фреймворку React.js. Це не тільки надає широкий функціональний спектр, але й сприяє збільшенню продуктивності розробки завдяки його модульності та ефективним інструментам [23].

Для створення серверної частини обрано NestJS, що базується на Node.js. Цей фреймворк надає розробникам високий рівень абстракції, спрощуючи розробку серверних додатків через використання TypeScript та патернів програмування. NestJS дозволяє створювати масштабовані та досконало структуровані додатки, що важливо для успішної розробки веб-месенджера.

HTML (Hypertext Markup Language) - це мова розмітки, яка використовується для створення структури веб-сторінок. За допомогою HTML визначаються елементи та їх взаємодія на сторінці. У найпростішому вигляді HTML складається з парних тегів, які оточують контент та непарних тегів, які задають властивості елементів.

HTML використовується для створення веб-сторінок, що мають структуру та логічний контент. В основі HTML лежить система тегів, які описують тип та призначення елементів на сторінці. Елементи можуть включати заголовки, абзаци, списки, зображення, посилання, форми та багато іншого[24].

Основні елементи HTML включають в себе:

– <html>: Кореневий елемент, який обгортає весь вміст веб-сторінки;

– <head>: Містить мета-інформацію про документ, таку як заголовок, метатеги та посилання на зовнішні ресурси;

– <title>: Визначає заголовок сторінки, який відображається в заголовку вікна браузера або в закладках;

– <body>: Містить весь видимий контент веб-сторінки, такий як текст, зображення, посилання та інші елементи;

– <h1>, <h2>, ..., <h6>: Заголовки різних рівнів, використовуються для визначення ієрархії заголовків на сторінці;

– <p>: Визначає абзац тексту;

– <a>: Створює посилання на інші веб-сторінки або ресурси.

– , <ol>, <li>: Використовуються для створення неупорядкованих (маркованих) та упорядкованих (нумерованих) списків;

– : Групує елементи і надає можливість застосування стилів до цілої групи.

HTML є основною складовою веб-розробки, і використовується разом з CSS та JavaScript для створення повноцінних веб-додатків. За допомогою HTML можна створювати різноманітні елементи, від текстового контенту до мультимедіа та форм для збору даних від користувачів.

HTML визначає структуру сторінок, розташовуючи елементи у логічні блоки. Наприклад, розміщення тексту в абзацах <p>, використання заголовків <h1>, <h2>, тощо, для ієрархії інформації. Теги також можуть вкладатися один в одного, створюючи складніші структури.

Додатково, HTML включає можливості роботи з посиланнями <a> та зображеннями <img>, роблячи сторінку інтерактивною та візуально привабливою. Це дозволяє створювати зручний та логічно організований контент для користувачів в Інтернеті.

HTML5, остання версія HTML, принесла багато нововведень, таких як нові елементи для відео, аудіо, геолокації, а також покращення у сфері доступності та можливості роботи офлайн. Загалом, HTML залишається основою для будь-якого веб-ресурсу, визначаючи його основну структуру та взаємодію з користувачем.

Теги HTML можна комбінувати для створення складніших структур, атрибути тегів дозволяють налаштовувати їхню поведінку та вигляд. HTML використовується спільно з CSS для надання стилів та JavaScript для реалізації динамічної взаємодії на сторінці. Всі ці елементи разом допомагають створити зручний та ефективний інтерфейс для користувачів у Всесвітній павутині.

CSS (Cascading Style Sheets) – це мова стилів, що використовується в веб-розробці для деталізованого визначення зовнішнього вигляду веб-сторінок. Головна мета CSS полягає в окремості дизайну від HTML-структури, забезпечуючи консистентний та ефективний спосіб керування оформленням.

Ця технологія дозволяє розробникам визначати такі аспекти як кольори, шрифти, розташування, розміри та інші стилізаційні властивості, що забезпечують естетично приємний та користувацьки зручний вигляд веб-додатків та сайтів.

Завдяки концепції "каскаду" (cascading), CSS дозволяє застосовувати стилі до елементів відповідно до їхнього порядку в коді, а також враховувати специфічність селекторів. Це робить CSS потужним інструментом для створення структурованих, легко змінюваних та елегантних веб-інтерфейсів.

CSS допомагає розробникам забезпечувати єдність дизайну на всьому веб-сайті чи додатку, що полегшує підтримку та розвиток проєкту. Зокрема, використання класів та ідентифікаторів дозволяє точно визначати елементи для стилізації, забезпечуючи легкість та читабельність коду.

Окрім основних стилізаційних можливостей, CSS включає в себе адаптивні та респонсивні техніки, що дозволяють адаптувати дизайн до різних розмірів екранів, щоб забезпечити оптимальний вигляд як на комп'ютерах, так і на мобільних пристроях [25].

З введенням нових стандартів, таких як CSS Grid та Flexbox, розробники отримали ще більше можливостей для легкої та ефективної організації макетів сторінок. Це робить CSS важливим інструментом для сучасних веб-розробників, дозволяючи їм створювати стильні та функціональні веб-інтерфейси для користувачів.

CSS дозволяє розробникам вирізнятися та надавати унікальний вигляд своїм веб-проєктам. Гнучкість, яку надає ця мова стилів, дозволяє не лише естетично оформити сторінки, але й оптимізувати їх для взаємодії з користувачем.

Важливо також зазначити, що CSS не обмежується лише виглядом елементів. За допомогою анімацій та переходів, які можна визначити у CSS, розробники можуть створювати плавні та динамічні зміни в інтерфейсі, покращуючи враження користувачів від взаємодії з веб-сайтом чи додатком.

Додатково, CSS допомагає у вирішенні проблеми кросбраузерності, дозволяючи створювати стилі, які можуть коректно відображатися на різних браузерах. Це важливо для того, щоб забезпечити єдність та консистентність вигляду для всіх користувачів, незалежно від того, який браузер вони використовують.

TypeScript представляє собою мову програмування, розроблену для покращення та розширення можливостей JavaScript у сфері створення великих та складних програмних систем. Однією з ключових особливостей TypeScript є введення статичної типізації, що сприяє підвищенню безпеки та стабільності кодової бази.

Використання TypeScript дозволяє розробникам ефективно впроваджувати принципи об'єктно-орієнтованого програмування та використовувати концепції, такі як класи та модулі, для створення чистого та організованого коду. Додатково, TypeScript пропонує інші аспекти сучасної розробки, такі як робота з інтерфейсами та визначенням типів даних.

Ця мова програмування дозволяє забезпечити високу міру масштабованості веб-додатків та полегшує процес рефакторингу. Завдяки компіляції в стандартний JavaScript, TypeScript стає доступним для використання у різних середовищах, що робить його популярним інструментом для професійної веб-розробки [26].

TypeScript виникає як рішення для комплексних веб-додатків, де необхідна більш висока ступінь строгості та структурності коду. За допомогою статичної типізації, розробники можуть визначати типи даних та використовувати їх для виявлення та усунення помилок на етапі розробки, що сприяє більшій надійності програмного забезпечення.

Однією з ключових переваг TypeScript є його відмінна сумісність з екосистемою JavaScript. Він розпізнає та компілює існуючий JavaScript-код, що дозволяє поступово впроваджувати TypeScript у вже існуючі проєкти без великого обсягу переписування коду.

Крім того, TypeScript підтримує останні тенденції у світі веб-розробки, включаючи асинхронні операції та нові стандарти JavaScript. За допомогою TypeScript розробники можуть ефективно працювати з новітніми функціональностями та бібліотеками, що полегшує втілення сучасних розробничих підходів у практиці.

У сукупності, TypeScript є потужним інструментом для професійної веб-розробки, сприяючи підвищенню продуктивності та якості програмних рішень.

TypeScript вносить значний внесок у парадигму розробки програмного забезпечення, дозволяючи розробникам вести проекти великої масштабованості з високим рівнем структурованості та контролю. Статична типізація допомагає зменшити ризик виникнення помилок, покращує автодокументацію та сприяє збільшенню зрозумілості кодової бази.

Однією з ключових переваг TypeScript є його розширена підтримка об'єктно-орієнтованого програмування, що дозволяє розробникам використовувати класи, успадкування та інші концепції для створення модульного та легко управляємого коду.

Також, TypeScript інтегрується з багатьма популярними інструментами розробки, включаючи редактори коду та фреймворки, що сприяє високій продуктивності розробників.

Завдяки активній підтримці великої спільноти та постійному вдосконаленню мови, TypeScript залишається важливим інструментом для тих, хто прагне підняти рівень якості, ефективності та надійності свого веб-програмного забезпечення.

React.js — це бібліотека JavaScript для створення користувацьких інтерфейсів. Розроблена компанією Facebook, React зорієнтована на простоту та ефективність розробки веб-додатків, зокрема тих, які вимагають частої зміни відображення даних.

React використовує концепцію компонентів — повторно використовуваних блоків для побудови інтерфейсу. Один із головних принципів React - це віртуальний DOM, який дозволяє ефективно маніпулювати великою кількістю елементів на сторінці [27].

Більше того, React впроваджує концепцію "однозначного напрямку" (unidirectional data flow), що полегшує відстеження та управління станом додатку. Він також легко інтегрується з іншими бібліотеками та фреймворками, дозволяючи розробникам використовувати його у різних технологічних стеках.

React використовується для створення сучасних веб-додатків, які відрізняються високою ефективністю, гнучкістю та відмінною організацією коду. Особливо популярний у спільноті веб-розробників, React застосовується в різних сферах, включаючи створення односторінкових додатків, великих веб-сервісів та мобільних додатків.

На рисунку 2.1 показано архітектуру фреймворку React.js.

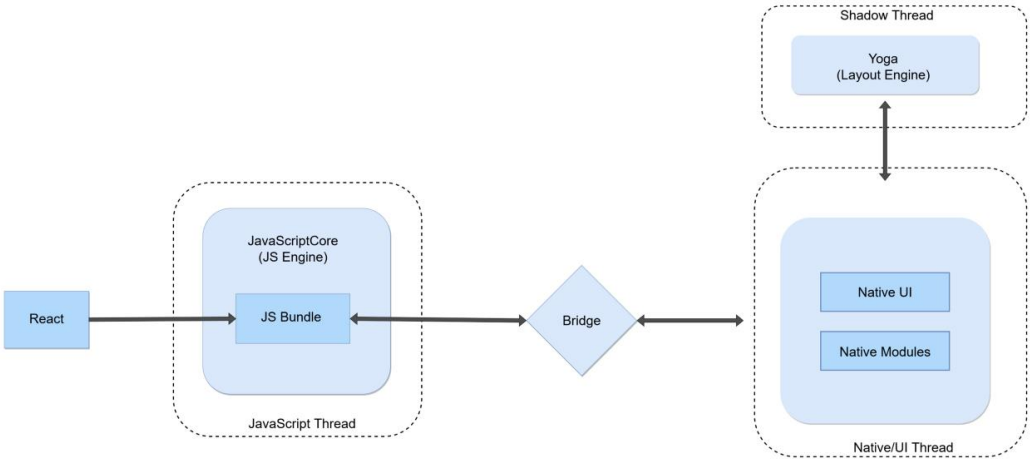

Рисунок 2.1 – Архітектура фреймворку React.js.

React.js є важливим інструментом у сучасній веб-розробці, і його популярність визначається не лише потужністю, але й послідовністю концепцій, які він вносить у процес створення веб-додатків. Ключовою складовою React є його підхід до компонентного програмування, що сприяє покращенню реюзабельності коду та підтримує легше управління станом додатку.

Інша важлива риса React - це узгоджений підхід до роботи з відображенням. Створення віртуального DOM та ефективна перерендерізація лише змінених частин дозволяють додаткам React бути високопродуктивними, зменшуючи навантаження на браузер.

React також інтегрується з іншими інструментами, такими як Redux для керування станом, або React Router для навігації в односторінкових додатках. Це

дозволяє розробникам будувати розширені та добре організовані додатки на основі відкритих стандартів.

Загалом, React продовжує залишатися в центрі веб-розробки, де він використовується для створення інтерактивних та високопродуктивних інтерфейсів, забезпечуючи розробникам потужний інструмент для будівництва сучасних веб-додатків.

React вирізняється також своєю спрощеною моделлю життєвого циклу компонентів, що спрощує управління станом та взаємодію з подіями в додатку. Крім того, велика спільнота та багатий екосистемний стек бібліотек і інструментів дозволяють розробникам вибирати оптимальні рішення для конкретних вимог проєкту.

React також активно підтримується і розвивається Facebook та іншими компаніями, що гарантує його актуальність та сучасність. Постійні оновлення та нововведення, які надходять у межах розробки React, дозволяють розробникам залишатися на передньому краї технологічних тенденцій.

Багато компаній та розробників обирають React для реалізації своїх проєктів, визнаючи його як потужний, гнучкий та ефективний інструмент для створення інтерактивних та високопродуктивних веб-додатків.

Node.js - це виконавче середовище, що дозволяє виконувати JavaScript код на сервері. Заснований на двигуні V8 від Google, Node.js дозволяє створювати ефективні та масштабовані мережеві додатки, зокрема сервери.

Основна особливість Node.js - це асинхронний та подієво-орієнтований підхід до програмування. Він використовує один потік для обробки багатьох запитів, що дозволяє створювати додатки, які швидко реагують на події та взаємодіють з багатьма клієнтами одночасно.

Node.js є частиною стеку MEAN (MongoDB, Express.js, Angular.js, Node.js), який використовується для створення повностекових веб-додатків. Він також популярний у сфері розробки засобів реального часу, таких як чат-додатки або ігри.

Завдяки широкому сприйняттю та активній спільноті, Node.js залишається важливим інструментом для веб-розробників, дозволяючи їм створювати високопродуктивні та масштабовані серверні додатки.

Node.js відкриває нові можливості для розробників, дозволяючи використовувати ту ж саму мову програмування (JavaScript), яка використовується на клієнтському боці, і на стороні сервера. Це спрощує взаємодію між фронтендом та бекендом, забезпечуючи єдність кодової бази та зменшуючи бар'єри для розробки [28].

Однією з ключових переваг Node.js є його можливість ефективно працювати з багатьма одночасними підключеннями. Це досягається завдяки використанню неблокуючих асинхронних операцій вводу/виводу. Це особливо важливо для додатків, які обробляють великий потік даних чи запитів в реальному часі.

На рисунку 2.2 показано діаграму що відображає архітектуру проекту на Node.js.

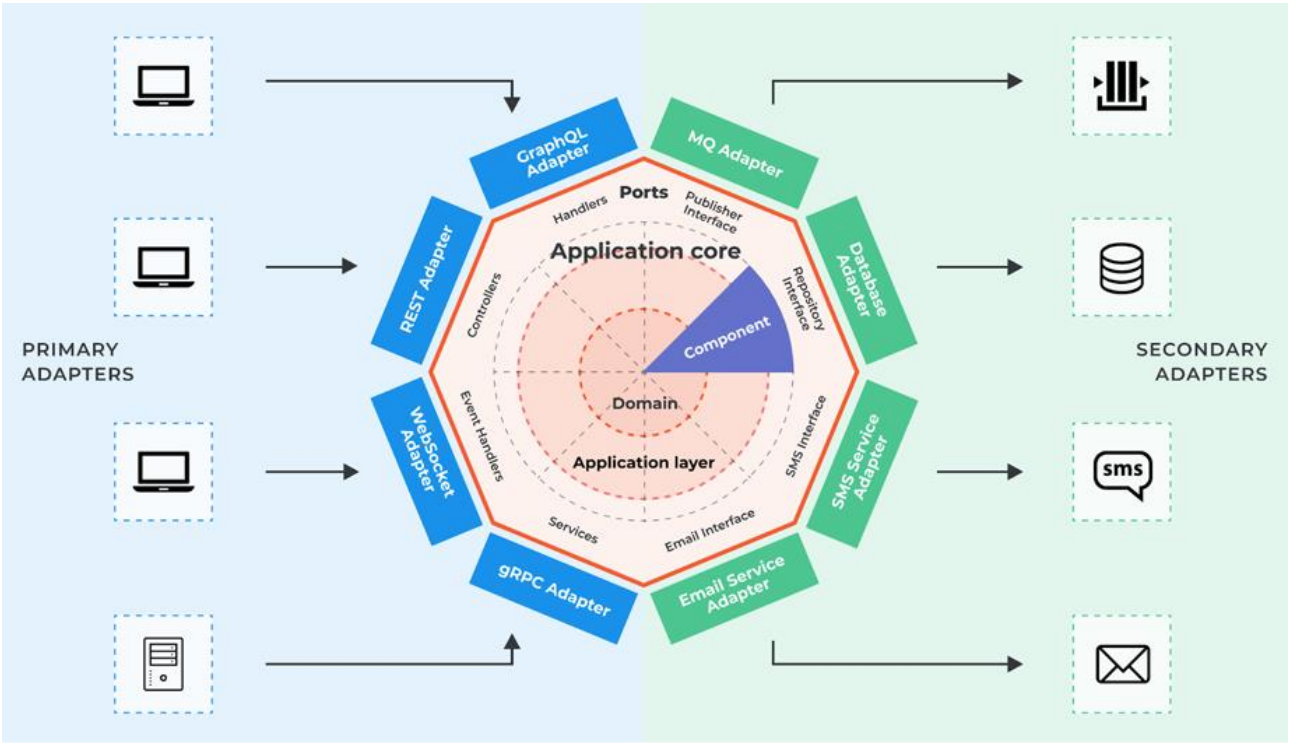

Рисунок 2.2 – Архітектура проекту на Node.js.

Node.js також активно використовується для створення API (інтерфейсів програмування додатків), мікросервісів та серверів для сучасних веб-додатків. Його легкість використання та швидкість роботи роблять його перспективним вибором для різних типів проєктів та додатків.

Node.js став популярним інструментом для створення додатків з великим обсягом вводу/виводу, таких як додатки реального часу, чати, онлайн-ігри та стрімінгові сервіси. Використання JavaScript для якості як на стороні клієнта, так і на сервері, робить веб-розробку ще більш консистентною та продуктивною.

Node.js володіє широкою спільнотою та розширеною бібліотекою модулів з найрізноманітніших областей розробки. З цим розширеним екосистемним стеком, розробники можуть легко впроваджувати різноманітні функціональності в свої проєкти без потреби повторного визначення колеса.

Окрім того, Node.js часто використовується для створення мікросервісної архітектури, де функціональні частини додатку реалізуються окремими, незалежними сервісами. Це полегшує масштабування, підтримку та розвиток складних систем.

NestJS є фреймворком для створення масштабованих та ефективних серверних додатків на базі Node.js. Він використовує TypeScript, що дозволяє вам використовувати принципи об'єктно-орієнтованого програмування та статичної типізації для створення добре структурованих та надійних додатків.

NestJS пропонує концепцію модульності, що полегшує організацію коду та його розширення. Він також використовує патерні проектування, такі як Dependency Injection, що сприяє створенню легкорозумілого та легкозмінюваного коду.

Однією з суттєвих особливостей NestJS є його вбудована підтримка WebSocket для реалізації взаємодії в режимі реального часу. Це робить його відмінним вибором для створення додатків, які вимагають обміну даними в режимі реального часу, наприклад, чати або сповіщення.

Крім того, NestJS добре інтегрується з іншими популярними бібліотеками та фреймворками, такими як TypeORM для роботи з базами даних, або Swagger для автоматичної генерації документації API.

NestJS також відзначається вбудованою підтримкою для створення мікросервісів. Фреймворк надає інструменти для організації додатків у вигляді невеликих, незалежних сервісів, що дозволяє розробникам будувати розподілені системи, які легко масштабувати та розвивати.

Використання NestJS забезпечує зручний підхід до взаємодії з базами даних. Фреймворк підтримує велику кількість баз даних та орієнтований на використання ORM (Object-Relational Mapping) для спрощення роботи з базовими операціями.

NestJS також пропонує можливості для автоматичної генерації документації API, використовуючи засоби, такі як Swagger. Це сприяє покращенню розуміння та взаємодії з API, що є важливим елементом в процесі розробки [29].

Фреймворк дуже активно підтримується та оновлюється, що гарантує актуальність інструментів та технологій у його екосистемі. NestJS продовжує завоювувати популярність серед розробників як в універсальній веб-розробці, так і в розробці мікросервісів та розподілених систем.

NestJS інтегрується з рядом додаткових інструментів та пакетів, що розширюють його можливості. Наприклад, використання middleware та пайпів дозволяє вам легко обробляти та валідувати дані перед тим, як вони потрапляють до вашого додатку.

Додатково, NestJS має вбудовану систему логування та обробки помилок, що полегшує відстеження та вирішення проблем. Інтеграція з різноманітними системами логування та моніторингу робить його ідеальним вибором для розгортання та підтримки продукційних систем.

NestJS активно підтримує принципи тестування, що дозволяє розробникам створювати надійний та стабільний код. Інтеграція з різними фреймворками для тестування, такими як Jest, сприяє розробці та підтримці тестового покриття.

2.4 Вибір СКБД для реалізації проекту

База даних - це структурована колекція даних, організованих для забезпечення ефективного доступу, управління та оновлення інформацією. Основні компоненти бази даних включають таблиці, які представляють основний елемент, атрибути, що описують властивості даних, а також відносини, які визначають зв'язки між таблицями.

Система керування базами даних (СКБД) використовується для управління та обробки цих даних, а мова структурованих запитів (SQL) - для взаємодії з базою даних. Бази даних використовуються для зберігання великих обсягів інформації у електронній формі, що дозволяє ефективно здійснювати роботу з даними в різних областях, включаючи бізнес, науку та інші галузі [30].

Сучасні системи керування базами даних (СКБД) включають в себе різноманітні технології та можливості, що роблять їх потужними і універсальними інструментами для зберігання та обробки даних. Нові тенденції в розвитку СКБД включають в себе розподілені системи, обробку даних в реальному часі, підтримку масштабованості та високу доступність. Багато СКБД також надають інтерфейси для взаємодії з іншими технологіями, такими як машинне навчання та штучний інтелект. Вони також часто підтримують різні моделі даних, такі як реляційні, NoSQL, та NewSQL, щоб задовольнити різноманітні вимоги застосувань. Важливими характеристиками є ефективність, безпека та забезпечення конфіденційності, а також можливість інтеграції з іншими інструментами та сервісами

На рисунку 2.3 показано таблицю порівння сучасних СКБД.

Порівняння найпопулярніших типів баз даних

	Тип ДБ	Ліцензія	Документ-тація	Масшта-бованість	Тип даних	Скл-ть вивчення
MySQL	SQL	Ліц. на GNI	Нормальна	вертикальний комплекс	Структурований, напівструктурований	Середня
Maria DB	SQL	Ліц. на GNI	Чудова	вертикальний	Структурований, напівструктурований	Середня
Oracle	Мульти-модель, SQL	Власність	Чудова	вертикальний	Структурований, напівструктурований, неструктурований	Складна
PostgresQL	Об'єктно-орієнтована модель. SQL	Відкрита	Нормальна	вертикальний	Структурований, напівструктурований, неструктурований	Складна
MSSQL Server	T-SQL	Власність	Чудова	вертикальний комплекс	Структурований, напівструктурований, неструктурований	Складна
MongoDB	NoSQL, документо орієнтована модель	SSPL	Чудова	Горизонтальна	Структурований, напівструктурований, неструктурований	Середня
Redis	NoSQL	Відкрита, BSD 3-клас	Чудова	Горизонтальна	Структурований, напівструктурований, неструктурований	Середня
Cassandra	NoSQL	Відкрита	Чудова	Горизонтальний	Структурований, напівструктурований, неструктурований	Складна
Elastic-Serch	NoSQL, документо орієнтована модель	Відкрита	Нормальна	Горизонтальний	Структурований, напівструктурований, неструктурований	Складна

Рисунок 2.3 – Таблиця порівняння актуальних СКБД.

MySQL - відкрита реляційна система керування базами даних (СКБД), розроблена та підтримується Oracle Corporation. Ця база даних використовує реляційну модель для організації даних у вигляді таблиць та використовує мову запитів SQL для взаємодії з базою даних. MySQL є однією з найпопулярніших та широко використовуваних СКБД у світі завдяки своїй високій швидкодії та ефективності.

MySQL також славиться своєю відкритістю, що робить його доступним для широкого кола розробників та організацій. Вона підтримує різноманітні операційні системи, включаючи Windows, Linux та macOS, що робить її універсальною для різних середовищ.

Однією з ключових переваг MySQL є активна спільнота користувачів та розробників, яка постійно вносить внески у розвиток системи, створює

допоміжні інструменти та розширення. Це сприяє постійному удосконаленню та вдосконаленню функціоналу MySQL.

Безпека також відіграє важливу роль у MySQL, і база даних надає різноманітні механізми для захисту даних, включаючи ролі користувачів, шифрування та інші заходи безпеки.

Cassandra - це розподілена система управління базами даних, яка призначена для високопродуктивної та стійкої роботи в розподілених середовищах. Дані автоматично розподіляються між вузлами кластера, забезпечуючи рівномірне розподілення навантаження. Система спроектована для уникнення одиничних точок відмов та має механізми для забезпечення надійності даних.

Cassandra підтримує гнучку модель даних, що дозволяє зберігати та обробляти велику кількість типів даних. Вона також надає мови CQL (Cassandra Query Language) для роботи з даними, що нагадує структуру SQL, що полегшує взаємодію з системою для тих, хто вже знайомий із мовою запитів SQL.

Ще однією важливою особливістю є можливість динамічного додавання нових вузлів до кластера без втрати продуктивності або доступності даних. Це робить Cassandra привабливою для великих і зростаючих обсягів даних, таких як ті, які зазвичай виникають у веб-сервісах та додатках для аналізу даних в реальному часі [31].

Крім того, Cassandra має вбудовані механізми для вирішення проблем конфліктів при реплікації даних. Вона використовує схему реплікації зі стратегією контролю за конфліктами, яка дозволяє ефективно управляти ситуаціями, коли різні копії даних виявляють розходження.

Також слід відзначити, що Cassandra побудована з урахуванням принципів AP (доступності та стійкості до розбиття), що робить її відмінним вибором для застосувань, де важлива висока доступність і можливість працювати в умовах розбиття мережі.

Узагальнюючи, Apache Cassandra - це потужний інструмент для роботи з великими обсягами даних у розподіленому середовищі, що відзначається високою масштабованістю, надійністю та можливістю працювати в умовах незавершених мереж.

PostgreSQL - це потужна та розширювана відкрита реляційна система керування базами даних (СКБД). Забезпечує повну реляційну модель, підтримує SQL та має високий рівень стандартів ANSI SQL. PostgreSQL славиться своєю гнучкістю, можливістю розширення та розподіленням даних. Вона підтримує різні типи даних, включаючи географічні та JSON-дані. Захист даних - пріоритет, з можливістю використання ролей, SSL-шифрування та інших заходів безпеки. PostgreSQL володіє великою спільнотою розробників, яка постійно вносить внески в його розвиток, роблячи його інноваційним та підтримуваним рішенням для різних проектів.

PostgreSQL також вражає своєю здатністю до обробки великих обсягів даних та високої продуктивності завдяки оптимізаціям та паралельним виконанням запитів. Однією з визначних особливостей є можливість створення власних функцій та типів даних, що надає розробникам велику гнучкість у створенні специфічних рішень.

Додатково, PostgreSQL включає в себе розширені засоби роботи з транзакціями, забезпечуючи консистентність та надійність даних. Підтримка вираження зовнішніх ключів та індексів дозволяє оптимізувати швидкодію системи. Його розподілені можливості дозволяють створювати географічно розподілені бази даних для ефективної роботи в різних регіонах.

Відкритий код PostgreSQL робить його доступним для широкого кола користувачів, а активна спільнота продовжує сприяти його розвитку та підтримці. Загалом, PostgreSQL є надійним та потужним інструментом для управління базами даних, відповідаючи високим стандартам якості та функціональності.

Oracle Database - це потужна пропрієтарна система керування базами даних (СКБД), розроблена корпорацією Oracle. Вона визначається високою

продуктивністю, надійністю та розширюваністю, що робить її популярним рішенням для великих корпоративних середовищ.

Oracle Database працює на реляційній моделі даних та використовує мову запитів SQL для взаємодії з базою даних. Вона підтримує розширені можливості, такі як вбудовані процедури, тригери та стандарти безпеки, що роблять її підходящою для роботи з великим обсягом даних та складних додатків.

Oracle Database славиться своєю високою доступністю за допомогою механізмів резервування та відновлення. Вона підтримує розподілені системи та великі об'єми даних завдяки оптимізаціям та архітектурним рішенням.

Однією з ключових особливостей Oracle Database є його широкий спектр додаткових можливостей, таких як аналітика, розумний аналіз даних, а також можливості для взаємодії з іншими технологіями, включаючи хмарні сервіси та машинне навчання.

З урахуванням своєї розвиненої функціональності та масштабованості, Oracle Database залишається однією з провідних баз даних у світі корпоративних та великих проектів.

MongoDB також вражає своєю гнучкістю та простотою в роботі, оскільки не вимагає строго визначеної схеми даних, і нові поля можуть додаватися до документів без значного навантаження на базу даних. Це особливо корисно в умовах швидкозмінюючихся вимог до даних у розробці програмного забезпечення [32].

Багато функцій MongoDB, таких як геопросторові запити, можливості розподіленої бази даних та вбудований механізм шардування, роблять її привабливим вибором для великих та розподілених систем. Ця база даних також має активну спільноту користувачів та широкий спектр документації, що полегшує вивчення та використання для розробників.

MongoDB підтримує багато мов програмування та має декілька інструментів управління та моніторингу, що робить його повноцінним інструментом для розробників та адміністраторів баз даних.

В результаті аналізу було визначено, що найкращим вибором для розробки проекту буде використання MongoDB. Ця база даних відповідає потребам проекту завдяки своїй гнучкості, можливості масштабування та простоті інтеграції з іншими технологіями. Враховуючи вимоги проекту до зберігання та обробки даних, MongoDB є оптимальним рішенням для досягнення успішного втілення ідеї.

Оскільки для розробки такого проекту використання реляційної бази даних з великою кількістю зв’язків може зробити проект неоптимізованим, для зберігання юзерів буде використовуватись окремий модуль створений засобами Microsoft SQL Server Management Studio (SSMS), в той час як для більшості функціоналу буде використовуватись MongoDB.

В результаті вивчення можливостей та врахування вимог проекту було вирішено обрати Microsoft SQL Server Management Studio (SSMS) як середовище для створення та керування базою даних.

На рисунку 2.4 показано приклад створення бази даних в Microsoft SQL Server Management Studio.

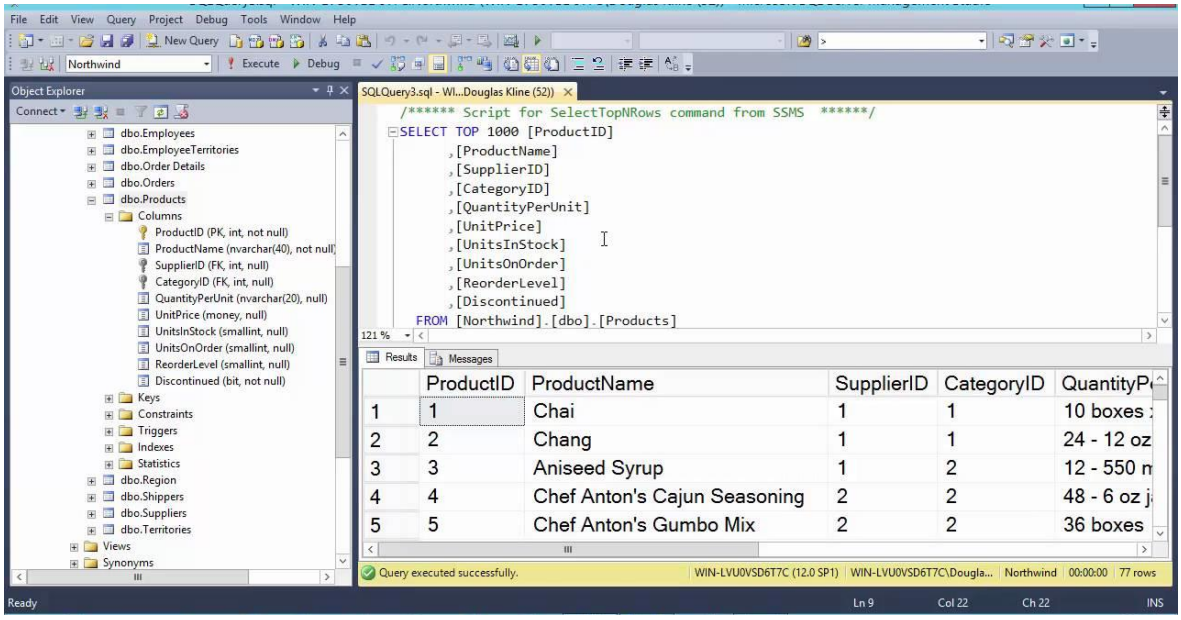

Рисунок 2.4 – Приклад створення бази даних з використанням засобів Microsoft SQL Server Management Studio.

Це рішення дозволяє ефективно взаємодіяти з Microsoft SQL Server, надаючи широкі можливості управління, моніторингу та оптимізації бази даних. SSMS добре інтегрується з різними інструментами розробки та забезпечує зручний інтерфейс для роботи з базою даних, що сприяє ефективному впровадженню проекту.

Крім того, вибір Microsoft SQL Server Management Studio обумовлений його високою надійністю та безпекою, що є важливими аспектами для забезпечення стійкості та захищеності бази даних в рамках проекту. Інтегрована система безпеки та можливості резервного копіювання дозволяють ефективно управляти доступом до інформації та забезпечити високий рівень захисту даних.

Microsoft SQL Server Management Studio пропонує широкі можливості для візуального моделювання бази даних, що полегшує розробку та аналіз схеми даних. Його інтуїтивний інтерфейс та підтримка T-SQL дозволяють розробникам зручно працювати з запитами та оптимізувати продуктивність бази даних.

2.5 Вибір середовища розробки проекту

У процесі розробки проекту використовуються різноманітні технології та інструменти, які спрямовані на створення ефективного та сучасного веб-додатку. Для фронтенд-розробки використовуються мови програмування TypeScript, HTML та CSS, а основною платформою є потужний фреймворк React.js. Цей фреймворк дозволяє ефективно створювати інтерактивні та швидкодіючі користувацькі інтерфейси, забезпечуючи високий рівень користувацького досвіду.

У сфері серверної частини обрано NestJS, який базується на платформі Node.js. Це рішення дозволяє створювати ефективні та масштабовані серверні додатки, використовуючи TypeScript для зручної розробки та підтримки високої продуктивності [33].

Для забезпечення зручного та продуктивного середовища розробки обрано Microsoft Visual Studio Code. Цей текстовий редактор надійно підтримує мови програмування, які використовуються в проекті, і надає широкий спектр розширень та інструментів для полегшення розробки, тестування та налагодження коду.

Об'єднуючи ці технології та інструменти, стек технологій проекту створює надійну та зручну основу для реалізації функціоналу та досягнення поставлених цілей.

На рисунку 2.5 показано огляд стандартних інструментів Microsoft Visual Studio Code.

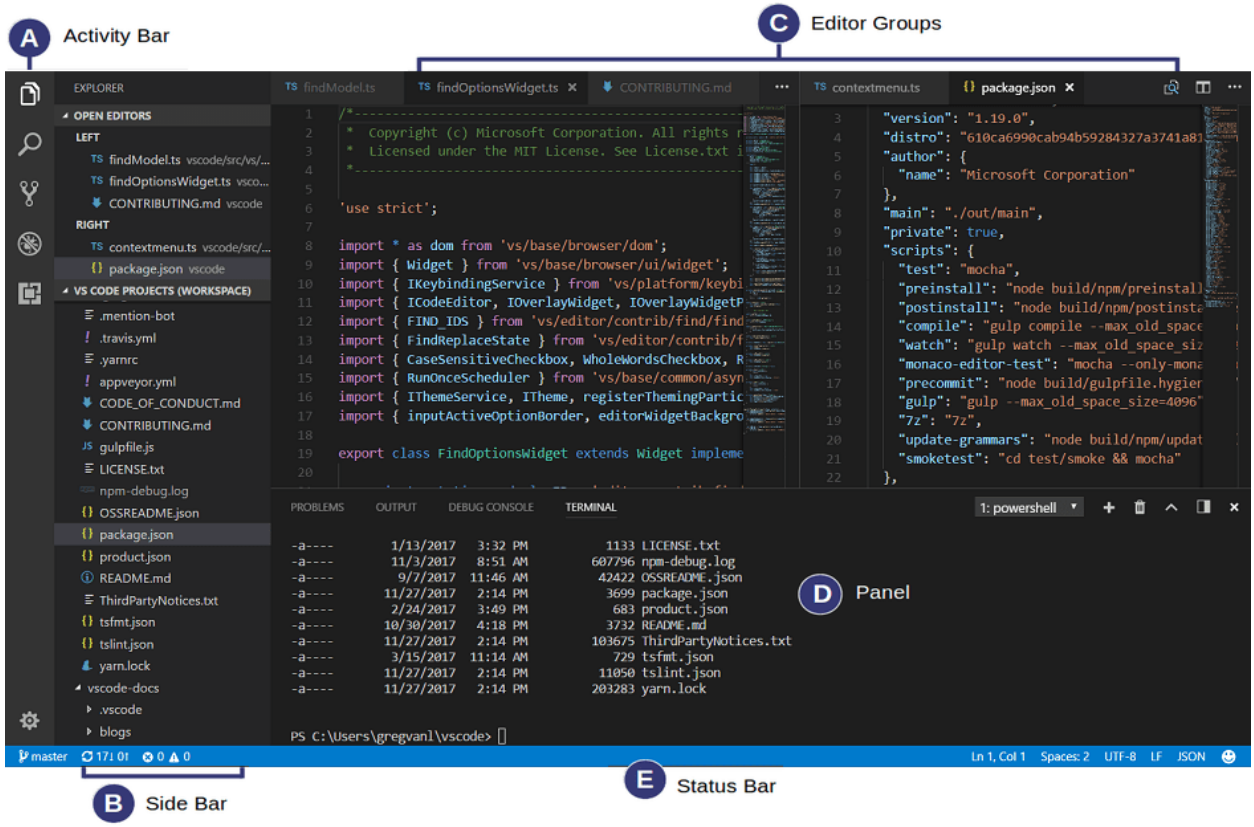

Рисунок 2.5 – Стандартні інструменти Microsoft Visual Studio Code.

Microsoft Visual Studio Code (VS Code) - це безкоштовний текстовий редактор з відкритим вихідним кодом, розроблений компанією Microsoft. Він підтримує багато мов програмування та інструментів для розробки. VS Code надає можливості редагування, відлагодження коду, розширюваність за

допомогою плагінів і широкий спектр інших функцій, що роблять його популярним серед розробників усього світу.

VS Code має інтегрований інтерфейс в управлінні версіями, а також підтримує розробку веб-застосунків і зручні інструменти для роботи з JavaScript, TypeScript, HTML, CSS і іншими технологіями веб-розробки. Редактор також включає в себе вбудовану підтримку для роботи з Docker і розширюється за допомогою великого кількості розширень, які доступні через інтернетовий магазин розширень. Загалом, Visual Studio Code визначається своєю легкістю використання, швидкістю та гнучкістю, що робить його важливим інструментом для розробки програмного забезпечення на різних платформах [34].

VS Code підтримує інтеграцію з популярними системами керування версіями, такими як Git, що дозволяє розробникам ефективно працювати з кодом та веденням його історії змін. Додатково, він включає в себе вбудовані інструменти для відлагодження коду, надаючи можливість розробникам виявляти та усувати помилки в програмах.

На рисунку 2.6 показано приклад інтеграції Git у VS Code.

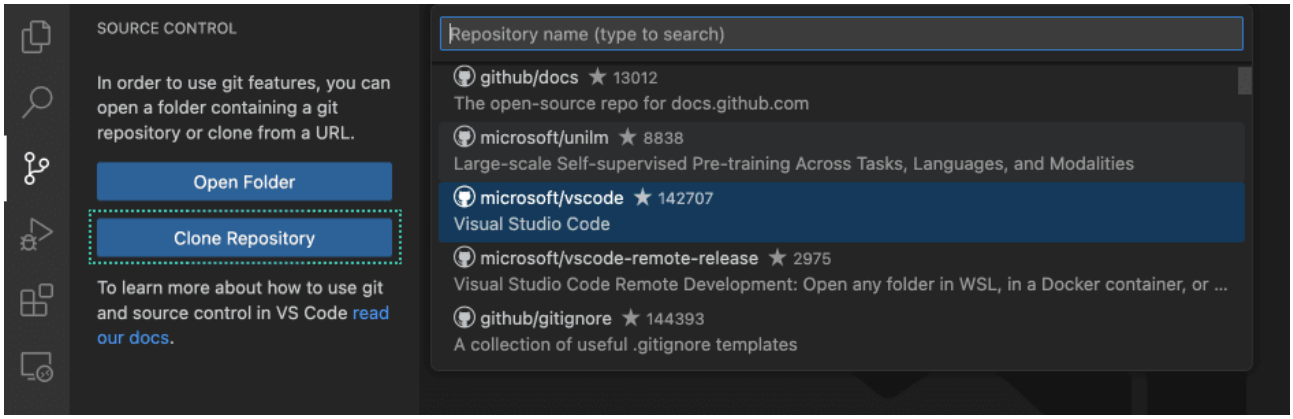

Рисунок 2.6 – Інтеграція Git у VS Code.

Інтеграція Git в Visual Studio Code дозволяє розробникам легко взаємодіяти з системою контролю версій, не залишаючи редактор коду. Розробники можуть відстежувати зміни в своєму коді, створювати коміти, переглядати історію змін, вирішувати конфлікти та виконувати інші операції Git

безпосередньо в VS Code. Це полегшує роботу команди над проектами та сприяє ефективній роботі з кодом.

Крім того, інтеграція Git в VS Code надає розробникам можливість спільно працювати над проектами через різні гілки. Вони можуть легко перемикатися між гілками, створювати нові гілки та злити їхні зміни. Вбудований візуальний граф гілок допомагає зрозуміти структуру роботи над проектом та спрощує навігацію між гілками.

Також розробники можуть використовувати VS Code для розв'язання конфліктів при злитті гілок або при отриманні змін від інших учасників проекту. Інтерфейс дозволяє легко переглядати внесені зміни, обирати, які зміни враховувати, та вирішувати конфлікти безпосередньо в редакторі коду.

Враховуючи ці можливості, інтеграція Git в VS Code стає потужним інструментом для ефективного управління версіями та спільної роботи розробників над проектами будь-якої складності.

VS Code також славиться своєю активною спільнотою та широким спектром документації. Розробники можуть з легкістю знаходити відповіді на свої питання, а також долучатися до внеску в розвиток редактора, щоб поліпшити його функціональність [35].

Завдяки своєму мінімалістичному дизайну та великому набору можливостей, Microsoft Visual Studio Code залишається одним з популярних виборів серед розробників, надаючи зручне середовище для продуктивної та приємної роботи над проектами різної складності.

2.6 Розроблення модулю фільтрації контенту з використанням машинного навчання

Фільтрація контенту за допомогою штучного інтелекту та машинного навчання може включати в себе аналіз не лише самого змісту, а й контексту в якому цей зміст використовується. Наприклад, системи можуть враховувати

контекстуальні особливості обговорення чи спілкування, щоб краще розрізняти між нейтральним та потенційно шкідливим висловлюванням [36].

Також, фільтрація може базуватися на обробці природної мови для виявлення семантичних зв'язків та вираження стосовно визначених критеріїв. Навчання моделей може відбуватися на великих наборах даних, які містять приклади безпечного та небезпечного контенту, щоб моделі могли вивчати характеристики, що визначають ризик чи безпечність.

Іншими словами, системи фільтрації контенту на основі машинного навчання стараються впроваджувати інтелектуальні методи для автоматичного виявлення та блокування потенційно шкідливого або небажаного матеріалу, забезпечуючи тим самим безпеку та комфорт користувачів в онлайн середовищі.

Для побудови алгоритму фільтрації небажаного контенту було вирішено обрати модель машинного навчання LSTM [36].

Модель довготривалої короткострокової пам'яті (Long Short-Term Memory або LSTM) є типом рекурентних нейронних мереж (RNN), спеціально розробленим для ефективного управління довгостроковою залежністю в послідовностях даних. Вони дозволяють моделі легше зберігати та оновлювати інформацію залежно від контексту, що робить їх особливо корисними для роботи з послідовністю текстових даних.

На рисунку 2.7 показано принцип роботи моделі LSTM.

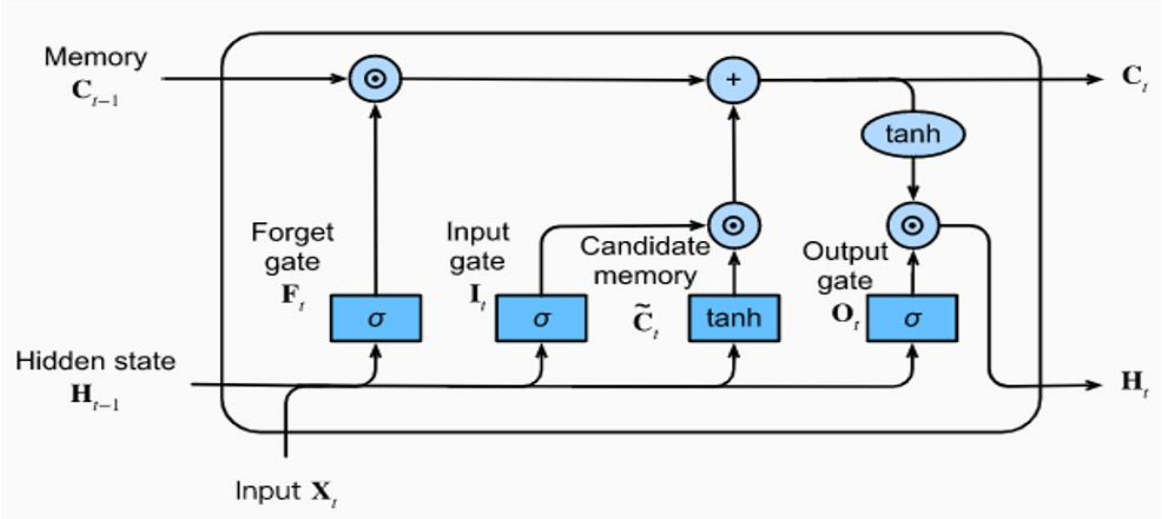

Рисунок 2.7 – Принцип роботи моделі LSTM.

Для навчання моделі було використано публічний датасет «SMS Spam Collection Dataset»: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset.

«SMS Spam Collection Dataset» – це набір SMS-повідомлень із тегами, зібраних для дослідження SMS-спаму та небажаного контенту. Він містить один набір з 5574 повідомлень, позначених тегами як ham або spam.

На рисунку 2.8 показано приклад даних з датасету.

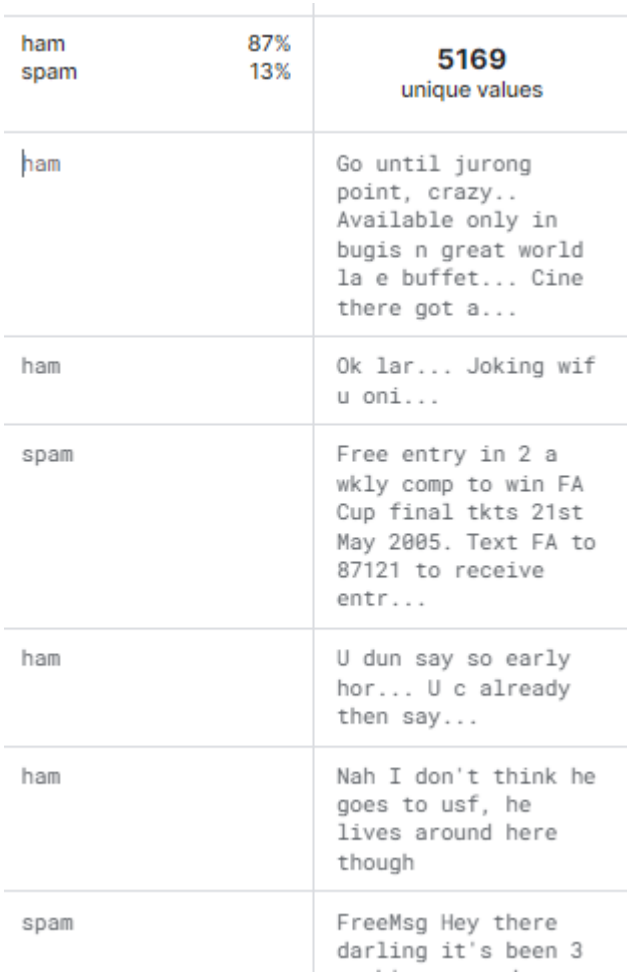

Рисунок 2.8 – Приклад даних з датасету «SMS Spam Collection Dataset».

Цей датасет було зібрано з вільних для дослідження джерел в Інтернеті: Колекція з 425 SMS-повідомлень зі спамом була вилучена вручну з веб-сайту Grumbletext. Це форум, на якому користувачі мобільних телефонів публічно заявляють про SMS-повідомлення зі спамом, більшість з яких не повідомляють про саме отримане спам-повідомлення. Ідентифікація тексту спам-повідомлень

у претензіях є дуже важким і трудомістким завданням, яке передбачало ретельне сканування сотень веб-сторінок..

Підмножина з 3375 випадково вибраних SMS-повідомлень NUS SMS Corpus (NSC), що є набором даних із приблизно 10 000 законних повідомлень, зібраних для дослідження на кафедрі комп'ютерних наук Національного університету Сінгапуру. Ці повідомлення були зібрані від волонтерів, яким було повідомлено, що їхні внески будуть оприлюднені.

Також, в датасет включено SMS Spam Corpus v.0.1 Big. Він містить 1002 SMS-повідомлення та 322 спам-повідомлення.

На рисунку 2.9 показано гістограму порівняння кількості слів у датасеті за категоріями.

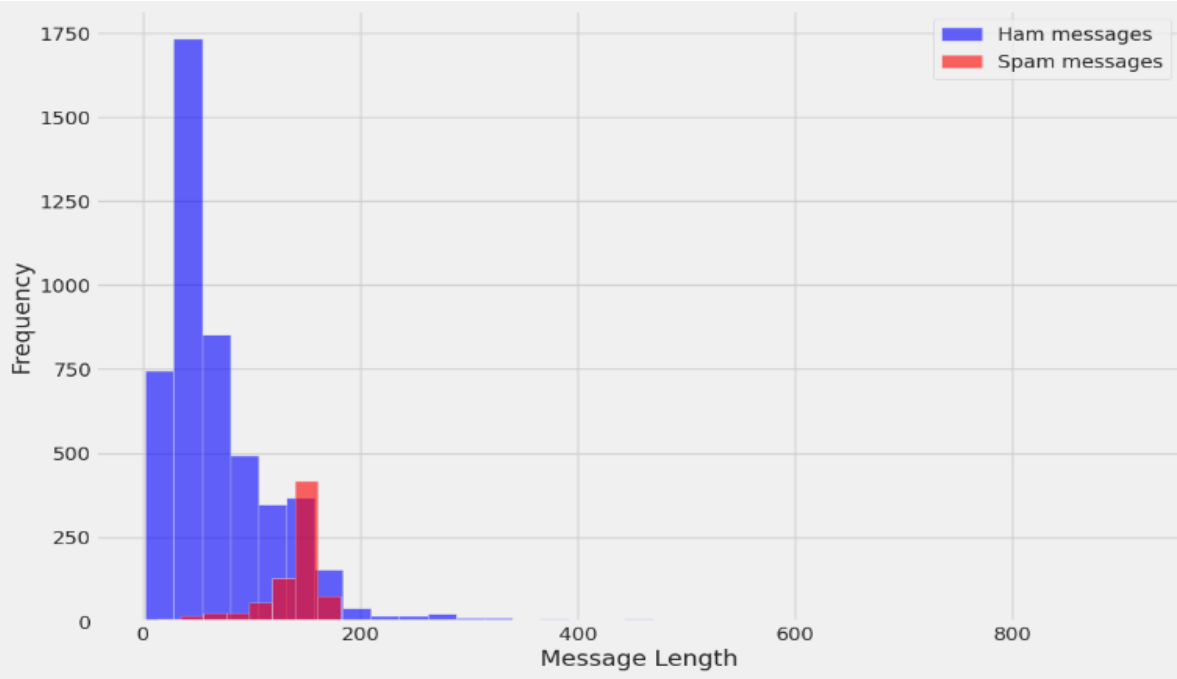

Рисунок 2.9 – Гістограма порівняння кількості слів у датасеті за категоріями.

Цей графік показує кількість спаму та безпечних SMS-повідомлень, відсортовану за довжиною повідомлення. На графіку видно, що спам-повідомлення, як правило, довші, ніж безпечні повідомлення. Це пов'язано з тим, що спамери хочуть, щоб їхні повідомлення були містили максимум інформації яку вони хочуть розповсюдити в якості спаму.

Графік також показує, що кількість спаму-повідомлень досягає піку при довжині повідомлення близько 200 символів. Це пов’язано з тим, що спамери часто використовують певні фрази та слова, які є характерними для спаму.

Перед навчанням моделі дані було попередньо оброблено та розбито на тренувальний та тестовий датасети.

На рисунку 2.10 показано алгоритм підготовки та розбиття датасету.

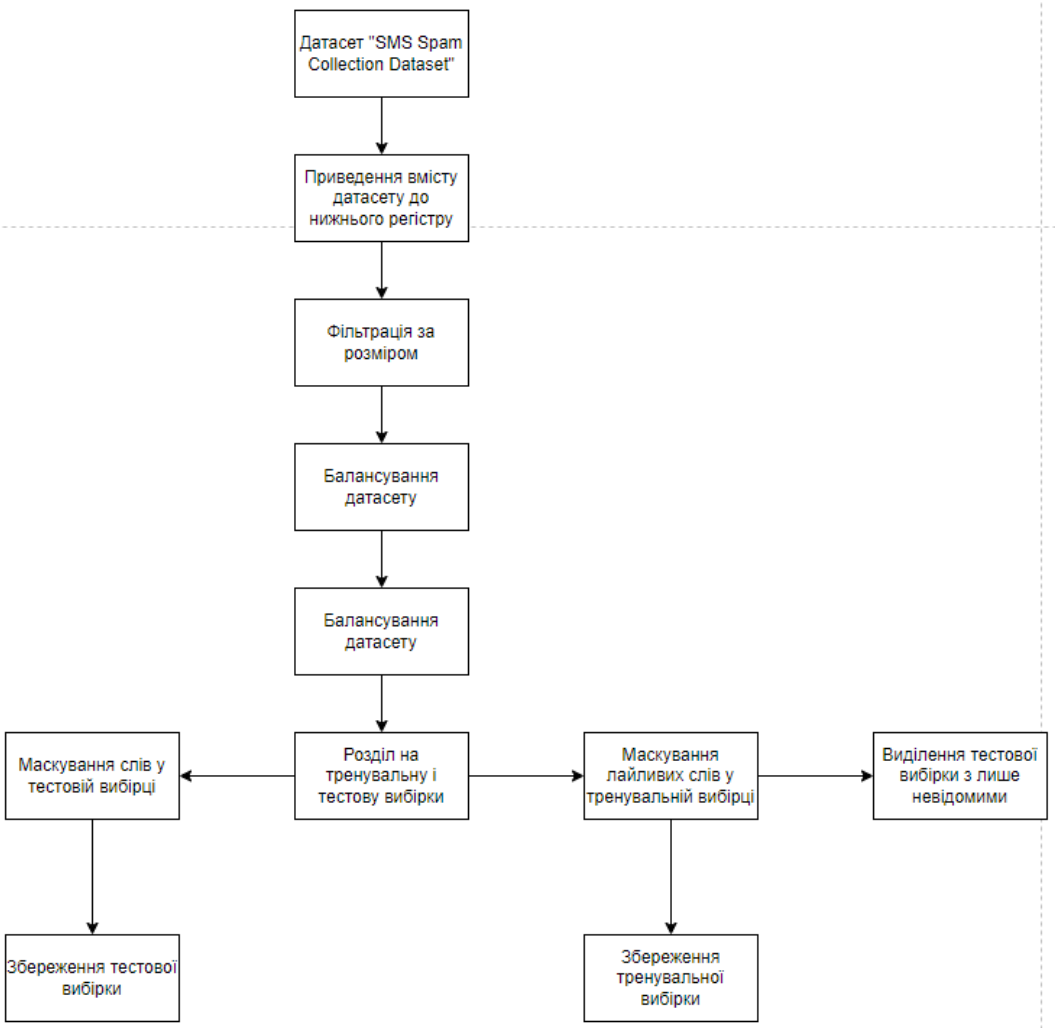

Рисунок 2.10 – Алгоритм підготовки та розбиття датасету.

Алгоритм, зображений на малюнку, є процесом підготовки даних для класифікації спаму. Він починається з завантаження набору даних SMS спаму та безпечних SMS.

Наступним кроком є приведення вмісту датасету до нижнього регістру. Це робиться для того, щоб всі слова в датасеті були однакової величини, що полегшує обробку даних.

Потім датасет фільтрується за розміром. Це робиться для видалення повідомлень, які занадто короткі або занадто довгі. Короткі повідомлення можуть бути спамом, але вони також можуть бути випадковими або помилковими. Довгі повідомлення можуть бути спамом, але вони також можуть бути діловими або особистими.

Після фільтрації за розміром датасет балансується. Це робиться для того, щоб кількість спаму та безпечних повідомлень у датасеті була однаковою. Незбалансований датасет може призвести до неточностей у класифікації.

Далі лайливі слова маскуються в тестовій вибірці. Це робиться для того, щоб уникнути упередженості класифікатора до певних слів.

Нарешті, датасет розділяється на тренувальну та тестову вибірки. Тренувальна вибірка використовується для навчання класифікатора, а тестова вибірка використовується для оцінки точності класифікатора.

Код побудови моделі машинного навчання LSTM з використанням мови програмування JavaScript показано на рисунку 2.11.

```
// Зчитуємо дані з CSV файлу
const rawData = fs.readFileSync('sms_spam_dataset.csv', 'utf-8');
const dataFrame = pandas.readCSV(rawData);

// Підготовка даних
const texts = dataFrame.get('text').toArray();
const labels = dataFrame.get('label').toArray();

// Конвертація текстових міток у числові (1 - spam, 0 - не spam)
const numericLabels = labels.map(label => label.toLowerCase() === 'spam' ? 1 : 0);

// Вибір параметрів моделі
const VOCAB_SIZE = 10000;
const EMBEDDING_DIM = 50;
const SEQUENCE_LENGTH = 100;

// Побудова LSTM моделі
const model = tf.sequential();
model.add(tf.layers.embedding({ inputDim: VOCAB_SIZE, outputDim: EMBEDDING_DIM, inputLength: SEQUENCE_LENGTH }));
model.add(tf.layers.lstm({ units: 50, returnSequences: true }));
model.add(tf.layers.flatten());
model.add(tf.layers.dense({ units: 1, activation: 'sigmoid' }));

// Компіляція моделі
model.compile({ optimizer: 'adam', loss: 'binaryCrossentropy', metrics: ['accuracy'] });

// Токенізація текстових даних та підготовка для подальшого використання

const tokenizedTexts = texts.map(preprocessText);
const paddedSequences = tokenizedTexts.map(sequence => padSequence(sequence, SEQUENCE_LENGTH));

const x_train = paddedSequences;
const y_train = numericLabels;

// Навчання моделі
model.fit(tf.tensor2d(x_train), tf.tensor1d(y_train), { epochs: 10, batchSize: 32, validationSplit: 0.2 })
  .then(info => {
    console.log('Навчання завершено:', info);
  });
```

Рисунок 2.11 – Побудова та навчання моделі з використанням тестових даних.

Далі, з використанням навченої моделі та бібліотеки TensorFlow було побудовано алгоритм для виявлення та фільтрації небажаного контенту у повідомленнях користувачів.

TensorFlow - це відкрита бібліотека для числових обчислень, спеціально розроблена для роботи з глибоким навчанням та машинним навчанням. Вона надає інструменти для побудови та навчання різноманітних моделей машинного навчання, включаючи нейронні мережі. TensorFlow використовує граф обчислень для представлення моделей та ефективно використовує обчислювальні ресурси, такі як процесори та GPU [37].

Код алгоритму показано на рисунку 2.12.

```
document.addEventListener('DOMContentLoaded', async () => {
  const chatMessages = document.getElementById('chat-messages');
  const messageForm = document.getElementById('message-form');
  const userMessageInput = document.getElementById('user-message');

  // Призначення події для форми
  messageForm.addEventListener('submit', async (event) => {
    event.preventDefault();

    const userMessage = userMessageInput.value;
    if (userMessage.trim() === '') {
      return;
    }

    // Класифікація повідомлення
    const classificationResult = await classifyMessage(userMessage);

    // Виведення результату у веб-чат
    const messageElement = document.createElement('div');
    messageElement.textContent = `Користувач: ${userMessage}`;

    if (classificationResult === 'Спам') {
      const warningMessage = document.createElement('div');
      warningMessage.textContent = 'Повідомлення містить небажаний контент, тому його відправку заборонено.';
      warningMessage.style.color = 'red';
      messageElement.appendChild(warningMessage);
    }

    chatMessages.appendChild(messageElement);
    userMessageInput.value = '';
  });
});
```

Рисунок 2.12 – Код фільтрації контенту з використанням навченої моделі.

Також, алгоритм було протестовано на тестовому датасеті та результати тестування підтвердили його коректну роботу та ефективність. Результати тестування на цих даних наведені на рисунку 2.13.

	precision	recall	f1-score	support
0	0.99	1.00	1.00	182175
1	0.99	0.70	0.82	3422
accuracy			0.99	185597
macro avg	0.99	0.85	0.91	185597
weighted avg	0.99	0.99	0.99	185597

Рисунок 2.13 – Результати моделі на частині датасету з невідомими лайливими словами

В результаті було отримано функціональний алгоритм фільтрації контенту з використанням машинного навчання.

2.7 Висновки

У даному розділі було досліджено архітектуру існуючих клієнт-серверних систем веб-месенджерів, визначено характеристики для оцінки успішності архітектури системи та здійснено вибір оптимальних технологій та СКБД для реалізації проекту. Також виконано вибір середовища розробки проекту та розроблено модуль фільтрації контенту з використанням машинного навчання.

Описані та розроблені в цьому розділі технології було використано у наступних розділах.

3 ПРОЕКТУВАННЯ МОДУЛІВ ФУНКЦІОНУВАННЯ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ

3.1 Загальний огляд функціональних модулів системи

Згідно із завданням на магістерську дипломну роботу система, що розробляється повинна мати графічний інтерфейс, модуль користувачів, чату, реєстрації та авторизації. Загальну структуру системи показано на рисунку 3.1.

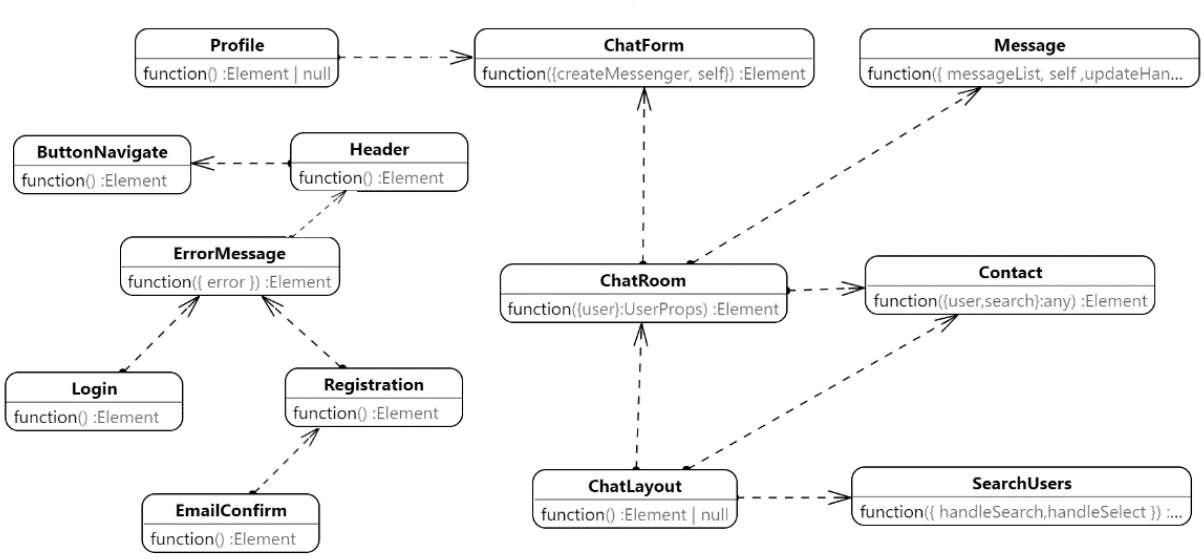

Рисунок 3.1 – Загальна структура системи.

Модулі організовані в три основні категорії:

Вхідна взаємодія - ці модулі відповідають за взаємодію з користувачем. Вони включають в себе модулі для відображення спливаючих вікон, форм і інших елементів інтерфейсу користувача [38].

Логіка чата - ці модулі відповідають за обробку повідомлень користувачів і відповідей чат-бота. Вони включають в себе модулі для обробки мови, генерації тексту та інших завдань, пов'язаних з чатом.

Вихідна взаємодія - ці модулі відповідають за відображення повідомлень чат-бота користувачам. Вони включають в себе модулі для відображення

повідомлень у чаті, відображення спливаючих вікон і інших елементів інтерфейсу користувача.

На рисунку 3.2 показано UML-flow діаграму діяльності для веб-месенджеру з автоматичною фільтрацією контенту.

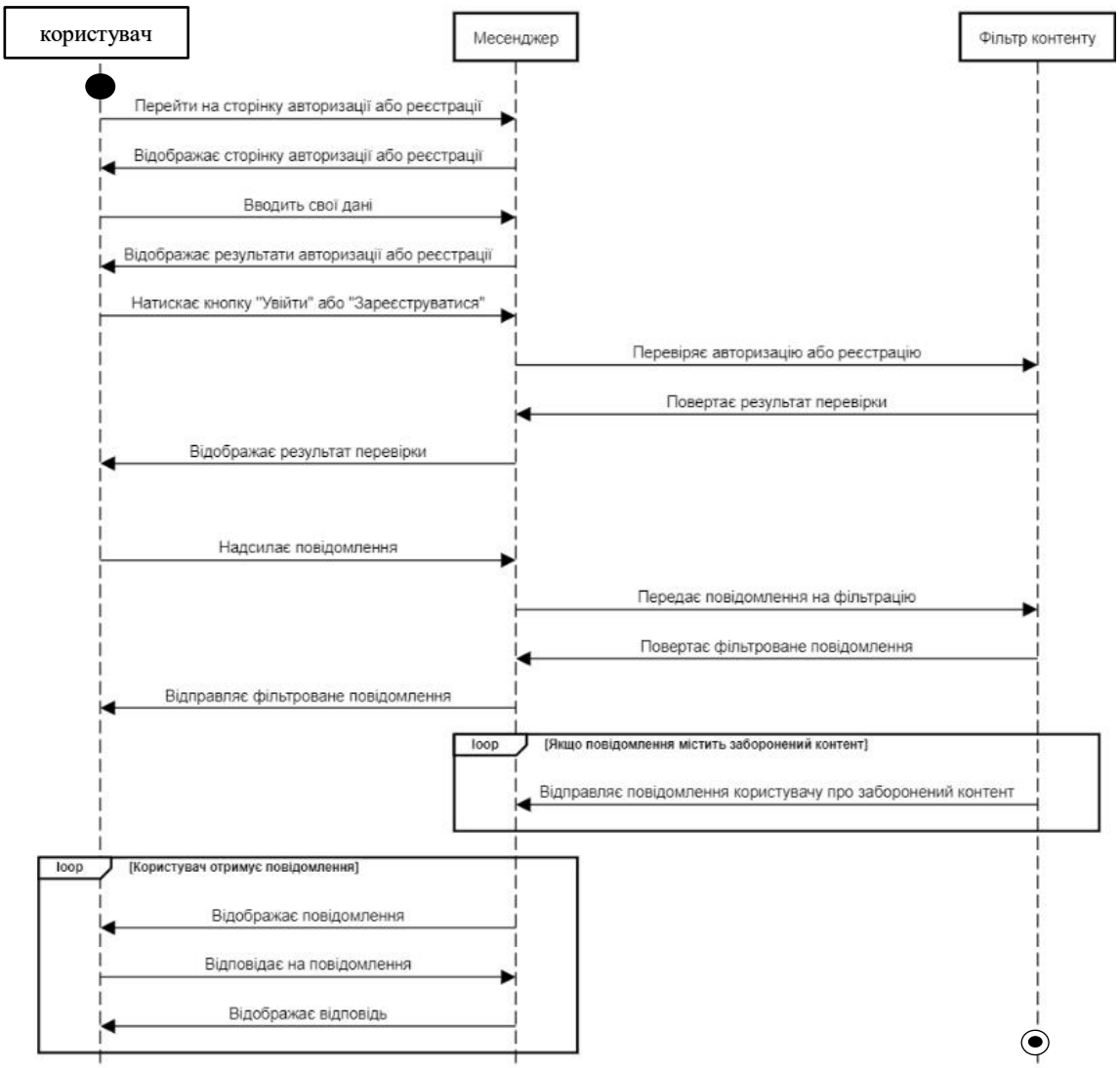

Рисунок 3.2 – UML-flow діаграма діяльності для веб-месенджеру з автоматичною фільтрацією контенту.

Розглянемо кожен модуль більш детально:

– Profile - цей модуль відповідає за відображення профілю користувача. Він включає в себе інформацію про ім'я користувача, його електронну адресу, номер телефону, а також інші відомості.

– ChatForm - цей модуль відповідає за відображення форми для відправки повідомлень чат-боту. Він включає в себе поля для введення тексту повідомлення, а також інші елементи управління, такі як кнопка відправки.

– Message - цей модуль відповідає за відображення повідомлень чат-бота користувачам. Він включає в себе текст повідомлення, а також інші елементи, такі як дата і час відправки, ім'я відправника та інші.

– ButtonNavigate - цей модуль відповідає за створення кнопки для навігації між різними частинами чата.

– Header - цей модуль відповідає за відображення заголовка чата. Він включає в себе назву чата, а також інші відомості, такі як логотип компанії або організації, яка створила чат.

– ErrorMessage - цей модуль відповідає за відображення повідомлення про помилку. Він може використовується для інформування користувачів про помилки, які виникли під час роботи чата.

– ChatRoom - цей модуль відповідає за відображення чат-кімнати. Він включає в себе список повідомлень, а також інші елементи, такі як поле для введення тексту повідомлення та кнопка відправки.

– Contact - цей модуль відповідає за відображення контактної інформації чат-бота. Він включає в себе адресу електронної пошти, номер телефону, а також інші контактні відомості.

– Login - цей модуль відповідає за реалізацію процесу входу в систему. Він вимагає від користувачів ввести ім'я користувача і пароль.

– Registration - цей модуль відповідає за реалізацію процесу реєстрації. Він вимагає від користувачів ввести ім'я користувача, пароль, а також іншу інформацію, таку як електронна адреса або номер телефону.

– ChatLayout - цей модуль відповідає за загальний макет чата. Він визначає розташування різних елементів чата на екрані.

– SearchUsers - цей модуль відповідає за реалізацію функції пошуку користувачів. Він використовується для пошуку користувачів за ім'ям, електронною адресою або іншими критеріями.

– EmailConfirm - цей модуль відповідає за реалізацію процесу підтвердження електронної адреси. Він вимагає від користувачів ввести код підтвердження, який був надісланий на їхню електронну адресу.

На рисунку 3.3 показано UML Use-case діаграму взаємодії користувача з веб-месенджером.

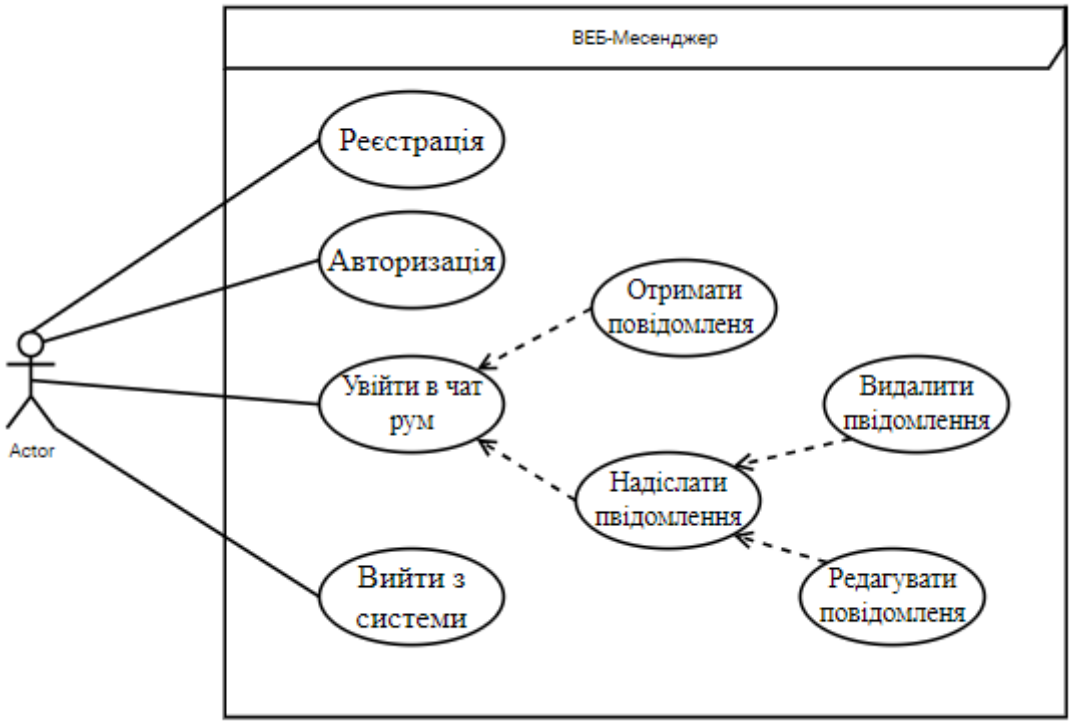

Рисунок 3.3 – UML Use-case діаграма взаємодії.

Ця діаграма ілюструє функціональність системи чату та взаємодію користувача з нею. Зокрема, вона визначає низку можливих дій, які користувач може виконати, починаючи від реєстрації в системі і закінчуючи виходом з неї. Важливим є те, що деякі функції доступні лише після виконання конкретних дій, вказуючи на послідовність взаємодій.

Реєстрація та авторизація слугують входом до системи, дозволяючи користувачеві використовувати її функціонал. Після цього користувач може увійти в чатрум і спілкуватися з іншими учасниками. Окрім загальних

повідомлень, в системі існує можливість особистого спілкування між користувачами через окремі повідомлення.

Деякі опції, такі як редагування та видалення повідомлень, вказують на можливість коригування власного контенту. Видалення повідомлень може бути важливим елементом контролю за інформацією в чатрумі.

3.2 Реалізація основних модулів функціонування сестеми

Першим на розгляді буде модуль системи авторизації. Даний модуль забезпечує процес аутентифікації та авторизації користувачів у системі.

Аутентифікація представляє собою процес верифікації особи користувача, що в системному контексті вирішується за допомогою локальної або JWT-стратегії. Локальна стратегія аутентифікації здійснює перевірку наявності користувача за вказаними електронною поштою та паролем. У разі підтвердження ідентичності видається токен доступу [39].

Спрощено JWT-стратегія аутентифікації розглядає валідність, представленого користувачем JWT-токену. Якщо токен підтверджується, як дійсний, відбувається ідентифікація користувача в системі.

Авторизація у цьому контексті описує процес надання користувачеві прав доступу до конкретних ресурсів. У даному випадку механізм авторизації реалізується через використання гвардіанів (гвардів доступу).

На рисунку 3.4 показано UML-діаграму розгортання для модуля авторизації.

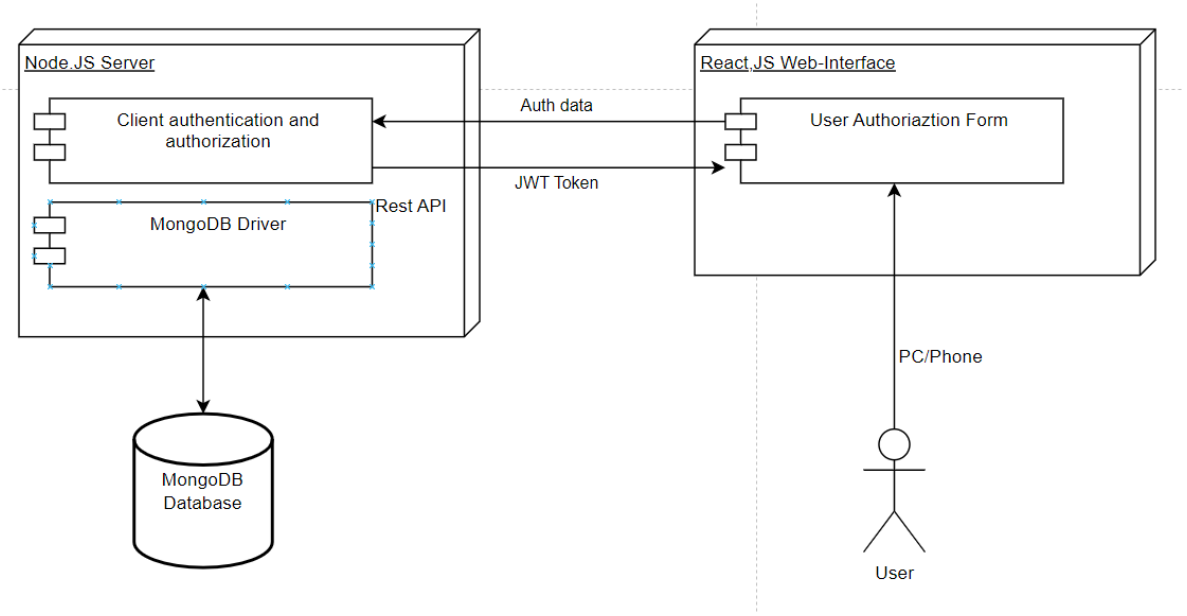

Рисунок 3.4 – UML-діаграма розгортання модуля авторизації.

Показана на рисунку 3.4 діаграма дозволяє прослідкувати зв'язок між модулями клієнт-серверного веб-месенджеру в процесі авторизації користувача.

На рисунку 3.5 показано UML-діаграму основних класів модуля авторизації.

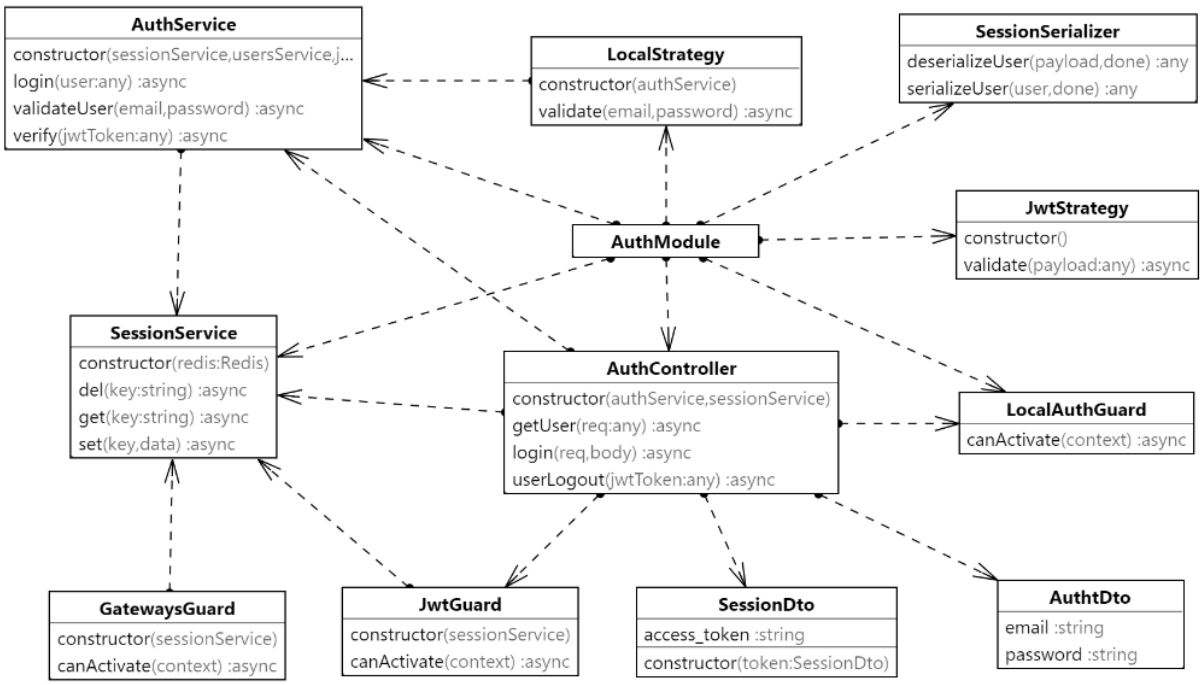

Рисунок 3.5 – UML-діаграма основних класів модуля авторизації.

LocalAuthGuard перевіряє, чи є користувач аутентифікованим. Якщо користувач не аутентифікований, то йому відмовлено в доступі до ресурсу.

JwtGuard перевіряє, чи є дійсним JWT-токен, який наданий користувачем. Якщо токен не дійсний, то йому відмовлено в доступі до ресурсу.

Основні взаємодії між компонентами модуля авторизації:

– Користувач надсилає запит на ресурс, який вимагає авторизації.

– AuthController отримує запит від користувача.

– AuthController передає запит AuthService.

– AuthService викликає локальну або JWT-стратегію для аутентифікації користувача.

– Локальна або JWT-стратегія повертає токен доступу, якщо користувач аутентифікований.

– AuthController повертає токен доступу користувачеві.

– Користувач надає токен доступу гварду.

– Гвард перевіряє дійсність токена доступу.

– Гвард дозволяє користувачеві отримати доступ до ресурсу, якщо токен доступу дійсний.

– SessionService відповідає за зберігання інформації про сеанси користувачів.

– SessionSerializer відповідає за серійну і десерійну інформацію про сеанси користувачів.

– AuthModule є модулем, який містить усі компоненти модуля авторизації.

– SessionDto і AuthtDto є DTO, які використовуються для передачі інформації про сеанси користувачів і аутентифікацію користувачів відповідно.

Далі, розглянемо структуру модуля модуля Messenger, який реалізує систему обміну повідомленнями. UML-діаграму основних класів модуля авторизації показано на рисунку 3.6.

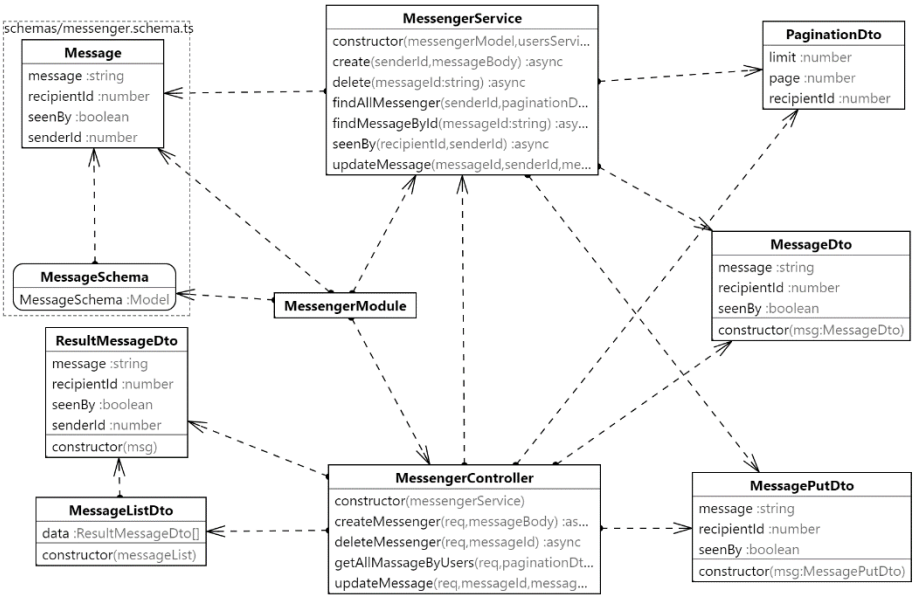

Рисунок 3.6 – UML-діаграма основних класів модуля Messenger.

Модуль складається з наступних компонентів:

MessengerService - це сервіс, який забезпечує основну функціональність системи обміну повідомленнями. Він включає такі методи:

– create(senderId, messageBody) - створює нове повідомлення

– delete(messageId) - видаляє повідомлення

– findAllMessenger(senderId, paginationDto) - повертає список всіх повідомлень для відправника

– findMessageByld(messageId) - повертає повідомлення за ідентифікатором

– seenBy(recipientId, senderId) - позначає повідомлення як прочитане для конкретного отримувача

– updateMessage(messageId, senderId, messageBody) - оновлює повідомлення

Message - це клас, який представляє повідомлення. Він має такі властивості:

– message - текст повідомлення
– recipientId - ідентифікатор отримувача
– senderId - ідентифікатор відправника
– seenBy - позначка про те, чи прочитане повідомлення

MessageDto` - це клас, який представляє повідомлення у вигляді об'єкта даних. Він має такі властивості:

– message - текст повідомлення
– recipientId - ідентифікатор отримувача
– senderId - ідентифікатор відправника

ResultMessageDto` - це клас, який представляє результат обробки повідомлення. Він має такі властивості:

– message - текст повідомлення
– recipientId - ідентифікатор отримувача
– senderId - ідентифікатор відправника
– seenBy - позначка про те, чи прочитане повідомлення

MessageListDto` - це клас, який представляє список повідомлень у вигляді об'єкта даних. Він має такі властивості:

– data - список об'єктів ResultMessageDto

PaginationDto` - це клас, який представляє параметри пагінації для обробки запитів до баз даних. Він має такі властивості:

– limit - кількість повідомлень на сторінці
– page - номер поточної сторінки

Діаграма також показує, що модуль Messenger залежить від модуля MessengerModel, який реалізує модель даних для повідомлень.

На рисунку 3.7 показано UML-діаграму основних класів модуля Users, який реалізує систему управління користувачами.

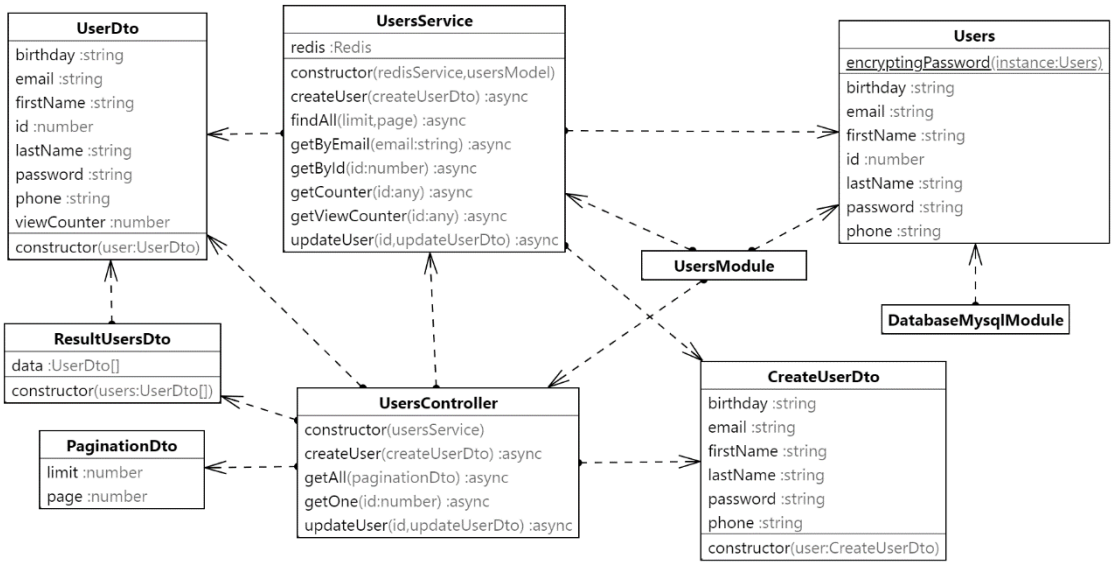

Рисунок 3.7 – UML-діаграма основних класів модуля Users.

Модуль складається з наступних компонентів:

UsersService - це сервіс, який забезпечує основну функціональність системи управління користувачами. Він надає такі методи:

– createUser(createUserDto) - створює нового користувача
– deleteUser(id) - видаляє користувача
– findAllUsers(limit, page) - повертає список всіх користувачів
– findUserById(id) - повертає користувача за ідентифікатором
– encryptPassword(instance) - шифрує пароль користувача
– updateUser(id, updateUserDto) - оновлює користувача

UserDto - це клас, який представляє користувача у вигляді об'єкта даних. Він має такі властивості:

– id - ідентифікатор користувача
– birthday - дата народження
– email - адреса електронної пошти
– firstName - ім'я
– lastName - прізвище
– password - пароль
– phone - номер телефону

UsersModel - це клас, який реалізує модель даних для користувачів. Він має такі властивості:

– id - ідентифікатор користувача
– birthday - дата народження
– email - адреса електронної пошти
– firstName - ім'я
– lastName - прізвище
– password - пароль
– phone - номер телефону

CreateUserDto - це клас, який представляє дані для створення нового користувача. Він має такі властивості:

– birthday - дата народження
– email - адреса електронної пошти
– firstName - ім'я
– lastName - прізвище
– password - пароль
– phone - номер телефону

ResultUsersDto - це клас, який представляє результат обробки запиту до модулю Users. Він має такі властивості:

– data - список об'єктів UserDto

PaginationDto - це клас, який представляє параметри пагінації для обробки запитів до баз даних. Він має такі властивості:

– limit - кількість користувачів на сторінці
– page - номер поточної сторінки

На рисунку 3.8 показано UML-діаграму основних класів модуля чат-рума.

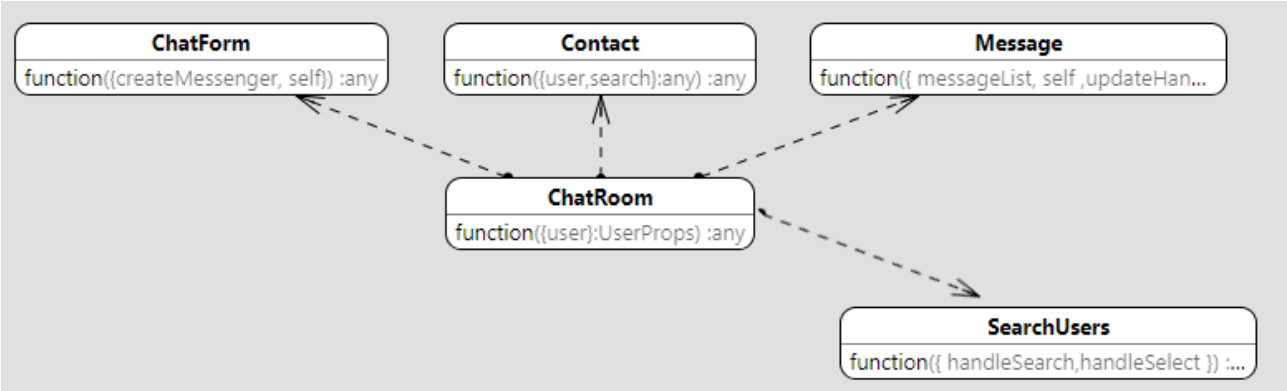

Рисунок 3.8 – UML-діаграма основних класів модуля ChatRoom.

Кожний компонент має свій власний функціонал. Компонент ChatForm відповідає за відображення чат-кімнати та обробку подій від елементів управління форми. Компонент Contact відповідає за виконання запитів пошуку користувачів. Компонент Message відповідає за зберігання повідомлень у чат-кімнаті. Компонент ChatRoom відповідає за загальне керування чат-кімнатою.

Структура модуля написана на JavaScript. Вона використовує функції та об'єкти для визначення поведінки компонентів.

Ось короткий опис функціоналу кожного компонента:

– ChatForm обробляє події від елементів управління форми.

– Contact виконує запити пошуку користувачів.

– Message зберігає повідомлення в чат-кімнаті.

– ChatRoom керує чат-кімнатою, відправляє повідомлення іншим користувачам

3.3 Висновки

В цьому розділі на основі аналізу існуючих реалізацій було розроблено структуру клієнт-серверної системи веб-месенджеру, що включає в себе основний інтерфейс, модуль користувачів, пошуковий модуль та модуль обміну повідомленнями. Також, було розроблено додаткові класи що містять в собі необхідний функціонал для забезпечення надійної роботи основних модулів.

4 РОЗРОБЛЕННЯ ІНСТРУКЦІЙ ДЛЯ ВСТАНОВЛЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Інструкція розгортання додатку для розробників

Інструкція розгортання програми для розробників забезпечує послідовні кроки для встановлення, налаштування та готовності програми до роботи в середовищі розробки. [40].

Ця інструкція розгортання для розробників детально описує процес встановлення та налаштування програмного продукту в середовищі розробки. Вона визначає всі необхідні технічні вимоги, вказівки щодо конфігурації та підготовки оточення, необхідного для ефективного використання програми.

Загалом, ця інструкція розгортання для розробників спрямована на забезпечення ефективного та безпечного процесу впровадження програмного забезпечення в середовищі розробки, забезпечуючи оптимальні умови для розробки та тестування.

Після цього можна запустити сервер за допомогою команди **npm run start** в терміналі Visual Studio Code для запуску NestJS. Потім можна запустити клієнтський додаток за допомогою команди **npm start**. Браузер автоматично відкриється і завантажить веб-додаток для чату.

Мінімальна конфігурація комп’ютера необхідна для роботи програми:

– Процесор: Двоядерний процесор з частотою більше 1.8 ГГц.

– Оперативна пам'ять (RAM): Мінімум 4 ГБ.

– Місце на жорсткому диску: 5ГБ.

– Операційна система: Підтримка Windows, macOS або Linux.

– Браузер: Сучасний веб-браузер, такий як Google Chrome, Mozilla Firefox, чи Safari.

4.2 Інструкція користувача

Інструкція користувача - це документ, що надає інформацію та вказівки щодо коректного використання продукту, послуги чи програмного забезпечення. Мета інструкції - забезпечити користувача необхідною інформацією та крок за кроком вказівками для ефективної взаємодії з продуктом. Цей документ може включати в себе вступну інформацію, вимоги до користування, установку, налаштування, використання, усунення проблем, питання безпеки, оновлення та підтримку, а також додаткову інформацію. Формати інструкцій різноманітні - паперові документи, електронні файли, відеоролики чи веб-сторінки [41].

Інструкція користувача має на меті допомогти користувачеві максимально ефективно використовувати надані йому ресурси чи засоби. Вона може містити різноманітні вказівки, починаючи від основних кроків для початку роботи і закінчуючи детальними технічними аспектами.

Важливо, щоб інструкція була чіткою та легкою для розуміння, оскільки її цільова аудиторія може бути різноманітною за рівнем технічної підготовки. Окрім того, інструкція може включати інформацію щодо оптимальних практик, порад з вирішення проблем, а також рекомендації з підтримки та поновлення.

В сучасному світі інструкції часто надаються у різних форматах, щоб відповідати різним потребам користувачів. Це може бути друкований документ, електронний файл, інтерактивна веб-сторінка чи навіть відеоурок. Незалежно від формату, головною метою залишається забезпечення ефективного та безпечного використання продукту чи послуги.

Першим кроком при ініціалізації системи необхідно відкрити один з сучасних браузерів, таких як Google Chrome, Mozilla Firefox або Microsoft Edge.

Наступним кроком необхідно ввести URL-адресу http://localhost:3000/login.jsp у вікні адресної стрічки та натискайте "Enter". Після цього в браузері буде відображено головне вікно веб-системи, де необхідно використати свої логін деталі для авторизації або зареєструватись як новий користувач.

На рисунку 4.1 показано вікно авторизації у веб-систему. На рисунку 4.2 показано вікно реєстрації.

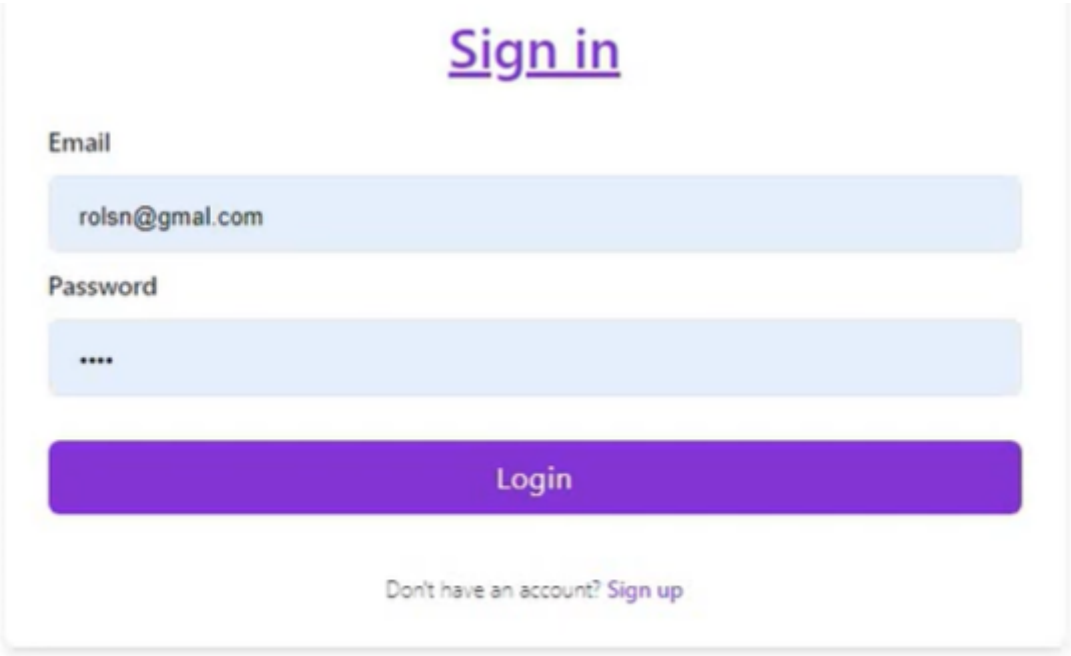

Рисунок 4.1 – Форма авторизації у систему

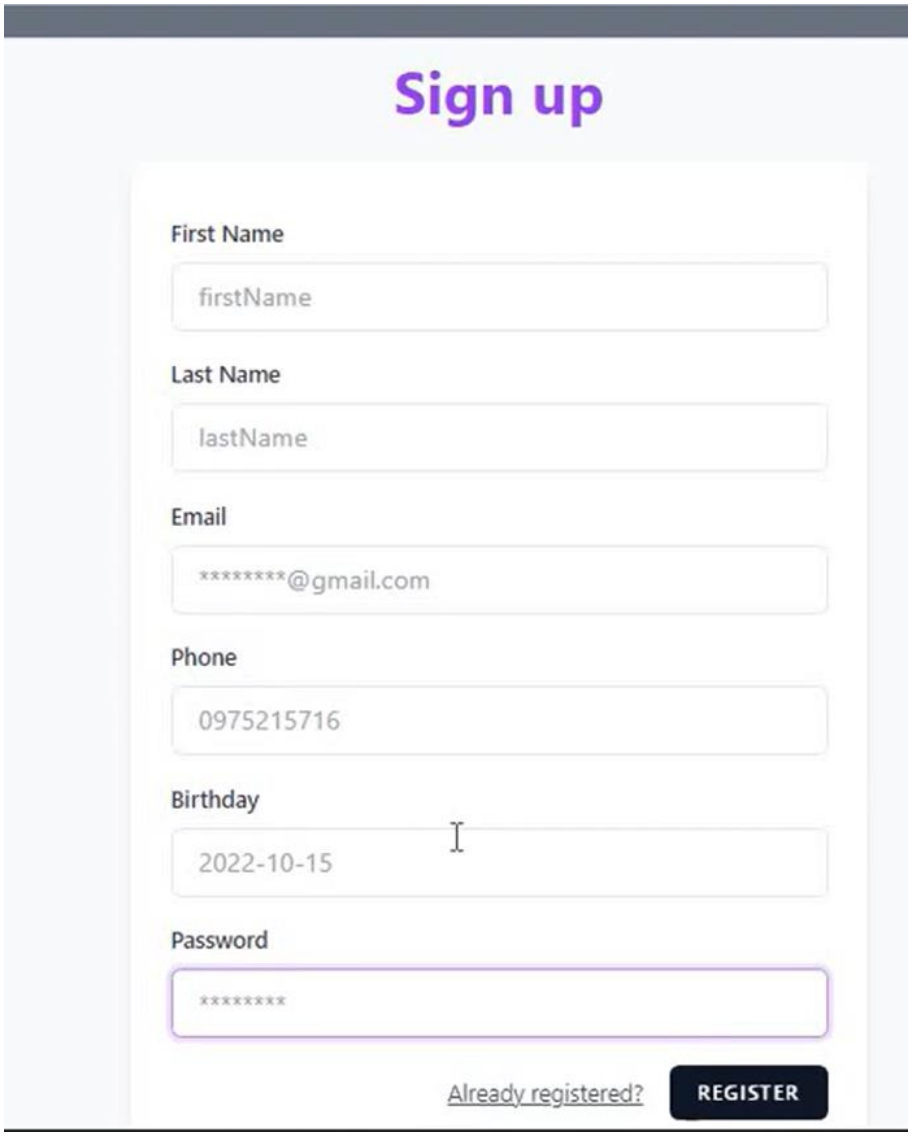

Рисунок 4.2 – Вікно реєстрації веб-системи.

Для реєстрації у системі необхідно вказати коректні дані, такі як ім’я, прізвище, електронну пошту, номер телефону, дату народження та пароль.

Також, на формі реєстрації присутня кнопка «Already registered?», у випадку якщо у користувача вже є акаунт, ця кнопка перенаправить його на сторінку авторизації.

Після авторизації, користувача буде перенаправлено на сторінку власного профілю. Приклад такої сторінки показано на рисунку 4.3.

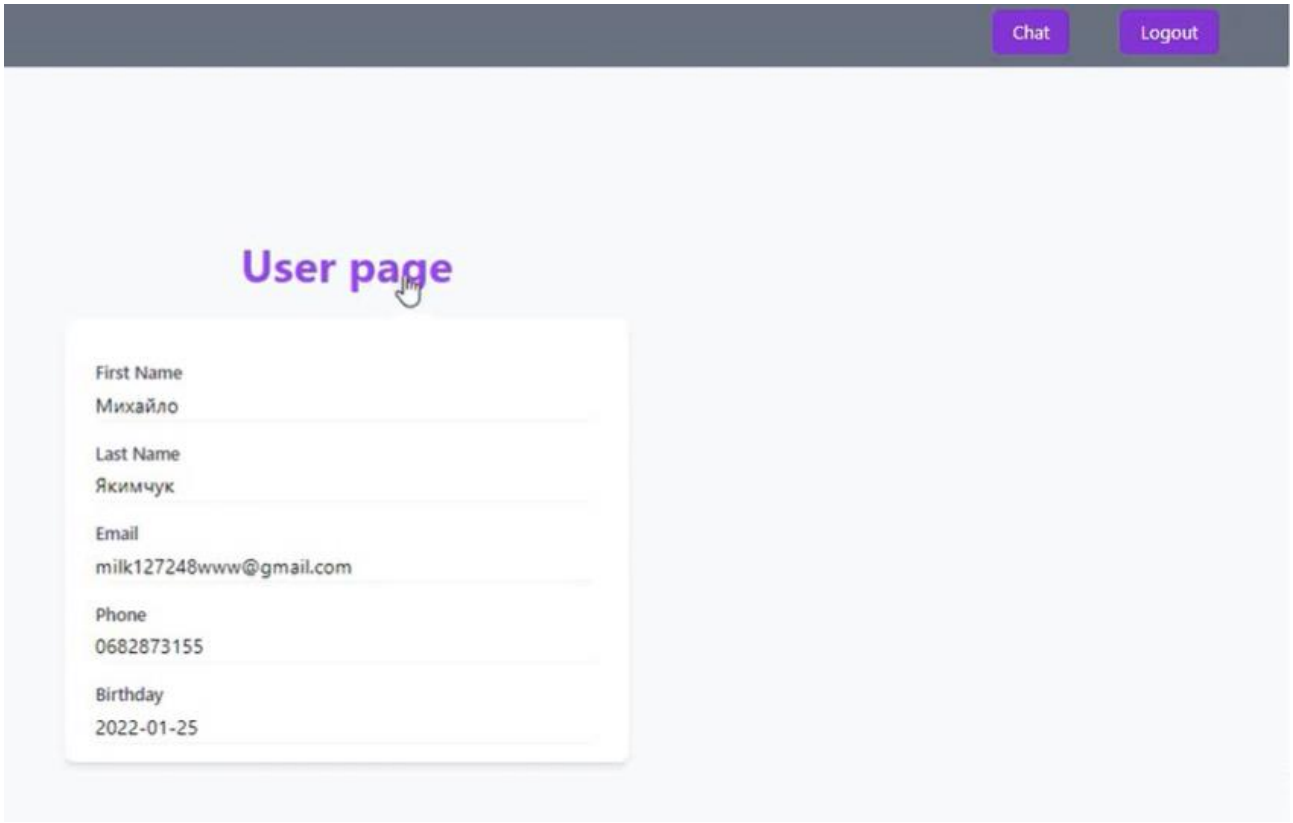

Рисунок 4.3 – Сторінка користувача.

Сторінка користувача - це веб-сторінка, яка дозволяє користувачам вводити свою особисту інформацію. Сторінка включає в себе інформацію про ім’я, прізище, адреса електронної пошти, номер телефона та дата народження.

Сторінка користувача, показана на рисунку 4.3, також має кнопки «Chat» і «Logout» які відповідають за вхід у чат та вихід з системи.

При натисканні на кнопку «Chat» користувача буде перенаправлено на сторінку з його активними чатами (рисунку 4.4).

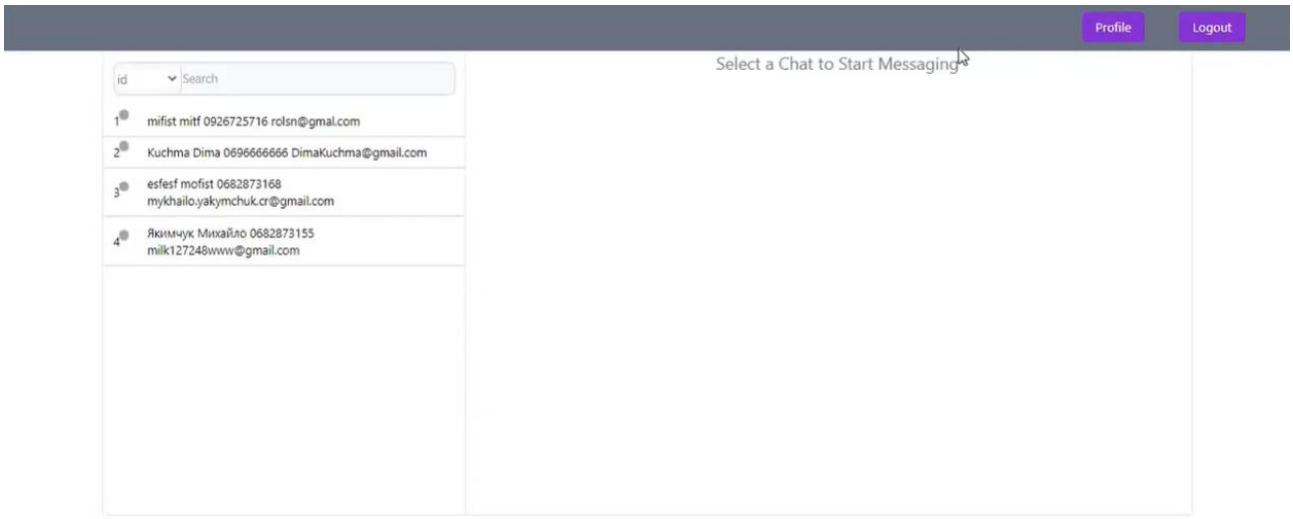

Рисунок 4.4 – Список активних чатів.

На сторінці чатів можна обрати один з активних чатів або ж додати новий, за допомогою пошуку користувачів за номером у полі «Search». Пошук можна проводити за такими атрибутами: айді, ім’я, прізвище, телефон, електронан пошта.

Приклад пошуку користувачів показано на рисунку 4.5.

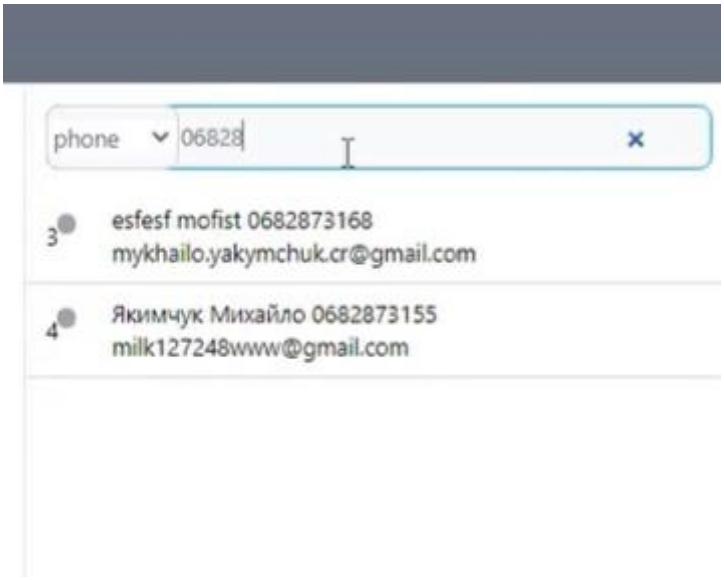

Рисунок 4.5 – Приклад успішного пошуку за номером телефону.

Після того як необхідний користувач був знайдений, можна почати чат. Для цього необхідно написати перше повідомлення, як показано на рисунку 4.6.

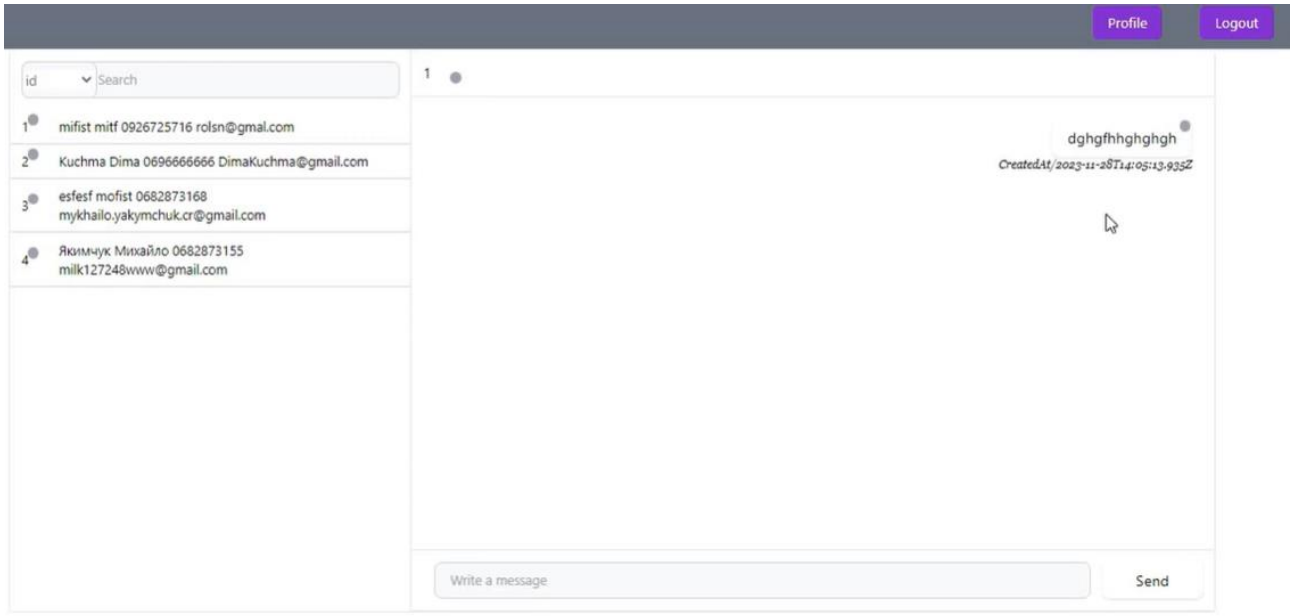

Рисунок 4.6 – Відкритий чат з надісланим повідомленням.

Після успішного відправлення повідомлення надається можливість видалити або відредагувати його в майбутньому. Це дозволяє виправити помилки, доповнити інформацію або адаптувати вміст відповідно до зміни обставин.

Вікно редагування та видалення повідомлення показано на рисунку 4.7.

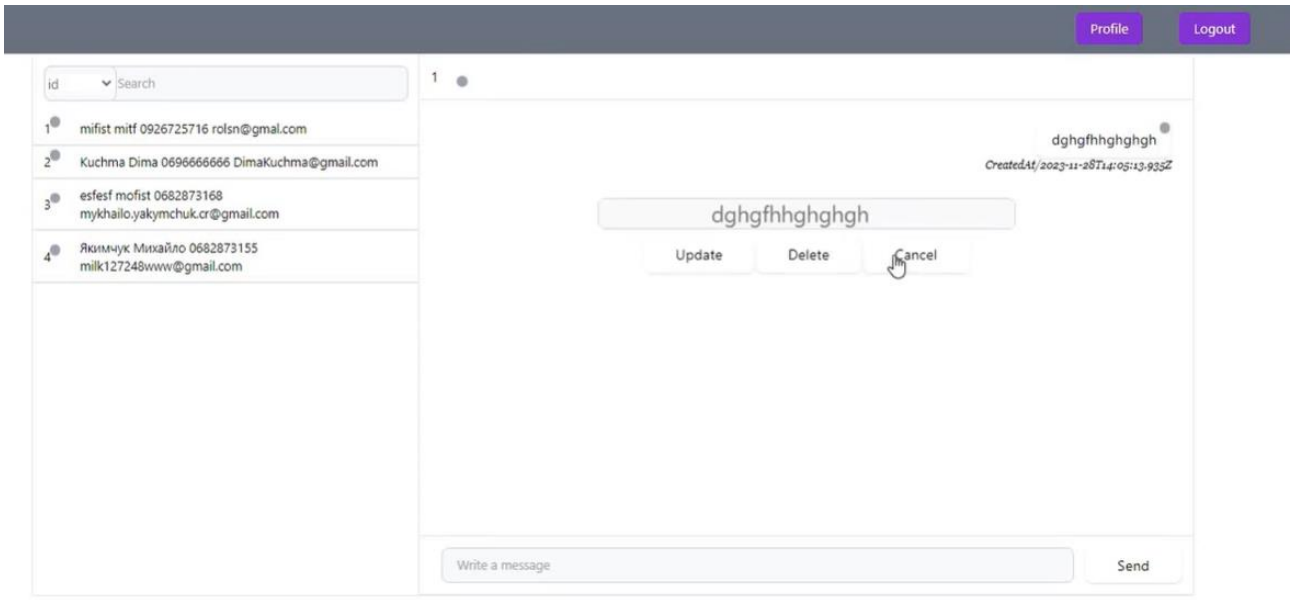

Рисунок 4.7 – Вікно для редагування або видалення повідомлення.

Також, необхідно враховувати що розроблена інформаційна ситема має вбудовано фільтрацію контенту. Для того щоб перевірити коректність її роботи, в словник нецензурних слів було додано слово «нецензурно» і при відправці повідомлення що містить це слово відправнику буде висвітлюватись повідомлення про помилку.

Приклад такої перевірки показано на рисунку 4.8.

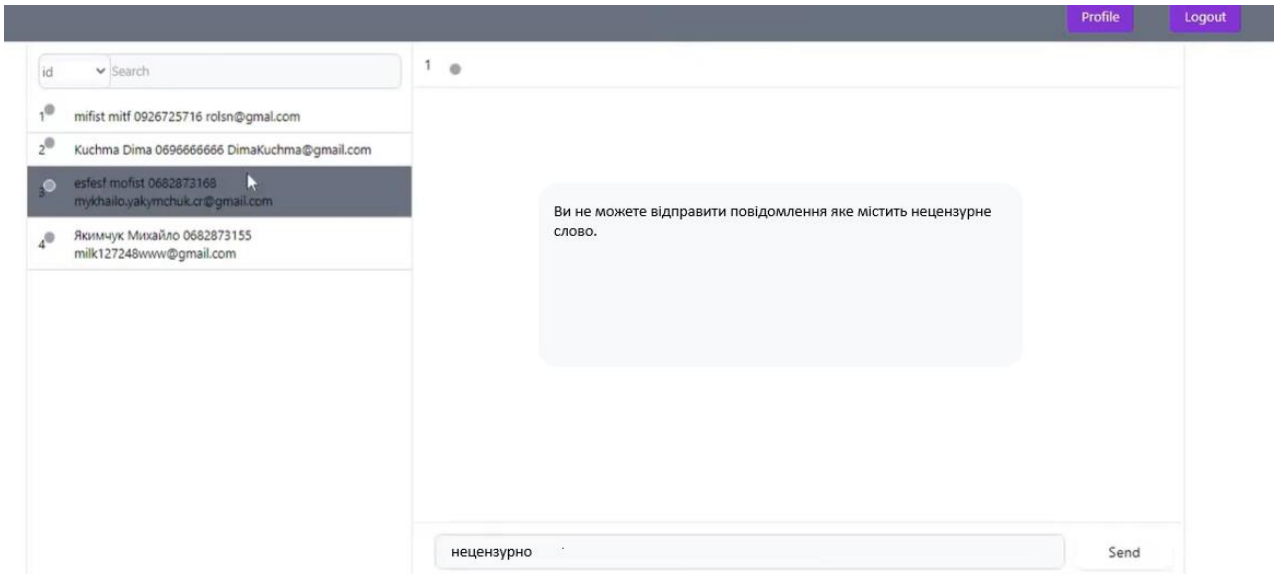

Рисунок 4.8 – Приклад фільтрації контенту.

Використання фільтрації контенту дозволяє ефективно контролювати надходження повідомлень, що містять неприпустимий вміст. Додатково, система виявляється надійною, оскільки при виявленні нецензурного контенту у повідомленні відправнику автоматично відображається повідомлення про помилку. Це свідчить про високий рівень точності та швидкодії фільтрації, сприяючи покращенню загального взаємодії та комунікації на платформі.

4.3 Тестування розробленої клієнт-серверної системи

Тестування методом чорної скриньки (Black Box Testing) - це метод тестування програмного забезпечення, коли тестувальник працює з програмою як з "чорною скринькою", не знаючи внутрішньої структури чи реалізації. Основна мета - перевірити функціональність програми.

Під час проведення тестування було прийнято рішення використовувати Swagger - інструмент для створення, документування та тестування API. Swagger надає специфікації API у форматі OpenAPI та генерує автоматичну документацію. Використання інтерактивної платформи Swagger UI спрощує вручне тестування ендпоінтів API у браузері, підвищуючи доступність та якість програмного забезпечення

Для успішності проекту важливо щоб як кожна окрема його частина, так і всі вони в сукупності працювали без будь-яких помилок. Для цього було перевірено кожен з розроблених модулів.

На рисунку 4.9-4.10 показано перевірку можливості успішної реєстрації нового користувача з валідними даними.

Рисунок 4.9 – Встановлення вхідних даних для тестування модулю реєстрації.

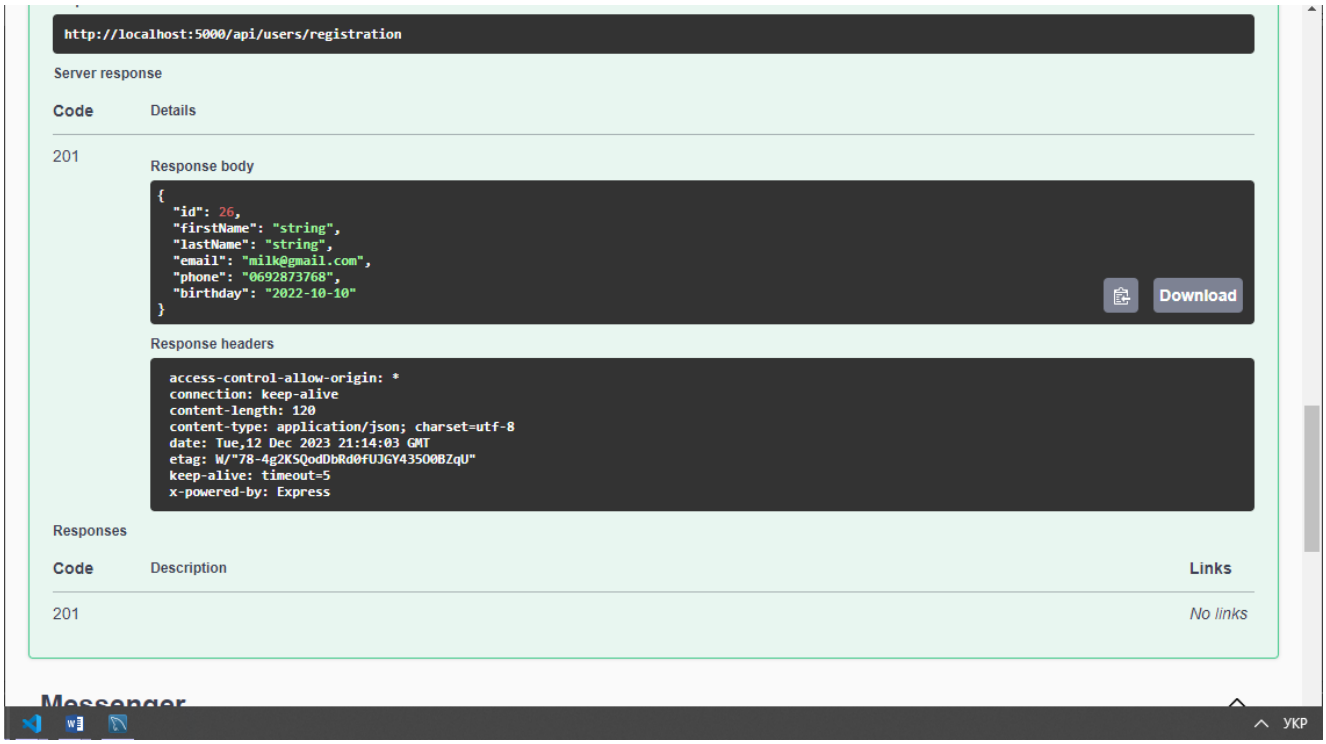

Рисунок 4.10 – Результати тестування модулю реєстрації.

Як показано на рисунку 4.9, результат тестування є успішним та з використанням валідних даних було успішно зареєстрованого нового користувача. Але для того щоб робити висновки про працездатність системи, важливо також протестувати можливість реєстрації використовуючи невалідні дані. Введення невалідних даних та результати тестування показано на рисунку 4.11-4.12.

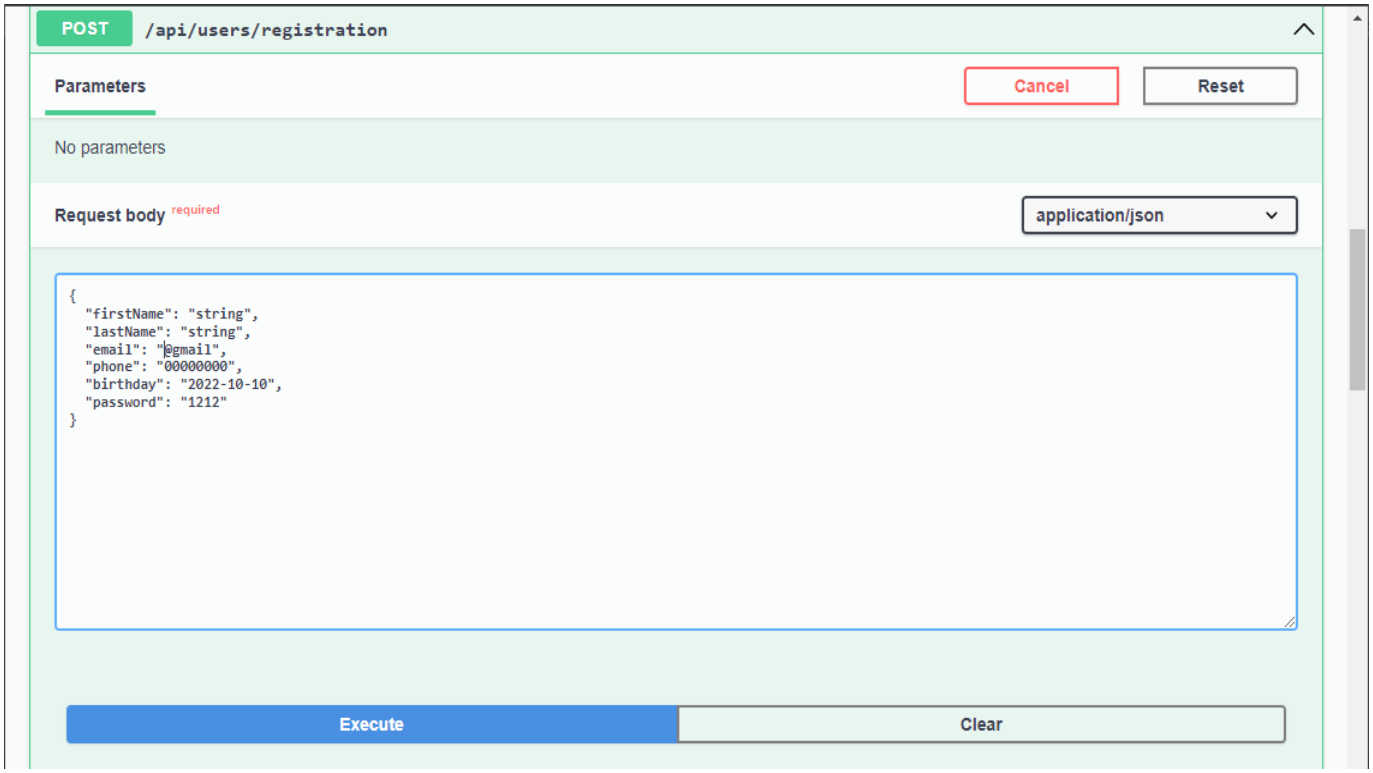

Рисунок 4.11 – Введення невалідних даних для тестування модулю реєстрації.

На рисунку 11 показано приклад використання невалідних даних, було спеціально використано неправильний формат для кожного поля, включаючи пошту, ім’я та прізвище, пароль який не відповідає мінімальним вимогам та номер телефону в якому відсутній код будь-якої країни.

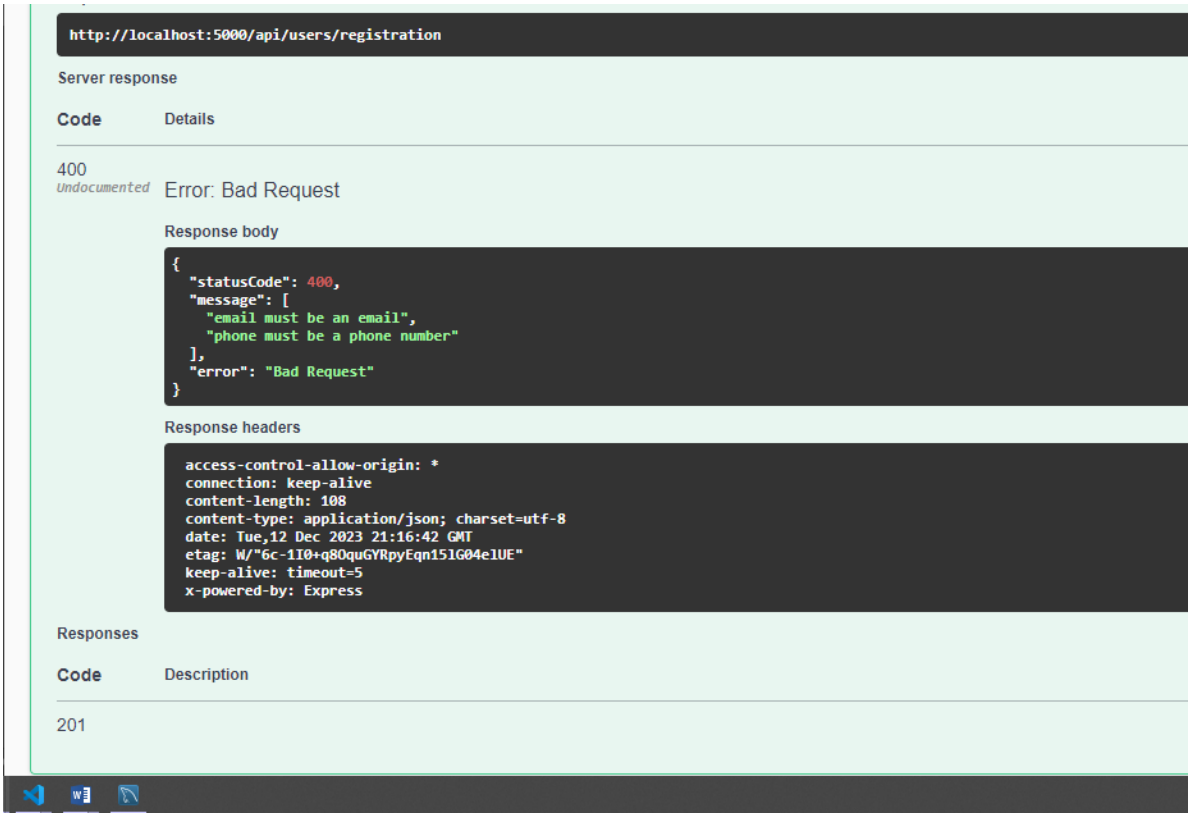

Рисунок 4.12 – Результат тестування модулю реєстрації з невалідними даними.

Як показано на рисунку 4.12, при використанні невалідних даних під час реєстрації сервер надав відповідь що необхідно використати коректний формат для введення пошти та номеру мобільного телефону.

Також, після тестування модулю автентифікації, важливо протестувати модулі авторизації та виходу з системи. На рисунку 4.13 показано приклад використання даних зареєстрованого користувача які будуть використовуватись для тестування.

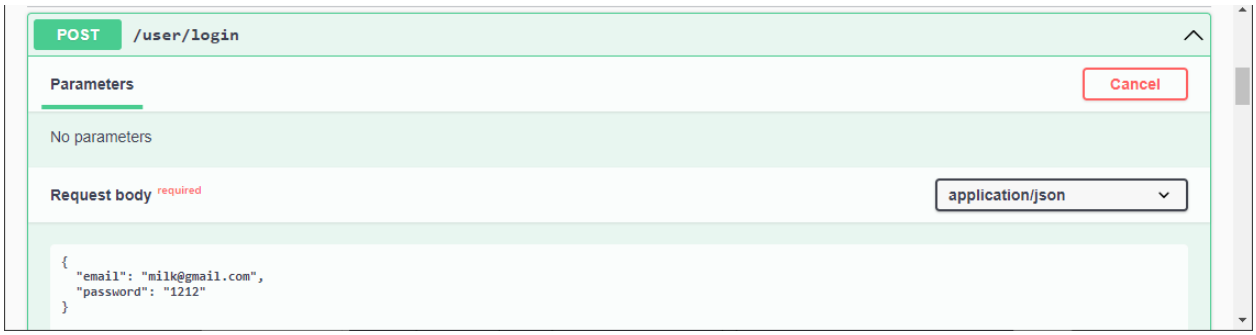

Рисунок 4.13 – Використання коректних даних для авторизації.

Як можна бачити на рисунку 4.14, від сервера було отримано відповідь що токен прийнято, записано в header та збережено в cookie, що надає користувачеві доступ до методів чату та власний профіль.

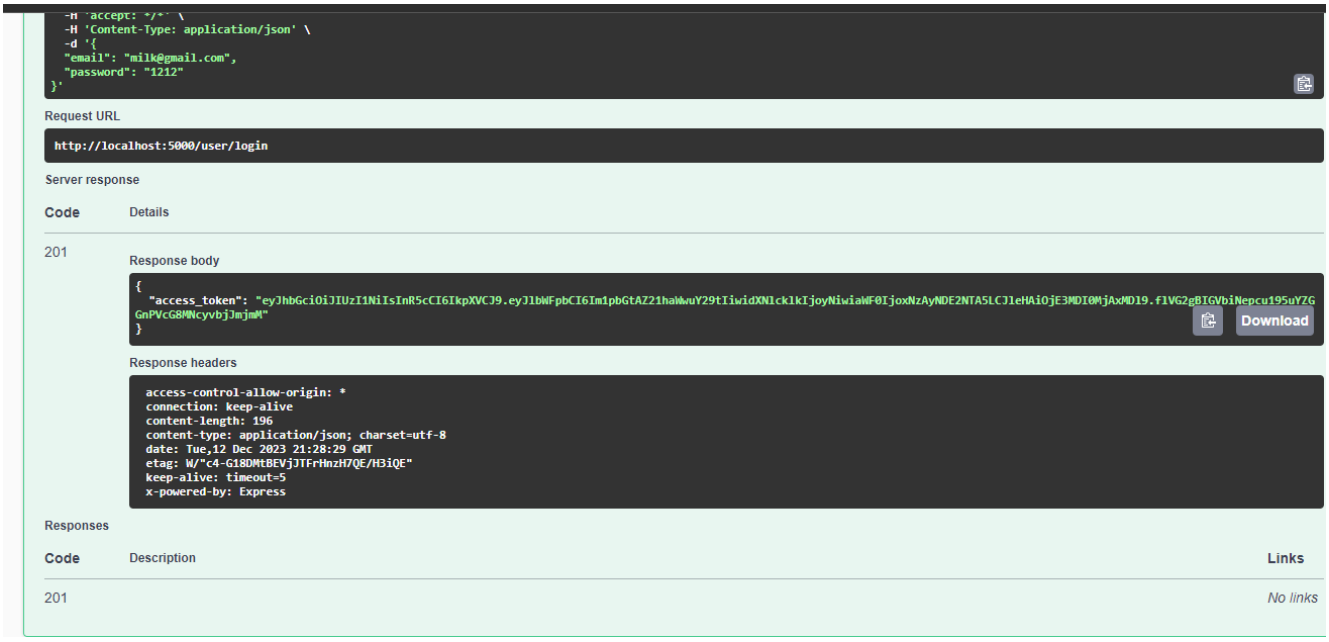

Рисунок 4.14 – Результат тестування авторизації з валідними даними.

На рисунку 4.15 показано налаштування тесту для функції виходу з системи авторизованим користувачем.

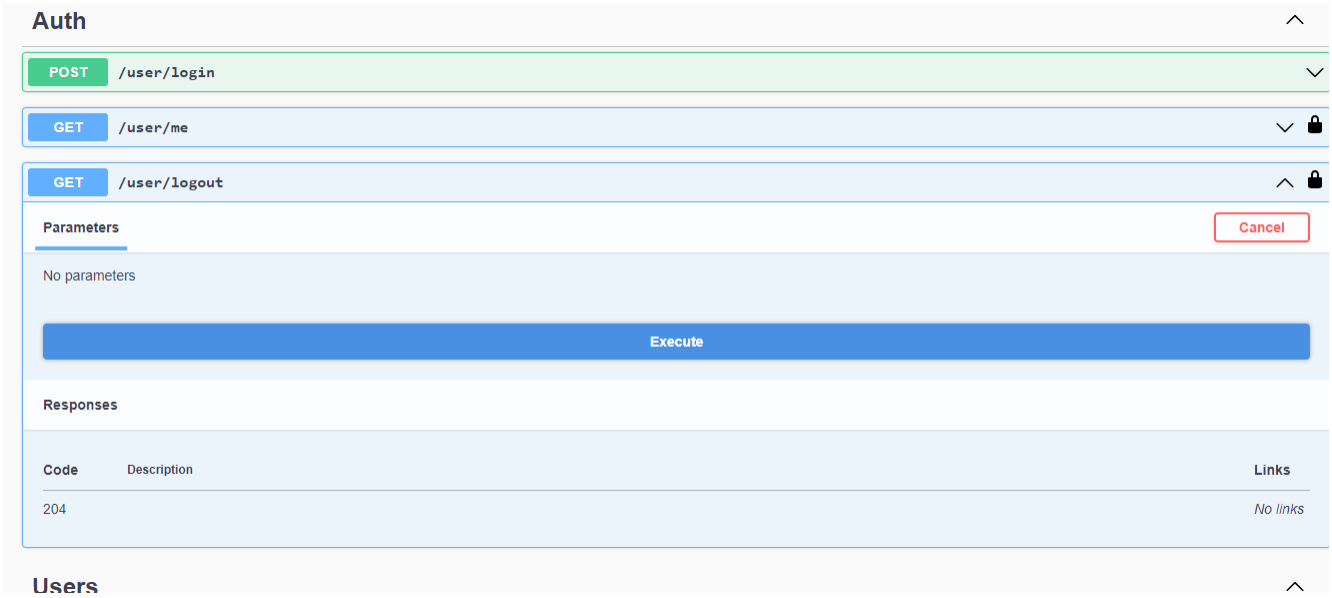

Рисунок 4.15 – Налаштування тесту для функції виходу з системи.

На рисунку 4.16 показано результат проведеного тесту функції виходу з системи. Як можна бачити, вихід з системи було проведено успішно та користувача було успішно перенаправлено на сторінку авторизації.

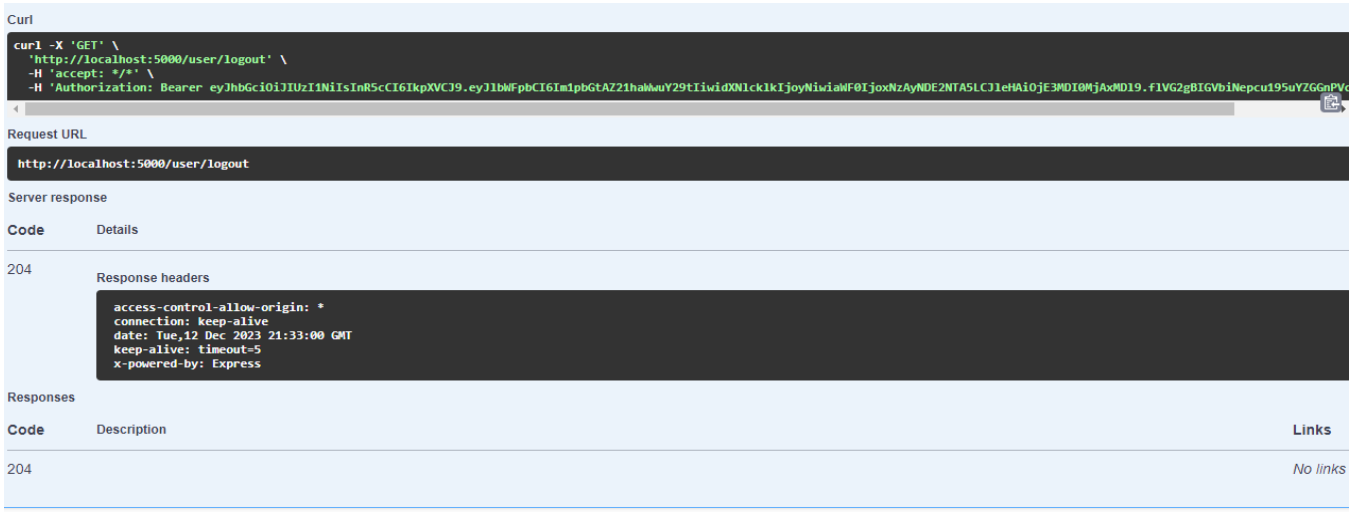

Рисунок 4.16 – Результат проведеного тесту.

Також, важливо було протестувати чи може неавторизований користувач скористатись функціоналом чату, наприклад написати повідомлення. Як можна бачити на рисунку 4.17, цей функціонал для нього закритий.

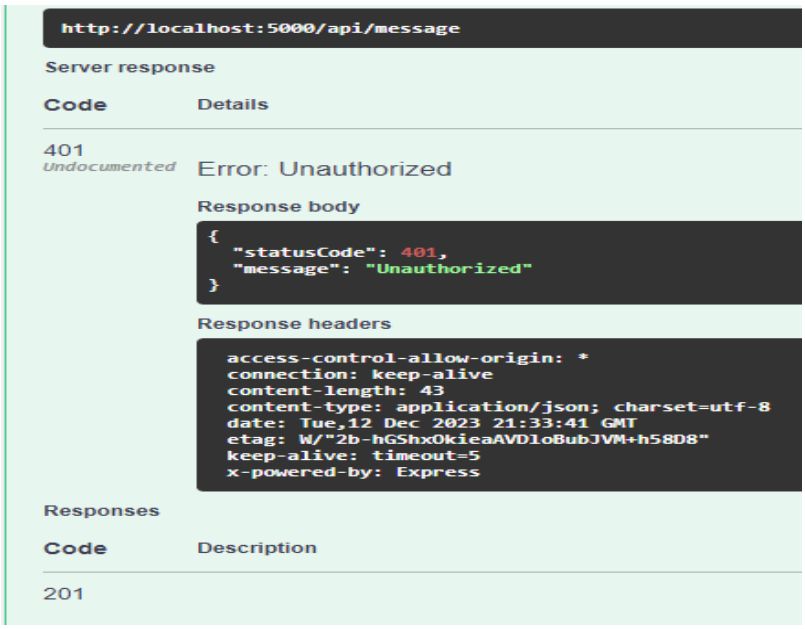

Рисунок 4.17 – Тестування виходу з системи для неавторизованого користувача.

Оскільки при авторизації користувач повинен вводити коректні логін та пароль, у випадку коли вони не співпадають з існуючими, у авторизації повинно бути відмовлено. Для цього, на рисунку 4.18-4.19 показано тестування модулю авторизації з використанням невірних даних.

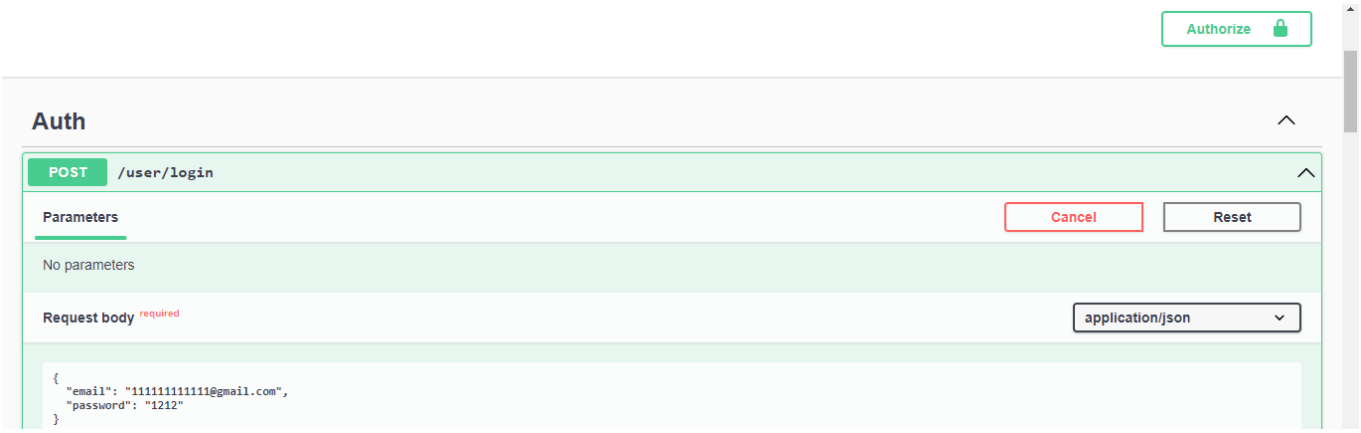

Рисунок 4.18 – Використання некоректних даних для авторизації.

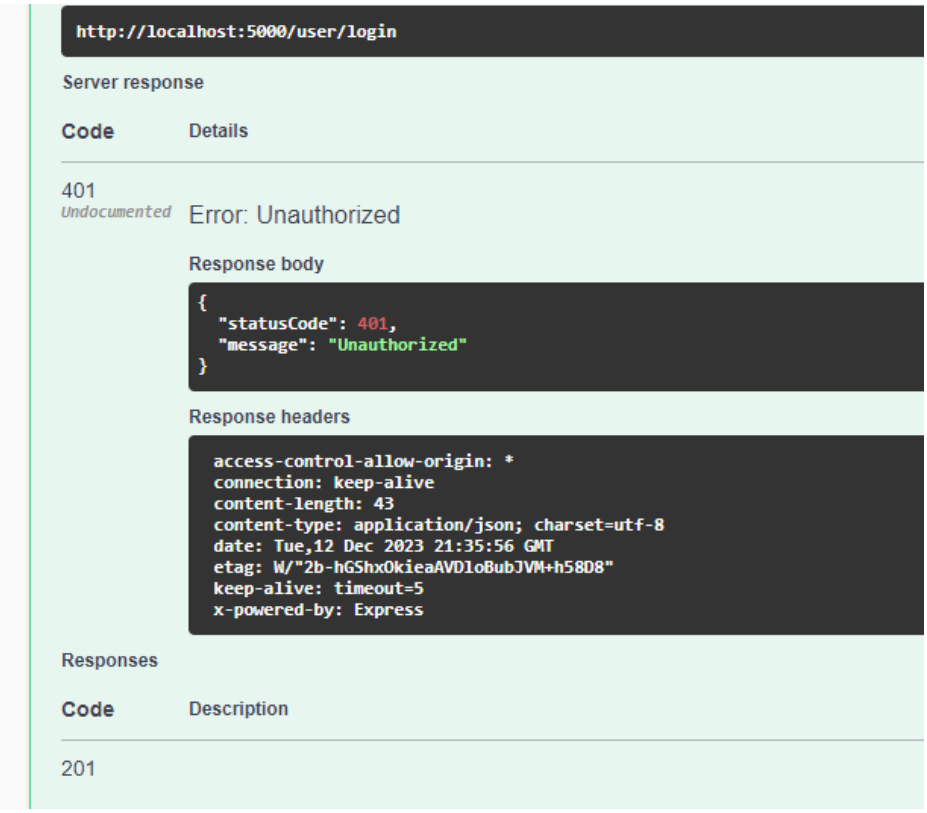

Рисунок 4.19 – Результат тестування модулю.

Як можна бачити на рисунку 4.19, при використанні невірних даних для авторизації сервер буде повертати повідомлення «Unauthorized», що означає що користувача не було авторизовано.

Важливим етапом є тестування модулю для пошуку користувачів за його унікальними даними, такими як ім’я, електронна пошта, тощо. На рисунку 4.20-4.21 показано введення коректного ідентифікатора користувача для пошуку та результат пошуку за цим ідентифікатором.

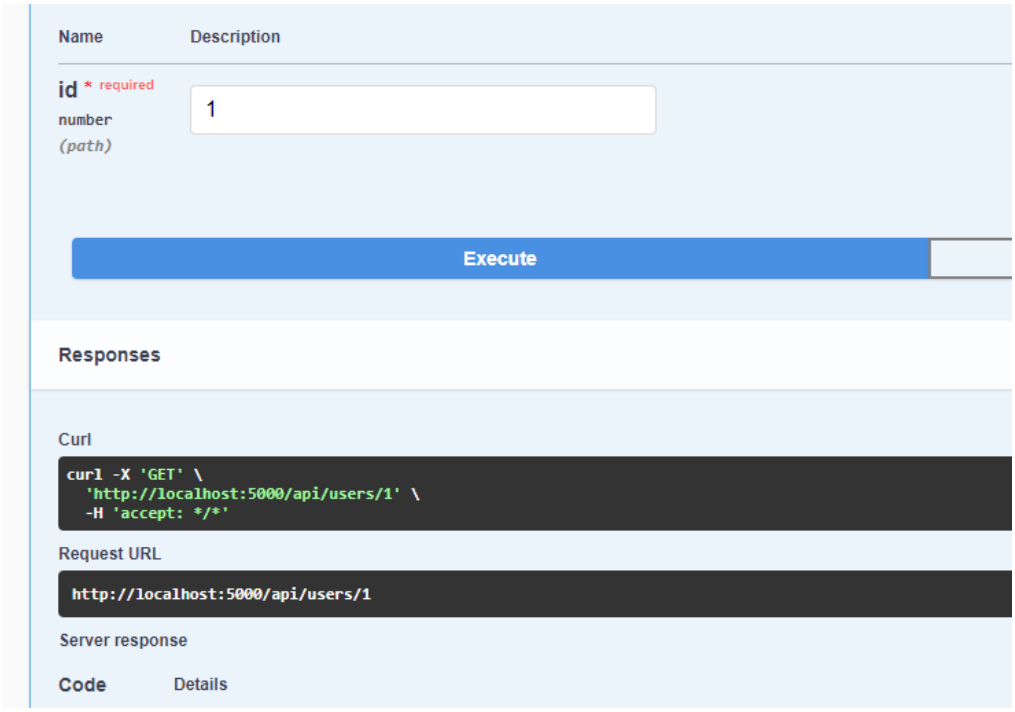

Рисунок 4.20 – Введення коректного ідентифікатору для пошуку.

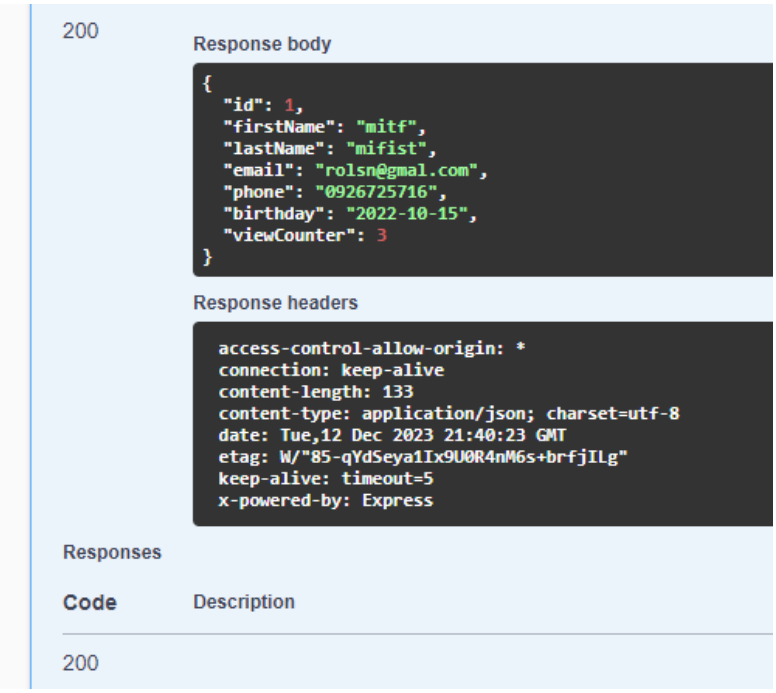

Рисунок 4.21 – Результат тестування модулю пошуку користувача.

Як можна бачити на рисунку 4.21, користувача було успішно знайдено та сервер повернув всі дані його профілю.

Оскільки дуже важливим є правильна робота модулю фільтрації контенту, необхідно також протестувати і його. На рисунку 4.22 показано результат тестування модулю фільтрації в ситуації коли користувач хоче відправити повідомлення що включає в себе недопустимий вміст.

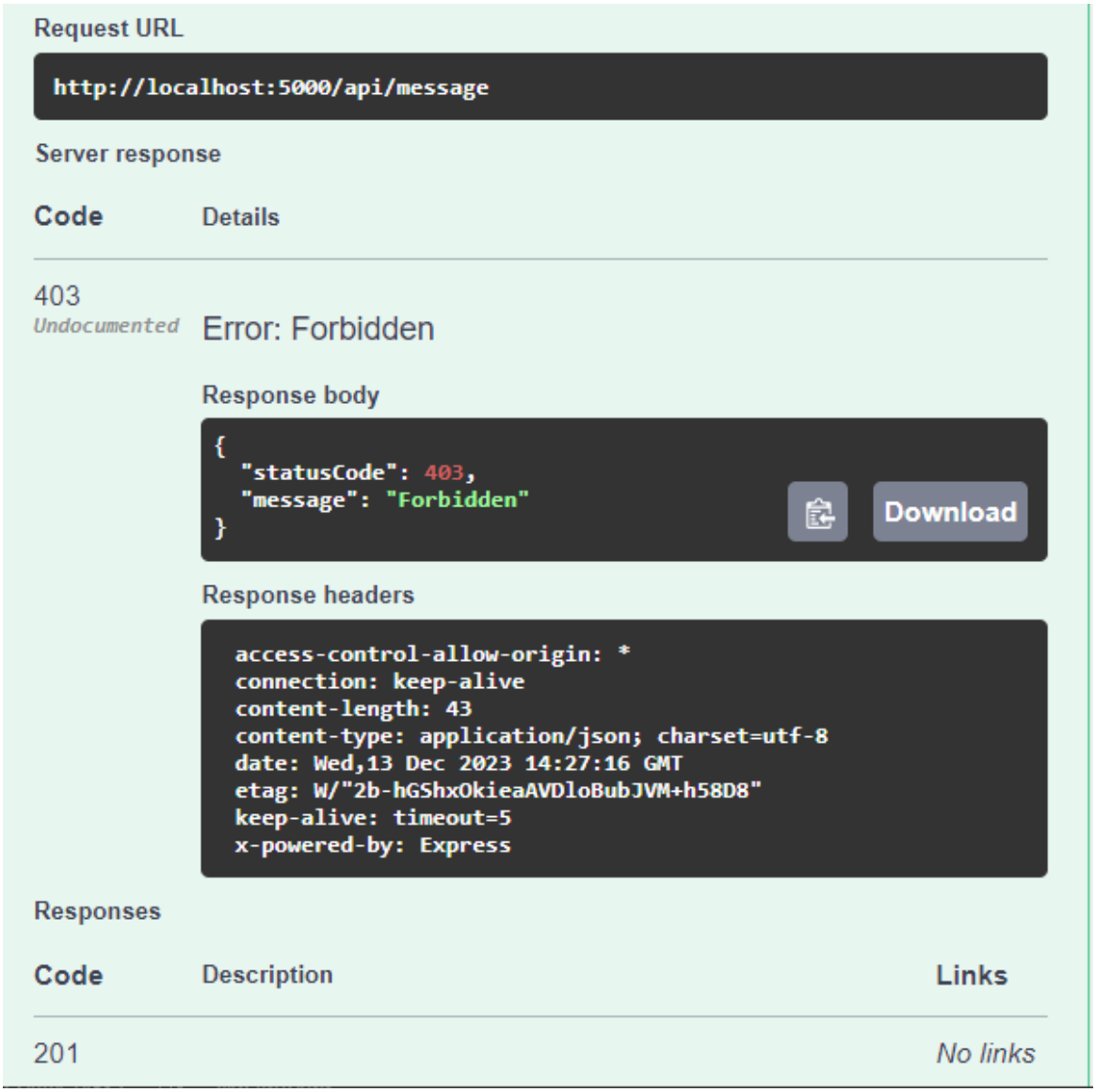

Рисунок 4.22 – Результат тестування модулю фільтрації контенту.

Показаний на рисунку 4.22 результат тестування свідчить про те, що у випадку якщо користувач хоче надіслати повідомлення з небажаним вмістом, сервер автоматично відповідає помилкою 403, що означає що йому було відмовлено в доступі відправки такого повідомлення.

4.4 Висновки

Цей розділ було присвячено створенню інструкції призначених для полегшення підтримки та інтеграції програмного продукту для розробників, а також забезпечення зручного використання кінцевими користувачами. Окрім того, було досконально протестовано всі модулі розроблено веб-месенджеру що дозволило впевнитись в його повній працездатності.

Основний акцент робиться на сприянні взаємодії розробників з продуктом та наданні інтуїтивно зрозумілих інструкцій для кінцевого користувача.

ВИСНОВКИ

В першому розділі магістерської кваліфікаційної роботи було присвячено ретельному аналізу об’єкту дослідження. Було проаналізовано принципи роботи веб-месенджерів, розглянуто основні аспекти клієнт-серверної архітектури, проведено огляд вже існуючих технологічних рішень, підкресливши плюси та мінуси кожного.

Також, було приділено окрему увагу розгляду концепції автоматичної фільтрації контенту, та це все в сумі дозволило скласти загальні вимоги до програмного забезпечення яке було розроблено в наступних розділах.

У другому розділі було детально розглянуто вимоги до клієнт-серверної системи, визначено характеристики оцінки успішності програмного забезпечення. Також, було визначено стек оптимальних технологій, які надали найбільшу кількість можливостей при розробці проекту. В цей стек увійшли мови програмування та фреймворки, СКБД та середовище розробки. Велику увагу було приділено розробленню модулю фільтрації контенту, під час розробки якого було використано нейронно-мережеві технології машинного навчання.

Обрана технічна архітектура виявилася добре збалансованою та спрямованою на забезпечення ефективності та гнучкості системи. Використання відомих та надійних технологій сприяє не лише створенню функціонального продукту, але й полегшує його подальше розширення та супровід.

Третій розділ було присвячено детальному опису та проектуванню функціональних модулів системи та програмній реалізації основних модулів функціонування системи. Проведена робота включає розробку модулів авторизації, аутентифікації, чат-кімнати, відправки та отримання повідомлень, модулю користувача який зберігає всю інформацію зареєстрованої людини та виведення їх на окремій сторінці з можливістю редагування.

У четвертому розділі було проведено тестування розробленої клієнт-серверної системи засобами «Black Box Testing» та «Swagger». Було ретельно

протестовано кожен з розроблених модулів з використанням валідних та невалідних даних для перевірки працездатності розробленої системи у різноманітних ситуаціях.

За результатами тестування можна визначити що розроблена клієнт-серверна система є повністю функціонуючою та відповідає усім поставленим вимогам.

Також, у четвертому розділі було розроблено детальні інструкції розгортання додатку для розробників, де було детально описано кроки та необхідне програмне та апаратне забезпечення для успішного розгортання проекту.

Окрім інструкції розгортання було також розроблено інструкцію користувача, яка детально описує функціональні можливості системи та яким чином необхідно з нею взаємодіяти.

В результаті виконання роботи була отримана повноцінна клієнт-серверна веб-система чату, що з допомогою модулю фільтрації контенту, розробленого з використанням моделей машинного навчання повністю відповідає усім поставленим вимогам. Клієнт-серверна архітектура дозволяє ефективно керувати взаємодією компонентів системи, забезпечуючи оптимальну продуктивність та доступність.

Принципи фільтрації контенту в системі відіграють важливу роль у забезпеченні безпеки та вибору вмісту для користувачів, особливо в сучасному цифровому середовищі, де конфіденційність та відсутність небажаного контенту є важливими аспектами.

Загальний результат роботи - це не лише технічно вдосконалений месенджер, але й стратегічно побудована система, яка відповідає змінним потребам користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. David Hancock, The History of Instant Messaging: From Chat Rooms to Social Media, Wiley-Blackwell, 2014.

2. Jens Grossklags, Stefan W. Kremer, and Peter Loos, The Evolution of Instant Messaging, Springer, 2006.

3. R. Mark Hall, Chris Sharples, and Michael J. Thomas, The State of Instant Messaging: A Research Report, University of Cambridge, 2007.

4. Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems: Principles and Paradigms, 5th ed., Pearson, 2017.

5. Andrew S. Tanenbaum, Computer Networks, 6th ed., Pearson, 2015.

6. Douglas Comer, Computer Networks and Internets, 5th ed., Pearson, 2014.

7. Rakesh Agrawal, Tomasz Imielinski, and Arun Swami, "Mining Association Rules Between Sets of Items in Large Databases," Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 20-22, 1993, pp. 207-216.

8. Jens Grossklags and Stefan W. Kremer, "A Survey of Content Filtering Techniques for Instant Messaging," ACM Computing Surveys, vol. 39, no. 2, article 13, 2007.

9. Amit Sheth, Vijay Kumar Verma, and Keng-Chih Chang, Data Mining: Concepts and Techniques, 2nd ed., Pearson, 2012.

10. Yvonne Chang, John C. Park, and Jin-Woo Choi, "A Survey of Content Filtering Techniques for Instant Messaging," Proceedings of the 2008 ACM SIGCOMM Conference on Internet Measurement, Vancouver, BC, Canada, August 25-28, 2008, pp. 363-374.

11. Ninghui Li, Rui Zhang, and Minghui Qiu, "A Survey of Content Filtering for Instant Messaging," IEEE Transactions on Multimedia, vol. 12, no. 1, pp. 10-22, 2010.

12. Shenghua Wang, Junjie Huang, and Changsheng Xu, "A Survey of Content Filtering for Instant Messaging: Techniques, Issues, and Challenges," Information Science, vol. 338, pp. 120-136, 2016.

13. IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, IEEE, 1998.

14. ISO/IEC/IEEE 29148:2011, Software and Systems Engineering — Requirements Engineering — Life Cycle Processes, ISO/IEC/IEEE, 2011.

15. ISO/IEC/IEEE 29149:2011, Software and Systems Engineering — Requirements Engineering — Vocabulary, ISO/IEC/IEEE, 2011.

16. Ralph R. Young and David M. Malveau, Software Architecture: An Engineering Approach, 2nd ed., Addison-Wesley Professional, 2003.

17. Michael C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, 3rd ed., Prentice Hall, 2003.

18. Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language: User Guide, 2nd ed., Addison-Wesley Professional, 2005.

19. Robert Martin, Agile Software Development: Principles, Patterns, and Practices, 2nd ed., Pearson Education, 2003.

20. Ken Schwaber and Mike Beedle, Agile Software Development with Scrum, 2nd ed., Prentice Hall, 2002.

21. Michael C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, 3rd ed., Prentice Hall, 2003.

22. David Flanagan, HTML & CSS: The Complete Reference, 8th ed., O'Reilly Media, 2023.

23. Eric Meyer, CSS: The Definitive Guide, 4th ed., O'Reilly Media, 2022.

24. Max Schwarzmüller, TypeScript in Depth, 2nd ed., Packt Publishing, 2022.

25. Max Schwarzmüller, React: The Complete Guide, 3rd ed., Packt Publishing, 2022.

26. Brian Holt, Node.js: Up & Running, 3rd ed., O'Reilly Media, 2022.

27. Jeff Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, Crown Business, 2011.

28. Ben Wilcock, NestJS: Up & Running, Packt Publishing, 2022.

29. C. J. Date, An Introduction to Database Systems, 9th ed., Addison-Wesley Professional, 2011.

30. Michael Stonebraker, Michael J. Carey, and David J. DeWitt, Readings in Database Systems, 6th ed., Morgan Kaufmann, 2012.

31. Martin L. Friedman and Abraham Silberschatz, Database Systems Concepts, 7th ed., McGraw-Hill Education, 2017.

32. David Thomas and Andrew Hunt, The Pragmatic Programmer: From Journeyman to Master, 2nd ed., Addison-Wesley Professional, 2009.

33. Scott W. Ambler, Agile Modeling: Effective Practices for Extreme Programming and Scrum, 2nd ed., Pearson Education, 2003.

34. Martin Fowler, Refactoring: Improving the Design of Existing Code, 2nd ed., Addison-Wesley Professional, 2018.

35. Tom M. Mitchell, Machine Learning, 3rd ed., McGraw-Hill Education, 2017.

36. Michael I. Jordan and Andrew Y. Ng, Machine Learning: A Probabilistic Perspective, 2nd ed., MIT Press, 2010.

37. Yoshua Bengio, Ian Goodfellow, and Aaron Courville, Deep Learning, MIT Press, 2016.

38. Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language: User Guide, 2nd ed., Addison-Wesley Professional, 2005.

39. David Thomas and Andrew Hunt, The Pragmatic Programmer: From Journeyman to Master, 2nd ed., Addison-Wesley Professional, 2009.

40. Agile Modeling: Effective Practices for Extreme Programming and Scrum від Скотта У. Амблера, 2-е видання, 2003.

41. Refactoring: Improving the Design of Existing Code від Мартіна Фаулера, 2-е видання, 2018.

ДОДАТКИ

Додаток А
(обов'язковий)

ЗАТВЕРДЖУЮ
Зав. кафедри АІІТ
д.т.н проф.
Олег БІСІКАЛО
«_____» _____2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи

Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту
(тема)

08-31.МКР.015.02.000 ТЗ

Студент групи 3АКІТ-22м
Михайло ЯКИМЧУК
_____

Керівник: к.т.н., доцент
кафедри АІІТ
Володимир КОЦЮБИНСЬКИЙ
_____

Вінниця 2023

1. Назва та галузь застосування

1.1. Назва – Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту.

1.2. Галузь застосування – Комп'ютеризовані системи управління бізнес-процесів підприємств та установ.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ №247 від "18" вересня 2023р.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є є покращення комунікції та підвищення достовірності публікованої інформації з допомогою створення клієнт-серверної веб-системи за принципом веб-месенджеру.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Ситник В. Ф. Інтелектуальний аналіз даних (дейтамайнінг):Навч. посібник. — К.: КНЕУ, 2007. — 376 с.
2. Чубукова И. А. Data Mining: учебное пособие. — М.: Интернет университет информационных технологий: БИНОМ: Лаборатория знаний, 2006. — 382 с. — ISBN 5-9556-0064-7.
3. Ralph Kimball, Joe Caserta. The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. — John Wiley & Sons, 2004. — 528 p. — ISBN 978-0-764-56757-5.
4. David Loshin. ETL (Extract, Transform, Load) // Business Intelligence. — 2nd. — Morgan Kaufmann, 2012. — 400 p. — ISBN 978-0-12-385890-07\

5. Вимоги до розробки.

5.1. Перелік головних етапів виконання розробки:

- визначити аспекти функціонування веб-месенджерів;
- дослідити функціональні можливості систем аналогів;
- скласти вимоги до веб-системи, що розробляється;
- спроектувати та розробити систему у відповідності до поставлених вимог та функцій:
- забезпечення доступу користувачів до системи та персональних аккаунтів;
- автоматична фільтрація контенту;

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- WINDOWS 7\8\10.

5.2.2. Умови експлуатації системи:

- робота на стандартних ПЕОМ в приміщеннях зі стандартними умовами;
- можливість цілодобового функціонування системи;
- текст програмного забезпечення системи є цілком закритим.

. Стадії та етапи розробки.

6.1 Пояснювальна записка:

- Дослідження актуальності поставленої задачі 21.09.2023р.
- Дослідження основних аспектів веб-месенджерів 28.09.2023р.
- Дослідження роботи та порівняльний аналіз існуючих веб-месенджерів 05.10.2023р.
- Проектування та розробка веб-месенджеру з автоматичною фільтрацією контенту 12.10.2023р.
- Апробація результатів дослідження 21.10.2023р.
- Оформлення пояснювальної записки, графічного матеріалу і презентації 29.10.2023р.

6.2 Графічні матеріали:

- Use-case UML-діаграма 05.11.2023р.
- UML-діаграма діяльності 14.11.2023р.
- вигляд екранів розробленого web-додатку 20.11.2023р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «28» листопада 2023 р.

7.2. Атестація проекту здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «10» грудня 2023р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК.

7.4. Захист магістерської кваліфікаційної роботи провести до «18» грудня 2023р.

Додаток Б
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту

(тема)

Студент групи 3АКІТ-22м
Михайло ЯКИМЧУК
_____

Керівник: к.т.н., доцент
кафедри АІІТ

Володимир КОЦЮБИНСЬКИЙ
_____

Вінниця 2023

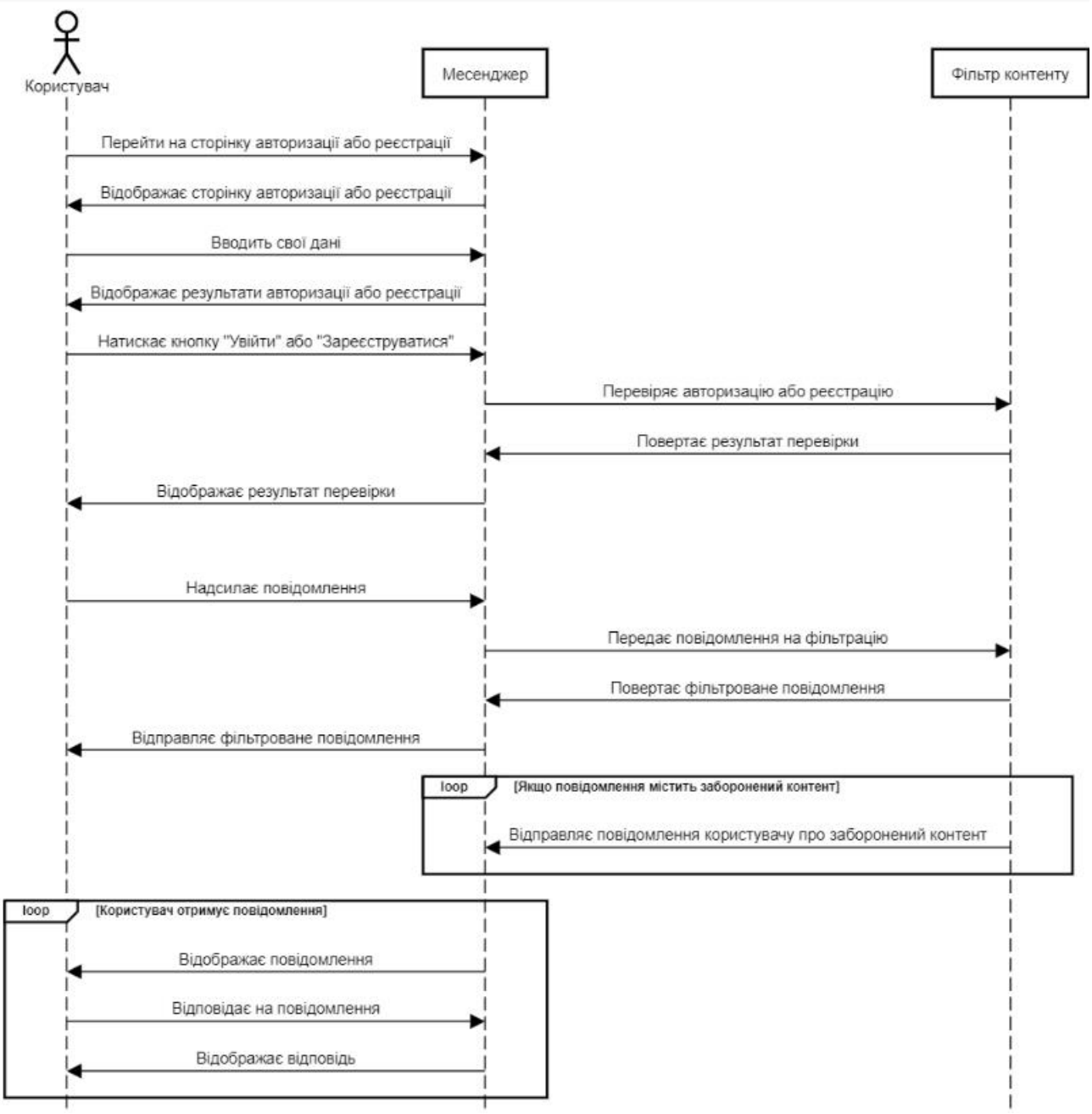

Рисунок Б.1 – UML-Flow діаграма діяльності для системи WEB месенджера з автоматичною фільтрацією контенту

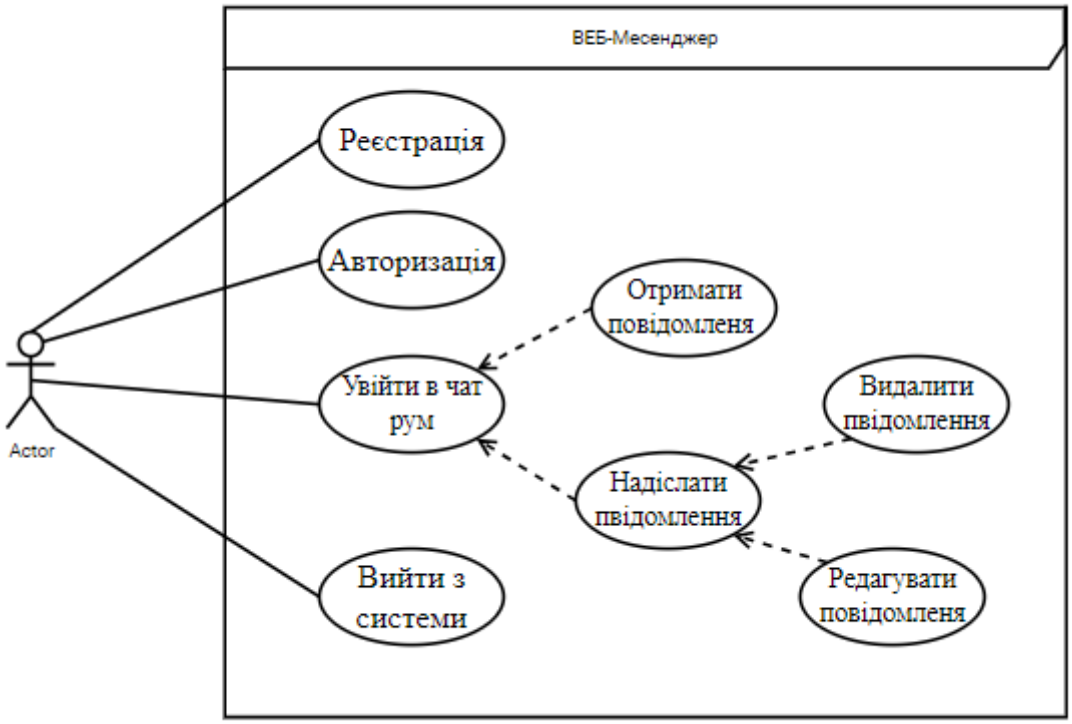

Рисунок Б.2 – USE-CASE UML-діаграма для системи WEB месенджера з автоматичною фільтрацією контенту

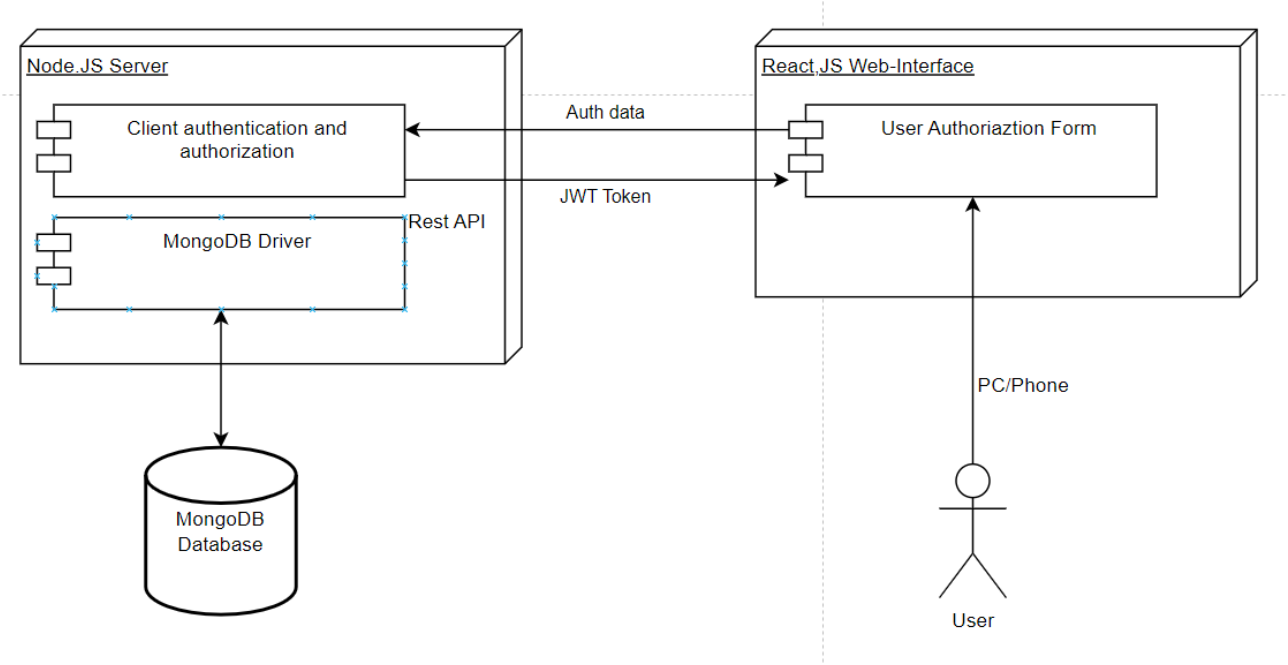

Рисунок Б.3 – Принцип роботи аутентифікації та авторизації

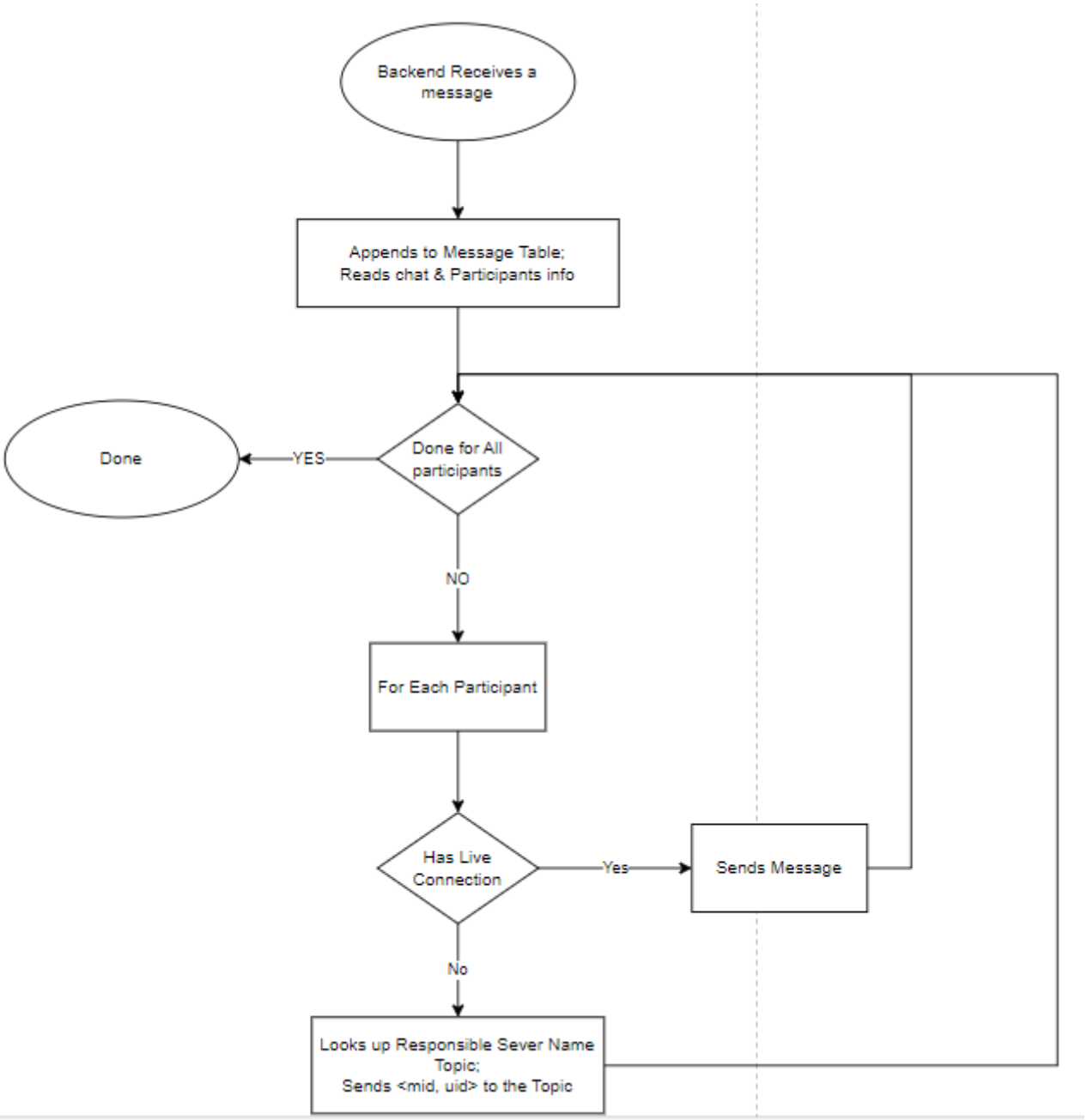

Рисунок Б.4 – Діаграма процесу надсилання повідомлення

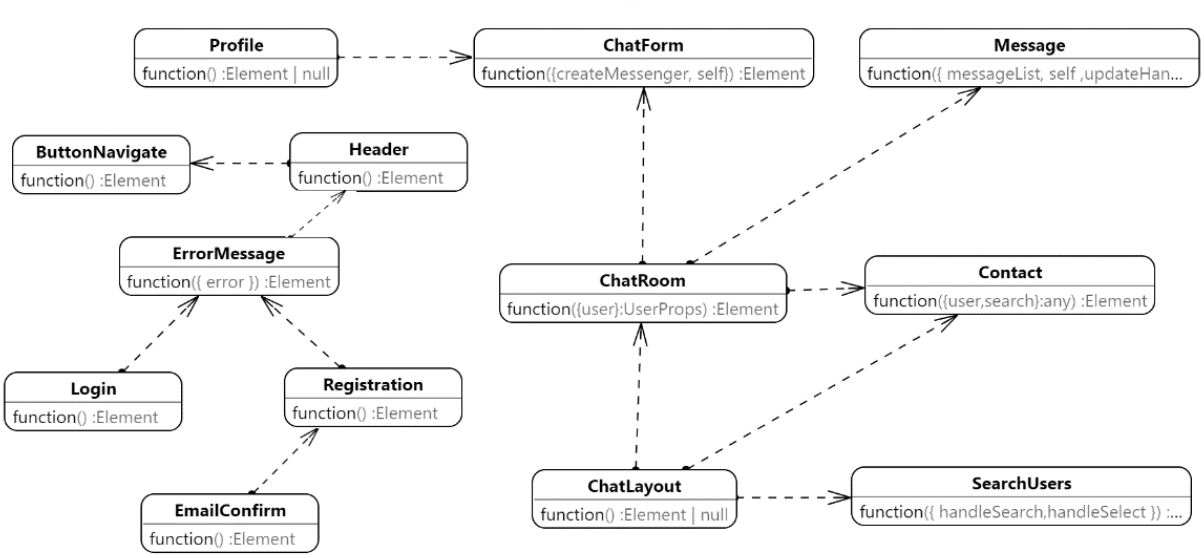

Рисунок Б.5 – Загальна структура системи.

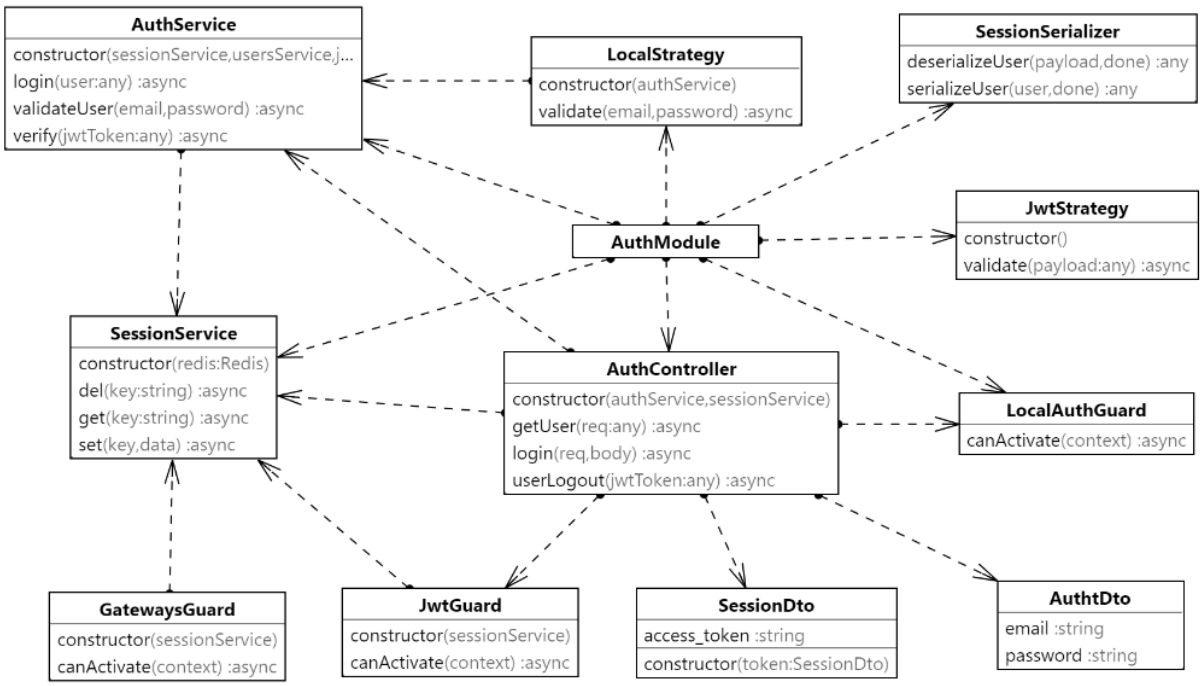

Рисунок Б.6 – UML-діаграма основних класів модуля авторизації.

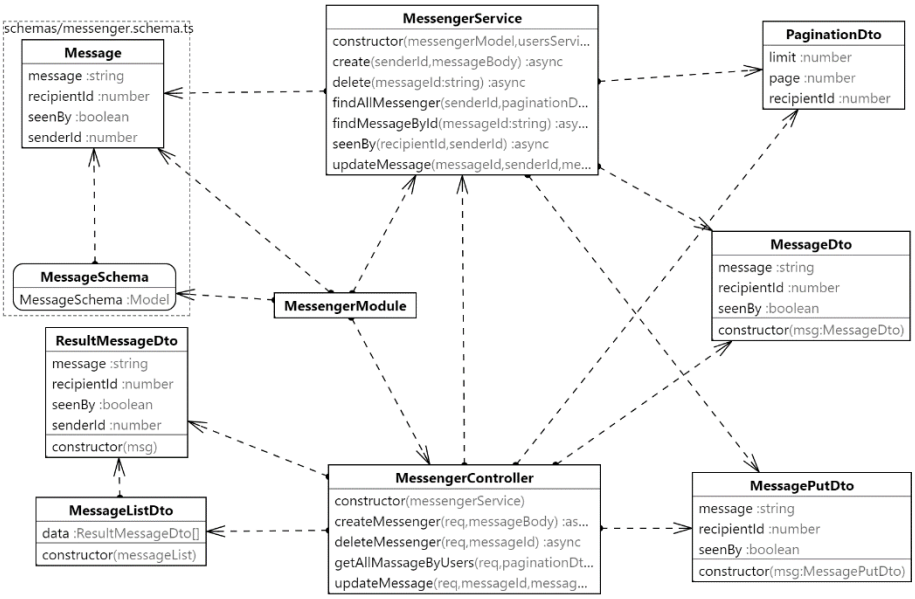

Рисунок Б.7 – UML-діаграма основних класів модуля Messenger.

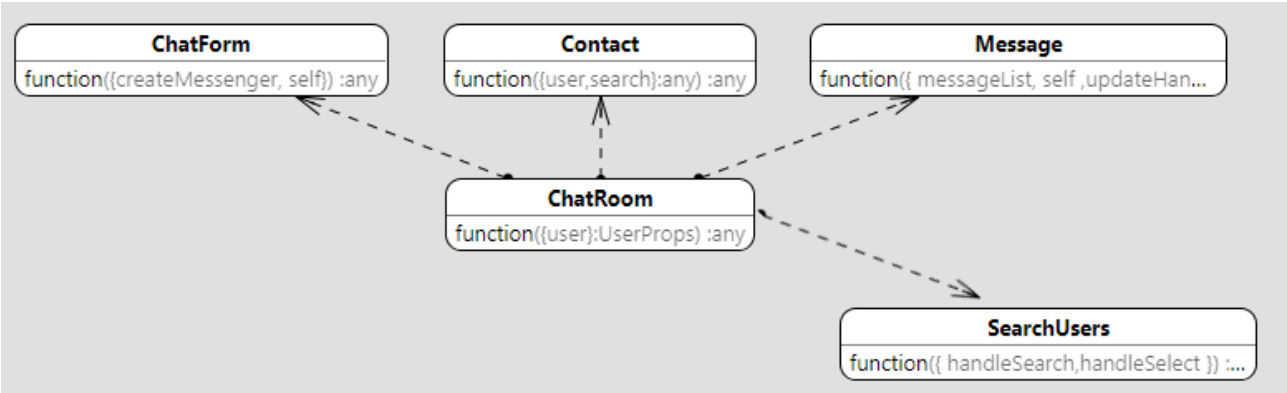

Рисунок Б.8 – UML-діаграма основних класів модуля ChatRoom.

Додаток В
(обов’язковий)

Лістинг програмного забезпечення

auth.controller

```
import {
  Controller,
  Body,
  Post,
  Request,
  UseGuards,
  Get,
  Headers,
  HttpCode,
  HttpStatus,
} from '@nestjs/common';
import { AuthService, } from '../service/auth.service';
import { AuthGuard, } from '@nestjs/passport';
import { AuthtDto, } from "../dto/auth.dto";
import { LocalAuthGuard, } from "../guard/local-auth.guard";
import { JwtGuard, } from "../guard/jwt.guard";
import { SessionDto, } from "../dto/session.dto";
import { SessionService, } from '../service/session';
import { UserDto, } from '../../users/dto/body-users.dto';
import { ApiBearerAuth, ApiTags, } from '@nestjs/swagger';

@ApiTags('Auth')
@Controller('user')
export class AuthController {
  constructor(
    private authService: AuthService,
    private readonly sessionService: SessionService,
  ) { }

  @UseGuards(LocalAuthGuard)
  @Post('login')
    async login(@Request() req, @Body() body: AuthtDto): Promise<SessionDto> {
        const jwtToken = await this.authService.login(req.user.get({ plain: true, }));
    return new SessionDto(jwtToken);
  }

  @ApiBearerAuth()
  @UseGuards(JwtGuard)
  @UseGuards(AuthGuard('jwt'))
  @Get('me')
      async getUser(@Request() req): Promise<UserDto> {
    return new UserDto({ ...req.user, });
  }

  @ApiBearerAuth()
  @UseGuards(AuthGuard('jwt'))
  @HttpCode(HttpStatus.NO_CONTENT)
  @Get('logout')
     async userLogout(@Headers('authorization') jwtToken): Promise<void> {
    await this.sessionService.del(jwtToken);
  }
}
```

auth.service

```
import { Injectable, NotAcceptableException, } from '@nestjs/common';
import { UsersService, } from 'src/users/service/users.service';
import * as bcrypt from 'bcrypt';
import { JwtService, } from '@nestjs/jwt';
import { SessionService, } from "./session";

@Injectable()
export class AuthService {
  constructor(
    private sessionService: SessionService,
    private readonly usersService: UsersService,
    private jwtService: JwtService) { }

    async validateUser(email: string, password: string): Promise<any> {
        const user = await this.usersService.getByEmail(email);
    if (!user) return null;
        const passwordValid = await bcrypt.compare(password, user.password);
    if (!user) {
      throw new NotAcceptableException('could not find the user');
    }
    if (user && passwordValid) {
      return user;
    }
    return null;
  }

  async verify(jwtToken: any) {
    try {
      const id = this.jwtService.verify(jwtToken.split(' ')[1]);
      return id.userId;
    }
    catch (err) {
      return false;
    }
  }

  async login(user: any) {
    const jwtToken = this.jwtService.sign({ email: user.email, userId: user.id, });
    await this.sessionService.set('Bearer ' + jwtToken, user);
    return { access_token: jwtToken, };
  }
}
```

Session

```
import { InjectRedis, } from '@liaoliaots/nestjs-redis';
import Redis from 'ioredis';

export class SessionService {
  constructor(@InjectRedis() private readonly redis: Redis) { }

  async set(key: string, data: string) {
    const user = JSON.stringify(data);
    await this.redis.set(key, user, "EX", process.env.EXPIRESIN);
  }

  async get(key: string) {
    const result = JSON.parse(await this.redis.get(key));
    return result;
  }

  async del(key: string) {
    await this.redis.del(key);
  }
}
```

gateways.guard

```
import { Injectable, UnauthorizedException, ExecutionContext, CanActivate, } from
'@nestjs/common';
import { SessionService, } from '../service/session';

@Injectable()
export class GatewaysGuard implements CanActivate {
  constructor(private sessionService: SessionService,) { }

  async canActivate(context: ExecutionContext) {
    const request = context.switchToHttp().getRequest();
    const user = await this.sessionService.get(request.handshake.headers.authorization);
    if (!user) throw new UnauthorizedException();
    return (request.user = user);
  }
}
```

jwt.guard

```
import { Injectable, UnauthorizedException, ExecutionContext, CanActivate, } from '@nestjs/common';
import { SessionService, } from '../service/session';

@Injectable()
export class JwtGuard implements CanActivate {
  constructor(private sessionService: SessionService,) { }

  async canActivate(context: ExecutionContext) {
    const request = context.switchToHttp().getRequest();
    const user = await this.sessionService.get(request.headers.authorization);
    if (!user) throw new UnauthorizedException();
    return (request.user = user);
  }
}
```

local-auth.guard

```
import { Injectable, ExecutionContext, } from '@nestjs/common';
import { AuthGuard, } from '@nestjs/passport';

@Injectable()
export class LocalAuthGuard extends AuthGuard('local') {
  async canActivate(context: ExecutionContext) {
    const result = (await super.canActivate(context)) as boolean;
    const request = context.switchToHttp().getRequest();
    await super.logIn(request);
    return result;
  }
}
```

Додаток Г
(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка клієнт-серверної системи WEB месенджера з автоматичною фільтрацією контенту»

Тип роботи: ____Магістерська кваліфікаційна робота_____
(БДР, МКР)

Підрозділ _____АІІТ, ФІІТА_____
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність ___97,3%_____ Схожість___2,7%_____

Аналіз звіту подібності (відмітити потрібне)

- Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.
- Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ _____Роман МАСЛІЙ_____
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek щодо роботи.
Автор роботи _____ _____ Михайло ЯКИМЧУК_____
(підпис) (прізвище, ініціали)

Керівник роботи _____ _____ Володимир КОЦЮБИНСЬКИЙ_____
(підпис) (прізвище, ініціали)