Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для соціальної взаємодії та комунікації
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-5

Юніс ХРУБ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Дмитро КОЛЕСНИКОВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Хруб Юнісу Кахеру
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для соціальної взаємодії та комунікації
Затверджена наказом по університету від 19.05. 2025р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 13.06.2025
3. Вихідні дані до роботи Розробити застосунок, що буде сприяти соціальної взаємодії та комунікації що включатиме веб-клієнт на React, серверну частину на ASP.NET Core з використанням Entity Framework Core, мобільний додаток на Kotlin для Android, реляційну базу даних Microsoft SQL Server, а також комплект документації і автоматизованих тестів.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	22.05.2025	*виконано*
2	Створення специфікації ПЗ	24.05.2025	*виконано*
3	Проєктування ПЗ	27.05.2025	*виконано*
4	Розробка ПЗ	29.05.2025	*виконано*
5	Тестування ПЗ	30.05.2025	*виконано*
6	Оформлення пояснювальної записки	31.05.2025	*виконано*
7	Підготовка презентації та доповіді	02.06.2025	*виконано*
8	Нормоконтроль, рецензування	08.06.2025	*виконано*
9	Здача роботи у електронний архів	10.06.2025	*виконано*
10	Попередній захист	10.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	12.06.2025	*виконано*

Дата видачі завдання « 19 » « травня » 2025р.

Здобувач _____
(підпис)

Керівник роботи_____ доц. кафедри ПІ Дмитро КОЛЕСНИКОВ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 62 с., 16 рис., 2 табл., 3 додатки, 15 джерел.

БАЗА ДАНИХ, КОРИСТУВАЛЬНИЦЬКИЙ ІНТЕРФЕЙС UI, МОВА ПРОГРАМУВАННЯ C#, МАСШТАБОВАНА АРХІТЕКТУРА, СОЦІАЛЬНА ВЗАЄМОДІЯ, ПЛАТФОРМА .NET, ПРОГРАМА VISUAL STUDIO, СУБД MICROSOFT SQL SERVER.

Мета роботи – розробка програмної системи, що забезпечує комфортну соціальну взаємодію, зручний обмін повідомленнями, а також створення простору для формування дружніх контактів та підтримання спілкування між користувачами. Методи вирішення поставлених задач базуються на використанні СУБД Microsoft SQL Server для надійного зберігання та управління даними, технології Entity Framework Core для зручного доступу до бази даних, середовища розробки Microsoft Visual Studio та фреймворку ASP.NET Core для реалізації серверної логіки застосунку. Веб-інтерфейс системи розроблений за допомогою JavaScript-бібліотеки React, що дозволяє створювати швидкий і зручний користувацький інтерфейс. Для реалізації мобільного застосунку було використано мову програмування Kotlin. Розроблена програмна система забезпечує можливість реєстрації користувачів, створення особистих профілів, обмін текстовими та мультимедійними повідомленнями, створення публікацій, коментування та лайки, а також підтримку дружніх контактів і підписок. Система також має вбудовані адміністративні функції для модерування контенту та управління користувачами.

У результаті роботи створено ефективний програмний продукт, який дозволяє користувачам комфортно спілкуватися, легко обмінюватися інформацією та підтримувати активну соціальну взаємодію у безпечному модерованому просторі. Програмна система повністю відповідає заданим вимогам і готова до практичного використання.

ABSTRACT

DATABASE, USER INTERFACE (UI), C# PROGRAMMING LANGUAGE, SCALABLE ARCHITECTURE, SOCIAL INTERACTION, .NET PLATFORM, VISUAL STUDIO IDE, MICROSOFT SQL SERVER DBMS.

The objective of this work is to develop a software system that ensures comfortable social interaction, convenient messaging, and creates a space for forming friendly contacts and maintaining communication among users.

The methods employed to achieve the objectives are based on utilizing Microsoft SQL Server DBMS for reliable data storage and management, Entity Framework Core technology for convenient database access, Microsoft Visual Studio development environment, and ASP.NET Core framework for implementing server-side application logic. The web interface of the system was developed using the JavaScript library React, allowing the creation of a fast and user-friendly interface. The mobile application was implemented using the Kotlin programming language.

The developed software system provides user registration, creation of personal profiles, exchange of text and multimedia messages, publication creation, commenting and liking functionalities, as well as support for friendly contacts and subscriptions. The system also includes built-in administrative features for content moderation and user management.

As a result, an effective software product has been developed, enabling users to comfortably communicate, easily exchange information, and maintain active social interactions within a secure, moderated environment. The software system fully meets the specified requirements and is ready for practical use.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

CDN – Content Delivery Network

DTO – Data Transfer Object

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

JSON – JavaScript Object Notation

JWT – JSON Web Token

MVVM – Model-View-ViewModel

REST – Representational State Transfer

SDK – Serial Development Kit

SMTP – Simple Mail Transfer Protocol

SQL – Structured Query Language

SSL – Secure Sockets Layer

UML – Unified Modeling Language

ВСТУП

Сучасне суспільство характеризується активним використанням цифрових технологій для спілкування та взаємодії між людьми [1]. З кожним роком зростає потреба в комфортному та безпечному просторі для соціального спілкування, який би дозволяв людям легко підтримувати зв'язки, обмінюватися інформацією та комунікувати з однодумцями. Саме тому було розроблено програмну систему, яка допомагає вирішити ці завдання, поєднуючи простоту використання, зручність та сучасні технології.

Створена програмна система – це платформа, що забезпечує користувачам можливість реєструватися, створювати та підтримувати власні профілі, спілкуватися у приватних і групових чатах, ділитися своїми думками, публікувати дописи з мультимедійним контентом, а також взаємодіяти з іншими користувачами через коментарі та оцінки публікацій.

Програмною системою зможуть користуватися різноманітні категорії людей, які цінують комфортну та безпечну соціальну взаємодію. Платформа дозволяє швидко встановлювати нові зв'язки та спілкуватися з іншими користувачами. Крім соціальної взаємодії, вона пропонує можливості для побудови та підтримки контактів з людьми, які поділяють професійні інтереси або захоплення, що сприяє ефективному обміну досвідом та знаннями.

Розроблена система використовує сучасні програмні технології, такі як ASP.NET Core, Entity Framework Core, React, Kotlin [2] та архітектурний підхід, який забезпечує стабільну та ефективну роботу [3]. Це дозволяє забезпечити користувачам інтуїтивно зрозумілий та зручний інтерфейс для швидкого обміну повідомленнями, комунікації у реальному часі, а також можливість перегляду аналітики та звітів для адміністраторів платформи. Завдяки цій програмній системі користувачі отримають комфортний та безпечний простір для ефективного спілкування, створення зв'язків і обміну інформацією, що позитивно вплине на якість їхньої повсякденної соціальної взаємодії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Предметною областю роботи є соціальна комунікація, взаємодія між користувачами та організація ефективного інформаційного обміну. Основою предметної області є процеси створення комфортного середовища, що дозволяє користувачам взаємодіяти між собою, розвивати соціальні зв’язки, обмінюватися інформацією у формі публікацій та повідомлень, а також забезпечення якісного адміністрування взаємодій користувачів. До предметної області входять такі фрагменти:

- соціальна комунікація та взаємодія між користувачами;
- створення та підтримка особистих профілів;
- взаємодія через публікації, коментарі та оцінки контенту;
- встановлення та підтримка міжособистісних зв’язків та обмін повідомленнями;
- модерація та управління контентом платформи.

У межах предметної області виділено певні компоненти.

Розглянемо фрагмент, що пов’язаний з соціальною комунікацією та взаємодією між користувачами:

- об’єкт: соціальна взаємодія;
- процеси: створення публікацій, коментування, обмін повідомленнями;
- користувачі: зареєстровані користувачі (звичайні користувачі, адміністратори);
- інформаційні потреби: користувачі потребують можливості ділитися інформацією, публікувати свої думки, коментувати та лайкати чужий контент; у результаті формується соціальна мережа взаємодій;
- загальні характеристики процесів: інформація зберігається в електронному вигляді, доступна для перегляду, коментування та оцінювання іншими користувачами.

Розглянемо фрагмент, що пов’язаний з створенням та підтримкою особистих профілів:

- об’єкт: користувацький профіль;
- процеси: реєстрація, автентифікація, редагування профілю;
- користувачі: звичайні користувачі, адміністратори;
- інформаційні потреби: користувачам необхідно створювати та редагувати інформацію про себе (ім’я, електронна пошта, аватар, біографія); результатом є профіль користувача, доступний іншим користувачам для перегляду;
- загальні характеристики процесів: інформація профілів зберігається в електронній базі даних та доступна для взаємодії та пошуку іншими учасниками платформи.

Розглянемо фрагмент, що пов’язаний з взаємодією через публікації, коментарі та оцінки контенту:

- об’єкт: публікація;
- процеси: створення, перегляд, коментування, оцінювання (лайки);
- користувачі: звичайні користувачі, адміністратори;
- інформаційні потреби: користувачі потребують можливості створювати текстові та мультимедійні публікації, коментувати та оцінювати їх; результатом є активна соціальна взаємодія на платформі;
- загальні характеристики процесів: контент зберігається та обробляється в електронному вигляді, створюючи умови для ефективної взаємодії та обміну інформацією між користувачами.

Розглянемо фрагмент, що пов’язаний з встановленням та підтримкою міжособистісних зв’язків і обміном повідомленнями:

- об’єкт: особистий контакт;
- процеси: додавання у друзі, підписки, надсилання приватних повідомлень;
- користувачі: зареєстровані користувачі;

- інформаційні потреби: користувачам необхідно підтримувати контакти, додавати друзів та обмінюватися приватними повідомленнями; результатом є мережа контактів користувача;
- загальні характеристики процесів: інформація щодо контактів та повідомлень зберігається в базі даних, забезпечуючи швидку та ефективну комунікацію.

1.2 Аналіз існуючих аналогів

На сучасному ринку представлено певні платформи, що реалізують комунікаційні сервіси, соціальну взаємодію [4]. У рамках аналізу було розглянуто такі аналогічні рішення: Slack [5], Microsoft Teams [6], Discord [7], WhatsApp [8] та Telegram [9].

Slack є одним із найпоширеніших інструментів командної комунікації, орієнтованим переважно на професійну співпрацю. Завдяки багатьма інтеграціям, Slack створює затишну атмосферу для продуктивної роботи та невимушеного спілкування та взаємодії команд. Він пропонує яскравий та дружній інтерфейс з доброю навігацією, що дозволяє користувачам легко орієнтуватися серед команд, каналів та приватних повідомлень (див.рис. 1.1).

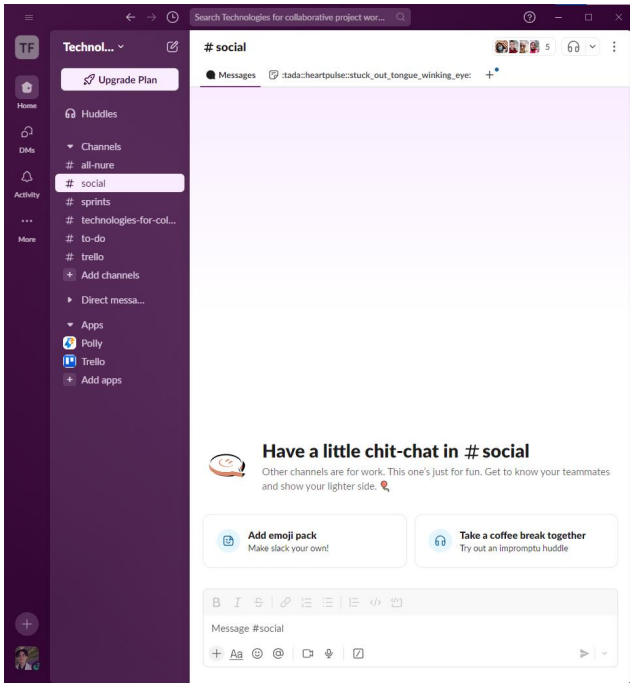

Рисунок 1.1 – Інтерфейс Slack (рисунок виконано самостійно)

Переваги Slack:

– потужна система інтеграцій (Google Drive, Trello, GitHub).
– простий, інтуїтивно зрозумілий інтерфейс;
– швидкий пошук інформації та файлів;
– масштабованість для команд будь-якого розміру;
– гарантії безпеки та відшкодування у випадку витіку інформації.

Недоліки Slack:

– висока вартість для великих компаній;
– інформаційне перевантаження через надмірну кількість повідомлень та каналів;
– обмеження у безкоштовному тарифі.

Отже, Slack зручний для професійної взаємодії, але менш підходить для комфортного особистого спілкування та соціальних аспектів.

Microsoft Teams – платформа для корпоративної комунікації, яка акцентує увагу на інтеграції з пакетом Microsoft Office (див.рис. 1.2).

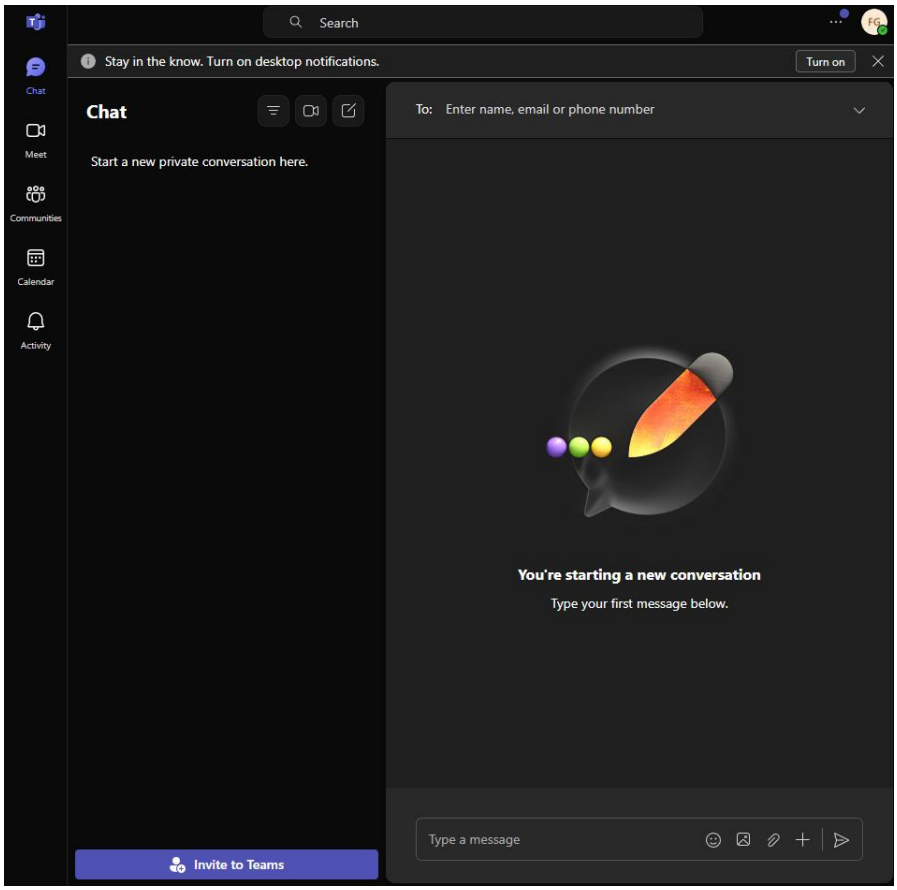

Рисунок 1.2 – Інтерфейс MS Teams (рисунок виконано самостійно)

Переваги Microsoft Teams:

– відмінна інтеграція з Office 365 (Word, Excel, PowerPoint);
– потужні можливості відеоконференцій;
– високий рівень безпеки і відповідність регуляторним стандартам;
– спільна робота над документами в реальному часі.

Недоліки Microsoft Teams:

– велика кількість функцій ускладнює засвоєння платформи новими користувачами;
– високі вимоги до системних ресурсів пристроїв;
– базова безкоштовна версія має обмеження у функціях безпеки.

Отже, Teams є гарним вибором для бізнесу, проте слабко охоплює аспекти вільної соціальної взаємодії між звичайними користувачами поза межами робочих завдань.

Discord є популярною платформою для неформального спілкування, орієнтованою переважно на спільноти та геймерів, але останнім часом розширює аудиторію (див.рис. 1.3).

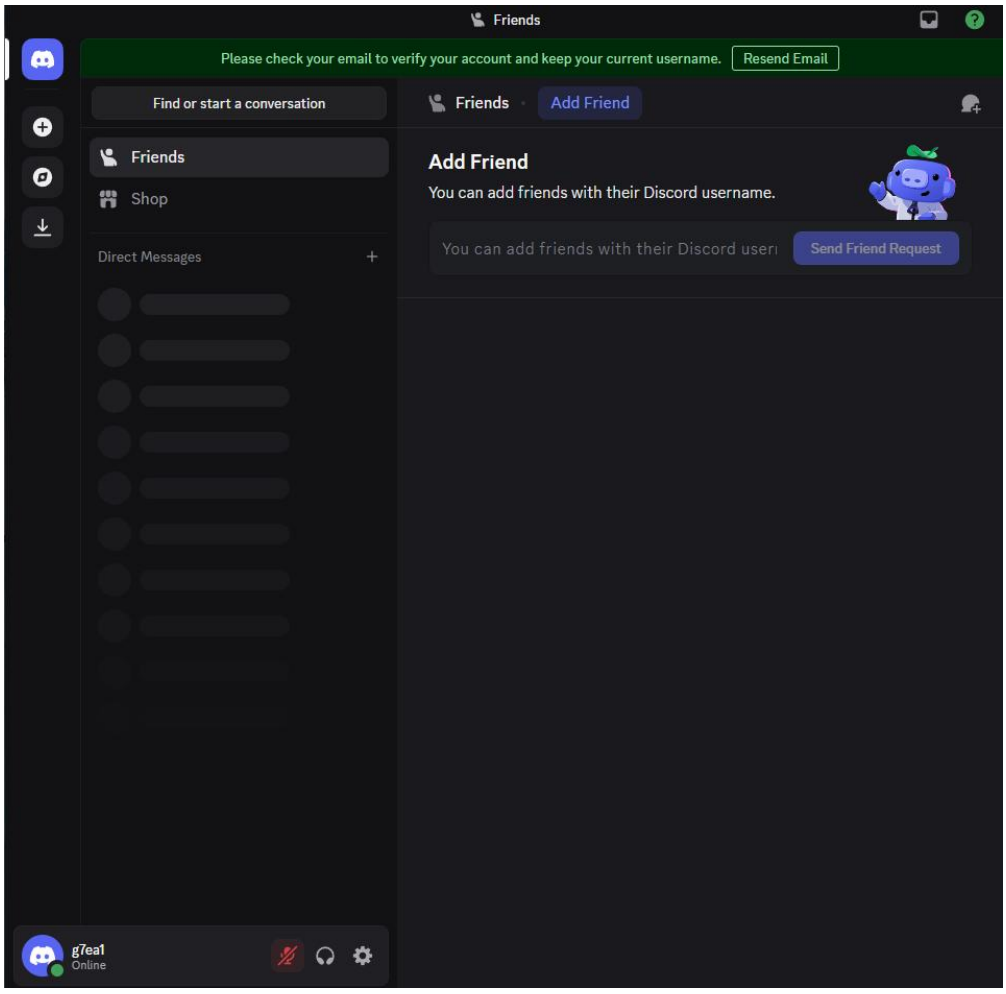

Рисунок 1.3 – Інтерфейс Discord (рисунок виконано самостійно)

Переваги Discord:

– безкоштовне використання більшості функцій;
– високоякісні голосові та відео канали з низькою затримкою;
– підтримка ботів, що автоматизують завдання;

– простий і зрозумілий інтерфейс.

Недоліки Discord:

– відсутність серйозних інтеграцій з бізнес-інструментами;
– менш строгий контроль за безпекою даних інформацією;
– платформа має неформальний характер, що може не задовольняти потреби бізнес-аудиторії.

Discord ідеально підходить для дружнього спілкування, проте не забезпечує належного рівня професійних комунікацій та мережевої взаємодії.

WhatsApp є глобальною комунікаційною платформою для швидкого обміну повідомленнями, яка орієнтована переважно на особисту комунікацію.

Переваги WhatsApp:

– велика кількість активних користувачів;
– простий, інтуїтивний мобільний інтерфейс;
– захищений обмін повідомленнями завдяки кінцевому шифруванню;
– широка підтримка мобільних пристроїв.

Недоліки WhatsApp:

– обмежені можливості для роботи з групами та проєктами;
– практично відсутні інтеграції з іншими сервісами;
– потрібен номер телефону для реєстрації, що зменшує анонімність та зручність.

Отже, WhatsApp є зручним засобом комунікації, але не надає інструментів для якісного професійного нетворкінгу та групової співпраці.

Telegram є універсальною платформою, яка підтримує як особисте спілкування, так і роботу з великими спільнотами (див.рис. 1.4).

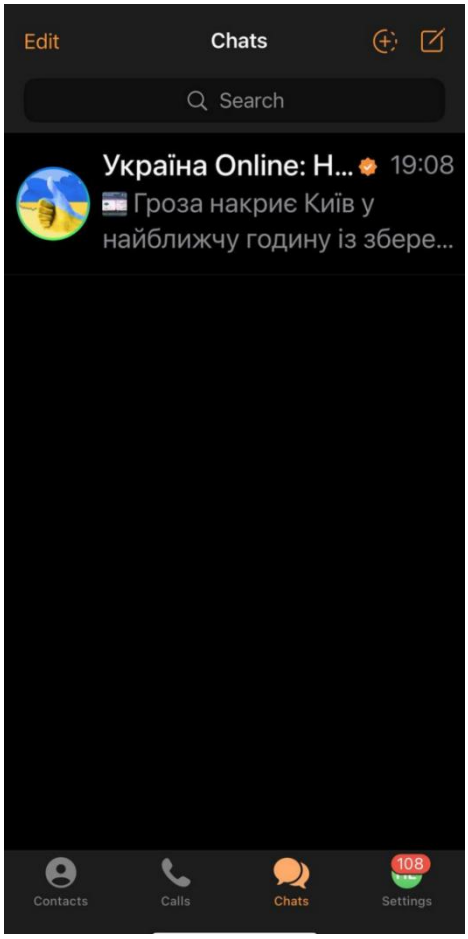

Рисунок 1.4 – Інтерфейс Telegram (рисунок виконано самостійно)

Переваги Telegram:

– високий рівень безпеки завдяки секретним чатам;
– підтримка потужних ботів і автоматизації;
– висока швидкість роботи та надійність передачі інформації;
– хмарне зберігання повідомлень і медіафайлів.

Недоліки Telegram:

– обмежені інтеграції з бізнес-інструментами;
– менше фокусування на організації професійних спільнот;
– менш строгий контроль контенту, що може бути проблематичним для професійної аудиторії.

Telegram добре підходить для широких спільнот, але йому не вистачає серйозних функцій, що сприяють професійному спілкуванню. Однією з ключових

особливостей Telegram є можливість запуску «Секретних чатів» із повноцінною end-to-end шифрацією. У Secret Chats повідомлення шифруються за допомогою MTProto-протоколу таким чином, що ключі зберігаються тільки на пристроях учасників розмови [10]. Порівняння аналогів підсумовано та наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння аналогів (таблиця виконана самостійно)

Параметр	Discord	Telegram	WhatsApp	Slack	MS Teams
Соціальна взаємодія	Висока	Висока	Середня	Середня	Середня
Складність використання	Легке	Низька	Низька	Низька	Середня
Інтеграції	Середньо	Середньо	Дуже мало	Найбільше	Багато
Нетворкінг	Низький	Середній	Низький	Найвищий	Високий
Інформаційне переванта ження	Середнє	Середнє	Середнє	Велике	Велике
Рівень безпеки	Середній	Середній	Середній	Дуже високий	Високий

ChatterLink пропонує баланс між простотою користування, соціальною складовою та базовими функціями, що дозволяють підтримувати професійні контакти в рамках соціальної мережі.

1.3 Постановка задачі

Сучасний ринок комунікаційних систем вимагає програмних рішень, які поєднують зручність соціальних мережі. Бізнес потребує ефективного інструменту для обміну інформацією, а також модерування контенту для підтримки репутації та інформаційної безпеки. Саме таким рішенням є проєкт ChatterLink.

– основні бізнес-вимоги до проєкту;
– забезпечити зручне та захищене середовище для спілкування;
– створити систему адміністрування для керування користувачами, постами, коментарями та іншими елементами контенту;
– реалізувати сучасний інтерфейс;
– гарантувати масштабованість системи для збільшення кількості користувачів та даних;
– забезпечити надійність та безперервність роботи сервісу.

Основною метою проєкту ChatterLink є створення зручної та безпечної онлайн-платформи, що інтегрує функції соціальної мережі та модерації контенту. Успіх проєкту визначатиметься залученням не менше 10 000 активних користувачів упродовж першого року роботи, середнім показником перебування користувача на платформі від 15 хвилин за сеанс, стабільною роботою системи з доступністю не нижче 99,9%, позитивними відгуками про зручність використання (не менше 80% схвалення) та швидкістю реагування на критичні адміністративні запити в межах однієї години.

Основні зацікавлені сторони (stakeholders):

− власники бізнесу (замовники) – зацікавлені у зростанні активності користувачів та ефективності платформи;
− розробники – відповідають за технічну реалізацію та підтримку проєкту;
− адміністратори та модератори – контролюють контент, керують користувачами, забезпечують підтримку безпеки;

- кінцеві користувачі – поділяються на два основні класи;
- звичайні користувачі – використовують платформу для створення та перегляду контенту.

До основного функціоналу проєкту входять:

- можливість створювати, керувати та оновлювати особистий профіль;
- створення та перегляд постів: публікація текстових повідомлень із додаванням фото та відеоматеріалів;
- коментування контенту: користувачі можуть коментувати пости та брати участь у дискусіях;
- функції підписок та підписників: формування професійних зв’язків та перегляд профілів інших користувачів;
- обмін особистими повідомленнями між користувачами;
- завантаження та керування медіафайлами (фото та відео).
- функції модерування та видалення постів, коментарів та користувачів.

До проєкту ChatterLink не входять наступні функції:

- реалізація системи голосових та відео-дзвінків (передбачено лише текстові повідомлення та завантаження медіафайлів);
- інтеграція з платіжними системами та проведення фінансових транзакцій;
- використання штучного інтелекту для автоматичного створення контенту чи автоматичного модерації (на даному етапі це не планується).

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Зовнішні інтерфейсні вимоги

Вебклієнт на React: адаптивний дизайн, підтримка різних ОС; екрани реєстрації/авторизації, головна стрічка публікацій, чат, профіль користувача, адміністрування.

Мобільний клієнт на Kotlin (Android): екрани авторизації, списку чатів, вікно повідомлень, профіль.

Усі бізнес-операції доступні через REST-контролери ASP.NET Core; мережевий протокол – HTTPS/JSON, схема аутентифікації – JWT.

2.2 Функціональні вимоги

По-перше, платформа забезпечує реєстрацію нових користувачів та їх автентифікацію. Користувачі мають можливість створити обліковий запис, вказавши адресу електронної пошти та пароль, увійти до системи за допомогою цих даних і відновити забутий пароль через лист на пошту.

По-друге, кожен користувач може керувати власним профілем: переглядати і редагувати особисті дані (ім’я, аватар, біографію), змінювати контактну інформацію та переглядати хронологію останніх входів.

Третя ключова функція – створення публікацій у загальній стрічці. Користувачі можуть публікувати текстові дописи, додавати до них зображення або відео, а також переглядати пости інших учасників.

Четвертою важливою можливістю є система лайків і коментарів: будь-хто може висловити підтримку допису натисненням “лайк”, а також залишити текстовий коментар, редагувати чи видаляти свої коментарі.

П’ята функція стосується приватного обміну повідомленнями: користувачі обирають одержувача зі свого списку контактів і відправляють йому текст, файли або мультимедіа;

Адміністративний блок включає можливість модерації контенту й користувачів. Адміністратори можуть видаляти неприйнятні публікації, блокувати чи розблоковувати облікові записи, призначати ролі «модератор» або «адміністратор», а також переглядати лог дій для аудиту.

2.3 Нефункціональні вимоги

1. Продуктивність
   - Час відповіді API < 2 секунд;
2. Масштабованість
   - Підтримка до 10 000 одночасно підключених користувачів;
3. Безпека
   - Всі передані дані шифруються, паролі зберігаються хешовано, аудит дій адміністраторів;
4. Доступність
   - Uptime ≥ 99.5 % на місяць;
5. Юзабіліті
   - Відповідність WCAG 2.1 AA; інтуїтивний інтерфейс, час навчання < 1 год;
6. Сумісність
   - Підтримка останніх версій Chrome, Firefox, Safari; Android 8+; iOS 13+;
7. Підтримка REST API
   - Документація Swagger/OpenAPI;
8. Стабільність при пікових навантаженнях
   - Контролювати помилки < 0.1 % під час стрес-тестів.

2.4 Вимоги до даних та бази даних

У рамках формування вимог до програмної системи необхідно чітко визначити характеристики та обмеження, що стосуються зберігання, обробки й захисту даних. Для ChatterLink:

Серед вимог – необхідність використовувати сучасну реляційну базу даних із високою продуктивністю, надійністю та можливостями масштабування. Рекомендується використовувати СУБД MS SQL Server, яка надає розвинені інструменти для управління, моніторингу та резервного копіювання даних. База даних повинна забезпечувати підтримку зовнішніх ключів для збереження цілісності даних. Необхідно реалізувати каскадні дії (видалення, оновлення) відповідно до логіки та політик платформи.

Є потреба у оптимізації запитів EF Core є відбирати лише необхідні поля замість завантаження цілих сутностей, що значно зменшує обсяг передаваних даних і підвищує швидкодію [11].

Потрібно використовувати індекси для ключових полів, таких як UserID, Username, Email, PostID тощо, що забезпечить ефективне виконання частих запитів до бази даних [12].

База даних повинна мати налаштоване регулярне резервне копіювання та відновлення даних. Система повинна бути здатна швидко відновлюватися після збоїв без втрати важливої інформації [13].

База даних повинна витримувати значні навантаження зі зростанням кількості користувачів та обсягів інформації. Має бути передбачена можливість легкого масштабування та оптимізації бази даних в разі зростання навантаження.

Сформульовані вимоги до даних та бази даних забезпечують створення надійної та ефективної платформи для соціальної взаємодії та комунікації.

2.5 Обмеження та припущення

Припущення:

- Користувачі мають стабільне підключення до Інтернету з мінімальною швидкістю завантаження 5 Мбіт/с для коректної роботи в режимі реального часу;
- Клієнтські пристрої (браузери, мобільні застосунки) підтримують сучасні стандарти WebSocket та HTTPS.
- Серверне середовище працює під управлінням ОС Windows, з правильно налаштованим часом і зонами;
- Для автентифікації використовується поштовий сервер із можливістю відправки листів (SMTP), а користувачі мають дійсні email адреси;
- Усі дані користувачів, повідомлення та мультимедіа зберігаються в реляційній базі, де передбачається цілісність транзакцій і регулярне резервне копіювання;
- Для мобільного застосунку передбачається, що пристрої працюють під Android 8.0+.

Обмеження:

- Серверна платформа: .NET 8 та ASP.NET Core для реалізації REST API;
- ORM: Entity Framework Core 8 для взаємодії з MS SQL Server;
- База даних: Microsoft SQL Server 2019+, налаштована з шифруванням на рівні стовпців і плановим резервним копіюванням
- Вебклієнт: React 18+, бібліотеки React Router, Axios
- Мобільний застосунок: Kotlin 1.6+, Android SDK 31+, Retrofit;
- Тестування: xUnit для юніт-тестів серверної логіки;
- CI/CD: GitHub;
- Зовнішні сервіси: SMTP-сервер.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ

3.1 UML проєктування ПЗ

Розглянемо вступ до діаграми прецедентів (Use Case) у системі, у якій виокремлено дві основні ролі – звичайного користувача та адміністратора

Це полегшує як планування розробки окремих модулів вебклієнта, так і складання повноцінного тест-плану з урахуванням прав доступу та бізнес-логіки кожної ролі. Розглянемо побудову діаграми прецедентів (див. рис. 3.1).

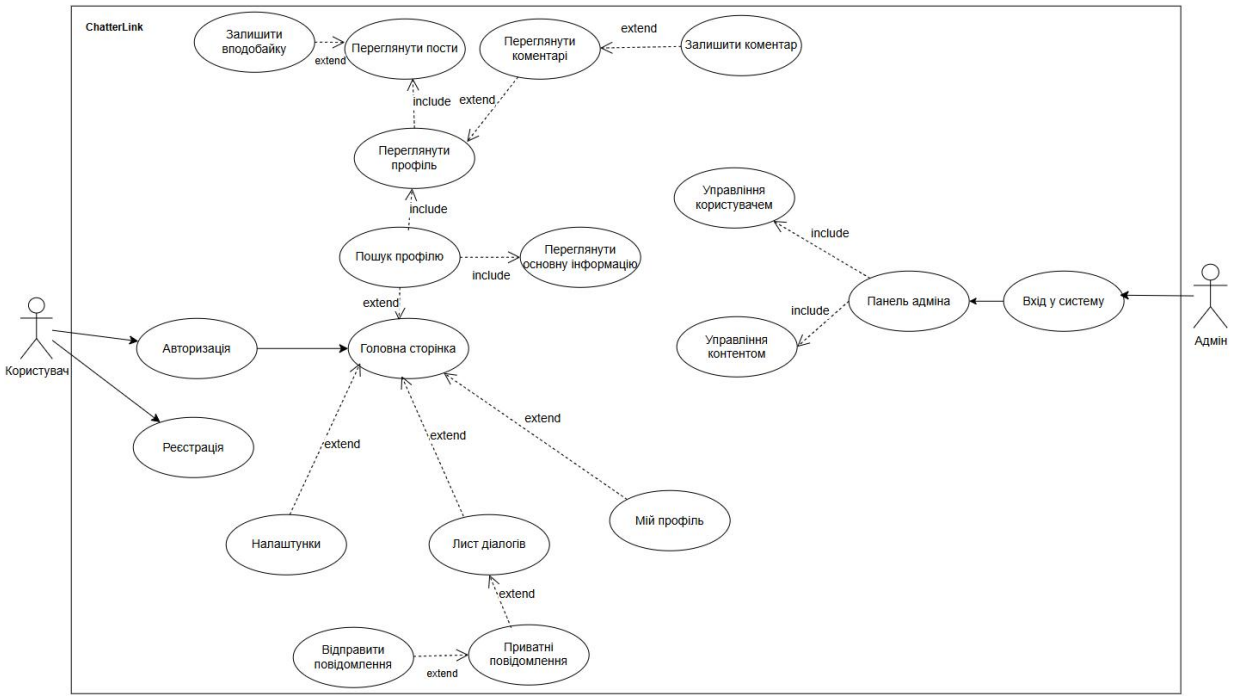

Рисунок 3.1 – Побудова діаграми прецедентів (рисунок виконано самостійно)

Метою цієї діаграми є наочно відобразити горизонтальні і вертикальні зв’язки, щоб чітко розмежувати, хто і за яких умов може виконувати ту чи іншу дію.

3.2 Архітектура програмного забезпечення

Архітектура та проєктування серверу ChatterLink побудовані за принципом моноліту з чітким поділом на шари, що дозволяє зберегти простоту розгортання та забезпечує високу продуктивність при взаємодії компонентів.

Розглянемо, як взаємодіють частини системи через діаграму розгортання (див.рис. 3.2).

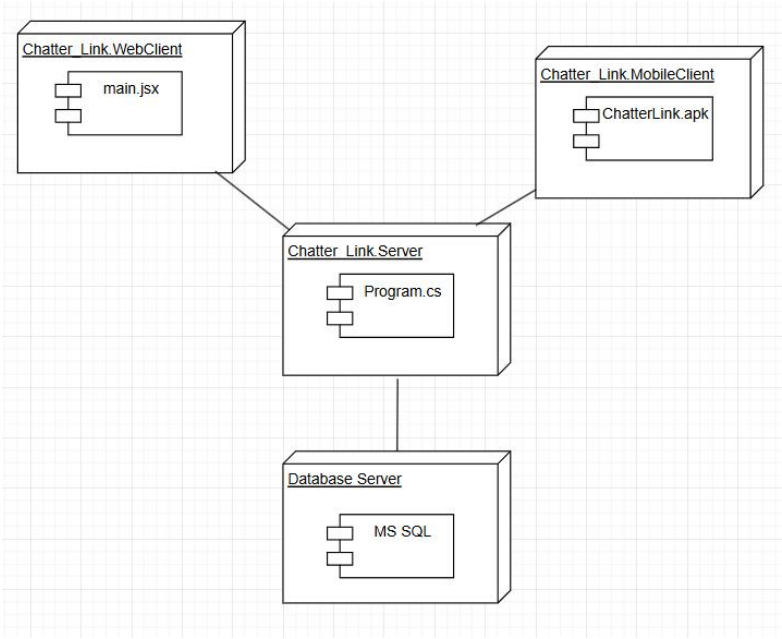

Рисунок 3.2 – Побудова діаграми розгортання (рисунок виконано самостійно)

На діаграмі розгортання представлено фізичну структуру та взаємодію основних компонентів платформи ChatterLink. Система складається з чотирьох вузлів.

Усі три компоненти – сервер, вебклієнт і мобільний застосунок – взаємодіють по HTTP, використовуючи стандартизовані методи GET, POST, PUT, DELETE для роботи з ресурсами платформи. MS SQL Server обрано як реляційну СУБД із регулярним резервним копіюванням і шифруванням на рівні таблиць
Наприклад клієнт викликає в браузері або мобільному додатку REST-ендпоінт на сервері; сервер, у свою чергу, читає, обробляє лоігку, в залежності від ситуації,

записує, редагує дані і повертає результат у JSON. Більш детально розглянути REST-ендпоінти можна у специфікації REST (див. додаток А).

3.2.1 Серверна частина

Як сервер використовується фреймворк ASP.NET. Сервер має багатошарову архітектуру, яка складається з контролерів, моделей, контексту бази даних, конфігураційних файлів та головного файлу програми.

Серверна частина реалізована на базі ASP.NET Core як RESTful Web API, у якому сервер успішно обробляє HTTP-запити та повертає відповідь у форматі JSON.

Шар бізнес-логіки містить сервіси які інкапсулюють основні алгоритми та правила роботи платформи, а також відповідають за перевірку прав доступу й валідацію даних.

Шар даних має доступ до бази даних, саме він відповідає за функціонал, який робить запити до бази даних, які вносять зміни до збережених даних програмної системи.

Розглянемо діаграму пакеті серверу (див.рис. 3.3).

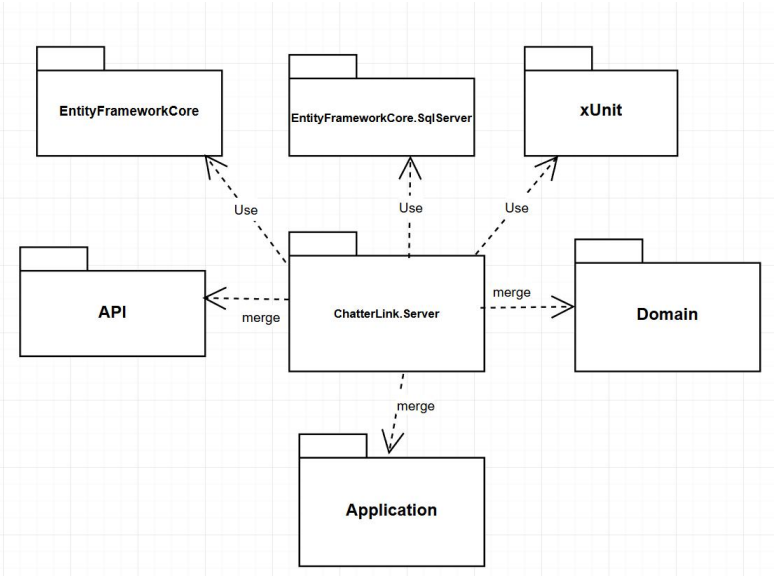

Рисунок 3.3 – Побудова діаграми пакетів (рисунок виконано самостійно)

Моделювання сутностей представлено набором класів (User, Chat, Message, Post, Comment, Like), а контекст бази даних (ApplicationDbContext) успадковує DbContext з Entity Framework Core і містить DbSet-и для кожної сутності. Конфігурація – рядок підключення, параметри JWT-автентифікації, налаштування зовнішніх сервісів – зберігається у файлах appsettings.json та appsettings.Development.json. Точка входу Program.cs використовує dependency injection для реєстрації сервісів, контексту БД(автентифікація, логування, обробка помилок).

3.2.2 Вебчастина

Вебклієнт побудований на React як SPA із використанням функціональних компонентів. HTTP-клієнт (axios) звертається до REST-ендпоїнтів сервера, обробляє відповіді та оновлює стан додатка. Структура проєкту поділена на шари: компоненти презентації, сервіси для роботи з API, утиліти та ресурси стилів .

Розглянемо, один з кейсів сценарію, що буде робити користувач на вебклієнті (див.рис. 3.4).

Рисунок 3.4 – Побудова діаграми діяльності (рисунок виконано самостійно)

Ця діаграма діяльності моделює процес взаємодії користувача з клієнтською частиною застосунку. Починається все з дії "Вхід до клієнту", після чого користувач виконує авторизацію. На етапі перевірки введених даних система визначає, чи є авторизація успішною. Якщо дані некоректні, користувач повертається до введення повторно.

3.2.3 Мобільна частина

Мобільний застосунок для платформи Android розроблено на мові програмування Kotlin із використанням сучасної архітектури MVVM, яка забезпечує чітке розділення бізнес-логіки та презентаційного шару. Центральною

складовою цієї архітектури є ViewModel, яка служить проміжним елементом між інтерфейсом користувача та репозиторієм. Саме репозиторії відповідають за отримання та зберігання даних, абстрагуючи складні деталі взаємодії з мережею та локальним сховищем. Зокрема, репозиторії забезпечують уніфікований доступ до віддалених REST-ендпоїнтів, спільних як для мобільної, так і для веб-версії, завдяки використанню бібліотеки Retrofit разом із OkHttp. Для кожного мережевого запиту репозиторій створює запит через інтерфейс Retrofit, а OkHttp-перехоплювач автоматично додає авторизаційний JWT-токен у HTTP-заголовки, що забезпечує безпечну автентифікацію користувачів.

Отримані відповіді на мережеві запити повертаються до ViewModel, яка за допомогою механізму LiveData (або StateFlow у більш сучасних реалізаціях) повідомляє інтерфейс користувача про зміну стану даних. Це дозволяє UI автоматично реагувати на оновлення, забезпечуючи плавний та адаптивний користувацький досвід без потреби вручну керувати станом інтерфейсу. Використання такого підходу сприяє кращій тестованості коду, оскільки кожен шар може бути ізольовано протестований: ViewModel можна перевіряти за допомогою юніт-тестів, а репозиторії – шляхом мокінгу мережевих викликів та роботи з локальним сховищем.

Така архітектура гарантує легку підтримку, розширення функціоналу та покращує загальну стабільність мобільного застосунку.

3.3 Структура зберігання даних

Структура зберігання даних платформи ChatterLink побудована навколо центральної таблиці Users, яка містить інформацію про кожного користувача (див.рис. 3.5).

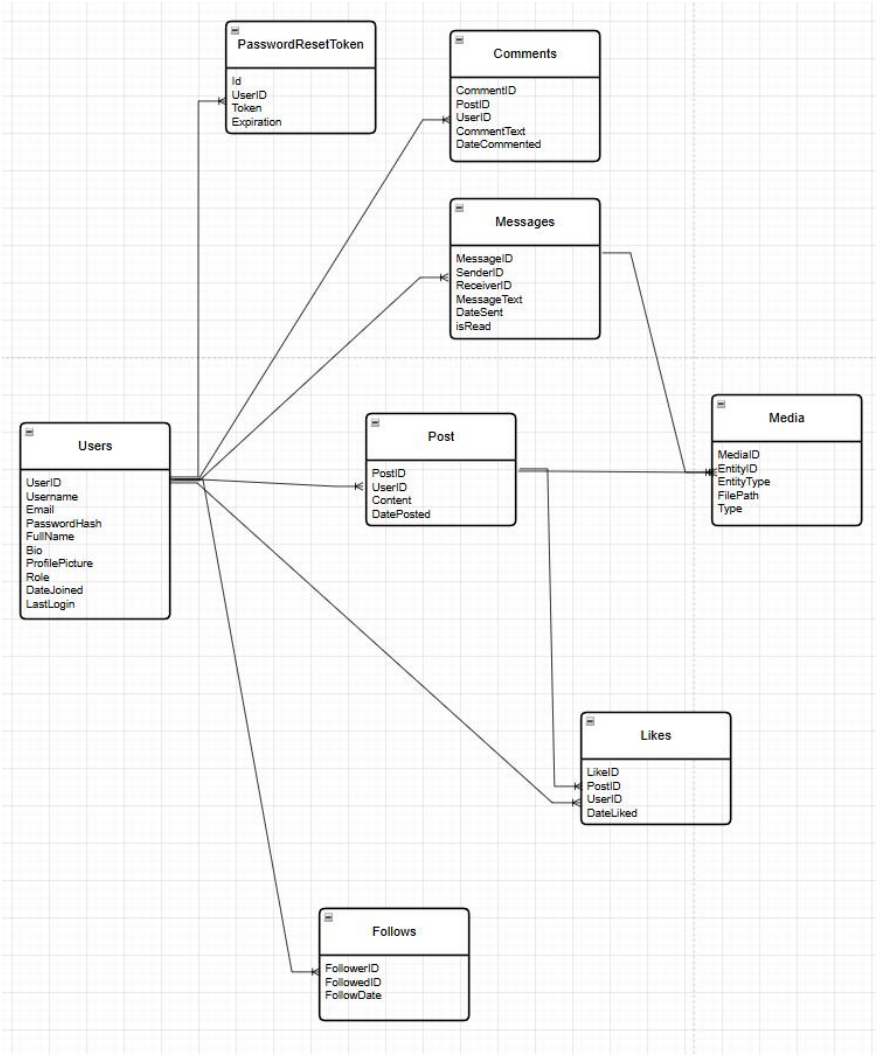

Рисунок 3.5 – Побудова діаграми зберегання даних (рисунок виконано самостійно)

Для організації процесу скидання пароля існує пов’язана з Users таблиця PasswordResetToken, що зберігає унікальний токен, дату його закінчення строку дії та посилання на того ж користувача через зовнішній ключ UserID.

Публікації представляються сутністю Post із власним ідентифікатором PostID, текстом (Content), відміткою часу створення (DatePosted) та зовнішнім ключем UserID, який вказує на автора з таблиці Users. Кожен пост може мати довільну кількість коментарів (таблиця Comments) та вподобайок (таблиця Likes). У коментарях зберігаються CommentID, текст коментаря (CommentText), час його додавання (DateCommented) і два зовнішні ключі – PostID та UserID, що гарантують зв’язок із відповідним постом і автором. Вподобайки трактується

аналогічно: кожен запис у Likes має власний ключ LikeID, посилання на пост і на користувача, який поставив «лайк», а також дату (DateLiked).

Завдяки такій моделі даних забезпечується гнучкість (універсальна прив’язка медіа), швидкий доступ за індексованими полями (Username, Email, UserID у дочірніх таблицях), а також чітка референтна цілісність, яка перешкоджає появі некоректних записів у базі.

3.4 Інтерфейс користувача

Інтерфейс користувача ChatterLink побудований за принципом єдності та простоти взаємодії. Головна сторінка платформи, де стрічка новин, постів, які організовані як нескінченний вертикальний лістинг. У верхній частині екрану розташовано адаптивну навігаційну панель – логотип програми, пошук користувачів, сповіщення про нові події й меню профілю, що забезпечує швидкий доступ до особистих налаштувань та виходу з системи, а завдяки гнучким Flex і Grid-контейнерам усі компоненти зберігають читабельність та зручність керування.

Він поєднує мінімалізм і функціональність, щоб користувач міг швидко орієнтуватися в просторі платформи, не відволікаючись на зайві деталі, але водночас отримувати всю необхідну інформацію та інструменти для комфортного спілкування й обміну контентом.

Розглянемо інтерфейс вікна при авторизації у вебклієнті (див.рис. 3.6).

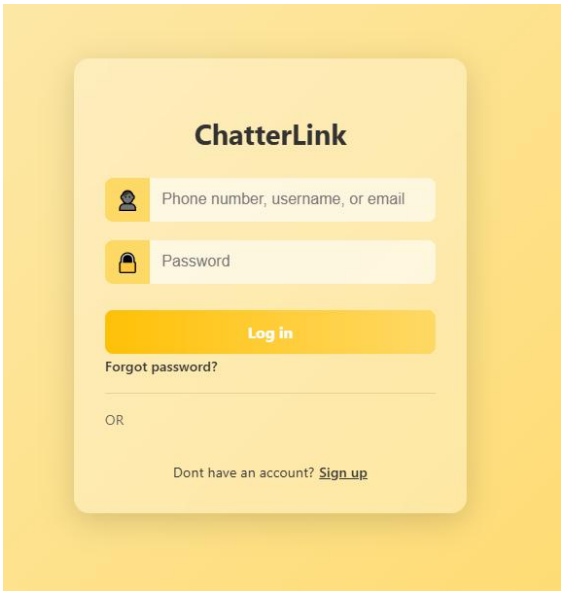

Рисунок 3.6 – Вікно авторизації (рисунок виконано самостійно)

Розглянемо інтерфейс бокової панелі (див.рис. 3.7).

Рисунок 3.7 – Бокова панель (рисунок виконано самостійно)

Розглянемо інтерфейс приватних повідомлень (див.рис. 3.8).

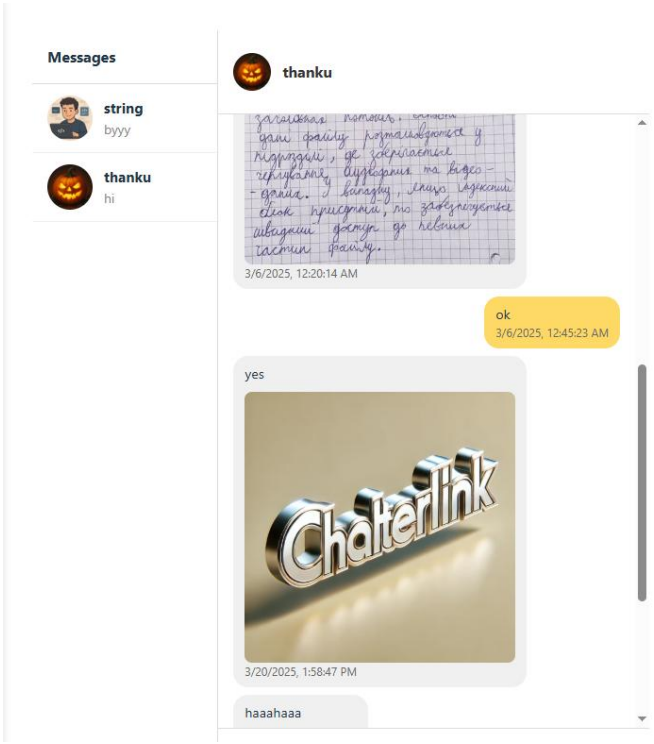

Рисунок 3.8 – Приватні повідомлення (рисунок виконано самостійно)

Розглянемо те, як виглядає стрічка постів (див.рис. 3.9).

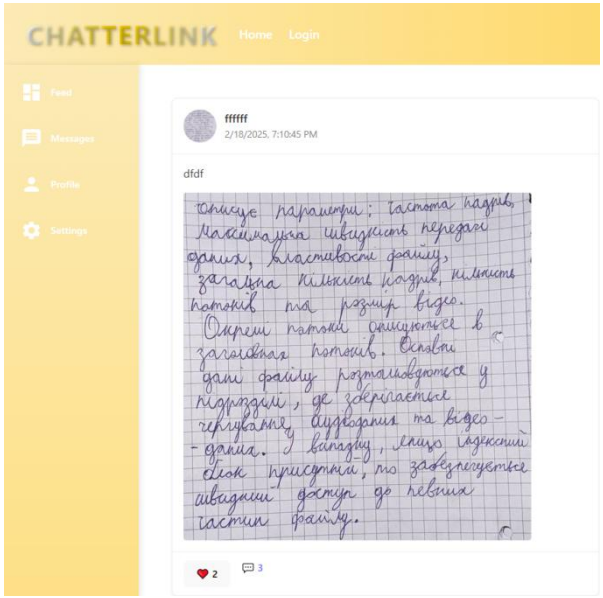

Рисунок 3.9 – Фід (рисунок виконано самостійно)

У кожному користувача відображаються його пости, фідбек до них, але зверху цього контенту - сама шапка профілю (див.рис. 3.10).

Рисунок 3.10 – Шапка профілю (рисунок виконано самостійно)

Розглянемо стиль логотипу (див.рис. 3.11).

Рисунок 3.11 – Логотип (рисунок виконано самостійно)

Загалом інтерфейс виглядає досить сучасно й лаконічно: шрифти однакові по всьому додатку, відступи й поля дотримані стало, що створює відчуття єдиного стилю.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Токен сесії користувача

Для забезпечення захищеного обміну даними між клієнтом і сервером застосовується механізм токена сесії (JWT). У наших рішеннях цей підхід реалізовано окремо для веб-фронтенду (React) та мобільного клієнта (Android).

Після успішного входу користувача (логіна) сервер повертає дві речі: сам JWT-токен для автентифікації (наприклад, у полі token HTTP-відповіді) та термін його дії (опційно), у разі потреби (якщо використовується refresh-механізм) можна також зберігати час закінчення дії чи refresh-токен під окремим ключем. У React-додатку після отримання відповіді ми зберігаємо токен у localStorage, наприклад:

```
const token = localStorage.getItem('token');
```

Щоб автоматично додавати JWT до заголовків кожного запиту, створюється окремий файл конфігурації.

Перед відправкою будь-якого запиту інтерсептор звертається до localStorage та читає там токен. Якщо він присутній, додає заголовок Authorization: Bearer <token>. Якщо ж токена немає (наприклад, користувач ще не пройшов авторизацію або термін дії закінчився), заголовок не додається:

```
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
}, (error) => {
    return Promise.reject(error);
});
```

Після виходу з системи (logout) — видаляємо токен:

```
localStorage.removeItem('token');
```

Таким чином будь-який виклик до API автоматично містить актуальний JWT, і сервер може перевіряти права доступу.

У мобільному застосунку для авторизації через JWT спочатку використовується ініціалізація SharedPreferences, у першому кроці ми готуємо сховище для зберігання токена. Для цього викликаємо метод, який отримує стандартний SharedPreferences під ключем "auth" у приватному режимі:

```
private lateinit var prefs: SharedPreferences

fun init(context: Context) {
 prefs = context.getSharedPreferences("auth", Context.MODE_PRIVATE)
}
```

Після цього в prefs можна читати й записувати значення під усіма необхідними ключами (зокрема, під ключем "token")

Далі налаштовується власний клієнт на базі OkHttp, у якому реалізовано налаштування, щодо довіри до SSL-сертифікатів та додавання заголовка Authorization: Bearer <token> у кожен вихідний запит. Interceptor для підстановки Authorization:

```
.addInterceptor(Interceptor { chain ->
        val token = prefs.getString("token", null)
        val reqBuilder = chain.request().newBuilder()
        if (!token.isNullOrBlank()) {
            reqBuilder.addHeader("Authorization", "Bearer $token")
        }
        chain.proceed(reqBuilder.build())
    })
```

Підключаємо ще один інтерсептор для логування HTTP-заголовків і тіла (рівень HEADERS), щоб було видно, які саме дані вирушають на сервер:

```
.addInterceptor(HttpLoggingInterceptor().apply {
    level = HttpLoggingInterceptor.Level.HEADERS
 })
```

OkHttpClient конфігурується через інтерсептор, який на кожному запиті дістає токен із SharedPreferences й додає його у заголовок Authorization: Bearer <token>.Коли OkHttpClient готовий, на його основі будуємо екземпляр Retrofit:

```
  private val retrofit: Retrofit by lazy {
Retrofit.Builder()
    .baseUrl(BASE_URL)
    .client(okHttpClient)
    .addConverterFactory(GsonConverterFactory.create())
    .build()
}
```

Зберігаємо отриманий токен:

```
prefs.edit().putString("token", receivedToken).apply()
```

При завершенні сесії (logout):

```
prefs.edit().remove("token").apply()
```

Отже, для зберігання JWT між сесіями використовується SharedPreferences, куди після успішного логіну записується отриманий токен

4.2 Медіафайли

У підсистемі роботи з файлами на сервері соціальної мережі передбачено зберігання та отримання медіа (зображень і відео), пов’язаних як з повідомленнями, так і з постами. Основний механізм завантаження файлів реалізовано у методі SaveFileAsync. Цей метод отримує об’єкт IFormFile (який містить дані файлу від клієнта) та стрічку subfolder, яка вказує, до якої папки. Фізично файли зберігаються у файловій системі сервера (у папці wwwroot/uploads/<subfolder>), а не на CDN. Спочатку визначається кореневий каталог веб-додатка через _env.WebRootPath. Якщо цей шлях повертає null, у якості аварійного варіанту використовується поточний робочий каталог плюс "wwwroot". Далі, всередині цьому кореневому каталозі формується шлях до папки для завантажень. Для стартапу та невеликих команд це цілком прийнятний підхід, адже дозволяє мінімізувати початкові витрати та спростити архітектуру. Проте за зростання навантаження і кількості файлів у майбутньому може виникнути необхідність перенести медіа на CDN або окреме файлове сховище.

Для уникнення колізій імен файлів кожне ім’я формується як поєднання нового GUID та початкового імені файлу:

```
var fileName = $"{Guid.NewGuid()}_{file.FileName}";
```

Після цього формується повний шлях (filePath) у файловій системі за допомогою:

```
var filePath = Path.Combine(uploadsFolder, fileName);
```

Після цього формується повний шлях (filePath) у файловій системі за допомогою:

```
using (var stream = new FileStream(filePath, FileMode.Create))
{
   await file.CopyToAsync(stream);
}
```

Після успішного запису байтів із IFormFile на диск метод повертає відносний шлях, який буде збережено в базі даних, щоб пізніше формувати публічні URL-адреси. Для зручності веб-доступу зворотні слеші переведено на прямі:

```
return Path.Combine("uploads", subfolder, fileName).Replace("\\", "/");
```

Для збереження у системів даних шляхів файлів було визначено та смодельовано самі сутність та її структуру, яка за це відповідає. Інформація про кожен завантажений файл зберігається в таблиці (або еквівалентній сутності) Media, яка має такі поля:

```
public class Media
{
[Key]
 public int MediaID { get; set; }

[Required]
  public int EntityID { get; set; } // ID поста або повідомлення

[Required]
public string EntityType { get; set; } // "Post" або "Message"

[Required]
public string FilePath { get; set; }

[Required]
 public MediaType Type { get; set; } // Enum (Image / Video)
}
```

У обох DTO-файлах поле Media є списком/масивом об’єктів MediaDto, кожен з яких містить:

– MediaID, що є унікальним ідентифікатором файлу;
– FilePath, що є відносним шляхом, за яким клієнт пізніше формує повний URL;

– Type, що є типом файлу (Image або Video).

Вибір єдиної сутності Media з полем Type (де значенням може бути, наприклад, Image чи Video) замість окремих таблиць/класів Images і Videos ґрунтується на кількох ключових аргументах. Якщо розділити медіаконтент на дві незалежні сутності (Image та Video), в базі даних і в коді доведеться підтримувати дві різні моделі, дві окремі таблиці (або DbSet у Entity Framework), два набори репозиторних методів (наприклад, AddImageAsync, GetImageAsync та AddVideoAsync, GetVideoAsync), а також дві колекції у DTO. У той самий час, коли по суті ми зберігаємо майже однакові дані (файл, шлях до файла, належність до повідомлення чи поста), цього можна досягти однією структурою Media.

Після збереження повідомлення або поста з медіа контролер віддає клієнту об’єкт DTO у форматі JSON. Приклад відповіді, наведений нижче, ілюструє, як виглядають поля:

```
public class MediaDto
{
    public int MediaID { get; set; }
    public string FilePath { get; set; }
    public MediaType Type { get; set; }
}
```

Коли клієнт надсилає запит, який стосується конкретних повідомлень або постів, серверна частина отримує сутність із бази та перетворює її у відповідні DTO. Для повідомлень використовується клас MessageDto, у якому є поле Media. Клієнт (фронтенд або мобільний застосунок) надсилає форму або multipart-запит на сервер, у якому міститься один або кілька файлів (IFormFile[]) та необхідні метадані (наприклад, ID отримувача, текст повідомлення чи контент поста). На сервері викликається контролер, який у відповідному action-методі бере кожен переданий IFormFile і послідовно передає його у SaveFileAsync.

У вебзастосунку користувач бачить та надсилає медіа (зображення чи відео) через компонент, який об’єднує кілька кроків у єдиний потік. Спочатку, ще до завантаження будь-яких файлів, користувач формує повідомлення, вводячи текстовий контент і, за потреби, вибирає зображення або відео зі свого пристрою. У коді ця логіка реалізована у функції handleSendMessage, яка створює екземпляр FormData і додає в нього поля ReceiverUsername та MessageText (рядкові значення), а потім і масиви файлів. Якщо користувач вибрав одну чи кілька картинок, кожне зображення додається так:

```
const formData = new FormData();
formData.append('ReceiverUsername', selectedUsername);
formData.append('MessageText', newMessage);
if (newImages.length > 0) {
   for (let i = 0; i < newImages.length; i++) {
      formData.append('Images', newImages[i]);
   }
}
```

Аналогічно у тому самому фрагменті перевіряється, чи є відеофайли в масиві newVideos, і кожне відео додається під ключем Videos.

Розглянемо те, як виглядають медіа-файли на інтерфейсі клієнта (див. рис. 4.1).

Рисунок 4.1 – Відправлене зображення (рисунок виконано самостійно)

Як ми бачимо, у кожний аватар співбеседника є також файлом зображенням, який також викликається з файлової системи серверу. Авторизований користувач може змінити її за надібності.

Поле response.data містить масив об’єктів MessageDto. Кожен MessageDto включає стандартні поля (messageID, senderID, receiverID, messageText, dateSent, isRead) і також масив media, що складається з об’єктів MediaDto. Клас MediaDto має поля mediaID, filePath та type. За допомогою останнього можна визначити, чи цей елемент є зображенням (type === 0), чи відео (type === 1).

Після оновлення стану messages компонент у JSX починає рендерити розмову, пробігаючи масив messages методом map. Спочатку визначається, чи повідомлення від поточного користувача, порівнюючи msg.senderID із currentUserId. Це дозволяє застосувати різні стилі для блоків із повідомленнями «від мене» або «від іншого».

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для тестування певних модулів серверу, було було реалізовано набір автоматизованих юніт-тестів на основі фреймворку xUnit та бібліотеки Moq, що дозволяє імітувати залежності без реальних викликів до бази даних або файлової системи [14].

Одним із прикладів є тестування модуля повідомлень (MessageService), яке охоплює критичні сценарії його використання.

Розглянемо певний тестовий випадок з реалізованих автоматизованих тестів наведено в таблиці 5.1.

Таблиця 5.1 – Тестовий випадок (таблиця виконана самостійно)

Інформація про тестовий випадок	
Ідентифікатор тестового випадку	TC03 ver1.0
Власник тесту	Хруб Ю.К
Місцезнаходження тесту (шлях)	TestServer:D:\TestProject\TestSuite\TC03.doc
Дата останнього перегляду	30/11/2024
Технічна вимога, що тестується	FR201
Конфігурація засобів тестування	ST03
Взаємозалежність тестових випадків	Виконати прогін тесту TC02 перед прогоном даного тесту.

Мета тесту	Перевірити, що користувач може успішно увійти відправляти повідомлення через клієнт.	
Методика тестування		
Налаштування прогону теста	Не проводиться	N/A
Крок	**Дія**	**Очікуваний результат**	**Відмітка(V)**
1.	Перейти на сторінку входу в систему ChatterLink.	Користувач успішно входить у систему та бачить список контактів користувача.	V
2.	Ввести коректний Email у поле "Email".	Відкривається чат з обраним контактом.	V
3.	Ввести коректний пароль у поле "Пароль".	Повідомлення з'являється у чаті та відображається як відправлене.	V
4.	Натиснути кнопку "Увійти".	Повідомлення доставлено та відображається у чаті іншого користувача.	V

5.	Вибрати контакт для відправки повідомлення.	Відкривається чат з обраним контактом.	V
6.	Ввести повідомлення у поле введення.	Текст повідомлення з'являється у полі введення.	V
7.	Відправити текстове повідомлення.	Повідомлення доставлено та відображається у чаті іншого користувача.	V
Очистка після прогону теста	Видалення створеного користувача з бази даних для забезпечення чистоти тестового середовища.		N/A
Результати тесту			
Тестувальник: Хруб Ю.К	**Дата прогона теста:** 30/05/2025	**Результат тесту (P/F/B)*: ПРОЙДЕНО (P)**	
Примітка: - Тест успішно пройдено			

Спочатку тестувальник переходить на сторінку входу в систему, перевіряючи, що після введення коректних даних користувач успішно входить і бачить список контактів. Потім в поле «Email» вводиться дійсна електронна адреса, після чого має відкриватися чат з обраним контактом. Далі в поле «Пароль» вводиться правильний пароль, і натискання кнопки «Увійти» запускає відкриття чату та відображення поля введення повідомлення.

Приведений тест був реалізовано у якості модульного тестування слоя сутності, розглянемо частину його реалізації:

```
        int senderID = 1;
        var receiver = new User { UserID = 2, Username = "receiver" };
        var request = new SendMessageRequest
        {
            ReceiverUsername = "receiver",
            MessageText = "Hello",
            Images = null,
            Videos = null
        };
        _userServiceMock
            .Setup(s => s.GetUserByUsernameAsync(request.ReceiverUsername))
            .ReturnsAsync(receiver);
        _messageRepositoryMock
            .Setup(r => r.AddMessageAsync(It.IsAny<Message>()))
            .Returns(Task.CompletedTask)
            .Callback<Message>(m => m.MessageID = 100);
        await _messageService.SendMessageAsync(senderID, request);
        _messageRepositoryMock.Verify(r =>
r.AddMessageAsync(It.IsAny<Message>()), Times.Once);
        _messageRepositoryMock.Verify(r =>
r.AddMediaAsync(It.IsAny<List<Media>>()), Times.Never);
```

Отже, ми імітуємо ситуацію, коли користувач із senderID = 1 надсилає текстове повідомлення «Hello» користувачу з іменем «receiver» після чого викликаємо метод SendMessageAsync. Завдяки налаштуванню _messageRepositoryMock, під час виклику AddMessageAsync створеному об’єкту Message присвоюється MessageID = 100, і сам метод повертає завершений Task, імітуючи успішне збереження повідомлення в базі.

Також серед тестованих функцій можно виділити проставлення вподобайки, розгялнемо частину коду тесту:

```
    int postId = 1, userId = 2;
     var existing = new Like { PostID = postId, UserID = userId };

     _repoMock.Setup(r => r.PostExistsAsync(postId))
```

```
        .ReturnsAsync(true);
_repoMock.Setup(r => r.GetLikeAsync(postId, userId))
        .ReturnsAsync(existing);


await _service.ToggleLikeAsync(postId, userId);


_repoMock.Verify(r => r.PostExistsAsync(postId), Times.Once);
_repoMock.Verify(r => r.GetLikeAsync(postId, userId), Times.Once);
_repoMock.Verify(r => r.RemoveLikeAsync(existing), Times.Once);
_repoMock.Verify(r => r.AddLikeAsync(It.IsAny<Like>()), Times.Never);
```

У цьому тесті ми перевіряємо сценарій, коли користувач із userId = 2 уже поставив лайк на допис із postId = 1. Спочатку ми налаштовуємо мок _repoMock так, щоб при виклику PostExistsAsync(1) він повертав true, а при виклику GetLikeAsync(1, 2) – об’єкт Like, який позначає вже існуючий лайк. Далі виконуємо await _service.ToggleLikeAsync, що має зняти існуючий лайк замість створювати новий. У секції Assert ми перевіряємо, що метод PostExistsAsync(1) викликався саме один раз.

ВИСНОВКИ

У результаті виконання роботи було створено програмну систему для соціальної комунікації. ChatterLink вирішує задачі інтеграції соціальної комунікації та створює комфортний цифровий простір.

Програмне забезпечення успішно реалізує необхідні функції, використовуючи сучасні технології, фреймворки ASP.NET Core, React та інструменти Kotlin і Entity Framework Core, забезпечило високу продуктивність, стабільність, масштабованість і легкість у підтримці системи. Використання архітектури REST API сприяло чіткому розподілу функціональності між серверною та клієнтськими частинами, що дозволило значно спростити процес тестування.

Архітектурні рішення проєкту ChatterLink враховують сучасні підходи до розробки програмного забезпечення: використовується масштабована монолітна архітектура з чітко визначеними рівнями взаємодії (API, сервісний рівень, репозиторії даних). Така структура забезпечує легкість у підтримці, тестуванні та подальшому розширенні функціональності платформи. Використання сучасних технологій, таких як ASP.NET Core, React, Kotlin для мобільного клієнта та Entity Framework Core для взаємодії з базою даних Microsoft SQL Server, дозволяє досягти високої продуктивності та надійності. Завдяки чітко організованій структурі, зручним інтерфейсам та розвинутій функціональності, платформа ChatterLink спрощує взаємодію користувачів, надаючи їм можливість обмінюватися повідомленнями, створювати публікації, коментувати контент, підписуватися на цікаві профілі.

Отже, створена система повністю відповідає заявленим цілям, ефективно об'єднуючи соціальну взаємодію та комунікацію. ChatterLink має потенціал стати зручним та надійним інструментом як для звичайних користувачів, так і для бізнесу, що робить його привабливим рішенням для широкого кола аудиторії. Код застосунку також можно переглянути на сторінці GitHub [15].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Smart Insights. Global social media statistics research summary Smart Insights. – 2025. – URL: https://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/ (дата звернення: 31.05.2025).

2. NextAge. 10 Frameworks That Will Dominate 2025 — And When to Use Them // NextAge. – 9 січня 2025. – URL: https://nextage.com.br/blog/en/10-frameworks-that-will-dominate-2025-and-when-to-use-them/ (дата звернення: 31.05.2025).

3. Мартін Р. Чистий код: створення і рефакторинг за допомогою AGILE. – ФАБУЛА, 2019. – 416 с.

4. Shkarupa D. Platform Showdown: Slack, Discord, Telegram, or WhatsApp? // Know Your Group. – 18.05.2024. – URL: https://gokyg.com/blog/platform-showdown-slack-discord-telegram-or-whatsapp (дата звернення: 31.05.2025).

5. Slack [Електронний ресурс] // Slack. – URL: https://slack.com (дата звернення: 31.05.2025).

6. Microsoft Teams [Електронний ресурс] // Microsoft Teams. – URL: https://teams.microsoft.com (дата звернення: 31.05.2025).

7. Discord [Електронний ресурс] // Discord. – URL: https://discord.com (дата звернення: 31.05.2025).

8. WhatsApp [Електронний ресурс] // WhatsApp. – URL: https://www.whatsapp.com (дата звернення: 31.05.2025).

9. Telegram [Електронний ресурс] // Telegram. – URL: https://telegram.org (дата звернення: 31.05.2025).

10.Telegram API Reference. End-to-End Encryption (Secret Chats). URL: https://core.telegram.org/api/end-to-end (дата звернення: 31.05.2025).

11. Microsoft Learn. Efficient Querying // Microsoft Learn. – 2022. – URL: https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying (дата звернення: 31.05.2025).

12. Atlassian. How indexing works // Atlassian. – 2019. – URL: https://www.atlassian.com/data/sql/how-indexing-works (дата звернення: 31.05.2025).

13. Microsoft. Back up and restore of SQL Server databases // Microsoft Learn. – 2025. – URL: https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver16 (дата звернення: 31.05.2025).

14. Microsoft Learn. Unit test controller logic in ASP.NET Core // Microsoft Learn. – 2025. – URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/testing?view=aspnetcore-9.0 (дата звернення: 31.05.2025).

15. Хруб.Ю.К. ChatterLink // GitHub. – URL: https://github.com/NureKhrubYunis/2025_B_PI_PZPI-21-5_Khrub_Y_K (дата звернення: 01.06.2025).