МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

«ВЕБЗАСТОСУНОК «СОЦІАЛЬНА МЕРЕЖА»»

Спеціальність 122 «Комп'ютерні наук»

122 – КРБ – 401.2010107

Виконав студент 4-го курсу, групи 401
_____ І. Є. Клішевич
«19» червня 2024 р.

Керівник: канд. фіз-мат. наук, доцент
_____І. В. Кулаковська
«19» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Клішевичу Івану Євгеновичу.

1. Тема кваліфікаційної роботи «Вебзастосунок «Соціальна мережа»».
Керівник роботи Кулаковська Інесса Василівна, канд. фіз-мат. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271
2. Строк представлення кваліфікаційної роботи студентом «19» червня 2024 р.
3. Вхідні (початкові) дані до роботи: що являє собою вебзастосунок соціальної мережі; соціальні мережі в сучасному ІТ; інформація про обрані технології та їх технічні характеристики; вимоги до надійності, масштабованості та безпеки вебзастосунку соціальної мережі; специфікації серверів і мережевих налаштувань для реалізації проєкту; аналіз технологічних рішень, включаючи мови програмування та фреймворки. Очікуваний результат: створений вебзастосунок

соціальної мережі, що забезпечує високий рівень надійності, масштабованості та безпеки.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- сутність вебзастосунку соціальної мережі та його роль в сучасному ІТ;
- аналіз сучасних соціальних мереж у сфері ІТ;
- інформація про обрані технології та їх технічні характеристики;
- вимоги до надійності, масштабованості та безпеки вебзастосунку соціальної мережі;
- специфікації серверів і мережевих налаштувань для реалізації проєкту;
- аналіз технологічних рішень, включаючи мови програмування та фреймворки.

5. Перелік графічного матеріалу: 93 сторінок, 26 рисунків, 0 таблиць, 25 використаних джерел та 5 додатків.

6. Завдання до спеціальної частини: «Розробка заходів з поліпшення умов праці»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А. О., доцент кафедри екології	

Керівник роботи __канд. фіз-мат. наук, доцент Кулаковська І. В._____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Клішевич І. Є._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «__» ___ 2024

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «Вебзастосунок «Соціальна мережа»».

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: Аналіз сучасних вебзастосунків соціальних мереж та розробка власного програмного продукту	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
12	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
13	Захист КРБ перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	Виконано

Розробив студент _Клішевич І. Є._____ _____

(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __канд. фіз-мат. наук, доцент Кулаковська І. В. _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_29_» _____01_____ 2024 р

АНОТАЦІЯ

кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили
Клішевича Івана Євгеновича

Тема: «Вебзастосунок «Соціальна мережа»»

Актуальність: Соціальні мережі залишаються актуальними завдяки їхньому впливу на спілкування, розповсюдження інформації та бізнес. Вони зберігають зв'язок між людьми, допомагають швидко дізнаватися про новини та події, а також використовуються компаніями для реклами та залучення клієнтів. Соцмережі стали необхідним елементом сучасного життя.

Об'єкт роботи – процеси проектування та впровадження вебсистеми для підтримки взаємодії користувачів в соціальній мережі.

Предмет роботи – методи та технології, які використовуються для створення вебсистеми, що забезпечує автоматизацію та оптимізацію процесів управління механізмами постів, підписок, взаємодії з контентом та іншими користувачами у соціальній мережі.

Мета кваліфікаційної роботи: розробка соціальної мережі для користувачів з функціоналом мікроблогів, з метою покращення користувацького досвіду та ефективності взаємодії між користувачами, яка буде враховувати переваги та недоліки існуючих рішень на ринку і відповідати потребам користувачів.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка містить вступ, три розділи та висновки.

У першому розділі було проведено аналіз сучасних соціальних мереж. У другому розділі проведено аналіз технологій, що використовуються для створення соціальних мереж, та вибрано найбільш підходящі для реалізації практичної частини. У третьому розділі було розроблено соціальну мережу, сторінку авторизації, головна сторінка, сторінка профілю, сторінка рекомендацій і підписок. Бакалаврська кваліфікаційна робота містить 93 сторінок, 26 рисунків, 0 таблиць, 25 використаних джерел та 5 додатків.

Ключові слова: соціальна мережа, Node.js, React, TypeScipt, Sequilize, JWT.

ABSTRACT

for bachelor's qualification work of a student of group 401 of Petro Mohyla Black Sea National University

Klishevych Ivan

Topic: "Web Application 'Social Network'"

Relevance: Social networks remain relevant due to their impact on communication, information dissemination, and business. They maintain connections between people, help quickly learn about news and events, and are used by companies for advertising and customer engagement. Social networks have become an essential element of modern life.

Object: the processes of designing and implementing a web system to support user interaction in a social network.

Subject: methods and technologies used to create a web system that ensures the automation and optimization of processes for managing posts, subscriptions, interactions with content, and other users in a social network.

Purpose of the qualification work: development of a social network for users with microblogging functionality to improve user experience and efficiency of interaction between users, considering the advantages and disadvantages of existing solutions on the market and meeting the needs of users.

The work consists of a professional section and a special part on occupational safety. The explanatory note contains an introduction, three chapters, and conclusions.

In the first chapter, an analysis of modern social networks was conducted. In the second chapter, an analysis of the technologies used to create social networks was conducted, and the most suitable ones for the practical part were selected. In the third chapter, a social network was developed, including an authorization page, a main page, a profile page, a recommendations page, and a subscriptions page.

The bachelor's qualification work contains 93 pages, 26 figures, 0 tables, 25 used sources, and 5appendices.

Keywords: social network, Node.js, React, TypeScript, Sequelize, JWT.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

CI/CD – Continuous Integration/Continuous Deployment

CMS – Content Management System

CRUD – Create, Read, Update, Delete

CSS – Cascading Style Sheets

DNS – Domain Name System

DOM – Document Object Model

FTP – File Transfer Protocol

GUI – Graphical User Interface

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IP – Internet Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

PHP – Hypertext Preprocessor

REST – Representational State Transfer

SQL – Structured Query Language

UI – User Interface

URL – Uniform Resource Locator

UX – User Experience

V8 – Google Chrome’s JavaScript Engine

ВСТУП

У сучасному світі соціальні мережі стали невід'ємною частиною повсякденного життя мільйонів людей по всьому світу. Вони впливають на наші взаємодії, спосіб спілкування, спосіб сприйняття інформації та навіть на формування громадської думки. Розвиток веб-технологій відкриває безліч можливостей для створення нових соціальних мереж, які відповідають нашим сучасним потребам та вимогам.

Ця робота присвячена дослідженню та розробці вебзастосунку у формі соціальної мережі. Вона спрямована на аналіз ключових аспектів соціальних мереж, вивчення сучасних технологій розробки вебзастосунку, а також на практичне впровадження отриманих знань у створення функціонального і користувацького-орієнтованого середовища для спілкування та обміну інформацією.

Мета: розробка вебзастосунку соціальної мережі, спрямованої на створення платформи для публікації та обміну постами, підписки на інших користувачів та перегляд їхніх профілів. Основною метою є створення функціонального, зручного у використанні та безпечного вебзастосуноку, що відповідає сучасним вимогам та потребам користувачів у створенні та споживанні контенту в онлайн-середовищі.

Для досягнення поставленої мети необхідно ретельно вивчити сучасні підходи до розробки соціальних мереж, спроектувати структуру додатку та його інтерфейс, реалізувати функціонал публікації та перегляду постів, а також систему підписок та профілів користувачів.

1 ДОСЛІДЖЕННЯ СОЦІАЛЬНИХ МЕРЕЖ

1.1 Історія створення соціальних мереж

Історія створення соціальних мереж налічує понад три десятиліття, починаючи з перших спроб об'єднати людей через інтернет і до сучасних глобальних платформ, які змінили спосіб спілкування та взаємодії мільярдів людей по всьому світу [1].

Ранні спроби: 1970-1980-ті роки.

Перші кроки до створення соціальних мереж почалися ще в 1970-х роках з появою електронної пошти та перших чатів. Важливим етапом стало створення Usenet у 1979 році – системи обміну повідомленнями, яка дозволяла користувачам спілкуватися на різні теми в групах новин. Usenet не був справжньою соціальною мережею у сучасному розумінні, але заклав основу для концепції обміну інформацією та обговорень в онлайн-середовищі.

Розвиток в 1990-х роках.

В 1990-х роках з'явилися перші проекти, які можна вважати попередниками сучасних соціальних мереж. Однією з таких платформ стала Six Degrees, створена в 1997 році. Six Degrees дозволяла користувачам створювати профілі, додавати друзів та переглядати їхні зв'язки, що є базовими функціями сучасних соціальних мереж. Попри те, що платформа була закрита у 2001 році, вона продемонструвала потенціал такого формату взаємодії.

Зростання та популяризація: початок 2000-х років.

На початку 2000-х років почалася справжня революція в сфері соціальних мереж. Однією з перших значущих платформ цього періоду стала Friendster, запущена в 2002 році. Friendster запропонувала користувачам розширені можливості для пошуку друзів та взаємодії з ними. Наступним кроком стала поява MySpace у 2003

році, яка швидко стала популярною завдяки можливості користувачів персоналізувати свої профілі та додавати музику [2].

Ера Facebook та глобалізація: середина 2000-х років – сьогодення.

У 2004 році був запущений Facebook, спочатку як платформа для студентів Гарвардського університету. Швидко розширюючись, Facebook став доступним для широкої аудиторії та привернув увагу мільйонів користувачів по всьому світу. Основними факторами успіху Facebook стали зручний інтерфейс, можливість легко знаходити друзів та обмінюватися контентом.

У наступні роки з'явилися інші важливі гравці на ринку соціальних мереж: Twitter (2006), Instagram (2010), LinkedIn (2003), Snapchat (2011) та інші. Кожна з цих платформ привнесла свої інновації, наприклад, Twitter став популярним завдяки своїм коротким повідомленням, а Instagram – завдяки акценту на візуальний контент.

На сьогодні соціальні мережі стали невід'ємною частиною повсякденного життя мільярдів людей. Вони відіграють важливу роль у спілкуванні, бізнесі, маркетингу, розвагах та політиці. Сучасні платформи продовжують розвиватися, інтегруючи нові технології, такі як штучний інтелект, доповнена та віртуальна реальність, що дозволяє створювати ще більш захоплюючі та інтерактивні користувацькі досвіди.

Історія соціальних мереж показує, як швидко можуть розвиватися технології та як вони можуть змінювати спосіб, яким ми взаємодіємо один з одним та з навколишнім світом.

1.2 Характеристики соціальних мереж та інновації в них

Соціальні мережі стали важливою частиною нашого повсякденного життя, пропонуючи широкий спектр можливостей для спілкування, обміну інформацією, розваг та бізнесу. Основні характеристики соціальних мереж включають їхні функції,

можливості взаємодії користувачів та інновації, які роблять ці платформи ефективними та популярними.

Основні характеристики соціальних мереж:

– **профілі користувачів:** соціальні мережі дозволяють користувачам створювати особисті профілі, де вони можуть вказувати інформацію про себе, свої інтереси, освіту, роботу та інші деталі. Профілі часто включають фотографії та відео, що дозволяє користувачам виражати свою індивідуальність;

– **дружні зв'язки:** користувачі можуть додавати один одного до списку друзів або підписників, що дозволяє їм залишатися на зв'язку та обмінюватися контентом. Ці зв'язки формують основну структуру соціальних мереж, створюючи віртуальні спільноти;

– **стрічка новин:** стрічка новин є основним елементом взаємодії, де користувачі бачать оновлення від своїх друзів, сторінок, на які вони підписані, та інших джерел. Стрічка новин забезпечує безперервний потік інформації, новин та контенту;

– **обмін контентом:** соціальні мережі дозволяють користувачам ділитися різноманітним контентом: текстами, фотографіями, відео, посиланнями, подіями та іншим. Це дозволяє користувачам висловлювати свої думки, ділитися важливими моментами життя та створювати цікаві пости;

– **комунікаційні інструменти:** соціальні мережі надають різноманітні засоби для комунікації, включаючи приватні повідомлення, групові чати, коментарі під постами та відеодзвінки. Це дозволяє користувачам спілкуватися як публічно, так і приватно;

– **групи та спільноти:** багато соціальних мереж дозволяють створювати та приєднуватися до груп та спільнот за інтересами. Це можуть бути тематичні групи, професійні спільноти, фан-клуби та інші об'єднання користувачів, що мають спільні

інтереси.

Інновації в соціальних мережах.

– **алгоритми рекомендацій:** соціальні мережі використовують складні алгоритми для персоналізації контенту, який бачать користувачі. Ці алгоритми аналізують поведінку користувачів, їхні вподобання та взаємодії, щоб рекомендувати найбільш релевантний контент, новини та рекламні оголошення;

– **відеоконтент та прямі трансляції:** з розвитком технологій мобільного зв'язку та інтернету, відеоконтент став ключовим елементом соціальних мереж. Платформи, такі як YouTube, Instagram та TikTok, спеціалізуються на відео. Прямі трансляції дозволяють користувачам взаємодіяти з аудиторією в режимі реального часу, що створює більш автентичний та інтерактивний досвід;

– **інтеграція доповненої та віртуальної реальності:** доповнена реальність (AR) та віртуальна реальність (VR) відкривають нові можливості для взаємодії користувачів. Наприклад, фільтри для фото та відео в Instagram та Snapchat використовують технологію AR, що дозволяє створювати захоплюючі та креативні ефекти;

– **інтерактивні сторіс:** формат "сторіс", впроваджений Snapchat і популяризований Instagram, дозволяє користувачам публікувати короткочасний контент, що зникає через 24 години. Це створює відчуття терміновості та ексклюзивності, заохочуючи користувачів частіше взаємодіяти з платформою;

– **месенджери та чат-боти:** інтегровані месенджери, такі як Facebook Messenger та WhatsApp, стають важливими комунікаційними інструментами всередині соціальних мереж. Чат-боти, які використовують штучний інтелект, допомагають автоматизувати відповіді на запитання користувачів, підтримувати клієнтів та навіть проводити транзакції;

– **соціальні комерційні платформи:** соціальні мережі все більше

інтегруються з електронною комерцією. Платформи, такі як Instagram і Facebook, пропонують можливості для покупок безпосередньо через додатки, що дозволяє брендам безпосередньо продавати свої товари та послуги користувачам;

– **захист даних та конфіденційність:** однією з важливих інновацій стало підвищення рівня захисту даних користувачів. Соціальні мережі постійно вдосконалюють свої політики конфіденційності та впроваджують нові технології для захисту особистої інформації, враховуючи зростаючі занепокоєння щодо безпеки даних.

Соціальні мережі продовжують еволюціонувати, пропонуючи нові функції та можливості для користувачів. Вони стали не лише платформами для спілкування, а й важливими інструментами для бізнесу, маркетингу, новин та розваг. Інновації в цій сфері сприяють розвитку технологій та змінюють спосіб, яким ми взаємодіємо зі світом навколо нас [3].

1.3 Аналіз наявних соціальних мереж та технологій

1.3.1 Аналіз Twitter

Переваги:

– **швидкість і актуальність**, Твіттер володіє вражаючою швидкістю поширення інформації, що дозволяє користувачам швидко дізнаватися про актуальні події та новини;

– **глобальне охоплення**, платформа має велику аудиторію, що охоплює користувачів з усього світу, що робить її ідеальним майданчиком для спілкування та взаємодії з широким спектром людей;

– **можливість взаємодії**, користувачі можуть коментувати, репости, лайкати та відповідати на повідомлення інших користувачів, що створює активну та захоплюючу спільноту;

– **сприяння відкритому діалогу**, Твіттер відомий своєю відкритістю до різних точок зору та здатністю сприяти діалогу між користувачами з різними поглядами;

– **зручний пошук та хештеги**, платформа надає зручний пошук за хештегами, що дозволяє знаходити та приєднуватися до різноманітних обговорень та тематичних груп.

Недоліки:

– **обмежена кількість символів**, хоча 280 символів може бути достатньо для коротких повідомлень, воно також обмежує можливості для розгортання складних думок та ідей;

– **проблеми зі змістом**, багато користувачів обирають поділитися повідомленнями коротко та прямо, що може привести до спрощення та спотворення інформації;

– **тролінг та негативний контент**, як і в більшості соціальних мереж, Твіттер стикається з проблемами тролінгу, ненависті та негативного контенту, що може впливати на користувацький досвід;

– **алгоритмічна фільтрація**, деякі користувачі скаржаться на алгоритмічну фільтрацію, яка може обмежувати їхню ефективність у досягненні аудиторії [4].

1.3.2 Аналіз Mastodon

Переваги:

– **децентралізація**, Mastodon базується на децентралізованій мережі, що дозволяє користувачам обирати сервери за своїм вибором та контролювати свої дані;

– **відкритий код**, платформа має відкритий код, що сприяє розвитку та вдосконаленню за участю спільноти;

– **безкоштовне використання**, Mastodon надає безкоштовну можливість створювати облікові записи та обмінюватися повідомленнями.

Недоліки:

– **невелика аудиторія**, в порівнянні з іншими соціальними мережами, Mastodon має меншу аудиторію, що може обмежити можливості для зв'язку;

– **складніше використання**. Децентралізована природа Mastodon може здаватися складною для новачків у порівнянні з централізованими платформами [5].

1.3.3 Аналіз Gab

Переваги:

– **свобода слова**, Gab активно пропагує свободу слова і створює середовище для обміну ідеями без цензури;

– **приватність**, платформа надає засоби для збереження приватності користувачів та захисту їхніх особистих даних;

– **можливість публікації різноманітного контенту**, Габ дозволяє користувачам публікувати текстові повідомлення, фото, відео та інший контент.

Недоліки:

– **контент**, що пропагує ненависть, через свою політику свободи слова, Gab може бути схильний до розповсюдження контенту, який пропагує ненависть та дискримінацію;

– **обмежена популярність**, у порівнянні з більш відомими соціальними мережами, Gab має обмежену аудиторію [6].

1.3.4 Аналіз Parler

Переваги:

– **свобода слова**, платформа активно пропагує свободу слова та відсутність цензури;

– **приватність**, Parler надає інструменти для захисту приватності користувачів та контролю над їхніми особистими даними.

Недоліки:

– **послаблені правила модерації**, Parler може бути схильний до поширення контенту, який порушує правила щодо ненависті та дискримінації через свої послаблені правила модерації;

– **обмежена функціональність**, у порівнянні з більш відомими соціальними мережами, Parler може мати обмежену функціональність та можливості [7].

1.3.5 Аналіз Minds

Переваги:

– **свобода слова**, Minds пропагує свободу слова та створює середовище для обміну ідеями без цензури;

– **криптовалюта**, платформа пропонує систему криптовалюти, що дозволяє користувачам отримувати винагороду за активність на сайті.

Недоліки:

– **обмежена популярність**, у порівнянні з іншими соціальними мережами, Minds має обмежену аудиторію, що може обмежити можливості для зв'язку та взаємодії;

– **невиправдані очікування щодо криптовалюти**, система криптовалюти може бути складною для розуміння та використання для деяких користувачів[8].

Висновки до розділу 1

Після проведення аналізу можна визначити спільну проблему для соціальних мереж, таких як Twitter, Mastodon, Gab, Parler, Minds , а саме проблему негативного контенту та недостатньої модерації. Багато з цих платформ, які активно підтримують свободу слова та відкритість, можуть стикатися з проблемами, такими як розповсюдження ненависті, дискримінації, спаму та іншого негативного вмісту. Це

може негативно впливати на користувацький досвід та затемнювати позитивні аспекти соціальних мереж, такі як сприяння відкритому діалогу та комунікації.

Отже, одним із ключових завдань для цих платформ є розробка та впровадження ефективних стратегій модерації контенту, які б дозволили зберегти баланс між свободою слова та захистом від небезпечного чи шкідливого вмісту. Тільки таким чином вони зможуть забезпечити безпечне та приємне середовище для своїх користувачів.

2 ДОСЛІДЖЕННЯ ТА ВИБІР ТЕХНОЛОГІЧНИХ РІШЕНЬ

2.1 Дослідження технологічних рішень популярних платформ

2.1.1 Twitter

Twitter є однією з найпопулярніших соціальних мереж у світі, яка має унікальні характеристики та функції, що зробили її популярною серед мільйонів користувачів. Розглянемо деякі ключові аспекти платформи, її технологічні рішення та архітектуру.

Технологічні рішення:

– фронтенд: Twitter використовує різноманітні веб–технології для створення інтуїтивно зрозумілого інтерфейсу. Це включає HTML, CSS та JavaScript, а також сучасні фреймворки, такі як React, що забезпечує швидке оновлення контенту без необхідності перезавантаження сторінки;

– бекенд: Twitter побудована на основі масштабованих та надійних технологій, таких як Scala, Java та Ruby. Для забезпечення високої продуктивності та обробки великої кількості запитів використовуються різноманітні бази даних, включаючи MySQL та Cassandra;

– обробка даних: Twitter використовує Hadoop для зберігання та обробки великих обсягів даних. Це дозволяє аналізувати поведінку користувачів, визначати тренди та покращувати рекомендації [9].

Архітектура Twitter є складною та розподіленою, що дозволяє обробляти мільйони запитів на секунду.

Мікросервіси Twitter застосовує архітектуру мікросервісів, що дозволяє розділити функціональність на окремі, незалежні сервіси. Це полегшує масштабування та обслуговування системи.

Відкрита API Twitter дозволяє розробникам створювати сторонні додатки, що інтегруються з платформою. Це сприяє розвитку екосистеми додатків навколо Twitter.

Кешування для підвищення швидкості роботи використовується кешування за допомогою Memcached та Redis. Це зменшує навантаження на базу даних та покращує час відгуку системи.

Безпека та конфіденційність.

Twitter приділяє велику увагу безпеці користувачів. Впроваджено багатофакторну аутентифікацію, шифрування даних та регулярні перевірки безпеки, що забезпечує захист особистої інформації користувачів.

Монетизація.

Twitter реалізував різноманітні стратегії монетизації, включаючи рекламні твіти, промо–аккаунти та промо-тренди. Це дозволяє брендам та компаніям просувати свій контент серед цільової аудиторії.

2.1.2 Facebook

Facebook є найбільшою соціальною мережею у світі, з мільярдами активних користувачів щомісяця. Платформа надає широкий спектр функцій, включаючи можливість публікації статусів, обміну фотографіями та відео, створення подій, груп та сторінок.

Технологічні рішення:

– фронтенд: Facebook активно використовує React, JavaScript бібліотеку, розроблену всередині компанії, для створення динамічних та інтерактивних інтерфейсів. Це дозволяє оновлювати лише необхідні частини сторінки без перезавантаження;

– бекенд: основні мови програмування для серверної частини включають PHP, з оптимізаціями у вигляді мови Hack. Використання HHVM (HipHop Virtual Machine) забезпечує підвищену продуктивність PHP-коду;

– обробка даних: Facebook застосовує Hadoop для обробки великих обсягів

даних. Це дозволяє здійснювати аналіз користувацької поведінки, персоналізувати контент та покращувати рекомендаційні системи.

Архітектура системи.

Розподілена архітектура Facebook використовує розподілену архітектуру для забезпечення високої доступності та масштабованості. Дані зберігаються у розподілених базах даних, таких як MySQL та HBase.

Мікросервіси для кращої масштабованості та обслуговування, Facebook використовує архітектуру мікросервісів. Кожен сервіс відповідає за окрему функціональність та може незалежно масштабуватися.

Використання кешування за допомогою Memcached та TAO (Facebook's graph database) дозволяє зменшити навантаження на основні бази даних та підвищити швидкість відповіді системи.

Безпека та конфіденційність.

Захист даних Facebook впровадив різні заходи для захисту даних користувачів, включаючи багатофакторну аутентифікацію, шифрування даних та регулярні перевірки безпеки.

Платформа дотримується міжнародних стандартів та регуляцій щодо конфіденційності, таких як GDPR.

Монетизація

Facebook реалізує різноманітні стратегії монетизації, включаючи таргетовану рекламу, спонсоровані пости, рекламні відео та продаж даних для аналітичних цілей.

2.1.3 Instagram

Instagram, придбаний Facebook у 2012 році, є соціальною мережею, орієнтованою на візуальний контент, включаючи фотографії та відео. Платформа також пропонує функції сторіс, прямі трансляції та торговельні можливості.

Дослідження популярних платформ для створення соціальних мереж показує, що кожна з них має свої унікальні характеристики та технологічні рішення. Вибір відповідної платформи залежить від цілей проекту, аудиторії та функціональних вимог. Використання передових технологій та інновацій дозволяє створювати ефективні та масштабовані соціальні мережі, що відповідають потребам користувачів.

Технологічні рішення:

– фронтенд: Instagram використовує React Native для розробки мобільних додатків, що забезпечує кросплатформну сумісність та ефективність;

– бекенд: серверна частина побудована на Python та Django, що забезпечує швидкий розвиток та надійність. Також використовується Celery для асинхронних завдань та обробки черг;

– обробка даних: Instagram застосовує PostgreSQL для зберігання даних користувачів та Cassandra для зберігання часових рядів даних, таких як лайки та коментарі.

Архітектура системи.

Мікросервіси, як і Facebook, Instagram використовує архітектуру мікросервісів для кращого управління та масштабування окремих компонентів системи.

Використання кешування за допомогою Redis та Memcached дозволяє швидко обробляти запити та зменшити навантаження на базу даних.

Безпека та конфіденційність.

Instagram застосовує шифрування даних на всіх рівнях для забезпечення конфіденційності користувачів.

Двофакторна аутентифікація та постійний моніторинг безпеки допомагають запобігти несанкціонованому доступу

Монетизація.

Instagram пропонує різноманітні рекламні можливості, включаючи спонсоровані пости, історії та відео. Інтеграція з Facebook Ads дозволяє ефективно таргетувати аудиторію.

2.1.4 LinkedIn

LinkedIn є професійною соціальною мережею, яка спеціалізується на кар'єрних можливостях, мережевих зв'язках та обміні професійним контентом.

Технологічні рішення:

– фронтенд: LinkedIn використовує Ember.js для побудови клієнтської частини, що забезпечує динамічний та інтерактивний користувацький інтерфейс;

– бекенд: основні технології включають Java та Scala, що забезпечують високу продуктивність та надійність. LinkedIn також використовує Play Framework для швидкої розробки веб-додатків;

– обробка даних: LinkedIn застосовує Kafka для обробки потоків даних та Voldemort для зберігання розподілених даних.

Архітектури системи.

LinkedIn використовує мікросервісну архітектуру для розподілу функцій та забезпечення масштабованості.

Кешування, використання Espresso для зберігання даних у пам'яті та кешування забезпечує швидкий доступ до найбільш запитуваної інформації.

Безпека та конфіденційність.

LinkedIn активно працює над забезпеченням конфіденційності даних користувачів, застосовуючи різні методи шифрування та багатофакторної аутентифікації.

Модерація та перевірка контенту допомагають підтримувати високу якість та професійність платформи.

Монетизація.

LinkedIn монетизується через преміум-підписки, рекламу та рекрутингові рішення, які допомагають компаніям знаходити та наймати талановитих працівників.

Дослідження популярних платформ для створення соціальних мереж показує різноманітність технологічних рішень та архітектура, які використовуються для забезпечення високої продуктивності, надійності та безпеки. Кожна платформа має свої унікальні характеристики, які визначають її успіх та популярність серед користувачів. Використання передових технологій та інновацій дозволяє створювати ефективні соціальні мережі, що відповідають потребам сучасного суспільства.

2.2 Використанні технологій

React - це бібліотека JavaScript для створення інтерфейсів користувача. Він був розроблений Facebook і вперше представлений у 2013 році. З того часу React став основою для численних додатків, включаючи саму платформу Facebook [10].

Ось деякі ключові особливості React:

– **компонентний підхід React**, дозволяє вам будувати інтерфейси користувача з окремих частин, які називаються компонентами. Компоненти можуть бути повторно використані, що полегшує розробку та підтримку коду;

– **JSX** React використовує JSX, розширення синтаксису JavaScript, яке дозволяє вам писати HTML-подібний код безпосередньо в JavaScript. Це робить код більш читабельним та зрозумілим;

– **стан**, компоненти в React можуть мати свій власний стан. Стан - це дані, які компонент використовує та може змінювати. Коли стан компонента змінюється, React автоматично оновлює відображення компонента;

– **віртуальний DOM**, React використовує віртуальний DOM для ефективного оновлення та рендерингу компонентів1. Він порівнює поточний стан інтерфейсу користувача з новим станом та вносить лише мінімальні зміни, необхідні для оновлення відображення. React широко використовується для веб-розробки, але також може використовуватися для створення мобільних додатків за допомогою React Native та настільних додатків за допомогою Electron [11].

Express.js - це найпопулярніший веб-фреймворк для Node.js. Він був розроблений для створення веб-додатків та API з простим та гнучким API.

Ось деякі ключові особливості Express.js:

– **проміжне програмне забезпечення та маршрутизація**, Express.js спрощує організацію функціональності вашого додатку за допомогою проміжного програмного забезпечення та маршрутизації. Функції проміжного програмного забезпечення дозволяють обробляти завдання, такі як аутентифікація, ведення журналів та обробка помилок. Маршрутизація гарантує, що вхідні запити направляються до відповідних обробників;

– **мінімалістичний дизайн**, Express.js слідує простій та мінімалістичній філософії дизайну. Ця простота дозволяє вам швидко налаштувати сервер, визначити маршрути та ефективно обробляти HTTP-запити;

– **гнучкість та налаштування**, Express.js не накладає строгу архітектуру додатку. Ви можете структурувати свій код відповідно до своїх вподобань1. Незалежно від того, створюєте ви RESTful API або повноцінний веб-додаток, Express.js адаптується до ваших потреб;

– **масштабованість**, розроблений для того, щоб бути легким та масштабованим, Express.js обробляє велику кількість запитів асинхронно. Його архітектура, заснована на подіях, забезпечує відгук, навіть при великих навантаженнях;

– **активна підтримка спільноти**, з активною спільнотою, Express.js регулярно отримує оновлення та поліпшення. Ви знайдете достатньо документації, посібників та плагінів, щоб покращити свій досвід розробки [12].

Node.js - це кросплатформне, відкрите середовище виконання JavaScript, яке може працювати на Windows, Linux, Unix, macOS та багатьох інших системах. Node.js працює на двигуні JavaScript V8 та виконує код JavaScript поза веб-браузером. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для серверного скриптингу [13].

Ось декілька ключових особливостей Node.js:

– **асинхронне програмування**, Node.js використовує асинхронне програмування, що дозволяє йому ефективно обробляти велику кількість запитів. Завдяки цьому Node.js може обробляти великі обсяги даних, що робить його ідеальним для реального часу та інтенсивних за даними застосунків;

– **модулі/пакети**, Node.js має npm, менеджер пакетів node, з бібліотекою понад 350 000 пакетів, що допомагає швидко розпочати роботу над вашим проектом або застосунком;

– **Google Chrome V8 JavaScript Engine** – це середовище виконання побудоване на двигуні JavaScript Google Chrome V8. Так само, як віртуальна машина Java перекладає байт-код, двигун JavaScript Chrome V8 бере JavaScript і робить його зрозумілим;

– **відкрите джерело та кросплатформність**, Node.js є відкритим джерелом і може працювати на різних платформах, включаючи MacOS, Linux та Windows [14].

PostgreSQL - це потужна, відкрита система об’єктно-реляційних баз даних. Вона використовує та розширює мову SQL, додаючи багато функцій, які безпечно зберігають та масштабують найскладніші робочі навантаження даних. PostgreSQL

працює на всіх основних операційних системах, включаючи Linux, FreeBSD, OpenBSD, macOS та Windows [15].

Ось декілька ключових особливостей PostgreSQL:

– **об’єктно-реляційна база даних**, PostgreSQL дозволяє вам визначати свої власні типи даних, створювати власні функції, навіть писати код на різних мовах програмування без перекомпіляції вашої бази даних;

– **ACID-сумісність**, PostgreSQL є ACID-сумісною базою даних з 2001 року, що означає, що вона підтримує атомарність, узгодженість, ізоляцію та стійкість транзакцій;

– **розширюваність**, PostgreSQL є високо розширюваною базою даних. Ви можете визначати свої власні типи даних, створювати власні функції, навіть писати код на різних мовах програмування без перекомпіляції вашої бази даних;

– **підтримка спільноти**, з активною спільнотою, PostgreSQL регулярно отримує оновлення та поліпшення. Ви знайдете достатньо документації, посібників та плагінів, щоб покращити свій досвід розробки.

Docker - це відкрита платформа, яка спрощує та автоматизує контейнеризацію, процес пакування та запуску додатків разом з їх залежностями. Docker дозволяє вам відокремити ваші додатки від вашої інфраструктури, щоб швидко доставляти програмне забезпечення. З Docker ви можете керувати своєю інфраструктурою так само, як ви керуєте своїми додатками.

Ось декілька ключових особливостей Docker:

– **швидка, послідовна доставка ваших додатків**, Docker спрощує життєвий цикл розробки, дозволяючи розробникам працювати в стандартизованих середовищах, використовуючи локальні контейнери, які надають ваші додатки та послуги. Контейнери чудово підходять для неперервної інтеграції та неперервної доставки (CI/CD) робочих процесів;

– **відповідне розгортання та масштабування**, контейнерна платформа Docker дозволяє виконувати високо портативні робочі навантаження. Docker контейнери можуть працювати на локальному ноутбуці розробника, на фізичних або віртуальних машинах в дата-центрі, на хмарних провайдерах або в суміші середовищ. Легкість та портативність Docker також робить легким динамічне управління робочими навантаженнями, масштабування додатків та послуг вгору або вниз відповідно до бізнес-потреб, майже в реальному часі;

– **виконання більше робочих навантажень на тому ж обладнанні**, Docker легкий та швидкий. Він надає реальну, економічно вигідну альтернативу віртуальним машинам на основі гіпервізора, тому ви можете використовувати більше своєї серверної потужності для досягнення своїх бізнес-цілей;

– **активна підтримка спільноти**, з активною спільнотою, Docker регулярно отримує оновлення та поліпшення. Ви знайдете достатньо документації, посібників та плагінів, щоб покращити свій досвід розробки [16].

Sequelize — це ORM (Object-Relational Mapping) бібліотека для Node.js, яка дозволяє розробникам легко працювати з реляційними базами даних, такими як PostgreSQL, MySQL, MariaDB, SQLite та MSSQL. Вона забезпечує абстракцію над SQL-запитами, що дозволяє писати код для взаємодії з базою даних, використовуючи об'єктно-орієнтований підхід.

Основні можливості Sequelize:

– моделі, Sequelize дозволяє визначати моделі, які представляють таблиці у базі даних. Кожна модель відображає структуру таблиці, включаючи поля та їх типи;

– запити, використовуючи Sequelize, можна легко виконувати CRUD (Create, Read, Update, Delete) операції над даними без написання складних SQL-запитів;

– асоціації, Sequelize підтримує визначення зв'язків між моделями (один до одного, один до багатьох, багато до багатьох);

– міграції, Sequelize надає інструменти для управління версіями схеми бази даних, що дозволяє легко застосовувати та відміняти зміни в структурі таблиць;

– валідації та хуки, можливість додавати валідації для полів моделі та використовувати хуки для виконання логіки до або після певних операцій.

Sequelize — це потужний інструмент для роботи з реляційними базами даних у середовищі Node.js, який значно спрощує взаємодію з базами даних, надаючи зрозумілий та зручний API для розробників. Завдяки Sequelize можна легко визначати моделі, виконувати запити, управляти міграціями та використовувати асоціації для створення складних взаємозв'язків між даними [17].

REST (Representational State Transfer) — це архітектурний стиль, що використовується для створення веб-сервісів. REST API дозволяє взаємодіяти з ресурсами через HTTP-запити. Це простий і ефективний спосіб для передачі даних між клієнтськими додатками та сервером [18].

Основні принципи REST.

Ресурси можуть бути будь-чим, що може бути ідентифіковано у веб-сервісі, наприклад, користувачі, замовлення, товари тощо.

URI (Uniform Resource Identifier), кожен ресурс має унікальний URI, який використовується для його ідентифікації.

REST використовує стандартні методи HTTP для операцій над ресурсами:

– GET: отримання ресурсу або колекції ресурсів;

– POST: створення нового ресурсу;

– PUT: оновлення існуючого ресурсу;

– DELETE: видалення ресурсу;

– PATCH: часткове оновлення ресурсу.

HTTP-запит у REST API має наступну структуру:

– метод: HTTP-метод (GET, POST, PUT, DELETE);

– URI: шлях до ресурсу;

– заголовки (Headers): містять метаінформацію про запит, наприклад, формат даних (Content-Type), авторизацію (Authorization) тощо;

– тіло (Body): Вміст запиту, зазвичай використовується в POST, PUT та PATCH запитах для передачі даних.

Приклади REST API запитів:

– GET /users: отримання списку всіх користувачів;

– GET /users/1: отримання інформації про користувача з ID 1;

– POST /users: створення нового користувача;

– PUT /users/1: оновлення інформації про користувача з ID 1

– DELETE /users/1: видалення користувача з ID 1.

2.3 Архітектура вебзастосунку

Архітектура клієнт-сервер полягає в розподілі завдань між клієнтами та серверами. Клієнт запитує сервіси або ресурси, а сервер обробляє ці запити та надає відповідні дані. Це дозволяє розподілити навантаження та покращити масштабованість системи.

Для покращення зручності обслуговування та гнучкості застосунки часто діляться на кілька логічних рівнів. Рівень є абстракцією, призначеною для задоволення певної потреби бізнесу.

Кожному рівню надається певний набір обов’язків, і він може мати доступ лише до нижчого рівня або на тому самому рівні (тобто розділення завдань) [19].

На практичному рівні цей низхідний підхід дозволяє розробникам легко організовувати свій код і змінювати деталі реалізації одного або кількох рівнів, не впливаючи на всю програму.

На рисунку 2.1 зображений зв’язок між клієнтом, сервером та базою даних.

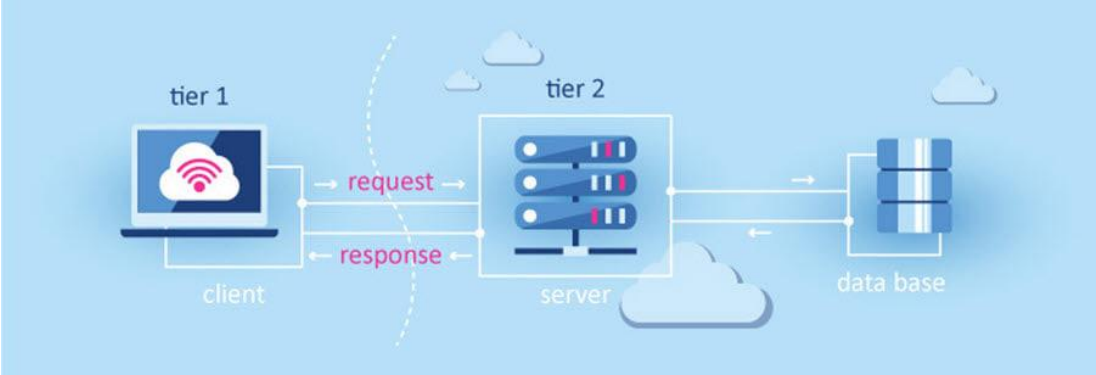

Рисунок 2.1 – Трирівнева архітектура [20]

Клієнтська частина веб-архітектури займає верхній рівень і відповідає за відображення інформації, яка пов'язана з веб-службами. Ця інформація зазвичай доступна через веб-браузер або веб-додаток у формі графічного інтерфейсу користувача (GUI), який надає інтерфейс для безпосередньої взаємодії з кінцевими користувачами. Цей рівень зазвичай розробляється на основі веб-технологій, таких як CSS або JavaScript, і взаємодіє з іншими рівнями, передаючи результати до браузера та інших компонентів системи. Клієнтська частина може включати різні форми взаємодії з користувачем, такі як форми введення даних, кнопки навігації та інтерактивні елементи.

Серверна частина, також відома як середній рівень, рівень логіки або бізнес-логіки, відокремлена від рівня презентації. Коли користувачі взаємодіють з веб-сайтом, вводячи вхідні дані, такі як клацання миші або введення тексту, серверна частина керує програмуванням виходів, наприклад текстом, який з'являється на екрані. Внутрішня веб-архітектура визначає, як різні елементи коду взаємодіють для обробки вхідних даних від користувача та створення відповідних виходів [21].

Створення архітектури на серверній частині веб-сайту дозволяє створювати оптимізовані веб-сайти, які надають користувачам можливість легко переміщатися по

сторінках та використовувати різні функції, незважаючи на складність коду і без використання зовнішніх пристроїв. Це також дозволяє користувачам отримувати доступ до веб-сайтів без використання виключно обчислювальної потужності свого комп’ютера, що робить веб-сайти більш доступними [22].

База даних в архітектурі веб-додатків часто відома як рівень даних або рівень зберігання даних. Цей рівень відповідає за зберігання, відновлення та обробку даних, які використовуються в додатку. Ось деякі ключові функції рівня бази даних:

– **зберігання даних**, база даних зберігає всю інформацію, яка потрібна для роботи веб-додатку. Це можуть бути дані користувачів, інформація про продукти, транзакції, історія взаємодії користувачів та багато іншого;

– **обробка запитів**, база даних обробляє запити на читання та запис, які надходять від серверного рівня. Це включає в себе вибірку, вставку, оновлення та видалення даних;

– **безпека даних**, база даних забезпечує безпеку даних, контролюючи доступ до інформації. Це включає в себе авторизацію користувачів, шифрування даних та резервне копіювання;

– **оптимізація запитів**, бази даних використовують різні алгоритми та структури даних для ефективної обробки великих обсягів даних та складних запитів;

– **інтеграція даних**, база даних дозволяє інтегрувати дані з різних джерел і представляти їх у єдиному форматі;

– **підтримка транзакцій**, бази даних забезпечують механізми для управління транзакціями, що дозволяє забезпечити цілісність даних при одночасному доступі до них декількох користувачів [23].

Взаємодія між компонентами:

а) клієнт відправляє запит до сервера, користувач взаємодіє з клієнтським додатком, який генерує запит (наприклад, натискання кнопки для отримання списку

користувачів). Цей запит відправляється до сервера через HTTP. (наприклад, користувач натискає кнопку для отримання списку користувачів, клієнт відправляє запит GET /users до сервера);

б) сервер обробляє запит. Сервер отримує запит від клієнта, аналізує його, та виконує відповідні дії. Якщо запит вимагає доступу до даних, сервер взаємодіє з базою даних (нариклад, сервер отримує запит GET /users, обробляє його та викликає запит до бази даних для отримання списку користувачів);

в) сервер взаємодіє з базою даних. Сервер використовує SQL (для реляційних баз даних) або інші запити (для нереляційних баз даних) для отримання необхідної інформації. (наприклад: Сервер виконує SQL-запит SELECT * FROM users для отримання списку всіх користувачів);

г) сервер повертає відповідь клієнту, отримавши дані з бази даних. Сервер формує відповідь та відправляє її назад клієнту через HTTP. (наприклад сервер отримує результат запиту з бази даних, формує JSON-відповідь з даними користувачів та відправляє її клієнту);

д) клієнт отримує відповідь та відображає дані. Клієнт отримує відповідь від сервера, обробляє її та відображає дані у користувацькому інтерфейсі (наприклад, клієнт отримує JSON-відповідь зі списком користувачів та відображає його на екрані).

Рівень маршрутизатора.

У трирівневій архітектурі рівень маршрутизатора є першим рівнем, який містить маршрути API програми та відповідає за:

– **розбір і перевірка корисного навантаження вхідних запитів**, надісланих клієнтом, щоб позбавити їх будь-яких властивостей, специфічних для HTTP;

– **пересилання проаналізованих даних на рівень обслуговування**, відповідальний за обробку бізнес-логіки програми;

– **перетворення результату виклику**, зробленого на рівень обслуговування,

у дійсну відповідь HTTP перед надсиланням назад клієнту.

Сервісний рівень розташований між рівнем маршрутизатора та рівнем доступу до даних. Він абсолютно незалежний від будь-якого транспортного механізму, такого як HTTP або AMQP, що означає, що він може отримувати дані з багатьох джерел і ефективно їх обробляти .

Цей рівень містить бізнес-логіку програми та відповідає за:

– **виконання завдань**, пов’язаних із програмою, з використанням проаналізованих і підтверджених даних, надісланих рівнем маршрутизатора, відповідно до визначеного набору бізнес-правил (наприклад, створення маркерів нових сеансів, надсилання електронних листів тощо);

– **виклик рівня доступу до даних у випадку**, якщо йому потрібно зв’язатися із зовнішнім компонентом, таким як база даних.

Рівень доступу до даних — це рівень, який відповідає за виконання операцій введення/виведення поза межами програми, наприклад зв’язок із базою даних для виконання операцій CRUD (тобто створення, читання, оновлення, видалення).

Одним із найпростіших способів досягти цього є використання ORM або ODM, бібліотеки, яка автоматизує передачу даних, в об’єкти, які частіше використовуються в застосунках. Тому ORM забезпечує високорівневу абстракцію бази даних, що дозволяє розробникам писати об’єктно-орієнтований код замість запитів до бази даних, або збережених процедур для виконання операцій CRUD [24].

2.4 Опис структури клієнтської та серверної частини.

На рисунках 2.2 – 2.3 зображена структура вебзастосунку

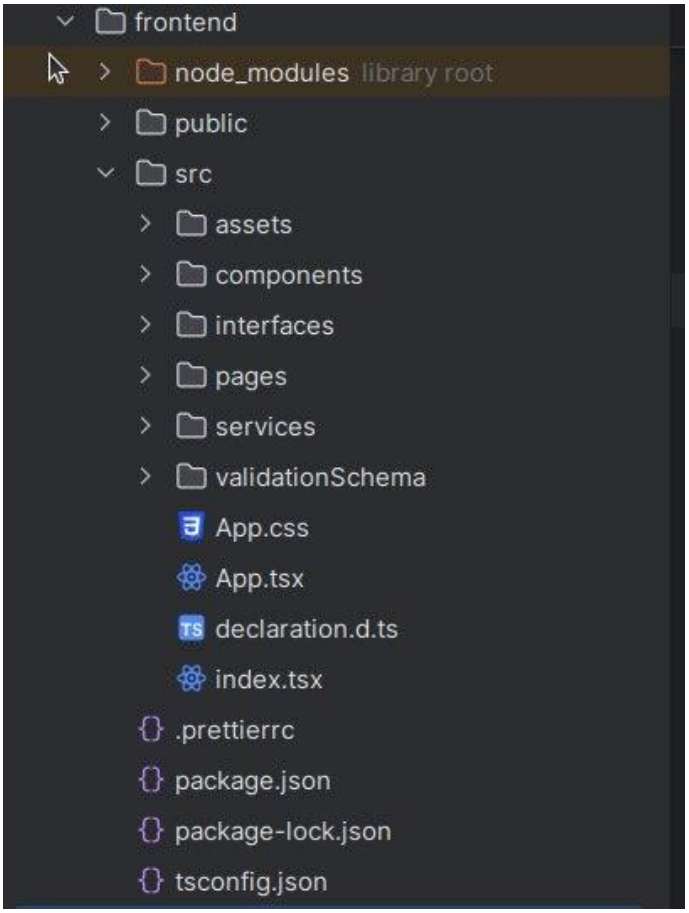

Рисунок 2.2 – Структура клієнтської частини

Опис структури клієнтської частини:

– public/index.html, це основний HTML-файл проекту. Він служить як вихідна точка для додатку. React вставляє свої компоненти в div з id “root” в цьому файлі;

– src/components/App/App.js, це головний компонент додатку. Він зазвичай містить маршрутизацію та загальний макет додатку;

– src/components/Header/Header.js, цей компонент відображає заголовок вашого додатку. Він може містити навігаційні посилання, логотип та інші елементи, які ви хочете показати в заголовку вашого додатку;

– src/components/Footer/Footer.js, цей компонент відображає нижній колонтитул додатку. Він може містити інформацію про авторські права, посилання на

політику конфіденційності та інші ресурси;

– src/components/Friends/FriendsList.js, цей компонент відображає список друзів користувача;

– src/services/api.js, цей файл містить всю логіку взаємодії з API. Він містить функції для виконання GET, POST, PUT та DELETE запитів до API;

– src/index.js, це вхідна точка додатку React. Він рендерить кореневий компонент додатку (<App />) в div з id “root” в public/index.html;

– package.json, цей файл містить список всіх залежностей проекту, а також іншу інформацію про ваш проект, таку як назва проекту, версія та автор;

– README.md, цей файл містить інформацію про проект, включаючи інструкції з встановлення та використання, а також інформацію про ліцензію.

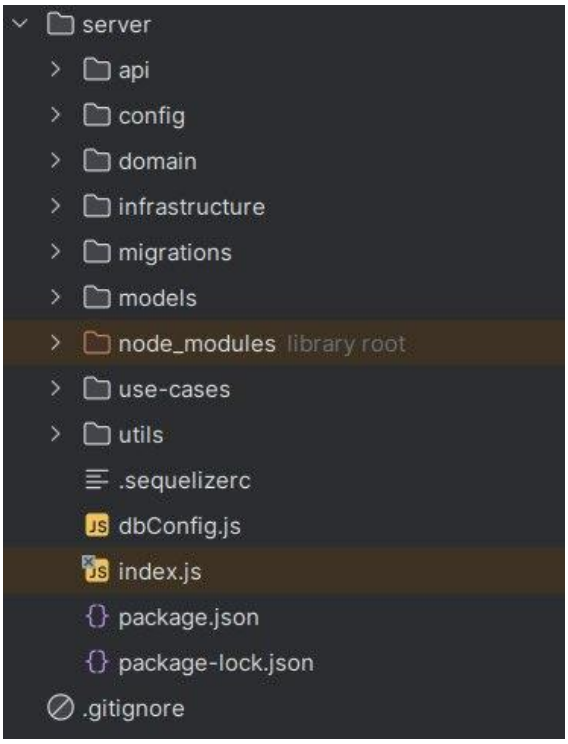

Рисунок 2.3 – Структура серверної частини

Опис структури серверної частини:

– api: папка для обробників запитів (routes, controllers). Тут зберігаються файли, які обробляють HTTP-запити і відповідають за виклик відповідних сервісів та

логіки додатку;

– config/: конфігураційні файли додатку, такі як налаштування для підключення до бази даних, змінні середовища тощо;

– domain/: містить бізнес-логіку додатку, зокрема, правила і принципи, що стосуються конкретних предметних областей;

– infrastructure/: класичний шар інфраструктури, який може включати взаємодію з базою даних, сторонніми API та іншими зовнішніми сервісами;

– migrations/: міграції для бази даних, використовуються для версіонування схем бази даних та застосування змін;

– models/: моделі бази даних, які відображають структуру даних та їх взаємозв’язки;

– node_modules/: папка, де зберігаються всі залежності Node.js;

– use-cases/: використання випадків (use cases) або сервіси, що містять основну бізнес-логіку додатку;

– utils/: утилітні функції та хелпери, які використовуються в різних частинах додатку;

– .sequelizerc: файл конфігурації для Sequelize CLI;

– dbConfig.js: файл конфігурації для підключення до бази даних;

– index.js: основний вхідний файл додатку, який запускає сервер;

– package.json: файл, що містить метаінформацію про проект і список залежностей;

– package-lock.json: файл, що фіксує версії встановлених залежностей;

– .gitignore: файл, що вказує, які файли або папки слід ігнорувати Git.

Висновки до розділу 2

В процесі дослідження та вибору технологічних рішень для розробки вебзастосунку були детально розглянуті та проаналізовані різні популярні платформи та технології. Обрані технології – Express, React, Sequelize та PostgreSQL – виявились найбільш відповідними для реалізації поставлених завдань завдяки їх потужним можливостям, високій продуктивності та широкій підтримці спільноти розробників. Архітектура вебзастосунку побудована за принципом розділення клієнтської та серверної частин, що забезпечує гнучкість та масштабованість системи. Серверна частина, реалізована за допомогою Express, відповідає за обробку запитів, управління бізнес-логікою та взаємодію з базою даних PostgreSQL через ORM Sequelize. Це дозволяє значно спростити процес управління даними та забезпечити високу продуктивність системи. Клієнтська частина, розроблена на базі React, забезпечує динамічний та інтерактивний інтерфейс користувача. Використання React дозволяє ефективно працювати з компонентами та легко управляти станом додатку, що значно покращує зручність роботи з ним.

Структура проекту розроблена таким чином, щоб забезпечити чітке розділення відповідальностей між різними компонентами системи. Серверна частина організована в папки, які містять код для роботи з API, конфігураціями, моделями даних, бізнес-логікою та утилітами. Це допомагає підтримувати порядок у проекті та спрощує його подальший розвиток. Клієнтська частина проекту також має чітку структуру з розділенням на компоненти, сторінки, сервіси та інтерфейси. Це сприяє легкій підтримці та розширенню функціональності додатку [25].

Використання технологій:

– Express: обрано за його простоту, гнучкість та можливість швидкої розробки серверної частини;

– React: вибір на користь цієї бібліотеки обумовлений її ефективністю в створенні динамічних інтерфейсів та великою підтримкою з боку спільноти;

– Sequelize: обраний як ORM для взаємодії з базою даних PostgreSQL, що дозволяє легко керувати схемами бази даних та здійснювати складні запити.

– PostgreSQL: потужна реляційна база даних, яка забезпечує високу надійність та продуктивність для зберігання та обробки даних.

Загалом, обрані технології та архітектурні рішення дозволяють створити ефективний, масштабований та зручний у використанні вебзастосунок, що відповідає сучасним вимогам та стандартам розробки.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Реалізація авторизації

На рисунку 3.1 показана сторінка входу до соціальної мережі, де користувачі можуть авторизуватися, ввівши свій електронний адрес та пароль.

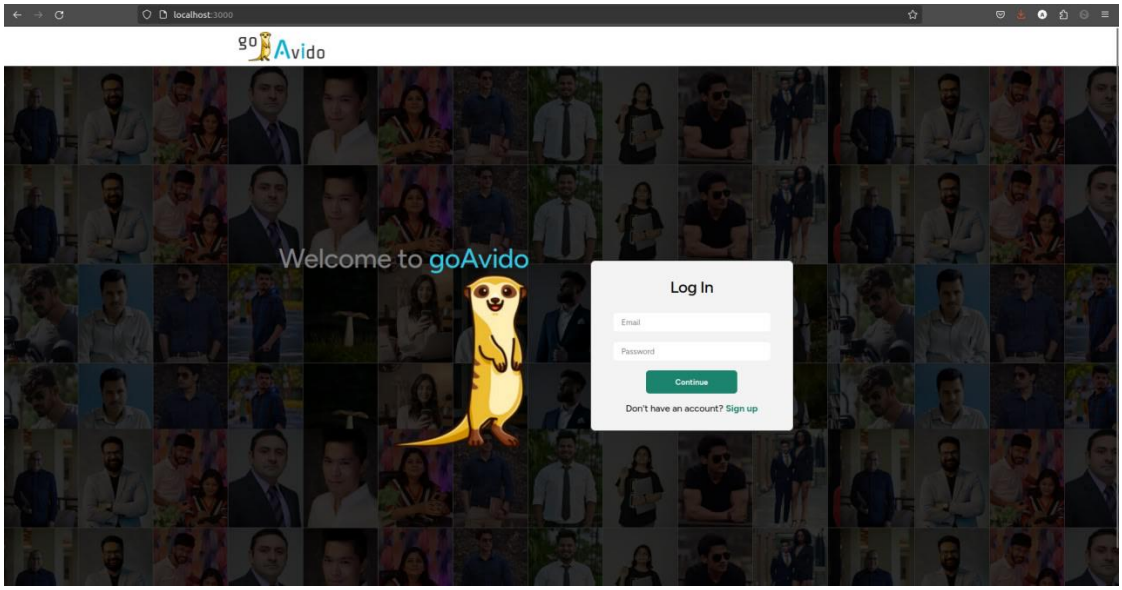

Рисунок 3.1 – Сторінка авторизації

Інтерфейс користувача.

Сторінка входу має сучасний та привабливий дизайн, що включає:

– ласкаво просимо: вітальний текст "Welcome to goAvido" для створення привітної атмосфери;

– форма авторизації: поля для введення електронної пошти та пароля, а також кнопка для підтвердження входу (рис. 3.2);

– навігація: посилання для переходу до сторінки реєстрації для нових користувачів.

Валідація даних на клієнтській стороні.

Перед надсиланням даних на сервер, відбувається їх перевірка на клієнтській стороні.

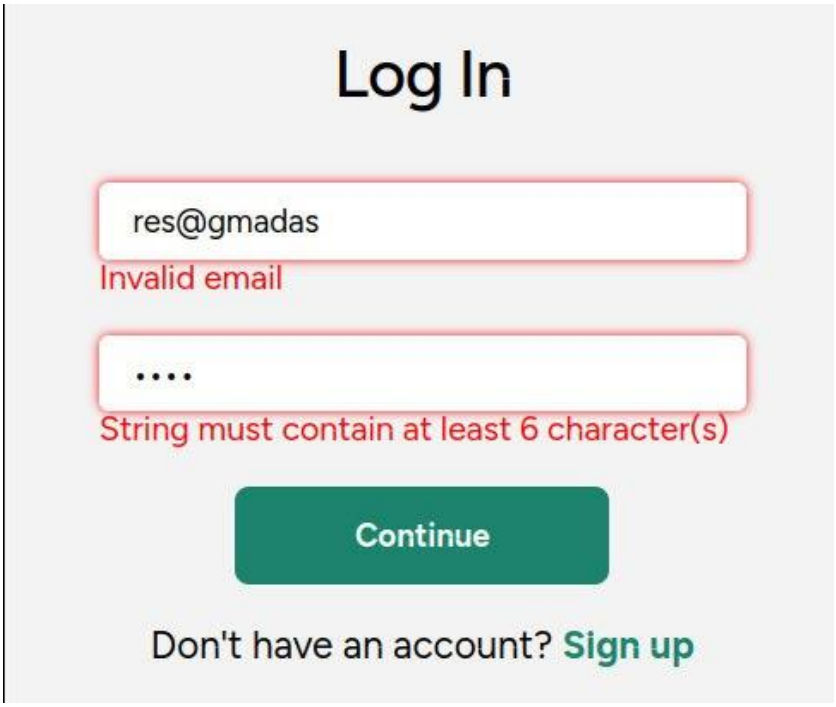

Рисунок 3.2 – Валідація форми

```
export const LoginSchema = zod.object({
  email: zod.string().email(),
  password: zod.string().min(6).max(48),
});
```

Надсилання даних на сервер.

Після успішної валідації дані надсилаються на сервер:

– API-запит. Використання HTTP-запиту (POST) для передачі даних форми до сервера;

– зашифровані дані. Пароль перед відправкою може бути зашифрований для додаткової безпеки.

Обробка запиту на сервері

На сервері відбуваються такі етапи:

– розбір запиту: сервер приймає запит та витягує з нього дані (електронна адреса та пароль);

– перевірка користувача: за електронною адресою шукається користувач у базі даних;

– перевірка пароля: витягнутий з бази даних пароль порівнюється з тим, що надіслав користувач (з використанням bcrypt або іншого алгоритму хешування).

За допомогою Node.js можна розробити вебсервер, обробляти HTTP-запити від клієнта та взаємодіяти з операційною системою на серверному пристрої. Для запуску самого сервера використовується менеджер пакетів npm та інструмент розробника nodemon (рис. 3.3). Nodemon є інструментом, який полегшує розробку додатків на основі Node.js шляхом автоматичного перезапуску програми, коли виявляються зміни у файлах каталогу. Налаштування запуску сервера здійснюється у конфігураційних файлах.

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
},
```

Рисунок 3.3 – Налаштування запуску сервера

Використовуючи бібліотеку Express, можна створювати обробники для запитів з різними HTTP методами для різних маршрутів URL. Також можна налаштовувати порти для підключення до сервера та бази даних. За допомогою функції use можна додавати функції до конвеєра проміжної обробки.

```
const app = express();
app.use(cors(corsOption));
```

```
app.options('*', cors(corsOption));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));
app.use(`${this._commonPrefix}/`, this.usersRouter.init());
app.use(`${this._commonPrefix}/`, this.postsRouter.init());
app.use(`${this._commonPrefix}/`, this.commentsRouter.init());
app.use(`${this._commonPrefix}/`, this.likesRouter.init());
```

Одним із важливих аспектів є кінцеві точки, за якими здійснюються запити на сервер з клієнта. За маршрутизацію відповідає маршрутизатор.

```
init = () => {
    return new Router().post(
      '/registration',
      asyncHandlerWrapperUtil(this.usersController.createUser),
    ).post(
      '/login',
      asyncHandlerWrapperUtil(this.usersController.userLogin),
```

Для перевірки даних необхідна проміжна функція, або, іншими словами, middleware. Для перевірки токена доступу та його нової генерації слід використовувати цю проміжну функцію. За допомогою JWT-токена можна безпечно передавати дані з сервера на клієнт. Для створення токена необхідно визначити заголовок (header) із загальною інформацією про токен, корисні дані (payload), такі як ID користувача, його роль, та підпис (signature).

```
 const token =
       req.headers['authorization'] &&
       req.headers['authorization'].split(' ');
     if (!token) {
       return res.sendStatus(401);
     }
     jwt.verify(token, jwtSecret, (error, payload) => {
       if (error) {
```

```
        return res.sendStatus(401);
      }
      usersUseCase.getUser({userId: payload.userId}).then(
        (user) => {
          if (!user) {
            return res.sendStatus(401);
          }
          req.user = user;
          next();
        },
        (error) => {
          console.error(error);
          return res.sendStatus(401);
        }
```

Перевірка і пошук користувача в базі даних

```
    userLogin = async ({email, password}) => {
        const user = await UserModel.findOne({where: {email,
password}});
        if (!user) {
          return null;
        }
        return await UserRepository.toDomain(user);
      };
```

Відповідь сервера.

Сервер надсилає відповідь на клієнт:

– успішна авторизація: у випадку успіху, відповідь включає JWT-токен (рис. 3.4);

– помилка авторизації: у разі невдачі (наприклад, неправильний пароль), сервер повертає відповідне повідомлення про помилку (рис 3.5).

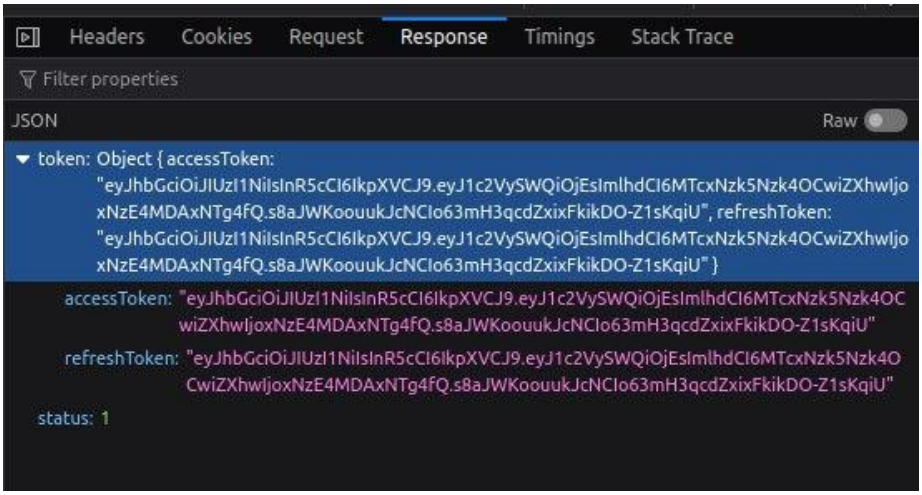

Рисунок 3.4 – Успішна авторизація (JWT-токен)

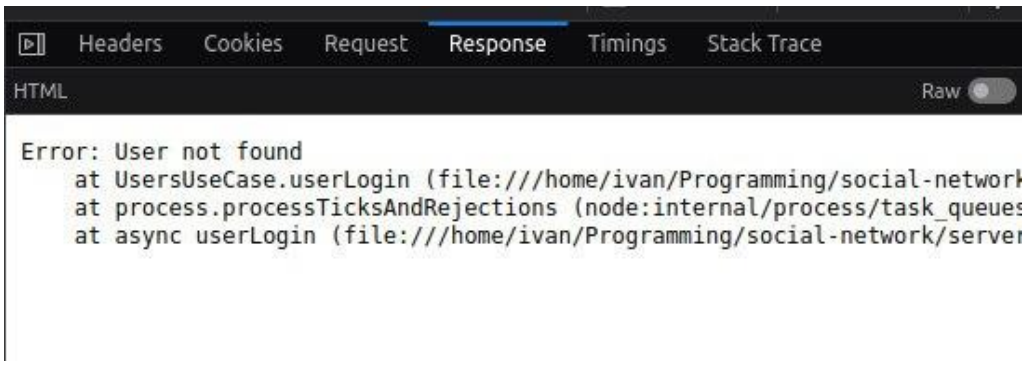

Рисунок 3.5 – Помилка авторизації (Такого користувача не існує)

На рисунку 3.6 отримаємо JWT-токен при успішній авторизації, і зберігаємо його в localstorage.

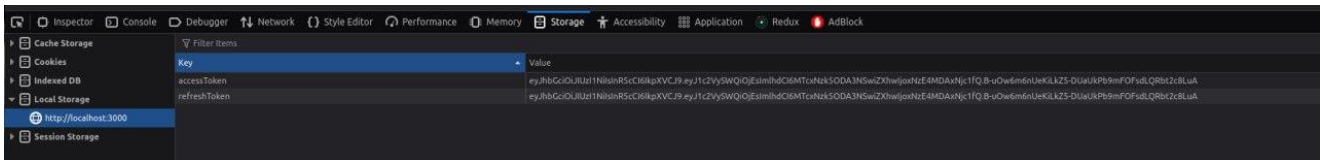

Рисунок 3.6 – Local storage

```
export default async function setAccessToken(data: Token) {
  if (data.token.accessToken && data.token.refreshToken) {
    const accessToken = data.token.accessToken;
```

```
        const refreshToken = data.token.refreshToken;
        localStorage.setItem('accessToken', accessToken);
        localStorage.setItem('refreshToken', refreshToken);
        console.log('Token save in localstorage');
      }
    }
```

Відображення захищених ресурсів.

Після успішної авторизації користувач отримує доступ до захищених ресурсів:

– навігація до головної сторінки: Після авторизації користувач перенаправляється до головної сторінки або панелі керування;

– захищені компоненти – компоненти, доступні лише авторизованим користувачам, рендеряться на основі присутності та валідності токена.

Реалізація авторизації, основні моменти:

– компонент форми авторизації: розміщений у папці components або pages, цей компонент відповідає за відображення форми та обробку подій введення користувача;

– API-сервіси: у папці services знаходяться функції, що здійснюють API-запити для авторизації;

– схеми валідації: папка validationSchema містить схеми для валідації даних форми перед їх надсиланням;

– маршрутизація: використання React Router для управління переходами між сторінками авторизації та захищеними маршрутами.

Схема валідації.

```
export const RegistrationSchema = zod.object({
  first_name: zod.string().min(3).max(25),
  last_name: zod.string().min(3).max(25),
  email: zod.string().email(),
  password: zod.string().min(6).max(48),
});
export const LoginSchema = zod.object({
```

```
  email: zod.string().email(),
  password: zod.string().min(6).max(48),
});
```

3.2 Представлення та презентації головної сторінки

На рисунку 3.7 зображена головна сторінка вебзастосунку.

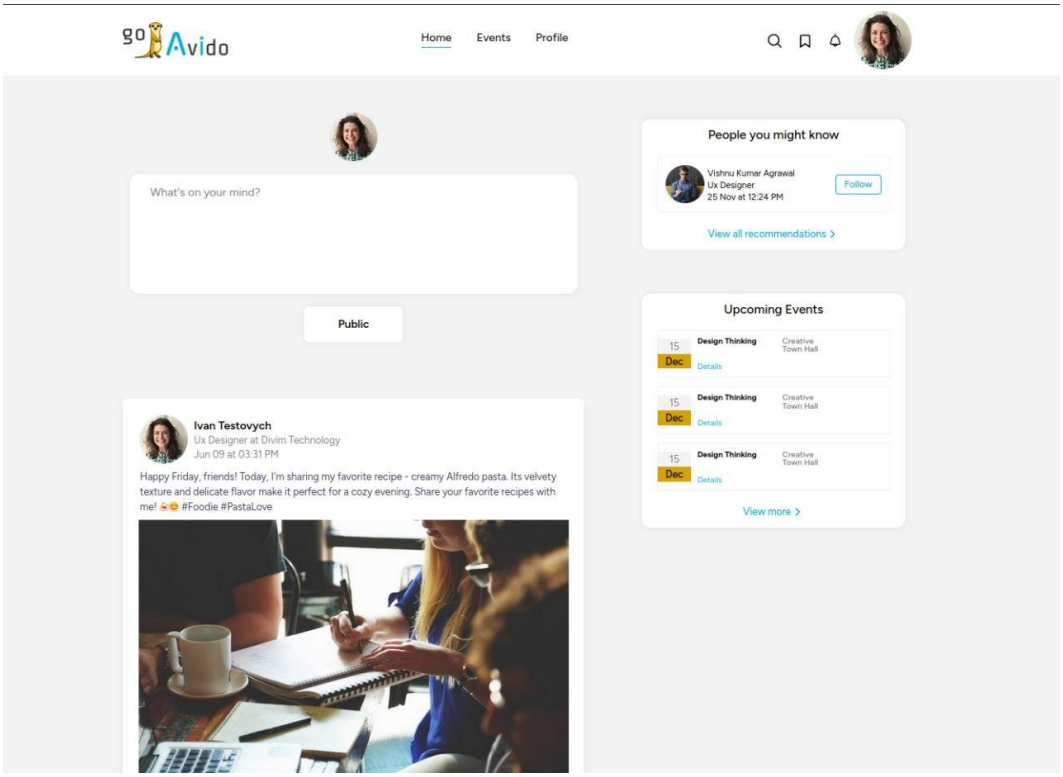

Рисунок 3.7 – Головна сторінка (Home)

Верхній навігаційний бар:

– логотип та назва: логотип мережі у вигляді суриката та назва "goAvido";

– навігаційне меню: "Home", "Events", "Profile".;

– активна вкладка "Home" підкреслена.

Основна область контенту:

– аватар користувача: зображення профілю користувача;

– поле вводу: поле для введення тексту публікації з підказкою "What's on your

mind?";

– кнопка "Public": Кнопка для публікації запису.

Публікації користувачів:

– профіль автора: ім'я автора публікації, посада та компанія;

– дата публікації: дата та час розміщення публікації;

– текст публікації: основний текст публікації;

– зображення: прикріплене зображення до публікації.

Бічна панель:

– рекомендації: рекомендований користувач із ім'ям, посадою, компанією та кнопкою "Follow" для підписки;

– посилання: лінк "View all recommendations" для перегляду всіх рекомендацій;

– список подій: перелік найближчих подій із датами, назвами подій та місцем проведення;

– деталі події: кнопки "Details" для перегляду додаткової інформації про події;

– посилання: лінк "View more" для перегляду більше подій.

Загальний дизайн:

– фонове зображення: фон, що складається з фотографій користувачів мережі, надає відчуття активності та спільноти;

– чистий інтерфейс: простий та зручний для користування інтерфейс з акцентом на контенті користувачів та подіях;

– кольори та шрифти: використання приємних для ока кольорів та читабельних шрифтів для забезпечення хорошого користувацького досвіду.

Процес створення постів у соціальній мережі включає кілька етапів, починаючи з введення тексту користувачем і закінчуючи відображенням нового посту на головній сторінці. Ось детальний опис цього процес:

– на головній сторінці користувач бачить поле для введення тексту публікації. Він може ввести текст та, за потреби, додати зображення чи інший мультимедійний контент. Після введення тексту користувач натискає кнопку "Public" (або іншу кнопку для підтвердження);

– перед надсиланням даних на сервер, вони проходять валідацію на клієнтській стороні;

– перевірка, що текст посту не є порожнім.

На рисунку 3.8 зображення створення поста.

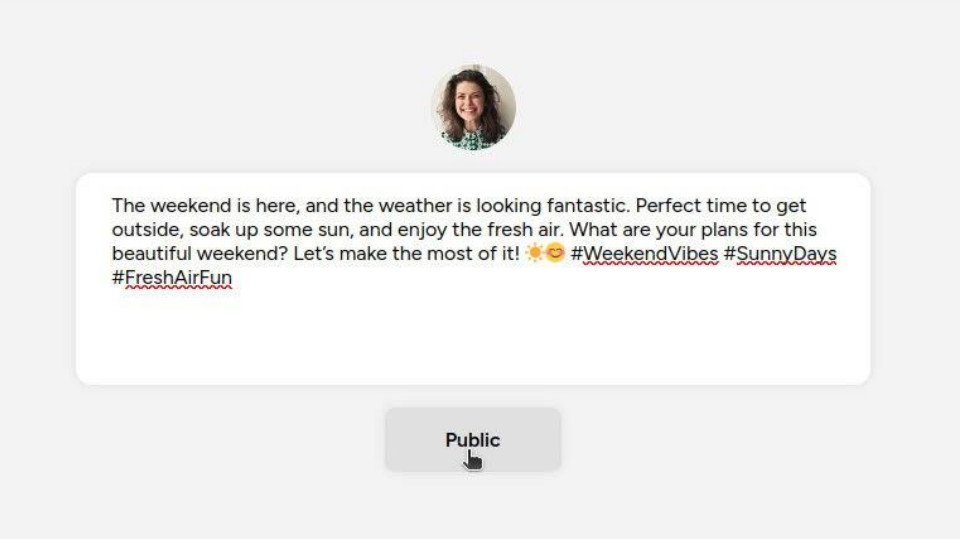

Рисунок 3.8 – Створення посту

Надсилання даних на сервер

Якщо дані пройшли валідацію, вони надсилаються на сервер за допомогою API-запиту:

– метод: зазвичай використовується HTTP POST запит до відповідного кінцевого пункту;

– тіло запиту: містить текст посту та інші необхідні дані.

Сервер отримує запит і виконує такі дії:

– аутентифікація: перевірка, чи включає запит дійсний JWT-токен для

підтвердження автентичності користувача;

– валідація даних: перевірка правильності отриманих даних (чи не перевищує текст посту максимальну довжину);

– збереження у базі даних: якщо дані валідні, сервер зберігає їх у базі даних (рис. 3.9).

```
createPost = async ({text, userId}) => {
    try {
      const post = await PostModel.create({text, user_id: userId});
      return await PostRepository.toDomain(post);
    } catch (error) {
      console.error('Error creating post', error);
    }
  };
```

id	text	createdAt	updatedAt	user_id
1	What's on your mind? What's on your mind? What's…	2024-05-30 08:15:45.920 +00:00	2024-05-30 08:15:45.920 +00:00	1
2	test test test test test test test test test t…	2024-05-30 09:23:40.710 +00:00	2024-05-30 09:23:40.710 +00:00	3
3	Ux Designer Ux Designer Ux Designer Ux Designer Ux …	2024-05-30 12:07:41.911 +00:00	2024-05-30 12:07:41.911 +00:00	2
4	6 June 6 June 6 June 6 June 6 June 6 June 6 June 6 …	2024-06-06 13:02:04.392 +00:00	2024-06-06 13:02:04.392 +00:00	2
5	7 June 7 June 7 June 7 June 7 June 7 June 7 June 7 …	2024-06-07 17:00:55.238 +00:00	2024-06-07 17:00:55.238 +00:00	1
6	8 june 8 june 8 june 8 june 8 june 8 june 8 june 8 …	2024-06-08 09:14:16.602 +00:00	2024-06-08 09:14:16.602 +00:00	1
7	Happy Friday, friends! Today, I'm sharing my favori…	2024-06-09 12:31:38.398 +00:00	2024-06-09 12:31:38.398 +00:00	1
8	The weekend is here, and the weather is looking fan…	2024-06-10 06:26:52.503 +00:00	2024-06-10 06:26:52.503 +00:00	1
9	The weekend is here, and the weather is looking fan…	2024-06-10 06:30:39.137 +00:00	2024-06-10 06:30:39.137 +00:00	1

Рисунок 3.9 – Збереження постів в базі даних

Після успішного збереження посту сервер відправляє відповідь клієнту:

– успіх: відповідь містить інформацію про створений пост, включаючи його унікальний ідентифікатор, час створення та інші метадані;

– помилка: у випадку помилки сервер повертає відповідний код статусу та повідомлення про помилку.

Оновлення інтерфейсу користувача.

Клієнт отримує відповідь від сервера і оновлює інтерфейс без перезавантаження сторінки:

– додавання нового посту: новий пост додається до списку постів у відповідному компоненті React.

– оновлення стану: використання стану (state) для повторного рендерингу компоненту з новими даними.

На рисунку 3.10 зображено створений пост користувача.

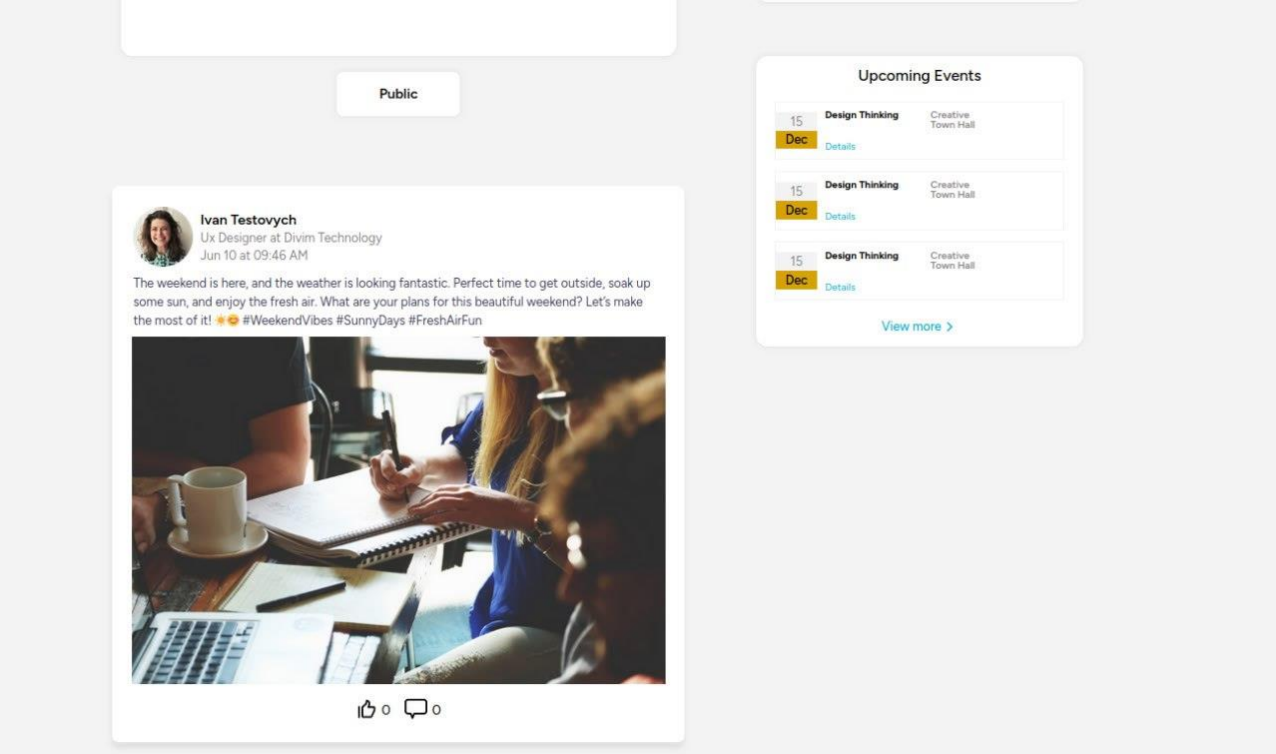

Рисунок 3.10 – Створений пост

Структура постів у соціальній мережі.

Кожен пост у соціальній мережі має комплексну структуру, що включає інформацію про користувача, який створив пост, кількість лайків, коментарі та інші метадані.

Отримання всіх постів.

```
getAllPosts = async () => {
    try {
```

```
      const posts = await PostModel.findAll({include: ['user', 'likes']});
      return await Promise.all(posts.map(PostRepository.toDomain));
    } catch (error) {
      console.error('Error getting all posts', error);
    }
```

Кожен пост прив'язаний до користувача, який його створив. Ця інформація включає:

– ідентифікатор користувача (user_id): унікальний ідентифікатор користувача в базі даних;

– ім'я користувача: ім'я або псевдонім користувача;

– аватар користувача: URL зображення профілю користувача.

Лайки відображають кількість користувачів, які позитивно оцінили пост. Інформація про лайки включає:

– кількість лайків: загальна кількість лайків посту (рис. 3.11);

– ідентифікатори користувачів, що лайкнули (liked_by): Масив ідентифікаторів користувачів, які лайкнули пост. Це може бути використано для перевірки, чи лайкнув поточний користувач пост.

На рисунку 3.11 зображення кнопка лайка, при натисканні відправляється запит на бекенд, передається user_id(хто натиснув на лайк) і post_id(на який пост саме був поставлений лайк).

Рисунок 3.11 – Кнопка лайку

На рисунку 3.12 зображена таблиця лайків в базі даних.

WHERE ORDER BY

	id	user_id	post_id	createdAt	updatedAt
1	1	1	3	2024-06-09 08:40:18.127000 +00:00	2024-06-09 08:40:18.128000 +00:00
2	2	1	1	2024-06-09 08:41:54.309000 +00:00	2024-06-09 08:41:54.309000 +00:00
3	3	1	4	2024-06-09 08:41:56.575000 +00:00	2024-06-09 08:41:56.575000 +00:00
4	4	1	6	2024-06-09 08:41:58.189000 +00:00	2024-06-09 08:41:58.189000 +00:00
5	5	1	5	2024-06-09 08:41:59.906000 +00:00	2024-06-09 08:41:59.906000 +00:00
6	6	1	2	2024-06-09 08:42:06.213000 +00:00	2024-06-09 08:42:06.213000 +00:00
7	39	1	7	2024-06-09 12:31:44.177000 +00:00	2024-06-09 12:31:44.177000 +00:00
8	41	1	8	2024-06-10 06:26:57.236000 +00:00	2024-06-10 06:26:57.236000 +00:00
9	44	1	9	2024-06-10 06:30:55.900000 +00:00	2024-06-10 06:30:55.900000 +00:00

Рисунок 3.12 – Таблиця лайків

Коментарі дозволяють користувачам взаємодіяти з постом (рис 3.14). Інформація про коментарі включає:

– ідентифікатор коментаря (id): унікальний ідентифікатор коментаря;

– текст коментаря(content): текст, що був залишений користувачем;

– користувач, що залишив коментар (user_id): ідентифікатор користувача, що залишив коментар;

– час створення коментаря (created_at): час, коли коментар був створений.

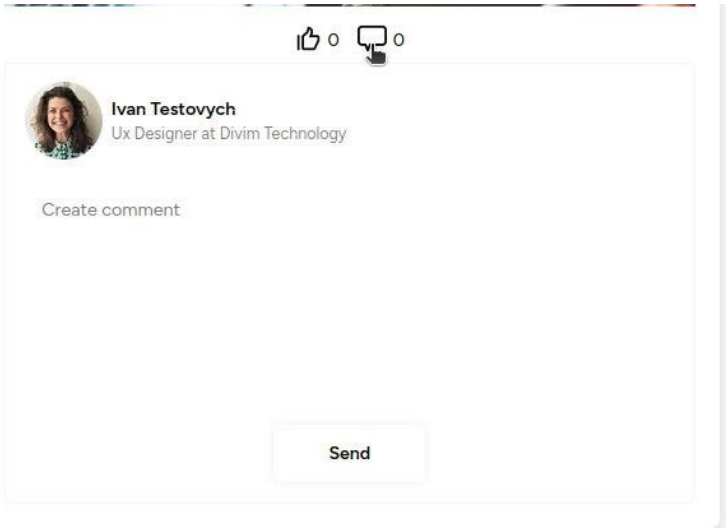

Рисунок 3.13 – Створення коментаря

При натисканні на кнопку коментарів з’являється інпут, в який можна ввести текст, і створити коментарій. Запит відправляється на бекенд, і коментар створюється в окремій таблиці (рис 3.14).

```
createComment = async ({content, user_id, post_id}) => {
    try {
      const comment = await CommentModel.create({content, user_id, post_id});
      return await CommentRepository.toDomain(comment);
    } catch (error) {
      console.error('Error creating comment', error);
    }
  };
```

WHERE ORDER BY

	id	content	user_id	post_id	createdAt	update
1	1	Can't wait to spend the weekend outdoors! Planning a p…	1	1	2024-06-10 08:51:26.553000 +00:00	2024-06-
2	2	"Looks like the perfect weather for a picnic in the pa…	2	10	2024-06-10 08:52:54.892000 +00:00	2024-06-

Рисунок 3.14 – таблиця коментарів

На рисунку 3.15 зображено рекомендації профілі користувачів.

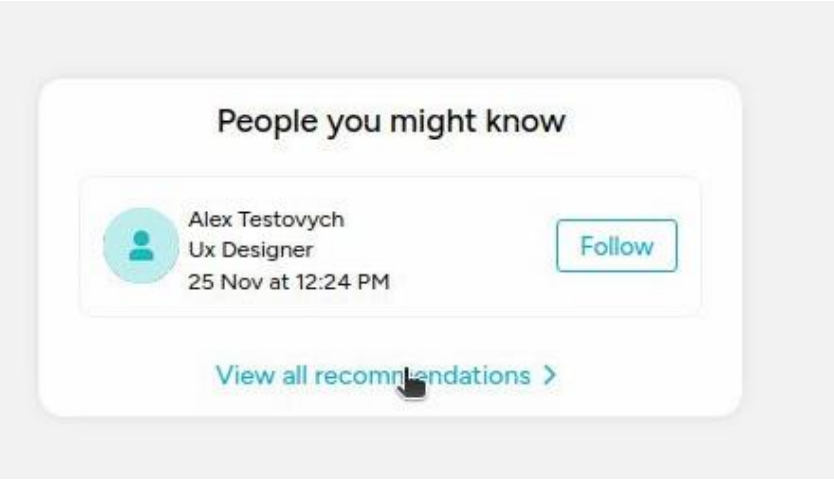

Рисунок 3.15 – Компонент рекомендацій користувачів

3.3 Опис сторінки рекомендацій і підписок

На сторінці рекомендацій можна бачити список користувачів (рис. 3.16), при натисканні на кнопку «follow», запит відправляється на бек енд.

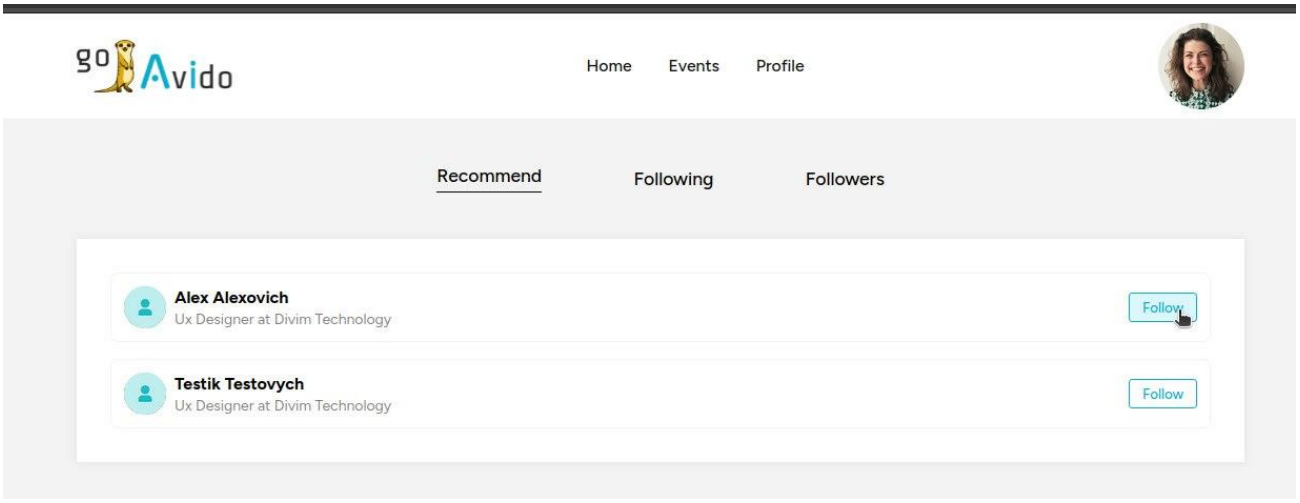

Рисунок 3.16 – Список рекомендованих користувачів

На сторінці підписок відправляється запит на бекенд, і отримуємо об’єкт підписок користувача (рис. 3.17), а далі бачимо цей список підписок (рис. 3.18).

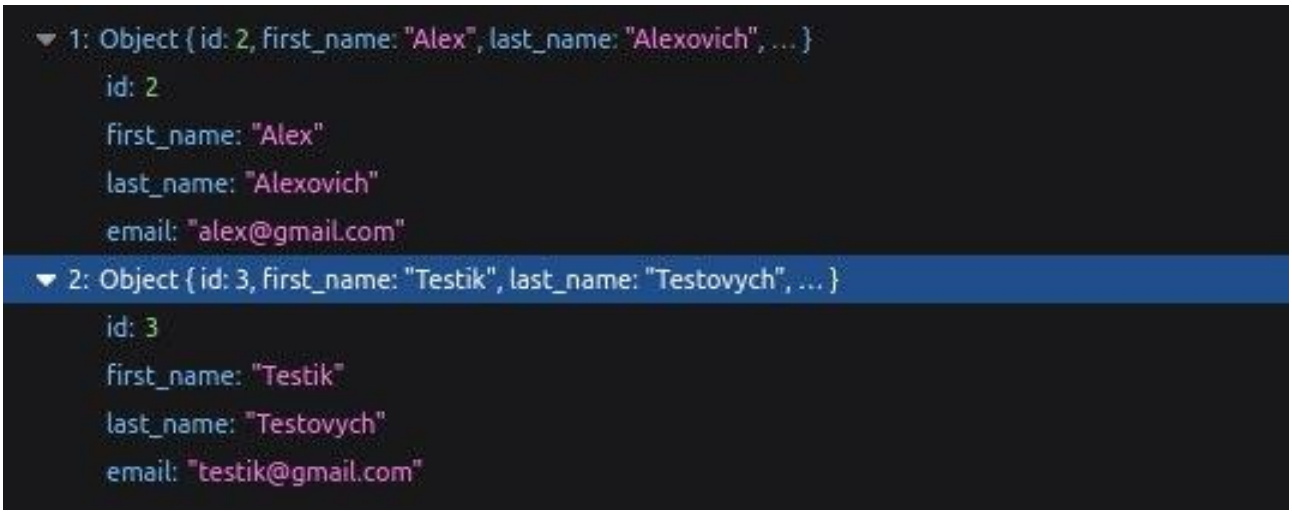

Рисунок 3.17 – Отриманий список підписок

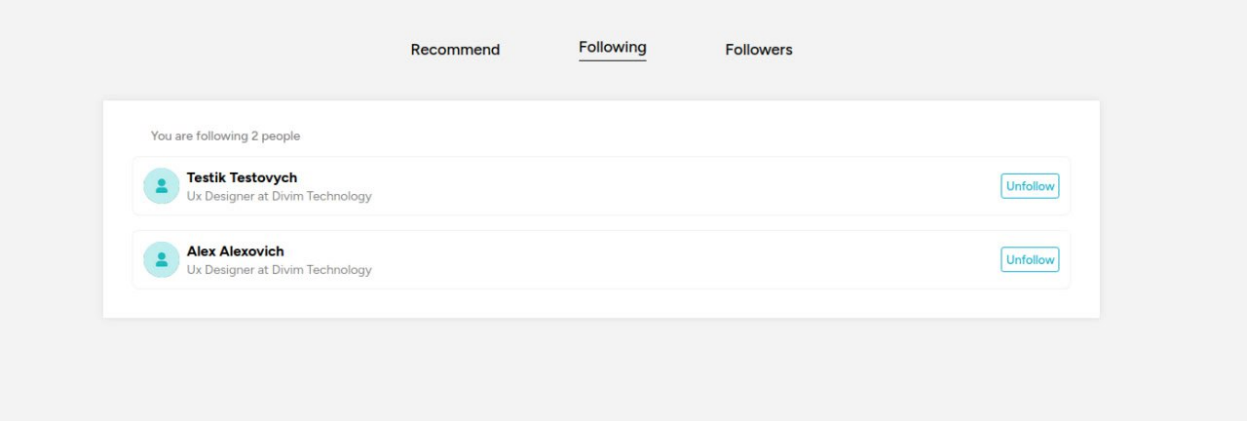

Рисунок 3.18 – Список підписок

```
createFollower = async ({userId, followerId}) => {
    try {
      await   FollowerModel.create({follow_from:   userId,   follow_to:
followerId});
    } catch (error) {
      console.error('error', error);
    }
  };
  unFollowById = async ({userId, followerId}) => {
    try {
      const result = await FollowerModel.destroy({
        where: {follow_from: userId, follow_to: followerId},
      });
      if (!result) {
        console.error('Error deleting user', error);
      }
      return true;
    } catch (error) {
      console.error('Error', error.message);
      return false;
    }
```

```
getFollowingById = async ({userId}) => {
    console.log('userID reposss', userId);
    try {
      const user = await UserModel.findOne({where: {id: userId},
include: 'following'});
      return Promise.all(user.following.map(following =>
UserRepository.toDomain(following)));
    } catch (error) {
      console.error('Error getting following', error);
    }
  };
  getFollowersById = async ({userId}) => {
    try {
      console.log('userID repo', userId);
      const user = await UserModel.findOne({where: {id: userId},
include: 'followers'});
      return Promise.all(user.followers.map(followers =>
UserRepository.toDomain(followers)));
    } catch (error) {
      console.error('Error:', error.message);
    }
```

3.4 Інтерфейс сторінки користувача

При наведенні на ім’я користувача в пості, ми можемо перейти на його профіль (рис. 3.19), і побачити інформацію про нього.

Рисунок 3.19 – Перехід на сторінку користувача

Отримаємо об’єкт користувача з бекенду (рис 3.20), ID з об’єкту підставляється в динамічний URL (рис 3.21) і переходимо на сторінку користувача.

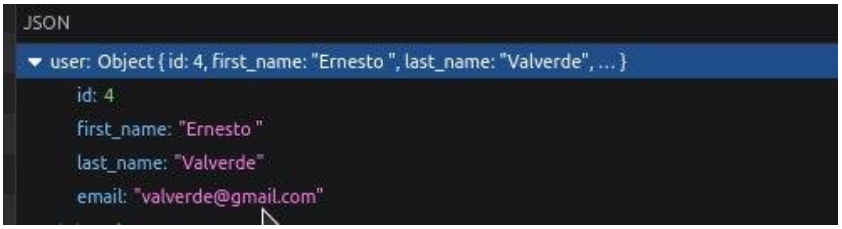

Рисунок 3.20 – Об’єкт користувача

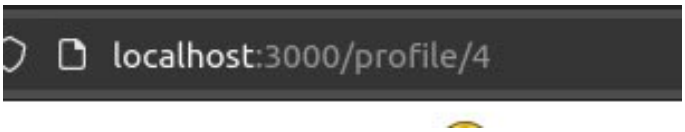

Рисунок 3.21 – Динамічний URL профілю користувача

```
getUserById = async ({userId}) => {
    try {
      const user = await UserModel.findByPk(userId);
      if (!user) {
        return null;
      }
      return await UserRepository.toDomain(user);
    } catch (error) {
      console.error('error user repository', error);
    }
  };
```

Бачимо можливість підписатись чи відправити лист на пошту, також можна побачити пости користувача (рис 3.22), його підписки та його підписників (рис 3.23-3.24).

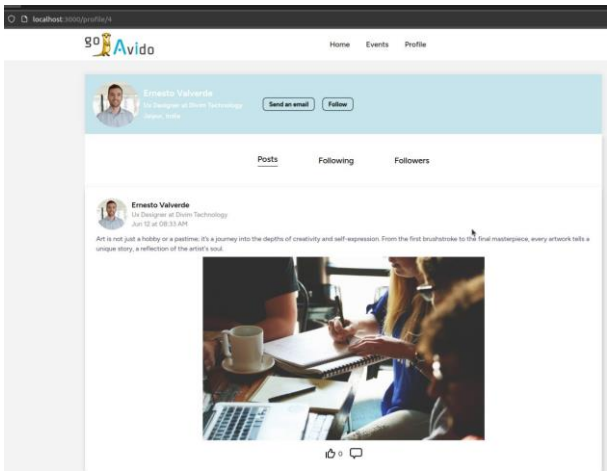

Рисунок 3.22 – Пости користувача

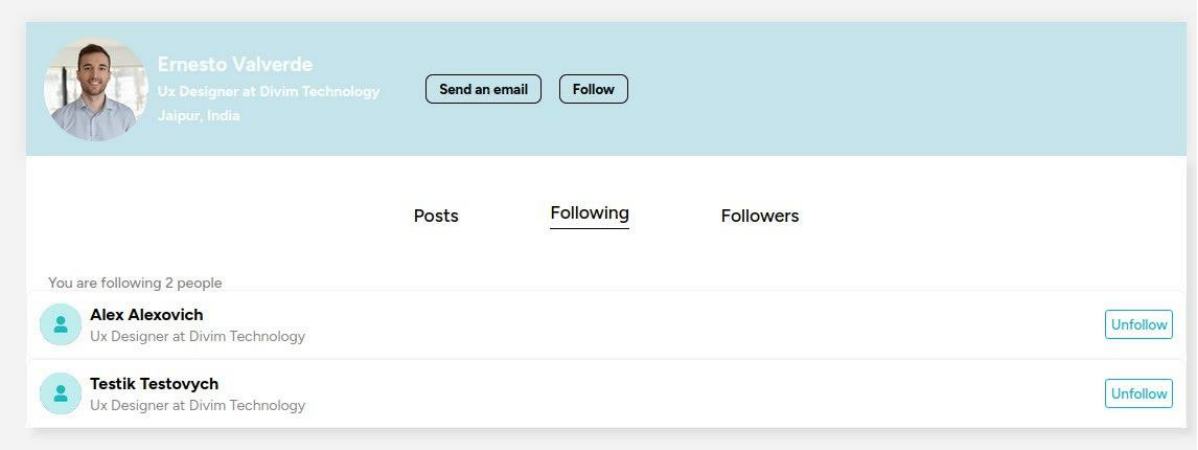

Рисунок 3.23 – Підписки користувача

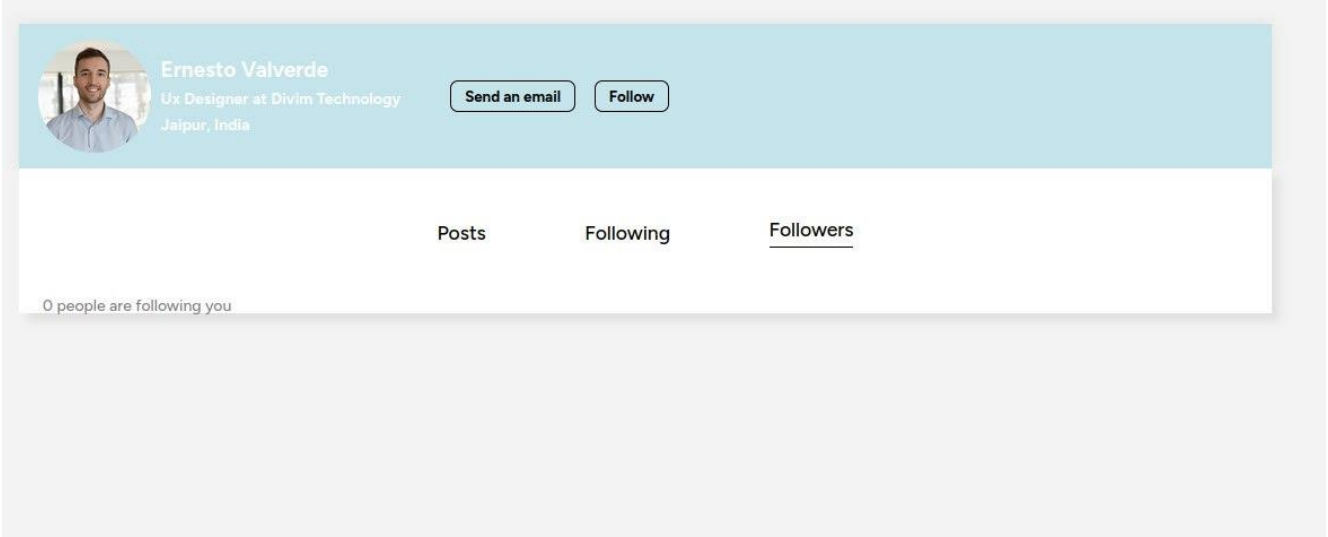

Рисунок 3.24 – Підписники користувача

```
getFollowingById = async ({userId}) => {
    try {
      const user = await UserModel.findOne({where: {id: userId},
include: 'following'});
      return Promise.all(user.following.map(following =>
UserRepository.toDomain(following)));
    } catch (error) {
      console.error('Error getting following', error);
    }
  };
  getFollowersById = async ({userId}) => {
    try {
```

```
      const user = await UserModel.findOne({where: {id: userId},
include: 'followers'});
      return Promise.all(user.followers.map(followers =>
UserRepository.toDomain(followers)));
    } catch (error) {
      console.error('Error:', error.message);
    }
  };
```

Висновки до розділу 3

Під час виконання розділу третього було розглянуто процес написання програмного забезпечення як на серверній частині, так і на клієнтській. Особлива увага приділялася технологіям розробки та мовам програмування, які використовувалися для створення застосунку. Було детально досліджено і реалізовано функції, сервіси, контролери, компоненти, проміжні функції та конфігурації. Розглянуто методи HTTP-запитів до сервера та компоненти, відповідальні за маршрутизацію в програмному забезпеченні.

Результати розробки програмного забезпечення були успішно продемонстровані. Кожна функція, відповідальна за виконання конкретних дій, була ретельно протестована після розробки. Були представлені такі можливості, як реєстрація та авторизація на сайті, створення постів, лайки, коментарі, підписка і відписка, перегляд підписників та профілів користувачів.

Клієнтська частина застосунку, реалізована за допомогою React, показала високу ефективність у створенні динамічних та інтерактивних інтерфейсів користувача. Компонентна архітектура React дозволила легко організувати код і забезпечити зручність у його підтримці та розширенні.

Серверна частина, побудована на Express, забезпечила надійну обробку запитів та інтеграцію з базою даних PostgreSQL через Sequelize. Це дало можливість створити стабільний і продуктивний бекенд для застосунку.

Крім того, було розроблено та протестовано всі необхідні маршрути для забезпечення належної функціональності вебзастосунку. Було продемонстровано виконання різних дій, таких як створення та управління контентом (пости, коментарі, лайки), а також управління користувачами (реєстрація, авторизація, підписка та відписка)

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було досягнуто мету створення вебзастосунку "Соціальна мережа", спрямованого на поліпшення взаємодії між користувачами за рахунок впровадження ефективної системи управління контентом та забезпечення високої функціональності.

Для досягнення цієї мети було вирішено такі завдання:

– розглянуто аналогічні застосунки та визначено основні функції, які повинні бути реалізовані у новій соціальній мережі;

– продемонстровано архітектуру застосунку та патерн проектування.

– досліджено бібліотеки та інструменти, що використовуються при розробці програмного забезпечення;

– спроектовано план вебзастосунку.

Були прийняті такі проектні рішення:

– створення зручного та інтуїтивно зрозумілого інтерфейсу;

– реалізація функцій для реєстрації та авторизації користувачів;

– можливість створення та управління постами, лайками, коментарями;

– реалізація функціоналу підписки та відписки на користувачів.

Результатом розробки вебзастосунку соціальної мережі є створення платформи, яка дозволяє користувачам взаємодіяти один з одним, обмінюватися контентом та розширювати свої соціальні зв'язки.

Отже, виконана робота дозволила не лише створити функціональний та ефективний вебзастосунок соціальної мережі, але й закласти основу для його подальшого розвитку та масштабування. Обрані технологічні рішення та підходи забезпечили високу продуктивність, надійність та зручність використання системи, що відповідає сучасним вимогам та очікуванням користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Вікіпедія: Соціальна мережа URL : https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%86%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0 (дата звернення: 04.05.2024)

2. Історія розвитку соціальних мереж: основні періоди URL: http://visnyk.ukrbook.net/article/view/287056 (дата звернення: 04.05.2024)

3. Соціальні мережі, як середовище для технологій маніпулятивного впливу URL: moz-extension://9199e496-1f04-46dc-83c4-6caae2845917/content/web/viewer.html?file=http%3A%2F%2Fwww.irbis-nbuv.gov.ua%2Fcgi-bin%2Firbis_nbuv%2Fcgiirbis_64.exe%3FC21COM%3D2%26I21DBN%3DUJRN%26P21DBN%3DUJRN%26IMAGE_FILE_DOWNLOAD%3D1%26Image_file_name%3DPDF%2Fszi_2016_1_8.pdf (дата звернення 09.05.2024)

4. Twitter. URL: https://twitter.com/ (дата звернення: 10.05.2024)

5. Mastodon. URL: https://joinmastodon.org/ (дата звернення: 10.05.2024)

6. Gab. URL: https://en.wikipedia.org/wiki/Gab_%28social_network%29 (дата звернення: 10.05.2024)

7. Parler. URL: https://parler.com/ (дата звернення: 10.05.2024)

8. Minds. URL: https://www.minds.com/ (дата звернення: 10.05.2024)

9. Технологічні рішення твітера URL: https://www.browserstack.com/guide/full-stack-development (дата звернення: 15.05.2024)

10. React URL: https://legacy.reactjs.org/ (дата звернення: 17.05.2024)

11. GitHub: React URL: https://github.com/facebook/react/ (дата звернення: 17.05.2024).

12. Express URL: https://expressjs.com/ (дата звернення: 17.05.2024)

13. Node.js URl: https://nodejs.org/ (дата звернення: 17.05.2024)

14. Грассманн Д., Нолте С. Node.js: програмування серверних застосунків. Київ: Пітер, 2015. 512 с. (дата звернення: 17.05.2024)

15. Postgresql URL: https://www.postgresql.org/ (дата звернення: 19.05.2024)

16. Docker URL: https://www.docker.com/ (дата звернення: 19.05.2024)

17. Sequelize URL: https://sequelize.org/ (дата звернення: 19.05.2024)

18. MDN Web Docs: HTTP URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 20.05.2024).

19. Іващенко С. Технології розробки веб-додатків. – Київ: Либідь, 2019. – 300 с. (дата звернення:20.05.2024)

20. Трирівнева архітектура URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 20.05.2024)

21. Резнік В., Кулик О. Програмування для веб-інтерфейсів. – Харків: Основа, 2017. – 280 с. (дата звернення 21.05.2024)

22. Архітектура веб-додатків URL : https://medium.com/@IvanZmerzlyi/%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%B2%D0%B5%D0%B1-%D0%B4%D0%BE%D0%B4%D0%B0%D1%82%D0%BA%D1%96%D0%B2-ca4c82f75bcf (дата звернення: 23.05.2024)

23. Архітектура систем баз даних URL: https://stud.com.ua/50162/informatika/arhitektura_sistemi_danih (дата звернення: 25.05.2024)

24. Використання мікросервісної архітектури URL : https://it-rating.ua/vikoristannya-mikroservisnoi-arhitekturi-v-rozrobtsi-veb-dodatkiv (дата звернення: 27.05.2024)

25. Medium: Full Stack Development URL: https://medium.com/full-stack-

development (дата звернення: 29.05.2024).

ДОДАТОК А

Код авторизації

SignIn.tsx

```
import React from 'react';
import Input from '../shared/Input';
import {AuthorizationProps, Login} from '../../interfaces/interfaces';
import {postDataRequest} from '../../services/api';
import setAccessToken from '../../services/setAccessToken';
import {useForm} from 'react-hook-form';
import {zodResolver} from '@hookform/resolvers/zod';
import {LoginSchema} from '../../validationSchema/schema';
import Button from '../shared/Button';
import {useNavigate} from 'react-router-dom';
const SignIn: React.FC<AuthorizationProps> = ({toggleForm}) => {
  const navigate = useNavigate();
  const onSubmit = async (data: Login) => {
    try {
      const response = await postDataRequest(data, '/login');
      await setAccessToken(response);
      navigate('/home');
    } catch (error) {
      console.error('Error', error);
    }
  };
  const {
    register,
    handleSubmit,
    formState: {errors},
  } = useForm<Login>({
    resolver: zodResolver(LoginSchema),
    mode: 'onBlur',
    reValidateMode: 'onChange',
    shouldFocusError: true,
  });
  return (
```

```
    <div>
      <form
        action=""
        className="registration__form"
        onSubmit={handleSubmit(onSubmit)}
      >
        <h1 className="form__title">Log In</h1>
        <Input name="email" register={register} error={errors.email}
placeholder="Email" />
        <Input name="password" register={register} error={errors.password}
typeText="password"
               placeholder="Password" />
        <Button text="Continue" style="authorization__btn" padding={'12px 50px'}
type="submit" />
        <p className="p-1"> Don't have an account? <span
          style={{color: '#1C836D', cursor: 'pointer', fontWeight: 700}}
onClick={toggleForm}>Sign up</span>
        </p>
      </form>
    </div>
  )
};
export default SignIn;
```

SignUp.tsx

```
import React from 'react';
import Input from '../shared/Input';
import {AuthorizationProps, Registration} from '../../interfaces/interfaces';
import {postDataRequest} from '../../services/api';
import setAccessToken from '../../services/setAccessToken';
import {useForm} from 'react-hook-form';
import {zodResolver} from '@hookform/resolvers/zod';
import {RegistrationSchema} from '../../validationSchema/schema';
import Button from '../shared/Button';
import {useNavigate} from 'react-router-dom';

const SignUp: React.FC<AuthorizationProps> = ({toggleForm}) => {
  const navigate = useNavigate();
```

```
  const onSubmit = async (data: Registration) => {
    try {
      console.log('data', data);
      const response = await postDataRequest(
        data, '/registration',
      );
      console.log('response', response);
      await setAccessToken(response);
      navigate('/home');
      console.log(response);
    } catch (error) {
      console.error('Error', error);
    }
  };
  const {
    register,
    handleSubmit,
    formState: {errors},
  } = useForm<Registration>({
    resolver: zodResolver(RegistrationSchema),
    mode: 'onBlur',
    reValidateMode: 'onChange',
    shouldFocusError: true,
  });
  return (
    <div>
      <form
        action=""
        className="registration__form"
        onSubmit={handleSubmit(onSubmit)}
      >
        <h1 className="form__title">Sign Up</h1>
        <Input name="first_name" register={register} error={errors.first_name}
placeholder="First Name" />
        <Input name="last_name" register={register} error={errors.last_name}
placeholder="Last Name" />
        <Input name="email" register={register} error={errors.email}
placeholder="Email" />
```

```
        <Input name="password" error={errors.password} register={register}
typeText="password"
               placeholder="Password" />
        <Button text="Continue" style="authorization__btn" padding={'12px 50px'} />
        <p className="p-1"> Already have an account? <span
          style={{color: '#1C836D', cursor: 'pointer', fontWeight: 700}}
onClick={toggleForm}>Log in</span>
        </p>
      </form>
    </div>
  );
};

export default SignUp;
```

setAccessToken.ts

```
import {Token} from '../interfaces/interfaces';

export default async function setAccessToken(data: Token) {
  if (data.token.accessToken && data.token.refreshToken) {
    const accessToken = data.token.accessToken;
    const refreshToken = data.token.refreshToken;
    localStorage.setItem('accessToken', accessToken);
    localStorage.setItem('refreshToken', refreshToken);
    console.log('Token save in localstorage');
  }
```

user.repository.js

```
import {FollowerModel, UserModel} from '../index.js';
import {User} from '../../domain/user.js';

export class UserRepository {
  createUser = async ({first_name, last_name, email, password}) => {
    try {
      const user = await UserModel.create({first_name, last_name, email, password});
      return await UserRepository.toDomain(user);
    } catch (error) {
      console.error('Error creating user', error);
    }
```

```
};
userLogin = async ({email, password}) => {
  const user = await UserModel.findOne({where: {email, password}});
  if (!user) {
    return null;
  }
  return await UserRepository.toDomain(user);
};
getUser = async ({userId}) => {
  const user = await UserModel.findByPk(userId);
  if (!user) {
    return null;
  }
  return await UserRepository.toDomain(user);
};
getUserById = async ({userId}) => {
  try {
    const user = await UserModel.findByPk(userId);
    if (!user) {
      return null;
    }
    return await UserRepository.toDomain(user);
  } catch (error) {
    console.error('error user repository', error);
  }
};
userUpdate = async (user, {refreshToken}) => {
  const ID = user.id;
  await UserModel.update({refreshToken}, {where: {id: ID}});
  user.refreshToken = refreshToken;
  console.log(user.refreshToken);
  return await UserRepository.toDomain(user);
};
getAllUsers = async () => {
  try {
    const users = await UserModel.findAll();
    return await Promise.all(users.map(UserRepository.toDomain));
  } catch (error) {
```

```
      console.error('Error getting all users', error);
      return [];
    }
  };
  createFollower = async ({userId, followerId}) => {
    try {
      await FollowerModel.create({follow_from: userId, follow_to: followerId});
    } catch (error) {
      console.error('error', error);
    }
  };
  getFollowingById = async ({userId}) => {
    try {
      const user = await UserModel.findOne({where: {id: userId}, include:
'following'});
      return Promise.all(user.following.map(following =>
UserRepository.toDomain(following)));
    } catch (error) {
      console.error('Error getting following', error);
    }
  };
  getFollowersById = async ({userId}) => {
    try {
      const user = await UserModel.findOne({where: {id: userId}, include:
'followers'});
      return Promise.all(user.followers.map(followers =>
UserRepository.toDomain(followers)));
    } catch (error) {
      console.error('Error:', error.message);
    }
  };
  unFollowById = async ({userId, followerId}) => {
    try {
      const result = await FollowerModel.destroy({
        where: {follow_from: userId, follow_to: followerId},
      });
      if (!result) {
        console.error('Error deleting user', error);
```

```
    }
    return true;
  } catch (error) {
    console.error('Error', error.message);
    return false;
  }
};
static toDomain = async (userModel) => {
  return User.create({
    id: userModel.id,
    first_name: userModel.first_name,
    last_name: userModel.last_name,
    email: userModel.email,
    password: userModel.password,
  });
};
```

ДОДАТОК Б

Головна сторінка вебзастосунку

Header.tsx

```
import React, {useEffect, useState} from 'react';
import logo from '../../assets/logo.svg';
import {Link, useLocation} from 'react-router-dom';
import search from '../../assets/header/search.svg';
import bookmark from '../../assets/header/bookmark.svg';
import notification from '../../assets/header/notification.svg';
import profilePhoto from '../../assets/header/woman.png';
import {exitSession, getUserIdFromToken} from '../../services/utils';
const Header = () => {
  const location = useLocation();
  const [openMenu, setOpenMenu] = useState(false);
  const [userId, setUserId] = useState<number>();
  const accessToken = localStorage.getItem('accessToken');
  useEffect(() => {
    const getData = async () => {
      const userId = await getUserIdFromToken();
      setUserId(userId);
    };
    getData();
  }, []);
  return (
    <header className="header">
      <div className="header__container">
        <img src={logo} alt="" />
        {
          accessToken &&
          <>
            <nav className="header__nav">
              <ul className="list__page">
                <li className="page__item">
                  <Link to="/home"
                        className={`page__link ${location.pathname === '/home' ?
'page__link--active' : ''}`}>
```

```
                    Home
                  </Link>
                </li>
                <li className="page__item">
                  <Link to="/events"
                        className={`page__link ${location.pathname === '/events' ?
'page__link--active' : ''}`}>
                    Events
                  </Link>
                </li>
                <li className="page__item">
                  <Link to={`/profile/${userId}`}
                        className={`page__link ${location.pathname === '/profile' ?
'page__link--active' : ''}`}>
                    Profile
                  </Link>
                </li>
              </ul>
            </nav>
            <ul className="profile-list">
              <li className="list-ico0ns">
                <img src={profilePhoto} alt="" onClick={() => setOpenMenu(!openMenu)}
/>
                {
                  openMenu && (
                    <div className="logout">
                      <Link to="/" onClick={() => exitSession()}>Logout</Link>
                    </div>
                  )
                }
              </li>
            </ul>
          </>
        }
      </div>
    </header>
  );
```

```
};
export default Header;
```

MainHome.tsx

```
import React from 'react';
import {Controller, useForm} from 'react-hook-form';
import woman from '../../assets/home/cards/woman.png';
import {zodResolver} from '@hookform/resolvers/zod';
import CardPostHome from '../../components/cards/card-post/CardPostHome';
import RecommendAccount from './recommend/RecommendAccount';
import RecommendEvents from './recommend/RecommendEvents';
import {createPost} from '../../services/api';
import './style.css';
import {createPostSchema} from '../../services/schema';
type FormData = {
  postText: string
};
const MainHome = () => {
  const {
    handleSubmit,
    formState: {errors},
    control,
  } = useForm<FormData>({
    resolver: zodResolver(createPostSchema),
    mode: 'onBlur',
    reValidateMode: 'onChange',
    shouldFocusError: true,
  });
  const onSubmit = async (data: FormData) => {
    try {
      const response = await createPost(data.postText);
      window.location.reload();
    } catch (error) {
      console.error('Error:', error);
    }
  };
  return (
    <main className="main">
```

```
      <div className="home__container">
        <section className="main__home">
          <div className="posts__wrapper">
            <form className="create-post__wrapper" onSubmit={handleSubmit(onSubmit)}>
              <div className="create-post__photo">
                <img src={woman} alt="" />
              </div>
              <Controller
                name="postText"
                control={control}
                render={({field}) => (
                  <textarea
                    className="create__post"
                    placeholder="What's on your mind?"
                    {...field}
                  />
                )}
              />
              <button className="create__post__btn" type="submit">Public</button>
            </form>
            <div className="card-post__wrapper">
              <CardPostHome />
            </div>
          </div>
          <div className="recomend__wrapper">
            <RecommendAccount />
            <RecommendEvents title={'Upcoming Events'} />
          </div>
        </section>
      </div>
    </main>
  );
};
export default MainHome;
```

CardPostHome.tsx

```
import React, {useState, useEffect} from 'react';
import {getDataRequest} from '../../../services/api';
```

```
import {Post, User} from '../../../interfaces/interfaces';
import CardInfoPost from '../card-user/CardInfoPost';
interface CardPostHomeProps {
  user?: User | null;
}
const CardPostHome: React.FC = ({user}: CardPostHomeProps) => {
  const [posts, setPosts] = useState<Post[]>([]);

  useEffect(() => {
    const getData = async () => {
      const posts = await getDataRequest('/posts');
      if (Array.isArray(posts.data)) {
        const reverseData = posts.data.reverse();
        setPosts(reverseData);
      }
    };
    getData();
    console.log('posts:', posts);
  }, []);
  return (
    <>
      {posts &&
        posts.map((post) => (
          <CardInfoPost key={post.id} post={post} />
        ))
      }
    </>
  );
};

export default CardPostHome;
```

post.repository.js

```
import {Post} from '../../domain/post.js';
import {PostModel} from '../index.js';

export class PostRepository {
  createPost = async ({text, userId}) => {
```

```
    try {
      const post = await PostModel.create({text, user_id: userId});
      return await PostRepository.toDomain(post);
    } catch (error) {
      console.error('Error creating post', error);
    }
  };
  getPostById = async ({userId}) => {
    try {
      const posts = await PostModel.findAll({where: {user_id: userId}});
      if (!posts || posts.length === 0) {
        return null;
      }
      return await Promise.all(posts.map(post => PostRepository.toDomain(post)));
    } catch (error) {
      console.error('Error getting posts by userId', error);
      throw error;
    }
  };

  getAllPosts = async () => {
    try {
      const posts = await PostModel.findAll({include: ['user', 'likes']});
      return await Promise.all(posts.map(PostRepository.toDomain));
    } catch (error) {
      console.error('Error getting all posts', error);
    }
  };
  static toDomain = async (postModel) => {
    return Post.create({
      id: postModel.id,
      text: postModel.text,
      createdAt: postModel.createdAt,
      user: postModel.user,
      likes: postModel.likes,
    });
  };
}
```

comment.repository.js

```
import {Comment} from '../../domain/comment.js';
import {CommentModel} from '../index.js';

export class CommentRepository {
  createComment = async ({content, user_id, post_id}) => {
    try {
      const comment = await CommentModel.create({content, user_id, post_id});
      return await CommentRepository.toDomain(comment);
    } catch (error) {
      console.error('Error creating comment', error);
    }
  };
  getCommentsByPostId = async ({postId}) => {
    try {
      console.log('postId repo', postId);
      const comments = await CommentModel.findAll({where: {post_id: postId}});
      if (!comments) {
        return null;
      }
      return await Promise.all(comments.map(comment =>
CommentRepository.toDomain(comment)));
    } catch (error) {
      console.error('Error creating comment', error);
    }
  };
  static toDomain = async (CommentModel) => {
    return Comment.create({
      id: CommentModel.id,
      content: CommentModel.comment,
      user_id: CommentModel.user_id,
      post_id: CommentModel.post_id,
    });
  };
}
```

like.repository.js

```
import {Like} from '../../domain/like.js';
import {LikeModel} from '../index.js';
export class LikeRepository {
  createLike = async ({user_id, post_id}) => {
    try {
      const like = await LikeModel.create({user_id, post_id});
      return await LikeRepository.toDomain(like);
    } catch (error) {
      console.error('error', error);
    }
  };
  static toDomain = async (likeModel) => {
    return Like.create({
      id: likeModel.id,
      user_id: likeModel.user_id,
      post_id: likeModel.post_id,
    });
  };
}
```

commentSchema.js

```
import {DataTypes} from 'sequelize';

export const commentSchema = {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  user_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'Users',
      key: 'id',
    },
```

```
    onUpdate: 'CASCADE',
    onDelete: 'SET NULL',
  },
  post_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'Posts',
      key: 'id',
    },
    onUpdate: 'CASCADE',
    onDelete: 'SET NULL',
  },
  content: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: DataTypes.NOW,
  },
  updatedAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: DataTypes.NOW,
  },
};
```

likeSchema.js

```
import {DataTypes} from 'sequelize';

export const likeSchema = {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
```

```
  },
  user_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'users',
      key: 'id',
    },
  },
  post_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'posts',
      key: 'id',
    },
    primaryKey: true,
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: DataTypes.NOW,
  },
};
```

postSchema.js

```
import {DataTypes} from 'sequelize';

export const postSchema = {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  text: {
    type: DataTypes.TEXT,
  },
};
```

userSchema.js

```
import {DataTypes} from 'sequelize';

export const userSchema = {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  first_name: {
    type: DataTypes.STRING,
  },
  last_name: {
    type: DataTypes.STRING,
  },
  email: {
    type: DataTypes.STRING,
    unique: true,
  },
  password: {
    type: DataTypes.STRING,
  },
  refreshToken: {
    type: DataTypes.STRING,
  },
};
```

followerSchema.js

```
import {DataTypes} from 'sequelize';

export const followerSchema = {
  follow_from: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'users',
      key: 'id',
    },
```

```
    primaryKey: true,
  },
  follow_to: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'users',
      key: 'id',
    },
    primaryKey: true,
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: DataTypes.NOW,
  },
};
```

ДОДАТОК В

Сторінка користувача

CardPostProfile.tsx

```
import React, {useEffect, useState} from 'react';
import woman from '../../../assets/home/cards/woman.png';
import testPost from '../../../assets/home/cards/test-post.png';
import like from '../../../assets/home/cards/like-outline.png';
import {formatDate} from '../../../services/utils';
import {CardPostProfileProps} from '../../../interfaces/interfaces';
import comment from '../../../assets/home/cards/comment.png';
import profileImg from '../../../assets/profile/profileImg.jpg';
const CardPostProfile: React.FC<CardPostProfileProps> = (props) => {
  const [formattedData, setFormattedData] = useState<string>();
  useEffect(() => {
    const getData = async () => {
      const date = await formatDate(props.post?.createdAt);
      setFormattedData(date);
    };
    getData();
  }, []);
  return (
    <div className="card-news__post">
      <div className="card-profile__info">
        <img src={profileImg} alt="" className='profile-img__post'/>
        <div className="card-user__info">
          <h2 className="card-user__name">{props.user?.first_name + ' ' +
props.user?.last_name}</h2>
          <p className="card-user__job">Ux Designer at Divim Technology</p>
          <p className="card-user__data">{formattedData}</p>
        </div>
      </div>
      <div className="card-post__text">
        {
          props.post.text
        }
      </div>
```

```
      <div className="card-post__img">
        <img className="post__img" src={testPost} alt="" />
        <ul className="post-analytics">
          <li className="post-analytics__item">
            <button className="post-analytics__btn">
              <img src={like} alt="" />
            </button>
            <p className="post-analytics__count">15</p>
          </li>
          <li className="post-analytics__item">
            <button className="post-analytics__btn">
              <img src={comment} alt="" />
            </button>
          </li>
        </ul>
      </div>
    </div>
  );
};

export default CardPostProfile;
```

ДОДАТОК Г

Сторінка рекомендацій та підписників

Recommend.tsx

```
import React, {useEffect, useState} from 'react';
import {getDataRequest} from '../../../services/api';
import {User} from '../../../interfaces/interfaces';
import CardInfoUser from '../../../components/cards/card-user/CardInfoUser';
import {getUserIdFromToken} from '../../../services/utils';
const Recommend = () => {
  const [recommendUsers, setRecommendUsers] = useState<User[]>([]);
  const [textButton, setTextButton] = useState('Follow');
  useEffect(() => {
    const getData = async () => {
      const myId = await getUserIdFromToken();
      const getData = await getDataRequest('/getAllUsers');
      if (Array.isArray(getData.users)) {
        const filteredUsers = getData.users.filter((user: {id: string}) => user.id
!== myId);
        setRecommendUsers(filteredUsers);
      }
      console.log('getData:', getData.users);
    };
    getData();
  }, []);
  return (
    <>
      {recommendUsers !== undefined && recommendUsers.map((recommendUser) => (
        <CardInfoUser key={recommendUser.id} user={recommendUser}
textButton={textButton}
                      setTextButton={setTextButton} />
      ))}
    </>
  );
};

export default Recommend;
```

Following.tsx

```
import React, {useEffect, useState} from 'react';
import {getDataRequest} from '../../../services/api';
import {getUserIdFromToken} from '../../../services/utils';
import CardInfoUser from '../../../components/cards/card-user/CardInfoUser';
import {User} from '../../../interfaces/interfaces';
const Following = () => {
  const [followings, setFollowings] = useState<User[]>();
  const [textButton, setTextButton] = useState('Unfollow');
  const countFollow = 176;

  useEffect(() => {
    const getData = async () => {
      const myId = await getUserIdFromToken();
      const user = await getDataRequest(`/following/${myId}`);
      if (Array.isArray(user.data)) {
        setFollowings(user.data);
      }
    };
    getData();
  }, []);

  return (
    <>
      <h2 className="following__title">You are following {countFollow} people</h2>
      {
        followings !== undefined && followings.map((user, index) => (
          <CardInfoUser key={index} user={user} textButton={textButton}
setTextButton={setTextButton} />
        ))
      }
    </>
  );
};

export default Following;
```

Followers.tsx

```
import React, {useEffect, useState} from 'react';
import CardInfoUser from '../../../components/cards/card-user/CardInfoUser';
import {User} from '../../../interfaces/interfaces';
import {getUserIdFromToken} from '../../../services/utils';
import {getDataRequest} from '../../../services/api';

const Followers = () => {
  const [followers, setFollowers] = useState<User[]>();
  const [textButton, setTextButton] = useState('Follow');



  useEffect(() => {
    const getData = async () => {
      const myId = await getUserIdFromToken();
      const followersUser = await getDataRequest(`/followers/${myId}`);
      if (Array.isArray(followersUser.data)) {
        setFollowers(followersUser.data);
      }
    };
    getData();
  }, []);
  return (
    <>
      <h2 className="following__title">{followers ? followers.length : 0} people are
following you</h2>
      {
        followers !== undefined && followers !== undefined && followers.map((user,
index) => (
          <CardInfoUser key={index} user={user} textButton={textButton}
setTextButton={setTextButton} />
        ))
      }
    </>
  );
};
```

ДОДАТОК Д Маршрутизація

App.tsx

```
import React from 'react';
import {BrowserRouter as Router, Route, Routes} from 'react-router-dom';
import Header from './components/header/Header';
import MainHome from './pages/home-page/MainHome';
import MainLogin from './pages/login-page/MainLogin';
import UsersPage from './pages/users-page/UsersPage';
function App() {
  const accessToken = localStorage.getItem('accessToken');
  return (
    <div className="App">
      <>
        <Router>
          <Header />
          <Routes>
            {
              accessToken ?
                (
                  <>
                    <Route path="/home" element={<MainHome />} />
                    <Route path="/users" element={<UsersPage />} />
                    <Route path="/events" element={<EventsMain />} />
                    <Route path="/profile/:id" element={<MainProfile />} />
                  </>
                ) : (
                  <Route path="/" element={<MainLogin />} />
                )
            }
          </Routes>
        </Router>
      </>
    </div>
  );
}

export default App;
```