Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *Приліпко Ксенії Геннадіївни*
(ПІБ)

академічної групи *122-20-1*
(шифр)

спеціальності *122 Комп'ютерних наук*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка програмного забезпечення соціальної мережі для організації спілкування між студентами НТУ «Дніпровська політехніка»*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *122-20-1* *Приліпко Ксенії Геннадіївни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення соціальної мережі для організації спілкування між студентами НТУ «Дніпровська політехніка»*

затверджена наказом ректора НТУ «ДП» від *23.05.2024р.* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***14.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***20.06.2024 р.***

Завдання видав _____ *доц. Спірінцев В.В*
(підпис) (посада, прізвище, ініціали)

Завдання прийняла до виконання _____ *Приліпко К.Г.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *21.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 117 с., 60 рис., 2 табл., 3 дод., 25 джерел.

Об’єкт розробки: веб-орієнтована соціальна мережа для організації спілкування студентів Національного технічного університету «Дніпровська політехніка».

Мета кваліфікаційної роботи: розробка веб-орієнтованої соціальної мережі для організації спілкування студентів Національного технічного університету «Дніпровська політехніка», що реалізується за допомогою технологій стеку MERN.

У вступі проаналізовано актуальність використання соціальних мереж у сучасному світі та важливість проблеми комунікації між студентами університету. Крім того, у цій частині роботи визначено мету кваліфікаційної роботи та перелік задач для вирішення проблеми.

У першому розділі виконано аналіз існуючих рішень, конкретизовано мету кваліфікаційної роботи, розглянуто призначення розробки та сформована постановка завдання, розтлумачено призначення розробки та область її використання.

У другому розділі визначено технології та засоби реалізації додатка, змодельовано проєкт веб-орієнтованого застосунку, позначено склад параметрів технічних засобів, визначено вхідні та вихідні дані, висвітлено роботу соціальної мережі та описано функціональні можливості додатка.

У третьому, економічному, розділі розраховано трудомісткість інформаційної системи, підраховано вартість розробки та визначено необхідний проміжок часу для реалізації проєкту.

Практичне значення полягає у створенні веб-орієнтованої інформаційної системи, що забезпечує зміцнення студентської спільноти і створення умов для успішного навчання та особистісного розвитку кожного учасника учбового процесу.

Актуальність розробки інформаційної системи для автоматизації діяльності компанії в сфері електронної комерції не викликає сумніву та визначається відсутністю готового рішення із необхідним функціоналом.

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, СОЦІАЛЬНА МЕРЕЖА, СТУДЕНТИ, КОМУНІКАЦІЯ, БАЗА ДАНИХ, REACT, MERN, MONGODB.

ABSTRACT

Explanatory note: 117 p., 60 figs., 2 tables, 3 app., 25 sources.

Object of development: a web-oriented social network for the organization of communication of students of the National Technical University "Dniprovsk Polytechnic".

The purpose of the qualification work: the development of a web-oriented social network for organization the communication of students of the Dnipro Univercity of technology, which is implemented using MERN stack technologies.

The introduction analyzes the relevance of using social networks in the modern world and the importance of the problem of communication between university students. In addition, this part of the work defines the purpose of the qualification work and the list of tasks for solving the problem.

In the first section, an analysis of existing solutions was carried out, the purpose of the qualification work was specified, the purpose of the development was considered and the statement of the task was formed, the purpose of the development and the scope of its use were explained.

In the second section, the technologies and means of implementation of the application are defined, the project of the web-oriented application is modeled, the composition of technical means parameters is marked, the input and output data are defined, the work of the social network is highlighted, and the functionality of the application is described.

In the third, economic, section, the labor intensity of the information system is calculated, the cost of development is calculated, and the necessary period of time for the implementation of the project is determined.

The practical significance lies in the creation of a web-oriented information system that ensures the strengthening of the student community and the creation of conditions for successful learning and personal development of each participant in the educational process.

The urgency of developing an information system for automating the company's activities in the field of e-commerce is beyond doubt and is determined by the lack of a ready-made solution with the necessary functionality.

List of keywords: INFORMATION SYSTEM, SOCIAL NETWORK, STUDENTS, COMMUNICATION, DATABASE, REACT, MERN, MONGODB.

ЗМІСТ

РЕФЕРАТ 3

ABSTRACT 4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8

ВСТУП 9

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ 11

1.1. Загальні відомості з предметної галузі 11

1.1.1. Основні поняття з предметної галузі 11

1.1.2. Аналіз існуючих рішень 12

1.2. Призначення розробки та область застосування 22

1.3. Підстава для розробки 25

1.4. Постановка завдання 26

1.5. Вимоги до інформаційної системи 29

1.5.1. Вимоги до функціональних характеристик 29

1.5.2. Вимоги до інформаційної безпеки 31

1.5.3. Вимоги до складу та параметрів технічних засобів 32

1.6. Вимоги до інформаційної та програмної сумісності 33

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ 34

2.1. Функціональне призначення програми 34

2.2. Опис застосованих математичних методів 35

2.3. Опис використаних технологій та мов програмування 35

2.3.1. Архітектура програмного забезпечення 35

2.3.2. Стек технологій 37
2.3.3. Додаткові модулі стеку технологій 40
2.3.4. Компонентний підхід 42
2.3.5. Використані мови програмування 44
2.4. Опис структури системи та алгоритмів її функціонування 46
2.4.1. Структура системи 46
2.4.1.1 Декомпозиція 46
2.4.1.2 Логічна структура 48
2.4.1.3 Сценарії діяльності користувачів 50
2.4.1.4 Файлова структура 51
2.4.2. Структура бази даних 54
2.4.2.1 Концептуальна модель бази даних 54
2.4.2.2 Логічна модель бази даних 57
2.4.2.3 Фізична модель бази даних 58
2.5. Обґрунтування та організація вхідних та вихідних даних 60
2.6. Опис розробленої системи 61
2.6.1. Використані технічні засоби 61
2.6.2. Використані програмні засоби 61
2.6.3. Виклик та завантаження програми 63
2.6.4. Опис інтерфейсу користувача 63
РОЗДІЛ 3 82
3.1. Розрахунок трудомісткості та вартості розробки програмного продукту 82
3.2. Розрахунок витрат на створення програми 86
ВИСНОВКИ 89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 91

ДОДАТОК А. Лістинг програми 93

ДОДАТОК Б. Відгук керівника економічного розділу 116

ДОДАТОК В. Перелік файлів на диску 117

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

НТУ ДП – Національний технічний університет «Дніпровська політехніка»;

API – Application Programming Interface;

HTTP – HyperText Transfer Protocol;

HTTPS – HyperText Transfer Protocol Secure;

БД – база даних;

ПЗ – програмне забезпечення;

REST – Representational State Transfer;

DOM – Document Object Model;

HTML – Hypertext Markup Language;

CSS – Cascading Style Sheets;

ЕОМ – електронна обчислювальна машина;

ЦП – центральний процесор;

ОЗП – операційна пам’ять.

ВСТУП

Сучасні Web-додатки відіграють надзвичайно важливу роль у цифровій епосі, дозволяючи людям створювати вражаючу онлайн-присутність. Завдяки Web-сайтам користувачі отримують можливість швидко і зручно знайти актуальну та комплексну інформацію на будь-які теми, крім того, онлайн-ресурси надають доступ до різноманітних банківських послуг, що дозволяє зручно та ефективно проводити фінансові операції. Ще декілька позитивних функцій виконують веб-ресурси: організацію робочих процесів, планування завдань, управління проєктами та спільну роботу над документами. Сайти стають не лише візитівкою в Інтернеті, але й потужним інструментом у спілкуванні та забезпеченні зручного обміну інформацією, ідеями та досвідом, пошуку нових друзів та партнерів для спільних проєктів.

Сьогодні у світі, де швидкість життя перевершує будь-які очікування, а кожен день приносить нові виклики та можливості, важливо мати місце, де можна знайти підтримку, знайомих та єдину мету, спільноту, що ділить ваші інтереси та цінності. Об'єктом дослідження є створення нового інструменту для спілкування студентів Національного технічного університету "Дніпровська політехніка" - соціальної мережі "DniprotechInConnect"!

Передбачається, що завдяки цьому веб-додатку студенти зможуть знайти своїх однодумців, обговорити найактуальніші теми навчання та студентського життя, отримати корисні поради від досвідчених колег та студентів старших курсів, виявити товаришів для спільних проєктів й поділитися своїми досягненнями та ідеями. "DniprotechInConnect" - це простір, де кожен студент впевнений в своїй приналежності, де кожен відчуває, що його думка буде почута і врахована.

Метою створення соціальної мережі є сприяння зміцненню студентської спільноти та створення умов для успішного навчання та особистісного розвитку кожного учасника учбового процесу, а відкрите та дружнє середовище, де кожен

студент може розвиватися як особистість та знайти підтримку у своєму навчанні та кар'єрному зростанні допоможе досягти необхідних результатів.

Для досягнення поставленої мети необхідно вирішити такий перелік задач:

– проаналізувати аналогічні веб-ресурси з метою визначення переваг та недоліків їхніх функціональних можливостей, рівня безпеки інформації, технічних параметрів та сумісності з іншими програмами та обладнанням;

– визначити функції та завдання веб-додатку, можливості, які він надаватиме, а також його відповідність потребам та очікуванням користувачів;

– сформувати основні вимоги до розробки програмного забезпечення із інтуїтивно зрозумілим та привабливим інтерфейсом, високим рівнем захисту даних та ефективною роботою з іншими системами, забезпечити масштабованість для обслуговування росту користувачів та утримання стабільної роботи платформи;

– обрати необхідний стек технологій та мов програмування з урахуванням їхньої придатності для розвитку та підтримки системи в майбутньому для ефективного та зручного використання під час створення програмного продукту;

– розробити структуру та алгоритми обробки вхідних та вихідних даних для забезпечення надійності та стабільності системи, ефективної взаємодії з базами даних та правильної обробки даних для подальшого використання в системі.

Практичне значення полягає у створенні веб-орієнтованого додатка, який об'єднає студентську громаду, стане зручним інструмент для обміну ідеями, знаннями, досвідом та можливостями та сучасним простором для активного спілкування та співпраці між студентами Національного технічного університету "Дніпровська політехніка".

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Основні поняття з предметної галузі

Сьогодні неможливо уявити існування людства без інформаційних технологій, адже вони охоплюють широкий спектр програмних засобів, комп’ютерних систем, мереж зв’язку та інших інструментів. Ці технології використовуються для селекції, комплексної обробки, безпечної передачі, надійного зберігання, ефективного захисту та стратегічного управління інформацією з метою автоматизації процесів, оптимізації робочих потоків, підвищення продуктивність у людській діяльності.

Слід відзначити, що інформаційні технології є важливим складовим компонентом у комп’ютерних науках. Комп'ютерна наука – це наука, що досліджує функціонування інформаційних технологій, фокусуються на теоретичних аспектах обчислювальних систем, таких як алгоритми, структури даних, архітектура комп'ютерів тощо. З точки зору теоретичної та математичної перспективи вона досліджує основні принципи створення програм і систем, а також забезпечує наукове обґрунтування процесів розгортання, інтеграції та взаємодії інформаційних технологій, які у свою чергу охоплюють практичне використання теоретичних знань з комп’ютерної науки для створення різноманітних програмних продуктів і систем, таких як програми, веб-сайти, бази даних, мережі тощо.

Предметною областю даної кваліфікаційної роботи є соціальні мережі, які на сьогодні є однією з найпопулярніших сфер застосування усіх можливостей інформаційних технологій у сучасному світі, адже являють собою сукупні проєкти, які мають виконувати задачі для досягнення деякої системної мети.

Соціальна мережа – це онлайн-платформа, яка дозволяє користувачам створювати профілі різних типів приватності, спілкуватися з іншими

користувачами, обмінюватись інформацією та будь-яким контентом. Вона є інструментом формування сучасного інформаційного простору, впливає на спосіб отримання, обробки та обміну інформацією. Крім того, соціальна мережа є не лише інструментом для особистого спілкування, а й відіграє не менш важливу роль у бізнесі, маркетингу, політичній діяльності, суспільному житті тощо.

Дана кваліфікаційна робота направлена на розробку веб-орієнтованої соціальної мережі. Основні поняття, що використовуються в ній, включають:

– веб-браузер: прикладне програмне забезпечення, яке дозволяє користувачеві взаємодіяти з інформацією, яка знаходиться на веб-сторінках;

– веб-сервер: апаратне та програмне забезпечення, що приймає запити від клієнта, обробляє їх та надає відповідь, використовуючи необхідні протоколи, наприклад, HTTP (англ. HyperText Transfer Protocol) – протокол передачі даних;

– клієнт-сервер: архітектурна модель програмного забезпечення, яка показує розподілення процесів обміну даними між клієнтською програмою, яка ініціює запити, та сервером, який надає відповіді на ці запити;

– база даних: організований комплекс взаємопов’язаних даних, які впорядковані за зручним методом для легкого маніпулювання ними.

1.1.2. Аналіз існуючих рішень

У сучасному світі важливе значення має здатність веб-сервісу задовольнити усі потреби користувача, тому поєднання різного функціоналу стає ідеальним рішенням для забезпечення максимального комфорту студентам під час використання розробки даної кваліфікаційної роботи. Отже для того, щоб досягти поставленої мети необхідно визначити переваги і недоліки готових рішень та зрозуміти реальність створення нового продукту, що відповідатиме вимогам студента найкращим чином.

Соціальні мережі поділяються на декілька типів:

– соціальні мережі загального спрямування – платформи для спілкування та обміну контентом;

– професійні – орієнтовані на професіоналів та бізнес, використовується для пошуку працівників або роботодавців, налагодження ділових контактів;

– мікроблоги – ресурси для обміну короткими повідомленнями, думками або актуальною інформацією;

– відеохостинг – платформи для розміщення, перегляду та коментування відеоконтенту;

– соціальні мережі для знайомств – призначенні для знайомств та пошуку пари, базуються на взаємній симпатії користувачів;

– соціальні мережі для геймерів – дозволяють налаштовувати стрімінг відеоігор, транслювати ігровий процес та взаємодіяти із глядачами;

– форуми – платформи для обміну знаннями у різних сферах діяльності.

Якщо в Інтернеті ввести запит на соціальну мережу для студентів, одразу отримуємо наукові об’єднання, які залучають студентів певних галузей або науковців до спільного обговорення досліджень та створення нових проєктів. Отже, першими розглядаємо професійні соціальні мережі. Аналізуючи такі платформи можна виділити декілька ключових рішень, які допоможуть у створенні ефективної платформи для студентів.

Так, наприклад, наукові соціальні мережі: Academia.edu, Social Science Research Network, ResearchGate, Computer Science Student Network – універсальні застосунки, які стали невід’ємною частиною сучасного ученого середовища. У цих платформах пропонуються універсальні інструменти для обміну науковими дослідженнями, публікацій робіт, а також доступу до широкого спектру академічних ресурсів. Вони створюють можливості для пошуку людей за спільними інтересами, спілкування та співпраці між користувачами. Запропоновані платформи дозволяють отримувати відгуки та зворотній зв’язок від колег однодумців та сприяють розвитку наукових дискусій. Огляд прикладів функціоналу сайтів приведено на наведено далі.

На першій сторінці Social Science Research Network (рис. 1.1) є можливості створення профілю користувача, пошуку необхідної інформації та переміщення сторінками сайту.

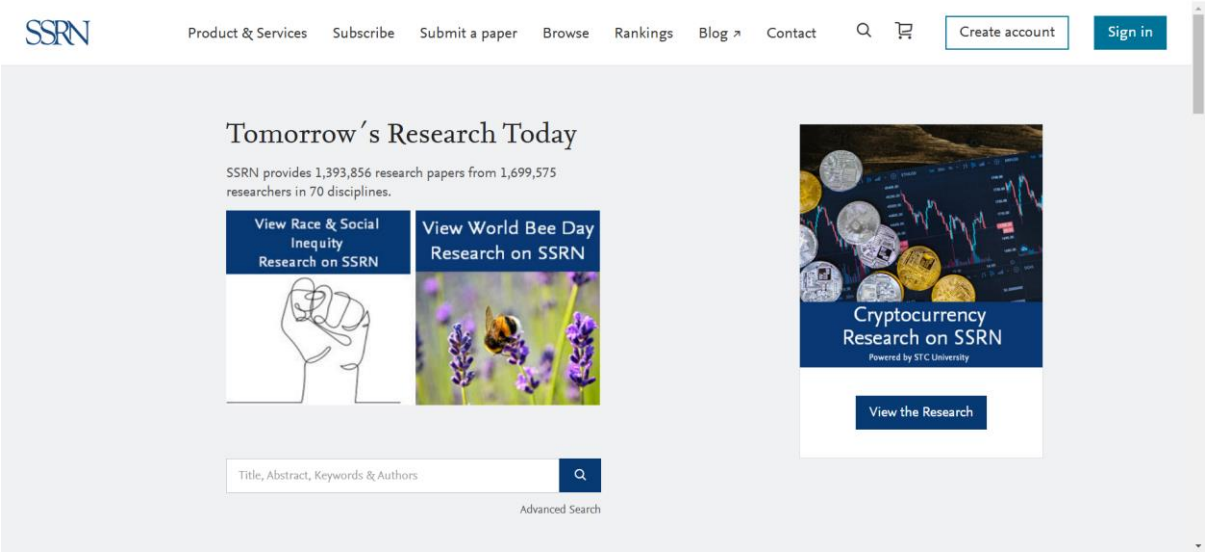

Рис. 1.1. Перша сторінка Social Science Research Network

Блок першої сторінки Academia.edu (рис.1.2) представляє наукові спілки за інтересами, які створені на цьому сайті.

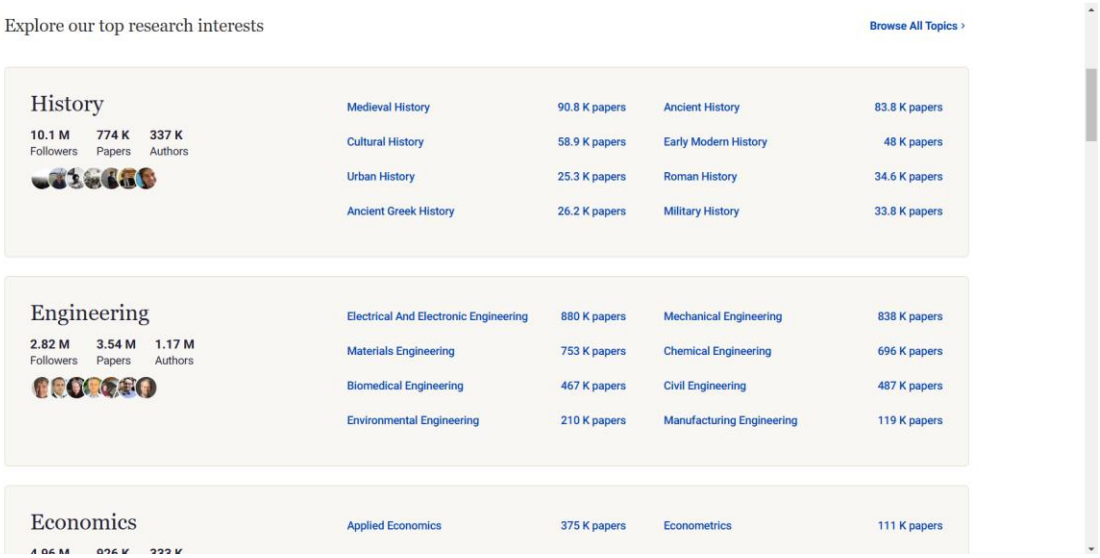

Рис. 1.2. Блок сторінки Academia.edu

На платформі Computer Science Student Network один з розділів дозволяє створення груп за інтересами (рис. 1.3).

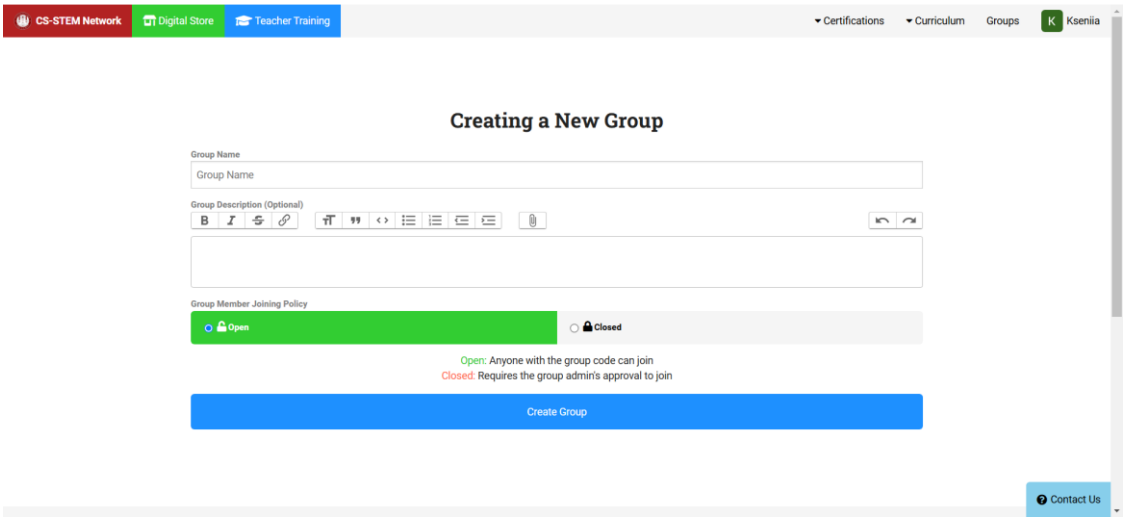

Рис. 1.3. Розділ платформи Computer Science Student Network

Для повноти картинки, було розглянуто мережу професійного спрямування LinkedIn, функціонал якої направлений на представлення своєї кандидатури у спеціалізованих напрямках та висвітлення навичок, пошук роботодавців або навпаки працівників на відкриту вакансію. Платформа надає можливість професіоналам не лише шукати роботу, але й подавати заявки безпосередньо через сайт, використовуючи свій профіль як основний документ. Головна сторінка цієї платформи показана на рис. 1.4.

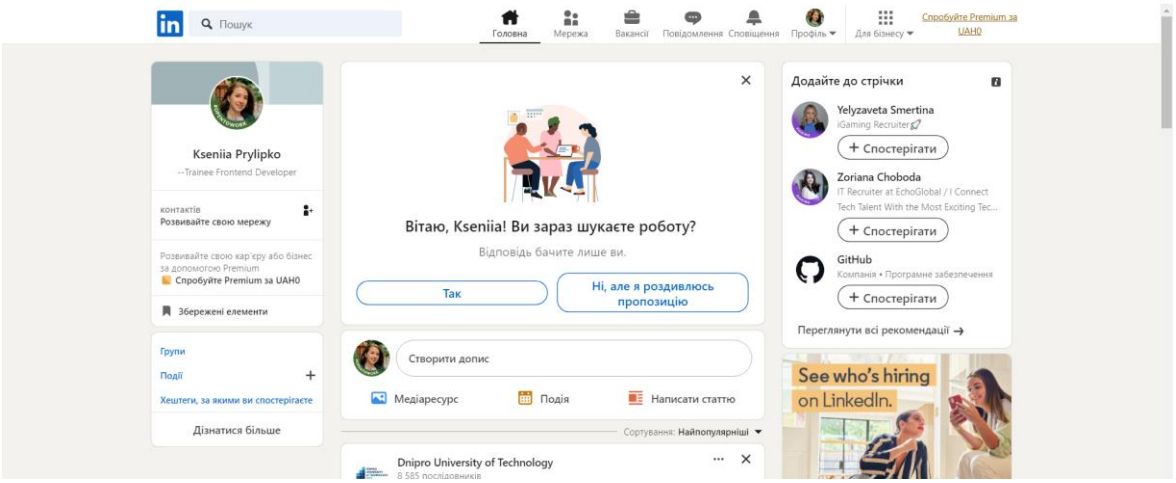

Рис. 1.4. Головна сторінка LinkedIn

Можна побачити, що у веб-додатку присутній функціонал для публікацій статті, обміну новинами, думками та досягненнями, а також взаємодії з контентом інших через коментарі та обговорення. Також користувач має можливість налагоджувати зв’язки з колегами, партнерами та клієнтами, що допомагає розширювати їхню професійну мережу та отримувати рекомендації. Профілі користувачів слугують віртуальними резюме, де вони вказують свої навички, досвід роботи, освіту та досягнення, що допомагає роботодавцям знайти відповідних кандидатів, а користувачам – відповідні вакансії. Рис. 1.5 – 1.6 демонструють розділи профілю.

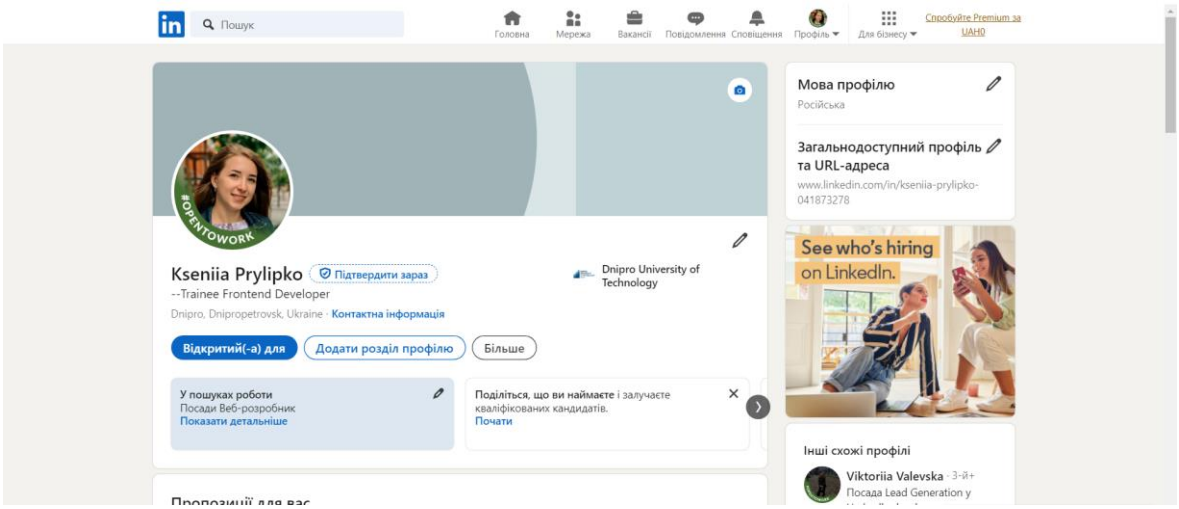

Рис. 1.5. Розділ з основною інформацією про користувача

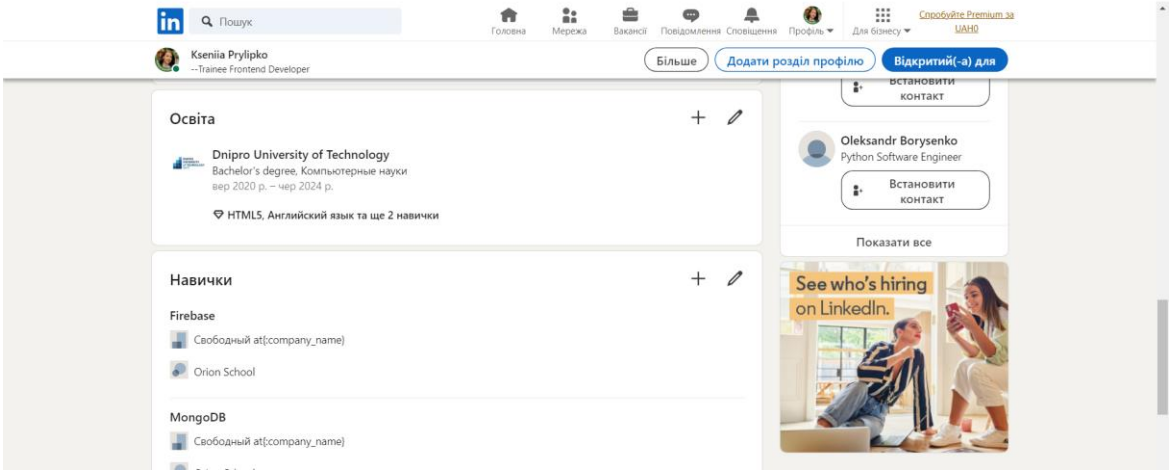

Рис. 1.6. Розділи з інформацією про освіту та навички

Загалом запропоновані веб-додатки мають майже весь необхідних функціонал: можливість створення профілю, комунікації, обміну інформацією та документами, аналітики, але всі вони занадто сильно зорієнтовані на академічних дослідженнях та наукових працях, що є надмірно складним та спеціалізованим для студентів, які шукають простішу форму взаємодії. Крім того, ці платформи не забезпечують тісного та зручного зв'язку між студентами одного університету, що є важливим аспектом для побудови міцних соціальних зв'язків в межах конкретного навчального закладу.

Оскільки застосунок даної кваліфікаційної роботи перш за все виступає соціальною мережею, головними функціями якої стають можливість спілкування й пошук нових знайомих та друзів, необхідно розглянути соціальні мережі загального спрямування. Для того, щоб ефективно реалізувати базові можливості, було проведено детальний аналіз відповідно вже існуючі рішень і у цій сфері. Подібні додатки мають на меті виконувати наступні функції: обмін повідомленнями, спільний доступ до мультимедійних матеріалів, створення груп за інтересами та подій, підтримка віртуальних спільнот тощо. Було розглянуто такі соціальні мережі як Facebook, Instagram, Telegram та Viber.

Огляд першої соціальної мережі Facebook показав, що ця платформа надає користувачам можливість створення профілів, що показано на рис. 1.7.

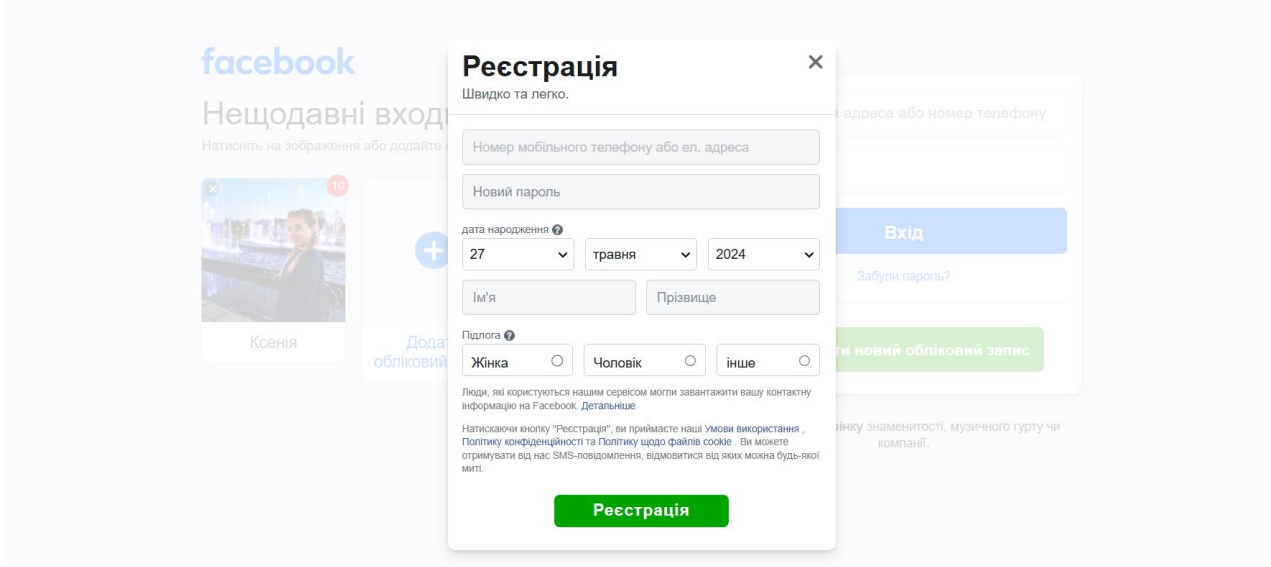

Рис. 1.7. Створення нового профілю в Facebook

Крім того, у веб-застосунку є можливості пошуку знайомих, обміну інформацією та повідомленнями, створення спільнот за інтересами, публікації фото та відео контенту, загальні функції управління Facebook, всі вони представлені на рис.1.8.

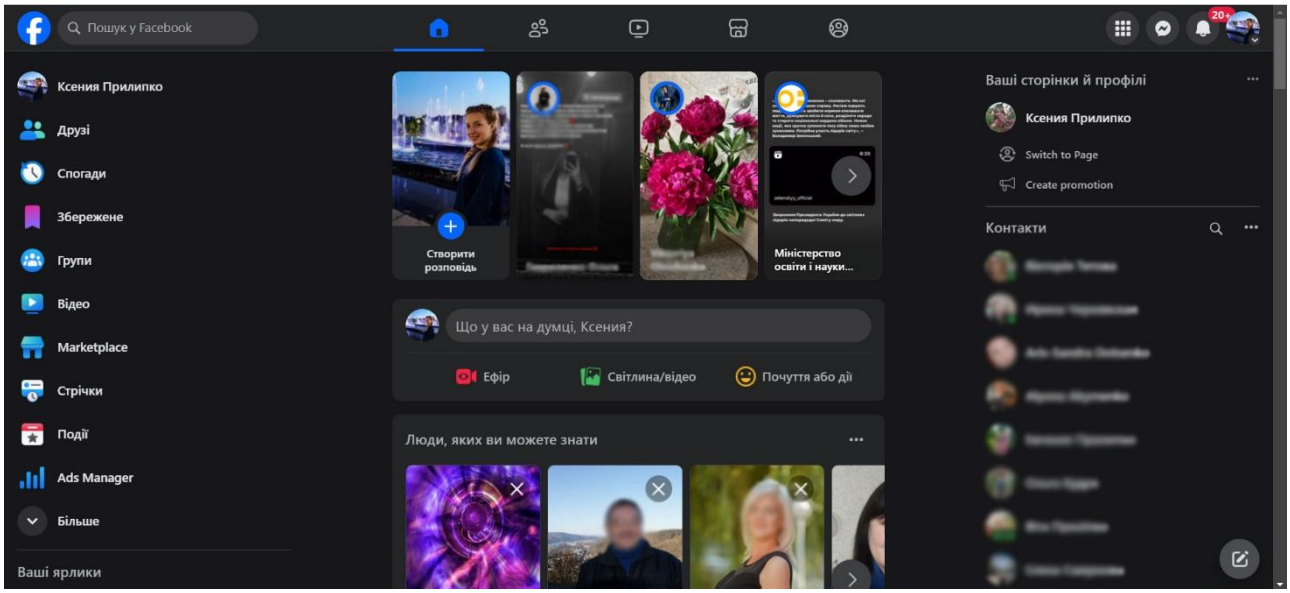

Рис. 1.8. Можливості управління веб-додатком Facebook

Не менш важливою можливістю для користувачів веб-платформи, що розглядається, є створення чатів (рис.1.9).

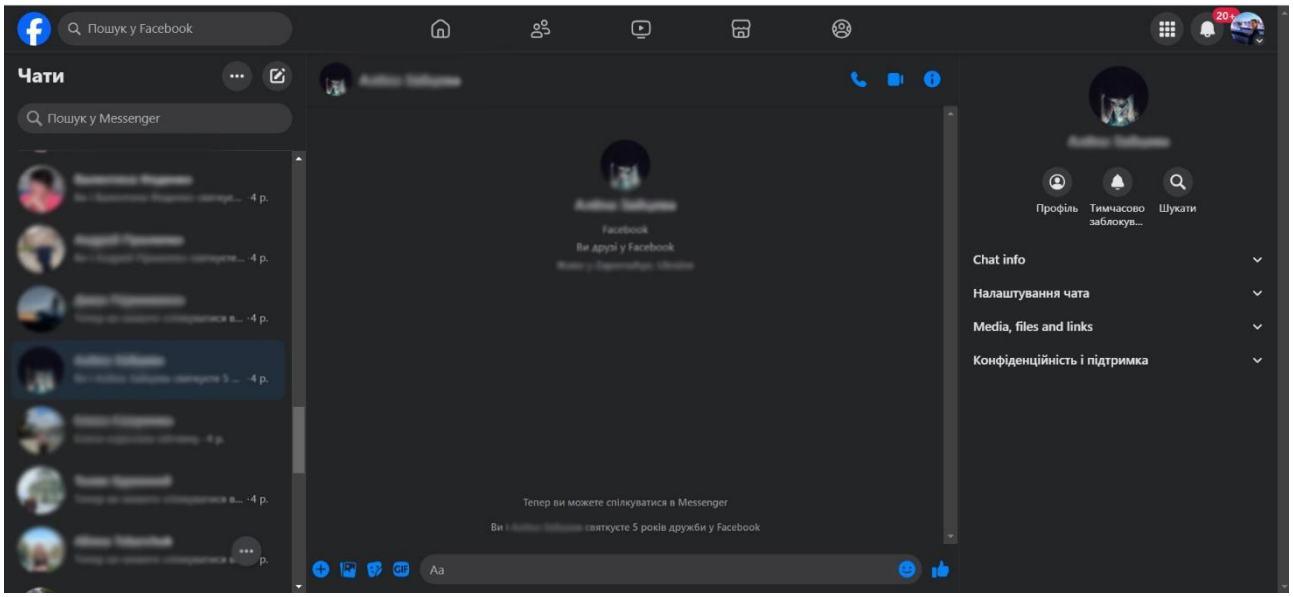

Рис. 1.9. Створення чатів

Крім того, ця соціальна мережа підтримує функціонал для бізнесу, що дозволяє компаніям створювати сторінки для своїх брендів та продуктів, де вони можуть розміщувати інформацію про свої товари та послуги й взаємодіяти із потенційними клієнтами. Додатковим функціоналом стає можливість просувати свої продукти через платні рекламні кампанії. Наприклад, продаж товарів, що показано на рис. 1.10.

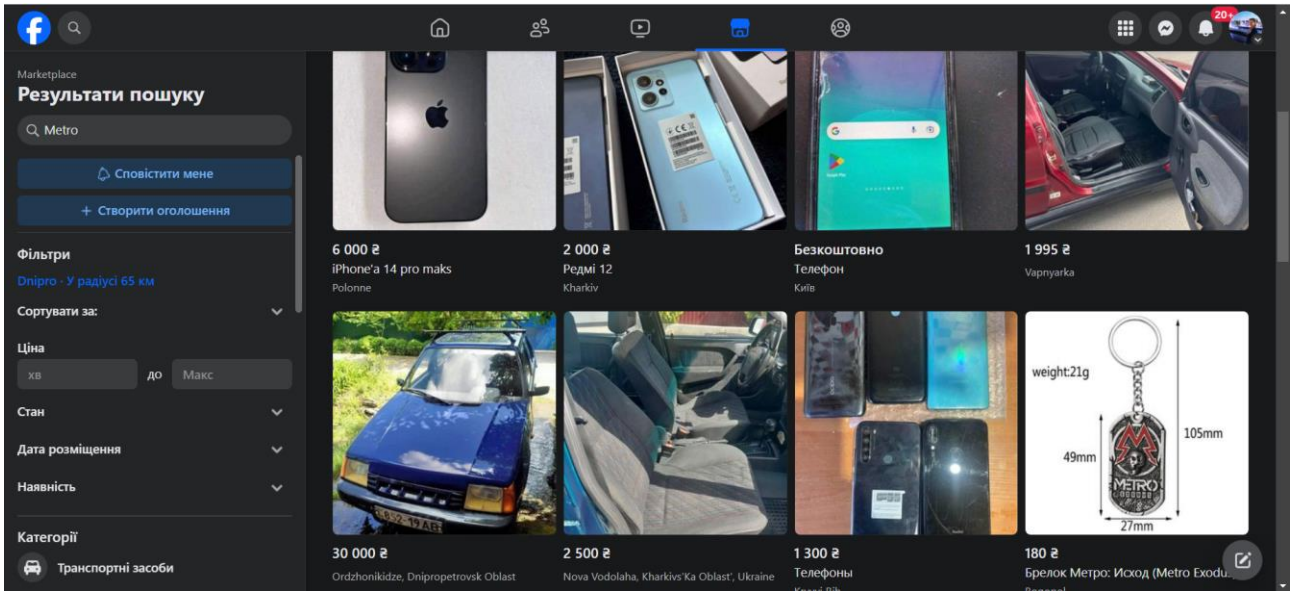

Рис. 1.10. Продаж товарів на Facebook

Аналіз Instagram показав, що ця соціальна мережа також багатофункціональна, як і попередня, але вона більшою мірою фокусується на обміні фото та відео контентом. Користувачі мають можливість створювати профіль, розробляти публікації у вигляді фотографій або відео, застосовувати різноманітні ефекти для вражаючого представлення контенту, а також взаємодіяти з контентом інших користувачів: ставили лайки, залишати коментарі, підписуватись на сторінку для перенесення необхідної інформації у загальну стрічку новин. Загалом ця платформа створена для самовираження та спілкування, більш орієнтована на молодь. Сторінка можливих цікавих публікацій для користувача показана на рис. 1.11.

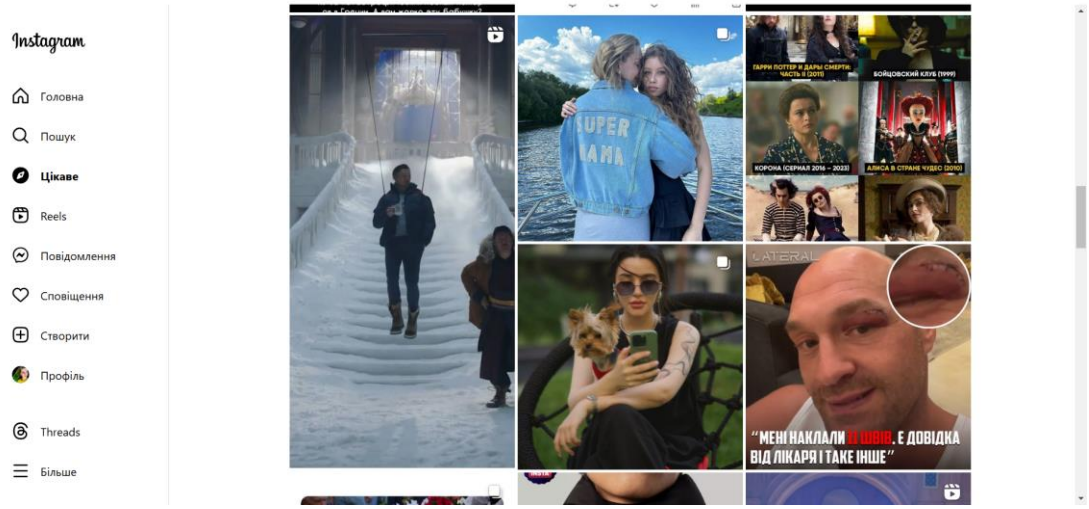

Рис. 1.11. Сторінка можливих цікавих публікацій для користувача

Якщо казати про такі популярні мережі, як Viber та Telegram, то стає зрозумілим, що їхній функціонал повністю відповідає можливостям сучасних месенджерів. Користувачі можуть сміливо обмінюватись текстовими повідомленнями, фотографіями, відео та аудіофайлами, здійснювати голосові та відео дзвінки, надсилати різноманітні типи файлів та документів. Ці месенджери також надають можливість створення загальних чатів та групових спільнот. У них також передбачено створення особистого профілю за номером телефона. Цікавим інструментом входу у веб-додаток Telegram є QR-код, який необхідно відсканувати за допомогою телефона, щоб увійти на свій профіль у месенджері. Сторінка входу у Telegram показана на рис. 1.12.

Рис. 1.12. Сторінка входу у Telegram

В ході вивчення усіх можливостей популярних застосунків стало зрозуміло, що загалом вони не враховують специфічні потреби студентів одного університету, а створені для великого кола користувачів, що ускладнює налагодження зв’язків. Крім того, ці платформи розроблені для соціальної взаємодії людей з усієї планети з розважальною метою, тому у них відсутня підтримка можливостей необхідних функцій для комфортного використання студентською спільнотою.

Оскільки розробка даної кваліфікаційної роботи має бути пов’язана із учбовою складовою, необхідно проаналізувати наявність рішень по комунікації в освітній сфері. Із таких маємо Zoom, Microsoft Teams, Outlook, Google Meet.

Zoom та Google Meet мають схожі функціональні можливості. Вони є популярними сервісами для відеоконференцій та віртуальних зустрічей в реальному часі. Крім того, вони інтегровані з іншими сервісами гугл, що полегшує організацію зібрань та роботу над спільними проєктами. Особливістю функціонування Zoom конференцій можна виділити можливість обміну екраном та спільної роботи над документом під час зустрічі.

Microsoft Teams та Outlook є складовими елементами пакету Office 365, тому ці сервіси інтегруються один з одним та іншими частинами цього програмного забезпечення.

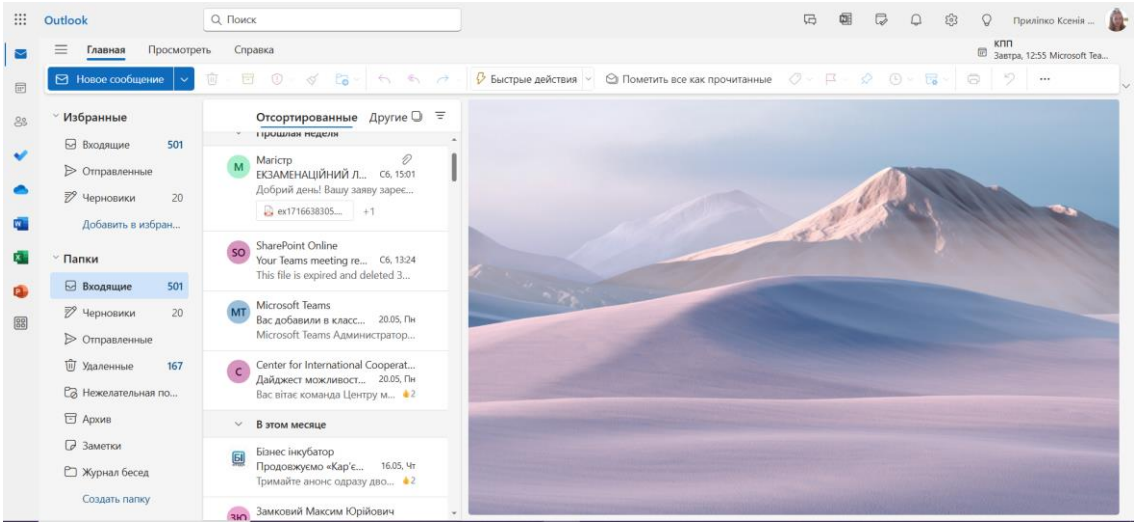

Рис. 1.13. Можливість спілкуватися у чатах Outlook

Запропоновані веб-додатки також надають можливість користувачам планувати й створювати відео зустрічі та конференції, спілкуватися у чатах (рис.1.13), обмінюватися файлами та проводити спільну роботу над документами. Основною відмінністю від попередніх веб-ресурсів є те, що Office 365 – це платний хмарний веб-сервіс.

Розглянуті рішення мають багато зручних функції для роботи студентів, пошуку друзів та взаємодії один з одним. Та відсутність загальної сторінки новин робить таке спілкування обмеженим одним факультетом або взагалі лише однією групою спеціальності.

Та попри наявність великої кількості застосунків, які можуть надавати потрібні послуги, не існує єдиної платформи, яка б стала зручним інструментом у спілкуванні та взаємодії студентів Національного технічного університету «Дніпровська політехніка». Жоден із запропонованих сервісів не включає усі функціональні потреби користувачів, які можливо задовольнити. Крім того, ці платформи мають зручний інтерфейс лише у мобільному або десктопному застосунку. Виходячи із того, що люди звикли до легкості та зручності при роботі із технічними перевагами сьогодення, вони прагнуть використовувати застосунки із зрозумілим та комфортним інтерфейсом, але завантаження великої кількості додатків стає недоліком через швидке зменшення пам’яті, яка є необхідним та вичерпним ресурсом у час кругообігу великих об’ємів даних.

1.2. Призначення розробки та галузь застосування

Розробка програмного забезпечення соціальної мережі для організації спілкування між студентами НТУ «Дніпровська політехніка» повинна забезпечити наявність усіх необхідних функцій, які потребує користувач для комфортного використання даної платформи.

Призначення розробки цієї соціальної мережі полягає у створенні інтерактивної платформи, яка сприятиме спілкуванню та взаємодії між користувачами, полегшенню соціалізації нових студентів та пошуку необхідної

інформації. Для досягнення поставленої мети основними завданнями розробки постають:

– розробка інтерфейсу та логотипу згідно фірмових кольорів НТУ ДП;

– реалізація формування профілю користувача із введенням прізвища, ім’я, по батькові, дати народження (не обов’язково), назви факультету, номеру групи, додаткової інформації;

– розробка ресурсів для підписки на певного користувача;

– впровадження системи текстових повідомлень;

– реалізація видалення повідомлень;

– розробка функцій створення особистих чатів;

– впровадження можливості створення публікації із додаванням тексту та фото;

– надання можливості оцінювати публікації інших користувачів;

– організація інструментів для створення подій із вказанням необхідних даних: часу, дати, створення можливості корегування інформації про подію та її видалення;

– створення помічника – сервісу, куди може звертатися користувач у разі виникнення будь-яких питань.

Основна термінологія, яка буде використовуватися у даній кваліфікаційній роботі, містить загальні терміни з області інформаційних технологій та сфери соціальних мереж. Ключові слова включають: програмне забезпечення, веб-орієнтований застосунок, веб-додаток, веб-сервіс, веб-дизайн, інтерфейс, база даних, розробка, впровадження, соціальна мережа, користувач, студент, адміністратор, профіль, чат, повідомлення, стрічка новин, публікація, контент, вподобайка, подія, календар подій, помічник, мультимедія, хмарні технології, корпоративні кольори.

Терміни, які потрібні для розуміння загальних нюансів побудованої структури системи:

– розробка – процес формування, створення та удосконалення застосунку;

– база даних – платформа для зберігання та управління великих об’ємів інформації;

– соціальна мережа – поняття усієї розроблюваної платформи із базовими та додатковими структурними елементами;

– користувач – студент Національного технічного університету «Дніпровська політехніка»;

– адміністратор – представник Ради студентів Національного технічного університету «Дніпровська політехніка», який буде коригувати інформацію на сторінках, оновлювати події та відповідати на запитання;

– інтерфейс – основний вид сторінки із функціоналом для взаємодії користувача з веб-додатком;

– профіль користувача – основний елемент структури системи, який містить особисту інформацію користувача, фото та статус, кількість підписників;

– чат – переписка між двома особами;

– повідомлення – текстова інформація, що передається між користувачами;

– контент – будь-яка інформація на сторінках соціальної мережі, може бути представлена у вигляді тексту, фото, відео;

– публікація – блок, який вміщає контент, та відображається на основній сторінці соціальної мережі;

– вподобайка – позитивна реакція користувача на публікацію;

– стрічка новин – основна сторінка застосунку, яка відображає публікації користувачів;

– подія – захід на рівні університету чи факультету, який висвітлюється у календарі подій;

– календар подій – сторінка із датами у вигляді таблиці;

– помічник – чат із адміністратором;

– мультимедія – будь-який вид медіаконтенту на сайті;

– корпоративні кольори – кольори, що є фірмовими для даного навчального закладу.

Причинами виникнення необхідності розробки даного програмного забезпечення стали:

– потреба у ефективній платформі для комунікації, обміну інформації, підтримки тісних зв’язків зі студентами університету та однокурсниками для організації спільної роботи;

– важливість комунікації між студентами для підтримки один одного у питаннях навчання;

– бажання студентів вливатися у студентські спільноти та знаходити нових друзів;

– потребу в інструменті для обміну навчальними матеріалами та спільного вирішення навчальних викликів;

– зменшення кількості учасників та організаторів у різних університецьких заходах: концертах, майстер-класах, інтелектуальних та спортивних іграх, навчальних конференціях та тренінгах;

– важливість оперативного поширення важливої інформації у студентську спільноту;

– пошук студентів, які можуть надати відповіді на запитання, що турбують їхніх колег;

– бажання студентів знаходити необхідну інформацію, витрачаючи найменшу кількість часу.

Областю, у якій може використовуватись застосунок, що розробляється, є освіта у вищому навчальному закладі Національному технічному університеті «Дніпровська політехніка»

1.3. Підстава для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма спеціальності 122 «Комп’ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» від 23.05.2024р. № 469-с;

– завдання кваліфікаційної роботи на тему «Розробка програмного забезпечення соціальної мережі для організації спілкування між студентами НТУ «Дніпровська політехніка».

1.4. Постановка завдання

Мета: розробити програмне забезпечення соціальної мережі, яке надасть студентам Національного технічного університету «Дніпровська політехніка» віртуальний простір для комунікації, створення ком'юніті за інтересами, організації навчального процесу, соціальної взаємодії між студентами та обміну знаннями, а також допоможе Раді студентів університету організувати зв'язок зі студентством для вирішення нагальних питань.

Призначення: спілкування студентів НТУ «Дніпровска політехніка» в онлайн режимі, ведення особистих сторінок, висвітлювання досягнень, взаємодія користувачів на тему навчання, обмін навчальними матеріалами, можливості для організації події та встановлення нагадувань, сторінки інформативного характеру, відслідковування активності участі студентської спільноти у житті університету, пошук нових друзів та знайомих, полегшення роботи Ради студентів Національного технічного університету «Дніпровська політехніка».

Техніко-економічна сутність завдання: покращити загальний емоційний стан студентів та рівень їхньої взаємодії один з одним; залучити якомога більше здобувачів освіти до спілкування та знайомств, полегшити доступ до навчальних ресурсів; надати можливість отримувати необхідну для комфортного навчання інформацію шляхом створення відповідних сторінок у веб-додатку. Забезпечити користувачів соціальної мережі зручним функціоналом та зрозумілим інтерфейсом.

Створення платформи, що базуватиметься на можливостях сучасних соціальних мереж з додаванням інформаційних сторінок, а також елементів сайту

візитівки, допоможе студентам швидко знаходити відповіді на запитання, ділитися конспектами та записами лекцій, організовувати групові проекти та спільноти за інтересами, а також отримувати підтримку від однолітків та старших студентів. Така ініціатива сприятиме покращенню організації та ефективності навчального процесу, зменшить витрати часу на комунікацію та пошук необхідної інформації. Не менш важливим стане те, що авторизуватися у соціальній мережі зможе лише студент Національного технічного університету «Дніпровська політехніка», що допоможе об’єднати здобувачів освіти та створити комфортні умови для соціалізації у даному навчальному середовищі. Слід також врахувати наявність додаткових можливостей для адміністратора соціальної мережі.

Структура соціальної мережі для студентів має складатися з профілів студентів, особистих сторінок користувачів, стрічки новин, чатів для спілкування, інформаційних сторінок, календаря подій, додаткових елементів для адміністратора сайту.

В свою чергу структура профілю користувача повинна мати наступні елементи:

– особисту інформацію: прізвище, ім’я, по батькові, дата народження (не обов’язково), назва факультету та спеціальності, номер групи,;
– навчальні інтереси та цілі;
– фото та статус;
– підписники, друзі;
– статистика: коментарі, пости, вподобайки.

Структура чатів користувача має складатися з особистих та групових чатів. Календар подій має містити день, місяць, рік, час, назву події, місце проведення, необов’язковий опис події: детальну інформацію про захід, його програму, та можливість ставити нагадування. Інформаційна сторінка виглядатиме як звичайна сторінка сайту візитівки, вона надаватиме можливість завантажувати всі необхідні дані. Структура сторінок для адміністратора сайту повинна складатися з форм для заповнення оновлення даних на інформаційних сторінках

та додавання подій у календар. Кожна сторінка сайту має бути оснащена спеціальною кнопкою для відкриття чату із адміністратором, який зможе надати відповіді на будь-які питання.

Основні показники, які мають відслідковуватись – активність користувача у мережі, кількість вподобань, кількість коментарів, участь у заході. На основі цих показників буде зручно регулювати пропозиції Ради студентів університету щодо дозвілля для учасників навчального процесу, а також відслідковувати зацікавленість студентів в участі у запропонованих заходах.

Вихідна інформація – це результат ретельної обробки та аналізу вхідних даних, представлених у зручній та доступній формі для сприйняття студентами. Ця інформація включає в себе різноманітні аспекти, що стосуються навчання та соціальної активності студентів: їхні профілі, навчальні матеріали, розклад подій, активність у групах за інтересами та інші важливі елементи. Особливу увагу слід приділити розкладу подій та стрічці новин, ключовим способом візуалізації яких стане графічне представлення. Таке рішення дозволить краще відображати складну інформацію, полегшуючи її розуміння, що допоможе користувачеві швидко орієнтуватися в новинах, відстежувати оновлення, планувати свій час та брати активну участь у житті вищого навчального закладу.

Для збору вхідної інформації будуть створені спеціально розроблені форми для введення даних. Основною вимогою до організації збору та передачі в обробку вхідної інформації стане інтуїтивно зрозумілий і зручний інтерфейс для користувача. Для полегшення процесу заповнення даних і мінімізації ймовірності помилок, необхідно забезпечити максимально зрозумілі назви комірок для заповнення та надати додаткові підказки, де це необхідно. За можливості створити заздалегідь підготовлені варіанти інформації, які можуть використовуватись при заповненні форм. Така автоматизація стосується переліку можливих факультетів, спеціальностей, курсів, дат.

Зважаючи на можливість введення помилкових або неактуальних даних, постає необхідність забезпечити можливість контролю та корегування даних.

Тому можливість редагування і видалення введених записів і даних у систему є однією з вимог до веб-додатку.

Слід розрізняти функціональні можливості для студентів та адміністратора сайту, адже два типи користувачів повинні мати змогу вирішувати свої завдання згідно статусу.

Користувач типу студент повинен мати в доступі такі функції:

– створення профілю користувача та його редагування;

– доступ до матеріалів пов’язаних із навчанням;

– перегляд актуальної інформації щодо подій та заходів в університеті;

– можливість приєднуватись до загальних чатів за інтересами, створювати особисті чати;

Користувача типу адміністратор необхідно забезпечити наступними функціональними можливостями:

– управління профілями користувачів;

– модерація контенту на інформаційних сторінках та сторінці календаря подій;

– доступ до статистичних даних про активність користувачів;

– надання допомоги студентам у разі виникнення проблем із веб-додатком або загальних навчальних питань.

1.5. Вимоги до інформаційної системи

1.5.1. Вимоги до функціональних характеристик

Користувач має вводити дані стандартним шляхом, за допомогою клавіатури, у відповідні комірки, що були розроблені під час формування інтерфейсу. Комірки, куди необхідно вводити інформацію, мають надавати можливість користувачам вводити відкриті відповіді або обирати з представлених, заздалегідь визначених даних, крім того, сформовані комірки мають бути елементами відповідних форм для заповнення. Даними, які мають варіанти для вибору, можуть бути назва факультету та спеціальності, номер групи,

дата народження, дата проведення заходу. Дата повинна обиратися за допомогою спеціально призначених для цього полів. Формат дати повинен виглядати – РРРР-ММ-ДД.

Необхідно врахувати наявність вже створеного профілю користувача та можливість пошуку такого профілю задля зменшення кількості помилок під час заповнення відповідних форм. Крім того, користувачі повинні мати можливість корегування введених даних у відкритих для них елементах соціальної мережі: профілі, публікацій тощо.

Для використання будь-якого контенту на сторінках соціальної мережі необхідно надати функціонал для завантаження файлів, фотографій, відеоконтенту та документів.

Усі дані, що були введені користувачем, мають бути розділені на основні та додаткові структурні елементи системи. Виходячи з цього основні елементи повинні відповідати за головні процеси і призначення системи. Вони містять в собі: профіль користувача, спільноти та чати. Додатковими елементами постають інформаційна сторінка та сторінка календаря подій.

Особливу увагу слід приділити типу користувачів, адже в залежності від цього будуть відрізнятися функціональні можливості та рівень доступу до даних. Для адміністратора дані можуть мати більш точний зміст та бути подані у вигляді, відмінному від того, що є загальним для інших користувачів.

Крім того, слід організувати можливість фільтрації та пошуку інформації. Так, наприклад, можна шукати користувача за його прізвищем або ім’ям, та реалізувати пошук спільнот так само за їхньою назвою, зручною можливістю стане пошук необхідного контенту на інформаційній сторінці. Ще однією гарною функцією може стати фільтрування публікацій і заходів за датою та назвою.

Важливо не забувати про функціональні процеси, які допоможуть збирати статистичні дані щодо участі студента у заходах. Збір повинен відбуватися на основі введеної користувачем інформації.

Весь контент, який знаходиться на інформаційній сторінці, має бути зручно розміщеним для кращого його розуміння користувачем. Додатковою можливістю необхідно зробити завантаження будь-якої інформації з цієї сторінки веб-сайту.

Використання документо-орієнтованої системи керування бази даних допоможе структурувати та правильно організувати необхідний набір даних, дозволить легко маніпулювати даними та створювати запити різної складності, забезпечить управління доступом до даних залежно від типу користувача.

1.5.2. Вимоги до інформаційної безпеки

Інформаційна безпека – це набір заходів та засобів, які розроблені для захисту даних від несанкціонованого використання, спотворення та зміни, привласнення, видалення тощо. Для забезпечення інформаційної безпеки у випадку розробки соціальної мережі слід обов'язково впроваджувати реєстрацію акаунту кожного нового користувача та авторизацію для входу у створений акаунт. Крім цього слід додержуватись основних вимог до надійності паролю, таких як:

- наявність символів верхнього та нижнього регістрів;
- обмеження мінімально допустимої кількість символів – 8, максимально допустимої – 25;
- заборона використання таких символів як: /, , *, &, %, $, та інших схожих;
- використання лише символів латинського алфавіту;
- наявність мінімум трьох цифр.

Зберігати усі паролі необхідно в зашифрованому вигляді для їхнього захисту у разі неправомірного доступу до бази даних. Крім того, при реєстрації та авторизації краще використовувати електронну пошту у якості логіну.

Особливу увагу слід приділяти управлінню доступом. Керований доступ до інформації та ресурсів забезпечить конфіденційність даних та допоможе уникнути їхнього небажаного витоку.

Забезпечення контролю за помилками та обробка потенційних ситуацій, що можуть загрожувати безпеці інформації, також допоможе зберегти застосунок від

небажаного втручання в його систему. Краще за все організувати виведення повідомлень про причини помилки та можливості їхнього вирішення.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для реалізації програмного забезпечення соціальної мережі для організації спілкування між студентами НТУ «Дніпровська політехніка» буде використана клієнт-серверна архітектура, що передбачатиме різні технічні параметри для кожної сторони.

Для забезпечення стабільної роботи соціальної мережі серверна частина складатиметься з двох основних серверів: серверу застосунку та серверу бази даних. Таким чином, щоб забезпечити нормальну продуктивність системи, необхідно виконувати наступні мінімальні вимоги до серверу застосунку:

- наявність стабільного інтернет-з’єднання;
- оперативна пам’ять не менше 8 GB;
- центральний процесор класу Intel Core i3 (тактова частота не менше 4,3 ГГц, кількість ядер не менше 4);
- твердотільний накопичувач не менше 250 GB.

Мінімальною вимогою для серверу бази даних є пам’ять не менше 1Tb для зберігання великого об’єму даних.

Оскільки розроблювана соціальна мережа є веб-орієнтованою платформою, для користувачів не буде необхідності в покупці та налаштуванні якогось складного обладнання або новітніх технічних засобів. Рішення створення саме веб-додатку надає можливість забезпечити користувачів комфортними умовами використання застосунку без зайвих складнощів по встановленню чи налаштуванню. Таким чином, щоб отримати доступ до функцій розроблюваного ПО, потрібно лише мати електронний пристрій, який підтримує веб-браузер та фізичну або віртуальну клавіатуру чи будь-який пристрій вводу даних і мишку або сенсорний екран. Для зручного використання веб-сервісу співвідношення сторін має бути 16:9, а найкраще відображення інтерфейсу розраховане на

дисплей з розширенням 1920x1080 пікселів. Не обов’язково повністю дотримуватись усіх запропонованих параметрів, але краще використовувати дисплеї з діагоналлю більше середньої для комфортного використання соціальної мережі.

1.6. Вимоги до інформаційної та програмної сумісності

Для забезпечення оптимальної працездатності та сумісності програмного забезпечення з різними середовищами, основною вимогою стає коректна робота з такими браузерами як Google Chrome, FireFox, Microsoft Edge і Safari. Крім того, використання останніх версій цих браузерів надасть більш коректне відображення інтерфейсу соціальної мережі.

Для програмного середовища висуваються наступні вимоги:

- операційна система: Windows 10 або вище;
- фреймворк React.js;
- серверна платформа Node.js останньої версії.

Для сервера бази даних основною вимогою є забезпечення належної конфігурації та підтримки MongoDB.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення програми

Соціальна мережа для студентів Національного технічного університету «Дніпровська політехніка» створена для розширення можливостей під час навчання та покращення комунікації студентів один з одним. Розроблений застосунок має такий основний функціонал:

– формування профілю користувача із введенням ім’я, електронної пошти та особистого паролю;

– у профілі користувача додавання та зміна такої інформації: імені, дати народження, назви факультету та спеціальності, номеру групи, додаткової інформації;

– система обміну текстовими повідомленнями із можливістю додавання різних типів мультимедії;

– реалізація зміни та видалення повідомлень;

– розробка функцій створення особистих та групових чатів із додатковими можливостями керування групами;

– створення користувацьких публікації із можливістю додавання тексту, фото або відео;

– надання можливості коментувати та оцінювати публікації інших користувачів;

– організація інструментів для створення подій із вказанням необхідних даних: часу, дати, місця, створення можливості корегування інформації про подію та її видалення;

– створення помічника – сервісу, куди може звертатися користувач у разі виникнення будь-яких питань.

2.2. Опис застосованих математичних методів

Так як функціонал соціальної мережі не включає застосування специфічних математичних або фізичних формул, завдання кваліфікаційної роботи за своєю сутністю не передбачає використання математичних методів.

2.3. Опис використаних технологій та мов програмування

2.3.1. Архітектура програмного забезпечення

При розробці веб-додатку згідно завдання кваліфікаційної роботи було використано клієнт-серверну архітектуру (рис. 2.1). Така модель обчислювальної системи широко використовується у розробці веб-орієнтованих інтерфейсів і представлена у вигляді двох основних компонентів: клієнта та сервера.

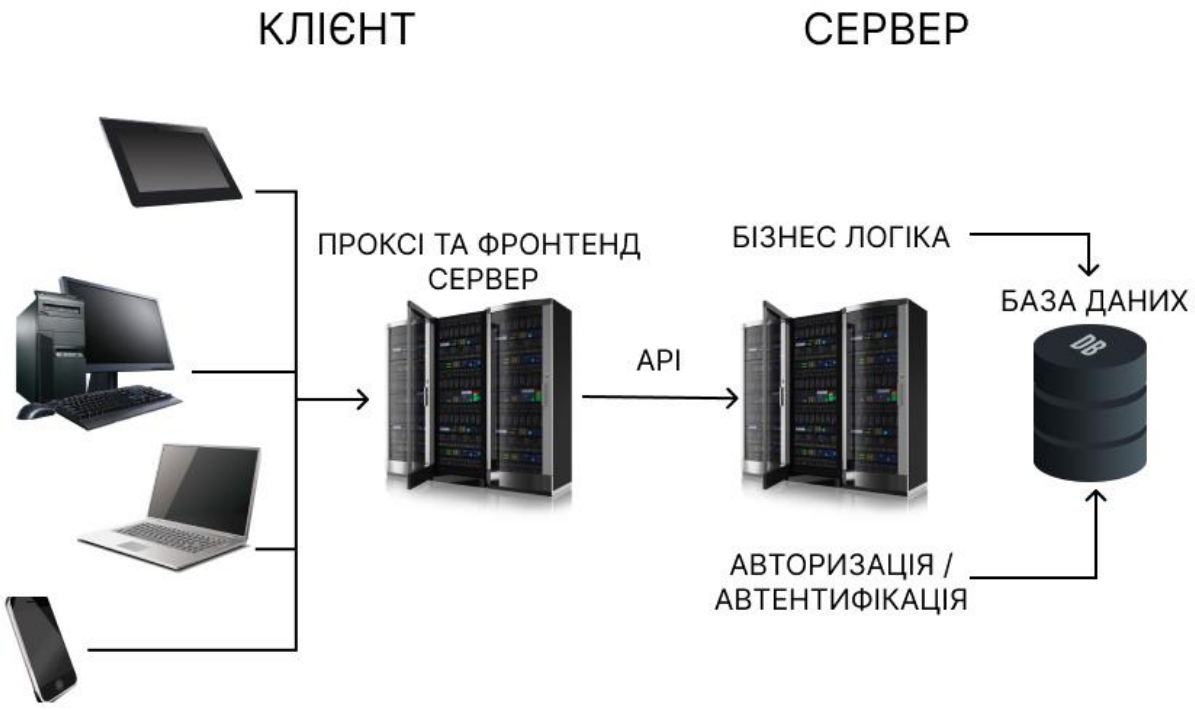

Рис. 2.1 Клієнт-серверна архітектура

Клієнтська сторона (клієнт) забезпечує взаємодію користувача із системою. Зазвичай, являє собою інтерфейс користувача, представлений веб-браузерами, мобільними додатками або іншими програмами, який обробляє введену користувачем інформацію, ініціює запити до сервера для отримання необхідних ресурсів або послуг та повертає відповідь у вигляді, зрозумілому для користувача.

Серверна частина (сервер) обслуговує HTTP-запити від клієнтських пристроїв. Представлений у вигляді модулів, що обробляють основні функції розроблюваного додатка, бази даних для структурованого й безпечного зберігання усіх даних користувачів та сервісів хешування. Загалом сервер забезпечує швидку та грамотну обробку бізнес-логіки застосунку.

Для обміну запитами та відповідями у клієнт-серверній архітектурі використовується набір протоколів та інструментів – API – для взаємодії між програмними компонентами або сервісами при розробці ПЗ. Основними типами протоколів є HTTP/HTTPS, SOAP, GraphQL, WebSockets, але найпоширенішими для веб-застосунків є протоколи HTTP/HTTPS. HTTP є основним протоколом для обміну даними у WWW (Всесвітній павутині) за допомогою передачі гіпертекстових документів HTML. Для забезпечення зашифрованим обміном інформацією та конфіденційності при взаємодії між клієнтом та сервером використовується HTTPS, який є розширенням HTTP. Обидва протоколи застосовують методи HTTP: GET (отримання даних), POST (створення нових даних), PUT (оновлення даних), DELETE (видалення даних) – для маніпулювання ресурсами.

У розробці проєкту згідно завдання кваліфікаційної роботи використовується трирівнева клієнт-серверна архітектура. Це обумовлено кількістю використовуваних серверів під час реалізації соціальної мережі. Наявність сервера бази даних, окрім веб-сервера та клієнта, надає більше можливостей та гнучкості у масштабованості системи, ніж дворівнева клієнт-серверна архітектура.

Загалом трирівнева клієнт-серверна архітектура має багато переваг у використанні:

– централізоване зберігання даних та їхня обробка за допомогою сервера спрощує управління системою;

– легка масштабованість системи як зі сторони сервера так і клієнта відповідно зростаючим потребам додає гнучкості системі;

– можливість розподілу навантаження для підвищення продуктивності та стійкості до збоїв системи;

– забезпечення високої надійності зберігання та захисту особистих даних користувачів;

– можливість розширення функціоналу за допомогою інтеграції з іншими сервісами.

2.3.2. Стек технологій

Реалізація програмного забезпечення соціальної мережі проводилась із використанням повнофункціонального стеку технологій для створення сучасних та зручних веб-додатків – MERN. Це середовище розробки складається із чотирьох ключових компонентів:

– бази даних MongoDB;

– веб-фреймворку Express.js;

– бібліотеки React.js;

– середовища виконання JavaScript на сервері Node.js.

Загалом структуру визначеного стеку технологій у графічному вигляді представлено на рисунку 2.2.

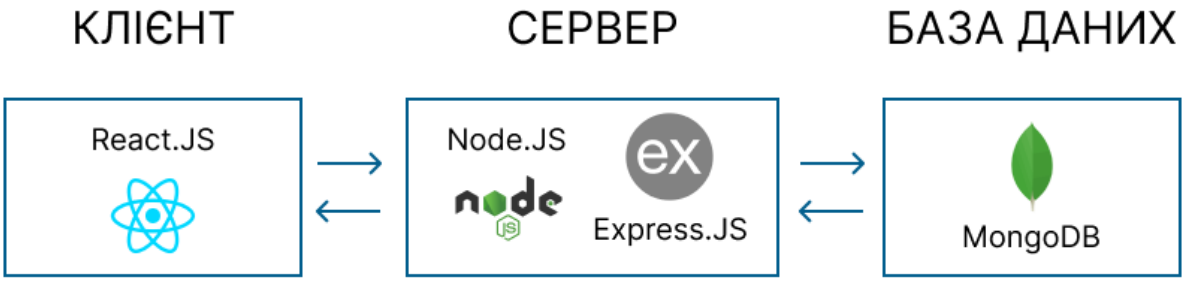

Рис. 2.2. Структура стеку технологій MERN

Клієнтська частина веб-додатку в стеку MERN реалізується за допомогою React.js, потужною бібліотекою JavaScript, яка використовується для створення динамічних та інтерактивних користувацьких інтерфейсів. Дана бібліотека застосовує компонентний підхід, що дозволяє повторне використання блоків інтерфейсу в різних частинах додатку. Такий підхід значно полегшує розробку та підтримку додатку, зменшує дублювання коду. Розширення мови JavaScript до JavaScript XML або JSX у React.js дозволяє писати HTML-подібний код безпосередньо у файлах JavaScript, спрощує створення елементів DOM, робить код більш читабельним і легким у підтримці. Наявність віртуальної об'єктної моделі документу, що є ідентичною копією реальної DOM, значно підвищує ефективність рендеренгу додатка. Крім того, React.js має велику спільноту розробників та багату екосистему інструментів, що дозволяє легко знаходити рішення для різноманітних завдань і швидко впроваджувати нові функціональні можливості при розробці веб-застосунку.

Express.js та Node.js наступні компоненти представленого стеку та залежні один від одного, адже Express.js – це веб-фреймворк з великим набором середовищ виконання та функцій для Node.js, які забезпечують обробку HTTP-запитів, формування відповідей, маршрутизацію та інші функціональні вимоги додатка на стороні сервера. Створення надійного REST API – інтерфейсу для обміну інформацією через Інтернет за допомогою визначених правил зв'язку із програмними компонентами системи – є основною функцією даного фреймворку, а його міжсерверні програми дозволяють додавати обробку запитів на різних етапах їх виконання.

Node.js являє собою середовище для виконання JavaScript на сервері. Він базується на двигуні V8 від Google та дозволяє виконувати JavaScript-код поза браузером. Ефективні та паралельні виконання і обробка багатьох одночасних запитів без очікування завершення попередніх базуються на використанні неблокуючих введення/виведення, подіях та колбеках. Крім того, Node.js має

достатньо великий репозиторій бібліотек та пакетів, що значно спрощує розробку та додає функціональності додаткам.

Загалом Node.js і Express.js разом утворюють потужний інструмент для розробки сучасних веб-додатків, що значно спрощує створення серверу завдяки простому і зрозумілому API, дозволяє визначати маршрути для різних HTTP-методів та URI, забезпечує простий механізм для обробки запитів і формування відповідей, включаючи підтримку JSON, а використання Middleware дозволяє створювати додаткову обробку запитів, наприклад, автентифікацію, логування, обробку помилок тощо.

І останній компонент стеку база даних MongoDB – нереляційна база даних, яка формує колекції із документами, що зберігаються у форматі BSON. Така бінарна JSON-подібна форма представлення структур даних у зазначеній БД дозволяє зручно моделювати складні об’єкти та легко налаштовувати взаємовідносини між ними.

MongoDB сервер зберігає декілька баз даних, які у свою чергу є контейнерами для колекцій. Колекції можна порівняти із таблицями у реляційних базах даних, але головною відмінністю є можливість зберігання в одній колекції документів із різною структурою. Документи в MongoDB є основними одиницями зберігання даних, гнучка структура яких дозволяє зберігати інформацію у форматі ключ-значення, де значення можуть бути представлені навіть у вигляді масивів або вкладених документів. Загалом MongoDB має ряд переваг:

– архітектура бази даних не вимагає визначення однієї чіткої схеми збереження даних, що робить MongoDB гарно адаптованою до сучасних викликів при розробці веб-додатків;

– індексування підвищує швидкість та зручність взаємодії із даними;

– вбудована підтримка реплікації даних підвищує доступність, надійність збереження даних та забезпечує розподіл навантаження;

– база даних надає потужні інструменти для агрегації даних, виконання складних операцій над даними.

Отже, стек технологій MERN працює наступним чином. За допомогою React.js створюється інтерфейс веб-додатку, із яким взаємодіє користувач через клієнтські компоненти, які відправляють запит до серверної частини проєкту через API. Node.js приймає запити від клієнта на стороні сервера, тоді як Express.js обробляє ці запити та організовує взаємодію із базою даних: виконання бізнес-логіки та формування відповідей на запити. База даних представлена MongoDB, яка у свою чергу забезпечує гнучке зберігання даних.

Стек технологій MERN має чисельні переваги у веб-розробці:

– використання єдиної мови програмування – JavaScript, як на клієнтській, так і на серверній стороні, значно спрощує процес розробки та підтримки коду;

– легкість створення динамічних та яскравих інтерфейсів користувача із усім необхідним функціоналом;

– зрозумілість і простота у налаштуванні серверної логіки, за допомогою мінімалістичного і гнучкого фреймворку для обробки HTTP-запитів і маршрутизації;

– забезпечення високопродуктивного виконання серверного коду для ефективної обробки багаточисельних одночасних запитів, гнучкість у роботі з даними для легкого масштабування додатка і обробки великих обсягів інформації.

Усе це забезпечує злагоджену та ефективну розробку сучасних веб-додатків з високою продуктивністю, масштабованістю та зручністю для розробників.

2.3.3. Додаткові модулі стеку технологій

Для швидкої, безпечної та зручної розробки було встановлено й використано наступні додаткові пакети бібліотек, які значно полегшили роботу із фреймворком Express.js:

– axios – надає простий і зручний інтерфейс для взаємодії з API і веб-серверами за допомогою промісів, підтримує різні типи запитів, можливість встановлення заголовків, передачу параметрів та обробку відповідей.

Можливість використання бібліотеки як у середовищі Node.js, так і у веб-браузері робить її відмінним вибором для розробників, які пишуть універсальні додатки;

– nodemon – інструмент, який надає можливість автоматичного перезавантаження сервера або додатка у разі внесення змін у код програми, підтримує відстеження змін у різних типах файлів, включаючи JavaScript, TypeScript, JSON, CSS, інших текстових файлів і навіть шаблонів, простий у налаштуванні під потреби конкретної розробки;

– dotenv – дозволяє завантажувати змінні середовища в процес додатка, що забезпечує легке управління конфігураційними параметрами, безпечне зберігання зашифрованих даних;

– bcrypt – бібліотека для хешування паролів перед їх зберіганням у відповідних системах. Вона перетворює пароль у хеш-значення, яке неможливо перетворити назад у вхідний пароль, автоматично включає унікальну «сіль» для кожного паролю, що надає безпеку навіть у випадку використання однакових паролів користувачами;

– passport – використовується для управління автентифікацією і авторизацією користувачів: реалізує стратегії з використанням локальних паролів, OAuth, OpenID, JWT, які визначають, як саме користувачі будуть автентифікуватися, забезпечує інфраструктуру для управління сесіями користувачів, зберіганням інформації про автентифікацію та авторизацію в сесійному сховищі;

– jsonwebtoken – інструмент для безпечного обміну даними в мережевих додатках. Забезпечує компактний, самостійний і захищений спосіб передачі інформації між сторонами у вигляді об'єкта JSON. Часто використовується для автентифікації користувачів після успішного входу у систему, для захисту API, забезпечуючи тимчасовий доступ до ресурсів без необхідності повторної аутентифікації на кожному запиті;

– cors – контролює доступ до ресурсів між різними доменами, що дозволяє запобігати небажаним запитам з інших сайтів;

– helmet – допомагає захистити додатки Express.js шляхом налаштування різних HTTP-заголовків: запобігає від різних типів атак, підвищує безпеку передачі даних, контролює доступ до певних браузерних API і функціональностей, дозволяє налаштування політики безпеки контенту для обмеження джерел, від яких браузер може завантажувати ресурси;

– morgan – застосовується в Node.js додатках, що підтримуються Express.js або іншими HTTP-серверами для створення журналу запитів сервера, що дозволяє відстежувати та аналізувати активність додатка. Він надає змогу додатку автоматично логувати всі HTTP-запити без необхідності ручного оброблення, підтримує різні типи логування та налаштування його формату;

– mongoose – надає простий та зручний спосіб моделювання даних додатка: створення схем для відображення структур даних, використання методів CRUD (Create, Read, Update, Delete) для взаємодії із даними з MongoDB, підтримка хуків життєвого циклу документів, визначення і керування відносинами між документами у базі даних;

– socket – це бібліотека JavaScript для створення веб-додатків у реальному часі. Вона забезпечує двосторонню, подієво-орієнтовану комунікацію між клієнтом і сервером. Socket.IO часто використовується разом з Node.js для побудови серверів у реальному часі, наприклад, для чат-додатків, систем відстеження місцезнаходження, ігор та інших додатків, які потребують миттєвого обміну даними.

2.3.3. Компонентний підхід

У розробці соціальної мережі було використано компонентний підхід до структурування і стилізації інтерфейсу. Він є методологією, що базується на створенні відокремлених, незалежних модулів, які мають чітко визначену функціональність і можуть бути повторно використані в різних частинах програми. Компонентний підхід описує взаємодію цих модулів між собою на основі подій.

Загалом такий підхід у програмуванні представляє собою еволюцію об'єктно-орієнтованого програмування (ООП), вирішуючи деякі його основні проблеми, зокрема, проблему з наслідуванням. Відмінність полягає в тому, що компоненти не є набором полів та методі, а представляють собою незалежні об'єкти, призначені для виконання конкретно визначених завдань.

Організація розробки інтерфейсу передбачає використання відокремлених компонентів шляхом згрупування схожих функціональних блоків коду за принципом DRY (Don't Repeat Yourself), який спрямований на уникнення повторення рядків коду. Саме компонент є структурною одиницею описуваного підходу, який має чітко визначений зовнішній вигляд, що повністю характеризує залежність елементу від програмного оточення. Так, наприклад, на рисунку 2.3 показано інтегрування компонента TopBar у різних сторінках сайту: Home та Profile.

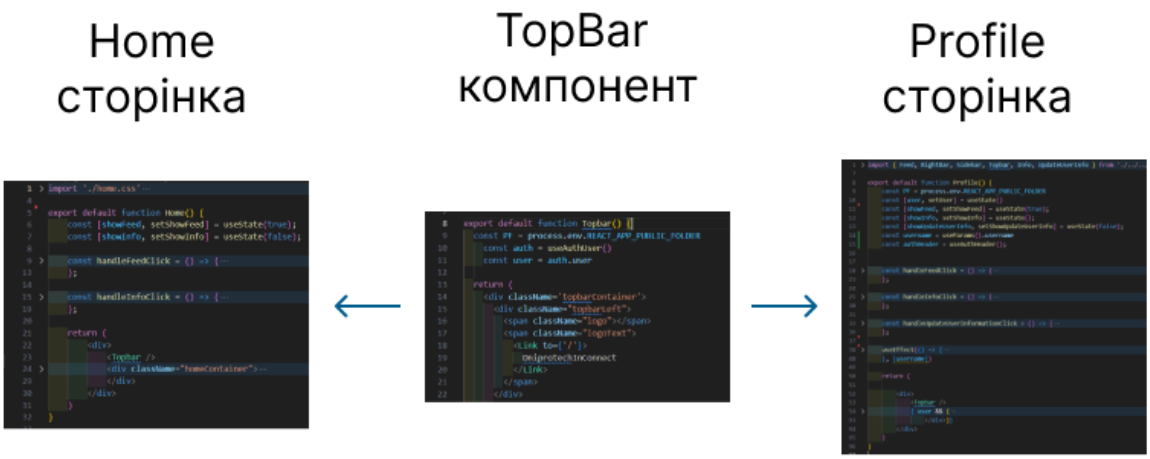

Рис. 2.3. Інтеграція компонента TopBar у сторінки проєкту: Home та Profile

Основними особливостями компонентного підходу є:

– самодостатність компонентів, кожен з яких виконує конкретну функцію і може існувати незалежно від інших частин інтерфейсу;

– ієрархічна структура, яка дозволяє вкладати компоненти один в одного та будувати складні інтерфейси з мінімальними зусиллями;

– модифікатори для зміни стану, які надають змогу легко налаштовувати стилі в залежності від різних станів елементів;

– легка підтримка і розширення завдяки самостійним елементам, що спрощує розробку нових функцій і додавання нових блоків інтерфейсу без зміни існуючого коду.

Цей підхід сприяє значній ефективності у розробці програмного забезпечення, оскільки спрощує управління кодом і забезпечує його легку масштабованість. Компонентний підхід підвищує рівень абстракції програмної системи і сприяє поліпшенню її підтримки та розширення у майбутньому.

2.3.4. Використані мови програмування

Веб-орієнтована розробка передбачає використання стандартної мови розмітки HTML (Hypertext Markup Language). Вона використовується для створення та представлення веб-сторінок у браузерах, формування структури інформації на веб-сторінках за допомогою різноманітних елементів та тегів, які визначають різні типи контенту, такі як текст, зображення, таблиці, форми тощо. HTML є базовим будівельним блоком веб-інтерфейсів і використовується разом з CSS і JavaScript для створення повноцінних інтерактивних та стилізованих веб-сторінок.

Для оформлення зовнішнього вигляду веб-сторінки використовувалась мова стилів CSS (Cascading Style Sheets). Вибір елементів HTML, до яких будуть застосовуватись стилі, виконується за допомогою селекторів, які можуть включати назви тегів, класи, ідентифікатори, атрибути тощо. Безпосередньо зовнішній вигляд компоненту веб-сайту визначається за допомогою властивостей та їхніх значень. CSS використовує каскадність для визначення стилю, який має застосовуватись до конкретного елемента, коли на нього впливає більше одного правила, та спадкування, що дозволяє елементам успадковувати стилі від їхніх батьківських елементів. Крім того, ця мова стилів дозволяє створювати анімації і переходи, що роблять веб-сайти більш інтерактивними і привабливими.

Обраний стек програмування MERN базується на використанні однієї мови програмування JavaScript. Такий підхід забезпечує універсальність застосування мови як у клієнтській, так і у серверній частинах та дозволяє легко взаємодіяти між елементами всієї платформи додатка. Крім того, розробка на одній мові зменшує час на розгортання, тестування і відладку, сприяє масштабуванню додатків через непотрібність вивчення нових мов та технологій для різних частин системи.

JavaScript – це мова програмування високого рівня, яка широко використовується для створення інтерактивних веб-застосунків і сайтів. Основні переваги використання JavaScript полягають у її інтерпретованій природі, що дозволяє розробникам відразу бачити результати змін у коді без необхідності попередньої компіляції. JavaScript легко інтегрується з HTML і CSS, що дозволяє створювати повнофункціональні веб-інтерфейси.

Ця мова також підтримує динамічну типізацію, що означає автоматичне визначення типів змінних під час виконання програми. Це спрощує розробку і зменшує кількість помилок. Завдяки великій кількості бібліотек і фреймворків JavaScript розробники можуть ефективно використовувати готові рішення для швидкого створення функціональності веб-додатків.

JavaScript підтримує основні парадигми програмування, такі як імперативне програмування, функціональне програмування і об'єктно-орієнтоване програмування, що дає розробникам гнучкість у створенні складних структур даних і покращенні організації коду.

Однією з ключових особливостей JavaScript є його асинхронна природа, яка дозволяє ефективно опрацьовувати операції, що не потребують негайного завершення без блокування інших задач. Інтеграція з DOM (Document Object Model) робить JavaScript ідеальним для взаємодії з елементами веб-сторінки і обробки подій.

Завдяки великій спільноті розробників і відкритому коду JavaScript продовжує розвиватися і залишається однією з найпопулярніших та зручних мов програмування для веб-розробки.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Структура системи

2.4.1.1. Декомпозиція

Декомпозиція – процес розбиття складної задачі, системи або проєкту на менші, більш керовані і прості компоненти чи модулі. Цей процес став ключовим аспектом на перших етапах розробки системи та був застосований для проєктування структури об'єктів і їх функціонування в додатку (табл. 2.1).

Таблиця 2.1

Декомпозиція структури соціальної мережі

<table>
<tr><td rowspan="15">Соціальна мережа</td><td rowspan="4">Дизайн</td><td colspan="4">Підбір корпоративних кольорів НТУ ДП</td></tr>
<tr><td colspan="4">Ілюстрація фото профіля</td></tr>
<tr><td colspan="4">Ілюстрація логотипу</td></tr>
<tr><td colspan="4">Ілюстрація обкладинки профіля</td></tr>
<tr><td rowspan="14">Користувач</td><td rowspan="8">Редагування / Додавання</td><td colspan="3">Ім’я</td></tr>
<tr><td colspan="3">Пошта</td></tr>
<tr><td colspan="3">Фото профілю</td></tr>
<tr><td colspan="3">Зображення обкладинки</td></tr>
<tr><td colspan="3">Місто</td></tr>
<tr><td colspan="3">Факультет</td></tr>
<tr><td colspan="3">Група</td></tr>
<tr><td colspan="3">Додаткова інформація</td></tr>
<tr><td rowspan="3">Створення</td><td rowspan="2">Пост</td><td colspan="2">Текст</td></tr>
<tr><td colspan="2">Картинка</td></tr>
<tr><td>Чат</td><td>Повідомлення</td><td>Текст</td></tr>
<tr><td>Підписка</td><td colspan="3">Інший користувач</td></tr>
</table>

Продовж. табл. 2.1

		Вподобання	Пост	
			Інформаційний блок	
		Перегляд	Інформаційний Блок	Текст
				Картинка
			Події/заходи	Назва
				Дата
				Час
	Адміністратор	Вхід	Пошта	
			Пароль	
		Створення / Перегляд / Видалення	Інформаційний блок	Текст
				Картинка
			Події/заходи	Назва
				Дата
				Час
		Вподобання	Пост	
			Інформаційний блок	
	Профіль	Реєстрація	Ім’я	
			Пошта	
			Пароль	
			Повторний пароль	
		Вхід	Пошта	
			Пароль	
	Домашня сторінка	Відображення	Інформаційний компонент	
			Пости користувача	
			Інформаційний компонент	
			Пости користучів	

Продовж. Табл. 2.1

			Інформаційного компоненту
	Сторінка подій/заходів		Календар із подіями/заходами
	Месенджер		Чат користувачів

Декомпозиція надала розуміння загальної структури проєкту, допомогла визначити основні функції і завдання, які необхідно реалізувати, виділити функціональні блоки, які можуть бути реалізовані у вигляді окремих компонентів, встановити зв’язки між елементами веб-додатку та їхню взаємодію. Загалом декомпозиція дозволила покращити керованість проєктом, спростити процес розробки, зменшити ризики помилок і полегшити масштабування системи.

2.4.1.2. Логічна структура

Для забезпечення чіткого уявлення про організацію та взаємозв’язок компонентів в системі, що визначають взаємодію елементів проєкту між собою, було побудовано логічну структуру. Таким чином соціальна мережа являє собою веб-додаток із такими сторінками:

- авторизація;
- реєстрація;
- головна сторінка;
- профіль користувача;
- месенджер;
- календар.

Усі ці компоненти та їхній зв’язок утворюють логічну структуру сайту.

Так, після реєстрації, користувач перенаправляється на сторінку авторизації, проходження якої відкриває сторінку особистого профілю. У профілі користувач може переглядати свої дані, редагувати їх, а також мати доступ до сторінки месенджера для обміну повідомленнями з іншими користувачами та

календаря для перегляду запланованих подій та заходів. Наявність необхідних компонентів надає змогу користувачеві із профілю пересуватися на головну сторінку сайту.

Головна сторінка також надає доступ до інших розділів веб-додатку, таких як месенджер і календар. Користувач також може переглядати опубліковані пости своїх друзів та інформаційний компонент сайту.

Наявність у користувача статусу адміністратора відкриває більше функціональних можливостей на головній сторінці та сторінці календаря.

Опис логічної структури сайту представлено на рисунку 2.4

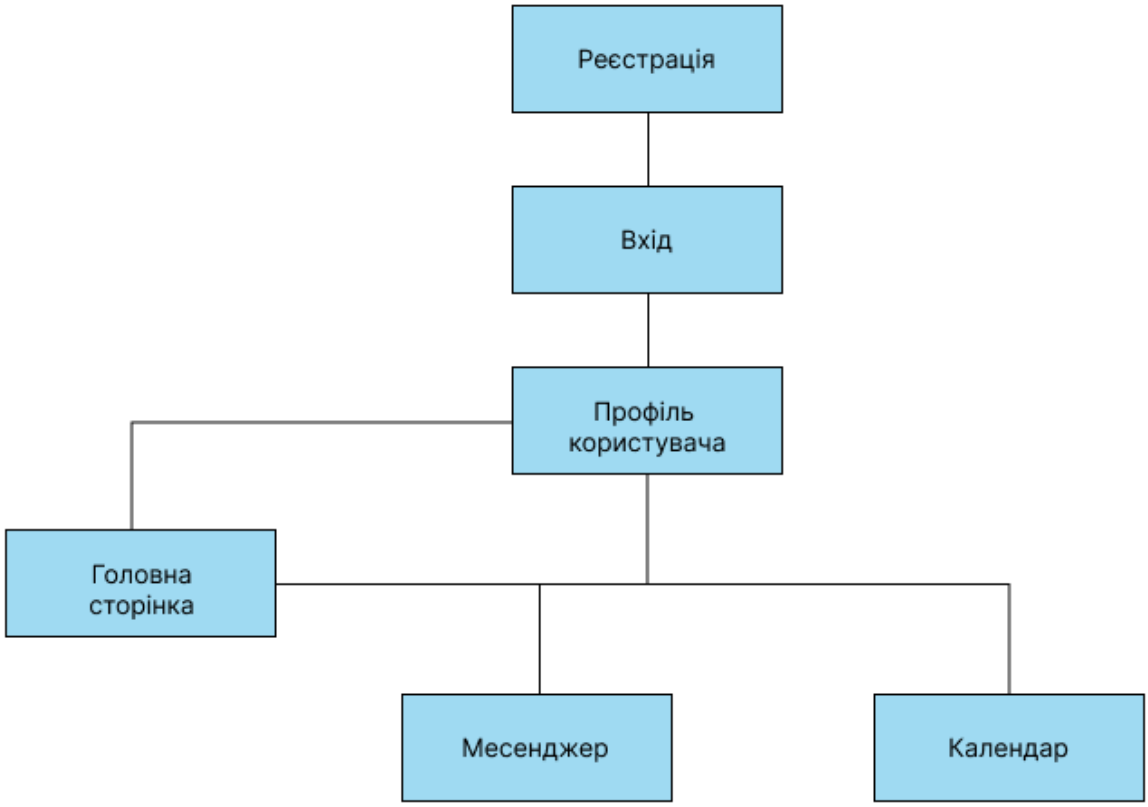

Рис. 2.4. Логічна структура сайту

Уся ця взаємодія між компонентами забезпечує інтегрованість і логічну цілісність соціальної мережі, дозволяючи користувачам легко перемикатися між різними функціями та розділами веб-додатку.

2.4.1.2. Сценарії діяльності користувачів

У розроблені системі існує два типи користувачів: звичайний користувач і адміністратор. Залежно від цих ролей відбувається розподіл доступу до компонентів системи та функціональних можливостей. Деякі з них були вже описані за допомогою декомпозиції та для кращого розуміння відмінності ролей та функціональних можливостей були розроблені сценарій перегляду подій зареєстрованим користувачем (рис. 2.5) та сценарій додавання подій в календар адміністратором (рис. 2.6).

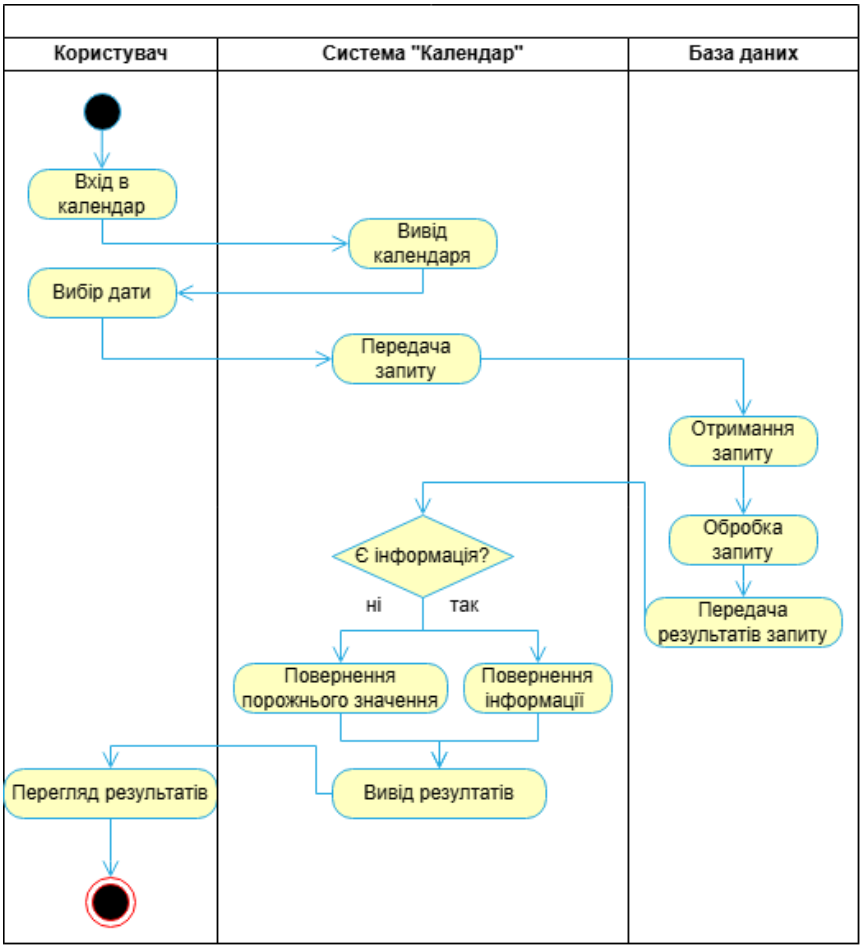

Рис. 2.5. Сценарій перегляду подій зареєстрованим користувачем

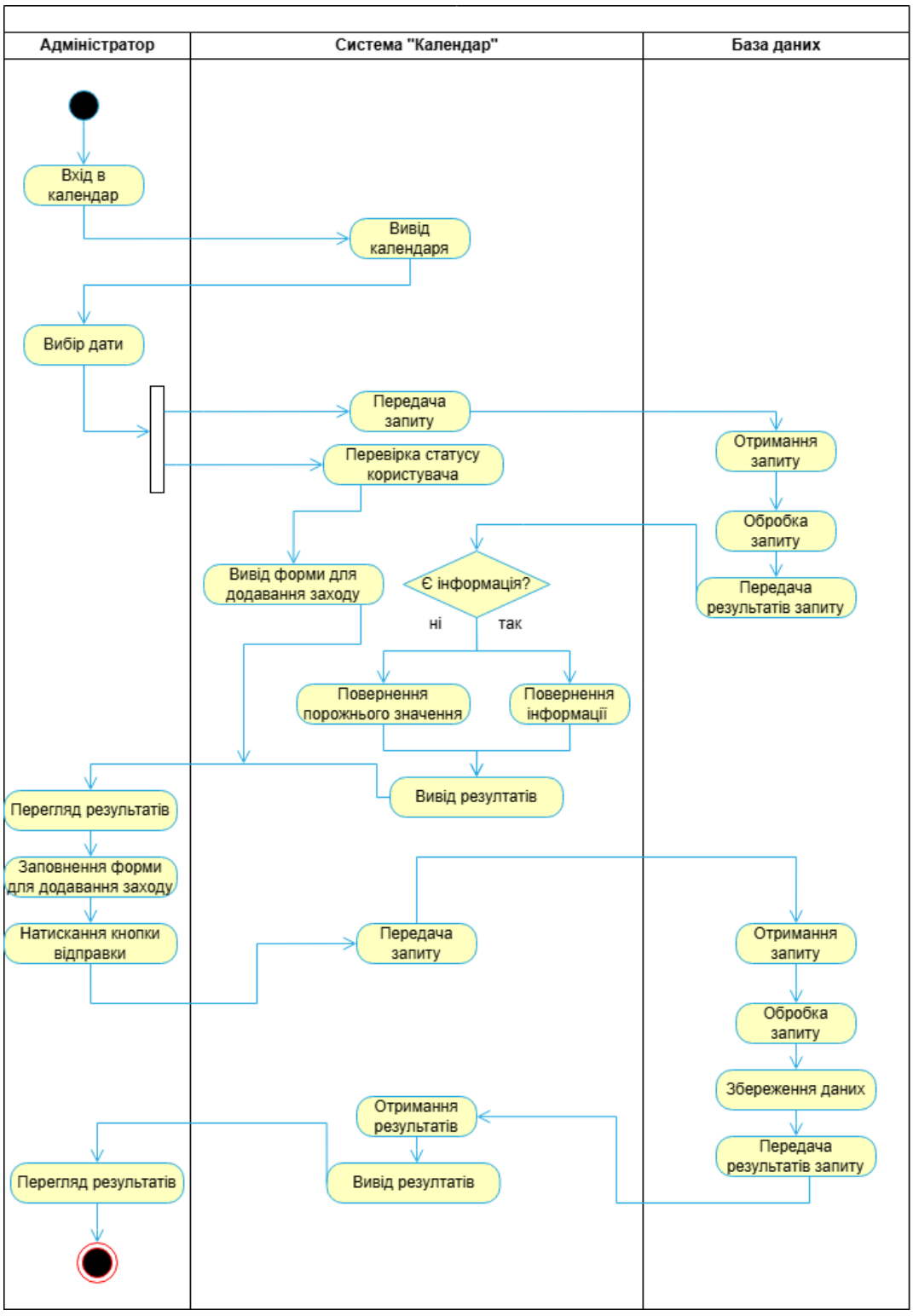

Рис. 2.6. Сценарій перегляду подій зареєстрованим користувачем

2.4.1.4. Файлова структура

Файлова структура веб-платформи складається з двох основних частин: серверної та клієнтської, й окремої папки «socket». Серверна частина

розташована в директорії під назвою «api», де знаходяїиться головний файл для запуску цієї частини додатка: «index.js». Тут також розташована папка «routes» із навігаційними файлами для визначення маршрутизації, папка «models» із файлами модулювання схем для відображення структур даних, папка «auth» із файлом налаштування автентифікації в Node.js застосунку за допомогою бібліотеки Passport.js. Також в «api» зберігається папка «public» із локальними файлами до проєкту і папка node_modules із додатково підключеними бібліотеками Node.js.

Взаємодія у серверній структурі відбувається наступним чином:

1) користувач надсилає HTTP-запит до сервера;

2) запит надходить до «index.js», який запускає сервер та визначає основні параметри конфігурації. Також він підключає необхідні середовища, які обробляють вхідні запити перед передачею їх до відповідних маршрутів;

3) відповідний файл у «routes» обробляє запит, викликаючи функції для використання моделей з папки «models» для подальшої взаємодії із базою даних. Якщо запит потребує автентифікації, файл з папки «auth» використовується для перевірки JWT автентифікації;

4) після обробки запиту та взаємодії з базою даних, сервер формує відповідь і надсилає її користувачеві;

5) статичні файли з «public» можуть бути включені у відповідь або доступні безпосередньо.

Клієнтська частина веб-платформи, розташована у директорії «client», орієнтована на використання бібліотеки React і відповідає вимогам цієї технології. Структура директорії включає основні папки: «node_modules», «public» і «src», кожна з яких виконує певні функціональні завдання.

У папці «src» розташовані всі модулі проєкту, які організовані в підпапки:

– «pages» – основна папка, де розміщені модулі, що реалізують сторінки інтерфейсу користувача, які відображаються у браузері, та модулі стилізації сторінок;

– «components» – ця папка містить окремі компоненти інтерфейсу, що використовуються для побудови складніших структур проєкту, та модулі стилізації компонентів;

– «context» – тут реалізовані контексти React, які дозволяють передавати дані через дерево компонентів без необхідності передачі пропсів особисто на кожному рівні;

– «public» – в цій папці знаходяться основні файли із загальними функціями, які використовуються для оформлення і стилізації компонентів та сторінок.

Файл запуску клієнтської частини розміщений безпосередньо в основній директорії «api».

Для налаштування і обробки всіх подій, пов'язаних з реальним часом комунікації у проєкті існує окрема папка «socket», головний файл якої налаштовує сервер на порту 8900 і дозволяє підключення з клієнтської частини, що працює на «http://localhost:3000». Файл «Messenger.js» на стороні клієнта встановлює з'єднання з сервером за допомогою Socket.IO і обробляє події. Файлова структура папки «socket» показана на рис. 2.7.

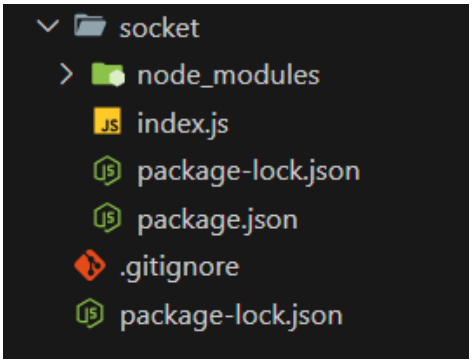

Рис. 2.7. Файлова структура папки «socket».

Розглянути детально повну структуру файлів серверної частини розроблюваного додатка можна на рис. 2.8, клієнтської сторони – на рис. 2.9.

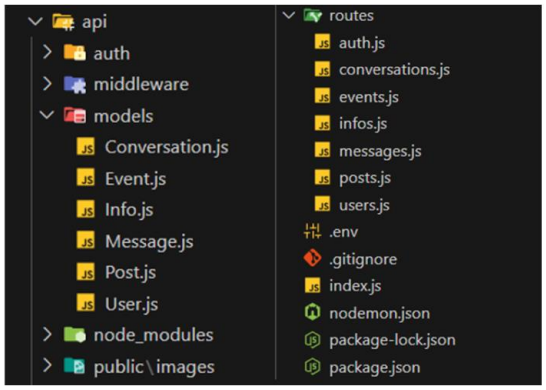

Рис. 2.8. Повна структура файлів серверної частини додатка

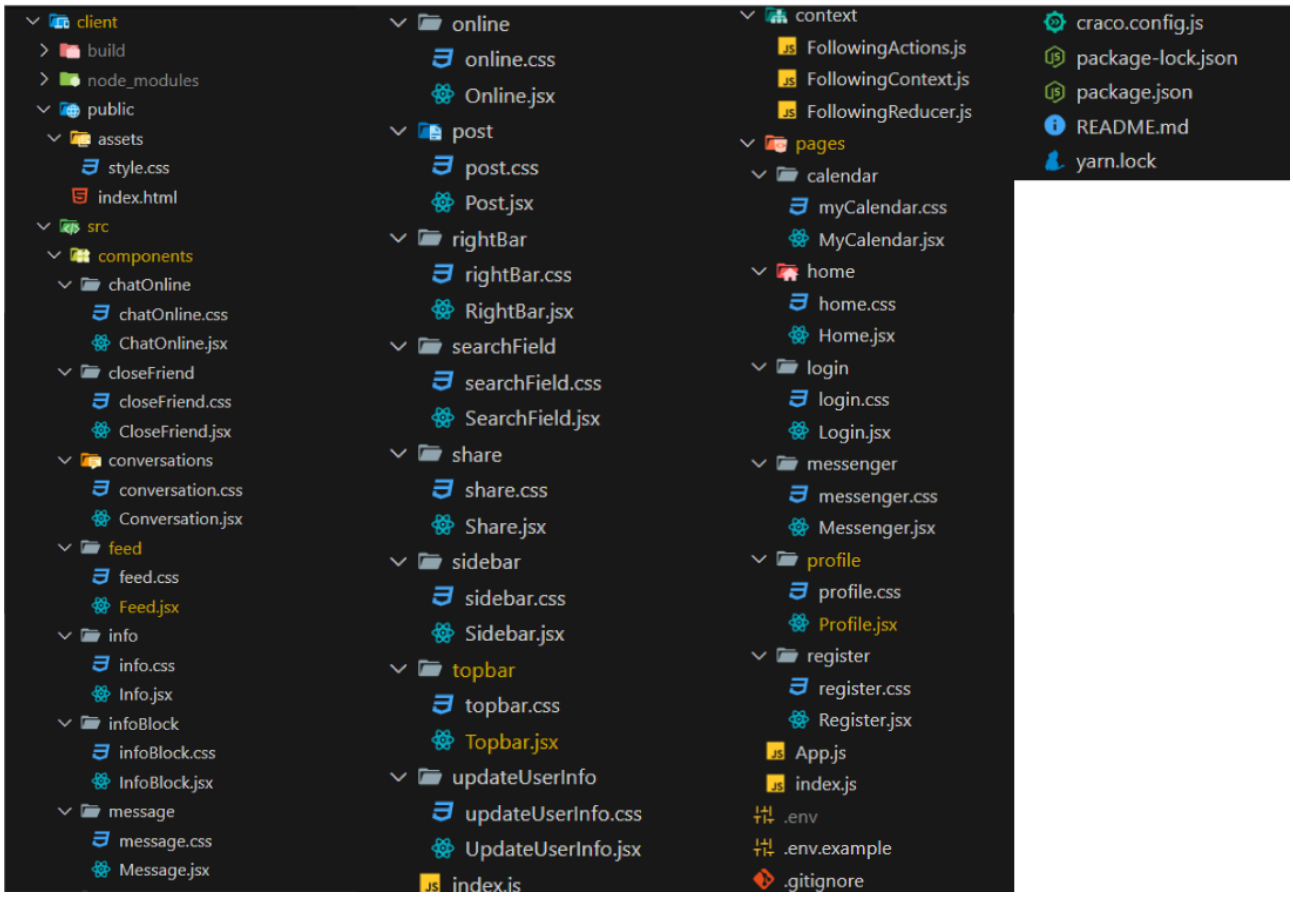

Рис. 2.9. Повна структура файлів клієнтської частини додатка

2.4.2. Структура бази даних

2.4.2.1. Концептуальна модель бази даних

Побудова правильної структури бази даних є критично важливою для системи соціальної мережі. Адже саме від БД залежить ефективне управління

даними користувачів та їхньою взаємодією. Таким чином, на основі аналізу функціональних вимог було виділено наступні основні елементи системи:

- користувач;
- друзі;
- пост;
- вподобайка;
- чат;
- повідомлення;
- подія.

Кожен з цих компонентів взаємодіє між собою, утворюючи складну систему зв’язків і залежностей. Узагальнено, логіку їх поєднання можна описати наступним чином: користувачі створюють профілі, додають один одного до списку друзів, обмінюються повідомленнями та створюють публікації (пости). Користувачі можуть ставити вподобайки на публікації один одного, а також переглядати календар подій та інформаційний блок.

Використання MongoDB як основи для зберігання даних дозволяє значно спростити структуру бази даних та скоротити кількість елементів. Вона гнучка та надає можливість зберігання складних вкладених структур у вигляді документів, зменшуючи кількість необхідних таблиць і спрощуючи запити до бази даних.

Отже, враховуючи можливості обраної бази даних, структура системи приймає наступний вигляд:

– користувач: особисті дані (ім'я, електронна пошта, дата народження), інформація для входу (пошта користувача, пароль), інформація про користувача (фото профілю, місце проживання, факультет, група, інтереси), перелік підписників, перелік підписок, фото обкладинки профілю;

– пост: публікації користувачів, контент (текст, зображення), час створення, вподобайки (перелік id користувачів, що залишили реакцію), розгалуження постів від звичайного користувача та від адміністратора;

– чат: приватна переписка між користувачами;

– повідомлення: приватні повідомлення між користувачами, час відправлення;

– подія: створені події, дата і час, місце проведення.

Для кращої візуалізації вигляду структури бази даних після спрощення було розроблено концептуальну модель, яка зображена на рис. 2.10.

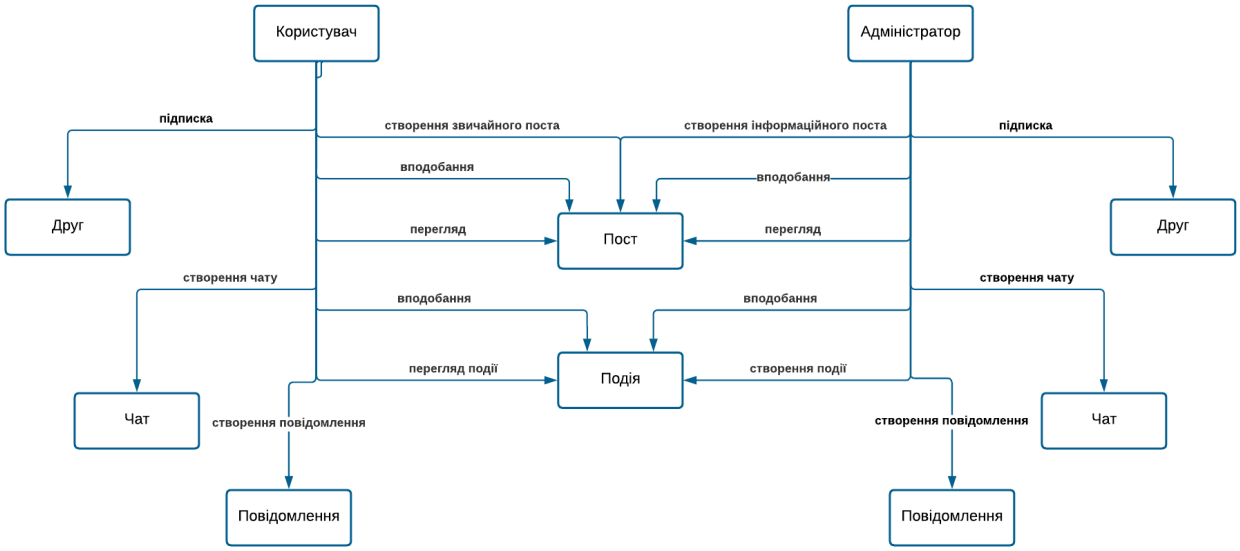

Рис. 2.10. Концептуальна модель структури бази даних

Опис взаємодії:

– користувачі реєструються та створюють профілі, де вказують особисті дані та інформацію про себе;

– користувачі можуть додавати один одного до списку друзів;

– чати забезпечують створення приватної комунікації між двома користувачами;

– повідомлення забезпечують створення користувацького повідомлення;

– пости створюються користувачами та можуть містити текст або зображення. Вони відображаються у стрічці новин друзів. Для користувача із статусом адміністратор створення поста означає створення інформаційного елементу. Будь-який користувач може залишити вподобайку на пості;

– події дозволяють користувачам переглядати заходи, що відбуватимуться в університет. Лише користувач зі статусом адміністратор може створювати події.

Ця система дозволяє забезпечити всебічне управління соціальною активністю користувачів, сприяє покращенню взаємодії між ними та забезпечує зручний доступ до інформації і комунікаційних інструментів.

2.4.2.2. Логічна модель бази даних

Для визначення характеристики об’єктів та зв’язків між ними без врахування аспекту їх зберігання і на основі концептуальної моделі було побудовано логічну модель, яка представлена на рис. 2.11.

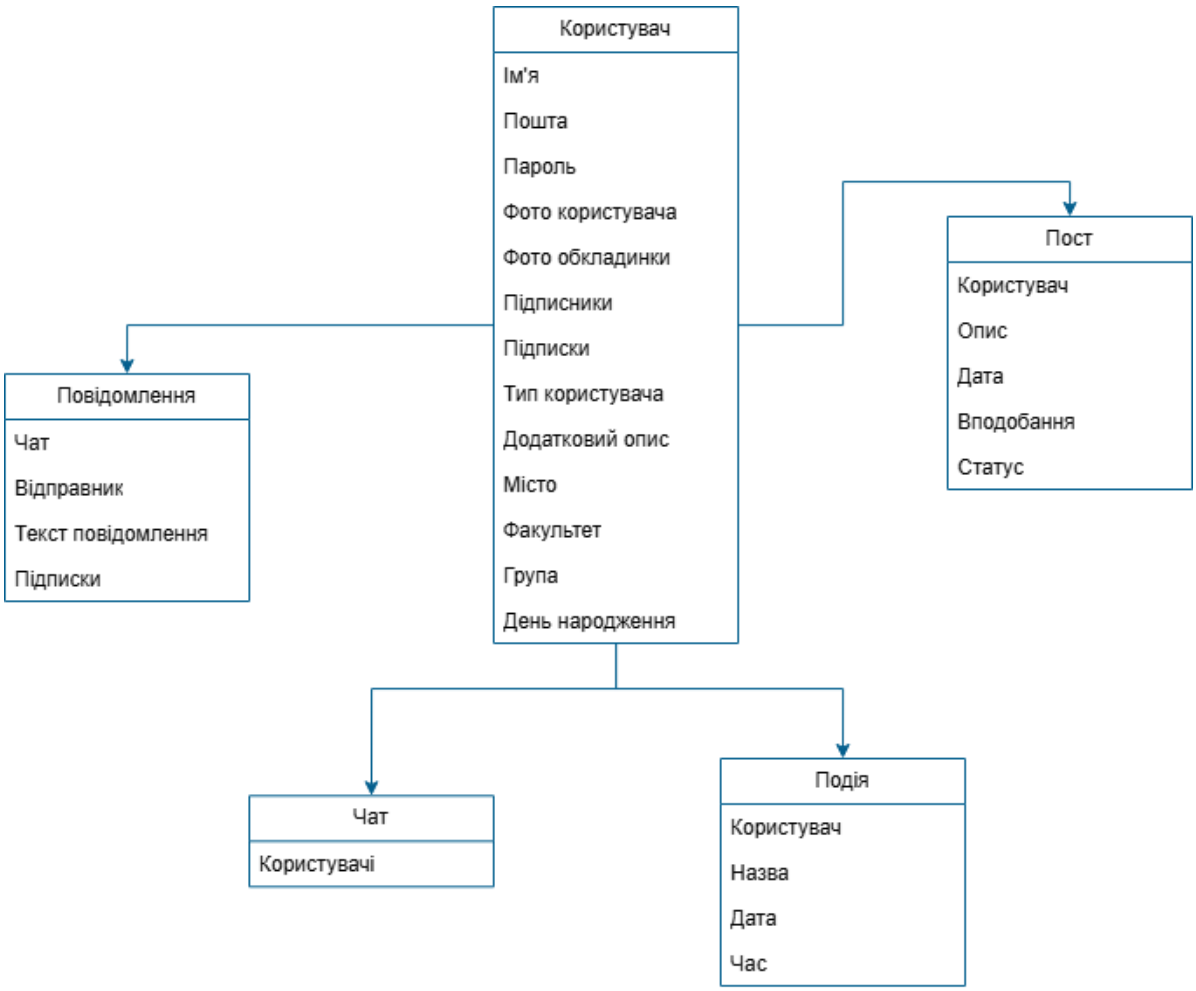

Рис. 2.11. Логічна модель бази даних

2.4.2.3. Фізична модель бази даних

Розробка фізичної моделі бази даних допомогла визначити типи створюваних даних та розміри. Структурування компонентів моделі показано у таблиці 2.2.

Таблиця 2.2

Компоненти фізичної моделі

Колекція	Документ	Тип
Users	id	String
	username	String
	email	String
	password	String
	profilePicture	String
	coverPicture	String
	followers	Array
	followings	Array
	isAdmin	Boolean
	desc	String
	city	String
	faculty	String
	group	String
	birthday	Date
Posts	id	String
	desc	String
	img	String
	likes	Array
	isInfo	Boolean
Events	id	String

Продовж. табл. 2.2

Колекція	Документ	Тип
	userId	String
	name	String
	eventDate	Date
	eventTime	String
Conversations	id	String
	members	Array
Messages	id	String
	conversationId	String
	senderId	String
	text	String

У результаті обробки усіх компонентів отримуємо фізичну модель бази даних, що зображено на рис. 2.12.

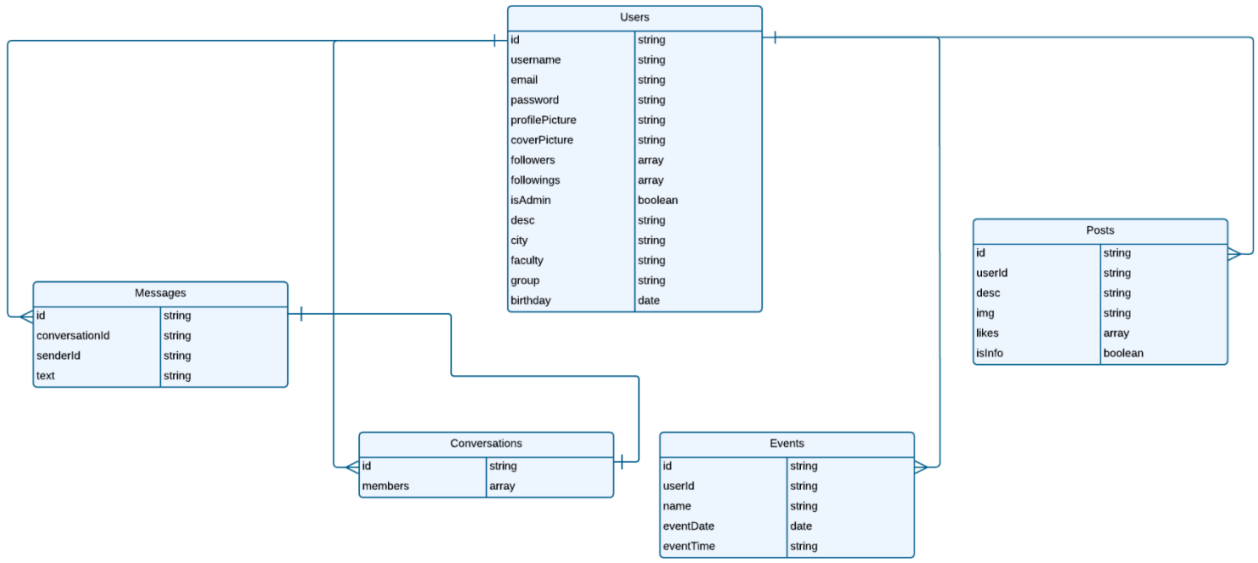

Рис. 2.12. Фізична модель бази даних

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Основні вхідні дані системи надходять виключно через клієнтський інтерфейс, де користувачі безпосередньо вводять необхідну інформацію. Введення даних здійснюється за допомогою спеціально розроблених форм зі спеціальними полями, які спрямовані на мінімізацію помилок користувачів при заповненні. Варто зазначити, що всі дані вводяться вручну без можливості вибору з попередньо визначених варіантів.

Вхідна інформація включає, але не обмежується, особистими даними користувачів, записами про події, повідомленнями та іншими видами даних, що вимагаються системою. Введені дані одразу зберігаються у базі даних, забезпечуючи оперативну обробку та доступність інформації для подальшого використання. Такий підхід гарантує високу гнучкість і точність введеної інформації, оскільки кожен запис створюється безпосередньо користувачем у реальному часі.

Загалом у кваліфікаційній роботі представлені такі типи вхідних даних: текстові, дати та файли. Текстові дані представлені українською та англійською мовами і можуть мати різні обмеження щодо кількості символів залежно від їхнього призначення та типу. Тип даних «дати» зберігається у форматі YYYY-mm-ddTHH:MM:ss. Файли зберігаються та обробляються за допомогою локального сховища на стороні сервера, при цьому до бази даних потрапляє лише назва файлу. Дозволено завантажувати файли всіх типів зображень.

Вихідні дані в програмі представлені загальною інформацією про користувача, текстом та картинками постів або інформаційних блоків, довідковими даними про події в календарі та повідомленнями у чатах користувачів.

Вихідні дані потрапляють до користувача в обробленій та готовій формі для зручного сприйняття. Надання даних у такому вигляді спрощує взаємодію користувача з системою та дозволяє уникнути додаткових кроків обробки або

інтерпретації, забезпечуючи оптимальне використання інформаційних ресурсів системи.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Повноцінна розробка та тестування соціальної мережі здійснювалось на технічних засобах із вказаною конфігурацією:

- диск SSD: HFM512GDJTNG-8310A;
- ЦП: Intel® Core™ i5-10210U CPU @ 1.60GHz;
- ОЗП: 8Гб;
- відео карта: Intel® UHD Graphics;
- операційна система: 64-розрядна Windows 10;
- дисплей:
  - частота оновлення: 60,030Гц;
  - розрядність: 8біт;
  - формат кольору: RGB;
  - роздільна здатність: 1920x1080;
  - співвідношення сторін: 16:9;
- стандартні засоби периферії.

2.6.2. Використані програмні засоби

Розробка проєкту соціальної мережі потребувала наступних програмних засобів:

– VisualStudioCode – безкоштовний, відкритий редактор коду, розроблений компанією Microsoft. Він підтримує велику кількість мов програмування і платформ, пропонуючи інструменти для розробки, налагодження, автоматизації завдань, а також інтеграцію з системами контролю версій, такими як Git.

Основними перевагами Visual Studio Code є його висока продуктивність, велика кількість розширень, потужний редактор з підтримкою інтелектуальних підказок, синтаксичної перевірки, автозавершення коду та зручний інтерфейс;

– Postman – популярний інструмент для розробки та тестування API (інтерфейсів програмування застосунків). Він надає користувачам можливість створювати, тестувати та керувати API-запитами зручним і ефективним способом;

– MongoDB – зручна у використанні база даних, яка використовує документо-орієнтовану модель даних. Вона розроблена для зберігання великих обсягів даних, забезпечення високої продуктивності та гнучкості у роботі з даними;

– Figma – хмарний інструмент, призначений для створення та тестування дизайну веб-сайтів, мобільних додатків та інших цифрових продуктів. Основні функції Figma включають розробку інтерфейсів користувача (UI), створення векторної графіки, побудову інтерактивних прототипів та організацію спільної роботи над проектами;

– Lucidchart – веб-додаток для створення діаграм та схем, який пропонує потужні інструменти для візуалізації процесів, систем і організаційних структур. Це інтуїтивно зрозумілий і гнучкий інструмент, який широко використовується для планування та документування;

– GitHub – веб-сервіс для розміщення та спільної розробки програмного забезпечення з використанням системи керування версіями Git. На GitHub користувачі можуть створювати репозиторії (проєкти), завантажувати в них свій код, відстежувати зміни, пропонувати зміни в проєкти інших користувачів через пул-реквести, співпрацювати з іншими розробниками, відкривати та вирішувати проблеми (issues) та багато іншого.

2.6.3. Виклик та завантаження програми

Виклик та завантаження програми виконується наступним чином:

– починаючи з підготовки інфраструктури, важливо забезпечити готовність і доступність необхідного технічного обладнання, яке надасть безперебійну роботу серверів застосунку і бази даних;

– наступним кроком необхідно встановити середовище виконання Node.js на сервері застосунку;

– для зберігання та організації даних, встановлюється нереляційна база даних MongoDB;

– після встановлення середовища і бази даних, виконуються скрипти для налаштування бази даних з урахуванням потреб конкретного застосунку;

– далі необхідно завантажити всі файли і код програми на цільовий сервер і забезпечити його готовність до роботи, включаючи налаштування оточення і підключення до бази даних. Цей процес приведе до готовності програмного застосунку для прийняття запитів від користувачів і ефективного управління даними.

2.6.4. Опис інтерфейсу користувача

2.6.4.1. Розробка графічних складових

У задачі була поставлена мета розробки соціальної мережі для студентів Національного технічного університету «Дніпровська політехніка». Так як університет має свої корпоративні кольори та логотип, було вирішено додержуватись використання цих елементів у дизайні соціальної мережі. Крім того, Рада студентів НТУ ДП має свій логотип, який також було враховано.

Таким чином, з урахуванням усіх деталей було розроблено дизайн веб-додатку.

Основним елементом графічної розробки став логотип сайту, адже необхідно було поєднати назву розроблюваної соціальної мережі, логотипи університету, Ради студентів та корпоративні кольори. Крім того, було розглянуто існуючі варіанти логотипів згідно тематики.

Інструментом для виконання дизайну логотипу було обрано веб-додаток Figma.

У результаті поєднання усіх підготовчих робіт було отримано набір графічних зображень, який показано на рис. 2.13.

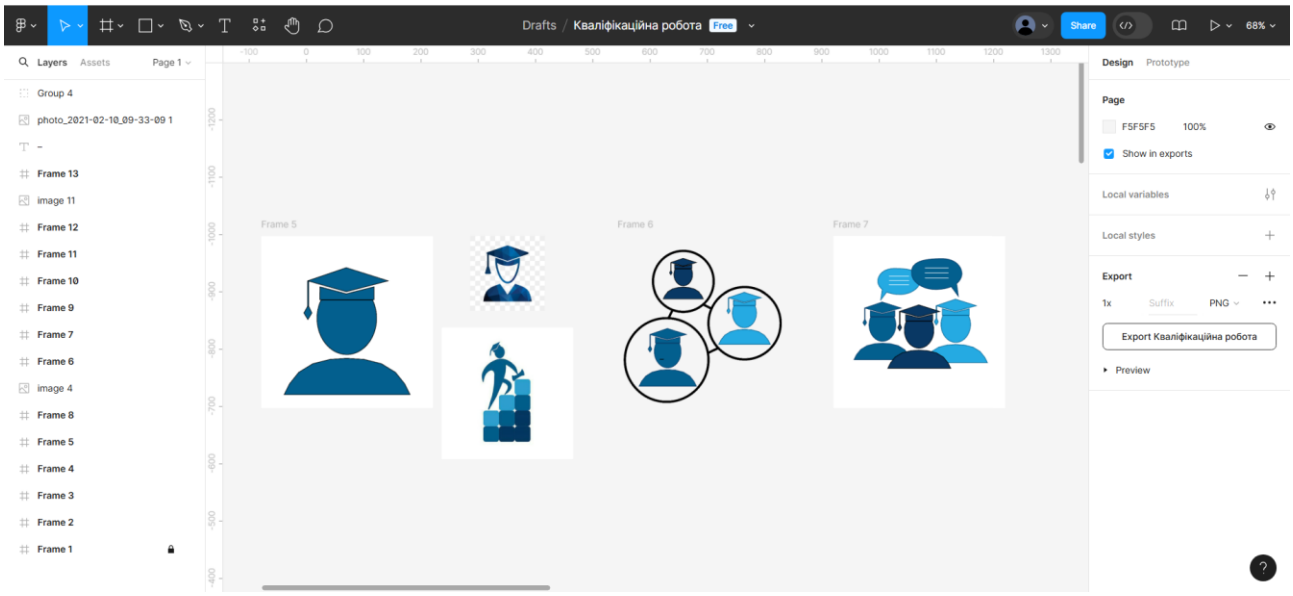

Рис. 2.13. Дизайн графічних зображень сайту

Варіант із студентами в колах, об’єднаних лініями зв’язку, виявився тим, що більше підходить до оформлення сайту, а окремий силует студента було обрано для заповнення фото профіля у разі його відсутності у базі даних.

2.6.4.2. Опис інтерфейсу користувача

При завантаженні додатку користувач одразу потрапляє на сторінку автентифікації (рис. 2.14). Якщо користувач вже має аккаунт у веб-ресурсі, він може ввести поштову адресу та пароль у відповідні поля, натиснути кнопку «Log in» і таким чином потрапити до сторінки свого профілю.

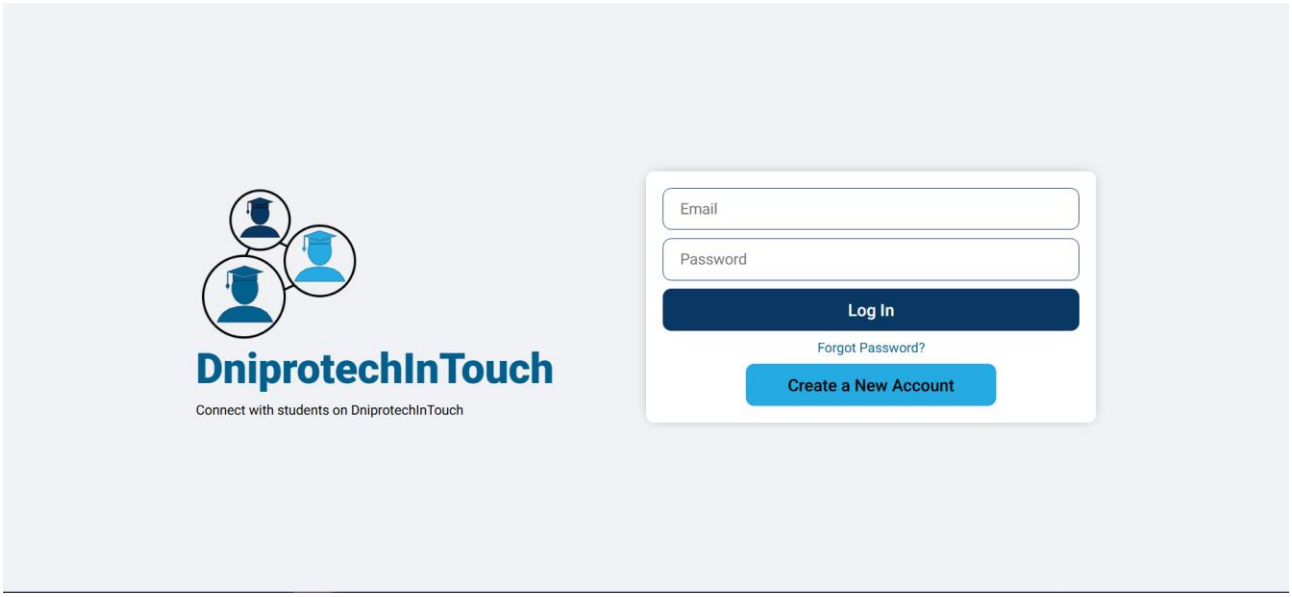

Рис. 2.14. Сторінка автентифікації

Якщо користувач не має створеного профілю, він може перейти до сторінки реєстрації натиснувши кнопку Create a New Account. На сторінці є форма для заповнення, користувач може заповнити її за наступним прикладом з рис. 2.15.

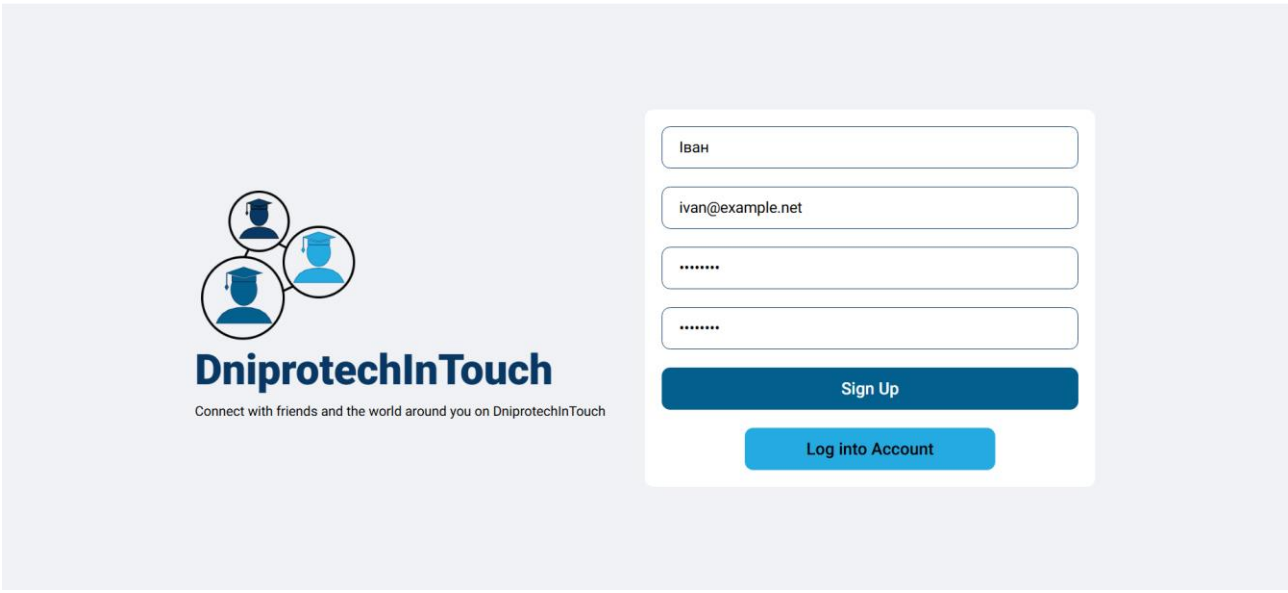

Рис. 2.15. Приклад вводу даних для реєстрації

Варто зазначити, що на цій сторінці є два поля для вводу пароля, які показано на рис. 2.16. Вони створені для того, щоб переконатися у вірності введеного паролю.

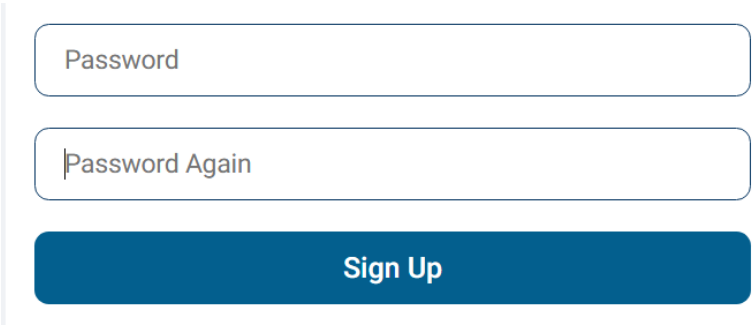

Рис. 2.16. Поля для вводу пароля

Після реєстрації користувача переносить на сторінку логіну для автентифікації, яку можна виконати як показано на рис. 2.17.

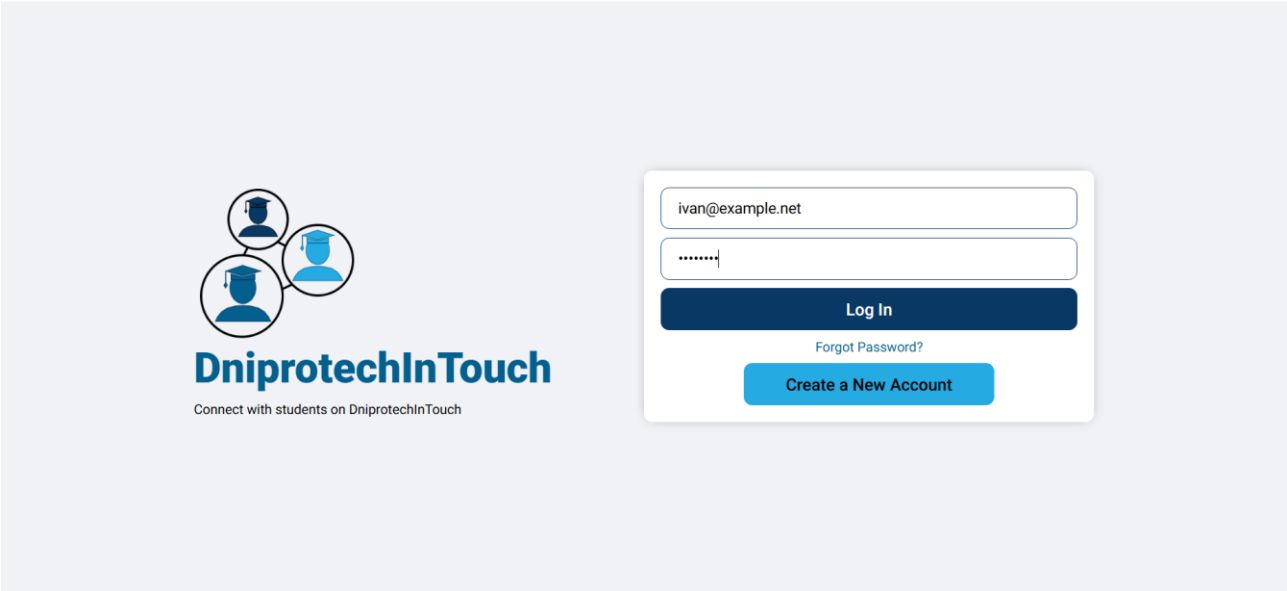

Рис. 2.17. Приклад автентифікації

Нарешті, після натискання кнопки «Log In» користувач потрапляє на сторінку свого профілю, яка має вигляд зображений на рис. 2.18.

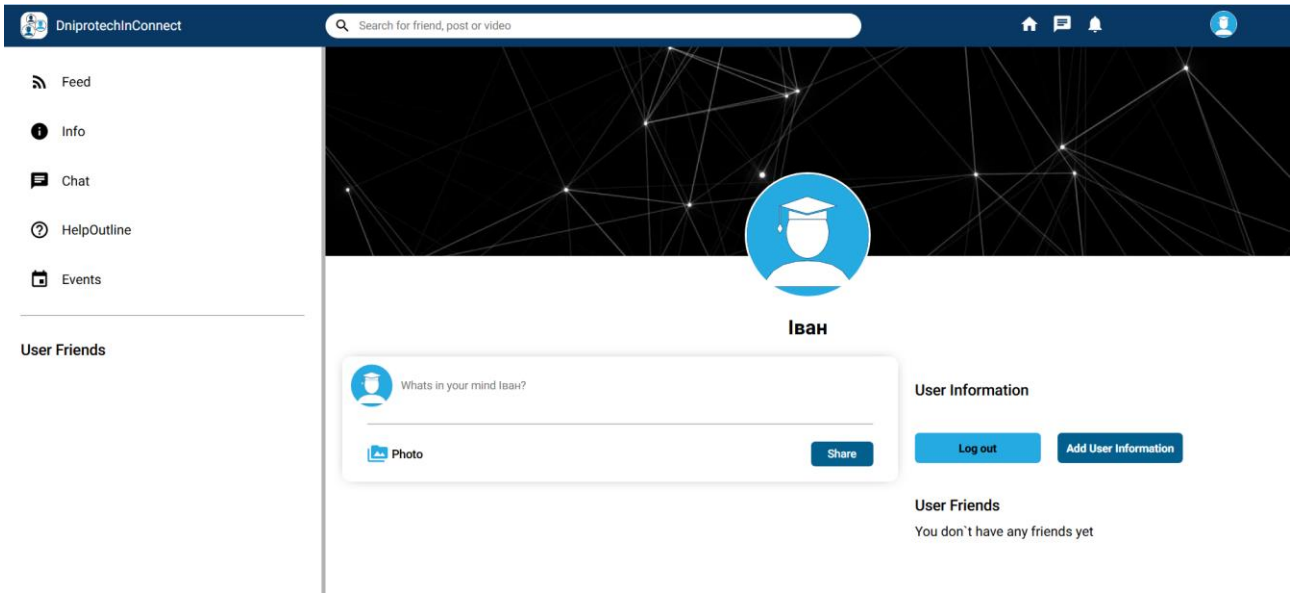

Рис. 2.18. Приклад профілю користувача

На своїй сторінці користувач може додати детальну інформацію про себе, натиснувши кнопку «Add User Information». Натискання цієї кнопки викличе компонент із формою для заповнення, вигляд якої показано на рис. 2.19.

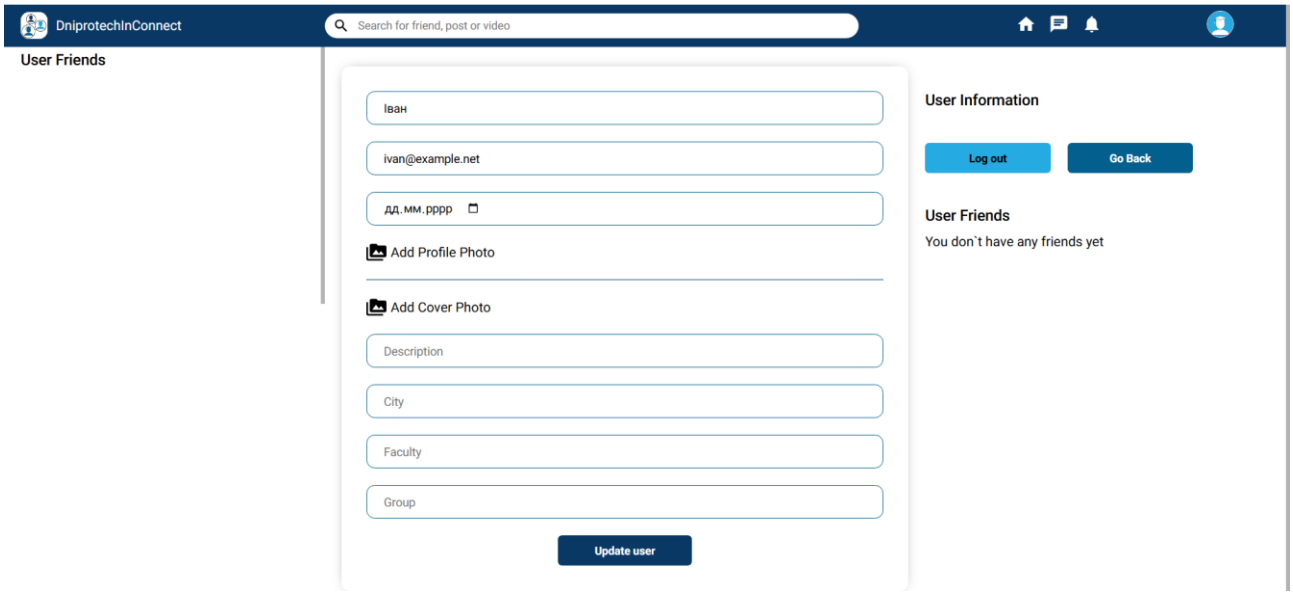

Рис. 2.19. Форма для додавання чи зміни інформації про користувача

Тут користувачу надана можливість вводу дати народження за допомогою влаштованої функції типу «дата», як показано на рис. 2.20.

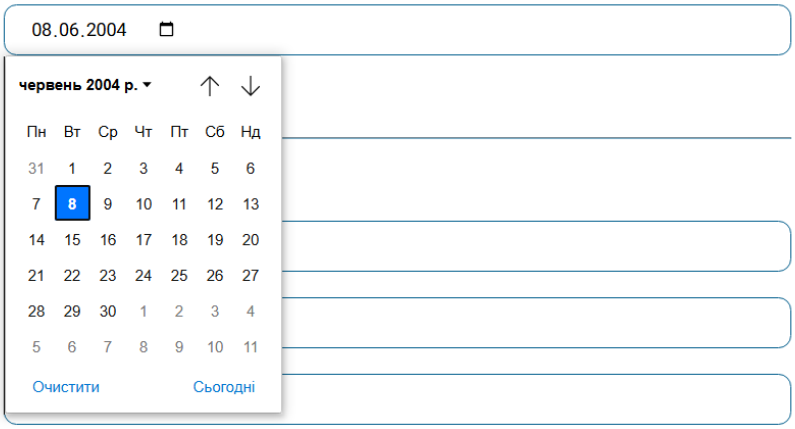

Рис. 2.20. Заповнення дати народження користувача

Додавання фото профіля та обкладинки профіля відбувається наступним чином: користувач натискає відповідну кнопку для додавання зображення, відкривається віконце, де надається можливість вибору зображення з пристрою користувача (рис.2.21).

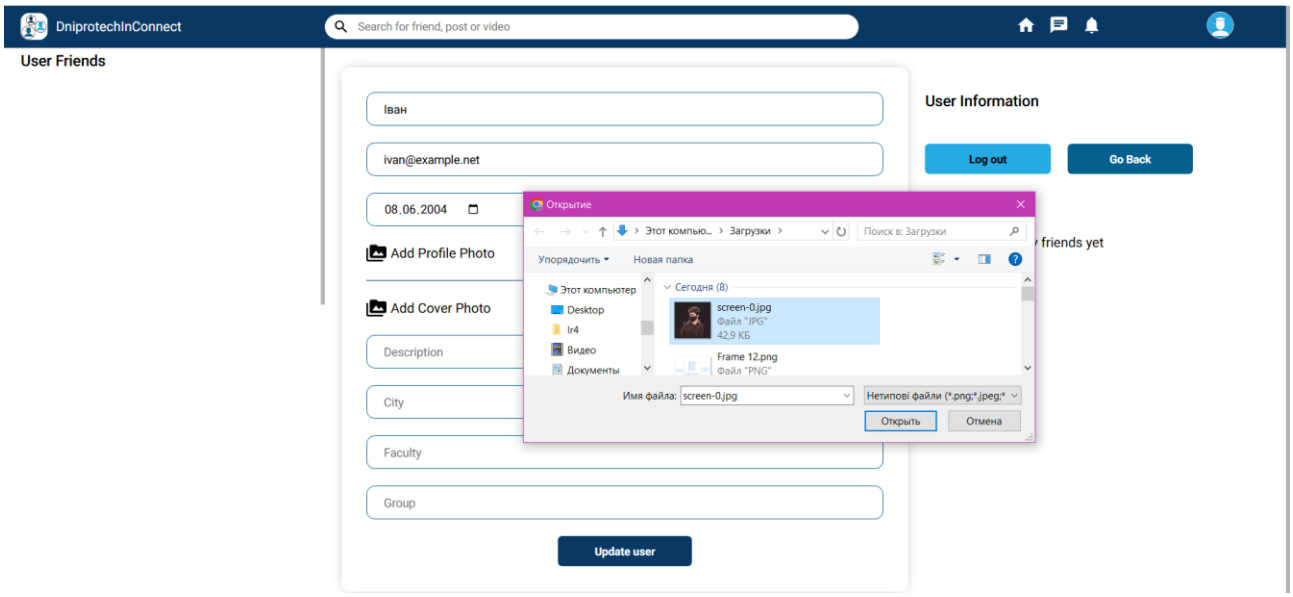

Рис. 2.21. Додавання фото профіля

Технічно додавання фото користувача та обкладинки профіля зображено на рис. 2.22, 2.23. Після виконання вибору фото профіля та фото обкладинки користувач отримує зображення у тому виді, у якому вони будуть розміщені на сторінці профілю, це показано на рис. 2.24.

```
if (profilePictureFile) {
    const data = new FormData();
    const fileName = Date.now() + profilePictureFile.name;
    data.append("name", fileName);
    data.append("file", profilePictureFile);
    userData.profilePicture = fileName;
    try {
        const config = {
            headers: { 'Authorization': authHeader }
        }
        await axios.post("/upload", data, config);
    } catch (err) {
        console.error('Error uploading profile picture:', err);
    }
}
```

Рис. 2.22. Технічне додавання фото користувача профіля

```
if (coverPictureFile) {
    const data = new FormData();
    const fileName = Date.now() + coverPictureFile.name;
    data.append("name", fileName);
    data.append("file", coverPictureFile);
    userData.coverPicture = fileName;
    try {
        await axios.post("/upload", data);
    } catch (err) {
        console.error('Error uploading cover picture:', err);
    }
}
```

Рис. 2.23. Технічне додавання фото обкладинки профіля

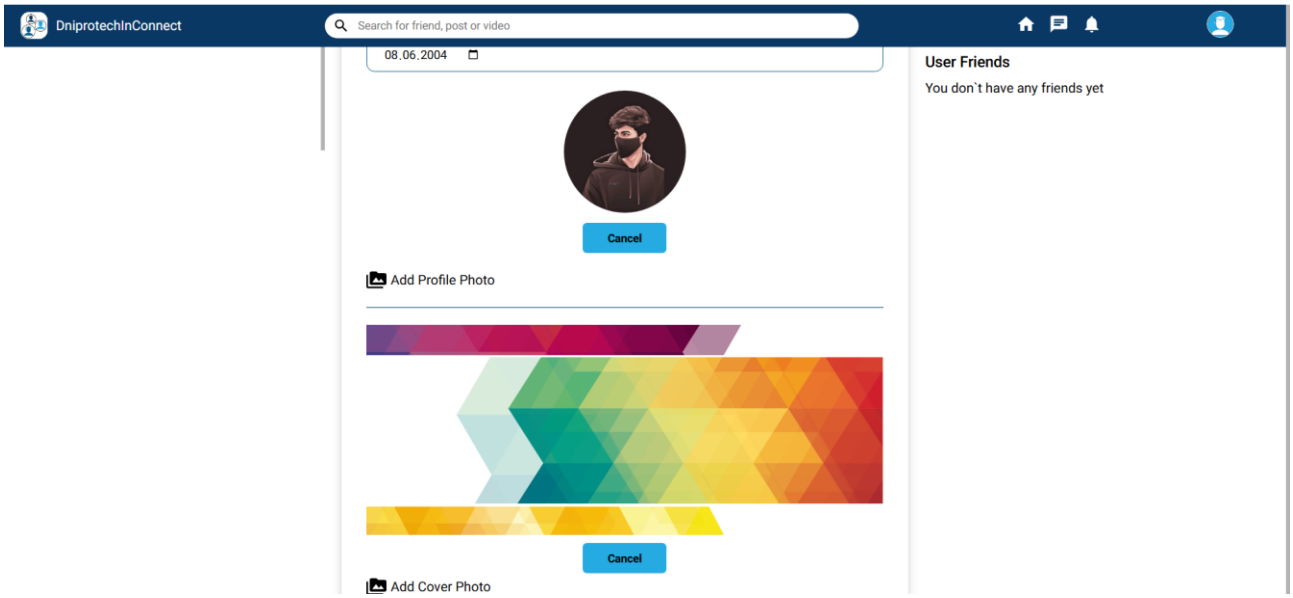

Рис. 2.24. Вигляд обраних зображень на сторінці користувача

Варто зазначити, що кнопка «Cancel» прибере обране зображення. Далі користувач може заповнити інші поля, як показано на рис. 2.25.

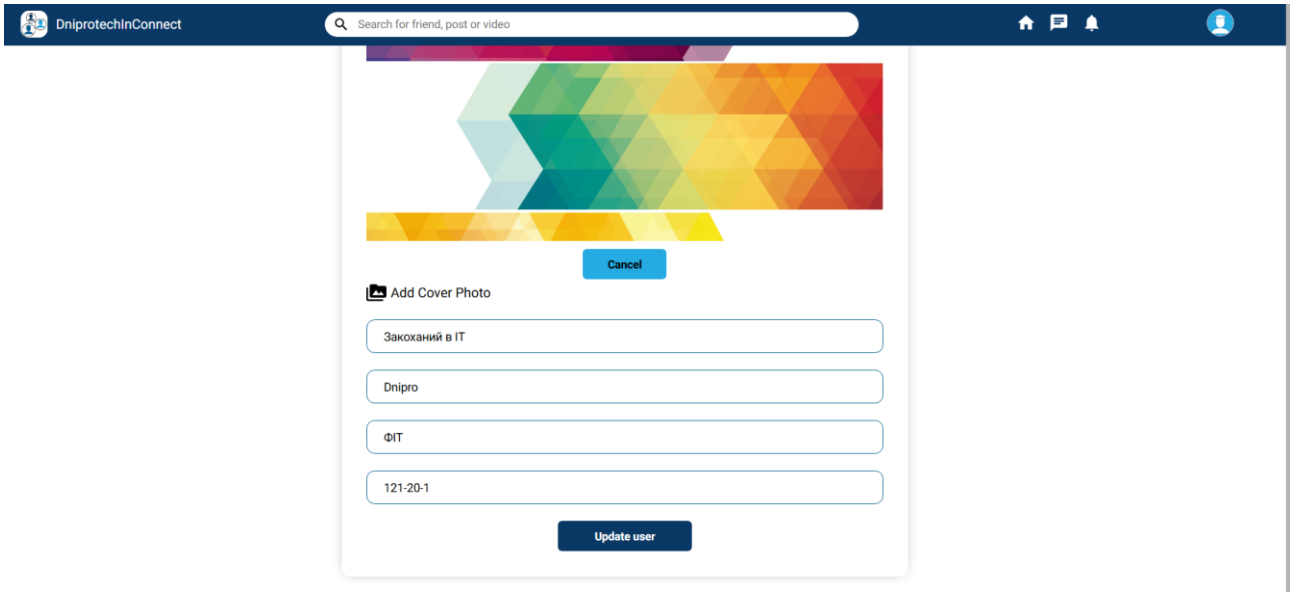

Рис. 2.25. Заповнення інших полів форми

Натискання кнопки «Update User» запустить функцію додавання інформації в базу даних та поверне користувача у сторінку профілю, де одразу будуть показані усі оновлені дані. Це можна побачити на рис. 2.26.

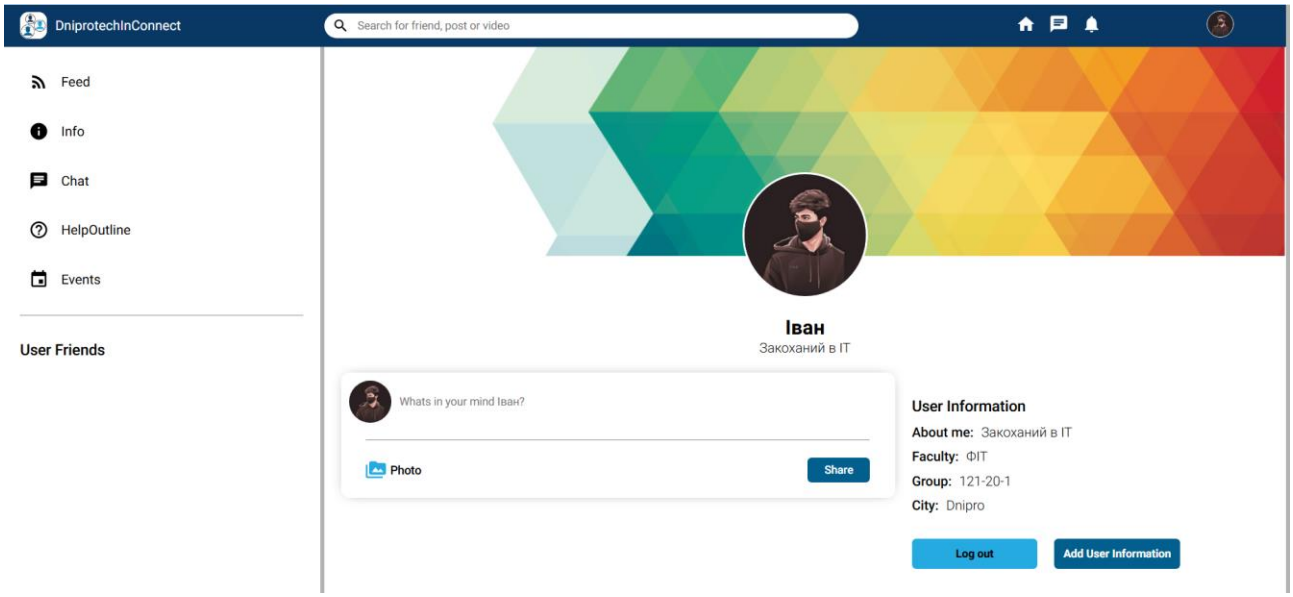

Рис. 2.26. Результати додавання нових даних користувача

Також на своїй сторінці користувач має можливість створювати нові публікації. Необхідно лише ввести відповідний текст та додати фото за тим принципом додавання, що і фото профіля та обкладинки. Приклад заповнення форми для додавання публікації показаний на рис. 2.27.

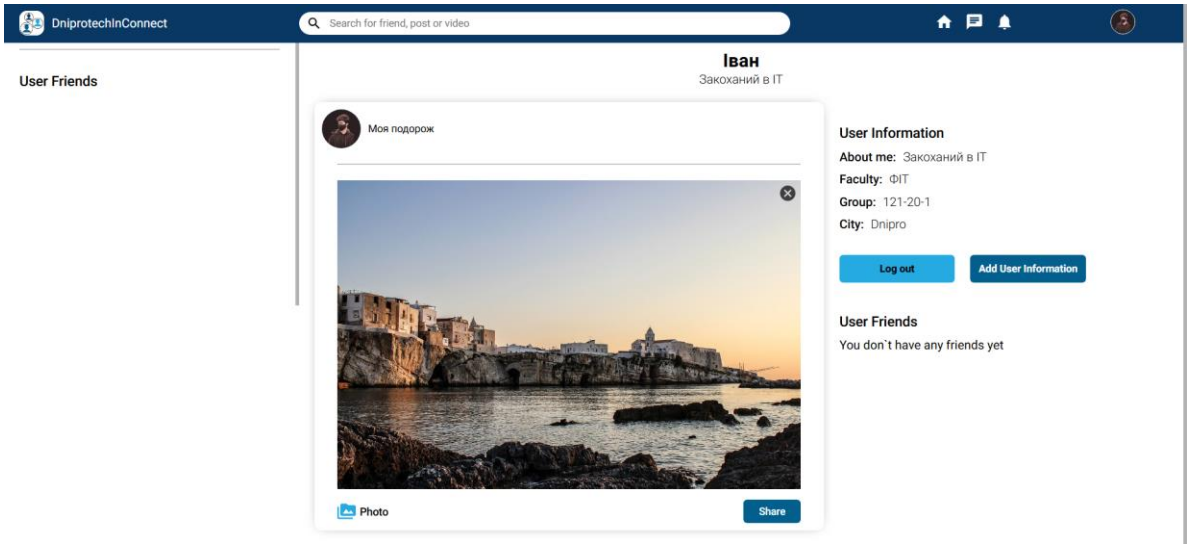

Рис. 2.27. Приклад заповнення форми додавання публікації.

Іконка «хрестик» дозволяє відмінити додавання фото у форму.

Натискання кнопки «Share» запускає процес додавання даних публікації в базу даних та формування графічного вигляду блоку публікації на сторінках. Результат додавання публікації показано на рис. 2.28.

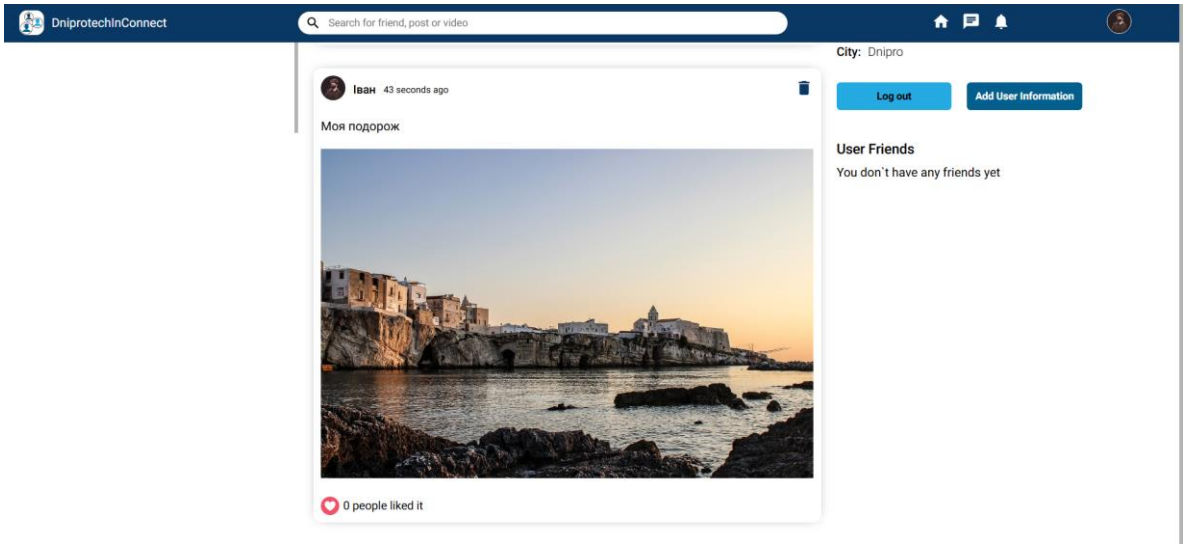

Рис. 2.28. Результат додавання публікації на сторінку

Технічно додавання публікації зображено на рис. 2.29.

```
const newPost = {
    userId: user._id,
    desc: desc.current.value,
    isAdmin: user.isAdmin,
}


if (file) {
    const data = new FormData();
    const fileName = Date.now() + file.name;
    data.append("name", fileName);
    data.append("file", file);
    newPost.img = fileName;
    try {
        const config = {
            headers: { 'Authorization': authHeader }
        }
        await axios.post("/upload", data, config);
    } catch (err) {
        console.error(err);
    }
}

try {
    const config = {
        headers: { 'Authorization': authHeader }
    }
    await axios.post(`/posts`, newPost, config);
    window.location.reload();
} catch (err) {
    console.error(err);
}
```

Рис. 2.29. Додавання публікації

Іконка у вигляді кошику для сміття у правому верхньому куті блоку дозволяє видалити публікації.

Для того, щоб знайти інших користувачів, достатньо ввести ім’я у відповідне поле та натиснути пошук, наприклад, як на рис. 2.30.

Рис. 2.30. Приклад вводу імені користувача у поле пошуку та натискання кнопки пошуку.

Коли користувач натисне на поле, що з’явилось після натискання кнопки пошуку, він перейде у відповідний профіль користувача, якого шукав, що показано на рис. 2.31.

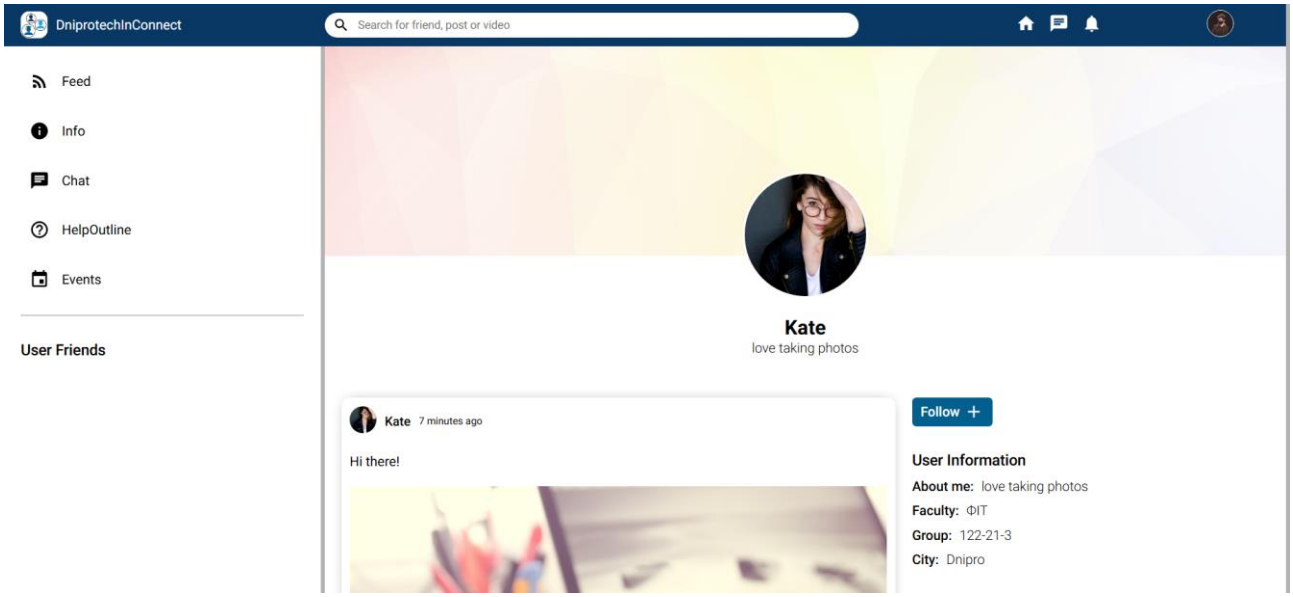

Рис. 2.31. Перехід на профіль користувача із пошуку

Тепер можна підписатися на профіль знайденого користувача натиснувши кнопку «Follow». Вона змінить своє значення як показано на рис. 2.32.

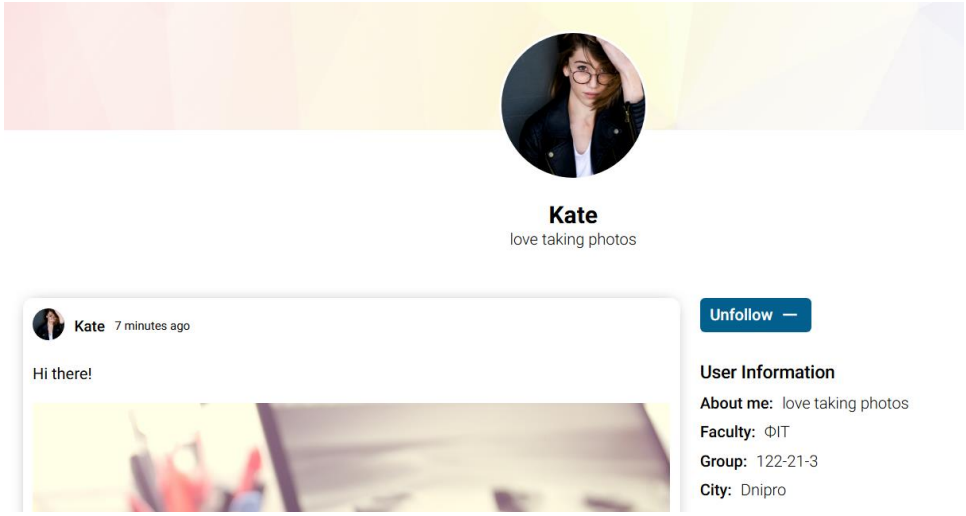

Рис. 2.32. Зміна значення кнопки підписки

Відповідно тепер користувач має змогу відписатися.

Коли повернемося на сторінку користувача, побачимо, що у блоці друзів з’явилось фото та ім’я користувача (рис. 2.33), на якого підписались до цього.

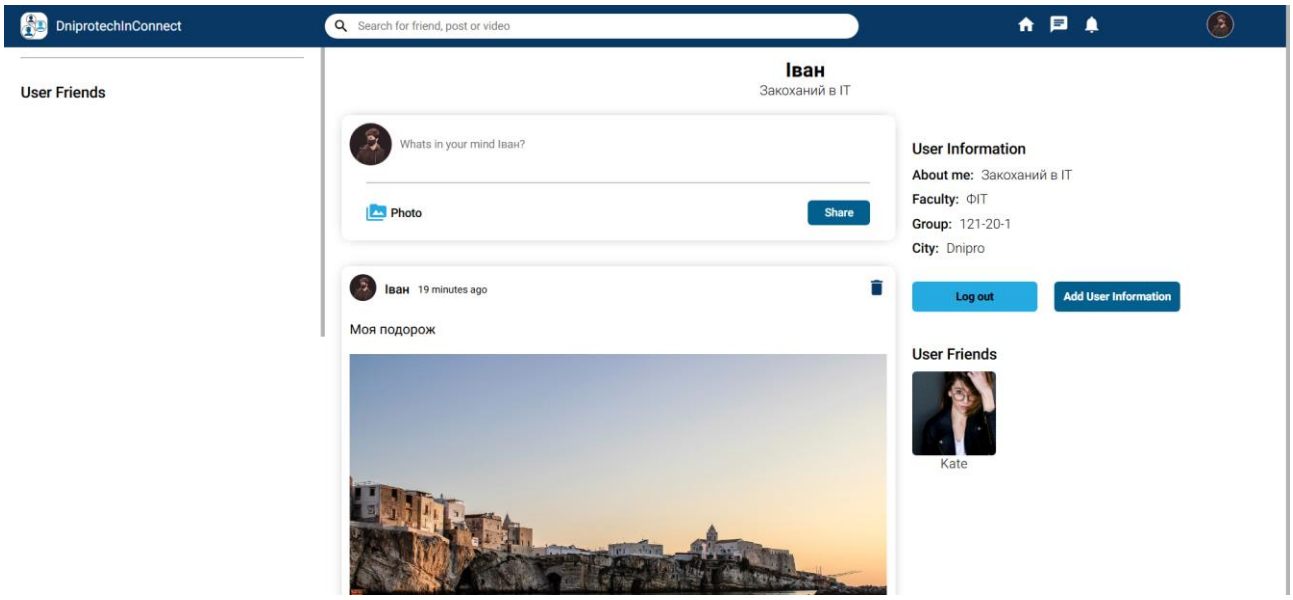

Рис. 2.33. Оновлення блоку друзів

Кожен користувач має можливість відкрити головну сторінку соціальної мережі. Це він може зробити декількома шляхами: натиснувши логотип сайту або іконку «хатинка». Результат натискання обох посилань показано на рис. 2.34.

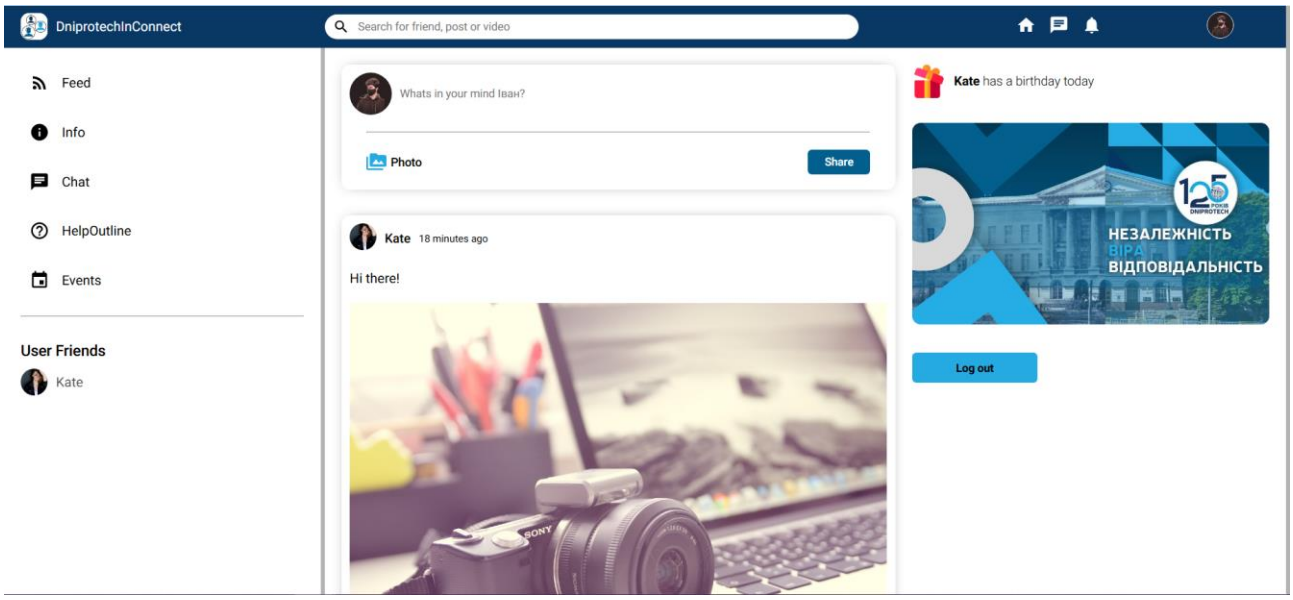

Рис. 2.34. Вигляд головної сторінки соціальної мережі

На цій сторінці користувач може створювати та видаляти свої публікації, бачити їх, бачити пости друзів, залишати вподобайки на публікаціях (рис.2.35). Також користувачеві показано на сьогодні наявність дня народження у друга, якщо воно є.

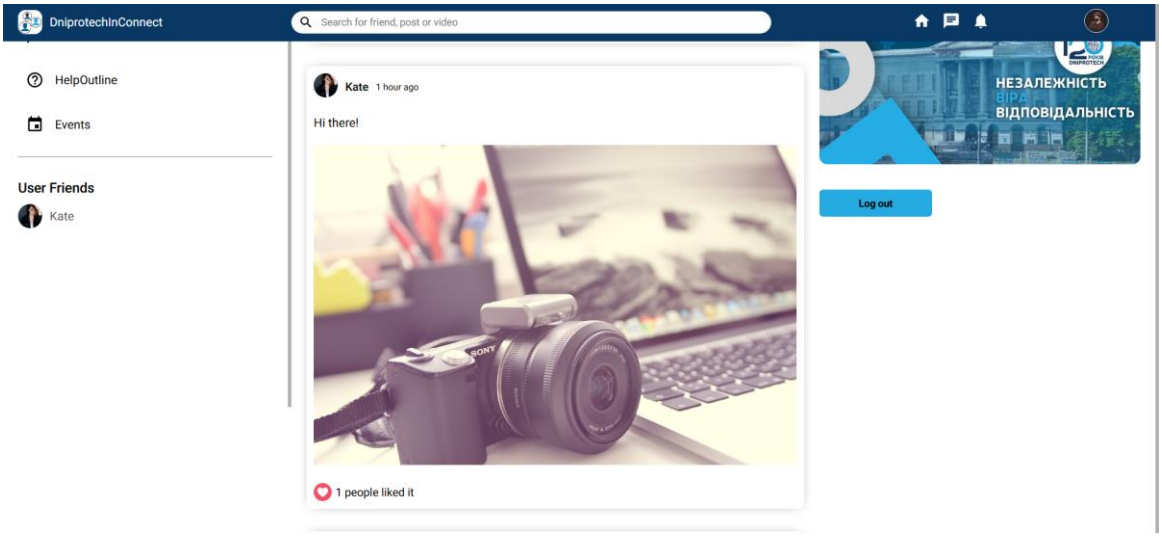

Рис. 2.35. Вподобайка на публікації

Перейдемо до ще однієї основної функції сайту – наявності інформаційного блоку. Так, необхідна для навчання інформація виводиться у компоненті при натисканні кнопки «Info» у лівому меню сайту, що показано на рис. 2.36.

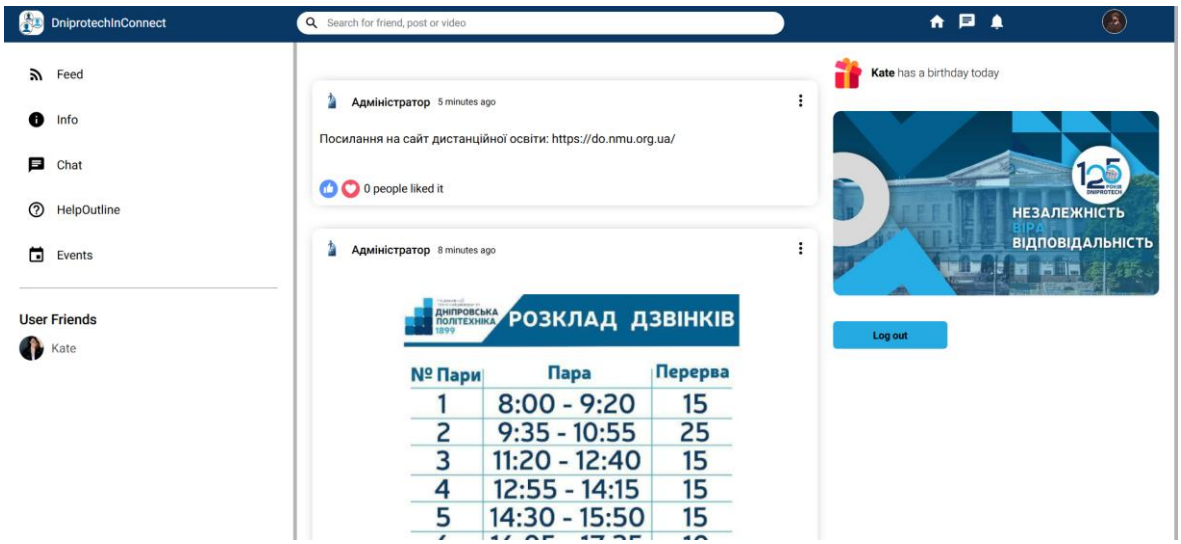

Рис. 2.36. Вивід блоку інформації на сторінці соціальної мережі

Крім того, користувач має можливість переглядати календар подій на сайті. Перехід на цю сторінку здійснюється при натисканні кнопки «Events» у лівому меню веб-ресурсу або іконки «дзвіночок» у верхньому полі веб-додатку. У результаті отримаємо сторінку, яка показана на рис. 2.37.

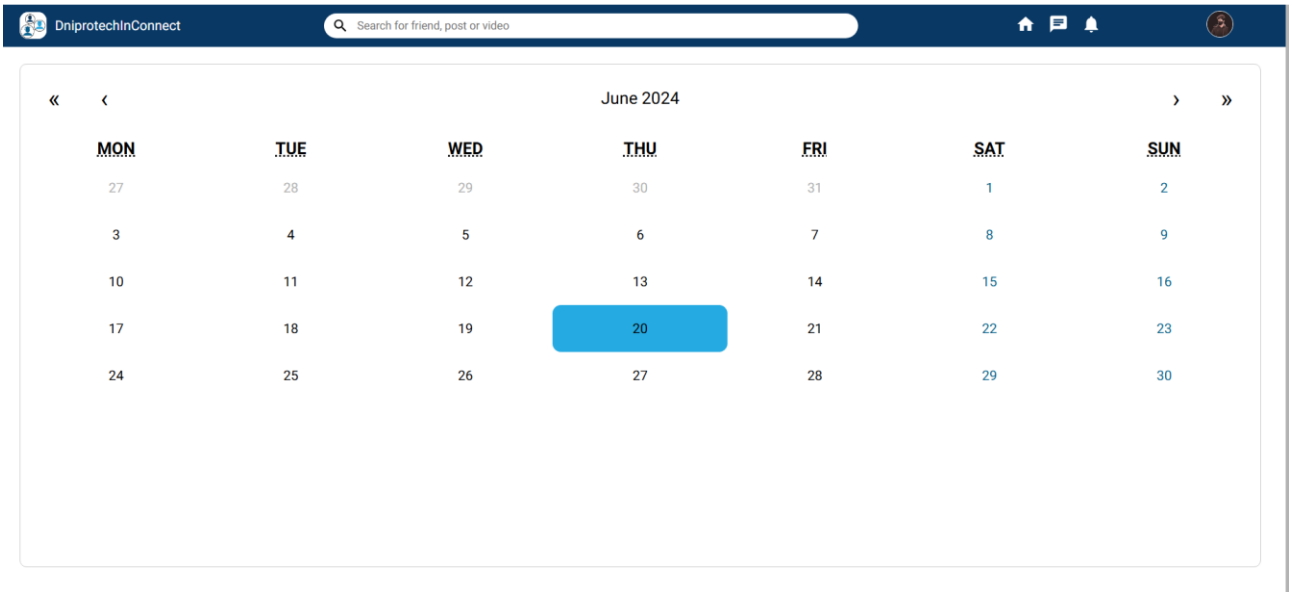

Рис. 2.37. Сторінка календаря

Натискання на дату дозволить користувачеві передивлятися події, які будуть відбуватися у відповідний день, це зображено на рис. 2.38.

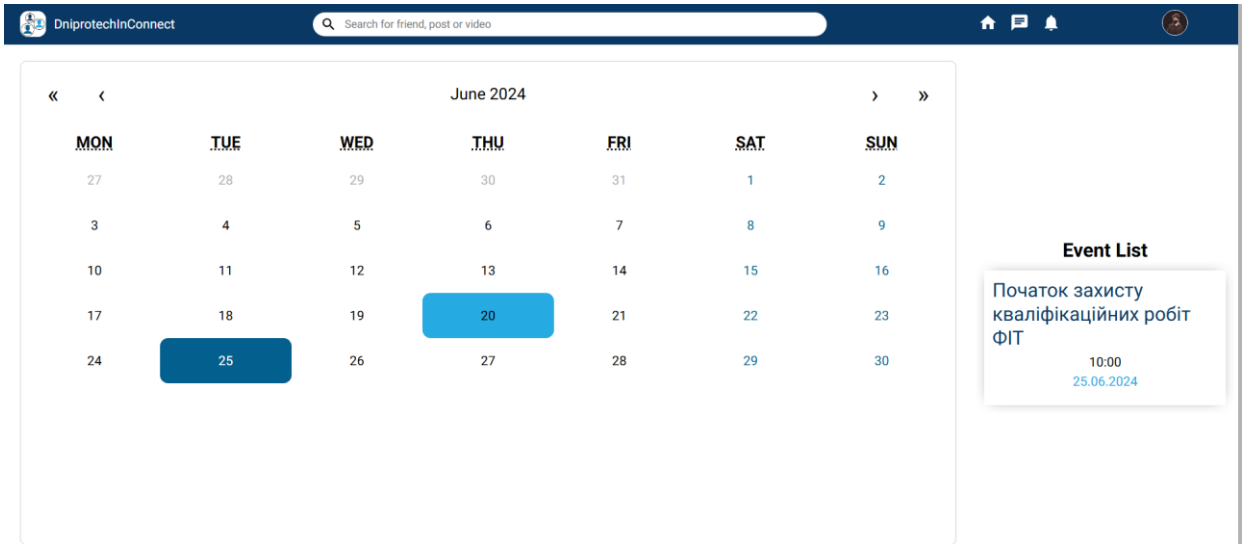

Рис. 2.38. Результати обрання дати в календарі

Технічно вивід списку подій показано на рис. 2.39.

```
{events.length > 0 && selectedDate &&  (
    <div className="eventList">
        <h2> Event List </h2>
        <div className="eventCards">
            {events.map((event) =>
                (formattedDate(event.eventDate) === formattedDate(selectedDate) ?
                    (<div
                        key={event._id}
                        className="eventCard"
                    >
                        <div className="eventCardContainer">
                            <div className="eventCardContainerInfo">
                                <p className="eventTitle">
                                    {event.name}
                                </p>
                                <p className="eventTime">
                                    {event.eventTime}
                                </p>
                                <span className="eventDate">
                                    {formattedDate(event.eventDate)}
                                </span>
                            </div>
                            <div className="eventButtons">
                                {isAdmin && (
                                    <button
                                        className="deleteButton"
                                        onClick={() => {handleDeleteEvent(event._id)}}
                                    >
                                        Delete Event
                                    </button>
                                )}
                            </div>
                        </div>
                    </div>
                ) : null)
            )}
        </div>
    </div>
)}
```

Рис. 2.39. Вивід списку подій

Основною сторінкою соціальної мережі є сторінка листування користувачів, перехід на неї здійснюється натисканням кнопки «Messenger» у лівому меню соціальної мережі або натисканням іконки «повідомлення» у верхньому полі ресурсу. Результати переходу на сторінку листування показано на рис. 2.40.

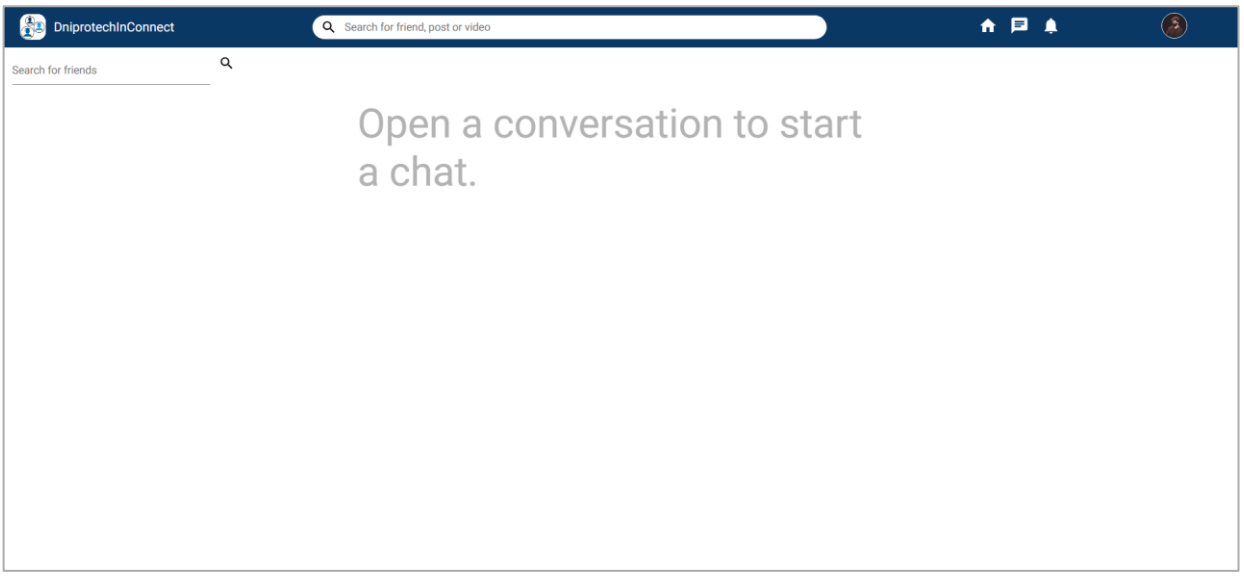

Рис. 2.40. Сторінка листування

Для початку листування необхідно створити чат, для цього треба повторити дії пошуку користувача та натискання результату пошуку. Після цього створиться чат, який можна відкрити натиснувши на нього, як показано на рис. 2.41.

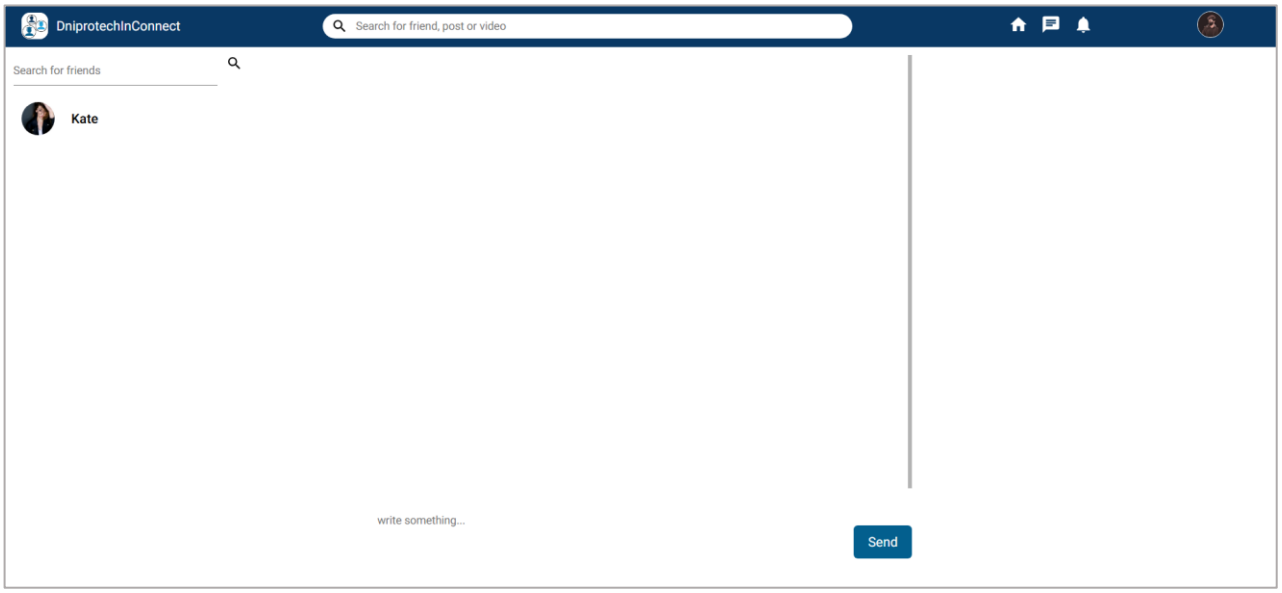

Рис. 2.41. Результати створення чату

Технічно відображення чату зображено на рис. 2.42.

```
useEffect(() => {
    const friendId = conversations.members.find(m => m !== currentUser._id)

    const getUser = async () => {
        try {
            const config = {
                headers: { 'Authorization': authHeader }
            }
            const res = await axios('/users?userId=' + friendId, config)
            setUser(res.data)
        } catch (err) {
            console.log(err)
        }
    }

    getUser()
}, [currentUser, conversations])

return (
    <div className='conversation'>
        <img
            className='conversationImg'
            src={
                user?.profilePicture !== ''
                    ? PF + '/' + user?.profilePicture
                    : PF + '/noAva.png'
            }
            alt={user.username}
        />
        <span className='conversationName'>{user?.username}</span>
    </div>
)
}
```

Рис. 2.42. Відображення чату на сторінці

Тепер користувач може відправити повідомлення, заповнивши відповідний рядок та натиснувши кнопку «Send». Результати відправлення повідомлення показані на рис. 2.43.

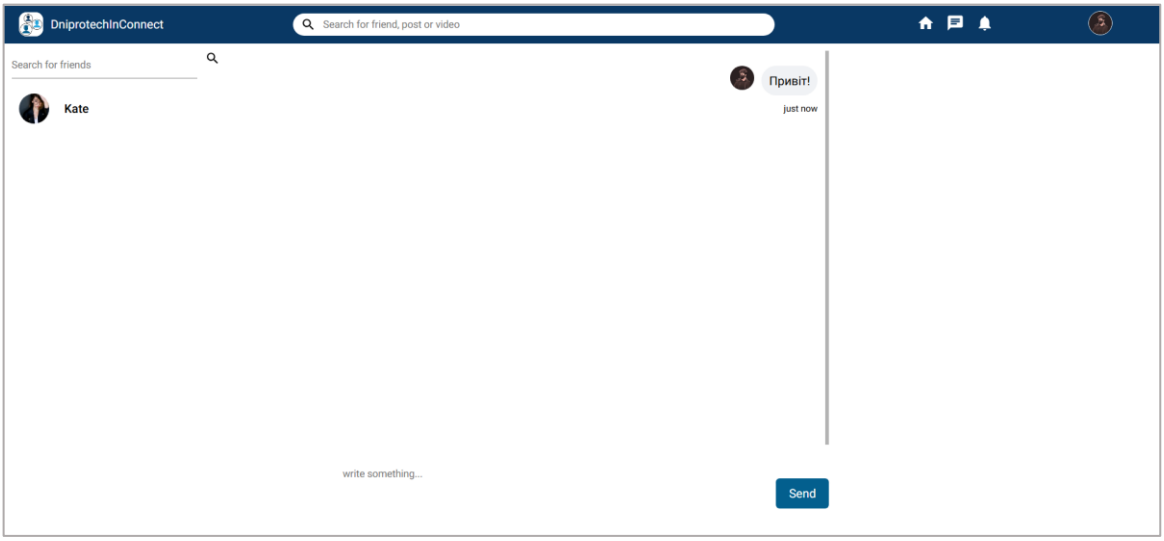

Рис. 2.43. Відправка повідомлення

Таким самим чином відправити повідомлення може і друг користувача, вийде переписка, що показана на рис. 2.44.

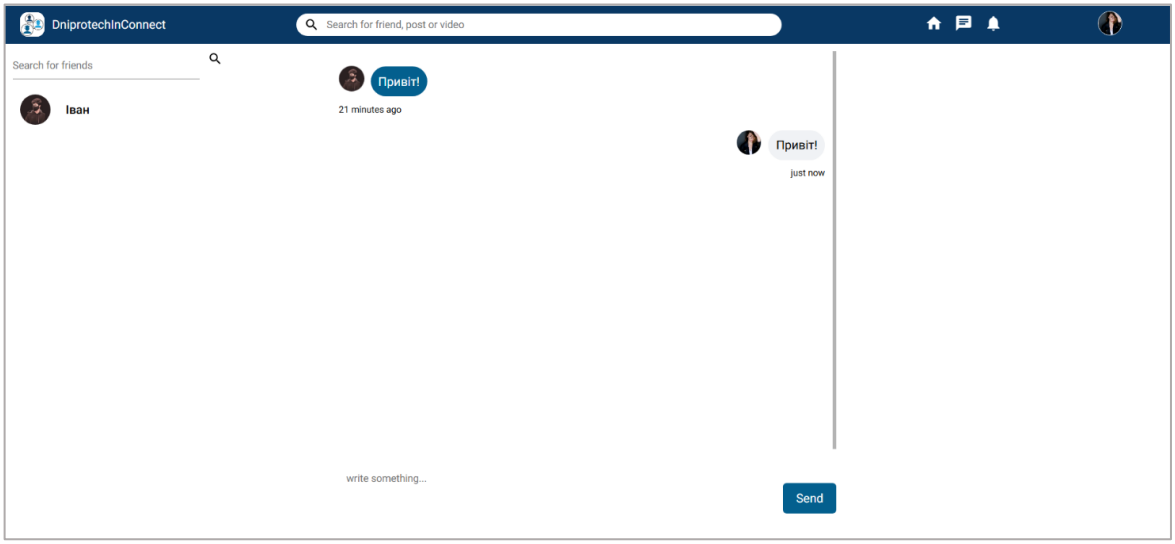

Рис 2.44. Переписка між двома користувачами

Користувач зі статусом адміністратор має деякі переваги у керуванні сайтом: можливість додавати нову інформацію, адже лише для нього з’являється відповідне поле заповнення, як показано на рис. 2.45.

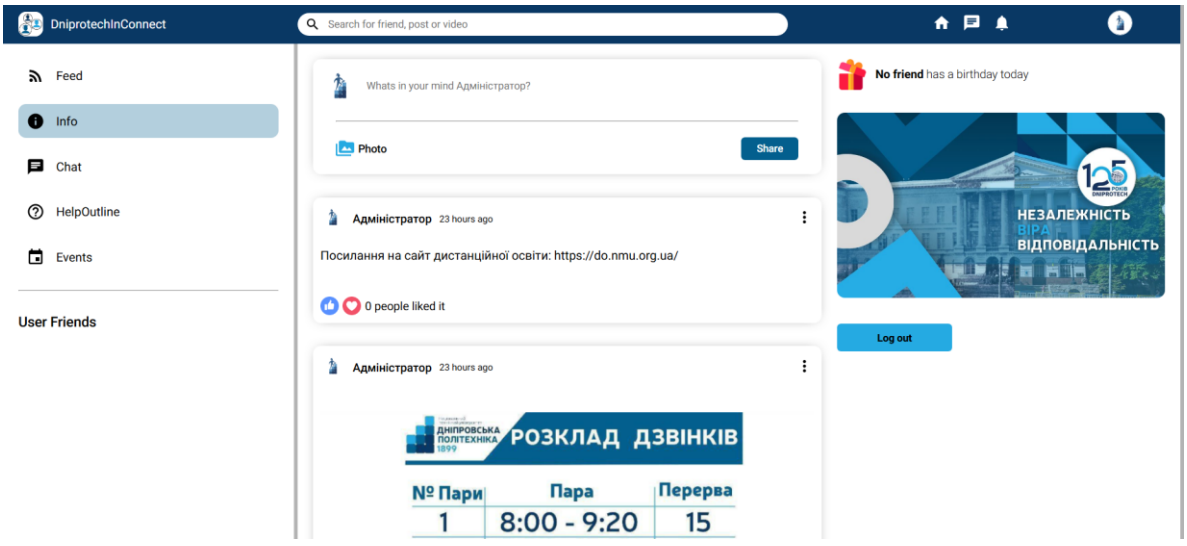

Рис. 2.45. Вигляд інформаційного блоку для адміністратора

Так, адміністратор може заповнити поле та отримати пости з необхідною для навчання інформацією. Етапи публікування таких постів не відрізняються від публикації постів звичайних користувачів.

Також для адміністратора існує форма для додавання нових подій до календаря. Приклад оформлення форми на сторінці показано на рис. 2.46.

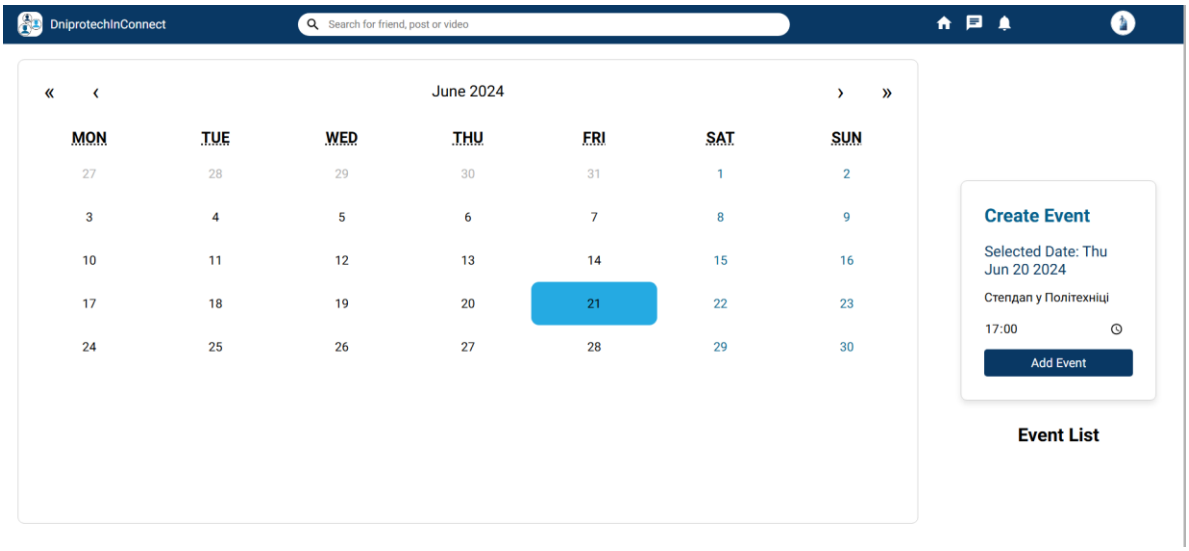

Рис. 2.46. Заповнення форми додавання події в календар

Технічно додавання нової події зображено на рис. 2.47.

```
const handleCreateEvent = async () => {
    const newEvent = {
        userId: user._id,
        name: eventName.current.value,
        eventDate: selectedDate,
        eventTime: eventTime.current.value,
    };
    try {
        const config = {
            headers: { 'Authorization': authHeader }
        }
        await axios.post(`/events`, newEvent, config);
        window.location.reload();
    } catch (err) {
        console.error(err);
    }
};
```

Рис. 2.47. Додавання нової події

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

1. q – передбачуване число операторів – 2157;

2. p – коефіцієнт кореляції програми в ході її розробки – 0,2;

3. C – коефіцієнт складності програми – 1,2.

4. Згідно даних «Української спільноти програмістів (DOU)» середня місячна зарплата Junior JavaScript Software Developer із рівнем англійської Upper-Intrmed в Україні становить 600$ [13]. Згідно даних Національного банку України за курсом валют на початок червня 2024 року один американський долар дорівнює 40,5 грн [14]. В розрахунку отримуємо середню зарплату в гривнях, що дорівнює 24 300 грн. При стандартному графіку (176 годин/місяць) середня зарплата за годину $C_{ПР}$ буде становити близько 138,1 грн.

5. B – коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 0,85;

6. k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;

7. Ноутбук фірми Acer серії Aspire 5, на якому проводилась розробка веб-додатку, в середньому за годину роботи споживає 60 Вт. Ціна за електроенергію в Україні при споживанні на період розробки становила 1,44 грн за 1 кВт [16]. Таким чином вартість 1 години роботи на зазначеному ноутбуці становить 0,06 · 1,44 = 0,1 грн. Вартість домашнього інтернету від провайдера Київстар за тарифним планом «ВСЕ РАЗОМ Міць» становить 16,75 грн в день [15], 0,7 грн за годину. За розрахунками 0,1 + 0,7 = 0,8 (грн/год). Отже, вартість машино-години ЕОМ $C_{мч}$ – 0,8 грн/год;

8. B_k – кількість розробників – 1.

Нормування праці при створенні програмного забезпечення суттєво ускладнюється в силу творчого характеру роботи програміста. Тому оцінка трудомісткості розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \,, \tag{3.1}$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \,, \tag{3.2}$$

де q – передбачуване число операторів (2157);

C – коефіцієнт складності програми (1,2);

p – коефіцієнт корекції програми в ході її розробки (0,2).

Звідси за формулою (3.2) умовне число операторів в програмі:

$$Q = 2157 \cdot 1{,}2 \cdot (1+0{,}2) = 3106{,}08 \text{ людино-годин.}$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \quad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста (0,8), обумовлений від стажу роботи з даної спеціальності. Враховуючи, що при стажі роботи від 2 до 3 років він складає 1.

Визначимо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 0,85). Враховуючи коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності k = 0,6, отримуємо витрати праці на вивчення опису завдання за формулою (3.3):

$$t_u = (3106{,}08 \cdot 0{,}85) / (85 \cdot 0{,}8) = 38{,}826 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин,} \quad (3.5)$$

де Q – умовне число операторів програми, людино-годин (3218,4 людино-годин);

k – коефіцієнт кваліфікації програміста (0,6).

Підставивши відповідні значення в формулу (3.5), отримаємо:

$$t_a = 3106{,}08 / (25 \cdot 0{,}8) = 155{,}304 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25)\cdot k}\text{, людино-годин.} \quad (3.6)$$

За формулою (3.6) та значенням параметру Q, обчислюємо витрати праці на програмування по готовій блок-схемі:

$$t_n = 3106{,}08 / (25 \cdot 0{,}8) = 155{,}304 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\text{, людино-годин.} \quad (3.7)$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.7):

$$t_{отл} = 3106{,}08 / (5 \cdot 0{,}8) = 776{,}52 \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 \cdot t_{отл}\text{, людино-годин.} \quad (3.8)$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.8):

$$t_{отл}^{k} = 1{,}5 \cdot 776{,}52 = 1056{,}78 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}\text{, людино-годин,} \quad (3.9)$$

де $t_{дp}$ – трудомісткість підготовки матеріалів і рукопису:

$$t_{дp} = \frac{Q}{(15..20)\cdot k}\text{, людино-годин,} \quad (3.10)$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{дp}\text{, людино-годин.} \quad (3.11)$$

За формулами (3.9), (3.10) та (3.11) обчислюємо витрати праці на документацію:

$t_{дp}$= 3106,08 / (20 · 0,8) = 194,13 людино-годин,

$t_{до}$= 0,75 · 194,13 = 145,6 людино-годин,

$t_{д}$= 145,6 + 194,13 = 339,728 людино-годин.

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 38,826 + 155,304 + 155,304 + 776,52 + 339,728 = 1515, 682 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ містять в собі витрати на заробітну плату розробника програми $З_{ЗП}$ і витрати машинного часу $З_{Мч}$, необхідного для налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{Мч}\text{, грн.} \quad (3.12)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}\text{ , грн,} \quad (3.13)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 138,1 грн / год, за формулою (3.13) отримуємо:

$$З_{ЗП} = 1515,682 \cdot 138,1 = 209\,315,684 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}\text{ , грн,} \quad (3.14)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год (1072,8 год);

$C_{мч}$ – вартість машино-години ЕОМ, грн/год (0,8 грн/год).

Підставивши в формулу (3.14) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 776,52 \cdot 0,8 = 621,216 \text{ грн.}$$

Звідси, за формулою (3.12), витрати на створення програмного продукту:

$$К_{ПО} = 209315,684 + 621,216 = 209936,9 \text{ грн.}$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k \cdot F_p}\text{ , міс,} \quad (3.15)$$

де B_k– число виконавців (дорівнює 1);

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси, за формулою (3.15), витрати на створення програмного продукту:

$$T = 1515, 682 / (1 \cdot 176) = 8,6 \text{ місяців}$$

Висновки: соціальну мережу було розроблено з метою забезпечення соціалізації між студентами університету. Програмне забезпечення включає функції реєстрації, авторизації, обміну повідомленнями та інформацією, спілкування, тому загальна трудомісткість розробки веб-додатку складає 1515, 682 людино-годин. Отже, очікуваний термін розробки дорівнює приблизно 8 з половиною місяців. Цей час було визначено з урахуванням кількості операторів у програмі та кваліфікації програміста, тривалості дослідження, розробки алгоритму, написання коду, налагодження програми та підготовки документації. Вартість однієї години роботи розробника становить 138,1 грн, а вартість машино-години – 0,8 грн. Таким чином, загальна вартість розробки складає 209 936,9 грн.

ВИСНОВКИ

Згідно з завданням і метою кваліфікаційної роботи було реалізовано соціальну мережу для спілкування студентів Національного технічного університету «Дніпровська політехніка».

Аналіз існуючих рішень, які можуть мати необхідний функціонал для вирішення проблеми, показав, що жоден веб-ресурс загалом не відповідає зазначеним критеріям. Основними перевагами розробленого веб-додатку є корпоративний стиль оформлення, закрита спільнота користувачів, можливість перегляду учбової інформації та календаря заходів конкретного університету.

Практичне значення проєкту полягає у забезпеченні студентської спільноти платформою для спілкування, успішного навчання та особистісного розвитку.

Розроблена соціальна мережа має функції реєстрації, авторизації та поділу користувачів згідно статусу.

Звичайний користувач має такі можливості:

– створення особистої сторінки з додаванням інформації про себе;

– створення публікацій з описом та картинками;

– додавання нових друзів, перегляд їхніх профілів, реагування на їхні публікації;

– організацію чату із другом;

– перегляд сторінки календаря із подіями та блоку з інформацією, корисною для навчання.

Користувач зі статусом адміністратор має ті ж самі функції, але додатково може створювати події у календарі та інформаційні блоки у відповідному компоненті.

Для розробки веб-додатку було використано наступні технології: трирівневу клієнт-серверну архітектуру побудови проєкту, стек технологій MERN, що складається з React.js, Node.js, Express.js та MongoDB.

Згідно економічної частини кваліфікаційної роботи, трудомісткість розробки складає 1515, 682 людино-годин, очікуваний період створення веб-

додатку приблизно становить 8 з половиною місяців, витрати на реалізацію застосунку складають 209 936,9 грн.

У майбутньому планується розширити функціональні можливості користувачів у соціальній мережі, а також створити взаємодію додатка з сайтом дистанційної освіти, що використовується університетом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. М. Байдак, .В О. Болотова, Н. О. Ляшенко сучасний студент у соціальних мережах. // Соціальні технології: актуальні проблеми теорії та практики, 2019, Вип. 84 / URL: https://www.researchgate.net/publication/341561053_SUCASNIJ_STUDENT_U_SOCIALNIH_MEREZAH. дата звернення: 10.06.2024

2. Використання інструментів соціальних мереж у процесі навчання Чафаі Хоуда, Марокко / URL: https://openarchive.nure.ua/server/api/core/bitstreams/c2dbfb8c-e9ea-4a02-84a1-941bfa88000a/content. дата звернення: 10.06.2024

3. Освіта.UA - Соціальні мережі та електронні платформи для науковців / URL: https://osvita.ua/vnz/76103/. дата звернення: 10.06.2024

4. Портал Academia.edu / URL: https://www.academia.edu/. дата звернення: 10.06.2024

5. Соціальна мережа Researchgate / URL: https://www.researchgate.net/. дата звернення: 10.06.2024

6. Соціальна мережа Computer Science Student Network / URL: https://www.cs2n.org/. дата звернення: 10.06.2024

7. Соціальна мережа Instagram / URL: https://www.instagram.com/. дата звернення: 10.06.2024

8. Соціальна мережа Facebook / URL: https://www.facebook.com/. дата звернення: 10.06.2024

9. Соціальна мережа X / URL: https://x.com/i/flow/login?input_flow_data=%7B%22requested_variant%22%3A%22eyJteCI6IjIifQ%3D%3D%22%7D. дата звернення: 10.06.2024

10. Месенджер Telegram / URL: https://web.telegram.org/k/ [10.06.2024]

11. Месенджер Viber / URL: https://account.viber.com/ru/login. дата звернення: 10.06.2024

12. Coursera - SQL vs. NoSQL: The Differences Explained + When to Use Each / URL: https://www.coursera.org/articles/nosql-vs-sql. дата звернення: 25.05.2024

13. Української спільноти програмістів (DOU). Заробітна плата Junior розробника / URL: https://jobs.dou.ua/salaries/?period=2023-12&position=Junior%20SE&technology=JavaScript&experience=0&english=1-4. дата звернення: 02.06.2024

14. Офіційний сайт Національного банку України. Курс валют / URL: https://bank.gov.ua/ua/markets/exchangerates. дата звернення: 03.06.2024

15. Офіційний сайт Київстар. Тарифи для домашнього інтернету / URL: https://kyivstar.ua/internet-tv-mobile/alltogetherstrength. дата звернення: 03.06.2024

16. Ясно. Тарифи на електоренергію для побутових споживачів / URL: https://yasno.com.ua/news/all_news/electricity-tariffs-for-household-consumers-from-01-06-2023. дата звернення: 03.06.2024

17. Visual Studio Code / URL: https://code.visualstudio.com/ [21.05.2024]

18. MongoDB / URL: https://www.mongodb.com/. [21.05.2024]

19. Figma / URL: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma. дата звернення: 20.05.2024

20. Fullstack React / Anthony Accomazzo, Ari Lerner, Clay Allsopp, David Gutman, Tyler Mcginnis. дата звернення: 27.05.2024

21. BEGINNING. ReactJS Foundations. Building User Interfaces with ReactJS. / Chris Minnick. дата звернення: 23.05.2024

22. How to code in React.js / Joe Morgan. дата звернення: 23.05.2024

23. Pro MERN Stack. Full Stack Web App Development with Mongo,Express, React, and Node / Vasan Subramanian. дата звернення: 25.05.2024

24. Mastering Node.js. Expert techniques for building fast servers and scalable, real-time network applications with minimal effort / Sandro Pasquali. дата звернення: 25.05.2024

25. MongoDB: The Definitive Guide / Kristina Chodorow. дата звернення: 26.05.2024

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

api/index.js

```
const express = require('express')
const dotenv = require('dotenv').config()
const mongoose = require('mongoose')
const helmet = require('helmet')
const morgan = require('morgan')
const app = express()
const multer = require('multer')
const path = require('path')

const userRoute = require('./routes/users')
const authRoute = require('./routes/auth')
const postRoute = require('./routes/posts')
const conversationRoute = require('./routes/conversations')
const messageRoute = require('./routes/messages')
const infoRoute = require('./routes/infos')
const eventRoute = require('./routes/events')
const passport = require('passport');
require('./auth/auth');
process.env.TOKEN_SECRET;
mongoose.connect(
   process.env.MONGO_URL, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
   }
)
   .then(console.log('Connected to MongoDB'))
app.use('/images', express.static(path.join(__dirname, 'public/images')))
// middleware
app.use(express.json())
app.use(helmet({
   crossOriginResourcePolicy: false
}))
app.use(morgan('common'))
app.use(passport.initialize());
const storage = multer.diskStorage({
   destination: (req, file, cb) => {
      cb(null, 'public/images')
   },
   filename: (req, file, cb) => {
      cb(null, req.body.name)
   },
})
const upload = multer({ storage })
app.post('/api/upload', upload.single('file'), (req, res) => {
   try {
      return res.status(200).json('File uploaded successfully.')
```

```
    } catch (err) {
        console.log(err)
    }
})
app.use('/api/auth', authRoute)
app.use('/api/conversations', passport.authenticate('jwt', { session: false }), conversationRoute)
app.use('/api/infos', passport.authenticate('jwt', { session: false }), infoRoute)
app.use('/api/messages', passport.authenticate('jwt', { session: false }), messageRoute)
app.use('/api/posts', passport.authenticate('jwt', { session: false }), postRoute)
app.use('/api/users', passport.authenticate('jwt', { session: false }), userRoute)
app.use('/api/events', passport.authenticate('jwt', { session: false }), eventRoute)
app.listen(8800, () => {
    console.log('Backend server is running!')
})
```

authenticateToken.js

```
const jwt = require('jsonwebtoken');
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization']
    const token = authHeader && authHeader.split(' ')[1]
    if (token == null) return res.sendStatus(401)
    jwt.verify(token, process.env.TOKEN_SECRET, (err, user) => {
        console.log(err)
        if (err) return res.sendStatus(403)
        req.user = user
        next()
    })
}
```

auth.js

```
const router = require('express').Router()
const User = require('../models/User')
const jwt = require('jsonwebtoken');
const passport = require('passport');
const localStrategy = require('passport-local').Strategy;
// register
router.post('/register',
    passport.authenticate('signup', { session: false }),
    async (req, res) => {
    try {
        // create new user
        let newUser = req.user
        newUser.username = req.body.username
        // save user and respond
        const user = await newUser.save()
        res.status(200).json({ user: user })
        // res.status(200).json(user)
    } catch (err) {
        res.status(500).json(err)
    }
})
// login
router.post(
    '/login',
```

```
  async (req, res, next) => {
    passport.authenticate(
      'login',
      async (err, user, info) => {
        try {
          if (err || !user) {
            const error = new Error('An error occurred.');
            return next(error);
          }

          req.login(
            user,
            { session: false },
            async (error) => {
              if (error) return next(error);

              const body = { _id: user._id, email: user.email };
              const token = jwt.sign({ user: body }, process.env.TOKEN_SECRET, {
                expiresIn: 3600 // in seconds
              } );

              return res.json({ user: user, token: token });
              // return (
              //  res.json({ user: user, token: token }),
              //  console.log(res)
              // )
            }
          );
        } catch (error) {
          return next(error);
        }
      }
    )(req, res, next);
  }
);
function generateAccessToken(username) {
  return jwt.sign(username, process.env.TOKEN_SECRET, { expiresIn: '1800s' });
}
module.exports = router
```

User.js

```
const mongoose = require('mongoose')
const bcrypt = require('bcrypt')
const UserSchema = new mongoose.Schema(
        {
                username: {
                        type: String,
                        require: true,
                        min: 3,
                        max: 20,
                        unique: true,
                },
                email: {
                        type: String,
```

```
            required: true,
            max: 50,
            unique: true,
},
password: {
            type: String,
            required: true,
            min: 6,
},
token: {
            type: String,
},
profilePicture: {
            type: String,
            default: '',
},
coverPicture: {
            type: String,
            default: '',
},
followers: {
            type: Array,
            default: [],
},
followings: {
            type: Array,
            default: [],
},
isAdmin: {
            type: Boolean,
            default: false,
},
desc: {
            type: String,
            max: 50,
},
city: {
            type: String,
            max: 50,
},
relationship: {
            type: Number,
            enum: [1, 2, 3],
},
faculty: {
            type: String,
            max: 50,
},
group: {
            type: String,
            max: 20,
},
birthday: {
            type: Date,
},
```

```
        },
        { timestamps: true }
)
UserSchema.pre(
        'save',
        async function(next) {
                const user = this;
                const hash = await bcrypt.hash(this.password, 10);
                this.password = hash;
                next();
        }
);
UserSchema.methods.isValidPassword = async function(password) {
        const user = this;
        const compare = await bcrypt.compare(password, user.password);
        return compare;
}
module.exports = mongoose.model('User', UserSchema)
```

auth.js

```
const passport = require('passport');
const localStrategy = require('passport-local').Strategy;
const JWTstrategy = require('passport-jwt').Strategy;
const ExtractJWT = require('passport-jwt').ExtractJwt;
const User = require('../models/User');
const CookieExtractor = function(req) {
   let token = null;
   if(req && req.cookies) {
      token = req.cookies['_auth'];
   }
   return token;
}
passport.use(
   'signup',

   new localStrategy(
      {
         usernameField: 'email',
         passwordField: 'password',
      },
      async (email, password, done) => {
         try {
            const user = new User({
               email: email,
               password: password
            })
            console.log(user);
            return done(null, user);
         } catch (error) {
            done(error);
         }
      }
   )
);
```

```
passport.use(
   'login',
   new localStrategy(
      {
         usernameField: 'email',
         passwordField: 'password'
      },
      async (email, password, done) => {
         try {
            const user = await User.findOne({ email: email });
            if (!user) {
               return done(null, false, { message: 'User not found' });
            }
            const validate = await user.isValidPassword(password);
            if (!validate) {
               return done(null, false, { message: 'Wrong Password' });
            }
            return done(null, user, { message: 'Logged in Successfully' });
         } catch (error) {
            return done(error);
         }
      }
   )
);
passport.use('jwt',
   new JWTstrategy(
      {
         secretOrKey: process.env.TOKEN_SECRET,
         jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken('Authorization')
      },
      async (token, done) => {
         try {
            return done(null, token.user);
         } catch (error) {
            done(error);
         }
      }
   )
);
```

client/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
   <App />
);
```

App.js

```
import { MyCalendar, Home, Login, Messenger, Profile, Register } from './components/index'
import {
```

```
  Navigate,
  Route,
  BrowserRouter,
  Routes,
} from 'react-router-dom'
import AuthProvider from 'react-auth-kit'
import AuthOutlet from '@auth-kit/react-router/AuthOutlet'
import createStore from "react-auth-kit/createStore";
const store = createStore({
  authName:'_auth',
  authType:'cookie',
  cookieDomain: window.location.hostname,
  cookieSecure: window.location.protocol === 'https:',
})
function App() {
return (
  <AuthProvider store={store}>
    <BrowserRouter>
      <Routes>
        <Route element={<AuthOutlet fallbackPath="/login" />}>
          <Route path='/' element={<Home/>} />
          <Route path='/messenger' element={<Messenger />}/>
          <Route path='/profile/:username' element={<Profile/>} />
        </Route>
        <Route path='/login' element={<Login />} />
        <Route path='/register' element={<Register />}/>
        <Route path='/calendar' element={<MyCalendar/>} />
      </Routes>
    </BrowserRouter>
  </AuthProvider>
  );
}

export default App;
```

Register.jsx

```
import axios from 'axios'
import { useRef } from 'react'
import { useNavigate, Link } from 'react-router-dom'
import './register.css'
export default function Register() {
        const PF = process.env.REACT_APP_PUBLIC_FOLDER
        const username = useRef()
        const email = useRef()
        const password = useRef()
        const passwordAgain = useRef()
        const history = useNavigate()
        const handleClick = async e => {
                e.preventDefault()
                if (passwordAgain.current.value !== password.current.value) {
                        console.log(passwordAgain.current.value, password.current.value)
                        passwordAgain.current.setCustomValidity('Password don`t match!')
                } else {
                        const user = {
```

```
                username: username.current.value,
                email: email.current.value,
                password: password.current.value,
            }
            try {
                await axios.post('auth/register', user)
                history('/login')
            } catch (err) {
                console.log(err)
            }
        }
    }
    return (
        <div className='register'>
            <div className='registerWrapper'>
                <div className='registerLeft'>
                    <div className="registerLogoIcon">
                        <img src={PF + '/logo.png'} alt="" />
                    </div>
                    <h3 className='registerLogo'>DniprotechInTouch</h3>
                    <span className='registerDesc'>
                        Connect with friends and the world around you on
DniprotechInTouch
                    </span>
                </div>
                <div className='registerRight'>
                    <form className='registerBox' onSubmit={handleClick}>
                        <input
                            className='registerInput'
                            placeholder='Username'
                            type='text'
                            ref={username}
                            required
                        />
                        <input
                            className='registerInput'
                            placeholder='Email'
                            type='email'
                            ref={email}
                            required
                        />
                        <input
                            className='registerInput'
                            placeholder='Password'
                            type='password'
                            ref={password}
                            required
                            minLength='6'
                        />
                        <input
                            className='registerInput'
                            placeholder='Password Again'
                            type='password'
                            ref={passwordAgain}
                            required
```

```
                                        minLength='6'
                                    />
                                    <button className='registerButton' type='submit'>
                                        Sign Up
                                    </button>
                                    <button className='registerLoginButton'>
                                        <Link to='/login'>
                                            Log into Account
                                        </Link>
                                    </button>
                                </form>
                            </div>
                        </div>
                    </div>
                )
}
```

Login.jsx

```
import { CircularProgress } from '@mui/material'
import { useEffect, useState } from 'react'
import './login.css'
import axios, {AxiosError} from "axios";
import useSignIn from "react-auth-kit/hooks/useSignIn";
import {useFormik} from "formik"
import {Link, useNavigate} from "react-router-dom";
export default function Login() {
        const PF = process.env.REACT_APP_PUBLIC_FOLDER
        const [error,setError ]  = useState('')
        const signIn = useSignIn()
        const navigate = useNavigate()
        const onSubmit = async (values) => {
            try {
                const res = await axios.post('auth/login', values);
                console.log('Credentials', values)
                setError('')
                if(signIn({
                    auth: {
                        token: res.data.token,
                        type: 'Bearer',
                    },
                    userState: { user: res.data.user}
                    })){
                    navigate(`/profile/${res.data.user.username}`)
                } else {
                    alert("Error Occoured. Try Again")
                }
            } catch (err){
                if (err && err instanceof AxiosError)
                    setError(err.res?.data.message);
                else if (err && err instanceof Error) setError(err.message)
            }
        }
        const formik = useFormik({
            initialValues: { email: '', password: '' },
```

```
        onSubmit
    })
    let isSubmitting = false;
    return (
        <div className='login'>
            <div className='loginWrapper'>
                <div className='loginLeft'>
                    <div className="loginLogoIcon">
                        <img src={PF + '/logo.png'} alt="" />
                    </div>
                    <h3 className='loginLogo'>DniprotechInTouch</h3>
                    <span className='loginDesc'>
                        Connect with students on DniprotechInTouch
                    </span>
                </div>
                <div className='loginRight'>
                    <form className='loginBox' onSubmit={formik.handleSubmit}>
                        <input
                            id="email"
                            className='loginInput'
                            placeholder='Email'
                            type='email'
                            value={formik.values.email}
                            onChange={formik.handleChange}
                            required
                        />
                        <input
                            id="password"
                            className='loginInput'
                            placeholder='Password'
                            type='password'
                            value={formik.values.password}
                            onChange={formik.handleChange}
                            minLength='6'
                            required
                        />
                        <button type="submit" className='loginButton'
disabled={isSubmitting}>
                            {isSubmitting ? (
                                <CircularProgress color='inherit' size='20px' />
                            ) : (
                                'Log In'
                            )}
                        </button>
                        <span className='loginForgot'>Forgot Password?</span>
                        <button className='loginRegisterButton'>
                            <Link to={'/register'}>
                                {isSubmitting ? (
                                    <CircularProgress color='inherit'
size='20px' />
                                ) : (
                                    'Create a New Account'
                                )}
                            </Link>
                        </button>
```

```
                                        </form>
                                    </div>
                                </div>
                            </div>
                    )
}
```

Profile.jsx

```
import { Feed, RightBar, Sidebar, Topbar, Info, UpdateUserInfo } from './../../components/index.js'
import axios from 'axios'
import { useEffect, useState } from 'react'
import  { useParams } from 'react-router-dom'
import './profile.css'
import useAuthHeader from "react-auth-kit/hooks/useAuthHeader";
export default function Profile() {
            const PF = process.env.REACT_APP_PUBLIC_FOLDER
            const [user, setUser] = useState()
   const [showFeed, setShowFeed] = useState(true);
   const [showInfo, setShowInfo] = useState();
   const [showUpdateUserInfo, setShowUpdateUserInfo] = useState(false);
            const username = useParams().username
            const authHeader = useAuthHeader();
   const handleFeedClick = () => {
      setShowFeed(true);
      setShowInfo(false);
                    setShowUpdateUserInfo(false);
                    console.log('feed clicked');
   };
   const handleInfoClick = () => {
      setShowInfo(true);
      setShowFeed(false);
                    setShowUpdateUserInfo(false);
                    console.log('info clicked');
   };
            const handleUpdateUserInformationClick = () => {
                    setShowUpdateUserInfo(true)
      setShowInfo(false);
      setShowFeed(false);
   };
            useEffect(() => {
                    const fetchUser = async () => {
                            const config = {
                                    headers: { 'Authorization': authHeader }
                            }
                            const res = await axios.get( `/users?username=${username}`, config);
                            setUser(res.data)
                    }
                    fetchUser()
            }, [username])
            return (
                    <div>
                            <Topbar  />
                            { user && (
                                    <div className='profile'>
```

```
                                        <Sidebar onFeedClick={handleFeedClick}
onInfoClick={handleInfoClick}
                                        />
                                        <div className='profileRight'>
                                                <div className='profileRightTop'>
                                                        <div className='profileCover'>
                                                                <img
                                                                        className='profileCoverImg'
                                                                        src={
                                                                                user.coverPicture !== ''
                                                                                        ? PF + '/' +
user?.coverPicture
                                                                                        : PF + '/cover.jpeg'
                                                                        }
                                                                        alt=''
                                                                />
                                                                <img
                                                                        className='profileUserImg'
                                                                        src={
                                                                                user.profilePicture !== ''
                                                                                        ? PF + '/' +
user?.profilePicture
                                                                                        : PF + '/noAva.png'
                                                                        }
                                                                        alt=''
                                                                />
                                                        </div>
                                                        <div className='profileInfo'>
                                                                <h4
className='profileInfoName'>{user?.username}</h4>
                                                                <span
className='profileInfoDesc'>{user?.desc}</span>
                                                        </div>
                                                </div>

                                                <div className='profileRightBottom'>
                                                        {showFeed ? <Feed username={username} /> : null}
                                                        {showInfo ? <Info /> : null}
                                                        {showUpdateUserInfo ? <UpdateUserInfo
user={user}/> : null}
                                                        {user ? <RightBar user={user}
onUpdateUserInformationClick={handleUpdateUserInformationClick}/> : null}
                                                </div>
                                        </div>
                                </div>)}
                        </div>
                )
}
```

Home.jsx

```
import './home.css'
import {Topbar, Sidebar, Info, Feed, RightBar } from './../../components/index'
import { useState } from 'react';
export default function Home() {
```

```
    const [showFeed, setShowFeed] = useState(true);
    const [showInfo, setShowInfo] = useState(false);
    const handleFeedClick = () => {
        setShowFeed(true);
        setShowInfo(false);
        console.log('showfeed')
    };
    const handleInfoClick = () => {
        setShowInfo(true);
        setShowFeed(false);
        console.log('showinfo')
    };
    return (
        <div>
            <Topbar />
            <div className="homeContainer">
                <Sidebar onFeedClick={handleFeedClick} onInfoClick={handleInfoClick}/>
                {showInfo ? <Info /> : null}
                {showFeed ? <Feed /> : null}
                <RightBar />
            </div>
        </div>
    )
}
```

Messenger.jsx

```
import './messenger.css'
import { ChatOnline, Conversation, Message, Topbar } from './../../components/index.js'
import axios from 'axios'
import { useEffect, useRef, useState } from 'react'
import { io } from 'socket.io-client'
import {Search} from '@mui/icons-material'
import useAuthUser from 'react-auth-kit/hooks/useAuthUser';
import useAuthHeader from "react-auth-kit/hooks/useAuthHeader";
export default function Messenger() {
            const PF = process.env.REACT_APP_PUBLIC_FOLDER
    const [conversations, setConversations] = useState([])
    const [currentChat, setCurrentChat] = useState(null)
    const [messages, setMessages] = useState([])
    const [newMessage, setNewMessage] = useState('')
    const [arrivalMessage, setArrivalMessage] = useState(null)
    const [onlineUsers, setOnlineUsers] = useState([])
    const socket = useRef()
    const auth = useAuthUser()
    const user = auth.user
    const scrollRef = useRef()
    const [searchQuery, setSearchQuery] = useState('');
    const [searchResults, setSearchResults] = useState(false);
    const [isResultsVisible, setIsResultsVisible] = useState(false);
    const searchResultsRef = useRef(null);
            const authHeader = useAuthHeader();
    useEffect(() => {
        socket.current = io('ws://localhost:8900')
        socket.current.on('getMessage', data => {
```

```
    setArrivalMessage({
      sender: data.senderId,
      text: data.text,
      createdAt: Date.now(),
    })
    })
}, [])
useEffect(() => {
    arrivalMessage &&
    currentChat?.members.includes(arrivalMessage.sender) &&
    setMessages(prev => [...prev, arrivalMessage])
}, [arrivalMessage, currentChat])
useEffect(() => {
    socket.current.emit('addUser', user._id)
    socket.current.on('getUsers', users => {
    setOnlineUsers(
      user.followings.filter(f => users.some(u => u.userId === f))
    )
    })
}, [user])
useEffect(() => {
    const config = {
      headers: { 'Authorization': authHeader }
    }
    const getConversations = async () => {
      try {
        const res = await axios.get('/conversations/' + user._id, config)
        setConversations(res.data)
      } catch (err) {
        console.log(err)
      }
    }
    getConversations()
}, [user._id])
useEffect(() => {
    const getMessages = async () => {
      const config = {
        headers: { 'Authorization': authHeader }
      }
      try {
        if(!currentChat) { return }
        const res = await axios.get('/messages/' + currentChat?._id, config)
        setMessages(res.data)
      } catch (err) {
        console.log(err)
      }
    }
    getMessages()
}, [currentChat])
const handleSubmit = async e => {
e.preventDefault()
const message = {
    senderId: user._id,
    text: newMessage,
    conversationId: currentChat._id,
```

```
    }
    const receiverId = currentChat.members.find(member => member !== user._id)
    socket.current.emit('sendMessage', {
      senderId: user._id,
      receiverId,
      text: newMessage,
    })
    try {
      const config = {
        headers: { 'Authorization': authHeader }
      }
      const res = await axios.post('/messages', message, config)
      setMessages([...messages, res.data])
      setNewMessage('')
    } catch (err) {
      console.log(err)
    }
}
    useEffect(() => {
      scrollRef.current?.scrollIntoView({
      behavior: 'smooth',
      })
    }, [messages])
    const handleInputChange = (event) => {
      setSearchQuery(event.target.value);
    };
    const handleSearchSubmit = async () => {
        // Send search query to backend

        try {
          const config = {
            headers: { 'Authorization': authHeader }
          }
          const response = await axios.get(`/users?username=${searchQuery}`, config);
          setSearchResults(response.data);
          setIsResultsVisible(true);
            console.log(response.data);
            console.log(searchResults)
          } catch (error) {
            console.error('Error searching:', error);
          }
      };
    const handleClickOutside = (event) => {
      if (searchResultsRef.current && !searchResultsRef.current.contains(event.target)) {
        setIsResultsVisible(false);
        setSearchQuery('');
      }
    };
    useEffect(() => {
      document.addEventListener('mousedown', handleClickOutside);
      return () => {
        document.removeEventListener('mousedown', handleClickOutside);
      };
    }, []);
    const handleResultClick = async () =>{
```

```
        setIsResultsVisible(false);
        setSearchQuery('');

        try {
          const senderId = user;
          const receiverId = searchResults._id;
          const config = {
            headers: { 'Authorization': authHeader }
          }
          const response = await axios.post('/conversations', {
            senderId,
            receiverId
          }, config);
          console.log('Conversation created:', response.data);
          window.location.reload();
        } catch (error) {
          console.error('Error creating conversation:', error);
        }
      };
  return (
    <>
      <Topbar />
      <div className='messenger'>
        <div className='chatMenu'>
        <div className='chatMenuWrapper'>
          <form  className="searchInputForm" onSubmit={(e) => { e.preventDefault(); handleSearchSubmit(); }}>
            <input className='chatMenuInput'
              placeholder='Search for friends'
              value={searchQuery}
              onChange={handleInputChange} />
            <Search className="searchIcon"  onClick={handleSearchSubmit}/>
          </form>
          {isResultsVisible && searchResults &&
            (<div className="searchResults" ref={searchResultsRef}
            onClick={handleResultClick}>
                <div className='searchResultsUser'>
                  <img
                    className='searchResultsUserImg'
                    src={
                      searchResults.profilePicture !== ''
                        ? PF + '/' + searchResults.profilePicture
                        : PF + '/noAva.png'
                    }
                    alt=''
                  />
                  <span className='searchResultsUserName'>
                    {searchResults.username}
                  </span>
                </div>
            </div>)}
          {conversations.map(c => (
          <div onClick={() => setCurrentChat(c)}>
            <Conversation
            key={c._id}
            conversations={c}
```

```
                currentUser={user}
                />
            </div>
            ))}
        </div>
        </div>
        <div className='chatBox'>
        <div className='chatBoxWrapper'>
            {currentChat ? (
            <>
                <div className='chatBoxTop'>
                {messages.map(m => (
                    <div ref={scrollRef}>
                    <Message
                        key={m._id}
                        messages={m}
                        own={m.senderId === user._id}
                    />
                    </div>
                ))}
                </div>
                <div className='chatBoxBottom'>
                <textarea
                    className='chatMessageInput'
                    placeholder='write something...'
                    onChange={e => setNewMessage(e.target.value)}
                    value={newMessage}
                ></textarea>
                <button className='chatSubmitButton' onClick={handleSubmit}>
                    Send
                </button>
                </div>
            </>
            ) : (
            <span className='noConversationText'>
                Open a conversation to start a chat.
            </span>
            )}
        </div>
        </div>
        <div className='chatOnline'>
        <div className='chatOnlineWrapper'>
            { user ? <ChatOnline
            onlineUsers={onlineUsers}
            currentId={user._id}
            setCurrentChat={setCurrentChat}
            /> : null }
        </div>
        </div>
    </div>
    </>
  )
}
```

MyCalendar.jsx

```
import './myCalendar.css'
import { Calendar} from 'react-calendar'
import React, { useEffect, useRef, useState } from "react";
import axios from 'axios'
import useAuthHeader from "react-auth-kit/hooks/useAuthHeader";
import useAuthUser from 'react-auth-kit/hooks/useAuthUser';
import { Topbar } from '../../components';
export default function MyCalendar() {
   const [selectedDate, setSelectedDate] = useState(null);
   const [events, setEvents] = useState([]);
            const auth = useAuthUser()
            const user = auth.user
   const [isAdmin, setIsAdmin] = useState(false);
            const authHeader = useAuthHeader();
            const eventName = useRef()
            const eventTime = useRef()
   const dateSelectedRef = useRef()
   useEffect(() => {
      const fetchAdminStatus = async () => {
                              const config = {
                                       headers: { 'Authorization': authHeader }
                              }
         try {
            const res = await axios.get(`/users/admin?userId=${user._id}`, config);
            setIsAdmin(res.data.isAdmin);
         } catch (err) {
            console.error('User is not admin:');
         }
      };
      fetchAdminStatus();
   }, [user.id]);
   const handleDateClick = (date) => {
      setSelectedDate(date);
      const fetchEvents = async () => {
         try {
            const config = {
               headers: { 'Authorization': authHeader }
            }
            const events = await axios.get(`/events`, config);
            const eventsList = events.data
            setEvents(eventsList);
         } catch (err) {
            console.error(err);
         }
      }
      fetchEvents();
   };
   const handleCreateEvent = async () => {
      const newEvent = {
         userId: user._id,
         name: eventName.current.value,
         eventDate: selectedDate,
         eventTime: eventTime.current.value,
      };
```

```
    try {
                        const config = {
                            headers: { 'Authorization': authHeader }
                        }
      await axios.post(`/events`, newEvent, config);
      window.location.reload();
    } catch (err) {
      console.error(err);
    }
  };
  const handleDeleteEvent = async (id) => {
    try {
                        const config = {
                            headers: { 'Authorization': authHeader }
                        }
      await axios.delete(`/events/${id}`, config);
      window.location.reload();
    } catch (err) {
      console.error(err);
    }
  };
  const formattedDate = (date) => {
    const newDate = new Date(date);
    const formattedDate = `${newDate.getDate().toString().padStart(2, '0')}.${(newDate.getMonth() + 1).toString()
                                                .padStart(2,
'0')}.${newDate.getFullYear().toString().padStart(4, '0')}`
    return formattedDate
  }
  const handleClickOutside = (event) => {
    if ((!event.target.className.includes('react-calendar__month-view__days__day')) &&
    (event.target.className.includes('react-calendar__viewContainer'))) {
      setSelectedDate(false);
    }
  };
  useEffect(() => {
    document.addEventListener('mousedown', handleClickOutside);
    return () => {
      document.removeEventListener('mousedown', handleClickOutside);
    };
  }, []);
  return (
    <div className='myCalendar'>
      <Topbar />
      <div className="myCalendarContainer">
        <div className="myCalendarBody">
          <Calendar
            locale="en-GB"
            value={selectedDate}
            onClickDay={handleDateClick}
          />
        </div>
          <div className="eventContainer">
            {isAdmin && selectedDate && (
              <div className="eventForm">
                <h2> Create Event </h2>
```

```
        <p ref={dateSelectedRef}>
          Selected Date: {selectedDate.toDateString()}
        </p>
        <input
          type="text"
          placeholder="Event Name"
          ref={eventName}
        />
        <input
          type="time"
          ref={eventTime}
        />
        <button
          className="createButton"
          onClick={handleCreateEvent}
        >
          Add Event
        </button>
      </div>
    )}
    {events.length > 0 && selectedDate &&  (
      <div className="eventList">
        <h2> Event List </h2>
        <div className="eventCards">
          {events.map((event) =>
            (formattedDate(event.eventDate) === formattedDate(selectedDate) ?
              (<div
                key={event._id}
                className="eventCard"
              >
                <div className="eventCardContainer">
                  <div className="eventCardContainerInfo">
                    <p className="eventTitle">
                      {event.name}
                    </p>
                    <p className="eventTime">
                      {event.eventTime}
                    </p>
                    <span className="eventDate">
                      {formattedDate(event.eventDate)}
                    </span>
                  </div>
                  <div className="eventButtons">
                    {isAdmin && (
                      <button
                        className="deleteButton"
                        onClick={() => {handleDeleteEvent(event._id)}}
                      >
                        Delete Event
                      </button>
                    )}
                  </div>
                </div>
              </div>
            ) : null)
```

```
                        )}
                      </div>
                    </div>
                  )}
                </div>
              </div>
            </div>
        );
}
```

FollowingContext.js

```
import { createContext, useReducer } from 'react'
import FollowingReducer from "./FollowingReducer";
import useAuthUser from 'react-auth-kit/hooks/useAuthUser';
const INITIAL_STATE = {
    user: null,
    followings: [],
    isFetching: false,
    error: false,
}
export const FollowingContext = createContext(INITIAL_STATE)
export const FollowingContextProvider = ({ children }) => {
    const [state, dispatch] = useReducer(FollowingReducer, INITIAL_STATE)
    const auth = useAuthUser()
    return (
        <FollowingContext.Provider
            value={{
                user: auth.user,
                followings: state.followings,
                isFetching: state.isFetching,
                error: state.error,
                dispatch,
            }}
        >
            {children}
        </FollowingContext.Provider>
    )
}
```

FollowingReducer.js

```
const FollowingReducer = (state, action) => {
  switch (action.type) {
    case 'FOLLOW':
      return {
        ...state,
        user: {
          ...state.user,
          followings: [...state.user.followings, action.payload],
        },
      }
    case 'UNFOLLOW':
      return {
        ...state,
        user: {
```

```
        ...state.user,
        followings: state.user.followings.filter(
          following => following !== action.payload
        ),
      },
    }
  default:
    return state
 }
}
export default FollowingReducer
```

FollowingAction.js

```
export const Follow = userId => ({
   type: 'FOLLOW',
   payload: userId,
})
export const Unfollow = userId => ({
   type: 'UNFOLLOW',
   payload: userId,
})
```

Topbar.jsx

```
import './topbar.css';
import { Person, Chat, Notifications, Home} from '@mui/icons-material';
import { useEffect, useState } from 'react'
import axios from 'axios';
import { Link } from 'react-router-dom'
import useAuthUser from 'react-auth-kit/hooks/useAuthUser';
import {SearchField} from './../index.js'
import './topbar.css'
import useAuthHeader from "react-auth-kit/hooks/useAuthHeader";
export default function Topbar() {
 const PF = process.env.REACT_APP_PUBLIC_FOLDER
   const auth = useAuthUser()
   const user = auth.user
 return (
   <div className='topbarContainer'>
     <div className="topbarLeft">
       <div className="logoIcon">
         <Link to={'/'}>
           <img src={PF + '/logo_1.png'} alt="" />
         </Link>
       </div>
       <span className="logoText">
         <Link to={'/'}>
           DniprotechInConnect
         </Link>
       </span>
     </div>
     <div className="topbarCenter">
       <SearchField />
     </div>
     <div className="topbarRight">
```

```
        <div className="topbarLinks">
        </div>
        <div className="topbarIcons">
          <div className="topbarIconItem">
            <Link to='/'>
              <Home />
            </Link>
          </div>
          <div className="topbarIconItem">
            <Link to='/messenger'>
              <Chat />
            </Link>
          </div>
          <div className="topbarIconItem">
            <Link to='/calendar'>
              <Notifications />
            </Link>
          </div>
        </div>
        <Link to={`/profile/${user.username}`}>
          <img
            className='topbarImg'
            src={
                user.profilePicture !== ''
                ? PF + '/' + user.profilePicture
                : PF + '/noAva.png'
            }
            alt=''
          />
        </Link>
      </div>
    </div>
  )
}
```

Решта програмового коду зберігається на електронному носії.

ДОДАТОК Б

ВІДГУК

на кваліфікаційну роботу бакалавра

на тему:

«Розробка програмного забезпечення соціальної мережі для організації спілкування між студентами НТУ «Дніпровська політехніка»

студентки групи 122-20-1 Приліпко Ксенії Геннадіївни

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н. Л.В. Касьяненко

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Приліпко К.Г..doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Приліпко К.Г..pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Qualification_work.rar	Архів. Містить коди програми.
Презентація	
Презентація Приліпко.pptx	Презентація кваліфікаційної роботи